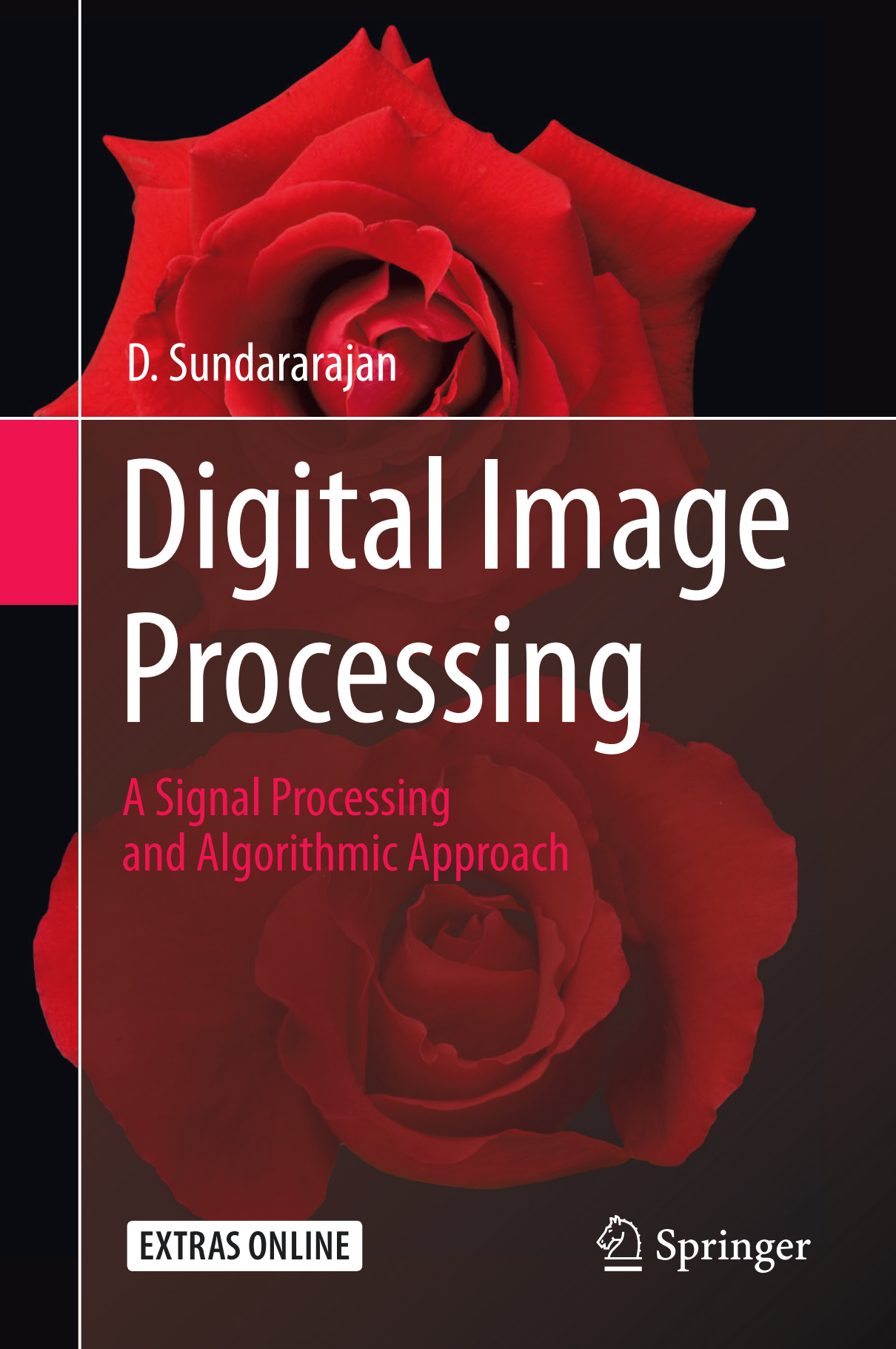

D. Sundararajan
Digital Image Processing
A Signal Processing and Algorithmic Approach
EXTRAS ONLINE
Springer

Digital Image Processing

D. Sundararajan

Digital Image Processing

A Signal Processing and Algorithmic Approach

Springer

D. Sundararajan
Formerly at Concordia University
Montreal
Canada

Additional material to this book can be downloaded from http://extras.springer.com.

ISBN 978-981-10-6112-7 ISBN 978-981-10-6113-4 (eBook)
DOI 10.1007/978-981-10-6113-4

Library of Congress Control Number: 2017950001

Printed on acid-free paper

This Springer imprint is published by Springer Nature
The registered company is Springer Nature Singapore Pte Ltd.
The registered company address is: 152 Beach Road, #21-01/04 Gateway East, Singapore 189721, Singapore

Preface

Vision is one of our strongest senses. The amount of information conveyed through pictures over the Internet and other media is enormous. Therefore, the field of image processing is of great interest and rapidly growing. Availability of fast digital computers and numerical algorithms accelerates this growth. In this book, the basics of *Digital Image Processing* is presented, using a signal processing and algorithmic approach. The image is a two-dimensional signal, and most processing requires algorithms. Plenty of examples, figures, tables, programs, and physical explanations make it easy for the reader to get a good grounding in the basics of the subject, able to progress to higher levels, and solve practical problems.

The application of image processing is important in several areas of science and engineering. Therefore, *Digital Image Processing* is a field of study for engineers and computer science professionals. This book includes mathematical theory, basic algorithms, and numerical examples. Thereby, engineers and professionals can quickly develop algorithms and find solutions to image processing problems of their interest using computers. In general, there is no formula for solving practical problems. Invariably, an algorithm has to be developed and used to find the solution. While every solution is a combination of the basic principles, several combinations are possible for solving the same problem. Out of these possibilities, one has to come with the right solution. This requires some trial-and-error process. A good understanding of the basic principles, knowledge of the characteristics of the image data involved, and practical experience are likely to lead to an efficient solution.

This book is intended to be a textbook for senior undergraduate- and graduate-level *Digital Image Processing* courses in engineering and computer science departments and a supplementary textbook for application courses such as remote sensing, machine vision, and medical analysis. For image processing professionals, this book will be useful for self-study. In addition, this book will be a reference for anyone, student or professional, specializing in image processing. The prerequisite for reading this book is a good knowledge of calculus, linear algebra, one-dimensional digital signal processing, and programming at the undergraduate level.

Programming is an important component in learning and practicing this subject. A set of MATLAB® programs are available at the Web site of the book. While the use of a software package is inevitable in most applications, it is better to use the software in addition to self-developed programs. The effective use of a software package or to develop own programs requires a good grounding in the basic principles of the subject. Answers to selected exercises marked $*$ are given at the end of the book. A Solutions Manual and slides are available for instructors at the Web site of the book.

I assume the responsibility for all the errors in this book and would very much appreciate receiving readers' suggestions and pointing out any errors (email:d_sundararajan@yahoo.com). I am grateful to my Editor and the rest of the team at Springer for their help and encouragement in completing this project. I thank my family for their support during this endeavor.

D. Sundararajan

About the Book

This book "Digital Image Processing—A Signal Processing and Algorithmic Approach" deals with the fundamentals of *Digital Image Processing*, a topic of great interest in science and engineering. *Digital Image Processing* is processing of images using digital devices after they are converted to a 2-D matrix of numbers. While the basic principles of the subject are those of signal processing, the applications require extensive use of algorithms. In order to meet these requirements, the book presents the mathematical theory along with numerical examples with 4×4 and 8×8 subimages. The presentation of the mathematical aspects has been greatly simplified with sufficient detail. Emphasis is given for physical explanation of the mathematical concepts, which will result in deeper understanding and easier comprehension of the subject. Further, the corresponding MATLAB codes are given as supplementary material. The book is primarily intended as a textbook for an introductory *Digital Image Processing* course at senior undergraduate and graduate levels in engineering and computer science departments. Further, it can be used as a reference by students and practitioners of *Digital Image Processing*.

Contents

About the Author

D. Sundararajan is a full-time author in signal processing and related areas. In addition, he conducts workshops on image processing, MATLAB, and LATEX. He was formerly associated with Concordia University, Montreal, Canada, and other universities and colleges in India and Singapore. He holds a M.Tech. degree in Electrical Engineering from Indian Institute of Technology, Chennai, India, and a Ph.D. degree in Electrical Engineering from Concordia university, Montreal, Canada. His specialization is in signal and image processing. He holds a US, a Canadian, and a British Patent related to discrete Fourier transform algorithms. He has written four books, the latest being "Discrete wavelet transform, a signal processing approach" published by John Wiley (2015). He has published papers in IEEE transactions and conferences. He has also worked in research laboratories in India, Singapore, and Canada.

Abbreviations

1-D	One-dimensional
2-D	Two-dimensional
3-D	Three-dimensional
bpp	Bits per pixel
DC	Sinusoid with frequency zero, constant
DFT	Discrete Fourier transform
DWT	Discrete wavelet transform
FIR	Finite impulse response
FT	Fourier transform
IDFT	Inverse discrete Fourier transform
IDWT	Inverse discrete wavelet transform
IFT	Inverse Fourier transform
LoG	Laplacian of Gaussian
LSB	Least significant bit
MSB	Most significant bit
PCA	Principal component analysis
SNR	Signal-to-noise ratio

Chapter 1
Introduction

Abstract The image of a scene or object is inherently a continuous two-dimensional signal. Due to the advantages of digital systems, this type of image has to be converted into a discrete signal. This change in form requires sampling and quantization. The characteristics of a digital image and its spatial- and frequency-domain representations are introduced. The sampling and quantization operations are described.

Most of the information received by humans is visual. A picture is a 2-D visual representation of a 3-D scene. A picture is worth a thousand words. That is, a certain amount of information can be quickly and effectively conveyed by a picture. It is obvious from the popularity of the film medium, Internet, and digital cameras. Digital image processing is the processing of images using digital computers and is used in many applications of science and engineering. It is implied that natural images are converted to digital form prior to processing.

While an image is a 2-D signal, a considerable amount of its processing is carried out in one dimension (row by row and column by column). Therefore, we start with 1-D signals. An example of a one-dimensional (1-D) signal is $x(t) = \sin(t)$. $x(t)$ is the amplitude of the signal at t, the independent variable. Variable t is usually associated with continuous time. As most of the practical signals are of continuous type and digital signal processing is advantageous, the signal is sampled and quantized. A 1-D discrete signal is usually specified as $x(n)$, where the independent variable n is an integer. The sampling interval T_s is usually suppressed. A discrete image is a two-dimensional (2-D) signal, $x(m, n)$, where m and n are the two independent variables. The amplitude of the image $x(m, n)$ at each point is called the pixel value. Pixel stands for picture element. The three major goals of digital image processing are: (i) to improve the quality of the image for human perception, (ii) to improve the quality and represent the image suitable for automatic machine perception, and (iii) to compress the image so that the storage and transmission requirements are reduced. The requirements for human and machine perceptions are, in general, different. These tasks are carried out by computers after the picture is represented in a numeric form. The use of digital cameras, which directly produce digital images, is in prevalent use. Scanners are available to digitize analog photographs. With some exceptions, the processing of an image, which is a 2-D signal, is a straightforward extension of

D. Sundararajan, *Digital Image Processing*, DOI 10.1007/978-981-10-6113-4_1

Table 1.1 Electromagnetic spectrum

Cosmic rays	Gamma rays	X-rays	Ultra violet	Visible spectrum	Infra-red	Microwaves	TV	Radio

that of 1-D signals. For example, with a good knowledge of important operations such as sampling, convolution, and Fourier analysis of 1-D signals, one can easily adapt to their extension for 2-D signals.

An image is some form of a picture giving a visual representation of a scene or an object for human or machine perception. Light is an electromagnetic radiation that can produce visual sensation. Photon is a quantum of electromagnetic radiation. Photons travel at the speed of light. The wavelength of the electromagnetic spectrum varies from $\lambda = 10^{-12}$ to 10^3 m. Components of the electromagnetic spectrum are shown in Table 1.1. Frequency f in Hz and wavelength λ in meters are related by the expression

$$f = \frac{(2.998)10^8}{\lambda}$$

High-frequency photons carry more energy than the low-frequency photons. That small part of the spectrum from $\lambda = (0.43)10^{-6}$ to $(0.79)10^{-6}$ m, which is visible for human beings, is called the visible spectrum.

The invisible part of the spectrum is also of interest in image processing, since it can be sensed by machines (e.g., X-ray is important in the medical field). As in the case of most naturally occurring signals, an image is also a continuous signal inherently. This signal has to be sampled and quantized to make it a digital image. Except that there are two frequency components in two directions to be considered, the sampling is governed by the 1-D sampling theorem. Both sampling and quantization are constrained by the two contradicting criteria, accuracy and processing time.

Each point in an image corresponds to a small part of the scene making the image. The brightness of the light received by an observer from a scene varies as the reflectivity of the objects composing the scene and the illumination vary. This type, which is most common, is called a reflection image. Another type, called the emission image, is obtained from self-luminous objects such as stars or lights. A third type, called the absorption image, is the result of radiation passing through objects. The variation of the attenuation of the intensity of radiation (such as X-ray) recorded by a film is the image. While camera produces most of the images, images are also formed by other sensors such as infrared and ultrasonic. Irrespective of the source, the processing of images involves the same basic principles.

1.1 Image Acquisition

The visual information is a function of two independent variables. It is a 2-D signal. Nowadays, digital cameras produce digital images. These cameras use some type of array of photosensitive devices to produce electrical signals proportional to the scene

brightness over small patches of a scene. The incident light on these devices create charge carriers (holes and electrons), and a voltage applied across the device causes the conduction of current. The potential difference across a resistor in the path of this current is proportional to the average intensity of the light received by the device. The resulting voltages of the array represent the scene being captured as an image. The set of analog signals is converted to a digital image by an interface, called the frame grabber. This interface is a constituent part of digital cameras, and the digital image is delivered in a standard format through an interface to the computer. Of course, it is understood that the sampling and quantization resolutions are set as required, at the time of taking the picture. Invariably, the digital image requires some processing either to enhance it with respect to some criteria and/or to extract useful information for various applications. That is digital image processing. In this chapter, we study the form and characteristics of the digital image.

1.2 Digital Image

While a scene is typically three dimensional, it is represented in two dimensions in the image. In digital image processing, an image is represented as a 2-D matrix of numbers. A $M \times N$ image with M rows and N columns is given by

$$x(m,n) = \begin{array}{c} m \\ \downarrow \end{array} \begin{array}{c} \\ \end{array} \overset{n \rightarrow}{\begin{bmatrix} x(0,0) & x(0,1) & x(0,2) \ldots & x(0,N-1) \\ x(1,0) & x(1,1) & x(1,2) \ldots & x(1,N-1) \\ & & \vdots & \\ x(M-1,0) & x(M-1,1) & x(M-1,2) \ldots & x(M-1,N-1) \end{bmatrix}}$$

With reference to the image, the pixel (picture element) located at (m,n) is with value $x(m,n)$. The image coordinates are m and n, and $x(m,n)$ is proportional to the brightness of the scene about that point. This domain of representation is called the spatial domain, similar to the representation of a 1-D discrete signal in the time domain. While the top-left corner is the origin in most cases, sometimes we also use the bottom-left corner as the origin.

1.2.1 Representation in the Spatial Domain

An image is usually represented in the spatial domain by three forms. A 1-D signal, such as the sine waveform $y(t) = \sin(t)$, is a curve, and we are familiar with its representation in a figure with t represented by the x-axis and $y(t) = \sin(t)$ represented by the y-axis. The independent variable is t and $y(t)$ is the dependent variable because the values of $y(t)$ depend on the values of t. While a 1-D signal is a

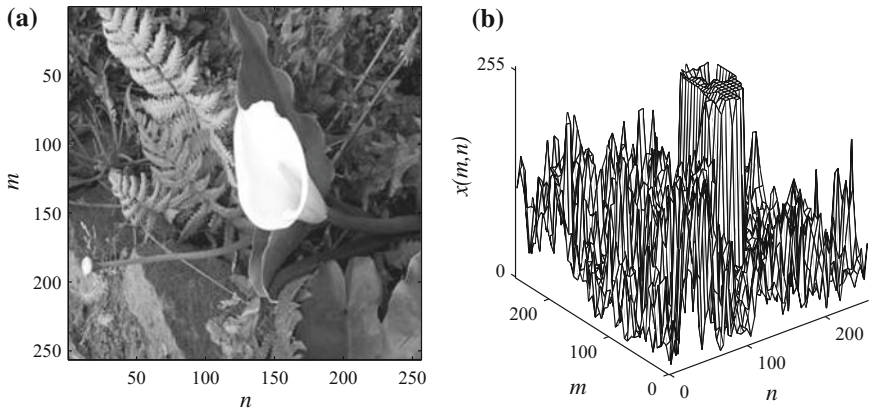


Fig. 1.1 **a** A 256 × 256 image with 256 *gray* levels; **b** its amplitude profile

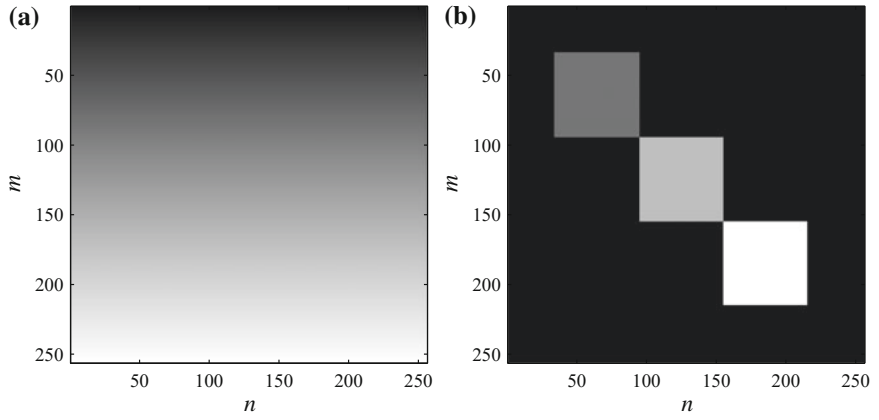


Fig. 1.2 **a** A 256 × 256 image with its intensity values increasing, for each row, from 0 to 255; **b** A 256 × 256 synthetic image with 256 *gray* levels

curve, a 2-D signal is a surface. Therefore, an image $x(m, n)$ can be represented as a surface with the m- and n-axes fixing the two coordinates and a third axis fixing its amplitude. Figure 1.1a shows a 256 × 256 image with 256 gray levels and (b) shows its amplitude profile. While the amplitude profile is mostly used to represent 1-D signals, images are mostly represented using the intensity of its pixels.

Figure 1.1a is the representation of an image by the intensity (gray level) values of its pixels. In a monochromatic or gray-level image, typically, a byte of storage is used to represent the pixel value. With 8 bits, the pixel values are integers from 0 to 255. Figure 1.2a shows a 256 × 256 image in which the intensity values, for each row, are increasing from 0 to 255. The value of all the pixels in the first row is 0, those of the second row is 1, and so on. The value of all the pixels in the bottom

Table 1.2 Pixel values of a 8 × 8 subimage

173	185	189	186	199	195	195	192
177	187	189	192	197	195	189	177
188	190	196	197	199	193	171	124
191	192	197	198	192	158	111	110
196	199	99	189	149	108	110	113
202	200	182	130	100	98	108	114
204	178	117	85	100	96	104	108
173	100	85	87	95	98	96	100

row is 255. Starting from black in the top, the image gradually becomes white at the bottom. Typically, zero is black and the maximum value is white. The value of all the white pixels is set to 255, and the value of the black ones is set to zero. The values between zero and the maximum value are shades of gray (a color between white and black).

A simple image is shown in Fig. 1.2b, which is composed of three squares, with various gray levels, in a black background. This is a synthetic image. This type of images is useful for algorithm design, development, debugging, and verification, since their values and the output of the algorithms are easily predictable. The image has 256 rows of pixels, and each row is made up of 256 pixels with the gray level varying from 0 to 255. The gray-level values of the three squares, from top to bottom, are 84, 168, and 255, respectively.

Another representation of an image is by the numerical values of its intensity, as shown in Table 1.2. While it is impossible to represent a large image in this form, it is, in addition to synthetic images, extremely useful in algorithm development, debugging, and verification (which is a major task in image processing applications) with subimages typically of sizes 4 × 4 and 8 × 8.

In a color image, each pixel is vector-valued. Typically, a color pixel requires 24 bits of storage. A color image is a combination of images with basis colors. For example, a color image is composed of its red, green, and blue components. If each component is represented with 8 bits, then a color pixel requires 24 bits. While most of the natural images are color images, the processing of gray-level images is given importance because its processing can be easily extended to color images in most cases and gray-level images contain essential information of the image. In a binary image, a pixel value is stored in a bit, 0 or 1. Typical binary images contain text, architectural plans, and fingerprints.

When operations, such as transforms, are carried out on images, the resulting images may have widely varying amplitude range and precision. In such cases, quantization is required. More often, images are square and typical sizes vary from 256 × 256 to 4096 × 4096. The numbers are usually a power of 2. Image processing operations are easier with these numbers. For example, in order to reduce the size of

an image to one-half, we simply discard alternate pixels. The all-important Fourier analysis is carried out, in practice, with these numbers.

Some examples of the requirement of storage for images are:

(i) 512×512 binary image,

$$512 \times 512 \times 1 = 262144 \text{ bits} = 32768 \text{ bytes}$$

(ii) 512×512 8-bit gray-level image,

$$512 \times 512 \times 1 = 262144 \text{ bytes}$$

(iii) 512×512 color image, with a byte of storage for each of the three color components of a pixel,

$$512 \times 512 \times 3 = 786432 \text{ bytes}$$

While the picture quality improves with increasing the size, the execution time of algorithms also increases at a fast rate. Therefore, the minimum size that satisfies the application requirements should be selected. The selection of fast algorithms is also equally critical. Even with the modern computers, processing of images could be slow depending on the size of the image and the complexity of the algorithm being executed. Therefore, the minimum size, the simplest type (binary, gray-level, or color image), and an appropriate algorithm must be carefully chosen for efficient and economical image processing for any application.

1.2.2 Representation in the Frequency Domain

One of the major tasks in image processing is to find suitable representations of images in other domains, in addition to the spatial domain, so that the processing becomes easier and efficient, as is the case in 1-D signal processing. The suitable representation invariably requires taking the transform of the image. Transforms approximate practical images, which usually have arbitrary amplitude profiles, as a weighted sum of a finite set of well-defined basis functions with adequate accuracy. There are many transforms used in image processing, and each one has a different set of basis functions and is suitable for some tasks. The most important of all the transforms is the Fourier transform. Sinusoidal curves are the Fourier basis functions for 1-D signals, and sinusoidal surfaces, such as that shown in Fig. 1.3, are the Fourier basis functions for 2-D signals (images). In a transformed form, important characteristics of the images, such as their frequency content, can be estimated. The interpretation of operations, such as filtering of images, becomes easier. Further, the computational complexity of operations and storage requirements are also reduced in most cases.

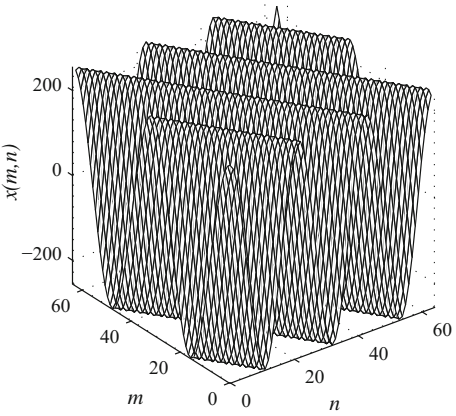


Fig. 1.3 A 64×64 sinusoidal surface, which is a typical basis function in the 2-D Fourier transform representation of images

1.3 Quantization and Sampling

Sampling is required due to limited spatial and temporal resolutions (number of pixels) of a digital image. Quantization is required due to limited intensity resolution (wordlength). A pixel value, typically, is the integral of the image intensity over a finite area. As most practical signals are continuous functions of continuous variables, both sampling and quantization are required to get a digital signal so that they can be processed by a digital computer. Sampling is converting a continuous function into a discrete one. The values of a sampled function are known only at the discrete values of its independent variable. Quantization is converting a continuous variable into a discrete one. The values of a quantized variable are fixed at discrete intervals.

Consider one period of the continuous sinusoidal signal

$$x(t) = \cos\left(\frac{2\pi}{16}t + \frac{\pi}{6}\right)$$

shown in Fig. 1.4. The signal is sampled with a sampling interval of 1 s. Therefore, starting with $t = 0$, we get 16 samples

$$\begin{aligned} x(n) = &\{0.8660, 0.6088, 0.2588, -0.1305, -0.5000, -0.7934, -0.9659, -0.9914, \\ &-0.8660, -0.6088, -0.2588, 0.1305, 0.5000, 0.7934, 0.9659, 0.9914\} \end{aligned}$$

These samples are further quantized with a quantization step of 0.2. That is, each sample value is restricted to one of the finite set of values

$$\{1, 0.8, 0.6, 0.4, 0.2, 0, -0.2, -0.4, -0.6, -0.8, -1\}$$

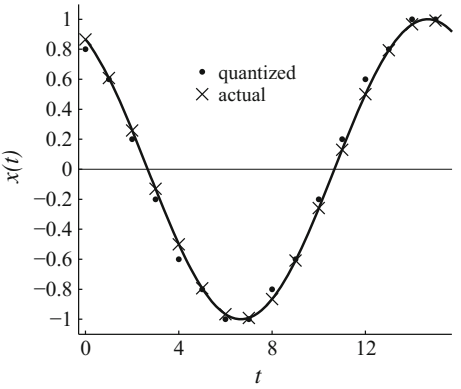


Fig. 1.4 Sampling and quantizing a 1-D signal

Each sample is assigned to the nearest allowed value. The samples of the sampled and quantized signal are

$$x_q(n) = \{0.8, 0.6, 0.2, -0.2, -0.6, -0.8, -1.0, -1.0, -0.8, -0.6, \\ -0.2, 0.2, 0.6, 0.8, 1.0, 1.0\}$$

shown by dots in Fig. 1.4. The actual sample values are shown by crosses. Maximum error is one-half of the quantization step. Both sampling and quantization operations introduce errors in the representation of a signal. According to the sampling theorem, the sampling frequency has to be more than twice that of the highest frequency content of the signal. The quantization step should be selected so that the quantization noise is within acceptable limit.

1.3.1 Quantization

Quantization is the process of mapping the amplitude of a continuous variable into a set of finite discrete values. For a digital representation, the pixel values of an image have to be quantized to some finite levels so that the image can be stored using a finite number of bits. Typically, 8 bits are used to represent a pixel value. Figure 1.5 shows the effect of quantization of the pixel values, using 8, 7, 6, 5, 4, 3, 2, and 1 bits. Reducing the number of bits reduces the number of gray levels and, in turn, reduces the contrast of the image. The deterioration in quality is not noticeable upto 6 bits of representation. From 5 bits onward, grayscale contouring effect is noticeable. False edges appear when the gradually changing pixel values in a region of the image are replaced by a single value. Due to the lower quantization levels, edges are created between adjacent regions. As the use of 6 or 7 bits does not save much and the 8-bit (byte) wordlength is in popular use in the computer architectures, the 8-bit representation is most often used.

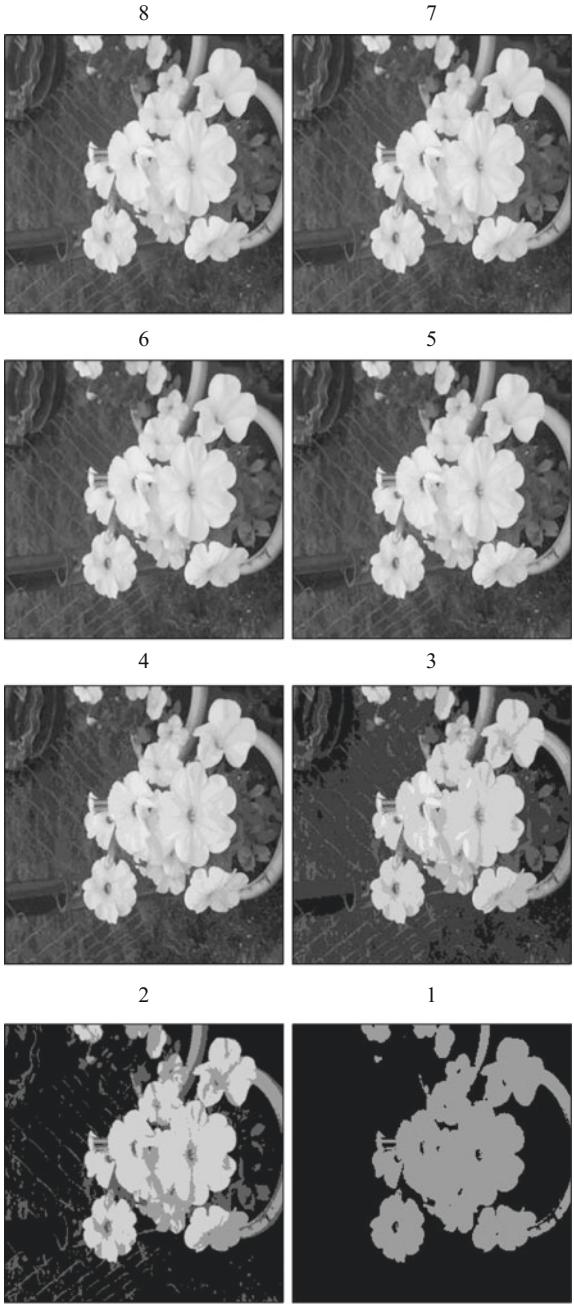


Fig. 1.5 Representations of an image using 8, 7, 6, 5, 4, 3, 2, and 1 bits

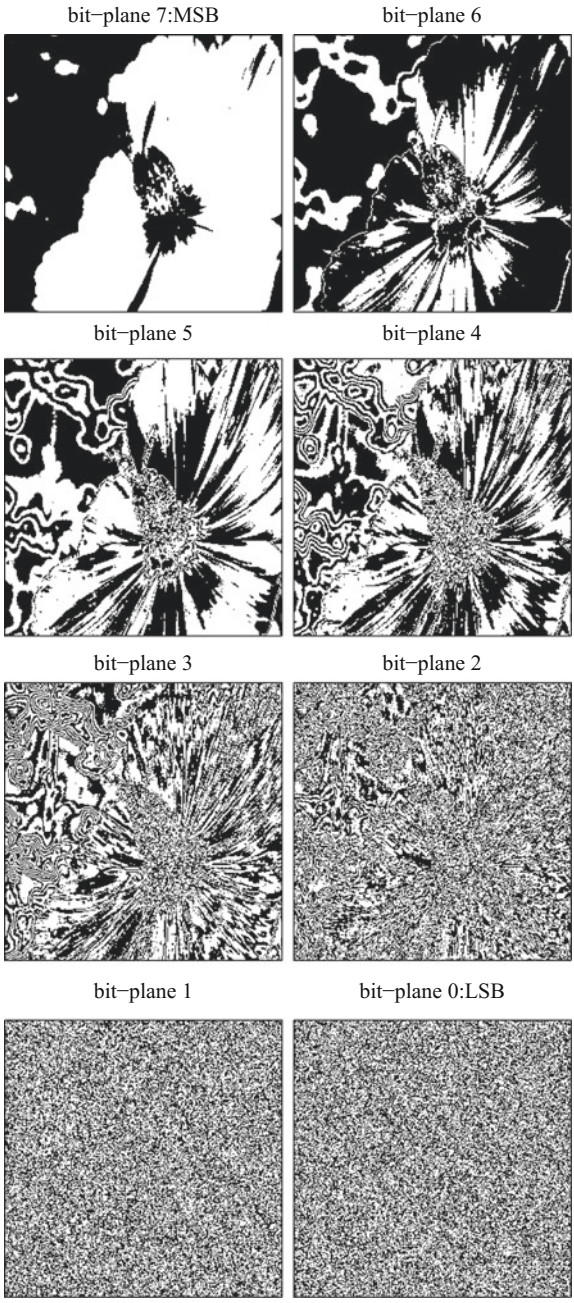


Fig. 1.6 Bit-plane representations of an image

The relative influence of the various bits in the formation of the image is shown in Fig. 1.6. A gray-level image can be decomposed into a set of binary images, which is useful in applications such as compression. The last image corresponds to the least significant bit. It looks like an image generated by a set of random numbers, and it is difficult to relate it to the original image. Higher-order bits carry more information. As expected, the most significant bit carries most information and the corresponding image (the first) resembles like its original. The bit-plane images can be isolated from the grayscale images by repeatedly dividing the image matrix by successive powers of 2 and taking the remainder of dividing the truncated quotient by 2. For example, let $x = \{0, 1, 2, 3, 4, 5\}$. Dividing x by 2 and taking the remainders, we get $x0 = \{0, 1, 0, 1, 0, 1\}$. Dividing x by 2 and taking the truncated quotients, we get $x2q = \{0, 0, 1, 1, 2, 2\}$. Dividing $x2q$ by 2 and taking the remainders, we get $x1 = \{0, 0, 1, 1, 0, 0\}$. Dividing x by 4 and taking the truncated quotients, we get $x4q = \{0, 0, 0, 0, 1, 1\}$. Dividing $x4q$ by 2 and taking the remainders, we get $x2 = \{0, 0, 0, 0, 1, 1\}$. Note that $2^0x0 + 2^1x1 + 2^2x2 = x$. The 4×4 4-bit image $x(m, n)$ and its bit-plane components from MSB to LSB are

$$\begin{bmatrix} 8 & 1 & 7 & 3 \\ 1 & 11 & 15 & 12 \\ 0 & 11 & 7 & 13 \\ 2 & 10 & 9 & 6 \end{bmatrix} = 2^3 \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} + 2^2 \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} + 2 \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Quantization levels with equal intervals is called linear quantization. In nonlinear quantization, the range of the frequently occurring pixels is quantized using more bits and vice versa. The average error due to quantization is reduced without increasing the number of bits. This type of quantization is often used in image compression. It should be noted that, while sampling and quantization are necessary to get the advantages of digital image processing, the image is corrupted to some extent due to the quantization noise and the aliasing effect. It should be ensured that image quality is within acceptable limits by proper selection of the sampling interval and the quantization levels. In general, a rapidly varying scene requires a higher sampling rate and fewer quantization levels and vice versa. A 256×256 image with 64 gray levels is typically the minimum for most practical purposes.

1.3.2 Spatial Resolution

An image represents an object of a certain area. The spatial resolution is the physical area of the object represented by a pixel. The resolution varies from nanometers in microscopic images to kilometers in satellite images. The number of independent pixel values per unit distance (pixel density) indicates the spatial resolution. A higher number of pixels improves the ability to see finer details of an object in the image. For example, the resolution of a digital image of size 512×512 formed from an analog image of size 32×32 cm is $512/32 = 16$ pixels per centimeter. Figure 1.7a–d shows,

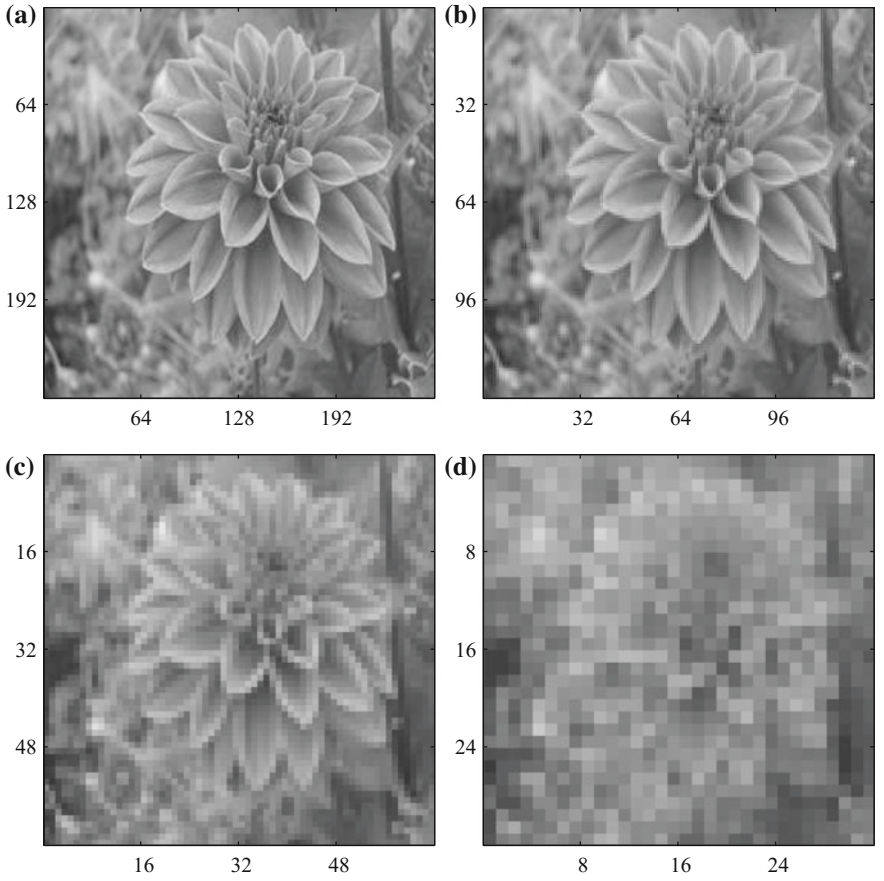


Fig. 1.7 Effects of reducing the spatial resolution. **a** Resolution 256×256; **b** resolution 128×128; **c** resolution 64×64; **d** resolution 32×32

respectively, an image with resolutions 256×256, 128×128, 64×64, and 32×32. Reducing the spatial resolution results in blockiness of the image. The blockiness is just noticeable in the image in (b) and clearly seen in the image in (c), while the image in (d) becomes unrecognizable.

1.3.3 Sampling and Aliasing

When sampling a signal, the sampling frequency must be greater than twice that of its highest frequency component in order to reconstruct the signal perfectly from its samples. In the case of an image, there are two frequency components (horizontal and vertical) to be considered. Aliasing effect is the impersonation of a higher

frequency sinusoid as a lower frequency sinusoid due to insufficient number of samples. An arbitrary sinusoid with frequency f Hz requires more than $2f$ samples for its unambiguous representation by its samples. Aliasing can be eliminated by suitable lowpass filtering of the image and then sampling so that the bandwidth is less than half of that of the sampling frequency. The price that is paid for eliminating aliasing is the blurring of the image, since high-frequency components, which provide the details, are removed in lowpass filtering.

The aliasing effect is characterized by the following formulas.

$$x(n) = \cos\left(\frac{2\pi}{N}(k + lN)n + \phi\right) = \cos\left(\frac{2\pi}{N}kn + \phi\right), \quad k = 0, 1, \ldots, \frac{N}{2} - 1$$

$$x(n) = \cos\left(\frac{2\pi}{N}(lN - k)n + \phi\right) = \cos\left(\frac{2\pi}{N}kn - \phi\right), \quad k = 1, 2, \ldots, \frac{N}{2} - 1$$

where the number of samples N and index l are positive integers. With N even, oscillations increase only upto $k = \frac{N}{2}$, decrease afterward, and cease at $k = N$, and this pattern repeats indefinitely. With frequency indices higher than $\frac{N}{2}$, frequency folding occurs. Therefore, sinusoids with frequency index upto $\frac{N}{2}$ can only be uniquely identified with N samples. Frequency with index $\frac{N}{2}$ is called the folding frequency. The implication is that, with the number of samples fixed, only a limited number of sinusoidal components can be distinctly identified. For example, with 256 samples, the uniquely identifiable frequency components are

$$x(n) = \cos\left(\frac{2\pi}{256}kn + \phi\right), \quad k = 0, 1, \ldots, 127$$

Figure 1.8a shows a 32×32 sinusoidal surface

$$x(m, n) = \cos\left(\frac{2\pi}{32}2m + \frac{2\pi}{32}1n + \frac{\pi}{2}\right)$$

with frequencies 2/32 and 1/32 cycles per sample along the m and n axes, respectively. The bottom peak of the sinusoidal surface is black, and the top peak is white. It is clear that the surface makes 2 cycles along the m axis and one along the n axis. Consider a 32×32 sinusoidal surface

$$x(m, n) = \cos\left(\frac{2\pi}{32}30m + \frac{2\pi}{32}31n - \frac{\pi}{2}\right)$$

with frequencies 30/32 and 31/32 cycles per sample along the m and n axes, respectively. Frequency with index $\frac{N}{2} = \frac{32}{2} = 16$ is called the folding frequency. The apparent frequencies are $(32 - 30)/32 = 2/32$ and $(32 - 31)/32 = 1/32$ cycles per sample, respectively. This sinusoidal surface also produces oscillations with the same frequency as in Fig. 1.8a.

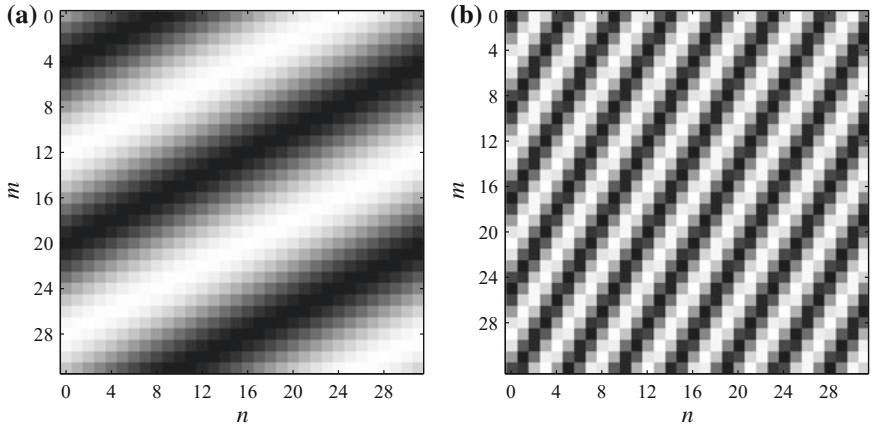


Fig. 1.8 Aliasing effect. **a** $x(m,n) = \cos(\frac{2\pi}{32}2m + \frac{2\pi}{32}1n + \frac{\pi}{2}) = \cos(\frac{2\pi}{32}30m + \frac{2\pi}{32}31n - \frac{\pi}{2})$; **b** $x(m,n) = \cos(\frac{2\pi}{32}4m + \frac{2\pi}{32}7n + \pi) = \cos(\frac{2\pi}{32}28m + \frac{2\pi}{32}25n - \pi)$

$$
\begin{aligned}
x(m,n) &= \cos\left(\frac{2\pi}{32}30m + \frac{2\pi}{32}31n - \frac{\pi}{2}\right) \\
&= \cos\left(\frac{2\pi}{32}(32-2)m + \frac{2\pi}{32}(32-1)n - \frac{\pi}{2}\right) = \cos\left(\frac{2\pi}{32}2m + \frac{2\pi}{32}1n + \frac{\pi}{2}\right)
\end{aligned}
$$

Figure 1.8b shows a 32×32 sinusoidal surface

$$
x(m,n) = \cos\left(\frac{2\pi}{32}4m + \frac{2\pi}{32}7n + \pi\right)
$$

with frequencies 4/32 and 7/32 cycles per sample along the m and n axes, respectively. It is clear that the surface makes 4 cycles along the m axis and 7 along the n axis. Consider the sinusoidal surface

$$
\begin{aligned}
x(m,n) &= \cos\left(\frac{2\pi}{32}28m + \frac{2\pi}{32}25n - \pi\right) \\
&= \cos\left(\frac{2\pi}{32}(32-4)m + \frac{2\pi}{32}(32-7)n - \pi\right) = \cos\left(\frac{2\pi}{32}4m + \frac{2\pi}{32}7n + \pi\right)
\end{aligned}
$$

This sinusoidal surface also produces oscillations with the same frequency as in Fig. 1.8b. If we double the number of samples, then aliasing is avoided in these cases.

To fix the sampling frequency for a class of real-valued images, find the Fourier spectra of typical images using the 2-D DFT for increasing sampling frequencies. The appropriate sampling frequency in each of the two directions is that which yields negligible spectral magnitude values in the vicinity of one-half of the sampling frequency.

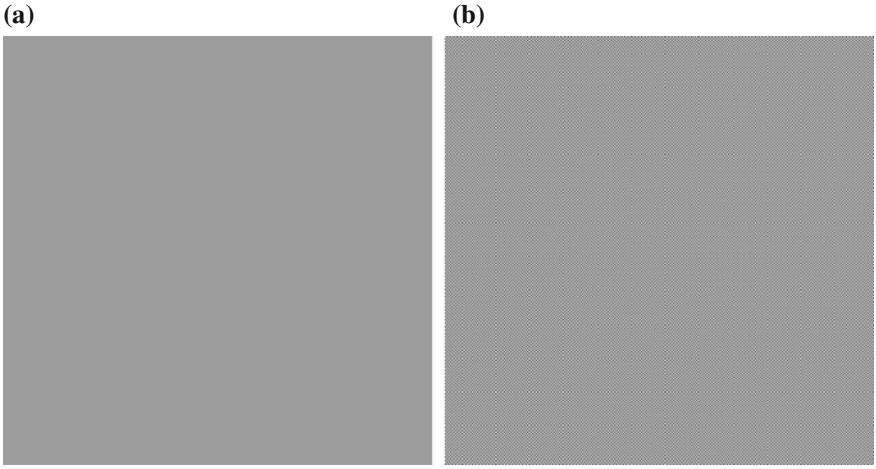


Fig. 1.9 **a** 256 × 256 flat image and **b** the image with Moiré effect

1.3.4 Image Reconstruction and the Moiré Effect

Moiré effect is the appearance of beat patterns in the reconstructed images. This problem is created due to the extension of the passband of the frequency response of practical reconstruction filters beyond half the sampling frequencies. To reconstruct a signal from its samples, we have to cut off everything except one period of the periodic spectrum. Practical filters are not ideal. This phenomenon occurs only when the difference between the two frequencies is small compared with either one. This type of periodic components typically occurs in images due to aliasing or the characteristics of practical reconstruction filters. The cutoff frequency of such filters extends beyond that of the ideal lowpass filters. When the image contains periodic components with frequencies close to that of the half the sampling frequency, due to the periodicity of the spectrum and improper reconstruction filter response, a pair of periodic components that could produce this effect can occur.

In the case that the image is flat with uniform gray levels (DC frequency component only) and the reconstruction filters pass the component with half the sampling frequency, then stripes will occur in the displayed images, as shown in Fig. 1.9a, b, respectively.

The Moiré effect may occur in images with strong periodic components such as streets in photos taken from high altitudes, photos of ocean waves, patterns in sand, and plowed fields. Therefore, as most of the practical images do not contain strong periodic components, we have to just ensure their existence by examining the Fourier spectra of the image. The Moiré effect can be mitigated with a higher sampling frequency or using a compensation filter in the reconstruction process.

1.4 Applications of Digital Image Processing

Digital image processing is widely used in entertainment, business, science, and engineering applications. Some applications are:

1. Image sharpening and restoration. Images taken by cameras often require processes, such as zooming, sharpening, blurring, and gray scale to color conversion, to make the input image more suitable for the intended purpose.
2. Medical Applications. X-ray and CT scan images are routinely used in the treatment of patients in hospitals.
3. Remote sensing. In the area of remote sensing, satellites scan the earth and take pictures to study the crop patterns and climatic changes.
4. Image compression and transmission. Live broadcasts are available of any event anywhere on earth through television, Internet, and other media.
5. Robots. Computer vision is a key component in robots.
6. Automatic inspection of components in industries. For example, printed circuit boards are inspected to check that all components are present and the quality of soldering is acceptable.
7. Security. Monitoring even homes, apart from offices, is commonly used by closed-circuit television-based security monitoring systems. Finger print analysis is used for identification. Number plate recognition is used for checking over speed and automated toll systems.

1.5 The Organization of This Book

This book emphasizes both the signal processing and algorithmic aspects of digital image processing. Both are indispensable for a good understanding of the subject and its practical use. For this purpose, most of the algorithms are explained with 4×4 or 8×8 subimages, although practical images are much larger. Only with a good understanding of the fundamentals, effective solutions to practical image processing problems can be obtained.

In this chapter, the characteristics of a digital image and its spatial- and frequency-domain representations are presented. While most practical images occur in continuous form, they are converted into digital form, processed, and stored for efficiency. Therefore, sampling and quantization are described next. Although the transform-domain processing is essential, as the images naturally occur in the spatial domain, image enhancement in the spatial domain is presented in Chap. 2. Point operations, histogram processing, and neighborhood operations are presented. The convolution operation, along with the Fourier analysis, is essential for any form of signal processing. Therefore, the 1-D and 2-D convolution operations are introduced. Fourier analysis is presented in Chap. 3. Transforms provide an alternate representation of images, which usually facilitates easier interpretation of operations and fast processing. The most important of all the transforms, the Fourier transform, decomposes an image in

terms of sinusoidal surfaces. This transform is of fundamental importance to image processing, as is the case in almost all areas of science and engineering. As in the case of the convolution operation, both the 1-D and 2-D versions are described. Although the image is a 2-D signal, some of the important operations are decomposable and can be carried out in one dimension with reduced execution time. Another advantage is that understanding of the 1-D version is simpler. Definition, properties, and examples of the transforms are presented. In Chap. 4, filtering operations, presented in Chap. 3, are described in the frequency domain. Depending on the problem, either the spatial-domain or frequency-domain processing is preferred.

In the filtering operations presented so far, it is assumed that no knowledge of the source of degradation of the image is available. In Chap. 5, the restoration of the images is presented. This is a filtering operation in which prior knowledge of the source of degradation of the image is known. Interpolation of an image is often required to change its size and in operations such as rotation. In Chap. 6, the interpolation of images is described first. Next, geometric transformations such as translation, scaling, rotation, and shearing are presented. Correlation operation is a similarity measure between two images. It can detect and locate the position of an object in an image, if present. The registration of images of the same scene, taken at different times and conditions, is also presented. In Chap. 7, the Radon transform is presented, which is important in computer tomography in medical and industrial applications. This transform enables to produce the image of an object, without intrusion, using its projections at various directions. In processing the color and grayscale images, which occur mostly, their binary version is often used. In Chap. 8, morphological processing of images is presented. The structure and shape of the objects are analyzed so that they can be identified. The basic operation in this processing is binary convolution that is based on logical operations rather than arithmetic operations.

Edge detection is an important step in the segmentation of the image and leads to object recognition. Edge detection in images is presented in Chap. 9. In Chap. 10, segmentation of an image is presented. Various segmentation methods are described. The objects of a segmented image are represented by various ways, and their features are extracted to enable object recognition. Object description and representation are described in Chap. 11. Each object, based on their description, is classified appropriately, as described in Chap. 12.

Image compression is as essential as its processing, since images require large amounts of memory, and in their original form, it is difficult to transmit and store them. Chapter 13 presents various methods of image compression. Emphasis is given to using the DWT, since it is a part of the current image compression standard. Human vision is more sensitive to color than gray levels. Therefore, color image processing is important, although it requires more memory to store and longer execution times to process. Chapter 14 presents color image processing. Some of the processings are based on those of gray-level images and some are exclusive to color images. In the Appendix, an algorithm for fast computation of the discrete Fourier transform is described. It is not an exaggeration to state that a single most important reason for the existence and continuing growth of digital signal and image processing is due to this algorithm.

Basically, there are two important components in signal and image processing theory and applications. One is the mathematics, and the other is algorithms and programming. The programming language is individual's choice depending on the application requirements. But the basic mathematical and algorithmic principles are common to everybody. In this book, an attempt has been made to present the basic principles clearly and concisely with a large number of examples. With a good knowledge of the basic principles, one can get a good expertise in image processing that is directly proportional to the amount of hardware and software realization put up. As usual, the basics can be learned in a finite amount of time but there is no end to learning by practice.

1.6 Summary

- Humans get most information through vision.
- Digital image is a 2-D matrix representation of a 3-D scene.
- Digital image is obtained by recording the luminous intensity received from a scene at discrete points.
- The light coming from an object may be due to reflection, emission, or absorption.
- Light is an electromagnetic radiation that can produce visual sensation.
- Light is transmitted at various frequencies, called the electromagnetic spectrum.
- That part of the electromagnetic spectrum, which humans can see, is called the visible spectrum.
- The whole electromagnetic spectrum is of interest in image processing, since machines can detect radiation at all frequencies.
- Since most naturally occurring images are in continuous form, sampling and quantization are required to obtain a digital image.
- Digital image processing is composed of processing images to make it more suitable for human or machine vision. As images require large amounts of memory to store, compression of images is an important part of image processing.
- Digital image is a 2-D signal, and its processing, for the most part, is an extension of 1-D signal processing.
- Digital image is a 2-D matrix of numbers. Each number is called a pixel (picture element). The numbers represent the intensity of the light at that point and usually represented by 8 bits.
- As in the case of 1-D signal processing, transforms, in particular the Fourier analysis, play a dominant role in image processing also.
- The representation of an image by a 2-D matrix should be sufficiently accurate in terms of sampling, resolution, and quantization.

Exercises

1.1 Find the memory required, in bytes, to store the following images.

(i) 64×64 binary image.
(ii) 128×128 8-bit gray-level image.
(iii) 64×64 24-bit full-color image.
(iv) 512×512 binary image.
(v) 1024×1024 8-bit gray-level image.
(vi) 4096×4096 24-bit full-color image.

1.2 Find the memory required, in bytes, to store the images given in Exercise (1.1) after

(i) doubling the number of rows and columns and
(ii) reducing the number of rows and columns by a factor of 2.

1.3 Find the pixel values of the 8×8 8-bit gray-level image

$$\{x(m,n), m = 0, 1, 2, \ldots, 7 \quad \text{and} \quad n = 0, 1, 2, \ldots, 7\}$$

corresponding to the given 2-D function. (Round the real values of the image to the nearest integer after necessary scaling.)

*(i)
$$x(m,n) = 1 + \cos\left(\frac{2\pi}{8}m + \frac{2\pi}{8}n - \frac{\pi}{4}\right)$$

(ii)
$$x(m,n) = 1 + \cos\left(\frac{2\pi}{8}m + \frac{2\pi}{8}2n - \frac{\pi}{6}\right)$$

(iii)
$$x(m,n) = 1 + \cos\left(\frac{2\pi}{8}0m + \frac{2\pi}{8}0n\right)$$

(iv)
$$x(m,n) = 1 + \cos\left(\frac{2\pi}{8}4m + \frac{2\pi}{8}4n\right)$$

(v)
$$x(m,n) = 1 + \cos\left(\frac{2\pi}{8}0m + \frac{2\pi}{8}n\right)$$

(vi)
$$x(m,n) = 1 + \cos\left(\frac{2\pi}{8}2m + \frac{2\pi}{8}0n\right)$$

1.4 Find the pixel values of the 8×8 binary image by setting the gray-level values between 0–127 to 0 and 128–255 to 1 for each of the 8-bit gray-level images

$$\{x(m,n), m = 0, 1, 2, \ldots, 7 \quad \text{and} \quad n = 0, 1, 2, \ldots, 7\}$$

obtained in Exercise (1.3).

1.5 Find the bit-plane components of the image and verify that the image can be reconstructed from them.

(i)

$$\begin{bmatrix} 8 & 3 & 7 & 3 \\ 4 & 11 & 15 & 12 \\ 0 & 10 & 11 & 1 \\ 2 & 10 & 3 & 6 \end{bmatrix}$$

(ii)

$$\begin{bmatrix} 2 & 1 & 7 & 3 \\ 1 & 1 & 15 & 12 \\ 0 & 13 & 5 & 13 \\ 2 & 10 & 4 & 6 \end{bmatrix}$$

(iii)

$$\begin{bmatrix} 8 & 1 & 7 & 8 \\ 5 & 11 & 15 & 12 \\ 0 & 6 & 7 & 13 \\ 2 & 10 & 7 & 6 \end{bmatrix}$$

(iv)

$$\begin{bmatrix} 7 & 1 & 7 & 3 \\ 1 & 6 & 15 & 12 \\ 0 & 11 & 1 & 13 \\ 2 & 10 & 9 & 9 \end{bmatrix}$$

(v)

$$\begin{bmatrix} 8 & 1 & 7 & 7 \\ 1 & 11 & 6 & 12 \\ 0 & 8 & 7 & 13 \\ 9 & 10 & 9 & 6 \end{bmatrix}$$

1.6

(i) Find the spatial resolution of an image if the scene of size 4 m by 4 m is represented by a 256×256 image.
(ii) Find the spatial resolution of an image if the scene of size 10 km by 10 km is represented by a 4096×4096 image.

(iii) Find the spatial resolution of an image if the scene of size 7 mm by 7 mm is represented by a 1024×1024 image.

1.7 Let the sampling frequencies along both the directions be 32 cycles per sample. Is there aliasing or not in the image $x(m, n)$? If so, what are the impersonated frequencies?

*(i)
$$x(m,n) = \cos\left(\frac{2\pi}{32}28m + \frac{2\pi}{32}30n - \frac{\pi}{6}\right)$$

(ii)
$$x(m,n) = \cos\left(\frac{2\pi}{32}15m + \frac{2\pi}{32}14n + \frac{\pi}{2}\right)$$

(iii)
$$x(m,n) = \cos\left(\frac{2\pi}{32}27m + \frac{2\pi}{32}22n - \frac{\pi}{3}\right)$$

(iv)
$$x(m,n) = \cos\left(\frac{2\pi}{32}3m + \frac{2\pi}{32}3n + \frac{\pi}{2}\right)$$

(v)
$$x(m,n) = \cos\left(\frac{2\pi}{32}32m + \frac{2\pi}{32}32n + \frac{\pi}{4}\right)$$

(vi)
$$x(m,n) = \cos\left(\frac{2\pi}{32}17m + \frac{2\pi}{32}11n - \frac{\pi}{3}\right)$$

Chapter 2
Image Enhancement in the Spatial Domain

Abstract Although the transform domain processing is essential, as the images naturally occur in the spatial domain, image enhancement in the spatial domain is presented first. Point operations, histogram processing, and neighborhood operations are presented. The convolution operation, along with the Fourier analysis, is essential for any form of signal processing. Therefore, the 1-D and 2-D convolution operations are introduced. Linear and nonlinear filtering of images is described next.

An image is enhanced to increase the amount of information that can be interpreted visually. Image enhancement improves the quality of an image for a specific purpose. The process depends up on the characteristics of the image and whether it is required for human perception or machine vision. Some features are enhanced to suit human or machine vision. For example, the spot noise is reduced in median filtering so that a better viewing of the original image is obtained. Edges are enhanced by highpass filtering and the output image is a step in computer vision. In this chapter, we present three types of operations. The simplest and yet very useful image enhancement process is point operation. The output pixel is a function of the corresponding input pixels of one or more images. Thresholding is an important operation in processing images. Another type is intensity transformations to contrast enhancement, called histogram processing. Linear and nonlinear filtering is a major type of processing in which the output pixel is a function of the pixels in a small neighborhood of the input pixel. An operation is linear, if the output to a linear combination of input signals is the same linear combination of the outputs to the individual signals.

2.1 Point Operations

In point processing, the new value of a pixel is a function of the corresponding values of one or more images. Let $x(m, n)$ and $y(m, n)$ be two images of the same size. Then, pointwise arithmetic operations of corresponding pixel values of the two images are given as

D. Sundararajan, *Digital Image Processing*, DOI 10.1007/978-981-10-6113-4_2

$$z(m,n) = x(m,n) + y(m,n)$$
$$z(m,n) = x(m,n) - y(m,n)$$
$$z(m,n) = x(m,n) * y(m,n)$$
$$z(m,n) = x(m,n)/y(m,n)$$

One of the operands in these operations can be a constant. For example, $z(m,n) = Cx(m,n)$ and $z(m,n) = C + x(m,n)$, where C is a constant. Logical operations AND (&), OR (|) and NOT (˜) are also used in a similar way on binary images.

2.1.1 Image Complement

The complement of an image is its photographic negative obtained by subtracting the pixel values from their maximum range. In a 8-bit gray-level image, the complement, $\tilde{x}(m,n)$, of the image $x(m,n)$ is given by

$$\tilde{x}(m,n) = 255 - x(m,n)$$

The new pixel value is obtained by subtracting the current value from 255. For example,

$$x(m,n) = \begin{bmatrix} 101 & 104 & 110 & 134 \\ 96 & 103 & 100 & 126 \\ 98 & 99 & 106 & 98 \\ 100 & 93 & 107 & 90 \end{bmatrix} \quad \tilde{x}(m,n) \begin{bmatrix} 154 & 151 & 145 & 121 \\ 159 & 152 & 155 & 129 \\ 157 & 156 & 149 & 157 \\ 155 & 162 & 148 & 165 \end{bmatrix}$$

Figure 2.1a, b show, respectively, a 256 × 256 8-bit gray level image and its complement. The flower in the middle is white in (a) and it has become black in (b), as expected. The dark areas have become white and vice versa. Sometimes, the complement brings out certain features better. For a binary image, the complement is given by

$$\tilde{x}(m,n) = 1 - x(m,n)$$

2.1.2 Gamma Correction

Image sensors and display devices often have nonlinear intensity characteristics. Since the nonlinearity is characterized by a power law and γ is the symbol used for the exponent, this operation is called gamma correction. To compensate such nonlinearity, an inverse transformation has to be applied to individual pixels of the image.

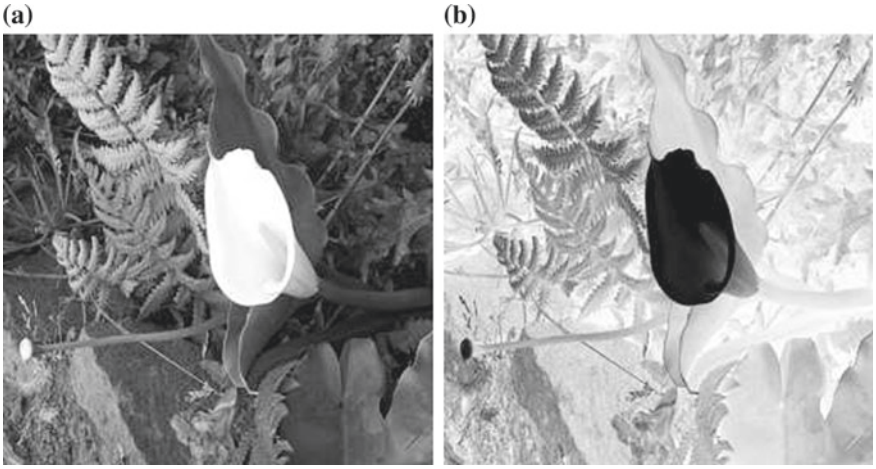


Fig. 2.1 **a** A 256 × 256 8-bit gray level image and **b** its complement

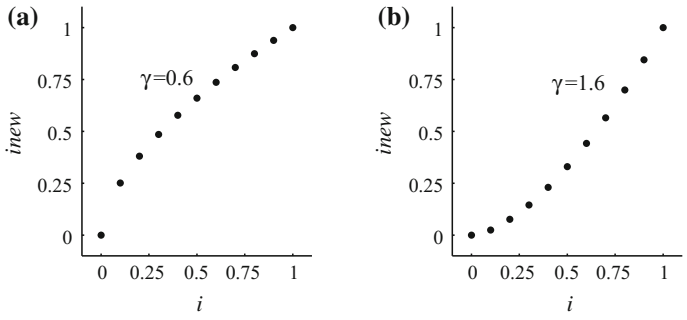


Fig. 2.2 Intensity transformation in γ correction **a** $\gamma = 0.6$; **b** $\gamma = 1.6$

In gamma correction, the new intensity value *inew* of a pixel is its present value i raised to the power of γ.

$$inew = i^{\gamma} \tag{2.1}$$

Let the maximum intensity value be 255. Then, all the pixel values are first divided by 255 to map the intensity values into the range 0–1. This step ensures that the pixel values stay in the range 0–255. Then, Eq. (2.1) is applied. The resulting values are multiplied by 255 and rounded to get the processed values.

Figure 2.2a, b show, respectively, the intensity mapping for values of $\gamma = 0.6$ and $\gamma = 1.6$. The pixel values are also tabulated in Table 2.1. For $\gamma < 1$, the intensity values are scaled up and the output image gets brighter. For $\gamma > 1$, the intensity values are scaled down. Figure 2.3a, b, show, respectively, the versions of the image in Fig. 2.1a after gamma correction with $\gamma = 0.8$ and $\gamma = 1.6$, respectively. The image is brighter in (a) and dimmer in (b). In addition to correcting nonlinearity of devices, this transformation can also be used for contrast manipulation of images.

Table 2.1 Gamma correction

i	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
$i^{0.6}$	0	0.2512	0.3807	0.4856	0.5771	0.6598	0.7360	0.8073	0.8747	0.9387	1
$i^{1.6}$	0	0.0251	0.0761	0.1457	0.2308	0.3299	0.4416	0.5651	0.6998	0.8449	1

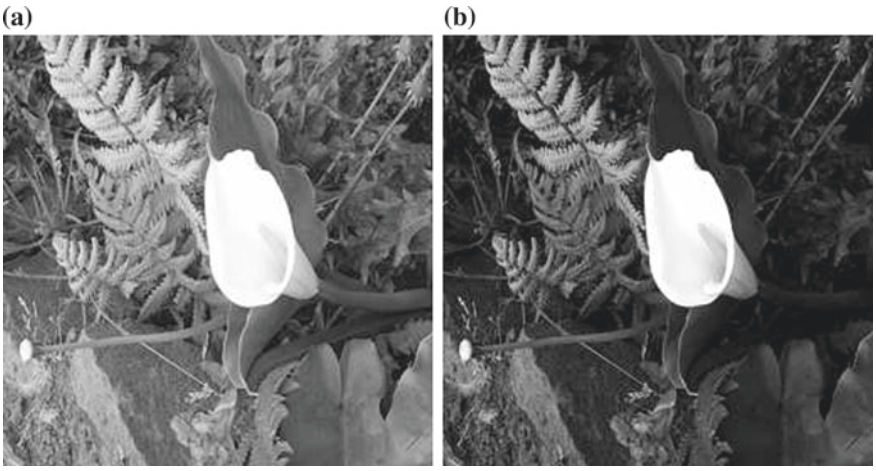


Fig. 2.3 Versions of the image in Fig. 2.1a after gamma correction with $\gamma = 0.8$ (**a**) and $\gamma = 1.6$ (**b**)

2.2 Histogram Processing

The histogram, which is an important entity in image processing, depicts the number of occurrences of each possible gray level in an image. Consider the 4×4 8-bit gray level image shown in Table 2.2 (left). In order to find the histogram of the image, the histogram vector is initialized to zero. Its length is 256 since the range of gray levels is 0–255. All the pixel values of the image are scanned. Depending on the pixel value, the corresponding element in the histogram vector is incremented by 1. For example, the first pixel value is 249 and it occurs only once as indicated in the last column of the middle row of the histogram, shown in Table 2.3. The pixels with zero occurrences are not shown in the table.

Table 2.2 Pixel values of a 4×4 8-bit image (*left*) and its contrast-stretched version (*right*)

249	108	110	113
10	98	108	114
85	100	96	104
85	87	95	98

255	201	219	245
0	114	201	254
1	131	96	166
1	18	88	114

Table 2.3 Histograms of the input image and its contrast-stretched version. Pixels with zero occurrences are not shown

Gray level	10	85	87	95	96	98	100	104	108	110	113	114	249
Count	1	2	1	1	1	2	1	1	2	1	1	1	1
Gray level	0	1	18	88	96	114	131	166	201	219	245	254	255

Two images can have the same histogram. By modifying the histogram suitably, the image can be enhanced. While it is a simple process to construct a histogram of an image, it is very useful in several image processing tasks such as enhancement and segmentation. It is also a feature of an image. The distribution of the gray levels of an image gives useful information. Then, the histogram is used as such or modified to suit the requirements. Large number of pixels with values at the lower end of the gray level range indicates that the image is dark. Large number of pixels with values at the upper end indicates that the image is too bright. If most of the pixels have values in the middle, then the image contrast will not be good. In all these cases, contrast stretching or histogram equalization is possible for improving the image quality. The point is that a well spread out histogram over most of the range gives a better image. Contrast stretching increases the contrast, while histogram equalization enhances the contrast. The shape of the histogram remains the same in contrast stretching and it changes in histogram equalization. As in the case of any processing, the enhancement ability of these processes varies depending on the characteristics of the histogram of the input image.

2.2.1 Contrast Stretching

Let the range of gray levels before and after the transformation be the same, for example 0–255. Contrast is the difference between the maximum and minimum of the gray level range of the image. A higher difference results in a better contrast. Due to the limited dynamic range of the image recording device or underexposure, the gray levels of pixels may be concentrated only at some part of the allowable range. In general, some gray levels will lie outside the range intended for stretching. Let i and *inew* are the gray levels before and after contrast stretching. In this case, using the transformation

$$inew = \left\lfloor \frac{(I_{max} - I_{min} - 2)}{(M - L)}(i - L) \right\rfloor + 1, \quad L \leq i \leq M$$
$$inew = I_{min}, \quad i < L$$
$$inew = I_{max}, \quad i > M$$

the contrast of the image can be enhanced, where I_{min} and I_{max} are the values of the minimum and maximum of the allowable gray level range, and L and M are the values of the minimum and maximum of the part of the gray level range to be stretched. The gray levels outside the main range are given only single values.

Consider the 4×4 8-bit image shown in Table 2.2 (left). The histogram is shown in Table 2.3 (first 2 rows). The range of the gray levels is 0–255. With only 16 pixels in the image, most of the entries in the histogram are zero and they are not shown in the table. The point is that the histogram is concentrated in the range 85–114. Gray levels 10 and 249 are extreme values. As only a small part of the range of gray levels is used, the contrast of this type of images is poor. Contrast stretching is required to enhance the quality of the image. Now, the scale factor is computed as

$$\frac{255 - 0 - 2}{114 - 85} = 8.7241$$

For all those gray levels in the range 0–84, we assign the new gray level 0. For all those gray levels in the range 115–255, we assign the new gray level 255. For those gray levels in the range 85–114, the new value *inew* is computed from i as

$$inew = \lfloor 8.7241(i - 85) \rfloor + 1$$

The computation involves the *floor* function which rounds the numbers to the nearest integer towards minus infinity. For example, gray level 114 is mapped to

$$inew = \lfloor 8.7241(114 - 85) \rfloor + 1 = 253 + 1 = 254$$

The contrast stretched image is shown in Table 2.2 (right). The new histogram, which is well spread out, is also shown in Table 2.3 (last 2 rows).

While we have presented the basic procedure, the algorithm can be modified to suit the specific requirements. For example, selection of the range to be stretched and the handling of the other values have to be suitably decided.

Figure 2.4a shows a 256×256 8-bit image and (b) shows its histogram. The horizontal axis shows the gray levels and the vertical axis shows the count of the occurrence of the corresponding gray levels. The distribution of pixels is very heavy in the first half of the histogram. Therefore, the range of the histogram 0–104 is stretched and the rest compressed. The resulting image is shown in Fig. 2.4c and its histogram is shown in (d). While the dark areas got enhanced, the contrast of the brighter areas got deteriorated. Ideally, the pixels outside the range of stretching should have zero occurrences. Since it is unlikely in practical images, judgment is required to select the part to be stretched.

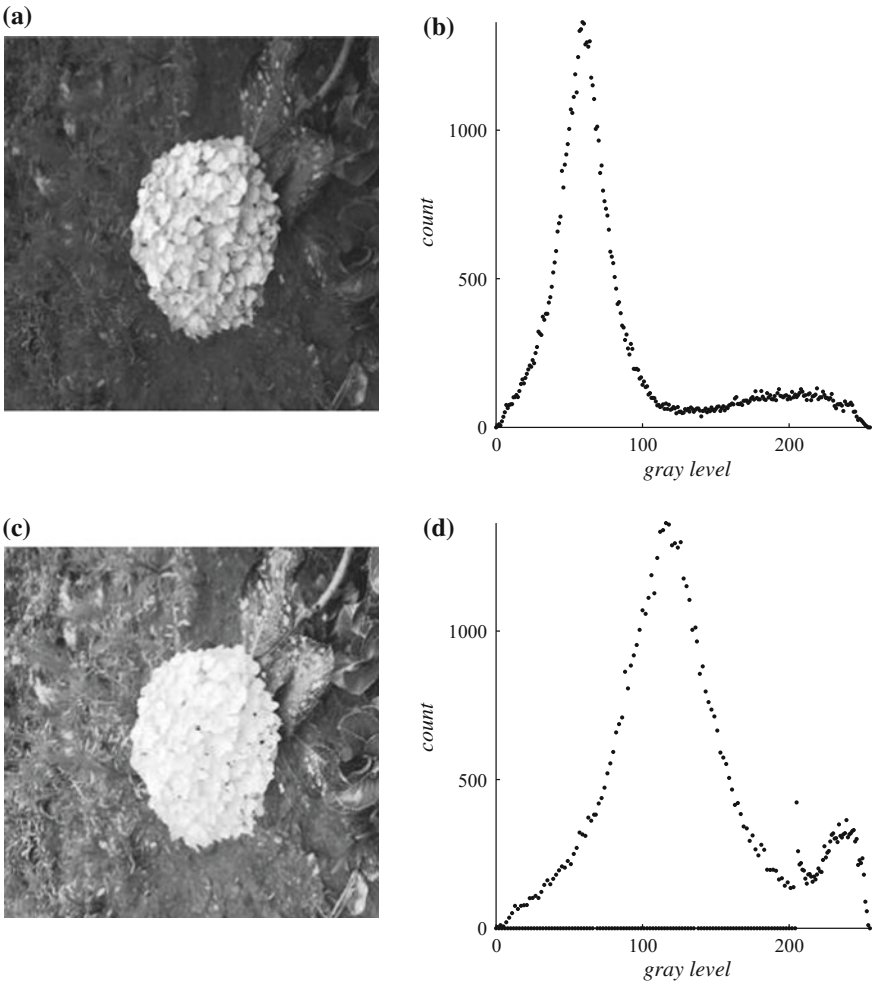


Fig. 2.4 **a** A 256×256 8-bit image; **b** its histogram; **c** the histogram-stretched image; **d** its histogram

2.2.2 Histogram Equalization

In both contrast stretching and histogram equalization, the objective is to spread the gray levels over the entire allowable gray level range. While stretching is a linear process and is reversible, equalization is a nonlinear process and is irreversible. Histogram equalization tries to redistribute about the same number of pixels for each gray level and it is automatic.

Consider the 4 × 4 4-bit image shown in Table 2.4 (left). The gray levels are in the range 0–15. The histogram of the image is shown in Table 2.5 (second row, *count_in*). It is more usually presented in a graphic form, as shown in Fig. 2.5a.

Table 2.4 A 4 × 4 4-bit image (*left*) and its histogram-equalized version (*right*)

13	14	2	14
10	2	5	9
15	15	3	15
15	8	13	1

9	11	3	11
8	3	5	7
15	15	4	15
15	6	9	1

Table 2.5 Histogram of the image and its equalized version

Gray level	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
count_in	0	1	2	1	0	1	0	0	1	1	1	0	0	2	2	4
count_eq	0	1	0	2	1	1	1	1	1	2	0	2	0	0	0	4

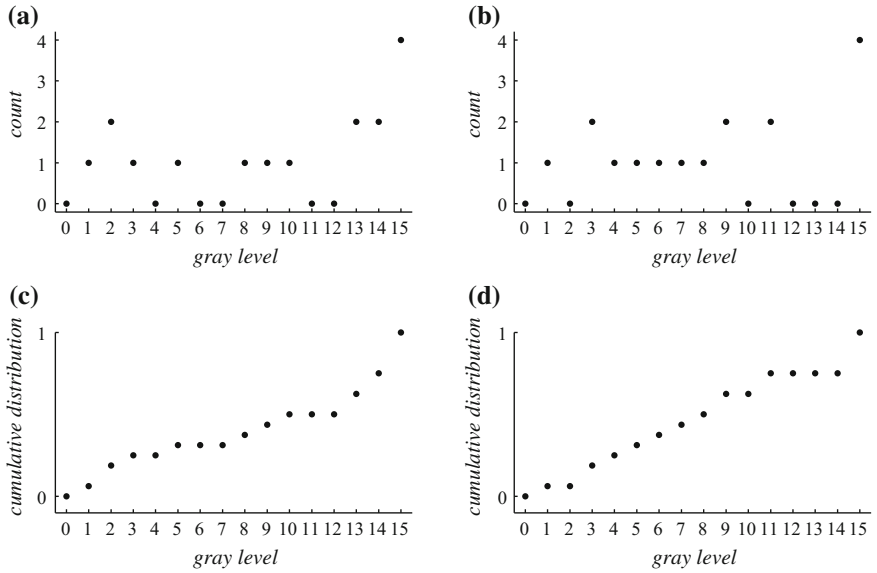


Fig. 2.5 **a** The histogram of the image shown in Table 2.4 (*left*); **b** the histogram of the histogram-equalized image shown in Table 2.4 (*right*); **c** the cumulative distribution of the image; **d** the cumulative distribution of the histogram-equalized image

The sum of the number of occurrences of all the gray levels must be equal to the number of pixels in the image. The histogram is normalized by dividing the number of occurrences by the total number of pixels. The normalized histogram of the image is obtained by dividing by 16 (the number of pixels in the image) as

$$\{0, 0.0625, 0.125, 0.0625, 0, 0.0625, 0, 0, 0.0625, 0.0625, 0.0625, 0, 0, 0.125, 0.125, 0.25\}$$

This is also the probability distribution of the gray levels. Often, the histograms of images are not evenly spread over the entire intensity range. The contrast of an image can be improved by making the histogram more uniformly spread. The more the number of occurrence of a gray level, the wider the spread it gets in the

equalized histogram. For a $N \times N$ image with L gray levels $\{u = 0, 1, \ldots, L-1\}$, the probability of occurrence of the uth gray level is

$$p(u) = \frac{n_u}{N^2}$$

where n_u is the number of occurrences of the pixel with gray level u. The equalization process for a gray level u of the input image is given by

$$v = (L-1)\sum_{n=0}^{u} p(n), \quad u = 0, 1, \ldots, L-1$$

where v is the corresponding gray level in the histogram equalized image. The justification for the process is as follows. The cumulative histogram value, up to gray level u, in the histogram of the input image should be covered up to gray level v in the histogram after equalization.

$$\sum_{n=0}^{u} hist(n) = \sum_{n=0}^{v} hist\_eq(n)$$

Since the new histogram is to be flat, for a $N \times N$ image with gray level values $0 - (L-1)$, the number of pixels for each gray level range is

$$\frac{N^2}{L-1}$$

The new cumulative histogram is

$$v\frac{N^2}{L-1}$$

Since

$$\sum_{n=0}^{u} hist(n) = v\frac{N^2}{L-1}, \quad v = (L-1)\frac{\sum_{n=0}^{u} hist(n)}{N^2} = (L-1)\sum_{n=0}^{u} p(n)$$

For the example image, the cumulative distribution of the pixel values are

$$\{0, 0.0625, 0.1875, 0.25, 0.25, 0.3125, 0.3125, 0.3125,$$
$$0.375, 0.4375, 0.5, 0.5, 0.5, 0.625, 0.75, 1\}$$

obtained by computing the cumulative sum of the probability distribution computed earlier and it is shown in Fig. 2.5c. These values, multiplied by $L - 1 = 15$, are

$$\{0, 0.9375, 2.8125, 3.75, 3.75, 4.6875, 4.6875, 4.6875,$$
$$5.625, 6.5625, 7.5, 7.5, 7.5, 9.375, 11.25, 15\}$$

The rounding of these values yields the equalized gray levels.

$$\{0, 1, 3, 4, 4, 5, 5, 5, 6, 7, 8, 8, 8, 9, 11, 15\}$$

Mapping the input image, using these values, we get the histogram equalized image shown in Table 2.4 (right). The equalized histogram of the image is shown in Fig. 2.5b and in Table 2.5 (third row, $count\_eq$). The cumulative distribution of the gray levels of the image is shown in Fig. 2.5d. It is clear from Fig. 2.5c, d that the gray level values are more evenly distributed in (d). In histogram equalization, the densely populated areas of the histogram are stretched and the sparsely populated areas are compressed. Overall, the contrast of the image is enhanced. So far, we considered the distribution of the pixels over the whole image. Of course, histogram processing can also be applied to sections of the image if it suits the purpose.

Figure 2.6a shows a 256 × 256 8-bit image and (b) shows the histograms of the image and its equalized version (c). Figure 2.6d shows the corresponding cumulative distributions of the gray levels. The cumulative distribution of the gray levels is a straight line for the histogram-equalized image. It is clear that equalization results in the even distribution of the gray levels. The histogram-equalized image looks better than that of the histogram-stretched image, shown in Fig. 2.4c.

As always, the effectiveness of an algorithm to do the required processing for the given data has to be checked out. Blind application of an algorithm for all data types is not recommended. For example, histogram equalization may or may not be effective for a certain image. If the number of pixels at either or both the ends of the histogram is large, equalization may not enhance the image. In these cases, an algorithm has to be modified or a new algorithm is used. The point is that the suitability of the characteristics of the image for the effective application of an algorithm is an important criterion in the selection of the algorithm.

2.2.3 Histogram Specification

In histogram equalization, the gray levels of the input image is redistributed in the equalized image so that its histogram approximates a uniform distribution. The distribution can be other than uniform. In certain cases where equalization algorithm is not effective, using a suitable distribution may become effective in enhancing the image. The histogram $a(n)$ of a reference image A is specified and the histogram $b(n)$ of the input image B is to be modified to produce an image C so that its distribution of pixels (histogram $c(n)$) is as similar to that of image A as possible. This process is useful in restoring an image from its modified version, if its original histogram is known. The steps of the algorithms are:

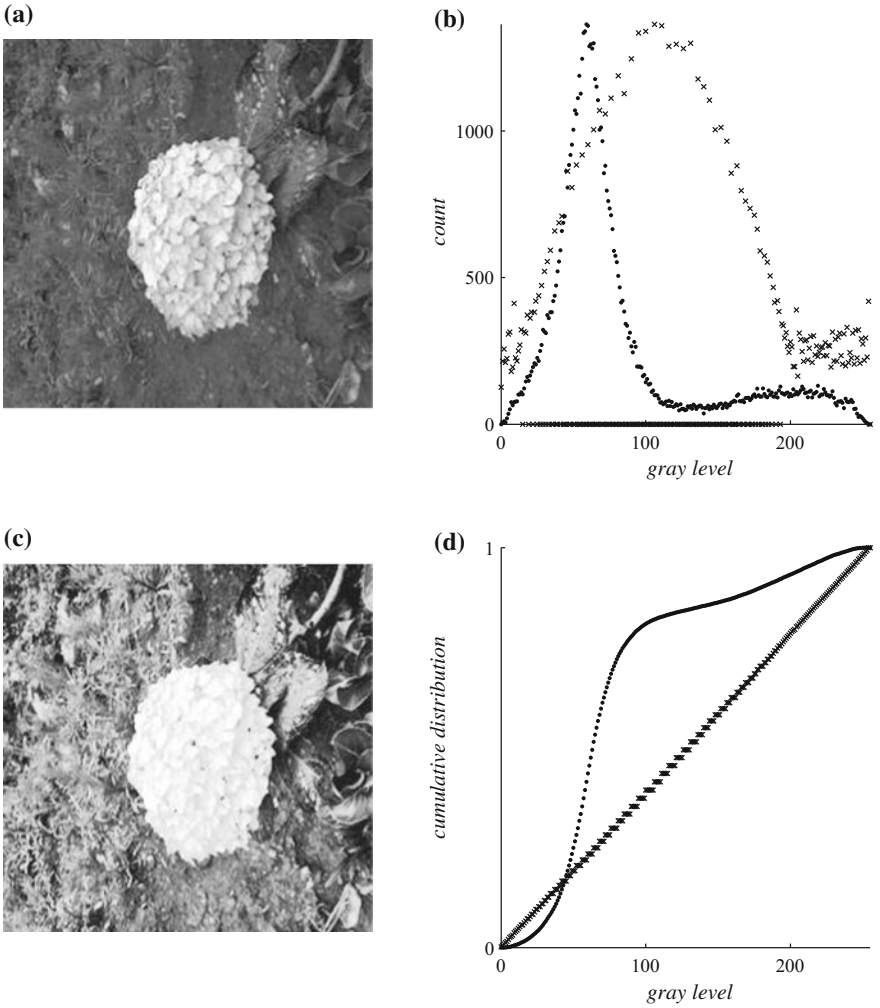


Fig. 2.6 **a** A 256 × 256 8-bit image; **b** the histograms of the image (*dot*) and its equalized version (*cross*) (**c**); **d** the corresponding cumulative distributions of the gray levels

1. Compute the cumulative distribution, $cum\_a(n)$, of the reference image A.
2. Compute the cumulative distribution, $cum\_b(k)$, of the input image B.
3. For each value in $cum\_b(k)$, find the minimum value in $cum\_a(n)$ that is greater than or equal to the current value in $cum\_b(k)$. That n is the new gray level in the image C corresponding to k in image B.

Consider the 4×4 4-bit reference (left) and input (right) images shown in Table 2.6. The histogram of the reference and input images, respectively, are

Table 2.6 4×4 4-bit reference (*left*) and input (*right*) images

8	8	8	8
8	8	8	8
8	8	8	8
8	8	8	8

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

$$\{0, 0, 0, 0, 0, 0, 0, 0, 16, 0, 0, 0, 0, 0, 0, 0\} \quad \text{and}$$
$$\{16, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0\}$$

The cumulative distribution, $cum\_a(n)$, of the reference image and the cumulative distribution, $cum\_b(k)$, of the input image, respectively, are

$$\{0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1\} \quad \text{and}$$
$$\{1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1\}$$

All the values in $cum\_b(k)$ map to $cum\_a(8)$ and all the pixels in the input image map to 8 in the output image. That is, the histograms of the reference and output images are the same. Let us interchange the reference and input images. Then, all the values in $cum\_b(k)$ map to $cum\_a(0)$ and all the pixels in the input image map to 0 in the output image.

As this problem is a generalization of the histogram equalization problem, let us do that example again following the 3 steps given above. In histogram equalization, the reference cumulative distribution values are those of the uniform probability distribution. Therefore, the values of $cum\_a(n)$ are

$$\{0, 0.0667, 0.1333, 0.2, 0.2667, 0.3333, 0.4, 0.4667, 0.5333,$$
$$0.6, 0.6667, 0.7333, 0.8, 0.8667, 0.9333, 1\}$$

From the equalization example, the values of $cum\_b(k)$ are

$$\{0, 0.0625, 0.1875, 0.25, 0.25, 0.3125, 0.3125, 0.3125,$$
$$0.375, 0.4375, 0.5, 0.5, 0.5, 0.625, 0.75, 1\}$$

The first value in $cum\_b(k)$ is zero. The minimum value greater than or equal to it in $cum\_a(n)$ is 0 and gray level value 0 maps to 0. Carrying out this process for all the values in $cum\_b(k)$, we get the equalized gray levels.

$$\{0, 1, 3, 4, 4, 5, 5, 5, 6, 7, 8, 8, 8, 10, 12, 15\}$$

These are about the same values obtained by equalization algorithm. Using these values the output image is created. Figure 2.7a shows the cumulative distributions of

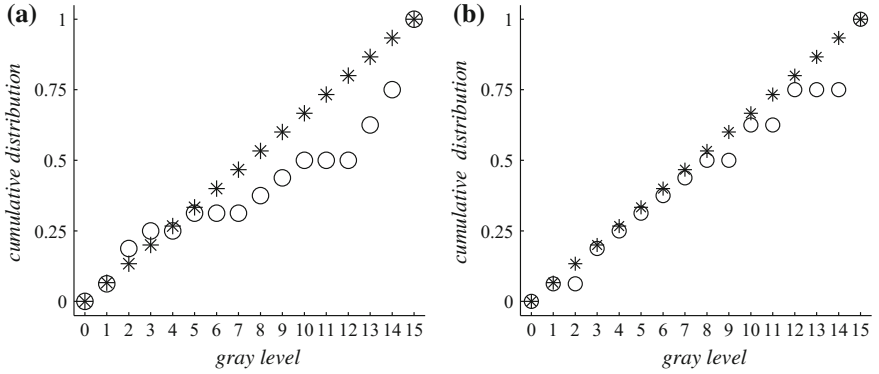


Fig. 2.7 **a** The cumulative distributions of the reference (∗) and input images (o); **b** The cumulative distributions of the reference (∗) and output images (o)

Table 2.7 4×4 reference, input and output images, respectively, from *left*

13	14	2	14
10	2	5	9
15	15	3	15
15	8	13	1

11	13	0	13
7	0	2	5
15	15	1	15
15	4	11	0

13	14	2	14
10	2	5	9
15	15	3	15
15	8	13	2

the reference (∗) and input images (o). Figure 2.7b shows the cumulative distributions of the reference (∗) and output images (o). The cumulative distribution of the output image is close to that of the uniform distribution.

Example images A, B and C are shown in Table 2.7. The normalized histogram of the reference image is

$$\{0, 0.0625, 0.1250, 0.0625, 0, 0.0625, 0, 0, 0.0625, \\ 0.0625, 0.0625, 0, 0, 0.1250, 0.1250, 0.25\}$$

The normalized histogram of the input image is

$$\{0.1875, 0.0625, 0.0625, 0, 0.0625, 0.0625, 0, 0.0625, \\ 0, 0, 0, 0.1250, 0, 0.1250, 0, 0.25\}$$

The cumulative distribution, $cum\_a(n)$, of the reference image is

$$\{0, 0.0625, 0.1875, 0.25, 0.25, 0.3125, 0.3125, 0.3125, \\ 0.3750, 0.4375, 0.5, 0.5, 0.5, 0.6250, 0.75, 1\}$$

The cumulative distribution, $cum\_b(k)$, of the input image is

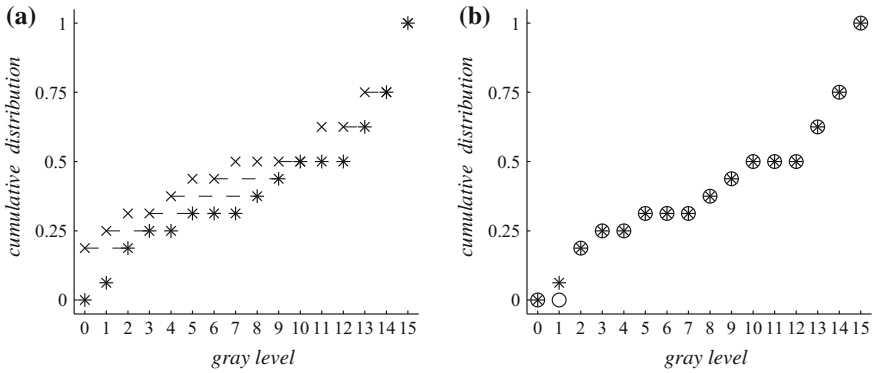


Fig. 2.8 **a** The cumulative distributions of the input (×) and reference (∗) images; **b** the cumulative distributions of the output (o) and reference (∗) images

$$\{0.1875, 0.25, 0.3125, 0.3125, 0.3750, 0.4375, 0.4375,$$
$$0.5, 0.5, 0.5, 0.5, 0.6250, 0.6250, 0.75, 0.75, 1\}$$

The cumulative distributions of the reference and input images are shown in Fig. 2.8a. Each value in $cum\_b(k)$ has to be mapped to the minimum value of $cum\_a(n)$ that is greater than or equal to $cum\_b(k)$. For example, the first value of $cum\_b(k)$ is 0.1875. The corresponding value is $cum\_a(2)$. That is, gray level 0 is mapped to 2 in the output image. Gray level with value 1 is mapped to 3 and so on. In Fig. 2.8a, the mappings are shown by dashed lines. Pixels of the input image in the range 0–15 are mapped to

$$\{2, 3, 5, 5, 8, 9, 9, 10, 10, 10, 10, 13, 13, 14, 14, 15\}$$

in the output image. Using these mappings, the output image is reconstructed (the rightmost in Table 2.7. The cumulative distribution of the output image is

$$\{0, 0, 0.1875, 0.25, 0.25, 0.3125, 0.3125, 0.3125, 0.3750, 0.4375,$$
$$0.5, 0.5, 0.5, 0.6250, 0.75, 1\}$$

The cumulative distributions of the reference and output images are almost the same, as shown in Fig. 2.8b.

Figure 2.9a, b show, respectively, a 256 × 256 8-bit image and its histogram. Figure 2.9c, d show, respectively, the restored image using histogram specification algorithm and its histogram.

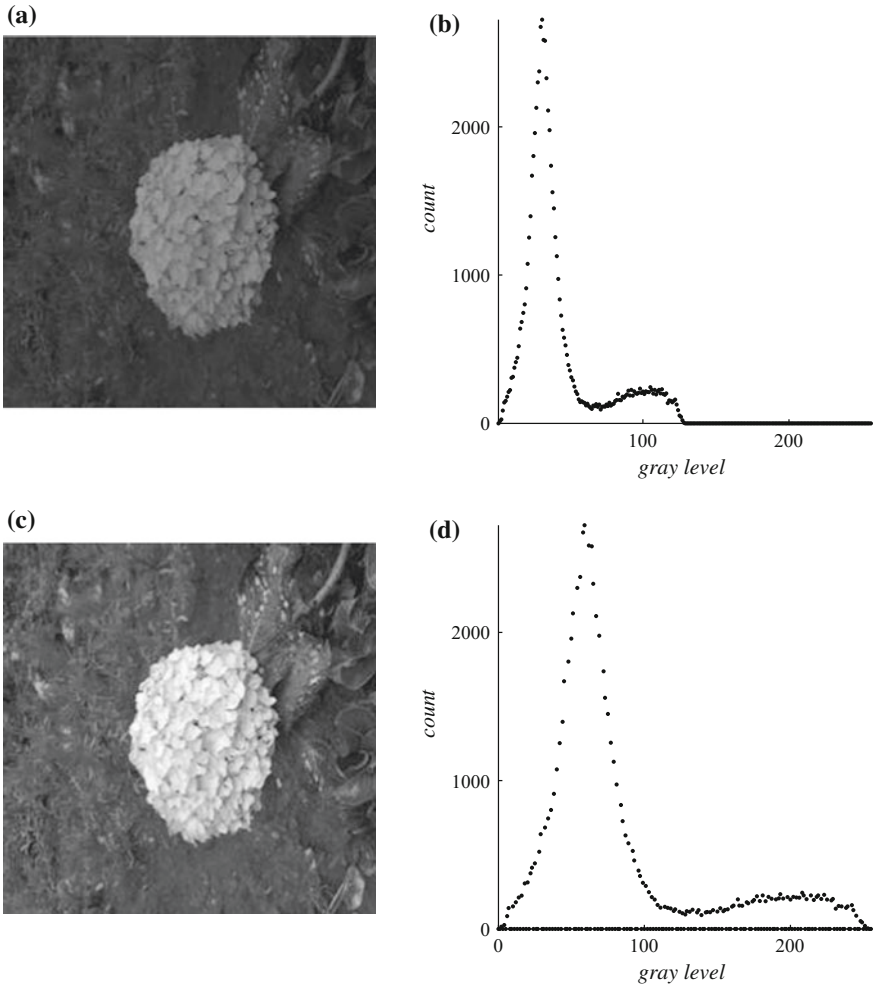


Fig. 2.9 **a** A 256 × 256 8-bit image and **b** its histogram; **c** the restored image using histogram specification algorithm and **d** its histogram

2.3 Thresholding

Thresholding operation is frequently used in image processing. It is used in tasks such as enhancement, segmentation and compression. A threshold indicates an intensity level of some significance. There are several variations of thresholding used in image processing. The first type is to threshold a gray level image to get a binary image. A threshold $T > 0$ is specified and all the gray levels with magnitude less than or equal to T are set to zero and the rest are set to 1.

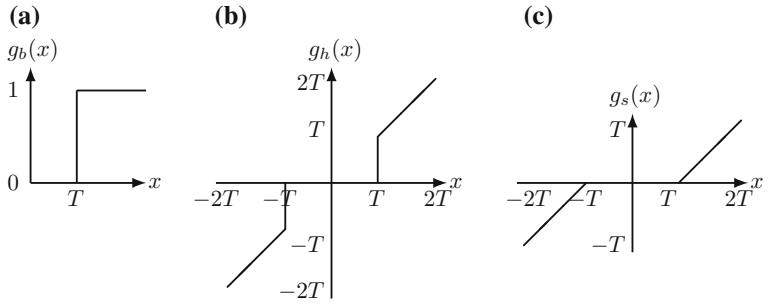


Fig. 2.10 **a** Binary thresholding; **b** Hard thresholding; **c** Soft thresholding

$$g_b(x) = \begin{cases} 0 & \text{if } x \leq \mathrm{T} \\ 1, & \text{otherwise} \end{cases}$$

This type of thresholding is shown in Fig. 2.10a.

In another type of thresholding, all the gray levels with magnitude less than or equal to T are set to zero and the rest are unaltered or set to the difference between the input values and the threshold. Hard thresholding, shown in Fig. 2.10b, is defined as

$$g_h(x) = \begin{cases} 0 & \text{if } |x| \leq \mathrm{T} \\ x, & \text{if } |x| > \mathrm{T} \end{cases}$$

In hard thresholding, the value of the function is retained, if its magnitude is greater than a chosen threshold value. Otherwise, the value of the function is set to zero. Typical application of this type of thresholding is in lossy image compression. A higher threshold gets a higher compression ratio at the cost of image quality. Soft thresholding, shown in Fig. 2.10c, is defined as

$$g_s(x) = \begin{cases} 0, & \text{if } |x| \leq \mathrm{T} \\ x - T, & \text{if } x > \mathrm{T} \\ x + T, & \text{if } x < -\mathrm{T} \end{cases}$$

The difference in soft thresholding is that the value of the function is made closer to zero by adding or subtracting the chosen threshold value from it, if its magnitude is greater than the threshold. A typical application of soft thresholding is in denoising. Thresholding is easily extended to multiple levels.

Figure 2.11a shows a damped sinusoid. Figure 2.11b shows the damped sinusoid hard thresholded with level $T = 0.3$. Values less than or equal to 0.3 have been assigned the value zero. Figure 2.11c shows the damped sinusoid soft thresholded with level $T = 0.3$. Values less than or equal to 0.3 have been assigned the value zero and values greater than 0.3 have been assigned values closer to zero by 0.3. Figure 2.11d shows the damped sinusoid binary thresholded with level $T = 0.3$.

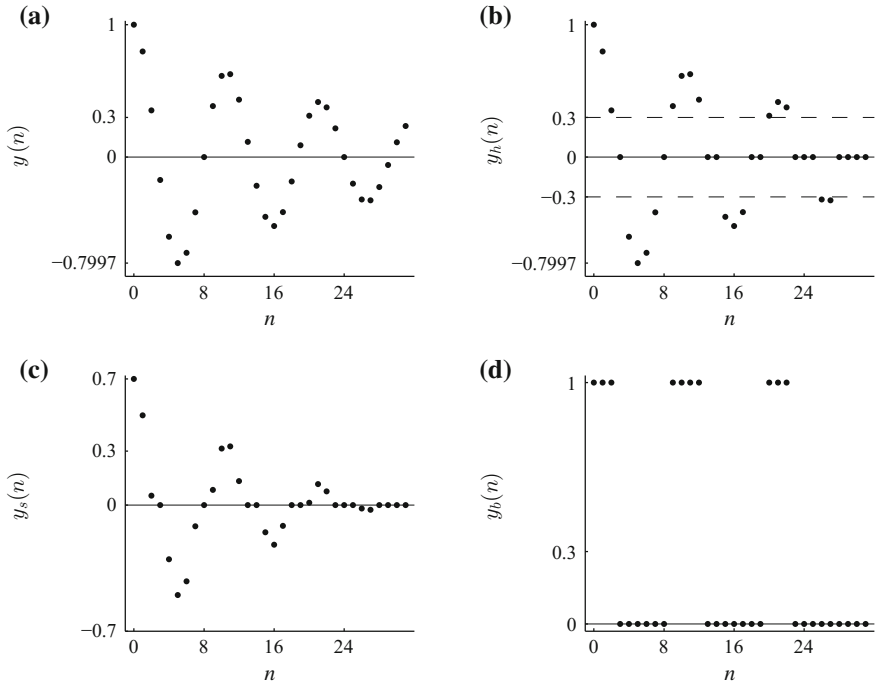


Fig. 2.11 **a** A damped sinusoid; **b** hard, **c** soft and **d** binary thresholding of the sinusoid with $T = 0.3$

Values less than or equal to 0.3 have been assigned the value zero and values greater than 0.3 have been assigned the value 1.

Consider the 8×8 8-bit gray level image shown by the left matrix.

117	170	130	54	84	209	164	148
135	151	137	96	56	157	225	189
136	152	174	146	64	84	146	90
123	139	182	133	51	71	56	74
119	137	172	146	119	67	65	70
90	123	166	184	203	101	49	64
85	102	162	194	164	80	38	56
73	84	155	185	147	163	87	57

0	1	1	0	0	1	1	1
1	1	1	0	0	1	1	1
1	1	1	1	0	0	1	0
1	1	1	1	0	0	0	0
0	1	1	1	0	0	0	0
0	1	1	1	1	0	0	0
0	0	1	1	1	0	0	0
0	0	1	1	1	1	0	0

The result of binary thresholding with $T = 120$ is shown in the right matrix. The results of hard and soft thresholding with $T = 120$ are shown in the left and right matrices, respectively.

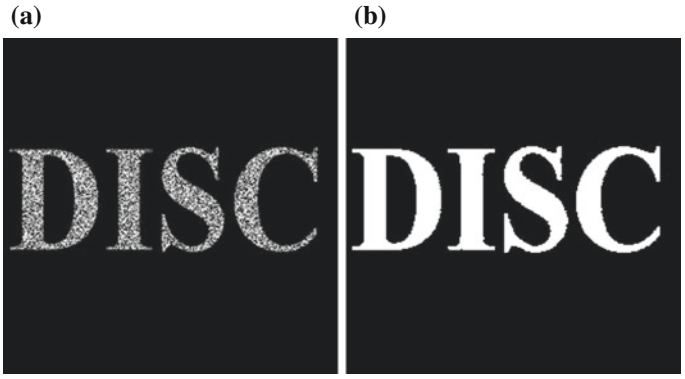


Fig. 2.12 **a** A 256 × 256 image and **b** its threshold version with $T = 1$

0	170	130	0	0	209	164	148
135	151	137	0	0	157	225	189
136	152	174	146	0	0	146	0
123	139	182	133	0	0	0	0
0	137	172	146	0	0	0	0
0	123	166	184	203	0	0	0
0	0	162	194	164	0	0	0
0	0	155	185	147	163	0	0

0	50	10	0	0	89	44	28
15	31	17	0	0	37	105	69
16	32	54	26	0	0	26	0
3	19	62	13	0	0	0	0
0	17	52	26	0	0	0	0
0	3	46	64	83	0	0	0
0	0	42	74	44	0	0	0
0	0	35	65	27	43	0	0

Figure 2.12a shows a 256 × 256 image. The image is corrupted with noise and the letters are not clear. The white pixels showing the letters have values varying from 2 to 255. Therefore, with the threshold $T = 1$, setting all the pixels greater than 1 to 255 with the rest set to 0 enhances the image, as shown in Fig. 2.12b.

2.4 Neighborhood Operations

In this type of processing, called neighborhood operation, each pixel value is replaced by another, which is a linear or nonlinear function of the values of the pixels in its neighborhood. The area of a square or rectangle or circle (sometimes of other shapes) forming the neighborhood is called a window. Typical window sizes vary from 3 × 3 to 11 × 11. If the window size is 1 × 1 (the neighborhood consists of the pixel itself), then the operation is called the point operation. The window is moved over the image row by row and column by column and the same operation is carried out for each pixel.

A 3×3 window of the pixel $x(m, n)$ is

$$\begin{bmatrix} x(m-1, n-1) & x(m-1, n) & x(m-1, n+1) \\ x(m, n-1) & x(m, n) & x(m, n+1) \\ x(m+1, n-1) & x(m+1, n) & x(m+1, n+1) \end{bmatrix}$$

The set of pixels (strong neighbors)

$$\{x(m-1, n), x(m, n+1), x(m+1, n), x(m, n-1)$$

is called the 4-neighbors of $x(m, n)$.

$$\begin{bmatrix} & x(m-1, n) & \\ x(m, n-1) & x(m, n) & x(m, n+1) \\ & x(m+1, n) & \end{bmatrix}$$

The distance between these pixels and $x(m, n)$ is 1. The other 4 pixels (weak neighbors) are diagonal neighbors of $x(m, n)$. All the neighbors in the window are called the 8-neighbors of $x(m, n)$.

Border Extension

If the complete window is to overlap the image pixels, then the output image after a neighborhood operation will be smaller. This is due to the fact that the required pixels are not defined at the borders. Then, we have to accept a smaller output image or extend the input image at the borders suitably. For example, many operations are based on convolving an image with the impulse response or coefficient matrix. When trying to find the convolution output corresponding to the pixels located in the vicinity of the borders, some of the required pixels are not available. Obviously, we can assume that the values are zero. This method of border extension is called zero-padding. Of course, when this method is not suitable, there are other possibilities. Consider the 4×4 image

23	51	23	32
32	44	44	23
23	23	44	32
44	44	23	23

Some of the commonly used image extensions are given below. Any other suitable extension can also be used.

The symmetric extension of the image by 2 rows and 2 columns on all sides yields

44	32	32	44	44	23	23	44
51	23	23	51	23	32	32	23
51	23	23	51	23	32	32	23
44	32	32	44	44	23	23	44
23	23	23	23	44	32	32	44
44	44	44	44	23	23	23	23
44	44	44	44	23	23	23	23
23	23	23	23	44	32	32	44

The extension is the mirror image of itself at the borders.

The replication method of extension of the image by 2 rows and 2 columns on all sides yields

23	23	23	51	23	32	32	32
23	23	23	51	23	32	32	32
23	23	23	51	23	32	32	32
32	32	32	44	44	23	23	23
23	23	23	23	44	32	32	32
44	44	44	44	23	23	23	23
44	44	44	44	23	23	23	23
44	44	44	44	23	23	23	23

Border values are repeated.

The periodic extension of the image by 2 rows and 2 columns on all sides yields

44	32	23	23	44	32	23	23
23	23	44	44	23	23	44	44
23	32	23	51	23	32	23	51
44	23	32	44	44	23	32	44
44	32	23	23	44	32	23	23
23	23	44	44	23	23	44	44
23	32	23	51	23	32	23	51
44	23	32	44	44	23	32	44

This extension considers the image as one period of a 2-D periodic signal. The top and bottom edges are considered adjacent and so are the right and left edges.

2.4.1 Linear Filtering

A filter, in general, is a device that passes the desirable part of its input. In the context of image processing, a filter modifies the spectrum of an image in a specified manner. This modification can be done either in the spatial domain or frequency domain.

The choice primarily depends of the size of the filter among other considerations. A linear filter is characterized by its impulse response, which is its response for a unit-impulse input with zero initial conditions. For enhancement purposes, a filter is used to improve the quality of an image for human or machine perception. The improvement in the quality of an image is evaluated subjectively. Two types of filters, lowpass and highpass, are often used to improve the quality. A lowpass filter is essentially an integrator, passing the low frequency components and suppressing the high frequency components. For example, the integral of $\cos(\omega t)$ is $\sin(\omega t)/\omega$. The higher the frequency, the higher is the attenuation of the frequency component after integration. A highpass filter is essentially a differentiator that suppresses the low frequency components. The derivative of $\sin(\omega t)$ is $\omega \cos(\omega t)$. The higher the frequency, the higher is the amplification of the frequency component after differentiation.

In linear filtering, convolution operation is a convenient system model. It relates the input and output of a system through its impulse response. Although the image is a 2-D signal, its processing can often be carried out using the corresponding 1-D operations repeatedly over the rows and columns. Conceptually, 1-D operations are easier to understand. Further, 2-D convolution is a straightforward extension of that of the 1-D. Therefore, we present the 1-D convolution briefly. First, as it is so important (along with Fourier analysis), we present a simple example to explain the concept.

Consider the problem of finding the amount in our bank account for the deposits on a yearly basis. We are familiar that, for compound interest, the amount of interest paid increases from year to year. Let the annual interest rate be 10%. Then, an amount of \$1 will be \$1 at the time of deposit, \$1.1 after 1 year, \$1.21 after 2 years and so on, as shown in Fig. 2.13a. Let our current deposit be \$200, \$300 a year before and \$100 two years before, as shown in Fig. 2.13b. The problem is to find the current balance in the account. From Fig. 2.13a, b, it is obvious that if we reverse the order of numbers in (a), shift and multiply with the corresponding numbers in (b) and sum the products, we get the current balance \$651, as shown in Fig. 2.13c.

Of course, we could have reversed the order of the numbers in (b) either. For longer sets of numbers, we can repeat the operation. This is convolution operation and it is simple. It is basically a sum of products of two sequences, after either one (not both) is time-reversed. In formal description, the set of interest rates is called as the system impulse response. The set of deposits is called the input to the system. The set of balances at different time periods is called the system output. Convolution relates the input and the impulse response of a system to its output.

1-D Linear Convolution

The 1-D linear convolution of two aperiodic sequences $x(n)$ and $h(n)$ is defined as

$$y(n) = \sum_{k=-\infty}^{\infty} x(k)h(n-k) = \sum_{k=-\infty}^{\infty} h(k)x(n-k) = x(n) * h(n) = h(n) * x(n)$$

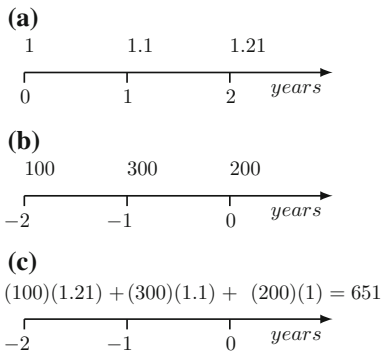


Fig. 2.13 Basics of linear convolution. **a** annual interest rate; **b** deposits; **c** computation of current balance

The convolution operation relates the input $x(n)$, the output $y(n)$ and the impulse response $h(n)$ of a system. The impulse response, which characterizes the system in the time-domain, is the response of a relaxed (initial conditions are zero) system for the unit-impulse $\delta(n)$. A discrete unit-impulse signal is defined as

$$\delta(n) = \begin{cases} 1, \text{ for } n = 0 \\ 0, \text{ for } n \neq 0 \end{cases}$$

It is an all-zero sequence, except that its value is one when its argument n is equal to zero. The input $x(n)$ is decomposed into a sum of scaled and delayed unit-impulses. The response to each impulse is found and the superposition summation of all the responses is the system output. It can also be considered as the weighted average of sections of the input with the weighting sequence being the impulse response.

Figure 2.14 shows the convolution of the signal $\{x(0) = 4, x(1) = 3, x(2) = 1, x(3) = 2$ and $\{h(0) = 1, h(1) = -2, h(2) = 1$. The output $y(0)$, from the definition, is

$$y(0) = x(k)h(0-k) = (4)(1) = 4,$$

where $h(0-k)$ is the time-reversal of $h(k)$. Shifting $h(0-k)$ to the right, we get the remaining outputs as

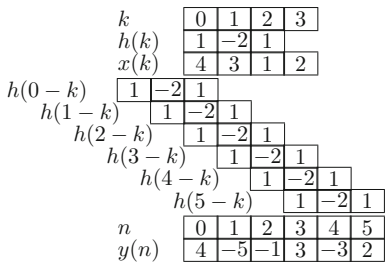


Fig. 2.14 1-D linear convolution

$$
\begin{aligned}
y(1) &= x(k)h(1-k) = (4)(-2) + (3)(1) = -5 \\
y(2) &= x(k)h(2-k) = (4)(1) + (3)(-2) + (1)(1) = -1 \\
y(3) &= x(k)h(3-k) = (3)(1) + (1)(-2) + (2)(1) = 3 \\
y(4) &= x(k)h(4-k) = (1)(1) + (2)(-2) = -3 \\
y(5) &= x(k)h(5-k) = (2)(1) = 2
\end{aligned}
$$

Outside the defined values of $x(n)$, we have assumed zero values. As mentioned earlier, a suitable extension of the input, to get a convolution output of the same length, should be made to suit the requirements of the problem. The six convolution output values are called the full convolution output. Most of the times, the central part of the output, of the same size as the input, is required. If the window is to be confined inside the input data, the size of the output will be smaller than that of the input.

2-D Linear Convolution

In the 2-D convolution, a 2-D window is moved over the image. The convolution of images $x(m, n)$ and $h(m, n)$ is defined as

$$
\begin{aligned}
y(m,n) &= \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} x(k,l)h(m-k, n-l) \\
&= \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} h(k,l)x(m-k, n-l) = h(m,n) * x(m,n)
\end{aligned}
$$

Four operations, similar to those of the 1-D convolution, are repeatedly executed in carrying out the 2-D convolution.

1. One of the images, say $h(k, l)$, is rotated in the (k, l) plane by 180° about the origin to get $h(-k, -l)$. The same effect is achieved by folding the image about the k axis to get $h(k, -l)$ and then, folding the resulting image about the l axis.
2. The rotated image is shifted by (m, n) to get $h(m - k, n - l)$.
3. The products $x(k, l)h(m - k, n - l)$ of all the overlapping samples are found.
4. The sum of all the products yields the convolution output $y(m, n)$ at (m, n).

Consider the convolution of the 3×3 image $h(k, l)$ and the 4×4 image $x(k, l)$

$$
h(k,l) = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \quad \text{and} \quad x(k,l) = \begin{bmatrix} 1 & -1 & 3 & 2 \\ 2 & 1 & 2 & 4 \\ 1 & -1 & 2 & -2 \\ 3 & 1 & 2 & 2 \end{bmatrix}
$$

shown in Fig. 2.15. Four examples of computing the convolution output are shown. For example, with a shift of $(0-k, 0-l)$, there is only one overlapping pair $(1, -1)$. The product of these numbers is the output $y(0, 0) = -1$. The process is repeated to

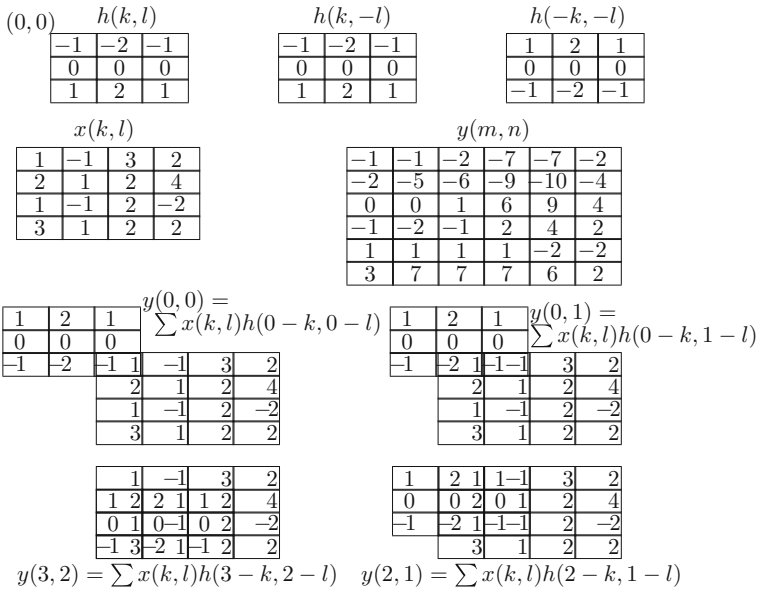


Fig. 2.15 2-D linear convolution

get the complete convolution output $y(m, n)$ shown in the figure. We assumed that the pixel values outside the defined region of the image are zero. This assumption may or may not be suitable. Some other commonly used borer extensions are based on periodicity, symmetry or replication, as presented earlier.

Lowpass Filtering

The output of convolution for a given input depends on the impulse response of the system. In lowpass filtering, the frequency response corresponding to the impulse response will be of lowpass nature. The system readily passes the low frequency components of the signal and suppresses the high frequency components. Low frequency components vary slowly compared with the bumpy nature of the high frequency components. Lowpass filtering is typically used for deliberate blurring to remove unwanted details of an image and reduce the noise content of the image. The impulse response of the simplest and widely used 3×3 lowpass filter, called the averaging filter, is

$$h(m,n) = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & \mathbf{1} & 1 \\ 1 & 1 & 1 \end{bmatrix}, \quad m = -1, 0, 1, \quad n = -1, 0, 1$$

The origin of the filter is shown in boldface. All the coefficient values are the same. Other filters produce weighted average outputs. This filter, when applied to an image, replaces each pixel in the input by the average of the values of a set of its neighboring pixels. Pixel $x(m, n)$ is replaced by the value

$$y(m,n) = \frac{1}{9}(x(m-1,n-1) + x(m-1,n) + x(m-1,n+1) + x(m,n-1) + x(m,n) + x(m,n+1) + x(m+1,n-1) + x(m+1,n) + x(m+1,n+1))$$

The bumps are smoothed out due to averaging. Blurring will proportionally increase with larger filters. This filter is separable. Multiplying the 3×1 column filter $h_c(m) = \{1, 1, 1\}^T/3$ with the 1×3 row filter $h_r(n) = \{1, 1, 1\}/3$, which is the transpose of the column filter, we obtain the 3×3 averaging filter.

$$h(m,n) = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \frac{1}{3}\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \frac{1}{3}\begin{bmatrix} 1 & 1 & 1 \end{bmatrix} = h_c(m)h_r(n)$$

This implies that the computational complexity is reduced by convolving each row of the input image with the row filter first and then convolving each column of the result with the column filter or vice versa. With the 2-D filter $h(m,n)$ separable, $h(m,n) = h_c(m)h_r(n)$ and, with input $x(m,n)$,

$$\begin{aligned} h(m,n) * x(m,n) &= (h_c(m)h_r(n)) * x(m,n) \\ &= (h_c(m) * x(m,n)) * h_r(n) = h_c(m) * (x(m,n) * h_r(n)) \end{aligned}$$

$$y(k,l) = \sum_m h_c(m) \sum_n h_r(n)x(k-m,l-n) = \sum_n h_r(n) \sum_m h_c(m)x(k-m,l-n)$$

Whenever a filter is separable, it is advantageous to decompose a 2-D operation into a pair of 1-D operations.

Let the input be

$$x(m,n) = \begin{bmatrix} 1 & -1 & 3 & 2 \\ 2 & 1 & 2 & 4 \\ 1 & -1 & 2 & -2 \\ 3 & 1 & 2 & 2 \end{bmatrix}$$

Assuming zero-padding at the borders, the output of 1-D filtering of the rows of the input and the output of 1-D filtering of the columns of the partial output are, respectively,

$$yr(m,n) = \frac{1}{3}\begin{bmatrix} 0 & 3 & 4 & 5 \\ 3 & 5 & 7 & 6 \\ 0 & 2 & -1 & 0 \\ 4 & 6 & 5 & 4 \end{bmatrix} \qquad y(m,n) = \frac{1}{9}\begin{bmatrix} 3 & 8 & 11 & 11 \\ 3 & 10 & 10 & 11 \\ 7 & 13 & 11 & 10 \\ 4 & 8 & 4 & 4 \end{bmatrix}$$

Assuming replication at the borders, the extended input and the output are, respectively,

$$xe(m,n) = \begin{bmatrix} 1 & 1 & -1 & 3 & 2 & 2 \\ 1 & 1 & -1 & 3 & 2 & 2 \\ 2 & 2 & 1 & 2 & 4 & 4 \\ 1 & 1 & -1 & 2 & -2 & -2 \\ 3 & 3 & 1 & 2 & 2 & 2 \\ 3 & 3 & 1 & 2 & 2 & 2 \end{bmatrix} \qquad y(m,n) = \frac{1}{9}\begin{bmatrix} 7 & 11 & 15 & 24 \\ 7 & 10 & 10 & 15 \\ 13 & 13 & 11 & 14 \\ 15 & 14 & 9 & 10 \end{bmatrix}$$

Only the output at the borders differ with different border extensions. The central part of the output is the same.

Gaussian Lowpass Filter

The 2-D Gaussian function is a lowpass filter, with a bell-shaped impulse response (frequency response) in the spatial domain (frequency domain). The Gaussian lowpass filters are based on Gaussian probability distribution function. The impulse response $h(m,n)$ of the Gaussian $N \times N$ lowpass filter, with the standard deviation σ, is given by

$$h(m,n) = \frac{e^{-\frac{(m^2+n^2)}{(2\sigma^2)}}}{K}, \qquad K = \sum_{m=-(N-1)/2}^{(N-1)/2} \sum_{n=-(N-1)/2}^{(N-1)/2} e^{-\frac{(m^2+n^2)}{(2\sigma^2)}}$$

assuming N is odd. The larger the value of the standard deviation σ, the flatter is the filter impulse response. For very large value of σ, as it appears squared in the denominator of the exponent of the exponential function of the defining equation, it tends to the averaging filter in the limit. The impulse response of the Gaussian lowpass filters with $\sigma = 2$, of size 11×11 and 12×12, are shown in Fig. 2.16a, b, respectively. The impulse response of the Gaussian 3×3 lowpass filter, with $\sigma = 0.5$, is

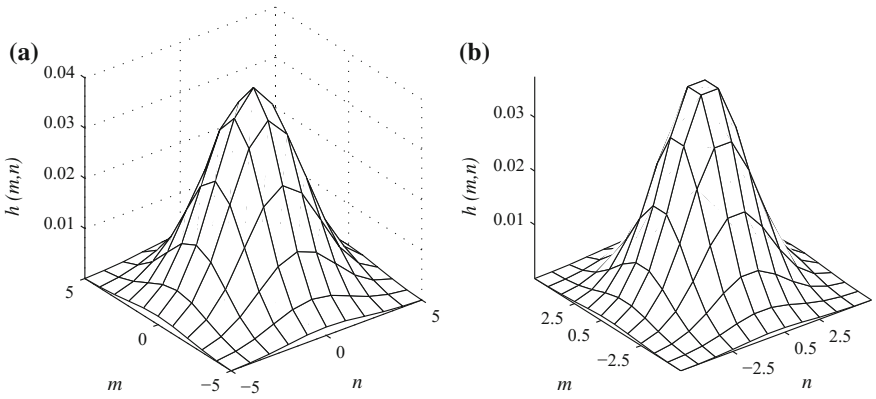


Fig. 2.16 The impulse response of the Gaussian lowpass filters with $\sigma = 2$. **a** 11×11; **b** 12×12

$$h(m,n) = \begin{bmatrix} 0.0113 & 0.0838 & 0.0113 \\ 0.0838 & \mathbf{0.6193} & 0.0838 \\ 0.0113 & 0.0838 & 0.0113 \end{bmatrix}, \quad m = -1, 0, 1, \quad n = -1, 0, 1$$

The origin of the filter is shown in boldface. For example, let $m = n = 0$ in the defining equation for $h(m, n)$. Then, the numerator is 1.

$$\begin{aligned} K &= (e^{-2(1+1)} + e^{-2(0+1)} + e^{-2(1+1)} + e^{-2(1+0)} + e^{-2(0+0)} \\ &\quad + e^{-2(1+0)} + e^{-2(1+1)} + e^{-2(0+1)} + e^{-2(1+1)}) \\ &= e^{-4} + e^{-2} + e^{-4} + e^{-2} + 1 + e^{-2} + e^{-4} + e^{-2} + e^{-4} \\ &= 4e^{-4} + 4e^{-2} + 1 = 1.6146 \end{aligned}$$

The inverse of 1.6146 is $0.6193 = h(0, 0)$. This filter is also separable. Multiplying the 3×1 column filter $\{0.1065, 0.7870, 0.1065\}^T$ with the 1×3 row filter $\{0.1065, 0.7870, 0.1065\}$, which is the transpose of the column filter, we obtain the 3×3 Gaussian filter.

The Gaussian filter is widely used. The features of this filter include:

1. There is no directional bias, since it is symmetric.
2. By varying the value of the standard deviation σ, the conflicting requirement of less blurring and more noise removal is controlled.
3. The filter is separable.
4. The coefficients fall off to negligible levels at the edges.
5. The Fourier transform of a Gaussian function is another Gaussian function.
6. The convolution of two Gaussian functions is another Gaussian function.

Let

$$x(m,n) = \begin{bmatrix} 1 & -1 & 3 & 2 \\ 2 & 1 & 2 & 4 \\ 1 & -1 & 2 & -2 \\ 3 & 1 & 2 & 2 \end{bmatrix}$$

Assuming zero-padding at the borders, the output of 1-D filtering of the rows of the input and the output of 1-D filtering of the columns of the partial output are, respectively,

$$\begin{bmatrix} 0.6805 & -0.3610 & 2.4675 & 1.8935 \\ 1.6805 & 1.2130 & 2.1065 & 3.3610 \\ 0.6805 & -0.4675 & 1.2545 & -1.3610 \\ 2.4675 & 1.3195 & 1.8935 & 1.7870 \end{bmatrix} \quad y(m,n) = \begin{bmatrix} 0.7145 & -0.1549 & 2.1663 & 1.8481 \\ 1.4675 & 0.8664 & 2.0542 & 2.7018 \\ 0.9773 & -0.0982 & 1.4133 & -0.5228 \\ 2.0144 & 0.9887 & 1.6238 & 1.2614 \end{bmatrix}$$

Assuming periodicity at the borders, the extended input and the output are, respectively,

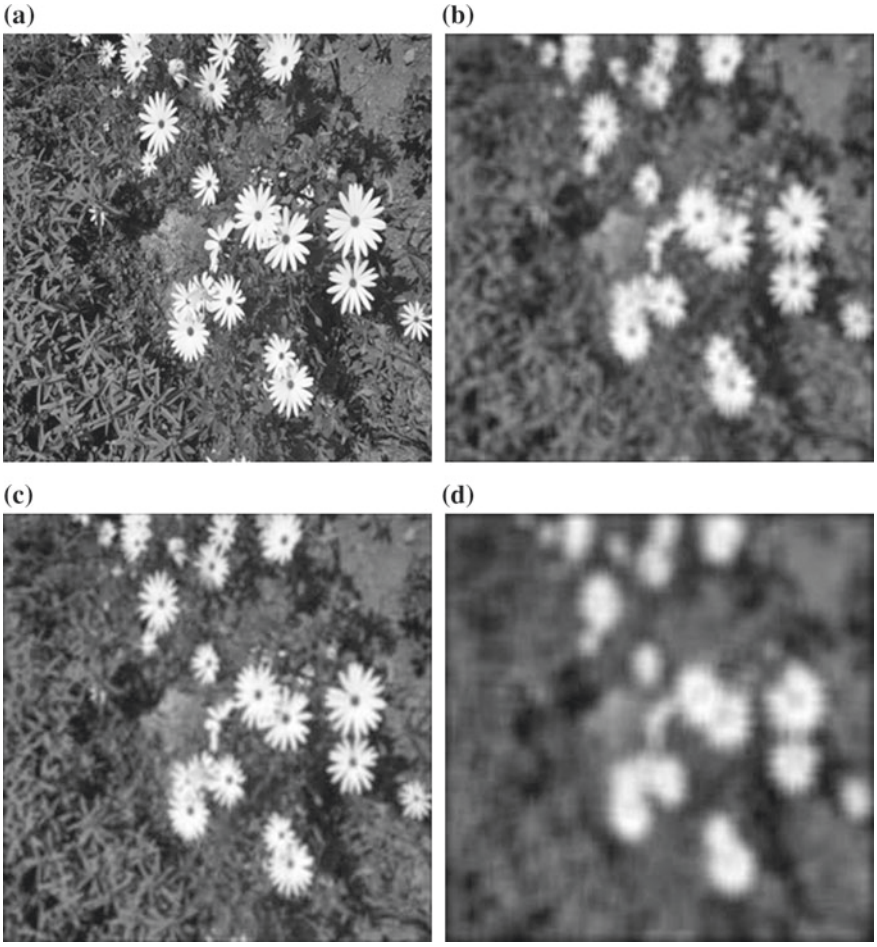


Fig. 2.17 **a** A 256 × 256 8-bit image; **b** filtered image with 5 × 5 averaging filter; **c** filtered image with 5 × 5 Gaussian filter with $\sigma = 1$; **d** filtered image with 11 × 11 averaging filter

$$xe(m,n) = \begin{bmatrix} 2 & 3 & 1 & 2 & 2 & 3 \\ 2 & 1 & -1 & 3 & 2 & 1 \\ 4 & 2 & 1 & 2 & 4 & 2 \\ -2 & 1 & -1 & 2 & -2 & 1 \\ 2 & 3 & 1 & 2 & 2 & 3 \\ 2 & 1 & -1 & 3 & 2 & 1 \end{bmatrix} \quad y(m,n) = \begin{bmatrix} 1.2130 & -0.0143 & 2.3679 & 2.1790 \\ 1.8027 & 0.8664 & 2.0542 & 2.8921 \\ 0.8777 & -0.0982 & 1.4133 & -0.3822 \\ 2.2545 & 0.9502 & 1.8866 & 1.7372 \end{bmatrix}$$

Figure 2.17a shows a 256 × 256 8-bit gray level image. Figure 2.17b, d show the filtered images with 5 × 5 and 11 × 11 averaging filters, respectively. Obviously, the blurring of the image is more with the larger filter. Figure 2.17c shows the filtered image with 5 × 5 Gaussian filter with $\sigma = 1$. As the passband spectrum of the

averaging filter, due to sharp transition at the borders, is relatively narrow, the blurring is more for the same size window. As the Gaussian filter is smooth, it has a relatively wider spectrum and the blurring is less.

Highpass Filtering

Frequency, in image processing, is the rate of change of gray levels of an image with respect to distance. A high frequency component is characterized by large changes in gray levels over short distances and vice versa. Highpass filters pass high frequency components and suppress low frequency components. This type of filters are used for sharpening images and edge detection. Images often get blurred and may require sharpening. Blurring corresponds to integration and sharpening corresponds to differentiation and they undo the effects of the other. High frequency components may have to be enhanced by suppressing low frequency components.

Laplacian Highpass Filter

While the first-order derivative is also a highpass filter, the Laplacian filter is formed using the second-order derivative. A peak is the indicator of an edge by the first-order derivative and it is the zero-crossing by the second-order derivative. The Laplacian operator of a function $f(x, y)$

$$\nabla^2 f(x, y) = \frac{\partial^2 f(x, y)}{\partial x^2} + \frac{\partial^2 f(x, y)}{\partial y^2}$$

is an often used linear derivative operator. It is isotropic (invariant with respect to direction). Consider the 4-neighborhood

$$\begin{bmatrix} & x(m-1,n) & \\ x(m,n-1) & x(m,n) & x(m,n+1) \\ & x(m+1,n) & \end{bmatrix}$$

For discrete signals, differencing approximates differentiation. At the point $x(m, n)$, the first differences along the horizontal and vertical directions, $\Delta_h(m, n)$ and $\Delta_v(m, n)$, are defined as

$$\Delta_h x(m, n) = x(m, n) - x(m, n-1) \quad \text{and} \quad \Delta_v x(m, n) = x(m, n) - x(m-1, n)$$

Using the first differences again, we get the second differences.

$$\begin{aligned}\Delta_v^2 x(m,n) &= \Delta_v x(m+1,n) - \Delta_v x(m,n)\\ &= (x(m+1,n) - x(m,n)) - (x(m,n) - x(m-1,n))\\ &= x(m+1,n) + x(m-1,n) - 2x(m,n)\\ \Delta_h^2 x(m,n) &= \Delta_h x(m,n+1) - \Delta_h x(m,n)\\ &= (x(m,n+1) - x(m,n)) - (x(m,n) - x(m,n-1))\\ &= x(m,n+1) + x(m,n-1) - 2x(m,n)\end{aligned}$$

Summing the two second differences, we get the discrete approximation of the Laplacian as

$$\begin{aligned}\nabla^2 x(m,n) &= \Delta_h^2 x(m,n) + \Delta_v^2 x(m,n)\\ &= x(m,n+1) + x(m,n-1) + x(m+1,n) + x(m-1,n) - 4x(m,n)\end{aligned}$$

The filter coefficients $h(m,n)$ are

$$h(m,n) = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{2.2}$$

By adding this mask by its 45° rotated version, we get the filter coefficients $h(m,n)$ for 8-neighborhood

$$h(m,n) = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{2.3}$$

Let the input be the same used for lowpass filtering. With zero-padded and replicated inputs, the outputs of applying the Laplacian mask (Eq. 2.2) are, respectively,

$$y(m,n) = \begin{bmatrix} -3 & 9 & -9 & -1 \\ -5 & -2 & 2 & -14 \\ 0 & 9 & -7 & 16 \\ -10 & 0 & -3 & -8 \end{bmatrix} \quad y(m,n) = \begin{bmatrix} -1 & 8 & -6 & 3 \\ -3 & -2 & 2 & -10 \\ 1 & 9 & -7 & 14 \\ -4 & 1 & -1 & -4 \end{bmatrix}$$

The output has large number of negative values. For proper display of the output, scaling is required. With 256 gray levels,

$$y_s(m,n) = \frac{(y(m,n) - y_{min})}{(y_{max} - y_{min})} 255$$

Figure 2.18a show a 256 × 256 8-bit image. Figure 2.18b shows the image after the application of the Laplacian filter (Eq. (2.2)). The low contrast of the image is

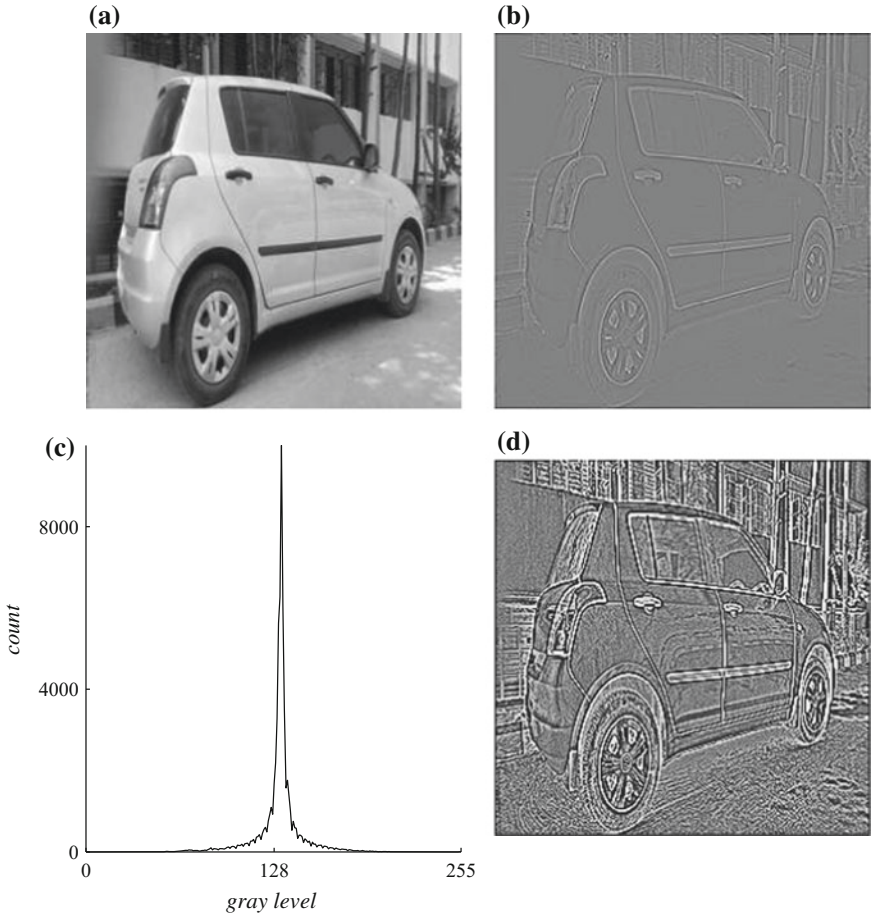


Fig. 2.18 **a** A 256×256 8-bit image; **b** the image after application of the Laplacian filter (Eq. (2.2)); **c** its scaled histogram; **d** the histogram equalized image

due to the concentration of the pixel values in the middle of the scaled histogram (Fig. 2.18c). The histogram equalized image is shown in Fig. 2.18d.

Subtracting the Laplacian from the image sharpens the image. Using the first mask,

$$\begin{aligned} x(m,n) - \nabla^2 x(m,n) &= 5x(m,n) - (x(m,n+1) + x(m,n-1) + x(m+1,n) + x(m-1,n)) \\ &= x(m,n) + 5(x(m,n) \\ &\quad -\frac{1}{5}(x(m,n+1) + x(m,n-1) + x(m,n) + x(m+1,n) + x(m-1,n))) \end{aligned}$$

The third line is a blurred and scaled version of the image $x(m,n)$. The high frequency components are suppressed. When the blurred version is subtracted from the input

image (called unsharp masking), the resulting image is composed of strong high frequency components and weak low frequency components. When this version is multiplied by the factor 5 and added to the image, the high frequency components are boosted (high-emphasis filtering) and the low frequency components remain about the same. The corresponding Laplacian sharpening filter is deduced from the last equation as

$$h(m,n) = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} \tag{2.4}$$

Using this filter, with the same input used for lowpass filtering, the outputs with the input zero-padded and replicated are, respectively,

$$y(m,n) = \begin{bmatrix} 4 & -10 & 12 & 3 \\ 7 & 3 & 0 & 18 \\ 1 & -10 & 9 & -18 \\ 13 & 1 & 5 & 10 \end{bmatrix} \quad y(m,n) = \begin{bmatrix} 2 & -9 & 9 & -1 \\ 5 & 3 & 0 & 14 \\ 0 & -10 & 9 & -16 \\ 7 & 0 & 3 & 6 \end{bmatrix}$$

Figure 2.19a shows the image in Fig. 2.18a after application of the Laplacian filter (Eq. (2.3)). The edges at the diagonal directions are sharper compared with Fig. 2.18b. Figure 2.19b shows the image in Fig. 2.18a after application of the Laplacian sharpening filter (Eq. (2.4)). The edges are sharper compared with Fig. 2.18a.

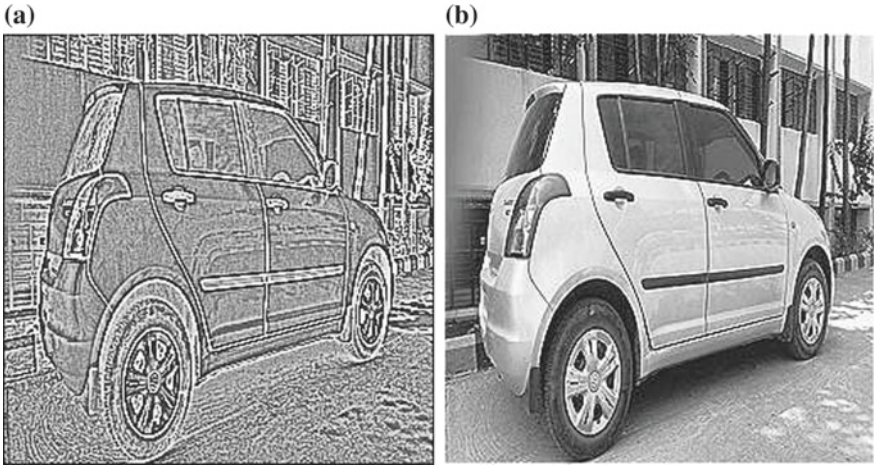


Fig. 2.19 **a** Image in Fig. 2.18a after application of the Laplacian filter (Eq. (2.3)); **b** Image in Fig. 2.18a after application of the Laplacian sharpening filter (Eq. (2.4))

2.4.2 Median Filtering

Some measures of the distribution of the pixel values in an image are the mean, the median, the standard deviation and the histogram. The mean, $\bar{x}$, of a $M \times N$ image $x(m, n)$ is given by

$$\bar{x} = \frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} x(m, n)$$

The median of a list of N numbers $x(n)$

$$\{x(0), x(1), \ldots, x(N-1)\}$$

is defined as the middle number of the sorted list of $x(n)$, if N is odd. If N is even, the median is defined as the mean of the two middle numbers of the sorted list. For 2-D data, all the samples in the window are listed as 1-D data for median computation. The mean and median gives an indication of the center of the data. The spread of the data is given by the variance and the standard deviation. The variance is a measure of the spread of each pixel from the mean of an image. A variance value of zero indicates that all the pixels are the same as the mean. A small variance value indicates that pixel values are distributed close to the mean and close to themselves and vice versa. It is a positive value. The variance σ^2 of a $M \times N$ image $x(m, n)$ is given by

$$\sigma^2 = \frac{1}{(M)(N)} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} (x(m, n) - \bar{x})^2$$

(Sometimes, the divisor $((M-1)(N-1))$ is used in the definition of σ^2.) The variance is the mean of the squared differences between each value and the mean of the data. The standard deviation σ is the square root of the variance. Consider the 4×4 image

23	51	23	32
32	44	44	23
23	23	44	32
44	44	23	23

The mean, variance and standard deviation are 33, 102 and 10.0995, respectively.

Median filtering, which is nonlinear, replaces a pixel by the median of a window of pixels in its neighborhood. It involves sorting the pixels in the window in ascending or descending order and selecting the middle value, if the number of pixels is odd. Otherwise, the average of the two middle values is the median. In this case, if the input is integer-valued then the output can be of the same type by using truncation or rounding. The window sizes used typically are 3×3, 5×5 and 7×7.

Consider the 4×4 image and its boundary replicated version

23	51	23	32
32	44	44	23
23	23	44	32
44	44	23	23

23	23	51	23	32	32
23	23	51	23	32	32
32	32	44	44	23	23
23	23	23	44	32	32
44	44	44	23	23	23
44	44	44	23	23	23

The image after median filtering with a 3×3 window is

32	32	32	32
23	32	32	32
32	44	32	23
44	44	23	23

Median filtering is effective in reducing the spot (or impulse or salt-and-pepper) noise, characterized by the random occurrence of black and white pixels. The probability distribution of this noise is given by

$$p(x) = \begin{cases} p_1, & \text{for } x = 1 \\ p_0, & \text{for } x = 0 \\ 0, & \text{otherwise} \end{cases}$$

Pixel value 1 indicates that it will be white and zero indicates that the pixel will be black. If the probabilities of the occurrence of the black and white pixels are about equal, then the effect of this noise is to look like flecks of salt and pepper spread all over the image. Hence, it is called as salt-and-pepper noise.

A pixel with a value that is much larger than those of its neighbors is probably a noise pixel. The image is enhanced if such pixels are replaced by the median in their neighborhood. On the other hand, if the pixel value is valid then median filtering will degrade the image quality. In any image processing, the most suitable operators with respect to size and response, and algorithms should be used. This requires some trial and error. While median filtering is commonly used, a pixel can be replaced by any other pixel in the sorted list of its neighborhood, such as the maximum and minimum values. Figure 2.20a, b show a 256×256 8-bit image and the image with spot noise, respectively. Figure 2.20c shows the median filtered image with a 3×3 window. The noise has been removed. Figure 2.20d shows the lowpass filtered image with a 3×3 window. Lowpass filtering is not effective to reduce the spot noise. Figure 2.20e shows the image with each pixel in the complement of input image replaced by the maximum value in its 5×5 neighborhood. It highlights the brightest parts of the image. The image has become brighter. Figure 2.20f shows the image with each pixel replaced by the minimum value in its 5×5 neighborhood. It highlights the darkest parts of the image.

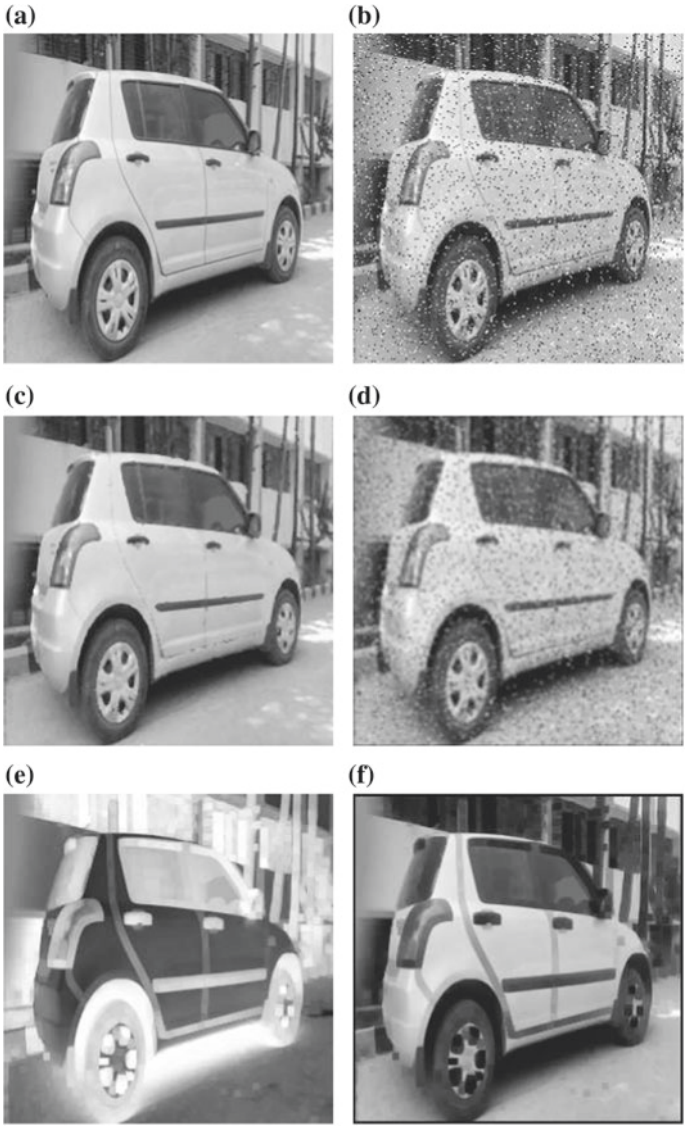


Fig. 2.20 **a** A 256 × 256 8-bit image and **b** the image with spot noise; **c** median filtered image with a 3 × 3 window; **d** lowpass filtered image with a 3 × 3 window; **e** image with each pixel in the complement of the input image replaced by the maximum value in its 5 × 5 neighborhood; **f** image with each pixel replaced by the minimum value in its 5 × 5 neighborhood

2.5 Summary

- Image enhancement involves modifying the pixel values to improve the quality of the image with some respect for human or machine vision.
- The simplest type is that in which each output pixel is a function of the input pixel only. Typical operations include complementing and gamma correction. Pointwise arithmetic and logical operations are carried out with corresponding pixels in two or more images.
- Histogram is the count of the number of occurrences of each gray level in the image. In addition to enhancement, histograms are useful for other operations such as segmentation.
- In histogram stretching, a part of the range of gray levels is stretched to enhance the image.
- In histogram equalization, the gray levels are redistributed uniformly over the gray level range to enhance the image.
- In histogram specification, the gray levels are redistributed, according to the specified histogram, over the gray level range to restore the image, with a knowledge of its original histogram.
- Thresholding is choosing a gray level of some significance and using it to do processing such as segmentation, compression and denoising.
- In neighborhood operations, the output value of a pixel is a linear or nonlinear function of a set of pixels in its neighborhood.
- In linear filtering, the spectrum of the image is modified in a desired way. This includes operations such as lowpass and highpass filtering.
- Typical lowpass filters are averaging and Gaussian. Laplacian filter is of highpass type.
- Filtering is implemented by the convolution operation in the spatial domain. Although an image is a 2-D function, 1-D convolution is also used for filtering with separable filters.
- In nonlinear filtering, the output value of a pixel is a nonlinear function of a set of pixels in its neighborhood.
- Typical example of nonlinear filtering is median filtering, in which the output pixel value corresponding to a pixel is the median of a set of pixels in its neighborhood.

Exercises

2.1 Find the complement of the 4×4 8-bit gray level image and verify that the image can be restored by complementing the complemented image.

(i)

$$\begin{bmatrix} 112 & 148 & 72 & 153 \\ 120 & 125 & 30 & 99 \\ 95 & 120 & 89 & 33 \\ 170 & 99 & 109 & 40 \end{bmatrix}$$

(ii)

$$\begin{bmatrix} 164 & 127 & 117 & 59 \\ 154 & 122 & 104 & 83 \\ 129 & 136 & 100 & 60 \\ 117 & 128 & 80 & 48 \end{bmatrix}$$

(iii)

$$\begin{bmatrix} 46 & 48 & 46 & 45 \\ 42 & 49 & 46 & 45 \\ 64 & 73 & 60 & 43 \\ 94 & 69 & 63 & 37 \end{bmatrix}$$

2.2 Find the complement of the 4×4 binary image and verify that the image can be restored by complementing the complemented image.

(i)

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

(ii)

$$\begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(iii)

$$\begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix}$$

2.3 For the list of gray levels, apply gamma correction and find the corresponding new gray levels. Apply the inverse transformation to the new gray levels and verify that the given gray levels are obtained.

$$\{0, 25, 50, 100, 150, 200, 250, 255\}$$

(i) $\gamma = 0.8$.
(ii) $\gamma = 1.1$.
(iii) $\gamma = 1.8$.

2.4 Given a 4×4 4-bit image, find the histogram equalized version of it.

*(i)

$$\begin{bmatrix} 4 & 4 & 3 & 3 \\ 4 & 4 & 4 & 3 \\ 5 & 4 & 4 & 4 \\ 4 & 4 & 4 & 4 \end{bmatrix}$$

(ii)

$$\begin{bmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 3 \\ 1 & 0 & 0 & 2 \\ 1 & 0 & 0 & 2 \end{bmatrix}$$

(iii)

$$\begin{bmatrix} 3 & 5 & 5 & 3 \\ 4 & 4 & 4 & 3 \\ 4 & 2 & 3 & 4 \\ 4 & 2 & 2 & 3 \end{bmatrix}$$

2.5 Given 4×4 4-bit reference and input images, use histogram matching to restore the input image.

*(i) The reference and input images, respectively, are

$$\begin{bmatrix} 4 & 4 & 3 & 3 \\ 4 & 4 & 4 & 3 \\ 5 & 4 & 4 & 4 \\ 4 & 4 & 4 & 4 \end{bmatrix} \quad \begin{bmatrix} 15 & 15 & 0 & 0 \\ 15 & 15 & 15 & 0 \\ 15 & 15 & 15 & 15 \\ 15 & 15 & 15 & 15 \end{bmatrix}$$

(ii) The reference and input images, respectively, are

$$\begin{bmatrix} 3 & 3 & 3 & 3 \\ 3 & 3 & 3 & 3 \\ 3 & 2 & 2 & 3 \\ 2 & 2 & 2 & 2 \end{bmatrix} \quad \begin{bmatrix} 15 & 15 & 15 & 15 \\ 15 & 15 & 15 & 15 \\ 15 & 0 & 0 & 15 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

(iii) The reference and input images, respectively, are

$$\begin{bmatrix} 3 & 5 & 5 & 3 \\ 4 & 4 & 4 & 3 \\ 4 & 2 & 3 & 4 \\ 4 & 2 & 2 & 3 \end{bmatrix} \quad \begin{bmatrix} 0 & 15 & 15 & 0 \\ 15 & 15 & 15 & 0 \\ 15 & 0 & 0 & 15 \\ 15 & 0 & 0 & 0 \end{bmatrix}$$

2.6 Given a 8×8 8-bit image, find the binary, hard and soft thresholded versions with the threshold $T = 160$.

$$\begin{bmatrix} 255 & 255 & 255 & 117 & 50 & 39 & 50 & 56 \\ 255 & 255 & 255 & 194 & 45 & 26 & 48 & 54 \\ 255 & 255 & 255 & 241 & 61 & 25 & 53 & 57 \\ 255 & 255 & 255 & 255 & 104 & 32 & 64 & 64 \\ 255 & 255 & 255 & 255 & 154 & 37 & 59 & 61 \\ 255 & 255 & 255 & 255 & 199 & 54 & 55 & 61 \\ 255 & 255 & 255 & 255 & 230 & 71 & 59 & 64 \\ 255 & 255 & 255 & 255 & 250 & 95 & 60 & 68 \end{bmatrix}$$

2.7 Given a 4×4 image, find the 4×4 lowpass filtered output using the 3×3 averaging filter with the borders zero-padded.

$$x(m,n) = \begin{bmatrix} 70 & 62 & 51 & 45 \\ 71 & 62 & 57 & 55 \\ 73 & 65 & 56 & 60 \\ 68 & 69 & 63 & 66 \end{bmatrix}$$

2.8 Given a 4×4 image, find the 4×4 lowpass filtered output using the 3×3 averaging filter with the borders replicated.

$$x(m,n) = \begin{bmatrix} 41 & 43 & 45 & 43 \\ 40 & 41 & 42 & 41 \\ 42 & 38 & 39 & 42 \\ 39 & 33 & 37 & 36 \end{bmatrix}$$

2.9 Given a 4×4 image, find the 4×4 lowpass filtered output using the 3×3 averaging filter with the borders periodically extended.

$$x(m,n) = \begin{bmatrix} 45 & 78 & 87 & 51 \\ 59 & 56 & 62 & 49 \\ 59 & 39 & 44 & 57 \\ 56 & 36 & 35 & 51 \end{bmatrix}$$

2.10 Given a 4×4 image, find the 4×4 lowpass filtered output using the 3×3 Gaussian filter ($\sigma = 0.5$) with the borders symmetrically extended.

$$x(m,n) = \begin{bmatrix} 202 & 195 & 192 & 191 \\ 216 & 211 & 200 & 209 \\ 224 & 212 & 215 & 227 \\ 224 & 205 & 227 & 230 \end{bmatrix}$$

2.11 Given a 4×4 image, find the 4×4 lowpass filtered output using the 3×3 Gaussian filter ($\sigma = 0.5$) with the borders periodically extended.

$$x(m,n) = \begin{bmatrix} 202 & 195 & 192 & 191 \\ 216 & 211 & 200 & 209 \\ 224 & 212 & 215 & 227 \\ 224 & 205 & 227 & 230 \end{bmatrix}$$

2.12 Given a 4×4 image, find the 4×4 lowpass filtered output using the 3×3 Gaussian filter ($\sigma = 0.5$) with the borders zero-padded.

$$x(m,n) = \begin{bmatrix} 95 & 82 & 54 & 33 \\ 84 & 78 & 56 & 64 \\ 73 & 71 & 53 & 60 \\ 73 & 73 & 54 & 36 \end{bmatrix}$$

2.13 Given a 4×4 image, find the 4×4 highpass filtered output using the 3×3 Laplacian filter

$$h(m,n) = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

with the borders zero-padded.

$$x(m,n) = \begin{bmatrix} 45 & 52 & 56 & 52 \\ 49 & 60 & 55 & 55 \\ 47 & 55 & 53 & 46 \\ 45 & 48 & 51 & 40 \end{bmatrix}$$

2.14 Given a 4×4 image, find the 4×4 highpass filtered output using the 3×3 Laplacian filter with the borders symmetrically extended.

$$x(m,n) = \begin{bmatrix} 64 & 62 & 62 & 68 \\ 68 & 66 & 58 & 64 \\ 75 & 70 & 60 & 58 \\ 72 & 69 & 59 & 60 \end{bmatrix}$$

2.15 Given a 4×4 image, find the 4×4 highpass filtered output using the 3×3 Laplacian filter with the borders replicated.

$$x(m,n) = \begin{bmatrix} 39 & 40 & 35 & 33 \\ 31 & 40 & 39 & 37 \\ 34 & 38 & 41 & 43 \\ 37 & 39 & 42 & 43 \end{bmatrix}$$

***2.16** Given a 4×4 image, find the 4×4 enhanced output using the 3×3 Laplacian sharpening filter

$$h(m,n) = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

with the borders replicated.

$$x(m,n) = \begin{bmatrix} 190 & 206 & 228 & 238 \\ 180 & 205 & 227 & 219 \\ 182 & 203 & 211 & 159 \\ 184 & 212 & 206 & 177 \end{bmatrix}$$

2.17 Given a 4×4 image, find the 4×4 enhanced output using the 3×3 Laplacian sharpening filter with the borders periodically extended.

$$x(m,n) = \begin{bmatrix} 138 & 163 & 162 & 177 \\ 148 & 157 & 167 & 175 \\ 153 & 165 & 160 & 178 \\ 157 & 162 & 164 & 188 \end{bmatrix}$$

2.18 Given a 4×4 image, find the 4×4 enhanced output using the 3×3 Laplacian sharpening filter with the borders zero-padded.

$$x(m,n) = \begin{bmatrix} 201 & 195 & 191 & 169 \\ 210 & 201 & 181 & 157 \\ 213 & 207 & 190 & 166 \\ 204 & 204 & 197 & 159 \end{bmatrix}$$

2.19 Given a 4×4 image, find the 4×4 median filtered output using the 3×3 window with the borders zero-padded.

(i)

$$x(m,n) = \begin{bmatrix} 201 & 195 & 191 & 169 \\ 210 & 201 & 181 & 157 \\ 213 & 207 & 190 & 166 \\ 204 & 204 & 197 & 159 \end{bmatrix}$$

(ii)

$$x(m,n) = \begin{bmatrix} 138 & 163 & 162 & 177 \\ 148 & 157 & 167 & 175 \\ 153 & 165 & 160 & 178 \\ 157 & 162 & 164 & 188 \end{bmatrix}$$

(iii)

$$x(m,n) = \begin{bmatrix} 190 & 206 & 228 & 238 \\ 180 & 205 & 227 & 219 \\ 182 & 203 & 211 & 159 \\ 184 & 212 & 206 & 177 \end{bmatrix}$$

Chapter 3
Fourier Analysis

Abstract Transforms provide an alternate representation of images, which usually facilitates easier interpretation of operations and fast processing. The most important of all the transforms, the Fourier transform, decomposes an image in terms of sinusoidal surfaces. This transform is of fundamental importance to image processing, as is the case in almost all areas of science and engineering. As in the case of the convolution operation, both the 1-D and 2-D versions are described. Although the image is a 2-D signal, some of the important operations are decomposable and can be carried out in one dimension with reduced execution time. Another advantage is that understanding of the 1-D version is simpler. Definition, properties, and examples of the transforms are presented.

Transform means change in form and, in general, transforms make the processing of data easier. For example, the more difficult multiplication operation becomes addition when the two numbers, to be multiplied, are represented in their logarithmic form. For the most part, image processing is easier and efficient using a transformed version of the image. A transform is of practical interest only if fast algorithms are available for its computation. Fortunately, in most cases, the row–column method is applicable, which requires the repeated use of 1-D transform algorithms. Fourier analysis and convolution are the two fundamental concepts those are indispensable in signal and system analysis. It is mandatory that these two concepts are well understood. With a good understanding of the concepts, these tools can be easily and efficiently used in applications. Fortunately, the concepts are simple. Convolution, in principle, is nothing more than computing the balance in a bank account, as presented in an earlier chapter.

In this chapter and in the Appendix, we present the DFT and its continuous version, its properties and the fast algorithms to compute them. As an image is a 2-D signal, the required transforms and processing are, in general, straightforward extensions of those of 1-D signals. The 1-D versions of the transforms are also directly used in image processing operations often and they are easier to understand. For these reasons, the 1-D versions of the transforms are presented first, followed by their 2-D versions.

D. Sundararajan, *Digital Image Processing*, DOI 10.1007/978-981-10-6113-4_3

Fourier analysis, in principle, is not more complex than finding the amount of a bag of coins. A bag contains a large number of coins of various denominations. There are two possible ways to find the amount of coins in the bag. One way is to arbitrarily pick up a coin, find its value, and add it to a partial sum. Then, pick up another coin and do the same. This process ends after the values of all the coins are added to find the amount. Another way is to sort the coins into various denominations, count the number of coins in each group, multiply by their respective values, and sum them up. The second procedure is, obviously, more efficient when the number of coins is relatively large. In addition, we know the relative number of coins in various denominations.

In Fourier analysis, a time-domain waveform is decomposed into its sinusoidal components of various frequencies. One advantage of the decomposition is that it gives the strength of the various components, which is called the spectrum of the signal. The spectrum is the starting point in most of the analysis. Another advantage is that, as in the case of finding the amount of coins, it is more efficient to find the system output using the sinusoidal components of the input signal. That is because the convolution operation becomes the much simpler multiplication operation, once a waveform is decomposed. The sinusoidal (and its mathematically equivalent complex exponential) waveform is the only one that retains its shape from the input to the output of a linear system. Then, using the linearity property of the linear systems and decomposing an arbitrary waveform using Fourier analysis, the system output is computed using multiplication and addition operations only. Further, the availability of fast algorithms for computing the transform makes the transform analysis essential.

Another point is that, although the signal is mostly real-valued, we would be using complex numbers in the analysis. There is nothing complex about complex numbers. They are ordered pairs of numbers (2-element vectors). The use of complex numbers is required because a sinusoidal waveform, at a given frequency, is characterized by its amplitude and phase and storing these two values in a vector is the most efficient way to represent and manipulate sinusoids (which are the basis waveforms in Fourier analysis). In summary, a good amount of practice (both paper-and-pencil and computer programming) of Fourier analysis and convolution will make anyone proficient in signal and system analysis.

3.1 The 1-D Discrete Fourier Transform

The DFT is the practically most often used version of the Fourier analysis. This is due to the fact that the input and output of this transform are discrete and finite and, therefore, it is inherently suitable for implementation on the digital computer. Further, fast algorithms are available for its computation. In the DFT, a periodically extended finite discrete signal is decomposed in terms of a finite number of discrete sinusoidal waveforms.

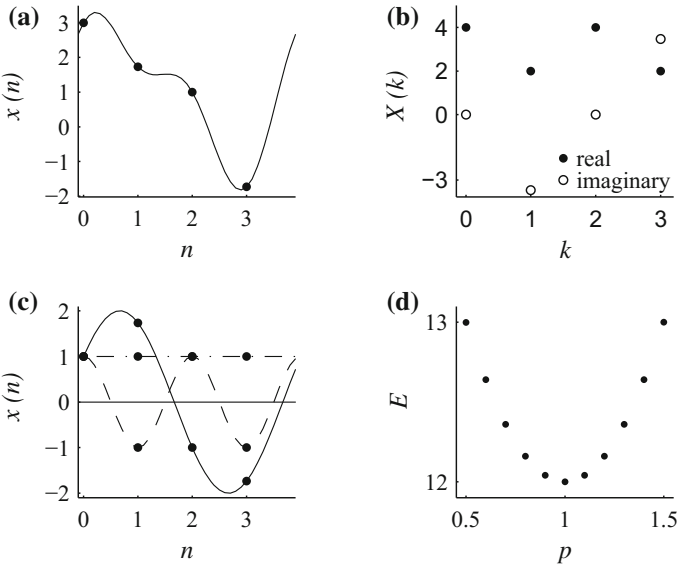


Fig. 3.1 **a** A periodic waveform, $x(n) = 1 + 2\cos(\frac{2\pi}{4}n - \frac{\pi}{3}) + \cos(2\frac{2\pi}{4}n)$, with period 4 samples and **b** its frequency-domain representation; **c** the frequency components of the waveform in (**a**); **d** the square error in approximating the waveform in (**a**) using only the DC component with different amplitudes

Consider the discrete periodic waveform,

$$x(n) = 1 + 2\cos\left(\frac{2\pi}{4}n - \frac{\pi}{3}\right) + \cos\left(2\frac{2\pi}{4}n\right)$$

with period 4 samples, shown in Fig. 3.1a. The independent variable n represents time and the dependent variable $x(n)$ is the amplitude. The 4 samples over one period are obtained from the equation for $n = 0, 1, 2, 3$

$$\{x(0) = 3, x(1) = \sqrt{3}, x(2) = 1, x(3) = -\sqrt{3}\}.$$

For example, letting $n = 1$, we get

$$x(1) = 1 + 2\cos\left(\frac{2\pi}{4}1 - \frac{\pi}{3}\right) + \cos\left(2\frac{2\pi}{4}1\right) = 1 + 2\frac{\sqrt{3}}{2} - 1 = \sqrt{3}.$$

Figure 3.1b shows the frequency-domain representation of the waveform in (a)

$$\{X(0) = 4, X(1) = 2 - j2\sqrt{3}, X(2) = 4, X(3) = 2 + j2\sqrt{3}\}.$$

It shows the complex amplitudes, multiplied by 4, of its constituent complex exponentials. To find the real sinusoids, shown in Fig. 3.1c, those constitute the signal, we add up the complex exponentials.

$$\begin{aligned} x(n) &= \frac{1}{4}\left(4e^{j0\frac{2\pi}{4}n} + (2-j2\sqrt{3})e^{j\frac{2\pi}{4}n} + 4e^{j2\frac{2\pi}{4}n} + (2+j2\sqrt{3})e^{j3\frac{2\pi}{4}n}\right) \\ &= \frac{1}{4}\left(4e^{j0\frac{2\pi}{4}n} + 4e^{j(\frac{2\pi}{4}n-\frac{\pi}{3})} + 4e^{j2\frac{2\pi}{4}n} + 4e^{-j(\frac{2\pi}{4}n-\frac{\pi}{3})}\right) \\ &= 1 + 2\cos\left(\frac{2\pi}{4}n - \frac{\pi}{3}\right) + \cos\left(2\frac{2\pi}{4}n\right) \end{aligned}$$

This example demonstrates the fact that Fourier analysis represents a signal as a linear combination of sinusoids or, equivalently, complex exponentials with pure imaginary exponents.

The Fourier reconstruction of a waveform is with respect to the least-squares error criterion. That is, the sum of the squared magnitude of the error between the given waveform and the corresponding Fourier reconstructed waveform is guaranteed to be the minimum if part of the constituent sinusoids of a waveform is used in the reconstruction and will be zero if all the constituent sinusoids are used. The reason this criterion, based on signal energy or power, is used rather than a minimum uniform deviation criterion is that: (i) it is acceptable for most applications and (ii) it leads to closed-form formulas for the analytical determination of the Fourier coefficients. Let $x_a(n)$ be an approximation to a given waveform $x(n)$ of period N, using fewer sinusoids than that is required. The error between $x(n)$ and $x_a(n)$ is defined as

$$E = \sum_{n=0}^{N-1} |x(n) - x_a(n)|^2$$

For a given number of sinusoids, there is no better approximation for the signal than that provided by the Fourier approximation when the least-squares error criterion is applied. Assume that, we are constrained to use only the DC component to approximate the waveform in Fig. 3.1a. Let the optimal value of the DC component be p. To minimize the square error,

$$(3-p)^2 + (\sqrt{3}-p)^2 + (1-p)^2 + (-\sqrt{3}-p)^2$$

must be minimum. Differentiating this expression with respect to p and equating it to zero, we get

$$2(3-p)(-1) + 2(\sqrt{3}-p)(-1) + 2(1-p)(-1) + 2(-\sqrt{3}-p)(-1) = 0.$$

Solving this equation, we get $p = 1$ as given by the Fourier analysis. The square error, for various values of p, is shown in Fig. 3.1d.

For two complex exponentials $e^{j\frac{2\pi}{N}ln}$ and $e^{j\frac{2\pi}{N}kn}$ over a period of N samples, the orthogonality property is defined as

$$\sum_{n=0}^{N-1} e^{j\frac{2\pi}{N}(l-k)n} = \begin{cases} N \text{ for } l = k \\ 0 \text{ for } l \neq k \end{cases}$$

where $l, k = 0, 1, \ldots, N-1$. If $l = k$, the summation is equal to N as $e^{j\frac{2\pi}{N}(l-k)n} = e^0 = 1$. Otherwise, by using the closed-form expression for the sum of a geometric progression, we get

$$\sum_{n=0}^{N-1} e^{j\frac{2\pi}{N}(l-k)n} = \frac{1 - e^{j2\pi(l-k)}}{1 - e^{j\frac{2\pi(l-k)}{N}}} = 0, \quad \text{for } l \neq k$$

It is also obvious from the fact that the sum of the samples of the complex exponential over an integral number of cycles is zero. That is, in order to find the coefficient, with a scale factor N, of a complex exponential, we multiply the samples of a signal with the corresponding samples of the complex conjugate of the complex exponential. Using each complex exponential in turn, we get the frequency coefficients of all the components of a signal as

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk}, \quad k = 0, 1, \ldots, N-1 \tag{3.1}$$

where $W_N = e^{-j\frac{2\pi}{N}}$. This is the DFT equation analyzing a waveform with harmonically related discrete complex sinusoids. $X(k)$ is the coefficient, scaled by N, of the complex sinusoid $e^{j\frac{2\pi}{N}kn}$ with a specific frequency index k (frequency $\frac{2\pi}{N}k$ radians per sample). DFT computation is based on assumed periodicity. That is, the N input values are considered as one period of a periodic waveform with period N. The summation of the sample values of the N complex sinusoids multiplied by their respective frequency coefficients $X(k)$ is the IDFT operation. The N-point IDFT of the frequency coefficients $X(k)$ is defined as

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) W_N^{-nk}, \quad n = 0, 1, \ldots, N-1 \tag{3.2}$$

The sum of the sample values is divided by N in Eq. (3.2) as the coefficients $X(k)$ have been scaled by the factor N in the DFT computation.

The DFT and IDFT definitions can be expressed in matrix form. Expanding the DFT definition with $N = 4$, we get

$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{bmatrix} = \begin{bmatrix} e^{-j\frac{2\pi}{4}(0)(0)} & e^{-j\frac{2\pi}{4}(0)(1)} & e^{-j\frac{2\pi}{4}(0)(2)} & e^{-j\frac{2\pi}{4}(0)(3)} \\ e^{-j\frac{2\pi}{4}(1)(0)} & e^{-j\frac{2\pi}{4}(1)(1)} & e^{-j\frac{2\pi}{4}(1)(2)} & e^{-j\frac{2\pi}{4}(1)(3)} \\ e^{-j\frac{2\pi}{4}(2)(0)} & e^{-j\frac{2\pi}{4}(2)(1)} & e^{-j\frac{2\pi}{4}(2)(2)} & e^{-j\frac{2\pi}{4}(2)(3)} \\ e^{-j\frac{2\pi}{4}(3)(0)} & e^{-j\frac{2\pi}{4}(3)(1)} & e^{-j\frac{2\pi}{4}(3)(2)} & e^{-j\frac{2\pi}{4}(3)(3)} \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{bmatrix}$$
$$= \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{bmatrix}$$

Using vector and matrix quantities, the DFT definition is given by

$$\boldsymbol{X} = \boldsymbol{W}\boldsymbol{x}$$

where $\boldsymbol{x}$ is the input vector, $\boldsymbol{X}$ is the coefficient vector, and $\boldsymbol{W}$ is the transform matrix, defined as

$$\boldsymbol{W} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix}$$

Expanding the IDFT definition with $N = 4$, we get

$$\begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{bmatrix} = \frac{1}{4} \begin{bmatrix} e^{j\frac{2\pi}{4}(0)(0)} & e^{j\frac{2\pi}{4}(0)(1)} & e^{j\frac{2\pi}{4}(0)(2)} & e^{j\frac{2\pi}{4}(0)(3)} \\ e^{j\frac{2\pi}{4}(1)(0)} & e^{j\frac{2\pi}{4}(1)(1)} & e^{j\frac{2\pi}{4}(1)(2)} & e^{j\frac{2\pi}{4}(1)(3)} \\ e^{j\frac{2\pi}{4}(2)(0)} & e^{j\frac{2\pi}{4}(2)(1)} & e^{j\frac{2\pi}{4}(2)(2)} & e^{j\frac{2\pi}{4}(2)(3)} \\ e^{j\frac{2\pi}{4}(3)(0)} & e^{j\frac{2\pi}{4}(3)(1)} & e^{j\frac{2\pi}{4}(3)(2)} & e^{j\frac{2\pi}{4}(3)(3)} \end{bmatrix} \begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{bmatrix}$$
$$= \frac{1}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & j & -1 & -j \\ 1 & -1 & 1 & -1 \\ 1 & -j & -1 & j \end{bmatrix} \begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{bmatrix}$$

Concisely,

$$\boldsymbol{x} = \frac{1}{4}\boldsymbol{W}^{-1}\boldsymbol{X} = \frac{1}{4}(\boldsymbol{W}^{*})\boldsymbol{X}$$

The inverse and forward transform matrices are orthogonal. That is,

$$\frac{1}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & j & -1 & -j \\ 1 & -1 & 1 & -1 \\ 1 & -j & -1 & j \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The DFT is defined for sequences of any length. However, it is usually assumed that the length N of a sequence

$$\{x(0), x(1), x(2), \ldots, x(N-1)\}$$

is a power of 2 in most applications. The reason is that practically efficient algorithms for the computation of the DFT are available only for these lengths. When necessary to meet this constraint, the signal can be appended by sufficient number of zero-valued samples.

Some examples of 4-point DFT computation are given below. The DFT of

$$\{x(0) = 3, x(1) = \sqrt{3}, x(2) = 1, x(3) = -\sqrt{3}\}$$

is computed as

$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \begin{bmatrix} 3 \\ \sqrt{3} \\ 1 \\ -\sqrt{3} \end{bmatrix} = \begin{bmatrix} 4 \\ 2 - j2\sqrt{3} \\ 4 \\ 2 + j2\sqrt{3} \end{bmatrix}$$

The DFT spectrum, as shown in Fig. 3.1b, is $\{X(0) = 4,\ X(1) = 2 - j2\sqrt{3},\ X(2) = 4,\ X(3) = 2 + j2\sqrt{3},\}$. Using the IDFT, we get back the input $x(n)$.

$$\begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{bmatrix} = \frac{1}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & j & -1 & -j \\ 1 & -1 & 1 & -1 \\ 1 & -j & -1 & j \end{bmatrix} \begin{bmatrix} 4 \\ 2 - j2\sqrt{3} \\ 4 \\ 2 + j2\sqrt{3} \end{bmatrix} = \begin{bmatrix} 3 \\ \sqrt{3} \\ 1 \\ -\sqrt{3} \end{bmatrix}$$

The DFT of

$$\{x(0) = 4, x(1) = 0, x(2) = 0, x(3) = 0\}$$

is computed as

$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \begin{bmatrix} 4 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 4 \\ 4 \\ 4 \\ 4 \end{bmatrix}$$

The DFT spectrum is $\{X(0) = 4, X(1) = 4, X(2) = 4, X(3) = 4\}$. This is an impulse $4\delta(n)$ and its spectrum is uniform. All the frequency components exist with equal amplitude and zero phase. Using the IDFT, we get back the input $x(n)$.

$$\begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{bmatrix} = \frac{1}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & j & -1 & -j \\ 1 & -1 & 1 & -1 \\ 1 & -j & -1 & j \end{bmatrix} \begin{bmatrix} 4 \\ 4 \\ 4 \\ 4 \end{bmatrix} = \begin{bmatrix} 4 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

The DFT of

$$\{x(0) = 2, x(1) = 2, x(2) = 2, x(3) = 2\}$$

is computed as

$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \begin{bmatrix} 2 \\ 2 \\ 2 \\ 2 \end{bmatrix} = \begin{bmatrix} 8 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

The DFT spectrum is $\{X(0) = 8, X(1) = 0, X(2) = 0, X(3) = 0\}$. This is the DC signal and its spectrum is nonzero only at $k = 0$. Using the IDFT, we get back the input $x(n)$.

$$\begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{bmatrix} = \frac{1}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & j & -1 & -j \\ 1 & -1 & 1 & -1 \\ 1 & -j & -1 & j \end{bmatrix} \begin{bmatrix} 8 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \\ 2 \\ 2 \end{bmatrix}$$

Figure 3.2 shows some standard signals and their DFT spectra. Figure 3.2a, b shows the unit-impulse signal $\delta(n)$ and its DFT spectrum. One of the reasons for the importance of the impulse signal is that its transform is a constant. The DC signal $x(n)$ and its DFT spectrum, shown in Fig. 3.2c, d, are almost the reversal of those of the impulse.

Figure 3.2e, f shows the sinusoid $2\cos\left(2\frac{2\pi}{8}n - \frac{\pi}{3}\right)$ and its DFT spectrum. The sinusoid

$$2\cos\left(2\frac{2\pi}{8}n - \frac{\pi}{3}\right) = \cos\left(2\frac{2\pi}{8}n\right) + \sqrt{3}\sin\left(2\frac{2\pi}{8}n\right)$$

Therefore, the real parts of the spectrum are $\{4, 4\}$ at $k = 2, 6$ since the frequency index is 2. The imaginary parts are $\sqrt{3}\{-4, 4\}$ at $k = 2, 6$.

The most well-known and probably the first example of Fourier reconstruction presented in teaching Fourier analysis is the square wave. Figure 3.3a shows one period of the square waveform with period 256 samples. Figure 3.3a also shows the reconstruction by the DC component alone. Figure 3.3b shows its magnitude spectrum $\log_e(1 + |X(k|)$ (log scale). Whenever the dynamic range of a function becomes large, it is usually presented in a logarithmic scale for a better display. As the logarithm is undefined for zero, the constant 1 is added before taking the logarithm. As the waveform has discontinuities, the rate of decay of the coefficients is slow (of the order of $1/k$). Discontinuities in the waveform is an extreme test for Fourier reconstruction as the sinusoids are very smooth functions and infinitely differentiable,

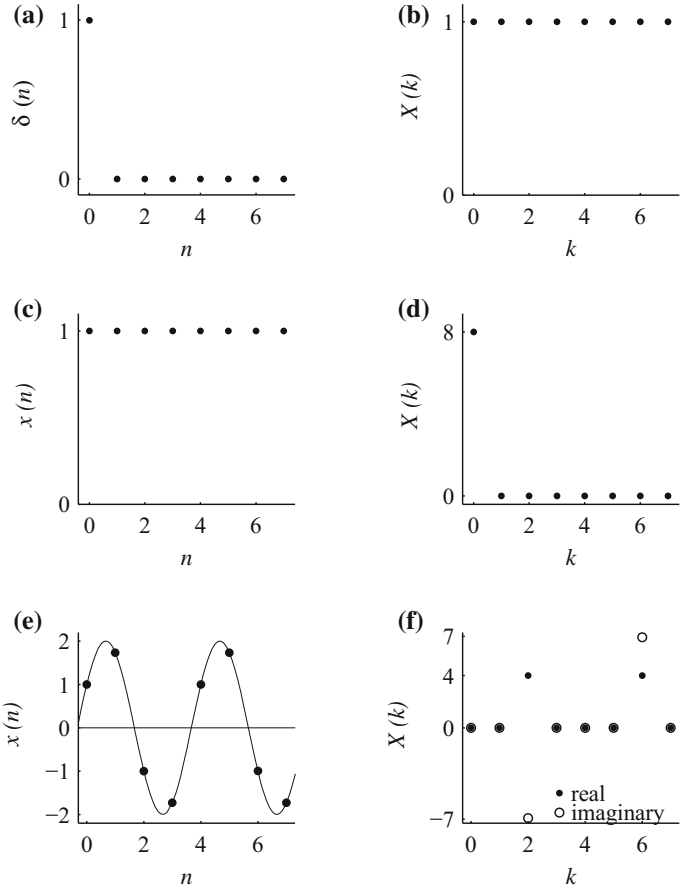


Fig. 3.2 **a** The unit-impulse signal $\delta(n)$ with 8 samples and **b** its DFT spectrum; **c** the DC signal $x(n)$ and **d** its DFT spectrum; **e** the sinusoid $2\cos\left(2\frac{2\pi}{8}n - \frac{\pi}{3}\right)$ and **f** its DFT spectrum

while a discontinuity has no derivative. In practice, such sharp discontinuities are less likely and the rate of decay is likely to be much faster.

Figure 3.3c–f shows the Fourier reconstruction of the square wave with 2, 4, 8, and 16 frequency components, respectively. As the number of components is increased, the reconstructed waveform becomes more closer to the original. The original waveform is reconstructed by constructive and destructive interference of the frequency components. For example, the peak value in (c) at sample 128 is less than 1 of that of the original waveform and it is higher at sample 64 than required. Adding more components in the reconstruction process, constructive interference occurs at sample 128 increasing the value and destructive interference occurs at sample 64 decreasing the value. This process continues until all the components are used.

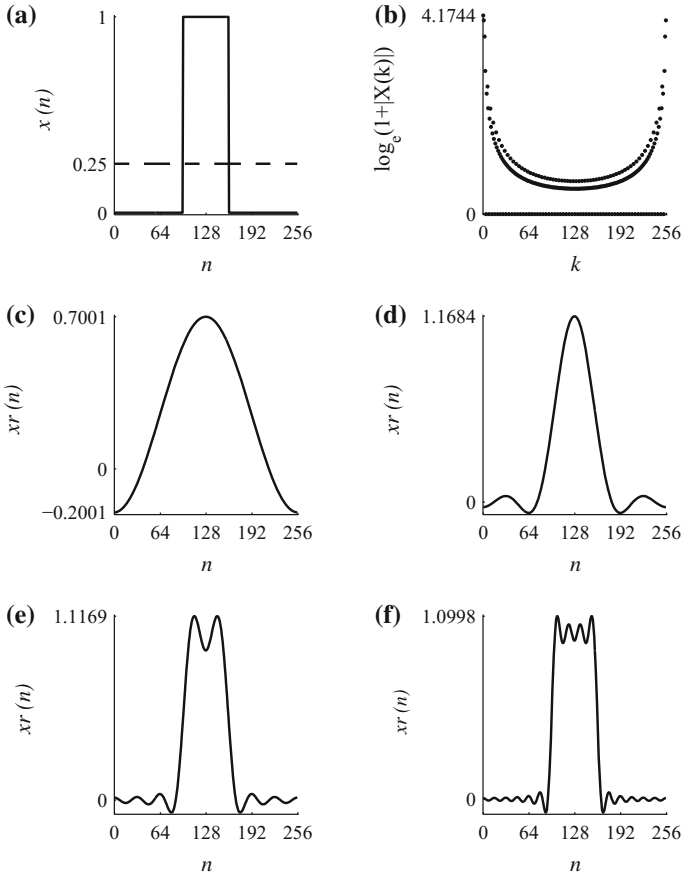


Fig. 3.3 **a** One period of a periodic square waveform and its representation with the DC component alone; **b** its magnitude spectrum $\log_e(1+|X(k|)$ (log scale); the Fourier reconstruction of the square waveform with (**c–f**) 2, 4, 8, and 16 frequency components, respectively

Fourier analysis, in theory, requires an infinite number of components to reconstruct an arbitrary waveform accurately. But, in practice, the approximation by a relatively small number of components is found to be adequate. This feature, along with fast algorithms for its computation, makes the Fourier analysis essential in practical applications. This figure also brings out the basic fact that Fourier analysis is generating a desired interference pattern using sinusoidal waveforms of various frequencies. Practice with reconstruction graphs, such as this figure, is recommended for a good understanding of Fourier analysis.

Parseval's Theorem

This theorem expresses the power of a signal in terms of its DFT spectrum. The orthogonal transforms have the energy preservation property. Let $x(n) \leftrightarrow X(k)$ with sequence length N. The double-headed arrow indicates that $X(k)$ is the representation of $x(n)$ in the transform domain. The sum of the squared magnitude of the samples of a complex exponential with amplitude one, over the period N, is N. Remember that these samples occur on the unit circle. The DFT decomposes a signal in terms of complex exponentials with coefficients $X(k)/N$. Therefore, the power of a complex exponential is $\frac{|X(k)|^2}{N^2}N = \frac{|X(k)|^2}{N}$. The power of a signal is the sum of the powers of its constituent complex exponentials and is given as

$$\sum_{n=0}^{N-1} |x(n)|^2 = \frac{1}{N} \sum_{k=0}^{N-1} |X(k)|^2$$

Example 3.1 Verify the Parseval's theorem for the DFT pair

$$\{4, 1, 2, 4\} \leftrightarrow \{11, 2 + j3, 1, 2 - j3\}$$

Solution
The sum of the squared magnitude of the data sequence is 37 and that of the DFT coefficients divided by 4 is also 37. ∎

3.2 The 2-D Discrete Fourier Transform

In taking a transform, an image is expressed as a sum of the scaled basis signals of the transform. For 1-D signals, the basis signals for Fourier analysis are the sinusoidal waveforms. That is, an arbitrary curve is expressed as a linear combination of a set of sinusoids. In the 2-D case, an image, usually with an arbitrary amplitude profile, is expressed as a linear combination of sinusoidal surfaces (Fig. 1.3), which are sinusoids with two frequency variables. The arbitrary amplitude profile of a practical image is shown in Fig. 1.1b. A sinusoidal surface is a corrugation made of alternate parallel ridges and grooves. The DFT is defined for any length. However, due to the availability of practically efficient algorithms, the length of a signal is usually assumed to be an integral power of 2. Further, we assume, for the most part, that the two dimensions of an image are equal. As the 2-D DFT is separable, almost all the computation required involves 1-D DFT algorithms. For ease of manipulation and compactness, the mathematically equivalent complex sinusoid is mostly used in the analysis instead of the real sinusoid. The two forms of the sinusoid are related by the Euler's formula.

The 2-D DFT decomposes an image into its components (Fourier analysis) by correlating the input image with each of the basis functions. The 2-D DFT of a $N \times N$ image $x(m, n)$ is defined as

$$X(k,l) = \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} x(m,n) e^{-j\frac{2\pi}{N}(mk+nl)}, \quad k,l = 0,1,\ldots,N-1. \tag{3.3}$$

The 2-D IDFT reconstructs the input image by summing the basis functions multiplied by the corresponding DFT coefficients (Fourier synthesis). The 2-D IDFT is given by

$$x(m,n) = \frac{1}{N^2} \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} X(k,l) e^{j\frac{2\pi}{N}(mk+nl)}, \quad m,n = 0,1,\ldots,N-1. \tag{3.4}$$

The DFT coefficients computed using Eq. (3.3) places the DC coefficient $X(0, 0)$ in the top-left-hand corner of the coefficient matrix. While that is the format used for most of the computation, for visual purposes, it is often desirable to place the coefficient $X(0, 0)$ in the center of the display. The center-zero format of the 2-D DFT, with N even, is defined as

$$X(k,l) = \sum_{m=-\frac{N}{2}}^{\frac{N}{2}-1} \sum_{n=-\frac{N}{2}}^{\frac{N}{2}-1} x(m,n) e^{-j\frac{2\pi}{N}(mk+nl)}, \quad k,l = -\frac{N}{2}, -\frac{N}{2}+1, \ldots, \frac{N}{2}-1$$

The corresponding 2-D IDFT is given by

$$x(m,n) = \frac{1}{N^2} \sum_{k=-\frac{N}{2}}^{\frac{N}{2}-1} \sum_{l=-\frac{N}{2}}^{\frac{N}{2}-1} X(k,l) e^{j\frac{2\pi}{N}(mk+nl)}, \quad m,n = -\frac{N}{2}, -\frac{N}{2}+1, \ldots, \frac{N}{2}-1$$

By swapping of the quadrants of the image or the spectrum, the desired format can be obtained from the other.

3.3 DFT Representation of Images

For real-valued images, the DFT coefficients always occur as real values or complex conjugate pairs. Coefficients

$$X(0,0),\ X\left(\frac{N}{2}, 0\right),\ X\left(0, \frac{N}{2}\right),\ X\left(\frac{N}{2}, \frac{N}{2}\right)$$

are real-valued, as the basis functions are of the form 1 and $(-1)^n$, and the rest are complex conjugate pairs. For example,

$$\begin{aligned} 2|X(k,l)|\cos\left(\frac{2\pi}{N}(mk+nl)+\angle(X(k,l))\right) \\ = X(k,l)e^{j\frac{2\pi}{N}(mk+nl)} + X^*(k,l)e^{-j\frac{2\pi}{N}(mk+nl)} \\ = X(k,l)e^{j\frac{2\pi}{N}(mk+nl)} + X^*(k,l)e^{j\frac{2\pi}{N}(m(N-k)+n(N-l))} \end{aligned}$$

With $X(k,l) = X_r(k,l) + jX_i(k,l)$, the magnitude is

$$|X(k,l)| = \sqrt{X_r^2(k,l) + X_i^2(k,l)}$$

and the phase is

$$\angle X(k,l) = \tan^{-1}\frac{X_i(k,l)}{X_r(k,l)}$$

Using Eq. (3.4), with N even, the image can be expressed as a sum its constituent sinusoidal surfaces.

$$\begin{aligned} x(m,n) = \frac{1}{N^2}\Bigg(X(0,0) + X\left(\frac{N}{2},0\right)\cos(\pi m) + X\left(0,\frac{N}{2}\right)\cos(\pi n) + X\left(\frac{N}{2},\frac{N}{2}\right)\cos(\pi(m+n)) \\ +2\sum_{k=1}^{\frac{N}{2}-1}\left(|X(k,0)|\cos\left(\frac{2\pi}{N}mk+\angle(X(k,0))\right)\right) \\ +2\sum_{l=1}^{\frac{N}{2}-1}\left(|X(0,l)|\cos\left(\frac{2\pi}{N}nl+\angle(X(0,l))\right)\right) \\ +2\sum_{l=1}^{\frac{N}{2}-1}\left(|X\left(\frac{N}{2},l\right)|\cos\left(\frac{2\pi}{N}\left(m\frac{N}{2}+nl\right)+\angle\left(X\left(\frac{N}{2},l\right)\right)\right)\right) \\ +2\sum_{k=1}^{\frac{N}{2}-1}\sum_{l=1}^{N-1}\left(|X(k,l)|\cos\left(\frac{2\pi}{N}(mk+nl)+\angle(X(k,l))\right)\right)\Bigg), \quad m,n=0,1,\ldots,N-1 \end{aligned} \tag{3.5}$$

Therefore,

$$\frac{N^2}{2}+2$$

different sinusoidal surfaces constitute a $N \times N$ real image.

In general, computing the transform coefficients is finding the correlation of a signal with each of the basis functions. In finding the DFT coefficients of an image, from Eq. (3.3), we multiply the image with the N^2 basis images. For each pair of

coefficient indices (k, l), we can find out the corresponding basis image by varying the indices m and n in

$$e^{j\frac{2\pi}{N}(mk+nl)}$$

Therefore, the computational complexity of computing the 2-D DFT from the definition is $O(N^4)$. The row–column method reduces this complexity to $O(N^3)$. The complexity is further reduced to $O(N^2 \log_2 N)$ using fast 1-D DFT algorithms with the length of the 1-D DFT being a power of 2.

Example 3.2 Find the 2-D DFT of the 4×4 2-D unit-impulse signal

$$x(m, n) = \delta(m, n)$$

and its constituent sinusoidal surfaces.

Solution
The impulse (on the left) and its 2-D DFT are

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \leftrightarrow \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

We assume that the top-left-hand corner of the image is the origin, with its coordinates $(0, 0)$. The row variable m increases downward and the column variable n increases toward right. An impulse has a uniform spectrum. That is, the value of all the coefficients is equal to 1. This DFT can be computed using the 2-D DFT definition. But it is more efficient to compute the 1-D DFT of the rows of the input followed by 1-D DFT of the columns of the resulting output or vice versa. Computing the 1-D DFT of the rows, we get

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

The 1-D DFT of the columns yields the 2-D DFT of the impulse.

A 4×4 real image is composed of 10 real sinusoidal surfaces.

$$\begin{aligned} \delta(m, n) &= \frac{1}{16} \sum_{k=0}^{3} \sum_{l=0}^{3} e^{j\frac{2\pi}{4}(km+ln)}, \quad m, n = 0, 1, 2, 3 \\ &= \frac{1}{16} \Big(1 + (-1)^m + (-1)^n + (-1)^{(m+n)} \\ &\quad + 2\cos\left(\frac{2\pi}{4}m\right) + 2\cos\left(\frac{2\pi}{4}n\right) + 2\cos\left(\frac{2\pi}{4}(2m+n)\right) \\ &\quad + 2\cos\left(\frac{2\pi}{4}(m+n)\right) + 2\cos\left(\frac{2\pi}{4}(m+2n)\right) + 2\cos\left(\frac{2\pi}{4}(m+3n)\right)\Big) \end{aligned}$$

We can find the 10 real sinusoidal surfaces that constitute the impulse signal using this equation. Four individual frequency coefficients (4 surfaces) and six pairs (6 surfaces) form the constituent sinusoidal surfaces.

For $X(0,0) = 1$,

$$x(m,n) = \frac{1}{16} e^{j\frac{2\pi}{4}(0m+0n)} = \frac{1}{16}, \quad m,n = 0,1,2,3$$

$$\frac{1}{16}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \leftrightarrow \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

This is the DC component.

For $X(2,0) = 1$,

$$x(m,n) = \frac{1}{16} e^{j\frac{2\pi}{4}(2m+0n)} = \frac{1}{16}\cos(\pi m) = \frac{1}{16}(-1)^m, \quad m,n = 0,1,2,3$$

$$\frac{1}{16}\begin{bmatrix} 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 \\ 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 \end{bmatrix} \leftrightarrow \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Odd-indexed rows are negative-valued. Four cosine waves, with frequency index 2, are stacked in the horizontal direction.

For $X(0,2) = 1$,

$$x(m,n) = \frac{1}{16} e^{j\frac{2\pi}{4}(0m+2n)} = \frac{1}{16}\cos(\pi n) = \frac{1}{16}(-1)^n, \quad m,n = 0,1,2,3$$

$$\frac{1}{16}\begin{bmatrix} 1 & -1 & 1 & -1 \\ 1 & -1 & 1 & -1 \\ 1 & -1 & 1 & -1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \leftrightarrow \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Odd-indexed columns are negative-valued. Four cosine waves, with frequency index 2, are stacked in the vertical direction.

For $X(2,2) = 1$,

$$x(m,n) = \frac{1}{16} e^{j\frac{2\pi}{4}(2m+2n)} = \frac{1}{16}\cos(\pi(m+n)) = \frac{1}{16}(-1)^{(m+n)}, \quad m,n = 0,1,2,3$$

$$\frac{1}{16}\begin{bmatrix} 1 & -1 & 1 & -1 \\ -1 & 1 & -1 & 1 \\ 1 & -1 & 1 & -1 \\ -1 & 1 & -1 & 1 \end{bmatrix} \leftrightarrow \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

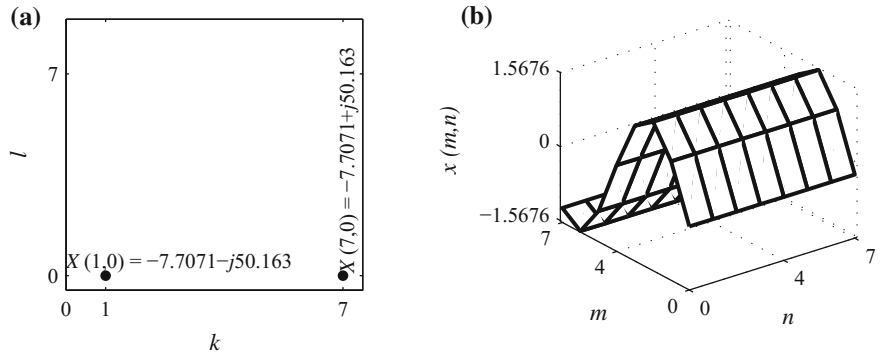


Fig. 3.4 **a** A 8 × 8 2-D DFT spectrum; **b** the corresponding sinusoidal surface, $x(m, n) = 1.586\cos(\frac{2\pi}{8}m - 98.7347°)$

The pixels with the sum of their coordinates odd are negative-valued. Four cosine waves, with frequency index 2, are stacked in the vertical direction with a shift of (πm) or vice versa.
For $X(1, 1) = 1$ and $X(3, 3) = X(4 - 1, 4 - 1) = X(-1, -1) = 1$,

$$x(m,n) = \frac{1}{16}\left(e^{j\frac{2\pi}{4}(m+n)} + e^{-j\frac{2\pi}{4}(m+n)}\right) = \frac{1}{8}\cos\left(\frac{2\pi}{4}(m+n)\right), \quad m, n = 0, 1, 2, 3$$

$$\frac{1}{8}\begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & -1 & 0 & 1 \\ -1 & 0 & 1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} \leftrightarrow \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Four cosine waves, with frequency index 1, are stacked in the vertical direction with a shift of $(\pi/2)m$ in the horizontal direction or vice versa. The other sinusoidal surfaces can be determined similarly. ∎

Consider the 8 × 8 sinusoidal surface shown in Fig. 3.4b and its 2-D DFT shown in Fig. 3.4a. The coefficients $X(1, 0) = (-7.7071 - j50.163)$ and $X(7, 0) = (-7.7071 + j50.163)$ represent a stack of sinusoids along the n-axis with frequency 1 and amplitude 1.586 and phase −98.7347°.

$$\begin{aligned} x(m,n) &= \frac{1}{64}\left((-7.7071 - j50.163)e^{j\frac{2\pi}{8}m} + (-7.7071 + j50.163)e^{-j\frac{2\pi}{8}m}\right) \\ &= \frac{2}{64}|(-7.7071 - j50.163)|\cos\left(\frac{2\pi}{8}m + \angle(-7.7071 - j50.163)\right) \\ &= 1.586\cos\left(\frac{2\pi}{8}m - 98.7347°\right) \end{aligned}$$

∎

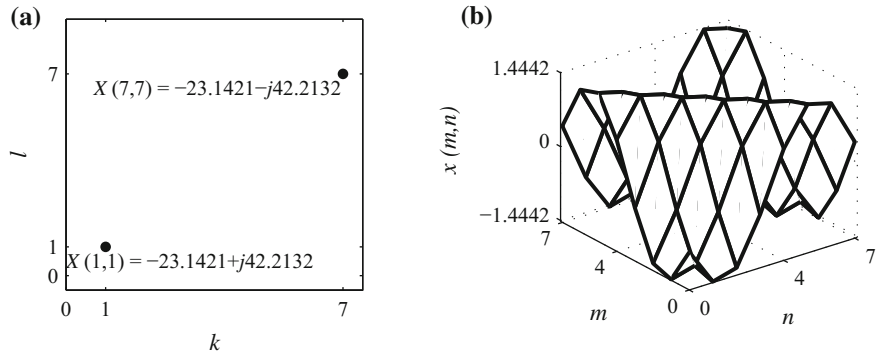


Fig. 3.5 **a** A 8 × 8 2-D DFT spectrum; **b** the corresponding sinusoidal surface, $x(m, n) = 1.5044\cos(\frac{2\pi}{8}(m+n) + 118.7324°)$

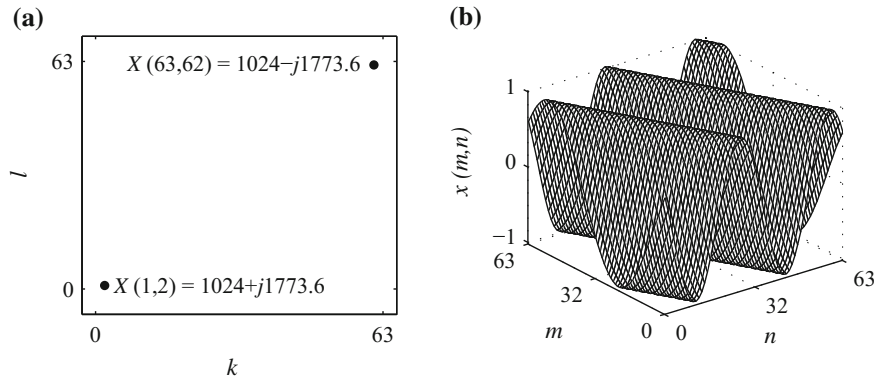


Fig. 3.6 **a** A 64 × 64 2-D spectrum; **b** the corresponding sinusoidal surface, $x(m, n) = \cos(\frac{2\pi}{64}(m + 2n) + \frac{\pi}{3})$

The sinusoidal surface shown in Fig. 3.5b and its DFT shown in Fig. 3.5a are a DFT pair. With the coefficients $X(1, 1) = (-23.1421 + j42.2132)$ and $X(7, 7) = (-23.1421 - j42.2132)$,

$$\begin{aligned} x(m,n) &= \frac{1}{64}\left((-23.1421 + j42.2132)e^{j\frac{2\pi}{8}(m+n)} + (-23.1421 - j42.2132)e^{-j\frac{2\pi}{8}(m+n)}\right) \\ &= \frac{2}{64}|(-23.1421 + j42.2132)|\cos\left(\frac{2\pi}{8}(m+n) + \angle(-23.1421 + j42.2132)\right) \\ &= 1.5044\cos\left(\frac{2\pi}{8}(m+n) + 118.7324°\right) \end{aligned}$$

Consider the sinusoidal surface shown in Fig. 3.6b and its DFT shown in Fig. 3.6a. With the coefficients $X(1, 2) = 1024 + j1773.6$ and $X(63, 62) = 1024 - j1773.6$,

$$x(m,n) = \frac{1}{4096}\left((1024 + j1773.6)e^{j\frac{2\pi}{64}(m+2n)} + (1024 - j1773.6)e^{-j\frac{2\pi}{64}(m+2n)}\right)$$
$$= \frac{1}{(2)}\left(e^{j(\frac{2\pi}{64}(m+2n)+\frac{\pi}{3})} + e^{-j(\frac{2\pi}{64}(m+2n)+\frac{\pi}{3})}\right) = \cos\left(\frac{2\pi}{64}(m+2n) + \frac{\pi}{3}\right)$$

Figure 3.7a shows a 256×256 black and white rose image. Therefore, the image has a sharp discontinuity between black and white parts of the image. In addition, there is a small white dot in the center similar to an impulse function. These two reasons make its spectrum, shown in Fig. 3.7b, rich in frequency components and the convergence (rate of decay of the magnitude of the frequency coefficients) is slow. Figure 3.7c shows its reconstruction with the DC component alone, which is a constant function, equal to the average value of the image. Figure 3.7d shows its reconstruction with its DC component and the first frequency components on the two coordinate axes of the spectrum. The l-axis component and the k-axis component are shown in Fig. 3.7e, f, respectively. These are sinusoidal surfaces in contrast to sinusoidal curves in the 1-D Fourier analysis. Starting with image (d), the addition of more frequency components makes the image more closer to the original by constructive and destructive interference of the components. Figure 3.8a–f shows the reconstruction of the image with the first 2×2, 4×4, 8×8, 16×16, 32×32, and all DFT the coefficients.

3.4 Computation of the 2-D DFT

For computing each coefficient, the use of Eq. (3.3), as such, results in a computational complexity of $O(N^2)$. Therefore, the computational complexity to compute N^2 coefficients becomes $O(N^4)$. The complex exponential basis functions are separable. That is,

$$e^{j\frac{2\pi}{N}(km+ln)} = e^{j\frac{2\pi}{N}(ln)}e^{j\frac{2\pi}{N}(km)}$$

Therefore,

$$X(k,l) = \sum_{n=0}^{N-1}\sum_{m=0}^{N-1} x(m,n)e^{-j\frac{2\pi}{N}nl}e^{-j\frac{2\pi}{N}mk} \tag{3.6}$$

The 2-DFT can be obtained by computing the 1-D DFT of each column of the image followed by the computation of the 1-D DFT of each row of the resulting data or vice versa. This method is called the row–column method. The two decomposed forms of Eq. (3.3) are

$$X(k,l) = \sum_{m=0}^{N-1}\left\{\sum_{n=0}^{N-1} x(m,n)e^{-j\frac{2\pi}{N}nl}\right\}e^{-j\frac{2\pi}{N}mk} \tag{3.7}$$

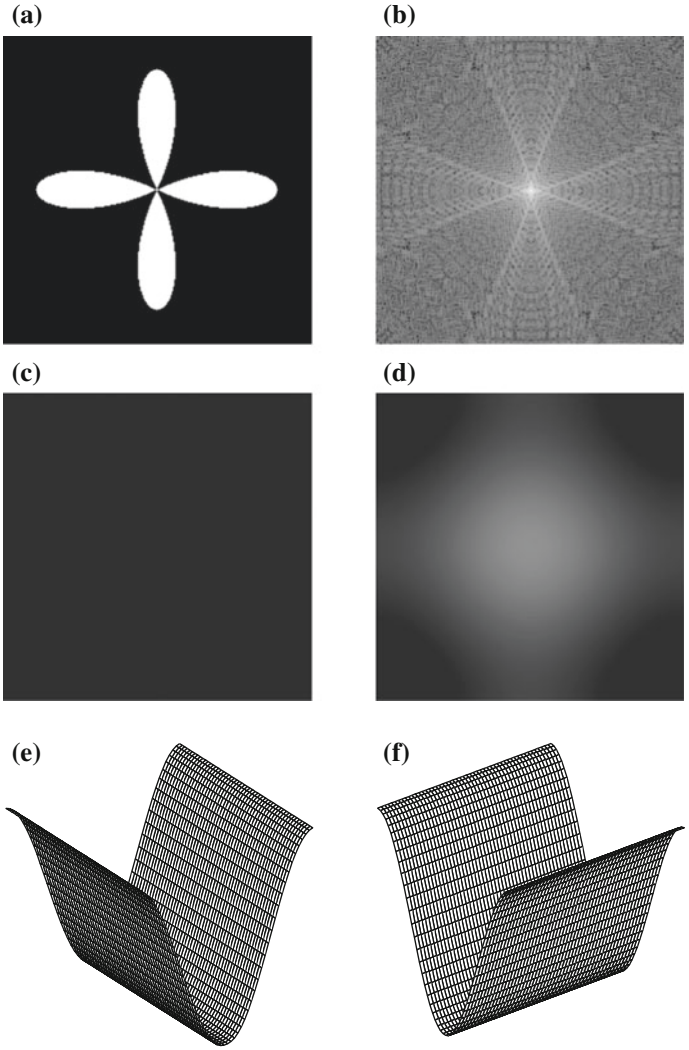


Fig. 3.7 **a** A 256×256 *black* and *white* rose image; **b** its 2-D DFT magnitude (log scale) spectrum $\log_e(1 + |X(k, l)|)$; **c** reconstruction with its DC component alone; **d** reconstruction with its DC component and the first frequency components on the k-axis and l-axis axis of the spectrum; **e** the l-axis component; **f** the k-axis component

$$X(k,l) = \sum_{n=0}^{N-1} \left\{ \sum_{m=0}^{N-1} x(m,n) e^{-j\frac{2\pi}{N}mk} \right\} e^{-j\frac{2\pi}{N}nl} \tag{3.8}$$

The expression inside the braces is 1-D DFT of each row in Eq. (3.7) and of each column in Eq. (3.8). The DFT of the $N \times N$ matrix $x(m, n)$ can be computed in two stages. For example, the 1-D DFT of each row of the input image results in

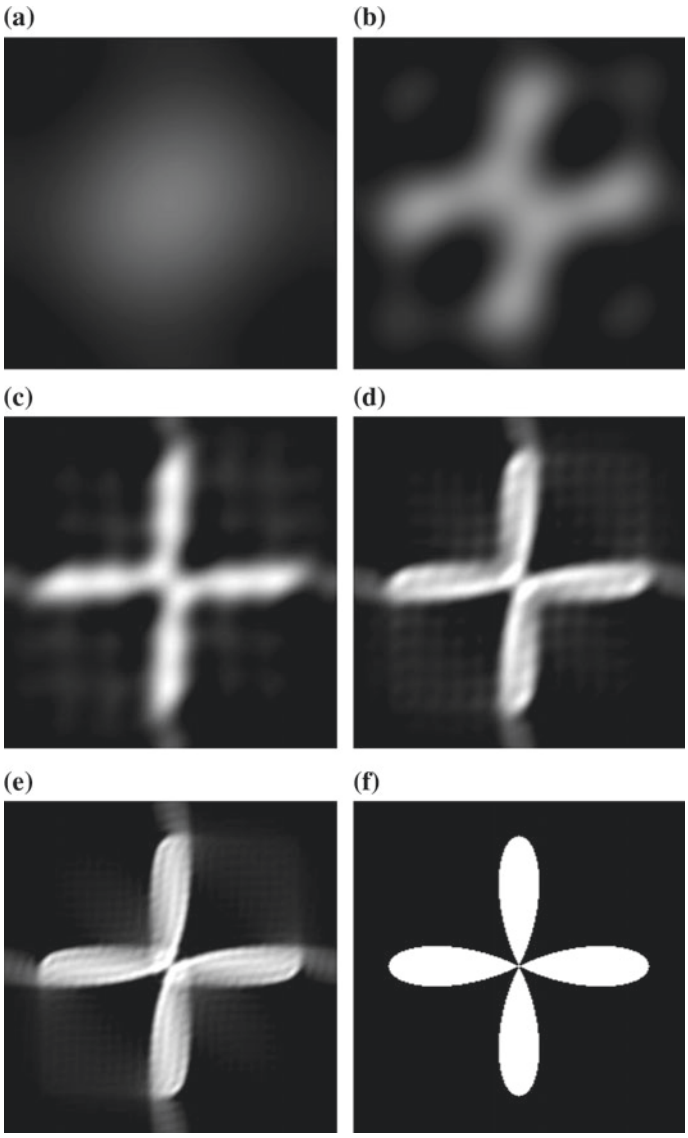


Fig. 3.8 **a–f** Reconstruction of the image in Fig. 3.7a with the first 2×2, 4×4, 8×8, 16×16, 32×32 and all the DFT coefficients

$$X(m,l) = \sum_{n=0}^{N-1} x(m,n) e^{-j\frac{2\pi}{N}nl}, \quad m,l = 0,1,\ldots,N-1$$

Then, the 1-D DFT of each column of $X(m,l)$ yields the 2-D DFT.

$$X(k,l) = \sum_{m=0}^{N-1} X(m,l) e^{-j\frac{2\pi}{N}mk}, \quad k,l = 0,1,\ldots,N-1$$

The decomposition of a 2-D DFT into $2N$ 1-D DFTs reduces the computational complexity to $O(N^3)$.

Example 3.3 Compute the 2-D DFT of the following 4×4 image using the row–column method.

$$\begin{array}{c} \\ m \\ \downarrow \end{array} \begin{array}{c} n \rightarrow \\ \begin{bmatrix} 1 & 2 & 3 & 1 \\ -2 & 3 & 1 & 4 \\ 1 & 1 & 2 & 2 \\ 3 & 1 & 2 & 4 \end{bmatrix} \end{array}$$

Reconstruct the image from the DFT coefficients using the IDFT.

Solution
As the 2-D DFT is decomposable, Eq. (3.3) can be written as the product of 3 matrices.

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 & 1 \\ -2 & 3 & 1 & 4 \\ 1 & 1 & 2 & 2 \\ 3 & 1 & 2 & 4 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix}$$

Premultiplying the image matrix by the left transform matrix is computing 1-D DFT of the columns, and we get

$$\begin{bmatrix} 3 & 7 & 8 & 11 \\ j5 & 1-j2 & 1+j & -1 \\ 1 & -1 & 2 & -5 \\ -j5 & 1+j2 & 1-j1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix}$$

Postmultiplying the partially transformed image matrix by the right transform matrix is computing 1-D DFT of the rows. The 2-D DFT of the image is

$$\begin{array}{c} \\ k \\ \downarrow \end{array} \begin{array}{c} l \rightarrow \\ \begin{bmatrix} 29 & -5+j4 & -7 & -5-j4 \\ 1+j4 & -3+j2 & 1+j8 & 1+j6 \\ -3 & -1-j4 & 9 & -1+j4 \\ 1-j4 & 1-j6 & 1-j8 & -3-j2 \end{bmatrix} \end{array} = X(k,l)$$

The input image can be reconstructed by using 1-D IDFTs and the row–column method. The DFT and IDFT transform matrices are closely related. Therefore, the 2-D IDFT can be computed using the DFT simply by swapping the real and imaginary parts in reading the input and writing the output values. The DFT after swapping the real and imaginary parts is given by

$$\begin{bmatrix} j29 & 4-j5 & -j7 & -4-j5 \\ 4+j1 & 2-j3 & 8+j1 & 6+j1 \\ -j3 & -4-j1 & j9 & 4-j1 \\ -4+j1 & -6+j1 & -8+j1 & -2-j3 \end{bmatrix}$$

The result of computing the column DFTs yields

$$\begin{bmatrix} j28 & -4-j8 & j4 & 4-j8 \\ j24 & 4-j12 & -j32 & -4-j12 \\ j24 & 4-j4 & 0 & -4-j4 \\ j40 & 12+j4 & 0 & -12+j4 \end{bmatrix}$$

The result of computing the row DFTs gives

$$j \begin{bmatrix} 16 & 32 & 48 & 16 \\ -32 & 48 & 16 & 64 \\ 16 & 16 & 32 & 32 \\ 48 & 16 & 32 & 64 \end{bmatrix}$$

These values divided by $N^2 = 16$, and swapping the real and imaginary parts yields the input image. ∎

As stated for the 1-DFT, the display of $\log_{10}(1+|X(k,l)|)$ instead of $|X(k,l)|$ gives a better contrast. For the example image, the magnitude of the spectrum, $|X(k,l)|$, in the center-zero format, is

$$\begin{bmatrix} 9.0000 & 4.1231 & 3.0000 & 4.1231 \\ 8.0623 & 3.6056 & 4.1231 & 6.0828 \\ 7.0000 & 6.4031 & 29.0000 & 6.4031 \\ 8.0623 & 6.0828 & 4.1231 & 3.6056 \end{bmatrix}$$

whereas $\log_{10}(1 + |X(k,l)|)$ is

$$\begin{bmatrix} 1.0000 & 0.7095 & 0.6021 & 0.7095 \\ 0.9572 & 0.6633 & 0.7095 & 0.8502 \\ 0.9031 & 0.8694 & 1.4771 & 0.8694 \\ 0.9572 & 0.8502 & 0.7095 & 0.6633 \end{bmatrix}$$

Table 3.1 A 8×8 input image

−2	−2	−3	−3	−3	3	5	4
−1	−3	0	1	4	3	4	4
2	5	3	2	1	0	−1	2
1	2	3	3	2	2	−6	6
2	−1	1	2	2	1	−4	3
2	−1	−2	−2	1	−1	−1	−2
2	−2	−5	−3	−1	1	0	−2
2	−2	−6	−2	−1	2	0	1

Using the log scale, the ratio between the largest and smallest coefficient has been reduced.

A 8×8 example of the 2-D DFT computation is given. While the 4×4 example is suitable for manual computation, this example is meant for implementing on the computer. The use of the row–column method along with fast 1-D DFT algorithms (presented in Appendix) is the practical method for DFT computation. Sufficient practice with this type of examples is recommended. The input image is shown in Table 3.1. The 1-D DFT of the rows of the input image is shown in Table 3.2. By computing the 1-D DFT of the columns of these values, the 2-D DFT, shown in Table 3.4, is obtained.

Alternately, the 1-D DFT of the columns of the input image can be computed first, shown in Table 3.3. By computing the 1-D DFT of the rows of these values, the 2-D DFT, shown in Table 3.4, is obtained. The 2-D DFT of the input image, in the center-zero format, is shown in Table 3.5. The magnitude of the 2-D DFT $|X(k,l)|$ in the center-zero format is shown in Table 3.6. The magnitude of the 2-D DFT in the center-zero format and in the log scale, $\log_{10}(1 + |X(k,l)|)$, is shown in Table 3.7.

3.5 Properties of the 2-D DFT

Properties relate the effect of the characteristics and operations on images in one domain into another. Further, the computation of the transforms becomes simpler.

Linearity

The 2-D DFT of a linear combination of a set of discrete images is equal to the same linear combination of their individual DFTs. Let $x_1(m,n) \leftrightarrow X_1(k,l)$ and $x_2(m,n) \leftrightarrow X_2(k,l)$. Then,

$$ax_1(m,n) + bx_2(m,n) \leftrightarrow aX_1(k,l) + bX_2(k,l)$$

Table 3.2 1-D DFT of the rows of the input image

$-1.00 + \text{j}0.00$	$2.41 + \text{j}16.49$	$-7.00 + \text{j}0.00$	$-0.41 + \text{j}0.49$	$-5.00 + \text{j}0.00$	$-0.41 - \text{j}0.49$	$-7.00 + \text{j}0.00$	$2.41 - \text{j}16.49$
$12.00 + \text{j}0.00$	$-7.12 + \text{j}10.36$	$-1.00 + \text{j}5.00$	$-2.88 + \text{j}2.36$	$2.00 + \text{j}0.00$	$-2.88 - \text{j}2.36$	$-1.00 - \text{j}5.00$	$-7.12 - \text{j}10.36$
$14.00 + \text{j}0.00$	$4.54 - \text{j}7.54$	$1.00 - \text{j}1.00$	$-2.54 + \text{j}0.46$	$-4.00 + \text{j}0.00$	$-2.54 - \text{j}0.46$	$1.00 + \text{j}1.00$	$4.54 + \text{j}7.54$
$13.00 + \text{j}0.00$	$1.12 - \text{j}6.88$	$6.00 + \text{j}5.00$	$-3.12 + \text{j}11.12$	$-13.00 + \text{j}0.00$	$-3.12 - \text{j}11.12$	$6.00 - \text{j}5.00$	$1.12 + \text{j}6.88$
$6.00 + \text{j}0.00$	$-0.71 - \text{j}2.88$	$7.00 + \text{j}5.00$	$0.71 + \text{j}7.12$	$-4.00 + \text{j}0.00$	$0.71 - \text{j}7.12$	$7.00 - \text{j}5.00$	$-0.71 + \text{j}2.88$
$-6.00 + \text{j}0.00$	$1.00 + \text{j}1.00$	$6.00 - \text{j}2.00$	$1.00 - \text{j}1.00$	$6.00 + \text{j}0.00$	$1.00 + \text{j}1.00$	$6.00 + \text{j}2.00$	$1.00 - \text{j}1.00$
$-10.00 + \text{j}0.00$	$1.59 + \text{j}7.83$	$6.00 - \text{j}4.00$	$4.41 - \text{j}2.17$	$2.00 + \text{j}0.00$	$4.41 + \text{j}2.17$	$6.00 + \text{j}4.00$	$1.59 - \text{j}7.83$
$-6.00 + \text{j}0.00$	$2.29 + j10.95$	$7.00 - \text{j}1.00$	$3.71 - \text{j}1.05$	$-4.00 + \text{j}0.00$	$3.71 + \text{j}1.05$	$7.00 + \text{j}1.00$	$2.29 - \text{j}10.95$

Table 3.3 1-D DFT of the columns of the input image

$8.00 + j0.00$	$-4.00 + j0.00$	$-9.00 + j0.00$	$-2.00 + j0.00$	$5.00 + j0.00$	$11.00 + j0.00$	$-3.00 + j0.00$	$16.00 + j0.00$
$-5.41 + j2.83$	$-5.24 - j8.41$	$-8.95 - j15.78$	$-6.41 - j10.66$	$-5.00 - j6.24$	$4.83 - j1.83$	$16.78 + j1.71$	$1.71 - j11.78$
$-4.00 + j2.00$	$-6.00 + j4.00$	$0.00 - j1.00$	$0.00 + j2.00$	$-1.00 - j4.00$	$3.00 + j2.00$	$2.00 - j9.00$	$7.00 + j5.00$
$-2.59 + j2.83$	$3.24 + j5.59$	$0.95 + j0.22$	$-3.59 - j0.66$	$-5.00 - j2.24$	$-0.83 - j3.83$	$1.22 - j0.29$	$0.29 - j3.78$
$0.00 + j0.00$	$4.00 + j0.00$	$1.00 + j0.00$	$-2.00 + j0.00$	$-7.00 + j0.00$	$-1.00 + j0.00$	$3.00 + j0.00$	$-2.00 + j0.00$
$-2.59 - j2.83$	$3.24 - j5.59$	$0.95 - j0.22$	$-3.59 + j0.66$	$-5.00 + j2.24$	$-0.83 + j3.83$	$1.22 + j0.29$	$0.29 + j3.78$
$-4.00 - j2.00$	$-6.00 - j4.00$	$0.00 + j1.00$	$0.00 - j2.00$	$-1.00 + j4.00$	$3.00 - j2.00$	$2.00 + j9.00$	$7.00 - j5.00$
$-5.41 - j2.83$	$-5.24 + j8.41$	$-8.95 + j15.78$	$-6.41 + j10.66$	$-5.00 + j6.24$	$4.83 + j1.83$	$16.78 - j1.71$	$1.71 + j11.78$

Table 3.4 2-D DFT of the input image

$22.00 + j0.00$	$5.12 + j29.33$	$25.00 + j7.00$	$0.88 + j17.33$	$-20.00 + j0.00$	$0.88 - j17.33$	$25.00 - j7.00$	$5.12 - j29.33$
$-7.71 - 50.16$	$-23.14 + j42.21$	$-6.05 + j6.36$	$14.59 + j1.66$	$2.54 + j15.19$	$-12.66 + j27.38$	$-30.44 + j14.95$	$19.56 - j34.97$
$1.00 + j1.00$	$2.88 + j22.85$	$-8.00 + j18.00$	$-10.29 + j11.78$	$-7.00 - j25.00$	$7.12 - j6.85$	$-6.00 - j2.00$	$-11.71 - j3.78$
$-6.29 - j2.16$	$17.41 + j9.66$	$-3.56 - j5.05$	$5.14 + j0.21$	$-4.54 + j3.19$	$-11.56 + j1.03$	$-15.95 + j6.36$	$-1.34 + j9.38$
$-4.00 + j0.00$	$10.54 - j1.54$	$-11.00 - j7.00$	$3.46 - j5.54$	$-2.00 + j0.00$	$3.46 + j5.54$	$-11.00 + j7.00$	$10.54 + j1.54$
$-6.29 + j2.16$	$-1.34 - j9.38$	$-15.95 - j6.36$	$-11.56 - j1.03$	$-4.54 - j3.19$	$5.14 - j0.21$	$-3.56 + j5.05$	$17.41 - j9.66$
$1.00 - j1.00$	$-11.71 + j3.78$	$-6.00 + j2.00$	$7.12 + j6.85$	$-7.00 + j25.00$	$-10.29 - j11.78$	$-8.00 - j18.00$	$2.88 - j22.85$
$-7.71 + j50.16$	$19.56 + j34.97$	$-30.44 - j14.95$	$-12.66 - j27.38$	$2.54 - j15.19$	$14.59 - j1.66$	$-6.05 - j6.36$	$-23.14 - j42.21$

Table 3.5 2-D DFT of the input image in the center-zero format

$-2.00 + 0.00$	$3.46 + 5.54$	$-11.00 + 7.00$	$10.54 + 1.54$	$-4.00 + 0.00$	$10.54 - 1.54$	$-11.00 - 7.00$	$3.46 - 5.54$
$-4.54 - 3.19$	$5.14 - 0.21$	$-3.56 + 5.05$	$17.41 - 9.66$	$-6.29 + 2.16$	$-1.34 - 9.38$	$-15.95 - 6.36$	$-11.56 - 1.03$
$-7.00 + 25.00$	$-10.29 - 11.78$	$-8.00 - 18.00$	$2.88 - 22.85$	$1.00 - 1.00$	$-11.71 + 3.78$	$-6.00 + 2.00$	$7.12 + 6.85$
$2.54 - 15.19$	$14.59 - 1.66$	$-6.05 - 6.36$	$-23.14 - 42.21$	$-7.71 + 50.16$	$19.56 + 34.97$	$-30.44 - 14.95$	$-12.66 - 27.38$
$-20.00 + 0.00$	$0.88 - 17.33$	$25.00 - 7.00$	$5.12 - 29.33$	$22.00 + 0.00$	$5.12 + 29.33$	$25.00 + 7.00$	$0.88 + 17.33$
$2.54 + 15.19$	$-12.66 + 27.38$	$-30.44 + 14.95$	$19.56 - 34.97$	$-7.71 - 50.16$	$-23.14 + 42.21$	$-6.05 + 6.36$	$14.59 + 1.66$
$-7.00 - 25.00$	$7.12 - 6.85$	$-6.00 - 2.00$	$-11.71 - 3.78$	$1.00 + 1.00$	$2.88 + 22.85$	$-8.00 + 18.00$	$-10.29 + 11.78$
$-4.54 + 3.19$	$-11.56 + 1.03$	$-15.95 + 6.36$	$-1.34 + 9.38$	$-6.29 - 2.16$	$17.41 + 9.66$	$-3.56 - 5.05$	$5.14 + 0.21$

Table 3.6 The magnitude of the 2-D DFT, $|X(k,l)|$, of the input image in the center-zero format

2.00	6.53	13.04	10.65	4.00	10.65	13.04	6.53
5.55	5.15	6.18	19.91	6.65	9.48	17.17	11.60
25.96	15.64	19.70	23.03	1.41	12.30	6.32	9.88
15.40	14.68	8.78	48.14	50.75	40.07	33.91	30.17
20.00	17.36	25.96	29.78	22.00	29.78	25.96	17.36
15.40	30.17	33.91	40.07	50.75	48.14	8.78	14.68
25.96	9.88	6.32	12.30	1.41	23.03	19.70	15.64
5.55	11.60	17.17	9.48	6.65	19.91	6.18	5.15

Table 3.7 The magnitude of the 2-D DFT of the input image in the center-zero format and in the log scale, $\log_{10}(1 + |X(k,l)|)$

0.48	0.88	1.15	1.07	0.70	1.07	1.15	0.88
0.82	0.79	0.86	1.32	0.88	1.02	1.26	1.10
1.43	1.22	1.32	1.38	0.38	1.12	0.86	1.04
1.21	1.20	0.99	1.69	1.71	1.61	1.54	1.49
1.32	1.26	1.43	1.49	1.36	1.49	1.43	1.26
1.21	1.49	1.54	1.61	1.71	1.69	0.99	1.20
1.43	1.04	0.86	1.12	0.38	1.38	1.32	1.22
0.82	1.10	1.26	1.02	0.88	1.32	0.86	0.79

where a and b are real or complex constants. Both the images must have the same dimensions. Zero-padding can be used to meet this constraint, if necessary. Linearity holds in both the spatial and frequency domains.

Example 3.4 Compute the DFT of $x_1(m,n)$ and $x_2(m,n)$. Using the linearity property, deduce the DFT of $x_3(m,n) = 3x_1(m,n) + 4x_2(m,n)$ from those of $x_1(m,n)$ and $x_2(m,n)$.

$$x_1(m,n) = \begin{bmatrix} 1 & 2 \\ 3 & 1 \end{bmatrix}, \quad x_2(m,n) = \begin{bmatrix} 1 & 1 \\ 4 & 3 \end{bmatrix}$$

Solution
The individual DFTs are

$$X_1(k,l) = \begin{bmatrix} 7 & 1 \\ -1 & -3 \end{bmatrix}, \quad X_2(k,l) = \begin{bmatrix} 9 & 1 \\ -5 & -1 \end{bmatrix}$$

The DFT of

$$x_3(m,n) = 3x_1(m,n) + 4x_2(m,n) = \begin{bmatrix} 7 & 10 \\ 25 & 15 \end{bmatrix}$$

is

$$X_3(k,l) = 3X_1(k,l) + 4X_2(k,l) = \begin{bmatrix} 57 & 7 \\ -23 & -13 \end{bmatrix}$$

The DFT $X_3(k,l)$ can be verified by directly computing that of $x_3(m,n)$. ∎

Periodicity

An image is periodic if it repeats its values over a period indefinitely, $x(m+M, n+N) = x(m,n)$ for all m, n. The smallest M, N satisfying the constraint are the periods in the two directions. Although a practical image is of finite extent, as the basis signals in Fourier analysis are the sinusoids (which are periodic), an image is assumed to be periodic in both the spatial and frequency domains.

$$\begin{aligned} x(m,n) &= x(m+aM, n+bN), \quad \text{for all } m,n \quad \text{and} \\ X(k,l) &= X(k+aM, l+bN), \quad \text{for all } k,l \end{aligned}$$

where a and b are arbitrary integers. Remember that the useful information in a periodic signal is contained in any one period. An example of periodic extension is given in Chap. 2. The top and bottom edges are considered adjacent and so are the right and left edges. For example, the periodic extension of image $x_1(m,n)$ in Example 3.4 is

$$\begin{bmatrix} & \vdots & \\ & 1\ 2\ 1\ 2\ 1\ 2 & \\ & 3\ 1\ 3\ 1\ 3\ 1 & \\ \ldots & 1\ 2\ \mathbf{1}\ \mathbf{2}\ 1\ 2 & \ldots \\ & 3\ 1\ \mathbf{3}\ \mathbf{1}\ 3\ 1 & \\ & 1\ 2\ 1\ 2\ 1\ 2 & \\ & 3\ 1\ 3\ 1\ 3\ 1 & \\ & \vdots & \end{bmatrix}$$

Circular shift of an image

A shift of a sinusoid results in changing its phase. Its magnitude is not affected. For a $N \times N$ image,

$$x(m,n) \leftrightarrow X(k,l) \rightarrow x(m-m_0, n-n_0) \leftrightarrow X(k,l)e^{-j\frac{2\pi}{N}(km_0+ln_0)}$$

The DFT of $x(m-m_0, n-n_0)$ is

$$= \sum_{m=0}^{N-1}\sum_{n=0}^{N-1} x(m-m_0, n-n_0)e^{-j\frac{2\pi}{N}(mk+nl)}$$
$$= e^{-j\frac{2\pi}{N}(km_0+ln_0)} \sum_{m=0}^{N-1}\sum_{n=0}^{N-1} x(m-m_0, n-n_0)e^{-j\frac{2\pi}{N}((m-m_0)k+(n-n_0)l)}$$
$$= e^{-j\frac{2\pi}{N}(km_0+ln_0)} X(k,l)$$

Example 3.5 Find the 2-D DFT of the shifted version $x(m-1, n-2)$ of the image $x(m, n)$ in Example 3.3 using the shift theorem and verify that by computing the DFT of the shifted signal.
Solution
For a 4×4 image, a right shift by one sample interval contributes a phase of $-j$ or $-90°$. Therefore, the 2-D DFT of the shifted image

$$x(m-1, n-2) = \begin{bmatrix} 2 & 4 & 3 & 1 \\ 3 & 1 & 1 & 2 \\ 1 & 4 & -2 & 3 \\ 2 & 2 & 1 & 1 \end{bmatrix}$$

is

$$(-j)^{(k+2l)}X(k,l) = \begin{bmatrix} 29 & 5-j4 & -7 & 5+j4 \\ 4-j1 & -2-j3 & 8-j1 & -6+j1 \\ 3 & -1-j4 & -9 & -1+j4 \\ 4+j1 & -6-j1 & 8+j1 & -2+j3 \end{bmatrix}, \quad \begin{matrix} k=0,1,2,3, \\ l=0,1,2,3 \end{matrix}$$

∎

Circular shift of a spectrum

For a $N \times N$ image,

$$x(m,n) \leftrightarrow X(k,l) \rightarrow x(m,n)e^{j\frac{2\pi}{N}(k_0m+l_0n)} \leftrightarrow X(k-k_0, l-l_0)$$

A specific use of this theorem is that the center-zero spectrum can be obtained with N even and $k_0 = l_0 = \frac{N}{2}$.

Example 3.6 By multiplying the image of Example 3.3 with $(-1)^{(m+n)}$, we get

$$(-1)^{(m+n)}x(m,n) = \begin{bmatrix} 1 & -2 & 3 & -1 \\ 2 & 3 & -1 & 4 \\ 1 & -1 & 2 & -2 \\ -3 & 1 & -2 & 4 \end{bmatrix}$$

The DFT of this image yields the center-zero spectrum of $x(m,n)$. The center-zero spectrum is

$$X(k-2,l-2)=\begin{bmatrix} 9 & -1+j4 & -3 & -1-j4 \\ 1-j8 & -3-j2 & 1-j4 & 1-j6 \\ -7 & -5-j4 & 29 & -5+j4 \\ 1+j8 & 1+j6 & 1+j4 & -3+j2 \end{bmatrix}$$

∎

Circular convolution in the spatial time domain

Let $x(m,n) \leftrightarrow X(k,l)$ and $h(m,n) \leftrightarrow H(k,l)$, $m,n,k,l=0,1,\ldots,N-1$. Then,

$$x(m,n) * h(m,n) \leftrightarrow X(k,l)H(k,l)$$

The circular convolution of $x(m,n)$ and $h(m,n)$ is given by

$$y(m,n)=\sum_{p=0}^{N-1}\sum_{q=0}^{N-1} x(p,q)h(m-p,n-q),\ m,n=0,1,\ldots,N-1$$

Taking the 2-DFT of $y(m,n)$, we get

$$\begin{aligned} Y(k,l) &= \sum_{m=0}^{N-1}\sum_{n=0}^{N-1} y(m,n)e^{-j\frac{2\pi}{N}(km+ln)} \\ &= \sum_{m=0}^{N-1}\sum_{n=0}^{N-1}\left(\sum_{p=0}^{N-1}\sum_{q=0}^{N-1} x(p,q)h(m-p,n-q)\right)e^{-j\frac{2\pi}{N}(km+ln)} \\ &= \sum_{p=0}^{N-1}\sum_{q=0}^{N-1} x(p,q)\left(\sum_{m=0}^{N-1}\sum_{n=0}^{N-1} h(m-p,n-q)\right)e^{-j\frac{2\pi}{N}(km+ln)} \\ &= \sum_{p=0}^{N-1}\sum_{q=0}^{N-1} x(p,q)\left(H(k,l)e^{-j\frac{2\pi}{N}(kp+lq)}\right) = X(k,l)H(k,l) \end{aligned}$$

The spatial-shift property has been used in the derivation. Convolution of two images in the spatial domain becomes multiplication of their transforms in the frequency domain.

Convolve

$$x(m,n)=\begin{bmatrix} 1 & 2 & 3 & 1 \\ -2 & 3 & 1 & 4 \\ 1 & 1 & 2 & 2 \\ 3 & 1 & 2 & 4 \end{bmatrix} \quad h(m,n)=\begin{bmatrix} 2 & 1 & 2 & 2 \\ 3 & 1 & 1 & 2 \\ 1 & 1 & -3 & 2 \\ 0 & 1 & 2 & 3 \end{bmatrix}$$

The DFT of $x(m,n)$ is

$$X(k,l) = \begin{bmatrix} 29.00 + j0.00 & -5.00 + j4.00 & -7.00 + j0.00 & -5.00 - j4.00 \\ 1.00 + j4.00 & -3.00 + j2.00 & 1.00 + j8.00 & 1.00 + j6.00 \\ -3.00 + j0.00 & -1.00 - j4.00 & 9.00 + j0.00 & -1.00 + j4.00 \\ 1.00 - j4.00 & 1.00 - j6.00 & 1.00 - j8.00 & -3.00 - j2.00 \end{bmatrix}$$

The DFT of $h(m,n)$ is

$$H(k,l) = \begin{bmatrix} 21.00 + j0.00 & 4.00 + j5.00 & -5.00 + j0.00 & 4.00 - j5.00 \\ 6.00 - j1.00 & -5.00 - j4.00 & 6.00 - j3.00 & -3.00 - j4.00 \\ -5.00 + j0.00 & 4.00 - j1.00 & -3.00 + j0.00 & 4.00 + j1.00 \\ 6.00 + j1.00 & -3.00 + j4.00 & 6.00 + j3.00 & -5.00 + j4.00 \end{bmatrix}$$

The pointwise product $Y(k,l) = X(k,l)H(k,l)$ is

$$\begin{aligned} Y(k,l) &= X(k,l)H(k,l) \\ &= \begin{bmatrix} 609.00 + j0.00 & -40.00 - j9.00 & 35.00 + j0.00 & -40.00 + j9.00 \\ 10.00 + j23.00 & 23.00 + j2.00 & 30.00 + j45.00 & 21.00 - j22.00 \\ 15.00 + j0.00 & -8.00 - j15.00 & -27.00 + j0.00 & -8.00 + j15.00 \\ 10.00 - j23.00 & 21.00 + j22.00 & 30.00 - j45.00 & 23.00 - j2.00 \end{bmatrix} \end{aligned}$$

The IDFT of $Y(k,l)$ is the convolution output in the spatial domain.

$$y(m,n) = \begin{bmatrix} 44 & 36 & 45 & 36 \\ 31 & 35 & 34 & 37 \\ 23 & 47 & 46 & 35 \\ 43 & 30 & 56 & 31 \end{bmatrix}$$

Circular convolution in the frequency domain

$$x(m,n)h(m,n) \leftrightarrow \frac{1}{N^2} \sum_{p=0}^{N-1} \sum_{q=0}^{N-1} X(p,q)H(k-p, l-q)$$

Circular cross-correlation in the spatial domain

The circular cross-correlation of $x(m,n)$ and $h(m,n)$ is given by

$$r_{xh}(m,n) = \sum_{p=0}^{N-1} \sum_{q=0}^{N-1} x(p,q)h(p-m, q-n) \leftrightarrow H^*(k,l)X(k,l)$$

Since $h(N-m, N-n) \leftrightarrow H^*(k,l)$, this operation can also be interpreted as the convolution of $x(m,n)$ and $h(N-m, N-n)$.

$$r_{hx}(m,n) = r_{xh}(N-m, N-n) = \text{IDFT}(X^*(k,l)H(k,l))$$

Cross-correlation of an image $x(m,n)$ with itself is the autocorrelation operation.

$$r_{xx}(m,n) = \text{IDFT}(|X(k,l)|^2)$$

Sum and difference of sequences

$$X(0,0) = \sum_{m=0}^{N-1}\sum_{n=0}^{N-1} x(m,n)$$

With N even,

$$X\left(\frac{N}{2}, \frac{N}{2}\right) = \sum_{m=0}^{N-1}\sum_{n=0}^{N-1} x(m,n)(-1)^{(m+n)}$$

$$x(0,0) = \frac{1}{N^2}\sum_{k=0}^{N-1}\sum_{l=0}^{N-1} X(k,l)$$

With N even,

$$x\left(\frac{N}{2}, \frac{N}{2}\right) = \frac{1}{N^2}\sum_{k=0}^{N-1}\sum_{l=0}^{N-1} X(k,l)(-1)^{(k+l)}$$

For the image in Example 3.3,

$$\{x(0,0) = 1, x(2,2) = 2, X(0,0) = 29, X(2,2) = 9\}$$

These values can be separately computed and used to check the transform.
The difference

$$x(m,n) - x(m-1,n) \leftrightarrow X(k,l)(1 - e^{-j\frac{2\pi}{N}k})$$

$$x(m,n) - x(m,n-1) \leftrightarrow X(k,l)(1 - e^{-j\frac{2\pi}{N}l})$$

Reversal property

$$x(m,n) \leftrightarrow X(k,l) \rightarrow x(N-m, N-n) \leftrightarrow X(N-k, N-l) = X^*(k,l)$$

Example 3.7 For the image in Example 3.3

$$x(4-m, 4-n) = \begin{bmatrix} 1 & 1 & 3 & 2 \\ 3 & 4 & 2 & 1 \\ 1 & 2 & 2 & 1 \\ -2 & 4 & 1 & 3 \end{bmatrix}$$

$$X(4-k,4-l) = \begin{bmatrix} 29.00+j0.00 & -5.00-j4.00 & -7.00+j0.00 & -5.00+j4.00 \\ 1.00-j4.00 & -3.00-j2.00 & 1.00-j8.00 & 1.00-j6.00 \\ -3.00+j0.00 & -1.00+j4.00 & 9.00+j0.00 & -1.00-j4.00 \\ 1.00+j4.00 & 1.00+j6.00 & 1.00+j8.00 & -3.00+j2.00 \end{bmatrix}$$

∎

Symmetry

The DFT of real-valued data is conjugate symmetric. As there are only $N \times N$ independent real values in a real-valued $N \times N$ image, there can be only the same number of independent values in the transform also. Redundancy is required to have a complex-valued transform, but the storage and computation can be reduced in practical implementation. As has been already pointed out, complex-valued transform is indispensable in Fourier analysis.

The DFT values at diametrically opposite points form complex conjugate pairs.

$$X^*(N-k, N-l) = X(k,l)$$

An equivalent form of the symmetry is

$$X\left(\frac{N}{2} \pm k, \frac{N}{2} \pm l\right) = X^*\left(\frac{N}{2} \mp k, \frac{N}{2} \mp l\right)$$

Example 3.8 Underline the left-half of the nonredundant DFT values of the image in Example 3.3.

Solution

$$\begin{array}{c} \\ k \\ \downarrow \end{array} \begin{array}{c} \quad\quad l \rightarrow \\ \begin{bmatrix} \underline{29} & \underline{-5+j4} & \underline{7} & -5-j4 \\ \underline{1+j4} & \underline{-3+j2} & \underline{1+j8} & 1+j6 \\ \underline{-3} & \underline{-1-j4} & \underline{9} & -1+j4 \\ 1-j4 & \underline{1-j6} & 1-j8 & -3-j2 \end{bmatrix} \end{array}$$

∎

The storage of the 2-D DFT values of rows (columns) 1 to $\frac{N}{2}-1$ and the first $\frac{N}{2}+1$ values of the zeroth and the $\frac{N}{2}$th rows (columns) is sufficient requiring the same amount of storage as for the image matrix. The computation of the DFT requires the computation of $N+2$ 1-D DFT of real-valued data and the computation of $\frac{N}{2}-1$ 1-D DFT of complex-valued data. Figure 3.9 shows one of the two forms of the conjugate symmetry of the 8×8 2-D DFT.

Image rotation

The rotation of an image by an angle θ, about its center, rotates its spectrum also by the same angle. Rotations other than multiple of 90° require interpolation. Consider the following image and its spectrum.

	0	1	2	3	4	5	6	7
0	$X(0,0)$	$X(0,1)$	$X(0,2)$	$X(0,3)$	$X(0,4)$	$X^*(0,3)$	$X^*(0,2)$	$X^*(0,1)$
1	$X(1,0)$	$X(1,1)$	$X(1,2)$	$X(1,3)$	$X(1,4)$	$X^*(7,3)$	$X^*(7,2)$	$X^*(7,1)$
2	$X(2,0)$	$X(2,1)$	$X(2,2)$	$X(2,3)$	$X(2,4)$	$X^*(6,3)$	$X^*(6,2)$	$X^*(6,1)$
3	$X(3,0)$	$X(3,1)$	$X(3,2)$	$X(3,3)$	$X(3,4)$	$X^*(5,3)$	$X^*(5,2)$	$X^*(5,1)$
4	$X(4,0)$	$X(4,1)$	$X(4,2)$	$X(4,3)$	$X(4,4)$	$X^*(4,3)$	$X^*(4,2)$	$X^*(4,1)$
5	$X^*(3,0)$	$X(5,1)$	$X(5,2)$	$X(5,3)$	$X^*(3,4)$	$X^*(3,3)$	$X^*(3,2)$	$X^*(3,1)$
6	$X^*(2,0)$	$X(6,1)$	$X(6,2)$	$X(6,3)$	$X^*(2,4)$	$X^*(2,3)$	$X^*(2,2)$	$X^*(2,1)$
7	$X^*(1,0)$	$X(7,1)$	$X(7,2)$	$X(7,3)$	$X^*(1,4)$	$X^*(1,3)$	$X^*(1,2)$	$X^*(1,1)$

Fig. 3.9 Conjugate symmetry of the 8×8 2-D DFT

$$x(m,n) = \begin{bmatrix} 1 & 2 & 3 & 1 \\ -2 & 3 & 1 & 4 \\ 1 & 1 & 2 & 2 \\ 3 & 1 & 2 & 4 \end{bmatrix} \quad X(k,l) = \begin{bmatrix} 29 & -5+j4 & -7 & -5-j4 \\ 1+j4 & -3+j2 & 1+j8 & 1+j6 \\ -3 & -1-j4 & 9 & -1+j4 \\ 1-j4 & 1-j6 & 1-j8 & -3-j2 \end{bmatrix}$$

The image and its spectrum rotated by an angle of 90° in the counterclockwise direction are

$$x(m',n') = \begin{bmatrix} 1 & 4 & 2 & 4 \\ 3 & 1 & 2 & 2 \\ 2 & 3 & 1 & 1 \\ 1 & -2 & 1 & 3 \end{bmatrix} \quad X(k',l') = \begin{bmatrix} -5-j4 & 1+j6 & -1+j4 & -3-j2 \\ -7 & 1+j8 & 9 & 1-j8 \\ -5+j4 & -3+j2 & -1-j4 & 1-j6 \\ 29 & 1+j4 & -3 & 1-j4 \end{bmatrix}$$

For rotation other than about the center, translation operation can be used in addition.

Separable signals

The 2-D DFT is a separable function in the variables m and n. Therefore, the DFT of a separable function $x(m,n) = x(m)x(n)$ is also separable. The product of the column vector with the row vector is equal to the 2-D function. That is,

$$x(m) \leftrightarrow X(k), x(n) \leftrightarrow X(l) \rightarrow X(k,l) = X(k)X(l)$$

$$X(k,l) = \sum_{m=0}^{N-1}\sum_{n=0}^{N-1} x(m,n)e^{-j\frac{2\pi}{N}mk}e^{-j\frac{2\pi}{N}nl} = \sum_{m=0}^{N-1}\sum_{n=0}^{N-1} x(m)x(n)e^{-j\frac{2\pi}{N}mk}e^{-j\frac{2\pi}{N}nl}$$
$$= \left\{\sum_{m=0}^{N-1} x(m)e^{-j\frac{2\pi}{N}mk}\right\}\left\{\sum_{n=0}^{N-1} x(n)e^{-j\frac{2\pi}{N}nl}\right\} = X(k)X(l)$$

Example 3.9 Compute the DFT of $x(m) = \{1,1,1,1\}$ and $x(n) = \{1,1,1,1\}$. Using the separability theorem, verify that the product $x(m,n) = x(m)x(n)$ of the

column vector $x(m)$ and the row vector $x(n)$ in the time domain and the 2-D IDFT of the product of their individual DFTs are the same.

Solution
The product $x(m,n) = x(m)x(n)$ is

$$\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

$X(k) = \{4, 0, 0, 0\}$ and $X(l) = \{4, 0, 0, 0\}$.

$$X(k,l) = X(k)X(l) = \begin{bmatrix} 4 \\ 0 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} 4 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 16 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

The 2-D IDFT is

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} = x(m,n) = x(m)x(n)$$

∎

Parseval's theorem

This theorem implies that the signal power can also be computed from the DFT representation of the image. Let $x(m,n) \leftrightarrow X(k,l)$ with the dimensions of the image $N \times N$. Since the magnitude of the samples of the complex sinusoidal surface

$$e^{j\frac{2\pi}{N}(mk+nl)}, \quad m = 0, 1, \ldots, N-1, \quad n = 0, 1, \ldots, N-1$$

is one and the 2-D DFT coefficients are scaled by N^2, the power of a complex sinusoidal surface is

$$\left(\frac{|X(k,l)|^2}{N^4}\right) N^2 = \frac{|X(k,l)|^2}{N^2}$$

Therefore, the sum of the powers of all the components of an image yields the power of the image.

$$\sum_{m=0}^{N-1}\sum_{n=0}^{N-1}|x(m,n)|^2 = \frac{1}{N^2}\sum_{k=0}^{N-1}\sum_{l=0}^{N-1}|X(k,l)|^2$$

For the image in Example 3.3, the power computed in both the domains is 85.

The generalized form of this theorem holds for two different images as given by

$$\sum_{m=0}^{N-1}\sum_{n=0}^{N-1}x(m,n)y^*(m,n) = \frac{1}{N^2}\sum_{k=0}^{N-1}\sum_{l=0}^{N-1}X(k,l)Y^*(k,l)$$

3.6 The 1-D Fourier Transform

As always, the summation operation in discrete signal analysis corresponds to integration in continuous signal analysis. The FT is continuous in both the time and frequency domains and is the most general version of the Fourier analysis. The FT $X(j\omega)$ of $x(t)$ is defined as

$$X(j\omega) = \int_{-\infty}^{\infty} x(t)e^{-j\omega t}dt \tag{3.9}$$

A sufficient condition for the existence of $X(j\omega)$ is that $x(t)$ is absolutely integrable. The IFT $x(t)$ of $X(j\omega)$ is defined as

$$x(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} X(j\omega)e^{j\omega t}d\omega \tag{3.10}$$

The FT spectrum is composed of components of all frequencies $(-\infty < \omega < \infty)$. The amplitude of any component is $X(j\omega)\ d\omega/(2\pi)$, which is infinitesimal. The FT is a relative amplitude spectrum.

Example 3.10 Find the FT of the rectangular pulse $x(t) = u(t+p) - u(t-p)$, where $u(t)$ is the unit-step function.

Solution

$$X(j\omega) = \int_{-p}^{p} e^{-j\omega t}dt = 2\int_{0}^{p}\cos(\omega t)dt = \frac{2\sin(\omega p)}{\omega}$$

$$u(t+p) - u(t-p) \leftrightarrow \frac{2\sin(\omega p)}{\omega}$$

The pulse and its FT are shown, respectively, in Fig. 3.10a, b with $p = 0.2$. ∎

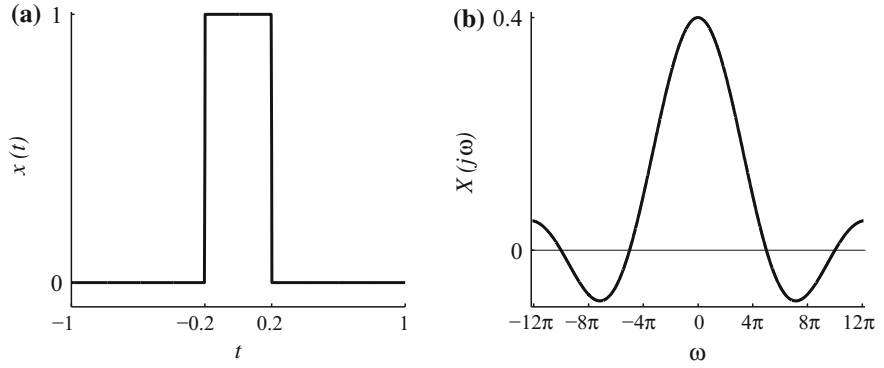


Fig. 3.10 **a** The pulse $x(t) = u(t + 0.2) - u(t - 0.2)$ and **b** its FT spectrum

3.7 The 2-D Fourier Transform

The 2-D FT is a straightforward extension of the 1-D FT. The FT $X(ju, jv)$ of $x(p, q)$ is defined as

$$X(ju, jv) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} x(p, q) e^{-jup} e^{-jvq} dp\, dq$$

The IFT is given by

$$x(p, q) = \frac{1}{4\pi^2} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} X(ju, jv) e^{jup} e^{jvq} du\, dv$$

Example 3.11 Find the 2-D FT of $x(p, q)$ analytically.

$$x(p, q) = \begin{cases} 1 & \text{for } 0 \le p < 2,\ 0 \le q < 3 \\ 0 & \text{elsewhere} \end{cases}$$

Let the record lengths be $P = 8$, $Q = 12$ seconds and the number of samples be 32 in both the directions. Approximate the spectrum of the signal using the DFT.
Solution
As the signal is separable, we use the 1-D FT results to get

$$X(ju, jv) = 4 \frac{\sin(u)}{u} \frac{\sin(1.5v)}{v} e^{-ju} e^{-j1.5v}, \quad u, v \neq 0$$

$$X(0, jv) = 4 \frac{\sin(1.5v)}{v} e^{-j1.5v}, \quad v \neq 0,$$

$$X(ju, 0) = 6 \frac{\sin(u)}{u} e^{-ju}, \quad u \neq 0, \quad X(0, 0) = 6$$

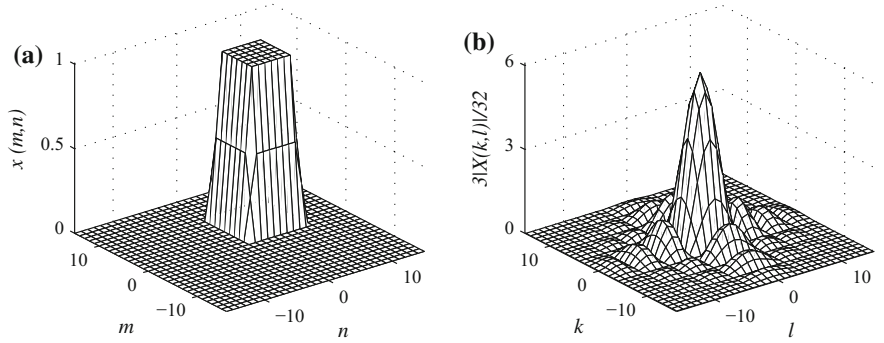


Fig. 3.11 Approximation of **a** the 2-D pulse $x(p, q) = u(p, q) - u(p - 2, q - 3)$ and **b** its FT magnitude spectrum using the DFT

Fig. 3.11a shows the signal with 32×32 samples in the center-zero format. The sample values along the border are set at 0.5 and those at the corners are set at 0.25, the average values are at the discontinuities. The given signal is defined in a 2×3 area. However, the signal appears square because we used a sampling interval of $\frac{8}{32} = \frac{1}{4}$ seconds in the p direction and $\frac{12}{32} = \frac{3}{8}$ seconds in the q direction. The scaled DFT magnitude spectrum is shown in Fig. 3.11b in the center-zero format. The scaling factor is the product of the sampling intervals, $(1/4)(3/8) = (3/32)$. The frequency increment in the k direction is $\frac{2\pi}{8}$ radians per second and it is $\frac{2\pi}{12}$ in the l direction.

3.8 Summary

- Transforms provide alternate representations of images and facilitates easier interpretation of the characteristics of images and faster execution of operations.
- Transforms represent an image as linear combinations of a set of basis functions.
- Fourier analysis is indispensable in many areas of science and engineering, including image processing.
- Fourier analysis represents an image as a linear combination of sinusoidal surfaces (sinusoidal functions with two frequencies).
- Fourier analysis provides the spectrum of an image and the spectrum is the basis for most of the analysis. The spectrum is an alternate representation of an image that represents an image in the frequency domain, frequency versus amplitude in contrast to spatial coordinates versus intensity in the spatial domain.
- The convolution operation, relating the input and output of a system, becomes much simpler multiplication operation in the frequency domain.
- Although the image is a 2-D signal, for the most part, we are able to use the 1-D transform and their algorithms due to the separability of the various filter and transform matrices.

- Properties of the transforms present the effect of the spatial domain characteristics and operations on images in the transform domain and vice versa.
- The DFT version of the Fourier analysis is most often used in practice due to its discrete and finite nature in both the spatial and frequency domains and, thereby, its amenability to implementation on digital computers.
- It is the availability of fast algorithms for its computation and the property of the sinusoidal waveforms to retain their shape from the input to the output of a linear system that makes the Fourier analysis so important for the analysis of images.
- The Fourier transform version of the Fourier analysis, along with the DFT, is most often used in image processing.

Exercises

3.1 The discrete periodic waveform $x(n)$ is periodic with period 4 samples. Express the waveform in terms of complex exponentials and, thereby, find its DFT coefficients $X(k)$. Find the 4 samples from both the expressions and check that they are the same. Find the least-squares errors, if $x(n)$ is represented by its DC component alone with the values $X(0)$, $0.9X(0)$, and $1.1X(0)$.

*(i)
$$x(n) = 1 + 3\cos\left(\frac{2\pi}{4}n + \frac{\pi}{3}\right) + 2\cos\left(2\frac{2\pi}{4}n\right)$$

(ii)
$$x(n) = -2 + \cos\left(\frac{2\pi}{4}n - \frac{\pi}{3}\right) + \cos\left(2\frac{2\pi}{4}n\right)$$

(iii)
$$x(n) = 2 + \cos\left(\frac{2\pi}{4}n + \frac{\pi}{6}\right) + \cos\left(2\frac{2\pi}{4}n\right)$$

3.2 Find the DFT of the 4 samples using the matrix form of the DFT definition. Reconstruct the input from the DFT coefficients using the IDFT and verify that they are the same as the input. Verify Parseval's theorem.

(i)
$$\{x(0) = 2, x(1) = 1, x(2) = 3, x(3) = 2\}$$

(ii)
$$\{x(0) = 1, x(1) = 1, x(2) = 2, x(3) = -3\}$$

(iii)
$$\{x(0) = -1, x(1) = 0, x(2) = -3, x(3) = 2\}$$

3.3 The discrete periodic image $x(m, n)$ is periodic with period 4 samples in both the directions. Express the image in terms of complex exponentials and, thereby, find its DFT coefficients $X(k, l)$. Find the 4×4 samples from both the expressions and check that they are the same. Find the least-squares errors, if $x(m, n)$ is represented by its DC component alone with the values $X(0, 0)$, $0.9X(0, 0)$, and $1.1X(0, 0)$.

*(i)

$$x(m, n) = 1 + 2\cos\left(\frac{2\pi}{4}(m + n) - \frac{\pi}{3}\right) + \cos\left(2\frac{2\pi}{4}(m + n)\right)$$

(ii)

$$x(m, n) = 2 + 2\cos\left(\frac{2\pi}{4}(m + 2n) + \frac{\pi}{3}\right) - \cos\left(2\frac{2\pi}{4}(m + n)\right)$$

(iii)

$$x(m, n) = -1 + 2\cos\left(\frac{2\pi}{4}(2m + n) - \frac{\pi}{6}\right) + 2\cos\left(2\frac{2\pi}{4}(m + n)\right)$$

3.4 Find the DFT of the image $x(m, n)$ using the row–column method. Reconstruct the input from the DFT coefficients using the IDFT and verify that they are the same as the input. Verify Parseval's theorem. Express the magnitude of the DFT coefficients in the center-zero format using the log scale, $\log_{10}(1 + |X(k, l)|)$. The origin is at the top-left corner.

*(i)

$$x(m, n) = \begin{bmatrix} 112 & 148 & 72 & 153 \\ 120 & 125 & 30 & 99 \\ 95 & 120 & 89 & 33 \\ 170 & 99 & 109 & 40 \end{bmatrix}$$

(ii)

$$x(m, n) = \begin{bmatrix} 143 & 107 & 183 & 102 \\ 135 & 130 & 225 & 156 \\ 160 & 135 & 166 & 222 \\ 85 & 156 & 146 & 215 \end{bmatrix}$$

(iii)

$$x(m, n) = \begin{bmatrix} 164 & 127 & 117 & 59 \\ 154 & 122 & 104 & 83 \\ 129 & 136 & 100 & 60 \\ 117 & 128 & 80 & 48 \end{bmatrix}$$

3.5 Find the DFT of the 4×4 impulse image $x(m, n)$ using (i) the row–column method and (ii) using the shift theorem.

(i)

$$x(m,n) = \begin{bmatrix} 0\,0\,0\,0 \\ 0\,0\,1\,0 \\ 0\,0\,0\,0 \\ 0\,0\,0\,0 \end{bmatrix}$$

(ii)

$$x(m,n) = \begin{bmatrix} 0\,0\,0\,0 \\ 0\,0\,0\,0 \\ 0\,0\,1\,0 \\ 0\,0\,0\,0 \end{bmatrix}$$

(iii)

$$x(m,n) = \begin{bmatrix} 0\,0\,0\,0 \\ 0\,0\,0\,0 \\ 0\,0\,0\,0 \\ 0\,0\,0\,1 \end{bmatrix}$$

3.6 Using the DFT and IDFT, find: (a) the periodic convolution of $x(m,n)$ and $h(m,n)$, (b) the periodic correlation of $x(m,n)$ and $h(m,n)$, and $h(m,n)$ and $x(m,n)$, (c) the autocorrelation of $x(m,n)$.

*(i)

$$x(m,n) = \begin{bmatrix} 2\,1\,3\,3 \\ 1\,0\,1\,2 \\ 4\,1\,0\,1 \\ 2\,0\,1\,2 \end{bmatrix} \quad h(m,n) = \begin{bmatrix} -2\,1 & 3 & 2 \\ 1\,1 & -1 & -2 \\ 4\,0 & 0 & -1 \\ 1\,0 & 2 & 2 \end{bmatrix}$$

(ii)

$$x(m,n) = \begin{bmatrix} 1\,2\,-2\,1 \\ -2\,0\,1\,4 \\ 1\,1\,-1\,2 \\ 0\,1\,2\,4 \end{bmatrix} \quad h(m,n) = \begin{bmatrix} 0\,-1 & 2 & 2 \\ -3 & 1 & 1\,-1 \\ 1 & 1\,-3 & 0 \\ 0 & 1 & 1 & 2 \end{bmatrix}$$

(iii)

$$x(m,n) = \begin{bmatrix} 2\,1\,3\,4 \\ -2\,0\,1\,4 \\ 1\,1\,3\,2 \\ 3\,1\,0\,4 \end{bmatrix} \quad h(m,n) = \begin{bmatrix} 3\,1 & 2\,4 \\ 0\,1\,-1\,2 \\ 1\,1\,-2\,2 \\ 0\,1 & 2\,1 \end{bmatrix}$$

3.7 Compute the DFT of the column vector $x(m) = \{1, 1, -1, -1\}$ and the row vector $x(n) = \{1, 1, -1, -1\}$. Using the separability theorem, verify that the product of the vectors in the time domain and the 2-D IDFT of the product of their individual DFTs are the same.

3.8 Compute the DFT of the column vector $x(m) = \{0.2741, 0.4519, 0.2741\}$ and the row vector $x(n) = \{0.2741, 0.4519, 0.2741\}$. Using the separability theorem, verify that the product of the vectors in the time domain and the 2-D IDFT of the product of their individual DFTs are the same.

3.9 Compute the DFT of the column vector $x(m) = \{0, 1, 0, -1\}$ and the row vector $x(n) = \{1, 0, -1, 0\}$. Using the separability theorem, verify that the product of the vectors in the time domain and the 2-D IDFT of the product of their individual DFTs are the same.

Chapter 4
Image Enhancement in the Frequency Domain

Abstract The frequency-domain representation of images is presented in the last chapter. Depending on the problem, either the spatial-domain or frequency-domain processing is advantageous. The interpretation of operations on images is often easier in the frequency domain. For longer filter lengths, frequency-domain processing provides faster processing. Linear filtering operations using a variety of filters are described in the frequency domain.

Most images occur in analog form in the spatial domain. Due to the advantages of digital processing, usually, they are digitized and processed. It turns out that still more representations are found to be advantageous in image processing. While there are several transforms to do the representation, the two most important representations are obtained by decomposing arbitrary images into a linear combination of impulses in the spatial domain and sinusoids in the frequency domain. The convolution operation, presented in Chap. 2, yields the processed image from the input and the system impulse response. It is based on decomposing images into a linear combination of impulses. Then, due to the linearity property of the linear systems, the output image is found with the knowledge of the response of the system to a single impulse signal.

The system input and output can also be related in a similar manner by decomposing an image into a linear combination of sinusoidal signals. System output can be obtained faster, and interpretation of image characteristics and operations is also easier. For example, the convolution operation is commutative. This property is obvious in the frequency-domain representation. While the impulse response characterizes a system in the spatial domain, it is the frequency response that does the job in the frequency domain. The representation of the image by its spectrum is all-important. The spectrum reveals the characteristics of an image and enables to decide the type of system required to process it. The conclusion is that certain properties of images and systems are easier to recognize in one of the domains. Similarly, one of the domains provides faster execution of certain operations. In this chapter, we study how the filters are designed and how to implement the convolution operation faster. Processing of images in the frequency domain consists of:

D. Sundararajan, *Digital Image Processing*, DOI 10.1007/978-981-10-6113-4_4

1. Transformation of the input image and the system response from the spatial domain to the frequency domain.
2. Processing the image in the frequency domain.
3. Transformation of the processed image back to the spatial domain.

Although it looks like a long route, processing of images in the frequency domain provides several advantages. While we have enhanced images in the spatial domain, enhancing images in the frequency domain reduces the execution time, particularly for relatively large filters. This is due to the fact that convolution in the spatial domain becomes much simpler multiplication operation in the frequency domain. Further, the interpretation of the various filtering operations is easier.

4.1 1-D Linear Convolution Using the DFT

Periodicity of the finite input data is assumed in DFT computation, since the basis functions are periodic. This implies that when carrying out operations such as convolution or approximating other versions of the Fourier analysis, the required output is represented in one period with adequate accuracy. The linear convolution of two sequences of length M and N yields a sequence of length $M + N - 1$. Therefore, both the input sequences must be appended by sufficient number of zeros to make the sequences of length $M + N - 1$, at the least. Due to the availability of practically efficient DFT algorithms only for sequence lengths of a power of 2, the sequences are zero-padded so that the length is greater than or equal to $M + N - 1$ and is a power of 2.

The linear convolution of the sequences $\{2, 3\}$ and $\{4, 1\}$ is $\{8, 14, 3\}$. The DFT of the sequences are $\{5, -1\}$ and $\{5, 3\}$, respectively. The pointwise product of the DFTs is $\{25, -3\}$, and the IDFT of the product is $\{11, 14\}$. This is the periodic convolution output. To get the linear convolution output, we pad the sequences with zeros to get $\{2, 3, 0, 0\}$ and $\{4, 1, 0, 0\}$. The DFT of the sequences are $\{5, 2 - j3, -1, 2 + j3\}$ and $\{5, 4 - j1, 3, 4 + j1\}$, respectively. The pointwise product of the DFTs is $\{25, 5 - j14, -3, 5 + j14\}$ and the IDFT of the product is $\{8, 14, 3, 0\}$. This is the linear convolution output. A sequence length of 3 is enough to solve this problem. We used a sequence length 4 so that the length is a power of 2.

The computational complexity of the 1-D convolution operation is $O(N^2)$ in the time domain, and it is $O(N \log_2 N)$ in the frequency domain. This is due to the convolution theorem and the fast DFT algorithms. A major reason for the widespread use of digital signal and image processing is due to the availability of fast algorithms for the computation of the DFT. It is understood that DFT is always computed using the fast algorithms. The principle behind the algorithm is simple, and it is presented in the Appendix.

4.2 2-D Linear Convolution Using the DFT

Let $xz(m,n)$ and $hz(m,n)$ are the zero-padded versions of the image $x(m,n)$ and the filter $h(m,n)$ to be linearly convolved. Let the 2-D DFT of $xz(m,n)$ be $X(k,l)$ and that of $hz(m,n)$ be $H(k,l)$. We find the pointwise product $Y(k,l) = X(k,l)H(k,l)$. The 2-D IDFT of $Y(k,l)$ yields the convolution output with some rows and columns with zero values at the end. The block diagram of the 2-D linear convolution using the 2-D DFT is shown in Fig. 4.1.

Two-dimensional filters, which can be decomposed into 1-D filters along the coordinate axes, are called separable filters. When the 2-D filter is separable, $H(k,l) = H(k)H(l)$, the 2-D convolution relation using the DFT is given by

$$Y(k,l) = X(k,l)H(k,l) = X(k,l)H(k)H(l)$$

The pointwise product of $X(k,l)H(k)$ can be first carried out and the result can be multiplied by $H(l)$. The order of computation can also be reversed. The block diagram of the 2-D convolution using the 2-D DFT with separable filters is shown in Fig. 4.2. It is more efficient to implement separable filters as shown in the diagram. Note that, apart from the filtering operation, the 2-D DFT and the IDFT are also computed using the row–column method with fast algorithms for 1-D DFT. While, basically, transformation, processing, and inverse transformation are the basic steps in frequency-domain processing, zero-padding of one or both of the images is also required before the transformation. Further, the origin of the image is usually at the

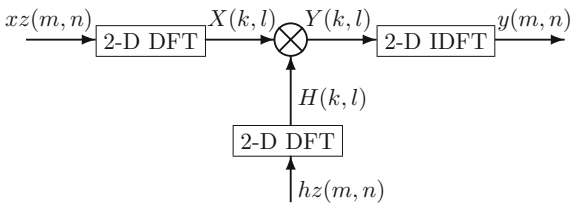


Fig. 4.1 2-D linear convolution using the 2-D DFT

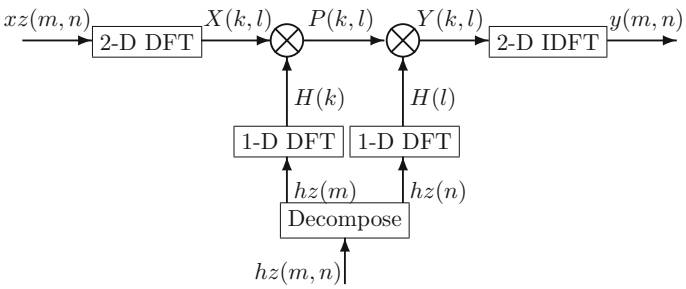


Fig. 4.2 2-D linear convolution using the 2-D DFT with separable filters

top-left corner, and the filters are usually specified in the center-zero format. Both must be represented in the same format. This requires changing the format of either the image or the filter. Summarizing,

1. The DFT assumes periodicity of the finite input data. It has to be ensured that the output is represented with adequate accuracy in one period.
2. To meet this constraint, sufficient zero-padding of the image and the filter is required. Further, the dimensions of one period have to be a power of 2 in order to use fast DFT algorithms.
3. It has to be ensured that both the image and the filter are in the same format with their origins aligned.

4.3 Lowpass Filtering

In this section, we present frequency-domain filtering using the averaging and Gaussian lowpass filters derived in the spatial domain in Chap. 2. Various border extensions are also taken into account.

4.3.1 The Averaging Lowpass Filter

Example 4.1 Convolve

$$x(m,n) = \begin{bmatrix} 1 & -1 & 3 & 2 \\ 2 & 1 & 2 & 4 \\ 1 & -1 & 2 & -2 \\ 3 & 1 & 2 & 2 \end{bmatrix} \quad \text{and} \quad h(m,n) = \begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix}$$

using the DFT. Assume zero-padding at the borders.

Solution
In the case of a 2-D signal, we have to zero-pads in two directions. Figure 4.3 shows the 8×8 image $xz(m,n)$, which is the zero-padded version of the 4×4 image $x(m,n)$. The 3×3 filter $h(m,n)$ is also shown. The convolution output is of size is 6×6. But we use matrices of size 8×8 so that the length is a power of 2 in both the directions.

As we have already seen, the averaging filter is separable.

$$h(m,n) = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & \mathbf{1} & 1 \\ 1 & 1 & 1 \end{bmatrix} = \frac{1}{3}\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \frac{1}{3}\begin{bmatrix} 1 & 1 & 1 \end{bmatrix} = h(m)h(n)$$

$xz(m,n)$

1	−1	3	2	0	0	0	0
2	1	2	4	0	0	0	0
1	−1	2	−2	0	0	0	0
3	1	2	2	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

$hz(m)$

1/3
1/3
0
0
0
0
0
1/3

$hz(n)$

1/3	1/3	0	0	0	0	0	1/3

$h(m,n)$

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Fig. 4.3 Zero-padding of images for implementing the 2-D linear convolution using the 2-D DFT

Therefore, the convolution can be carried out using two 1-D filters, $\{h(-1) = 1, h(0) = 1, h(1) = 1\}/3$. Since $xz(m,n)$ is of size 8×8, the zero-padded filter has to be of length 8 with $h(0)$ in the beginning. The filter can be written as

$$\{h(-1) = 1, h(0) = 1, h(1) = 1, h(2) = 0, h(3) = 0, h(4) = 0, h(5) = 0, h(6) = 0\}/3$$

Circularly shifting left by one position, we get the zero-padded column filter with the origin at the beginning.

$$hz(m) = \frac{1}{3}\{1, 1, 0, 0, 0, 0, 0, 1\}$$

Both the row and column filters, shown in Fig. 4.3, have the same coefficients. The 1-D DFT of this filter (division by 3 is deferred) is

$$H(k) = \{3, 2.4142, 1, -0.4142, -1, -0.4142, 1, 2.4142\}$$

The DFT is real-valued and even-symmetric, since the filter is also real-valued and even-symmetric. The DFT of the row filter $H(l)$ is the transpose of $H(k)$. The 2-D DFT, $X(k,l)$, of $xz(m,n)$ in Fig. 4.3 is shown in Table 4.1. The partial convolution output, $3P(k,l) = X(k,l)H(k)$ shown in Table 4.2, in the frequency domain is obtained by pointwise multiplication of each column of $X(k,l)$ by $H(k)$. The convolution output, $9Y(k,l) = 3P(k,l)H(l)$ shown in Table 4.3, in the frequency domain is obtained by pointwise multiplication of each row of $3P(k,l)$ by $H(l)$.

Since the factor $1/3$ was left out in the 1-D filters, the output, which is the same as that obtained in Chap. 2, is the 2-D IDFT of $9Y(k,l)$ divided by 9.

$$y(m,n) = \frac{1}{9}\begin{bmatrix} 3 & 8 & 11 & 11 & 6 & 0 & 0 & 3 \\ 3 & 10 & 10 & 11 & 4 & 0 & 0 & 4 \\ 7 & 13 & 11 & 10 & 4 & 0 & 0 & 6 \\ 4 & 8 & 4 & 4 & 0 & 0 & 0 & 4 \\ 4 & 6 & 5 & 4 & 2 & 0 & 0 & 3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 4 & 5 & 2 & 0 & 0 & 1 \end{bmatrix}$$

The top left 4×4 of $y(m,n)$ are the output values.

Table 4.1 The 2-D DFT, $X(k, l)$, of $xz(m, n)$ in Fig. 4.3

22.00+j0.00	2.76−j13.24	−2.00+j6.00	11.24+j4.76	10.00+j0.00	11.24−j4.76	−2.00−j6.00	2.76+j13.24
5.71−j12.02	−9.54−j7.95	−0.88+j4.71	6.36−j4.54	0.88−j6.71	0.46−j7.12	−4.54−j4.12	3.88+j1.46
5.00−j1.00	−4.24−j1.41	1.00+j5.00	1.41−j2.24	−3.00+j3.00	4.24+j1.41	−3.00−j3.00	−1.41+j6.24
4.29−j12.02	−6.36−j2.54	2.54−j0.12	−2.46−j1.95	5.12+j5.29	8.12−j8.54	−5.12−j3.29	7.54+j2.88
−12.00+j0.00	−1.59+j6.07	−4.00−j2.00	−4.41+j8.07	8.00+j0.00	−4.41−j8.07	−4.00+j2.00	−1.59−j6.07
4.29+j12.02	7.54−j2.88	−5.12+j3.29	8.12+j8.54	5.12−j5.29	−2.46+j1.95	2.54+j0.12	−6.36+j2.54
5.00+j1.00	−1.41−j6.24	−3.00+j3.00	4.24−j1.41	−3.00−j3.00	1.41+j2.24	1.00−j5.00	−4.24+j1.41
5.71+j12.02	3.88−j1.46	−4.54+j4.12	0.46+j7.12	0.88+j6.71	6.36+j4.54	−0.88−j4.71	−9.54+j7.95

Table 4.2 The partial convolution output $3P(k,l) = X(k,l)H(k)$ in the frequency domain

66.00+j0.00	8.27−j39.73	−6.00+j18.00	33.73+j14.27	30.00+j0.00	33.73−j14.27	−6.00−j18.00	8.27+j39.73
13.78−j29.02	−23.02−j19.19	−2.12+j11.36	15.36-j10.95	2.12−j16.19	1.12−j17.19	−10.95−j9.95	9.36+j3.54
5.00−j1.00	−4.24−j1.41	1.00+j5.00	1.41−j2.24	−3.00+j3.00	4.24+j1.41	−3.00−j3.00	−1.41+j6.24
−1.78+j4.98	2.64+j1.05	−1.05+j0.05	1.02+j0.81	−2.12−j2.19	−3.36+j3.54	2.12+j1.36	−3.12−j1.19
12.00+j0.00	1.59−j6.07	4.00+j2.00	4.41−j8.07	−8.00+j0.00	4.41+j8.07	4.00−j2.00	1.59+j6.07
−1.78−j4.98	−3.12+j1.19	2.12−j1.36	−3.36−j3.54	−2.12+j2.19	1.02−j0.81	−1.05−j0.05	2.64−j1.05
5.00+j1.00	−1.41−j6.24	−3.00+j3.00	4.24−j1.41	−3.00−j3.00	1.41+j2.24	1.00−j5.00	−4.24+j1.41
13.78+j29.02	9.36−j3.54	−10.95+j9.95	1.12+j17.19	2.12+j16.19	15.36+j10.95	−2.12−j11.36	−23.02+j19.19

Table 4.3 The convolution output $9Y(k,l) = 3P(k,l)H(l)$ in the frequency domain

198.00+j0.00	19.97−j95.91	−6.00+j18.00	−13.97−j5.91	−30.00+j0.00	−13.97+j5.91	−6.00−j18.00	19.97+j95.91
41.33−j87.06	−55.58−j46.33	−2.12+j11.36	−6.36+j4.54	−2.12+j16.19	−0.46+j7.12	−10.95−j9.95	22.61+j8.54
15.00−j3.00	−10.24−j3.41	1.00+j5.00	−0.59+j0.93	3.00−j3.00	−1.76−j0.59	−3.00−j3.00	−3.41+j15.07
−5.33+j14.94	6.36+j2.54	−1.05+j0.05	−0.42−j0.33	2.12+j2.19	1.39−j1.46	2.12+j1.36	−7.54−j2.88
36.00+j0.00	3.83−j14.66	4.00+j2.00	−1.83+j3.34	8.00+j0.00	−1.83−j3.34	4.00−j2.00	3.83+j14.66
−5.33−j14.94	−7.54+j2.88	2.12−j1.36	1.39+j1.46	2.12−j2.19	−0.42+j0.33	−1.05−j0.05	6.36−j2.54
15.00+j3.00	−3.41−j15.07	−3.00+j3.00	−1.76+j0.59	3.00+j3.00	−0.59−j0.93	1.00−j5.00	−10.24+j3.41
41.33+j87.06	22.61−j8.54	−10.95+j9.95	−0.46−j7.12	−2.12−j16.19	−6.36−j4.54	−2.12−j11.36	−55.58+j46.33

4.3.2 The Gaussian Lowpass Filter

Example 4.2 Convolve

$$x(m,n) = \begin{bmatrix} 1 & -1 & 3 & 2 \\ 2 & 1 & 2 & 4 \\ 1 & -1 & 2 & -2 \\ 3 & 1 & 2 & 2 \end{bmatrix}$$

and a 3×3 Gaussian lowpass filter with $\sigma = 0.5$. Assume periodicity at the borders.

Solution
This filter is also separable with the same coefficients in both the directions, $\{0.1065, 0.7870, 0.1065\}$, as shown in Chap. 2. Zero-padding and circularly shifting the column filter, we get

$$hz(m) = \{0.7870, 0.1065, 0, 0.1065\}$$

Only one zero is appended, since the convolution is periodic and the input is a 4×4 image. The 1-D DFT of this filter is

$$H(k) = \{1, 0.7870, 0.5740, 0.7870\}$$

Both the row and column filters have the same coefficients. The 2-D DFT, $X(k,l)$, of $x(m,n)$ is shown in Table 4.4. Since the periodic extension at the borders is assumed, no zero-padding of $x(m,n)$ is required. Note that these values are the same as the even-indexed values in the 8×8 zero-padded version of the input in the previous example, due to a DFT property. The partial convolution output, $P(k,l) = X(k,l)H(k)$ shown in Table 4.5, in the frequency domain is obtained by pointwise multiplication of each column of $X(k,l)$ by $H(k)$. The convolution output,

Table 4.4 The 2-D DFT, $X(k,l)$, of $x(m,n)$

22.00+j0.00	−2.00+j6.00	10.00+j0.00	−2.00−j6.00
5.00−j1.00	1.00+j5.00	−3.00+j3.00	−3.00−j3.00
−12.00+j0.00	−4.00−j2.00	8.00+j0.00	−4.00+j2.00
5.00+j1.00	−3.00+j3.00	−3.00−j3.00	1.00−j5.00

Table 4.5 The partial convolution output $P(k,l) = X(k,l)H(k)$ in the frequency domain

22.00+j0.00	−2.00+j6.00	10.00+j0.00	−2.00−j6.00
3.94−j0.79	0.79+j3.94	−2.36+j2.36	−2.36−j2.36
−6.89+j0.00	−2.30−j1.15	4.59+j0.00	−2.30+j1.15
3.94+j0.79	−2.36+j2.36	−2.36−j2.36	0.79−j3.94

Table 4.6 The convolution output $Y(k,l) = P(k,l)H(l)$ in the frequency domain

22.00+j0.00	−1.57+j4.72	5.74+j0.00	−1.57−j4.72
3.94−j0.79	0.62+j3.10	−1.36+j1.36	−1.86−j1.86
−6.89+j0.00	−1.81−j0.90	2.64+j0.00	−1.81+j0.90
3.94+j0.79	−1.86+j1.86	−1.36−j1.36	0.62−j3.10

$Y(k,l) = P(k,l)H(l)$ shown in Table 4.6, in the frequency domain is obtained by pointwise multiplication of each row of $P(k,l)$ by $H(l)$. The output, which is the same as that obtained in Chap. 2, is the 2-D IDFT of $Y(k,l)$.

$$y(m,n) = \begin{bmatrix} 1.2130 & -0.0144 & 2.3679 & 2.1790 \\ 1.8028 & 0.8664 & 2.0542 & 2.8921 \\ 0.8777 & -0.0982 & 1.4133 & -0.3823 \\ 2.2545 & 0.9502 & 1.8866 & 1.7372 \end{bmatrix}$$

Example 4.3 Convolve

$$x(m,n) = \begin{bmatrix} 1 & -1 & 3 & 2 \\ 2 & 1 & 2 & 4 \\ 1 & -1 & 2 & -2 \\ 3 & 1 & 2 & 2 \end{bmatrix}$$

and a 3×3 Gaussian lowpass filter with $\sigma = 0.5$. Assume replication at the borders.

Solution
The input with replication at the borders and zero-padded is

$$xz(m,n) = \begin{bmatrix} 1 & 1 & -1 & 3 & 2 & 2 & 0 & 0 \\ 1 & 1 & -1 & 3 & 2 & 2 & 0 & 0 \\ 2 & 2 & 1 & 2 & 4 & 4 & 0 & 0 \\ 1 & 1 & -1 & 2 & -2 & -2 & 0 & 0 \\ 3 & 3 & 1 & 2 & 2 & 2 & 0 & 0 \\ 3 & 3 & 1 & 2 & 2 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Zero-padding and circularly shifting the row filter, we get

$$hz(m) = \{0.7870, 0.1065, 0, 0, 0, 0, 0, 0.1065\}$$

The 1-D DFT of this filter is

$$H(k) = \{1, 0.9376, 0.7870, 0.6364, 0.5740, 0.6364, 0.7870, 0.9376\}$$

Table 4.7 The 2-D DFT, $X(k,l)$, of $xz(m,n)$

56.00+j0.00	−8.19−j10.61	21.00−j7.00	10.19−j10.61	−14.00+j0.00	10.19+j10.61	21.00+j7.00	−8.19+j10.61
−7.83−j10.76	−10.54+j11.54	0.24−j2.00	−4.54−j1.54	−3.00+j5.24	1.71−j3.54	−0.24−j8.00	−8.78−j1.29
6.00−j22.00	0.29+j4.71	−3.00−j7.00	3.71−j5.78	−4.00+j2.00	1.71+j3.29	9.00−j9.00	2.29+j9.78
−2.17+j19.24	2.54−j5.54	8.24+j8.00	−3.46−j4.46	−3.00+j3.24	6.78+j2.71	−8.24+j2.00	0.29−j3.54
16.00+j0.00	−8.54+j1.54	5.00−j7.00	−1.46+j5.54	2.00+j0.00	−1.46−j5.54	5.00+j7.00	−8.54−j1.54
−2.17−j19.24	0.29+j3.54	−8.24−j2.00	6.78−j2.71	−3.00−j3.24	−3.46+j4.46	8.24−j8.00	2.54+j5.54
6.00+j22.00	2.29−j9.78	9.00+j9.00	1.71−j3.29	−4.00−j2.00	3.71+j5.78	−3.00+j7.00	0.29−j4.71
−7.83+j10.76	−8.78+j1.29	−0.24+j8.00	1.71+j3.54	−3.00−j5.24	−4.54+j1.54	0.24+j2.00	−10.54−j11.54

Table 4.8 The partial convolution output $P(k,l) = X(k,l)H(k)$ in the frequency domain

56.00+j0.00	−8.19−j10.61	21.00−j7.00	10.19−j10.61	−14.00+j0.00	10.19+j10.61	21.00+j7.00	−8.19+j10.61
−7.34−j10.09	−9.88+j10.82	0.23−j1.88	−4.25−j1.44	−2.81+j4.92	1.60−j3.31	−0.23−j7.50	−8.23−j1.21
4.72−j17.31	0.23+j3.70	−2.36−j5.51	2.92−j4.55	−3.15+j1.57	1.34+j2.59	7.08−j7.08	1.80+j7.70
−1.38+j12.25	1.61−j3.52	5.25+j5.09	−2.20−j2.84	−1.91+j2.06	4.31+j1.72	−5.25+j1.27	0.19−j2.25
9.18+j0.00	−4.90+j0.88	2.87−j4.02	−0.84+j3.18	1.15+j0.00	−0.84−j3.18	2.87+j4.02	−4.90−j0.88
−1.38−j12.25	0.19+j2.25	−5.25−j1.27	4.31−j1.72	−1.91−j2.06	−2.20+j2.84	5.25−j5.09	1.61+j3.52
4.72+j17.31	1.80−j7.70	7.08+j7.08	1.34−j2.59	−3.15−j1.57	2.92+j4.55	−2.36+j5.51	0.23−j3.70
−7.34+j10.09	−8.23+j1.21	−0.23+j7.50	1.60+j3.31	−2.81−j4.92	−4.25+j1.44	0.23+j1.88	−9.88−j10.82

Table 4.9 The convolution output $Y(k,l) = P(k,l)H(l)$ in the frequency domain

56.00+j0.00	−7.68−j9.94	16.53−j5.51	6.49−j6.75	−8.04+j0.00	6.49+j6.75	16.53+j5.51	−7.68+j9.94
−7.34−j10.09	−9.26+j10.14	0.18−j1.48	−2.71−j0.92	−1.61+j2.82	1.02−j2.11	−0.18−j5.90	−7.72−j1.14
4.72−j17.31	0.22+j3.47	−1.86−j4.34	1.86−j2.89	−1.81+j0.90	0.85+j1.65	5.57−j5.57	1.69+j7.22
−1.38+j12.25	1.51−j3.30	4.13+j4.01	−1.40−j1.81	−1.10+j1.18	2.75+j1.10	−4.13+j1.00	0.17−j2.11
9.18+j0.00	−4.59+j0.83	2.26−j3.16	−0.53+j2.02	0.66+j0.00	−0.53−j2.02	2.26+j3.16	−4.59−j0.83
−1.38−j12.25	0.17+j2.11	−4.13−j1.00	2.75−j1.10	−1.10−j1.18	−1.40+j1.81	4.13−j4.01	1.51+j3.30
4.72+j17.31	1.69−j7.22	5.57+j5.57	0.85−j1.65	−1.81−j0.90	1.86+j2.89	−1.86+j4.34	0.22−j3.47
−7.34+j10.09	−7.72+j1.14	−0.18+j5.90	1.02+j2.11	−1.61−j2.82	−2.71+j0.92	0.18+j1.48	−9.26−j10.14

Table 4.10 The 2-D IDFT of $Y(k, l)$

0.7983	0.7032	−0.3226	2.2047	1.8822	1.5967	0.1903	0.0952
0.9887	**0.9048**	**−0.1934**	**2.4291**	**2.2855**	1.9773	0.2357	0.1178
1.5967	**1.6578**	**0.8664**	**2.0542**	**3.0371**	2.8127	0.3353	0.1903
1.1790	**1.1178**	**−0.0982**	**1.4133**	**−0.6224**	−0.8354	−0.0996	0.1405
2.4902	**2.5740**	**1.1292**	**1.8254**	**1.6194**	1.4064	0.1676	0.2968
2.3950	2.4902	1.1790	1.6918	1.7870	1.5967	0.1903	0.2855
0.2855	0.2968	0.1405	0.2017	0.2130	0.1903	0.0227	0.0340
0.0952	0.0838	−0.0384	0.2628	0.2243	0.1903	0.0227	0.0113

The 2-D DFT, $X(k,l)$, of $xz(m,n)$ is shown in Table 4.7. The partial convolution output, $P(k,l) = X(k,l)H(k)$ shown in Table 4.8, in the frequency domain is obtained by pointwise multiplication of each column of $X(k,l)$ by $H(k)$. The convolution output, $Y(k,l) = P(k,l)H(l)$ shown in Table 4.9, in the frequency domain is obtained by pointwise multiplication of each row of $P(k,l)$ by $H(l)$. The output is the 2-D IDFT of $Y(k,l)$. The 4×4 convolution output is shown in boldface in Table 4.10. The central part of the output is the same for both the border extensions, as it should be.

4.4 The Laplacian Filter

In this section, we present frequency-domain filtering using the Laplacian filter derived in the spatial domain in Chap. 2.

Example 4.4 Convolve

$$x(m,n) = \begin{bmatrix} 1 & -1 & 3 & 2 \\ 2 & 1 & 2 & 4 \\ 1 & -1 & 2 & -2 \\ 3 & 1 & 2 & 2 \end{bmatrix}$$

and a 3×3 Laplacian enhancement filter.

$$h(m,n) = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Assume zero-padding at the borders.

Solution
The Laplacian filter is inseparable. The zero-padding and shifting in two directions results in

$$hz(m,n) = \begin{bmatrix} 5 & -1 & 0 & 0 & 0 & 0 & 0 & -1 \\ -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Table 4.11 The 2-D DFT, $H(k,l)$, of zero-padded $hz(m,n)$

1.00	1.59	3.00	4.41	5.00	4.41	3.00	1.59
1.59	2.17	3.59	5.00	5.59	5.00	3.59	2.17
3.00	3.59	5.00	6.41	7.00	6.41	5.00	3.59
4.41	5.00	6.41	7.83	8.41	7.83	6.41	5.00
5.00	5.59	7.00	8.41	9.00	8.41	7.00	5.59
4.41	5.00	6.41	7.83	8.41	7.83	6.41	5.00
3.00	3.59	5.00	6.41	7.00	6.41	5.00	3.59
1.59	2.17	3.59	5.00	5.59	5.00	3.59	2.17

The 2-D DFT, $H(k,l)$, of $hz(m,n)$ is shown in Table 4.11. The DFT is real-valued and even-symmetric, since the filter is also real-valued and even-symmetric. The 2-D DFT of the zero-padded input $X(k,l)$ is the same as in Example 4.1. The convolution output $Y(k,l) = X(k,l)H(k,l)$ in the frequency domain is obtained by pointwise multiplication, shown in Table 4.12. The output, which is the same as that obtained in Chap. 2, is the top left 4×4 of the 8×8 2-D IDFT of $Y(k,l)$

$$y(m,n) = \begin{bmatrix} 4 & -10 & 12 & 3 \\ 7 & 3 & 0 & 18 \\ 1 & -10 & 9 & -18 \\ 13 & 1 & 5 & 10 \end{bmatrix}$$

4.4.1 Amplitude and Phase Distortions

The human eye is relatively more tolerant for amplitude distortion than phase distortion. The phase distortion results in shifting of the position of the pixels (smearing) and even a small amount of phase distortion may distort the image beyond recognition. Therefore, both the amplitude and phase characteristics of image processing systems, such as filters, must be considered carefully in processing the images. Figure 4.4a shows a 256×256 gray level image. The 2-D DFT of the image is computed in terms of its magnitude and phase spectra. Figure 4.4b shows its reconstruction using the square root of its magnitude spectrum and retaining the phase spectrum. The image is still identifiable. Figure 4.4c shows its reconstruction using the magnitude spectrum alone with zero phase. Distortion is quite severe. Figure 4.4d shows its reconstruction using a constant magnitude spectrum and retaining the phase spectrum. Although it is faint, the features of the original image are still intact.

Table 4.12 The convolution output $Y(k,l) = X(k,l)H(k,l)$ in the frequency domain

22.00+j0.00	4.37−j21.00	−6.00+j18.00	49.63+j21.00	50.00+j0.00	49.63−j21.00	−6.00−j18.00	4.37+j21.00
9.05−j19.06	−20.71−j17.26	−3.15+j16.88	31.82−j22.68	4.91−j37.46	2.32−j35.61	−16.26−j14.78	8.42+j3.18
15.00−j3.00	−15.21−j5.07	5.00+j25.00	9.07−j14.38	−21.00+j21.00	27.21+j9.07	−15.00−j15.00	−5.07+j22.38
18.95−j53.06	−31.82−j12.68	16.26−j0.78	−19.29−j15.26	43.09+j44.54	63.58−j66.82	−32.85−j21.12	37.68+j14.39
−60.00+j0.00	−8.86+j33.91	−28.00−j14.00	−37.14+j67.91	72.00+j0.00	−37.14−j67.91	−28.00+j14.00	−8.86−j33.91
18.95+j53.06	37.68−j14.39	−32.85+j21.12	63.58+j66.82	43.09−j44.54	−19.29+j15.26	16.26+j0.78	−31.82+j12.68
15.00+j3.00	−5.07−j22.38	−15.00+j15.00	27.21−j9.07	−21.00−j21.00	9.07+j14.38	5.00−j25.00	−15.21+j5.07
9.05+j19.06	8.42−j3.18	−16.26+j14.78	2.32+j35.61	4.91+j37.46	31.82+j22.68	−3.15−j16.88	−20.71+j17.26

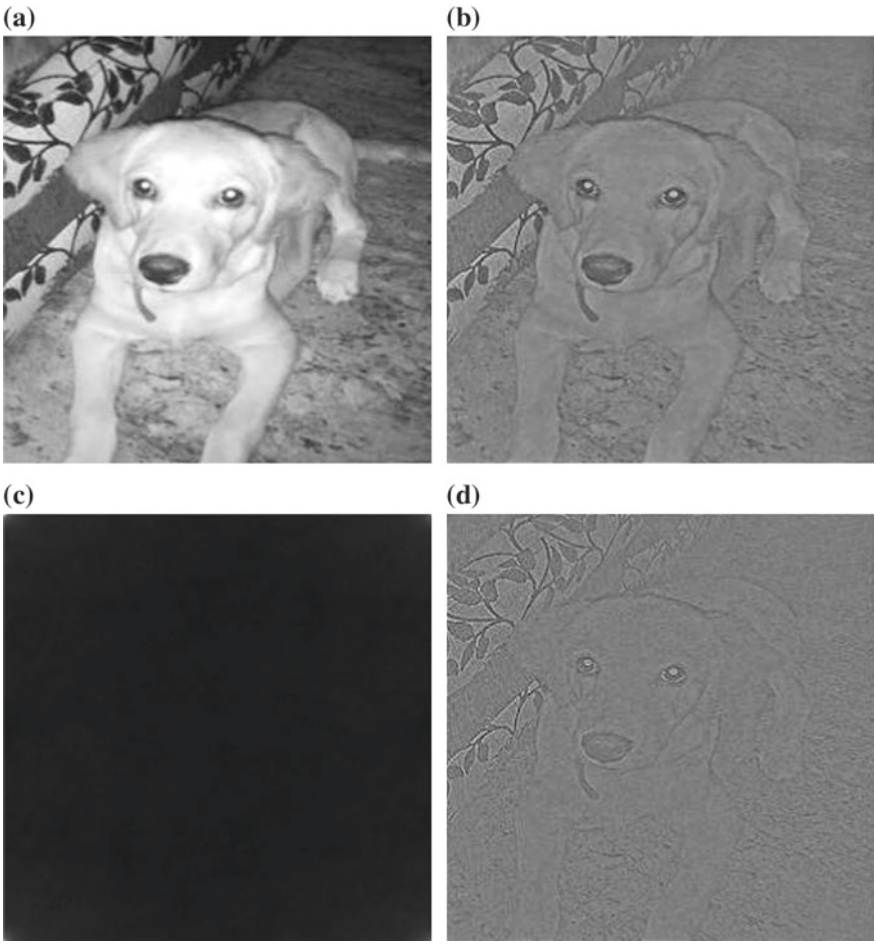


Fig. 4.4 **a** A 256×256 gray-level image; **b** its reconstruction using the square root of its magnitude spectrum and retaining the phase spectrum; **c** its reconstruction using the magnitude spectrum alone with zero phase; **d** its reconstruction using a constant magnitude spectrum and retaining the phase spectrum

4.5 Frequency-Domain Filters

4.5.1 Ideal Filters

As the frequency response is the Fourier transform of the impulse response, we found the frequency-domain versions of the averaging, Laplacian and Gaussian filters by computing the DFT of their impulse responses. Alternatively, filters can be directly

specified in the frequency domain itself. Consider the 1-D periodic waveform, presented in Chap. 3.

$$x(n) = 1 + 2\cos(\frac{2\pi}{4}n - \frac{\pi}{3}) + \cos(2\frac{2\pi}{4}n)$$

Its samples are

$$\{x(0) = 3, x(1) = \sqrt{3}, x(2) = 1, x(3) = -\sqrt{3}\}$$

There are three real frequency components. Its Fourier representation is

$$x(n) = \frac{1}{4}(4e^{j0\frac{2\pi}{4}n} + (2 - j2\sqrt{3})e^{j\frac{2\pi}{4}n} + 4e^{j2\frac{2\pi}{4}n} + (2 + j2\sqrt{3})e^{j3\frac{2\pi}{4}n})$$

The high-frequency components can be eliminated by making their DFT coefficients zero. Then, we get the lowpass filtered version of the waveform

$$x_l(n) = \frac{1}{4}(4e^{j0\frac{2\pi}{4}n}) = \{1, 1, 1, 1\}$$

The low-frequency components can be eliminated by making their DFT coefficients zero. Then, we get the highpass filtered version of the waveform

$$x_h(n) = \frac{1}{4}(4e^{j2\frac{2\pi}{4}n}) = \cos(\pi n) = \{1, -1, 1, -1\}$$

The low- and high-frequency components can be eliminated by making their DFT coefficients zero. Then, we get the bandpass filtered version of the waveform

$$x_{bp}(n) = \frac{1}{4}((2 - j2\sqrt{3})e^{j\frac{2\pi}{4}n} + (2 + j2\sqrt{3})e^{j3\frac{2\pi}{4}n}) = 2\cos(\frac{2\pi}{4}n - \frac{\pi}{3}) = \{1, \sqrt{3}, -1, -\sqrt{3}\}$$

The middle-frequency component can be eliminated by making its DFT coefficients zero. Then, we get the bandreject filtered version of the waveform

$$x_{br}(n) = \frac{1}{4}(4e^{j0\frac{2\pi}{4}n} + 4e^{j2\frac{2\pi}{4}n}) = (1 + \cos(\pi n)) = \{2, 0, 2, 0\}$$

Given the DFT coefficients of the waveform in the center-zero format,

$$\{X(-2) = 4, X(-1) = 2 + j2\sqrt{3}, X(0) = 4, X(1) = 2 - j2\sqrt{3}\}$$

we multiplied the set of coefficients, respectively, by the frequency responses

$$H_l(k) = \{0, 0, 1, 0\}, \quad H_h(k) = \{1, 0, 0, 0\}, \quad H_{bp}(k) = \{0, 1, 0, 1\}, \quad H_{br}(k) = \{1, 0, 1, 0\}$$

to implement the different filters. For each of the complex spectral component, the filtering operation is specified. As the complex coefficients are multiplied by 1 or 0, the phase of the filtered image is not distorted. They are called zero-phase filters. The lowpass filter passes the frequency components those are close to the zero frequency, while the highpass filter just does the opposite. The frequency response of the highpass filter is related to that of the lowpass filter. The frequency responses of the other two types of filters can be expressed as a linear combination of those of the lowpass and highpass filters.

4.5.1.1 Lowpass Filter

Let the DFT of the image be in the center-zero format. That is, the DC coefficient is located approximately in the center of the spectrum and the distance of a coefficient from the center indicates its frequency. Since the low-frequency coefficients are around the center, we can define a lowpass filter in the frequency domain as

$$H(k,l) = \begin{cases} 1, \text{ for } D(k,l) \leq D_c \\ 0, \text{ for } D(k,l) > D_c \end{cases}$$

where $D(k,l) = \sqrt{k^2 + l^2}$ is the distance between the spectral point (k,l) and the center of the spectrum, and D_c is the cutoff radius. This spectrum $H(k,l)$ of the filter is multiplied by the spectrum of the image $X(k,l)$ to filter the image. The frequency components in the image, which are located inside a circle of radius D_c, are passed (as they are multiplied by 1) and the rest are cut off (as they are multiplied by 0) to produce the filtered image. A 4×4 distance matrix with the center at coordinates $(2,2)$ is

$$D(k,l) = \begin{bmatrix} 2.8284 & 2.2361 & 2.0000 & 2.2361 \\ 2.2361 & 1.4142 & 1.0000 & 1.4142 \\ 2.0000 & 1.0000 & 0 & 1.0000 \\ 2.2361 & 1.4142 & 1.0000 & 1.4142 \end{bmatrix}$$

If we specify that the cutoff radius is 1.9, then the lowpass filter spectrum is

$$H(k,l) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}$$

If we specify that the cutoff radius is 0.9, then only the DC component of the image will be passed.

The ideal filters are ideal in defining the frequency response. However, they are not useful in practice since the sharp cutoff produces the ringing effect. The unit-step response is of ripply character, which is often undesirable. These filters are used to classify the various types of filters and serve as a standard for practical filters to

attain. They have a passband and a stopband, but no transition band. For practical filters, a transition band is a necessity and they approximate the frequency response of the ideal filters to a required degree.

4.5.1.2 Highpass Filter

Highpass filters pass the high-frequency components of a signal readily while suppressing the low-frequency components. It is used in image sharpening and edge detection. A highpass filter in the frequency domain is defined as

$$H(k,l) = \begin{cases} 0, \text{ for } D(k,l) \leq D_c \\ 1, \text{ for } D(k,l) > D_c \end{cases}$$

where $D(k,l) = \sqrt{k^2 + l^2}$ is the distance between the spectral point (k,l) and the center of the spectrum, and D_c is the cutoff radius. This spectrum $H(k,l)$ of the filter is multiplied by the spectrum of the image $X(k,l)$ to filter the image. The frequency components in the image, which are located outside a circle of radius D_c, are passed (as they are multiplied by 1) and the rest are cut off (as they are multiplied by 0) to produce the filtered image. A highpass filter is also defined, in terms of the spectrum of the lowpass filter, as

$$H_h(k,l) = 1 - H_l(k,l)$$

where $H_h(k,l)$ and $H_l(k,l)$ are, respectively, the spectra of highpass and lowpass filters. From the example given for the ideal lowpass filter, a highpass filter is given by

$$H_h(k,l) = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} - \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

4.5.2 The Butterworth Lowpass Filter

The practical filters have a transition band and reduce the ringing effect. Various functions define the filters. The spectrum of the lowpass Butterworth filter is defined as

$$H(k,l) = \frac{1}{1 + \left(\frac{D(k,l)}{D_c}\right)^{2n}}$$

where n is the order of the filter. The higher the order, the closer the spectrum becomes to that of the ideal filter. Therefore, the transition band width is controllable

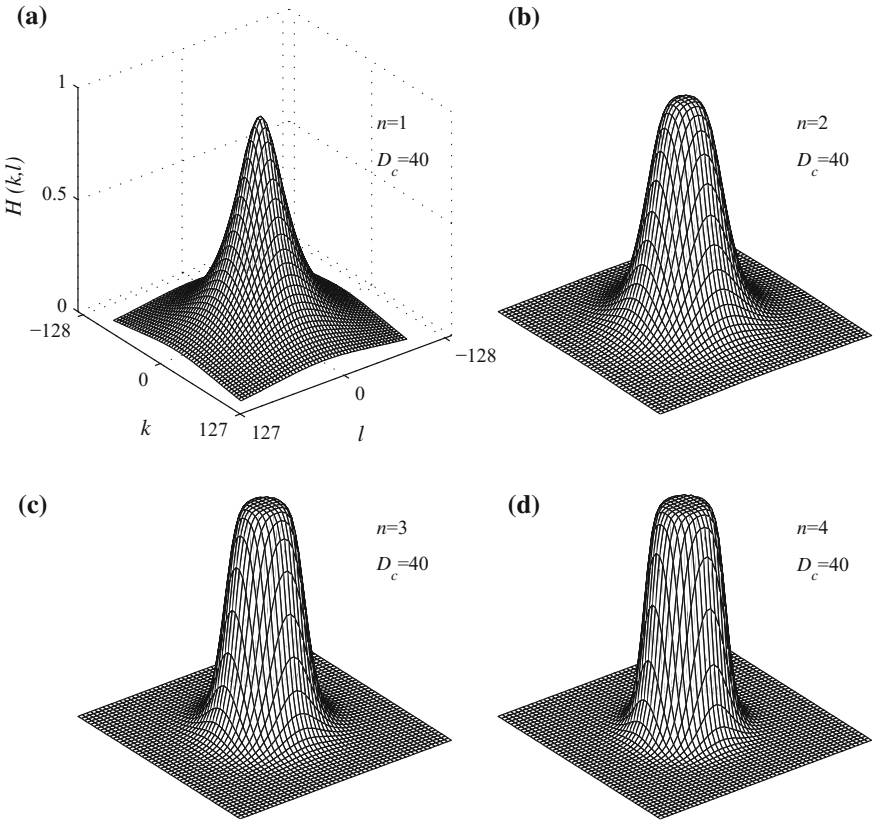


Fig. 4.5 The frequency response of the 256×256 lowpass Butterworth filters with $D_c = 40$. The amplitude of the response varies from 0 to 1. **a** Order $n = 1$; **b** order $n = 2$; **c** order $n = 3$; **d** order $n = 4$

by the order n. At low orders, the Butterworth filter spectrum is similar to that of the Gaussian filter. The attenuation is 0.5 at $D(k, l) = D_c$, for any order. Figure 4.5a–d shows the frequency responses of the 256×256 Butterworth lowpass filters with the cutoff frequency $D_c = 40$. The order of the filters, respectively, are $n = 1$, $n = 2$, $n = 3$, and $n = 4$. The amplitude of the response varies from 0 to 1. Figure 4.6a shows a 256×256 gray-level image. Its magnitude spectrum, in log scale, is shown in Fig. 4.6b. The image representation of a Butterworth lowpass filter with $D_c = 10$, $n = 3$ and the corresponding filtered image are shown, respectively, in Fig. 4.6c and d. The image representation of a Butterworth lowpass filter with $D_c = 60$, $n = 3$, and the corresponding filtered image are shown, respectively, in Fig. 4.6e and f. As the cutoff frequency is small, the filter attenuates most of the frequency components and the image is considerably blurred in (d). As the cutoff frequency is high, the filter passes most of the frequency components, and the image almost looks like the original in (f). Figure 4.7 shows the spectra of the image and its modified versions

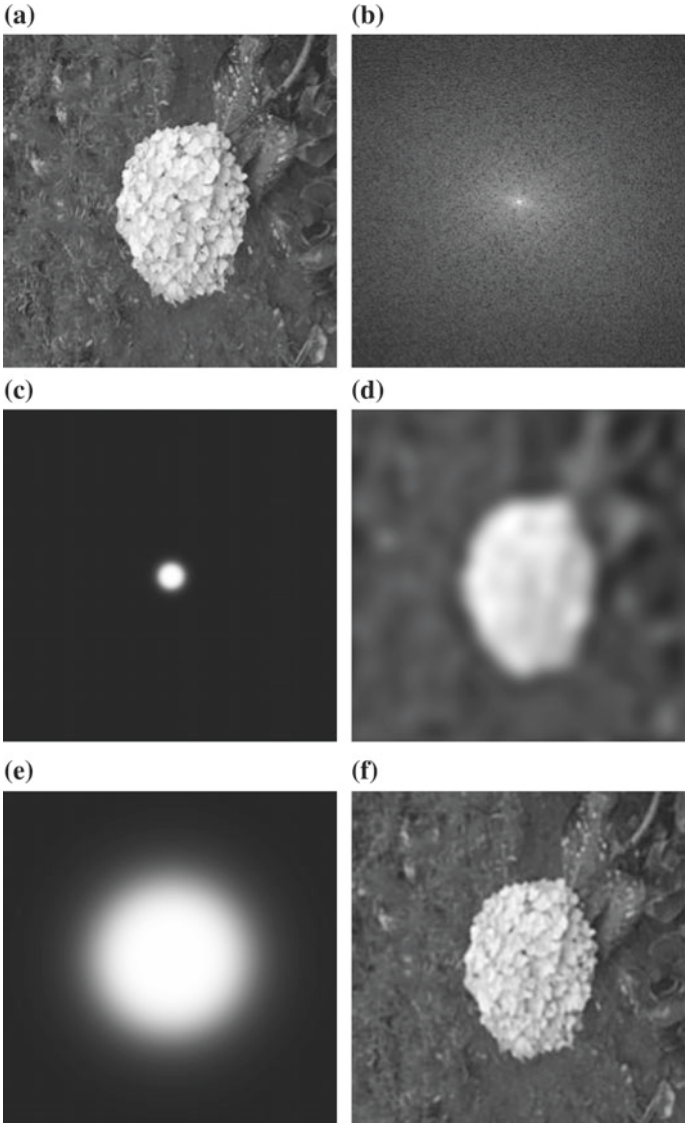


Fig. 4.6 **a** A 256 × 256 8-bit gray-level image; **b** its magnitude spectrum in log scale, $\log_{10}(1 + |X(k, l)|)$ in the center-zero format; **c** the image representation of a Butterworth lowpass filter with $D_c = 10$, $n = 3$ and **d** the corresponding filtered image; **e** a Butterworth lowpass filter with $D_c = 60$, $n = 3$ and **f** the corresponding filtered image

due to lowpass filtering. The filtering process is quite clear. As the cutoff frequency decreases from 60 in (d) to 30 in (c) and to 10 in (b), more and more frequency components are cutoff compared with the spectrum of the input image shown in (a). Therefore, the blurring of the image increases.

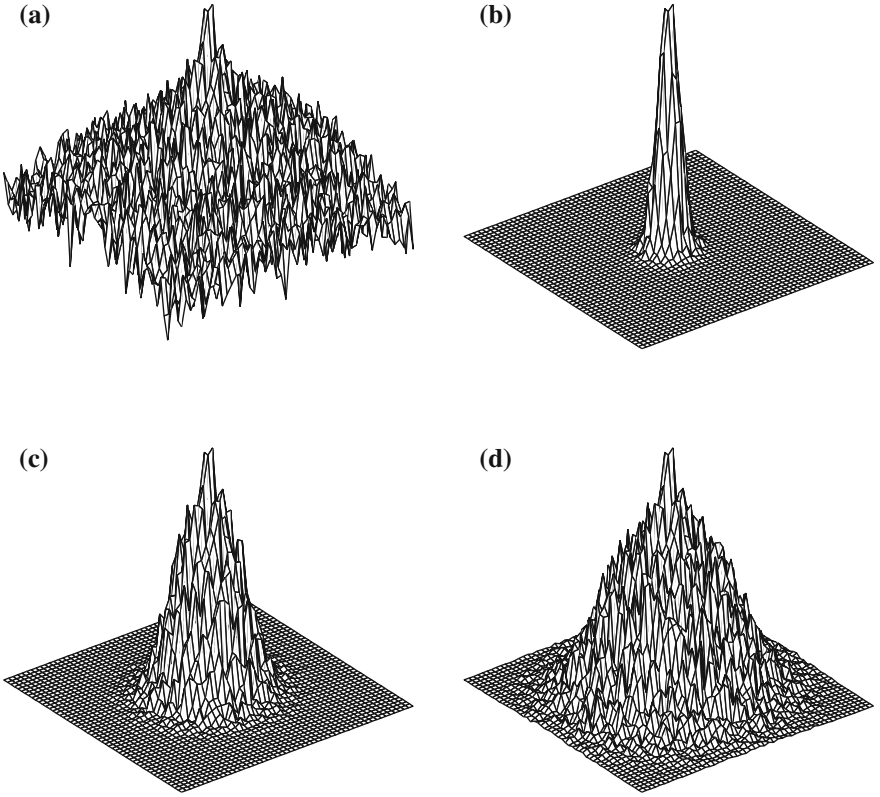


Fig. 4.7 **a** The magnitude spectrum of the image in Fig. 4.6a in a mesh plot; **b** the spectrum of the filtered image in Fig. 4.6d; **c** the spectrum of the filtered image with $D_c = 30$; **d** the spectrum of the filtered image in Fig. 4.6f

4.5.3 The Butterworth Highpass Filter

The spectrum of the Butterworth highpass filter is defined as

$$H(k,l) = \frac{1}{1 + \left(\frac{D_c}{D(k,l)}\right)^{2n}}$$

where D_c is the cutoff frequency and n is the order of the filter. The frequency-domain highpass filter with the same cutoff frequency can also be obtained using the relation

$$H_h(k,l) = 1 - H_l(k,l)$$

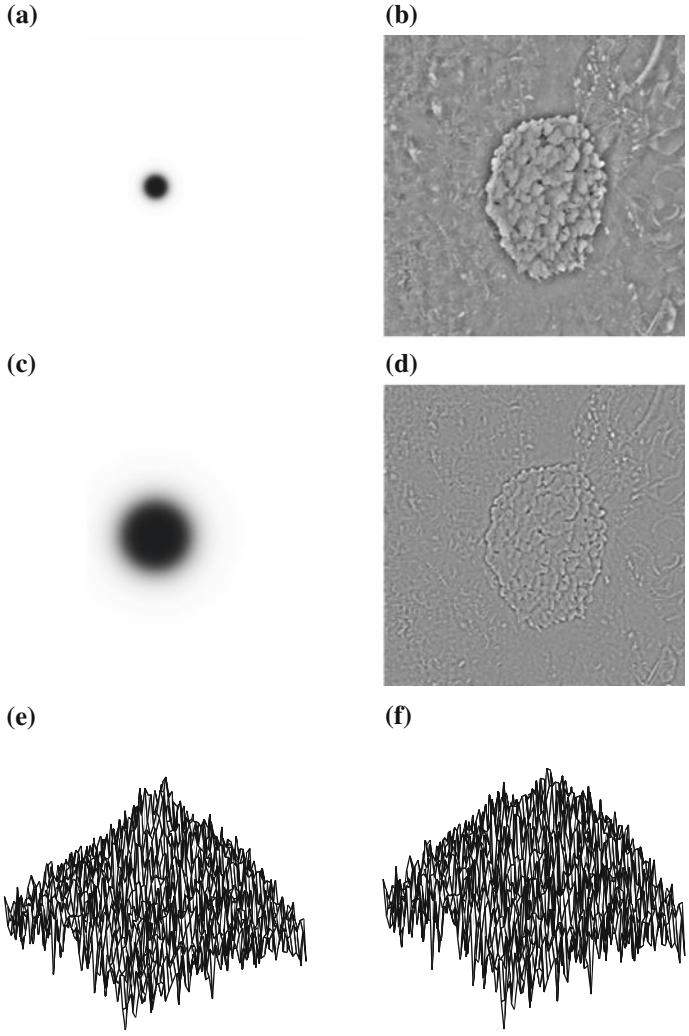


Fig. 4.8 **a** A butterworth highpass filter with $D_c = 10$, $n = 3$ and **b** the filtered image; **c** a butterworth highpass filter with $D_c = 20$, $n = 3$ and **d** the filtered image; **e** the spectrum of the image in (**b**); **f** the spectrum of the image in (**d**)

The $N \times N$ spectrum of the lowpass filter is subtracted from that of an all pass filter (a $N \times N$ matrix of all 1s). The filtering actions of lowpass and highpass filters are reciprocal. That is, the frequency components passed by the lowpass filter are attenuated by the corresponding highpass filter and vice versa.

Figure 4.8a and b shows, respectively, a Butterworth highpass filter with $D_c = 10$, $n = 3$, and the filtered image. The input image and its spectrum are shown, respectively, in Figs. 4.6a and 4.7a. Figure 4.8c and d shows, respectively, a Butterworth

highpass filter with $D_c = 20, n = 3$, and the filtered image. Figure 4.8e and f shows, respectively, the spectra of the highpass filtered images in Fig. 4.8b and d. The higher the cutoff frequency, the more is the attenuation of the low-frequency components.

4.5.4 The Gaussian Lowpass Filter

Both the impulse and frequency responses of a Gaussian filter are real-valued Gaussian functions. The spectrum of the lowpass Gaussian filter is defined as

$$H(k,l) = e^{-\frac{D^2(k,l)}{2D_c^2}}$$

The attenuation is $e^{-0.5} = 0.6065$ at $D(k,l) = D_c = \sigma$. Lower values of σ move the cutoff frequency toward zero. Figure 4.9a–d shows the frequency response $H(k,l)$ of the Gaussian 256×256 lowpass filter with $\sigma = 10$, $\sigma = 20$, $\sigma = 40$ and $\sigma = 80$, respectively.

4.5.5 The Gaussian Highpass Filter

The Gaussian highpass filter is defined, in terms of the spectrum of that of its lowpass filter, as

$$H_h(k,l) = 1 - H_l(k,l) = 1 - e^{-\frac{D^2(k,l)}{2D_c^2}}$$

where $H_h(k,l)$ and $H_l(k,l)$ are, respectively, the spectra of highpass and lowpass filters. Figure 4.10a shows a Gaussian highpass filter with $\sigma = 20$. The corresponding highpass filtered version, of the image in Fig. 4.6a, is shown in Fig. 4.10b. A Gaussian highpass filter with $\sigma = 50$ is shown in Fig. 4.10c. The corresponding filtered image is shown in Fig. 4.10d. As the cutoff frequency increases, more low-frequency components are attenuated.

4.5.6 Bandpass and Bandreject Filtering

Bandpass filters pass a band of frequencies in the middle of the spectrum of a signal. Bandpass filters can be realized using two lowpass filters as

$$H_{bp}(k,l) = Hh_c(k,l) - Hl_c(k,l)$$

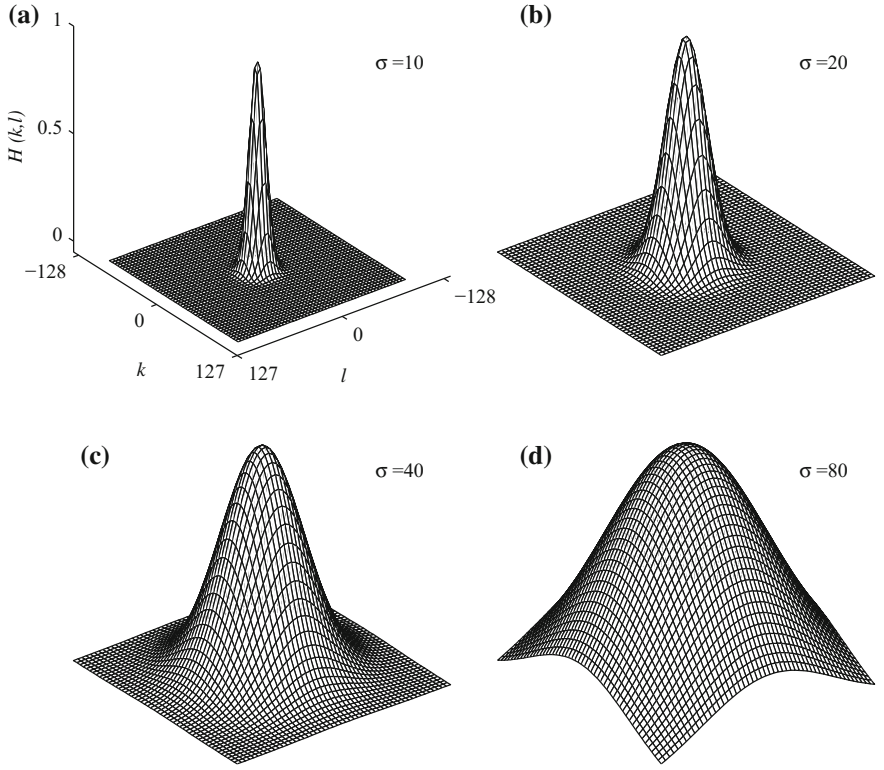


Fig. 4.9 The frequency response $H(k, l)$ of the Gaussian 256 × 256 lowpass filter with **a** $\sigma = 10$, **b** $\sigma = 20$, **c** $\sigma = 40$, **d** $\sigma = 80$

with filter $Hh_c(k, l)$ having a higher cutoff frequency. Gaussian bandpass filter:

$$H_{bp}(k, l) = e^{-\frac{D^2(k,l)}{2Dh_c^2}} - e^{-\frac{D^2(k,l)}{2Dl_c^2}}$$

Butterworth bandpass filter:

$$H_{bp}(k, l) = \frac{1}{1 + \left(\frac{D(k,l)}{Dh_c}\right)^{2n}} - \frac{1}{1 + \left(\frac{D(k,l)}{Dl_c}\right)^{2n}}$$

Bandreject filter:

$$H_{br}(k, l) = 1 - H_{bp}(k, l)$$

Examples of filtering with these filters are given in Chap. 5.

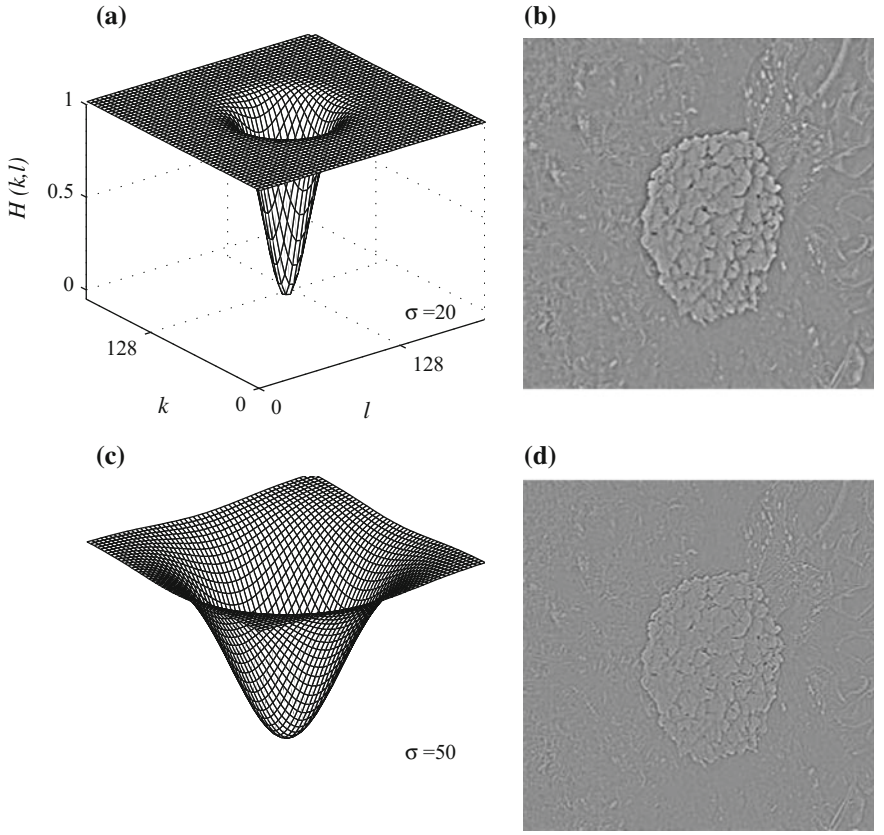


Fig. 4.10 **a** A Gaussian highpass filter with $\sigma = 20$ and **b** the filtered image; **c** a Gaussian highpass filter with $\sigma = 50$ and **d** the filtered image

4.6 Homomorphic Filtering

The intensity of light decreases with the inverse of the square of the distance from the source of the illumination. When the source is directed and strong, the image is corrupted due to illumination interference. Homomorphic filtering can enhance an image that is corrupted by multiplicative noise or interference. Let the corrupted and uncorrupted images be $y(m, n)$ and $x(m, n)$, respectively. Let the illumination interference be $z(m, n)$. Then,

$$y(m, n) = x(m, n)z(m, n)$$

It is assumed that $z(m, n)$ is constant. An image is modeled as the product of the illumination and reflectance components.

As the filtering is a linear process, we need the noise to be additive. Taking the natural logarithm, we get

$$\log_e(y(m,n)) = \log_e(x(m,n)) + \log_e(z(m,n))$$

Now, linear filtering can be applied and the exponential of the filtered output is the restored image. The low-frequency components dominate the illumination component, while the high-frequency components dominate the reflectance component. A homomorphic filter is similar to a highpass filter in that it passes high-frequency components more readily than the low-frequency components. Therefore, the effect of variable illumination on the image is reduced. The logarithmic version of the input image can be subjected to linear filtering yielding the logarithmic version of the filtered output image. Exponentiation, which is the inverse of logarithm, of this output is the filtered image. Figure 4.11a shows a 256 × 256 8-bit gray-level image.

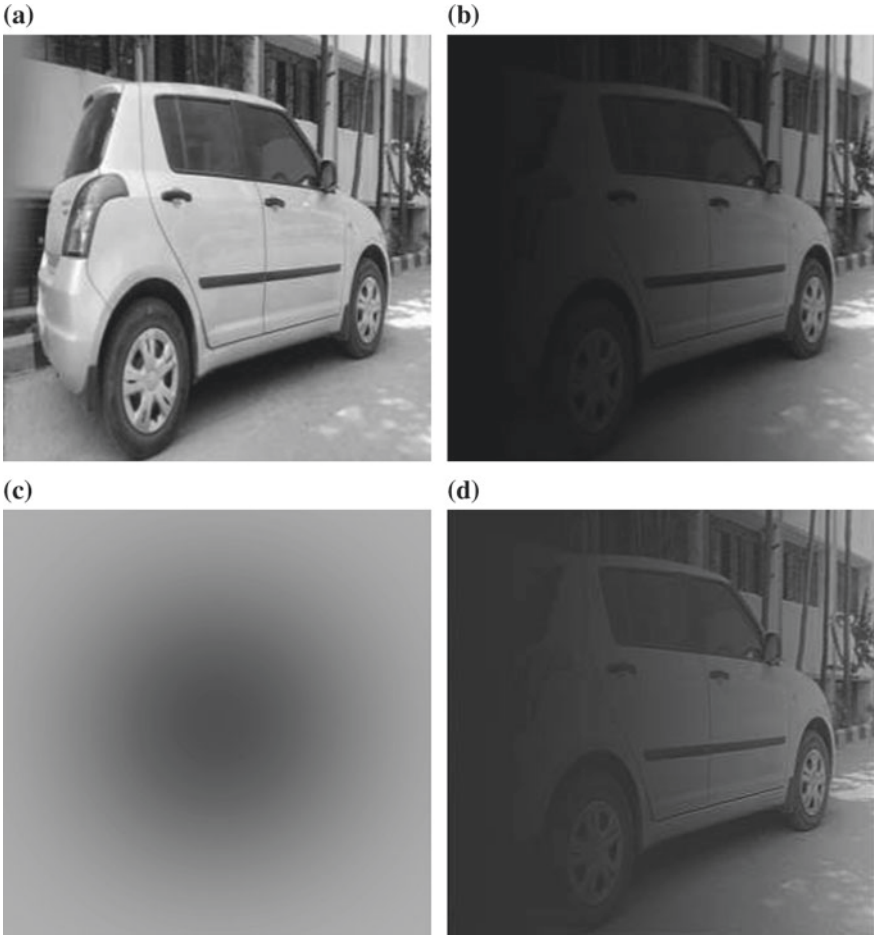


Fig. 4.11 **a** A 256 × 256 8-bit gray-level image; **b** the image with illumination interference; **c** homomorphic filter; **d** the image after homomorphic filtering

Figure 4.11b shows the image with illumination interference. The homomorphic filter is shown in Fig. 4.11c. The restored image after homomorphic filtering is shown in Fig. 4.11d

4.7 Summary

- In the frequency domain, images are expressed as a linear combination of their constituent sinusoidal surfaces. In this form, convolution becomes multiplication operation.
- The spectrum (the display of the amplitude of the frequency components versus frequency) of an image depicts the spectral characteristics of the image and enables the design of suitable image processing systems to process it.
- Filtering of an image is the alteration of its spectrum in a desired way. Suppressing the high-frequency components of the image is lowpass filtering. Suppressing the low-frequency components of the image is highpass filtering. Bandpass and bandreject filters are a combination of lowpass and highpass filters.
- Using the frequency domain, filtering is achieved by finding the spectra of the image and the filter, multiplying them, and finding the inverse transform.
- In using the DFT for implementing the linear convolution, it must be remembered that the output is contained in one period of the DFT.
- As periodicity is assumed in the DFT computation, zero-padding is required to implement the linear convolution in the frequency domain.
- Both the amplitude and phase distortion of the filters are important factors in processing images.
- Often, the 2-D filters are decomposable into 1-D filters resulting in faster execution of the filtering operation.
- Typical filters are averaging, Gaussian, Butterworth, and Laplacian.
- Filtering effect increases with the size of the filter. The size is an important factor in determining to implement the filter in the spatial domain or frequency domain.
- Filters can be specified in the frequency domain itself.
- In homomorphic filtering, the logarithm of the image is taken, and the antilogarithm of the convolution of the logarithmic version of the image and the filter is the output image. This type of filtering is effective to reduce multiplicative noise.

Exercises

4.1 Compute the linear convolution of $x(n)$, $n = 0, 1, \ldots$, and $h(n)$, $n = 0, 1, \ldots$, using the DFT and IDFT and verify your answer by directly computing the convolution. Assume zero-padding at the ends.
(i) $x(n) = \{2, 1, 3\}$ and $h(n) = \{1, -2\}$.
(ii) $x(n) = \{-1, 3\}$ and $h(n) = \{1, 3, -2\}$.

*(iii) $x(n) = \{4, -1\}$ and $h(n) = \{-3, 1, -2\}$.
(iv) $x(n) = \{-1, 2, 3\}$ and $h(n) = \{-2, 3\}$.
(v) $x(n) = \{2, 4\}$ and $h(n) = \{4, 3, -2\}$.

4.2 Compute the linear convolution of $x(m, n),\ m, n = 0, 1, 2$, and $h(m, n),\ m, n = 0, 1$ using the DFT and IDFT and verify your answer by directly computing the convolution. Assume zero-padding at the borders.
*(i)

$$x(m,n) = \begin{bmatrix} 1 & 2 & 3 \\ 2 & -3 & 1 \\ 1 & 2 & 1 \end{bmatrix} \quad \text{and} \quad h(m,n) = \begin{bmatrix} 2 & -1 \\ 1 & 2 \end{bmatrix}$$

(ii)

$$x(m,n) = \begin{bmatrix} 2 & 2 & 3 \\ 2 & -3 & -1 \\ 1 & -2 & 1 \end{bmatrix} \quad \text{and} \quad h(m,n) = \begin{bmatrix} -2 & -1 \\ 1 & 3 \end{bmatrix}$$

(iii)

$$x(m,n) = \begin{bmatrix} 1 & -4 & 3 \\ -2 & 3 & 1 \\ 1 & -3 & 1 \end{bmatrix} \quad \text{and} \quad h(m,n) = \begin{bmatrix} 4 & 1 \\ -1 & 2 \end{bmatrix}$$

(iv)

$$x(m,n) = \begin{bmatrix} 2 & 1 & 4 \\ 1 & -3 & 2 \\ 3 & 2 & 1 \end{bmatrix} \quad \text{and} \quad h(m,n) = \begin{bmatrix} 1 & -3 \\ 2 & 2 \end{bmatrix}$$

(v)

$$x(m,n) = \begin{bmatrix} 1 & 1 & 1 \\ 2 & -3 & 4 \\ 2 & 2 & 2 \end{bmatrix} \quad \text{and} \quad h(m,n) = \begin{bmatrix} 3 & 1 \\ 1 & -3 \end{bmatrix}$$

4.3 Using the DFT and IDFT, convolve $x(m, n)$ and a 3×3 Gaussian lowpass filter with $\sigma = 1$. Assume periodicity at the borders.
(i)

$$x(m,n) = \begin{bmatrix} 2 & 1 & 3 & 4 \\ 1 & 1 & 4 & 2 \\ 1 & -1 & 2 & -2 \\ 3 & 2 & -2 & 1 \end{bmatrix}$$

*(ii)

$$x(m,n) = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 1 & 1 & 4 \\ 1 & -1 & 0 & -2 \\ 0 & 2 & -2 & 1 \end{bmatrix}$$

(iii)

$$x(m,n) = \begin{bmatrix} 1 & 1 & 3 & 2 \\ 2 & -1 & -2 & 2 \\ 4 & -1 & 2 & -2 \\ 3 & 2 & -2 & 3 \end{bmatrix}$$

4.4 Using the DFT and IDFT, convolve $x(m,n)$ and a 3×3 averaging lowpass filter. Assume periodicity at the borders.

(i)

$$x(m,n) = \begin{bmatrix} 1 & -1 & 3 & 4 \\ 1 & 2 & -4 & 2 \\ 1 & -1 & 3 & -2 \\ 1 & -2 & -2 & 1 \end{bmatrix}$$

(ii)

$$x(m,n) = \begin{bmatrix} 1 & 3 & -3 & 4 \\ 2 & 0 & 1 & 4 \\ 1 & -1 & 0 & -2 \\ 0 & 2 & -2 & 3 \end{bmatrix}$$

(iii)

$$x(m,n) = \begin{bmatrix} 2 & 0 & 3 & -2 \\ 2 & -1 & 2 & 2 \\ 4 & -1 & 3 & -2 \\ 1 & 0 & -2 & 3 \end{bmatrix}$$

4.5 Using the DFT and IDFT, convolve $x(m,n)$ and a 3×3 Laplacian enhancement filter. Assume periodicity at the borders.

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

*(i)

$$\begin{bmatrix} 7 & 7 & 5 & 1 \\ 4 & 4 & 4 & 1 \\ 0 & 8 & 4 & 3 \\ 5 & 0 & 2 & 1 \end{bmatrix}$$

(ii)

$$\begin{bmatrix} 1 & 2 & 5 & 1 \\ 4 & 1 & 0 & 1 \\ 0 & 0 & 3 & 3 \\ 5 & 0 & 2 & 4 \end{bmatrix}$$

(iii)

$$\begin{bmatrix} 2 & 3 & 1 & 1 \\ 4 & 0 & -4 & 1 \\ 0 & 1 & 4 & 3 \\ 4 & 0 & -2 & 1 \end{bmatrix}$$

4.6 Let an ideal 2-D lowpass filter $H(k, l)$ is defined between the limits $\{-4, -3, -2, -1, 0, 1, 2, 3\}$ in both the directions. Find the filter coefficients for a given cutoff radius r such that $H(k, l) = 1$ inside the circle defined by r and zero elsewhere.
(i) $r = 1$
(ii) $r = 2$
(iii) $r = 3$

4.7 Let an ideal 2-D highpass filter $H(k, l)$ is defined between the limits $\{-4, -3, -2, -1, 0, 1, 2, 3\}$ in both the directions. Find the filter coefficients for a given cutoff radius r such that $H(k, l) = 1$ outside the circle defined by r and zero elsewhere.
(i) $r = 1$
(ii) $r = 2$
(iii) $r = 3$

Chapter 5
Image Restoration

Abstract In the filtering operations presented in the last chapter, it is assumed that no knowledge of the source of degradation of the image is available. In this chapter, the restoration of the images is presented. This is a linear filtering operation in which prior knowledge of the source of degradation of the image is known. A restoration filter is designed based on the nature of the degradation process and the noise. The convolution of this filter with the degraded image restores the image with respect to least-squares error criterion.

Due to nonideal characteristics, resulting from physical limitations, of image formation systems or incorrect operation of the image capturing systems, the captured image is not identical to the scene being captured. Some specific reasons for degradation are improper setting of the focus, motion of the object or the camera and faulty equipment. For image sensors, the restoration process can be post-processing. For image display devices, the restoration process has to be pre-processing. The goal in image restoration is to find a best estimate of the input image from the degraded image.

Another source of degradation of images is noise. Noise is an annoying signal that does not carry information and usually unwanted. Invariably, a signal or image gets corrupted by some type of noise during generation, transmission, and processing. A good estimation of the image from its noisy version is possible, if the image and noise characteristics are known.

5.1 The Image Restoration Process

In image restoration, a degraded image is restored. It is a linear space-invariant filtering operation, but with some knowledge of the process of degradation. The degradation process has to be modeled as accurately as possible by using test images

D. Sundararajan, *Digital Image Processing*, DOI 10.1007/978-981-10-6113-4_5

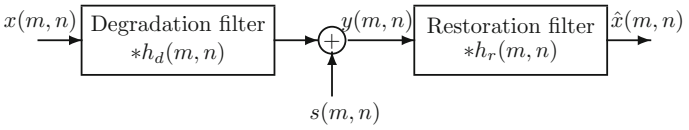


Fig. 5.1 The block diagram of the image restoration process

or otherwise. In addition, the noise is considered additive. The block diagram of the image restoration process is shown in Fig. 5.1.

The $N \times N$ image $x(m,n)$ is degraded by a process characterized by its impulse response $h_d(m,n)$. Further, a noise component $s(m,n)$ is also added, resulting in the degraded image $y(m,n)$. The task is to restore the image so that the least-squares error between the input image and the restored image $\hat{x}(m,n)$

$$E = \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} |x(m,n) - \hat{x}(m,n)|^2$$

is minimized. The degradation process is characterized by the 2-D linear convolution

$$y(m,n) = \sum_k \sum_l x(k,l) h_d(m-k, n-l) + s(m,n)$$

The problem is to design a restoration filter with impulse response $h_r(m,n)$ so that the convolution of the degraded signal with this filter restores the input signal.

$$\hat{x}(m,n) = \sum_k \sum_l y(k,l) h_r(m-k, n-l)$$

5.2 Inverse Filtering

Assuming that there is no noise component, the degradation process is given, in the frequency domain, by

$$Y(k,l) = X(k,l) H_d(k,l)$$

where $Y(k,l)$, $X(k,l)$, and $H_d(k,l)$ are the corresponding DFTs of the degraded image, the input image, and the impulse response of the degradation process. Obviously, the image can be restored by the operation, called inverse filtering,

$$X(k,l) = \frac{Y(k,l)}{H_d(k,l)}$$

This operation requires point-by-point division, which creates problem when the values of $H_d(k, l)$ are zero or very small. Further, the noise components, if present, get more and more amplified at high frequencies. This is due to the fact that the frequency response of the degradation process, typically, is of lowpass character and the noise is usually broadband. Although modifications of the inverse filtering is possible to alleviate the problem, the restored image does not satisfy the least-squares error criterion in an optimum manner.

5.3 Wiener Filter

The Wiener filter restores the true signal quite well from the degraded signal, even in the presence of noise. For the restoration procedure to be practically effective, the minimization of the error between the input and degraded images should be based on some criterion. Due to its mathematical tractability, the well-known least-squares error criterion is used in the restoration problem formulation. The Wiener filter is based on a statistical approach. It is assumed that the signal and noise (additive) are wide-sense stationary and linear stochastic processes with known spectral densities; the signal and the noise are uncorrelated and the mean of the noise or signal is zero. The derivation of the Wiener filter, using the orthogonality of the image and the noise, is similar to finding the Fourier coefficients in Chap. 3. The expression for the least-squares error is set up, differentiated with respect to the filter coefficients and set equal to zero. Solving this equation yields the coefficients of the Wiener restoration filter. For ease of understanding, we derive the 1-D Wiener filter first.

Let the true and restored signals be $x(n)$ and $\hat{x}(n)$, respectively, of length N and their DFT be $X(k)$ and $\hat{X}(k)$. Then, the power spectrum of $x(n)$ is $|X(k)|^2 = X(k)X^*(k)$. The power spectrum of a signal, which is the DFT of its autocorrelation, represents the distribution of the signal power with respect to frequency. The spectrum is a representation of the complex amplitudes of a signal with respect to frequency. When the spectrum is multiplied with its conjugate, it is the correlation of the signal with itself (autocorrelation) in the time domain. The product of the complex amplitude of a frequency component with its conjugate is the squared magnitude and represents the power at that frequency. Let the degraded signal be $y(n)$ and the additive Gaussian noise be $s(n)$ with its power spectrum $|S(k)|^2 = S(k)S^*(k)$. Let the frequency responses of the degradation and Wiener filters $h_d(n)$ and $h_r(n)$ be $H_d(k)$ and $H_r(k)$, respectively. The problem is to find the estimate, $\hat{x}(n)$, of $x(n)$ from $y(n)$ such that the least-squares error, E, is

$$E = \sum_{n=0}^{N-1} (x(n) - \hat{x}(n))^2$$

minimized. Assuming that the estimated signal $\hat{x}(n)$ is given by

$$\hat{x}(n) = \sum_k y(n-k)h_r(k)$$

the task is to find the filter coefficients so that the least-squares error is minimized. From Parseval's theorem, we get, in the frequency domain,

$$E = \frac{1}{N}\sum_{k=0}^{N-1} |(X(k) - \hat{X}(k))|^2$$

where

$$\hat{X}(k) = H_r(k)Y(k) = H_r(k)H_d(k)X(k) + H_r(k)S(k)$$

and

$$X(k) - \hat{X}(k) = (1 - H_r(k)H_d(k))X(k) - H_r(k)S(k)$$

Substituting for $X(k) - \hat{X}(k)$ in the previous expression for E, we get

$$\begin{aligned} E &= \frac{1}{N}\sum_{k=0}^{N-1} |(1 - H_r(k)H_d(k))X(k) - H_r(k,l)S(k)|^2 \\ &= \frac{1}{N}\sum_{k=0}^{N-1} |(1 - H_r(k)H_d(k))X(k)|^2 + |H_r(k)S(k)|^2 \\ &= \frac{1}{N}\sum_{k=0}^{N-1} |(1 - H_r(k)H_d(k))|^2|X(k)|^2 + |H_r(k)|^2|S(k)|^2 \end{aligned}$$

since $x(n)$ and $s(n)$ are uncorrelated. The derivative of the product of a complex function and its conjugate is equal to two times its conjugate. That is,

$$\frac{d|z|^2}{dz} = \frac{d(zz^*)}{dz} = 2z^*$$

Setting the derivative of the last expression with respect to $H_r(k)$ equal to zero, we get

$$2(-(1 - H_r^*(k)H_d^*(k))H_d(k)|X(k)|^2 + H_r^*(k)|S(k)|^2) = 0$$

$$H_r^*(k) = \frac{H_d(k)|X(k)|^2}{|H_d(k)|^2|X(k)|^2 + |S(k)|^2}$$

$$H_r(k) = \frac{H_d^*(k)}{|H_d(k)|^2 + |S(k)|^2/|X(k)|^2} \tag{5.1}$$

The power spectra of the true signal, noise, and the frequency response of the degradation filter determine the Wiener filter. Obviously, these parameters must be known

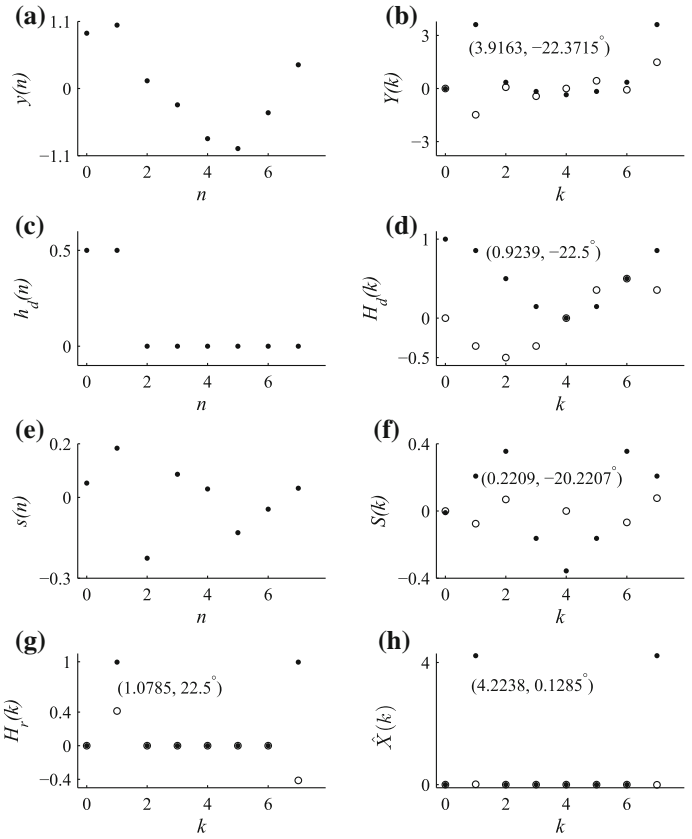


Fig. 5.2 The restoration process. **a** The degraded signal and **b** its DFT; **c** the impulse response of the degradation process and **d** its DFT; **e** the noise signal and **f** its DFT; **g** the DFT of the Wiener filter; **h** the DFT of the restored signal

or estimated with adequate accuracy. The IDFT of $H_r(k)$ is the set of optimum filter coefficients the degraded signal has to be passed to get the best estimate of the true signal with respect to the least-squares error criterion.

Example 5.1 A sinusoidal signal $x(n) = \cos\left(\frac{2\pi}{8}n\right)$, with samples

$$x(n) = \{1, \frac{1}{\sqrt{2}}, 0, -\frac{1}{\sqrt{2}}, -1, -\frac{1}{\sqrt{2}}, 0, \frac{1}{\sqrt{2}}\}$$

shown in Fig. 5.3a and b by dots, has been blurred by a process with finite impulse response $\{h_d(0) = 0.5, h_d(1) = 0.5\}$, shown in Fig. 5.2c, and corrupted by an additive Gaussian noise $s(n)$ with mean zero and standard deviation $\sigma = 0.1$

$$s(n) = \{0.0538, 0.1834, -0.2259, 0.0862, 0.0319, -0.1308, -0.0434, 0.0343\}$$

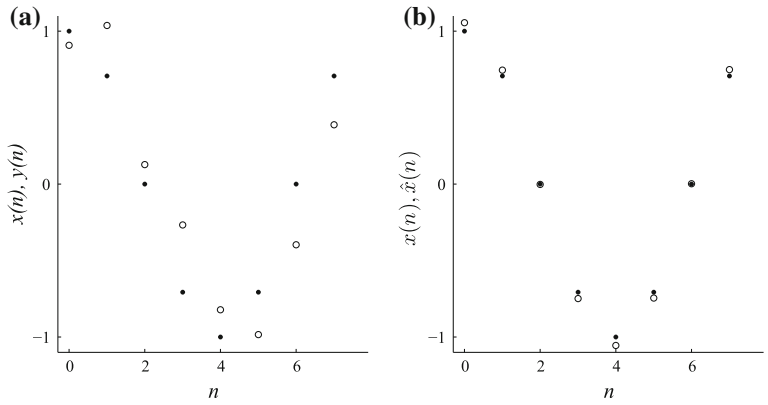


Fig. 5.3 **a** True signal $x(n)$ (*dots*) and the degraded signal $y(n)$ (*unfilled circles*); **b** true signal $x(n)$ (*dots*) and the restored signal $\hat{x}(n)$ (*unfilled circles*)

shown in Fig. 5.2e. The samples of the degraded signal are

$$y(n) = \{0.9073, 1.0369, 0.1277, -0.2673, -0.8217, -0.9843, -0.3969, 0.3878\}$$

shown in Fig. 5.2a with dots and in Fig. 5.3a with unfilled circles. Restore the true signal using the Wiener filter.

Solution
The circular convolution of $x(n)$ and $h_d(n)$ is

$$\{x(7) + x(0), x(0) + x(1), x(1) + x(2), x(2) + x(3), x(3) + x(4), x(4) + x(5), \\ x(5) + x(6), x(6) + x(7)\}/2$$

The resulting values are

$$\{0.8536, 0.8536, 0.3536, -0.3536, -0.8536, -0.8536, -0.3536, 0.3536\}$$

This signal added with the noise is the degraded signal $y(n)$. The DFT of the true signal $x(n)$ and its power spectrum are

$$X(k) = \{0, 4, 0, 0, 0, 0, 0, 4\} \quad \text{and} \quad |X(k)|^2 = \{0, 16, 0, 0, 0, 0, 0, 16\}$$

The DFT of $y(n)$

$$Y(k) = \{-0.0105, 3.6215 - j1.4906, 0.3549 + j0.0679, -0.1635 - j0.4414, \\ -0.3567, -0.1635 + j0.4414, 0.3549 - j0.0679, 3.6215 + j1.4906\}$$

is shown in Fig. 5.2b. The DFT of $h_d(n)$

$$H_d(k) = \{1, 0.8536 - j0.3536, 0.5 - j0.5, 0.1464 - j0.3536, 0, 0.1464 + j0.3536, 0.5 + j0.5, 0.8536 + j0.3536\}$$

is shown in Fig. 5.2d. Its power spectrum is

$$|H_d(k)|^2 = H_d(k)H_d^*(k) = \{1, 0.8536, 0.5000, 0.1464, 0, 0.1464, 0.5000, 0.8536\}$$

The conjugate of $H_d(k)$, $H_d^*(k)$, is obtained by changing the sign of the imaginary part of the values of $H_d(k)$.

The DFT of $s(n)$

$$S(k) = \{-0.0105, 0.2073 - j0.0764, 0.3549 + j0.0679, -0.1635 - j0.4414, -0.3567, -0.1635 + j0.4414, 0.3549 - j0.0679, 0.2073 + j0.0764\}$$

is shown in Fig. 5.2f. Its power spectrum is

$$|S(k)|^2 = S(k)S^*(k) = \{0.0001, 0.0488, 0.1305, 0.2216, 0.1272, 0.2216, 0.1305, 0.0488\}$$

With all the required quantities in Eq. (5.1) available, the DFT of the Wiener filter $H_r(k)$ is found to be

$$H_r(k) = \{0, 0.9964 + j0.4127, 0, 0, 0, 0, 0, 0.9964 - j0.4127\}$$

and it is shown in Fig. 5.2g. Note that all multiplications and divisions are pointwise operations. The product $H_r(k)Y(k)$ yields the DFT of the restored signal, shown in Fig. 5.2h.

$$H_r(k)Y(k) = \hat{X}(k) = \{0, 4.2238 + j0.0095, 0, 0, 0, 0, 0, 4.2238 - j0.0095\}$$

The IDFT of these values yields the restored signal $\hat{x}(n)$, shown in Fig. 5.3b with unfilled circles. Comparing Fig. 5.3a and b, the Wiener filter has restored the true signal to a good extent.

The frequency index of the true signal is one. That is, it makes one cycle in eight samples. Notice that the DFT of the Wiener filter is zero except at $k = 1$ and $k = 7$. This implies that the noise components at other frequencies are eliminated. In Fig. 5.2b, d, f, g, and h, the magnitude and angle in degrees of the DFT values with index one are given. The degradation filter attenuates the input signal with its magnitude 0.9239 and adds a phase of $-22.5°$. By having a magnitude 1.0785, the Wiener filter compensates the attenuation. Further, with a phase shift of 22.5°, the filter restores the true signal quite well. The only problem is that the noise component with frequency index one is passed. The least-squares error between the degraded

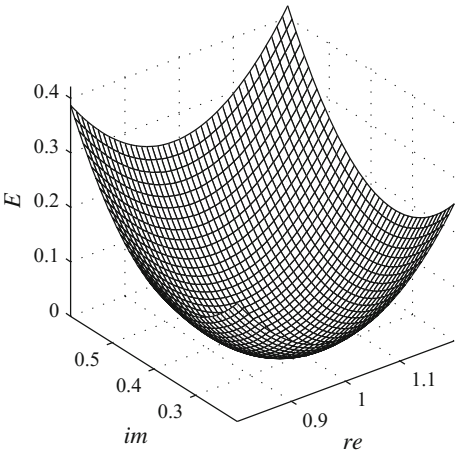


Fig. 5.4 The least-squares error with the real and imaginary parts of the DFT of the Wiener filter coefficients, $H_r(k)$, varying around the optimum point

and true signals is 0.6952 and that between the restored and true signals is 0.0125. The ratio of the errors is 55.616. The least-squares error with the real and imaginary parts of the DFT of the Wiener filter coefficients, $H_r(k)$, varying around the optimum point is shown in Fig. 5.4.

5.3.1 The 2-D Wiener Filter

The expression for the 2-D Wiener filter is readily obtained from that of the 1-D as

$$H_r(k,l) = \frac{H_d^*(k,l)}{|H_d(k,l)|^2 + |S(k,l)|^2/|X(k,l)|^2} \tag{5.2}$$

where $|H_d(k,l)|^2$ is the power spectrum of the degradation process, $H_r(k,l)$ is the DFT of the Wiener filter, $|S(k,l)|^2$ and $|X(k,l)|^2$ are the power spectral densities of the noise and the true image, respectively. The restored image $\hat{x}(m,n)$ is obtained from the degraded image $y(m,n)$ as

$$\hat{x}(m,n) = \text{IDFT}\,(H_r(k,l)Y(k,l))$$

The procedure is typical of frequency-domain filtering.

1. Find the DFT $Y(k,l)$ of the degraded image $y(m,n)$.
2. Derive the Wiener filter $H_r(k,l)$.
3. Multiply pointwise $Y(k,l)$ with the Wiener filter $H_r(k,l)$.
4. Compute the IDFT $H_r(k,l)Y(k,l)$ to get the restored image.

The second term in the denominator of the expression for the Wiener filter, Eq. 5.2,

$$|S(k,l)|^2/|X(k,l)|^2$$

makes the filter effective for restoration. It represents the inverse of signal-to-noise ratio. When the signal is relatively very strong (during the lower part of the spectrum of the degraded signal), the term becomes negligible and

$$H_r(k,l) \approx \frac{1}{H_d(k,l)}$$

the Wiener filter becomes an approximation of the inverse filter. On the other hand, when the noise is relatively very strong (during the upper part of the spectrum of the degraded signal), the term dominates and

$$H_r(k,l) \to 0$$

This behavior mitigates the noise amplification in the inverse filtering. In the case where the mean of the image or noise is nonzero, modification of the filtering process is required. As the Wiener filter, in general, is not separable, the row–column method of processing is not applicable. With no blurring ($H_d = 1$), the Wiener filter becomes a smoothing filter having a lowpass frequency response.

$$H_r(k,l) = \frac{1}{1 + |S(k,l)|^2/|X(k,l)|^2}$$

When the power spectra in the expression

$$|S(k,l)|^2/|X(k,l)|^2$$

are not available, the ratio can be substituted by a constant K (selected by trial and error).

5.4 Image Degradation Model

It often happens that, during the acquisition of an image, there is a relative uniform linear motion between the image and the sensor. Because of this, each pixel value is replaced by an average of some number of pixels in the neighborhood. In Example 5.1, the average of two values replaced the original value at each point of the signal. The model for the degradation of an image by motion is averaging, just summation of uniformly weighted pixel values of spatially shifted copies of an image. Let us find the 1-D degradation model. Let the correct exposure time be one sampling interval. With the sensor moving and two sampling intervals of exposure time, the sample

value becomes the average of two samples and so on. Let the exposure time be M sampling intervals, instead of one. Then, the degraded signal values $y(n)$ of the true signal $x(n)$ are

$$y(n) = \sum_{m=0}^{M-1} x(n-m)$$

The division by M to get the average is left out in this expression. Let $X(k)$ be the DFT of $x(n)$. From the DFT time-shift theorem, we get

$$x(n-m) \leftrightarrow e^{-j\frac{2\pi}{N}mk} X(k)$$

where N is the length of the sequence $x(n)$. Now, the DFT of $y(n)$ is

$$\begin{aligned} Y(k) &= X(k) \sum_{m=0}^{M-1} e^{-j\frac{2\pi}{N}mk} = X(k)(1 + e^{-j\frac{2\pi}{N}k} + e^{-j\frac{2\pi}{N}2k} + \cdots + e^{-j\frac{2\pi}{N}(M-1)k}) \\ &= X(k)\frac{1 - e^{-j\frac{2\pi}{N}Mk}}{1 - e^{-j\frac{2\pi}{N}k}} = X(k)e^{-j\frac{\pi}{N}(M-1)k}\frac{\sin(\frac{\pi}{N}Mk)}{\sin(\frac{\pi}{N}k)} = X(k)H_d(k) \end{aligned}$$

Therefore, the DFT of the impulse response of the process due to motion is

$$H_d(k) = e^{-j\frac{\pi}{N}(M-1)k}\left(\frac{\sin(\frac{\pi}{N}Mk)}{\sin(\frac{\pi}{N}k)}\right)$$

This is just the DFT of M samples with value 1, the averaging filter,

$$h_d(n) = \begin{cases} 1 \text{ for } n = 0, 1, \ldots, M-1 \\ 0 \text{ for } n = M, M+1, \ldots, N-1 \end{cases}$$

In Example 5.1, $M = 2$ and $N = 8$. Then,

$$H_d(k) = e^{-j\frac{\pi}{N}(M-1)k}\left(\frac{\sin(\frac{\pi}{N}Mk)}{\sin(\frac{\pi}{N}k)}\right) = e^{-j\frac{\pi}{8}k}\left(\frac{\sin(\frac{\pi}{4}k)}{\sin(\frac{\pi}{8}k)}\right)$$

which is just twice that of $H_d(k)$ in Example 5.1.

Let $x(m,n)$ be a $N \times N$ image and $h_d(m,n)$ be the $P \times Q$ impulse response of the process due to motion and $x(m,n) \leftrightarrow X(k,l)$. Then,

$$x(m-p, n-q) \leftrightarrow X(k,l)e^{-j\frac{2\pi}{N}(kp+lq)}$$

$$y(m,n) = \sum_{p=0}^{P-1}\sum_{q=0}^{Q-1} x(m-p, n-q)$$

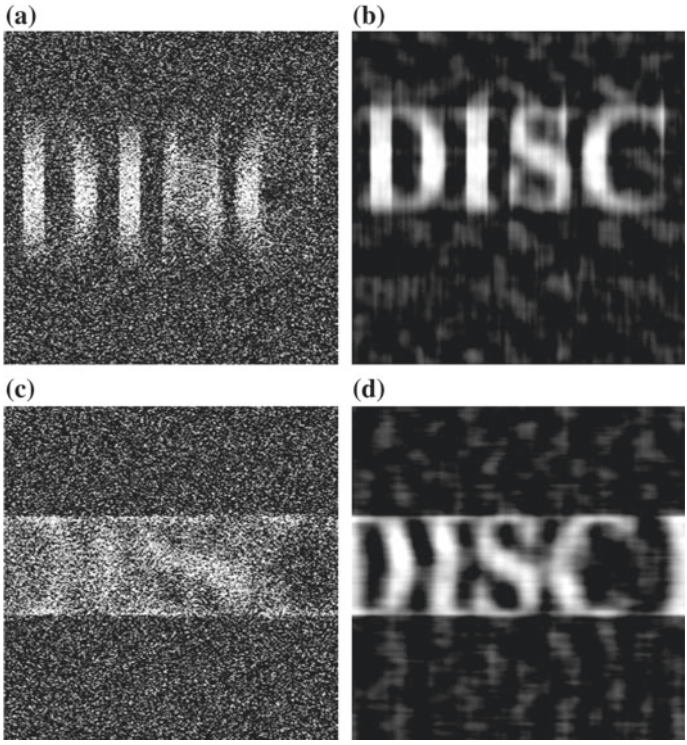


Fig. 5.5 **a** A noisy and blurred 256 × 256 image; **b** its restored version; **c** a noisy and blurred 256 × 256 image; **d** its restored version

$$H_d(k,l) = \sum_{p=0}^{P-1}\sum_{q=0}^{Q-1} e^{-j\frac{2\pi}{N}pk} e^{-j\frac{2\pi}{N}qk} = \sum_{p=0}^{P-1} e^{-j\frac{2\pi}{N}pk} \sum_{q=0}^{Q-1} e^{-j\frac{2\pi}{N}qk}$$
$$= e^{-j\frac{\pi}{N}(P-1)k}\left(\frac{\sin(\frac{\pi}{N}Pk)}{\sin(\frac{\pi}{N}k)}\right) e^{-j\frac{\pi}{N}(Q-1)l}\left(\frac{\sin(\frac{\pi}{N}Ql)}{\sin(\frac{\pi}{N}l)}\right)$$

Figure 5.5a shows a corrupted image. The original image got deblurred by vertical motion and Gaussian noise (variance 0.2). Figure 5.5c shows a corrupted image due to deblurring by horizontal motion and Gaussian noise (variance 0.2). Figure 5.5b and d show, respectively, the restored versions using the Wiener filter. The degradation was carried out by circular convolution. Considering the heavy degradation, the restored quality of the images is good.

5.5 Characterization of the Noise and Its Reduction

The future values of a random signal cannot be predicted. Noise is usually of random nature, and it is characterized by its probability density function. A random variable is characterized by two values, mean (center of gravity of the density), and variance (measure of the spreadness). If the variance is small, it is highly probable that its values are close to the mean and vice versa. Suppose the mean depth of a swimming pool is 1 m, it does not mean that a person without knowing swimming can jump into any part of the pool and hope to come out alive. Only with a knowledge of the spreadness of the depth, we can be sure. In image processing, the histograms of two images can have the same mean but the spreadness may vary widely. If the variance is small, then histogram equalization or stretching can enhance the image.

5.5.1 Uniform Noise

The distribution of the density between limits a and b is $p(x) = 1/(b-a)$ and zero otherwise. The mean is the center of the range of the interval, $(a+b)/2$. The variance is

$$\sigma^2 = \int_a^b \frac{(x - 0.5(a+b))^2}{(b-a)} dx = \frac{(b-a)^2}{12}$$

Figure 5.6 shows the uniform distribution with $a = -2$ and $b = 16$.

5.5.2 Gaussian Noise

Gaussian noise is the most commonly used noise model. One important property of Gaussian (also called normal) noise is that the weighted sum of its values is also Gaussian. That is, the output of any linear filter is also Gaussian for a Gaussian input. The Fourier transform of a Gaussian function is also another Gaussian. It is

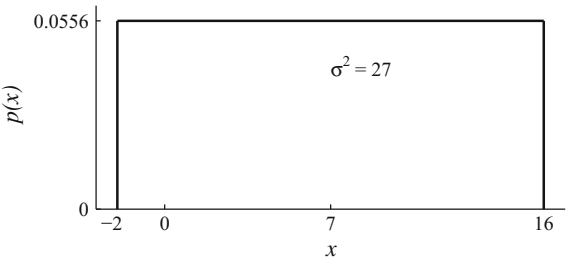


Fig. 5.6 Uniform probability density function

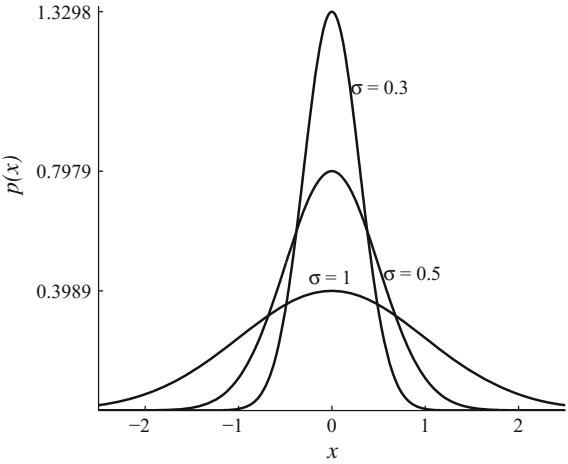


Fig. 5.7 Gaussian probability density function

smooth and isotropic. For all these reasons, it is mathematically tractable. Further, random variables occurring in practice can be approximated by this distribution. Its probability density function is defined as

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-0.5(\frac{x-\mu}{\sigma})^2}$$

where μ is the mean and σ is the standard deviation of the distribution. Figure 5.7 shows the Gaussian distribution with the mean $\mu = 0$ and the standard deviation $\sigma = 0.3$, $\sigma = 0.5$, and $\sigma = 1$. It is a bell-shaped curve and is symmetric with respect to the mean. For nonzero values of μ, the curves get shifted. The smaller the value of σ, the higher is the peak at the center and the steeper are the descents.

5.5.3 Periodic Noise

Periodic noise is the occurrence of unwanted sinusoidal components in the image, typically due to the interferences from electrical machines or power and signal transmission lines. It is effectively filtered in the frequency domain. The spectral values of a sinusoid are a complex conjugate pair and, therefore, they are placed on a circle, as shown in Fig. 5.9e.

5.5.4 Noise Reduction

The low-frequency components of a signal makes the smoother part of the signal, and the high-frequency components makes the jagged part. In the graphical representation

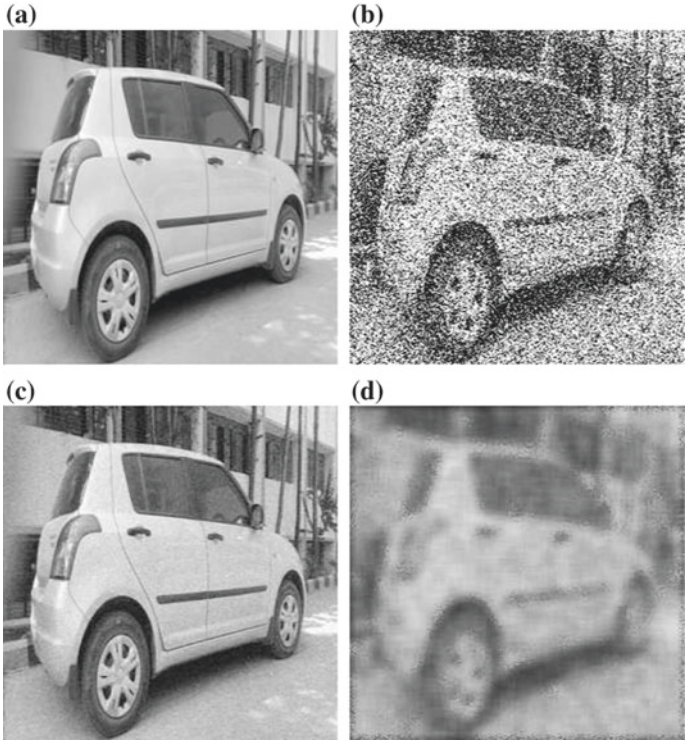


Fig. 5.8 **a** A 256 × 256 8-bit image; **b** its corrupted version; **c** noise reduced version by averaging; **d** noise reduced version by Wiener filtering

of the signal, a smooth curve through the samples represents the smoothed signal. This suggests that the filter should average out the rapidly varying component of the signal. The output of the simplest averaging filter, $y(n) = 0.5x(n) + 0.5x(n-1)$, is the average of the present and preceding input samples. Here, the rapidly varying component of a signal is the disturbance and the slowly varying component is desired. The opposite is the case with highpass filters, the simplest being $y(n) = 0.5x(n) - 0.5x(n-1)$. This filter readily passes high-frequency components and suppresses low-frequency components. For example, the output of this filter is zero for a constant signal. Here, the rapidly varying component of a signal is desired and the slowly varying component is the disturbance. Depending on the type of noise and its characteristics, suitable filters have to be used for its mitigation.

One of the effective ways of reducing Gaussian noise is to average a set of corrupted images of the same scene. For example, satellites periodically take the picture of the same scene. As the noise is normally distributed with the mean zero, the average of a set of corrupted images should have a noise power close to zero. Figure 5.8a shows a 256 × 256 8-bit image. It is corrupted with Gaussian noise with mean zero and standard deviation $\sigma = 0.2$, as shown in Fig. 5.8b. Figure 5.8c shows the average

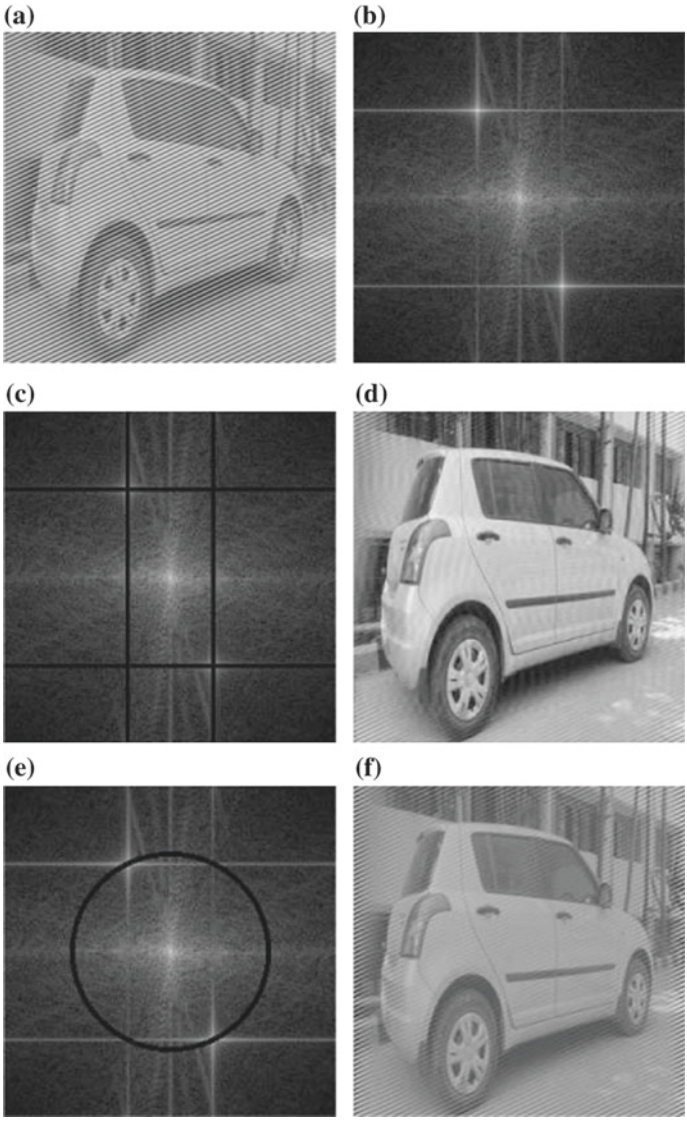


Fig. 5.9 **a** A 256 × 256 8-bit image corrupted by periodic noise; **b** its DFT spectrum; **c** filtered spectrum by a notch filter; **d** the filtered image; **e** filtered spectrum by a bandreject filter; **f** the filtered image

of ten different images of the same object with the noise substantially reduced. If we have only one image, then the averaging process, although very effective, is ruled out.

An adaptive Wiener filter is defined, in a local neighborhood, as

$$\hat{x}(m,n) = y(m,n) - \frac{\sigma_N^2}{\sigma_l^2}(y(m,n) - \bar{y}_l)$$

where $\hat{x}(m,n)$ is the restored signal, $y(m,n)$ is the signal $x(m,n)$ corrupted by noise with variance σ_N^2, σ_l^2 is the local variance, and $\bar{y}_l$ is the local average. If the local variance is relatively high, then the filter returns a value that is nearly the same as the input. This is appropriate as a high variance indicates features such as edges, which should be preserved. If the variances are about equal, a value close to the average is appropriate. If $\sigma_N^2 > \sigma_l^2$, then the ratio σ_N^2/σ_l^2 has to be set to 1 or allow negative values and rescale the output for display. Figure 5.8d shows the image filtered by the Wiener lowpass filter with a 11×11 neighborhood. Considering that the degradation is very severe, the restored image is reasonable.

A notch filter has notches (ideally nulls) at some frequencies. It is used to eliminate specific frequency components of a signal. For example, power system frequency 60 Hz and its harmonics are unwanted in signal transmission lines. A bandpass filter has high gain at an intermediate band of frequencies only. It passes intermediate frequency components and suppresses the rest. Figure 5.9a and b shows, respectively, a 256×256 8-bit image corrupted by periodic noise and its DFT spectrum. The two spikes in (b), away from the center and located diagonally and the horizontal and vertical lines originating from them, are the spectral components of the periodic noise. Note that the noise is composed of several sinusoidal components. A notch filter suppresses frequency components in a specified neighborhood. In Fig. 5.9c, the spectrum of the filtered image is shown. The frequency components in some number of rows and columns in the spectrum dominated by noise in (b) have been suppressed. The filtered image, with reduced noise, is shown in Fig. 5.9d. A bandreject filter is a ring of zeros and suppresses frequency components located in that ring. In Fig. 5.9e, the spectrum of the filtered image is shown. The frequency components in a ring of sufficient thickness encapsulating the spikes in the noisy spectrum in (b) have been suppressed. The filtered image, with reduced noise, is shown in Fig. 5.9f.

5.6 Summary

- Image restoration is image enhancement in which a knowledge of the degradation process is known.
- In restoration from degradation due to noise only, suitable filters can be used to restore the degraded image.
- Gaussian, salt-and-pepper, speckle, and periodic are typical noise models.
- If the degradation is due to improper setting or characteristics of the devices at the time of image formation or display, the degradation process can be characterized by its impulse response. Then, inverse filtering or deconvolution restores the image.
- In case where noise is also present, the inverse filtering is ineffective, since the degradation functions typically have spectrum that is similar to that of the lowpass

filter. Then, noise amplification at high frequencies occurs in inverse filtering, making it ineffective.
- An effective solution, called Wiener filtering, is to formulate the restoration problem to minimize the least-squares error between the degraded and the restored image.
- Wiener filtering requires the power spectra of the noise and the image before degradation. It is assumed that noise is additive and has zero mean and uncorrelated with the signal.
- As these spectra are not usually available, they have to be estimated in practice.
- The Wiener filter coefficients are derived using these power spectra.
- The restored signal is obtained by passing the degraded signal through the Wiener filter.

Exercises

5.1 A signal with power spectral density

$$|X(k)|^2 = \{64, 0, 0, 0, 0, 0, 0, 0\}$$

has been blurred by a process with finite impulse response

$$\{h_d(0) = 0.5, h_d(1) = 0.5\}$$

and corrupted by an additive Gaussian noise with power spectral density

$$\{0.0002, 0.0071, 0.0273, 0.0121, 0.0220, 0.0121, 0.0273, 0.0071\}$$

The samples of the degraded signal are

$$y(n) = \{0.9898, 0.9759, 1.0319, 1.0313, 0.9135, 0.9970, 0.9835, 1.0628\}$$

Restore the true signal using the Wiener filter.

***5.2** A signal with power spectral density

$$|X(k)|^2 = \{0, 16, 0, 0, 0, 0, 0, 16\}$$

has been blurred by a process with finite impulse response

$$\{h_d(0) = 0.5, h_d(1) = 0.5\}$$

and corrupted by an additive Gaussian noise with power spectral density

$$\{0.0119, 0.0058, 0.0807, 0.0107, 0.1725, 0.0107, 0.0807, 0.0058\}$$

The samples of the degraded signal are

$$y(n) = \{-0.4305, 0.3907, 0.8310, 0.9653, 0.2446, -0.3503, -0.7983, -0.7435\}$$

Restore the true signal using the Wiener filter.

5.3 A signal with power spectral density

$$|X(k)|^2 = \{0, 0, 16, 0, 0, 0, 16, 0\}$$

has been blurred by a process with finite impulse response

$$\{h_d(0) = 0.5, h_d(1) = 0.5\}$$

and corrupted by an additive Gaussian noise with power spectral density

$$\{0.0067, 0.1671, 0.1262, 0.1311, 0.1654, 0.1311, 0.1262, 0.1671\}$$

The samples of the degraded signal are

$$y(n) = \{0.6544, 0.5086, -0.6492, -0.5742, 0.3938, 0.7350, -0.5616, -0.4252\}$$

Restore the true signal using the Wiener filter.

5.4 A signal with power spectral density

$$|X(k)|^2 = \{0, 0, 0, 16, 0, 16, 0, 0\}$$

has been blurred by a process with finite impulse response

$$\{h_d(0) = 0.5, h_d(1) = 0.5\}$$

and corrupted by an additive Gaussian noise with power spectral density

$$\{0.0772, 0.0694, 0.0930, 0.0001, 0.0545, 0.0001, 0.0930, 0.0694\}$$

The samples of the degraded signal are

$$y(n) = \{0.1272, 0.2353, -0.4300, 0.2133, -0.2887, -0.0976, 0.3358, -0.3732\}$$

Restore the true signal using the Wiener filter.

5.5 A signal with power spectral density

$$|X(k)|^2 = \{0, 0, 16, 0, 0, 0, 16, 0\}$$

has been blurred by a process with finite impulse response

$$\{h_d(0) = 0.5, h_d(1) = 0.5\}$$

and corrupted by an additive Gaussian noise with power spectral density

$$\{0.1584, 0.0055, 0.0030, 0.1719, 0.0047, 0.1719, 0.0030, 0.0055\}$$

The samples of the degraded signal are

$$y(n) = \{-0.3581, 0.5292, 0.5198, -0.3412, -0.5804, 0.5697, 0.5835, -0.5244\}$$

Restore the true signal using the Wiener filter.

Chapter 6
Geometric Transformations and Image Registration

Abstract Interpolation of an image is often required to change its size and in operations such as rotation. The interpolation of images is described first. Next, geometric transformations such as translation, scaling, rotation, and shearing are presented. Correlation operation is a similarity measure between two images. It can detect and locate the position of an object in an image, if present. The registration of images of the same scene, taken at different times and conditions, is also presented.

In 1-D signal processing, operations, such as shifting, folding, and scaling, are often used in addition to arithmetic operations. In this chapter, we study the 2-D extensions of such operations. The image coordinates (m, n) are changed to (p, q) in geometric transformations. As they are, usually, not integers, rounding is required to make them integers. Therefore, interpolation is required to estimate the corresponding pixel values.

The images of the same scene, taken by different cameras or at different times, have to be aligned with a reference image, so that the change in the characteristics of the images yields information about the changes in the behavior of the objects in the scene. The alignment process usually requires operations such as rotation, scaling, and shifting. In turn, operations such as rotation requires interpolation. Therefore, we describe the interpolation operation first. Then, the geometric transformation is described. Correlation and image registration are presented last.

6.1 Interpolation

Interpolation is a basic tool in digital signal processing, since, after processing a signal in its digital form, interpolation is required to reconstruct the corresponding continuous signal. Remember that most naturally occurring signals are continuous, and the processed output is also required in that form, most of the times. For image reconstruction or image operations such as rotation, the coordinates of the output image are usually not the same of those of the input. Ideal interpolation requires

D. Sundararajan, *Digital Image Processing*, DOI 10.1007/978-981-10-6113-4_6

Table 6.1 Pixel values of a 2 × 2 image (*left*). Pixel values after interpolation along the rows (*middle*). Pixel values after nearest-neighbor interpolation (*right*)

2	1
3	4

2	1	1
3	4	4

2	1	1
3	4	4
3	4	4

reconstruction of the continuous signal from its samples and then resample the continuous signal, which is not practical. Appropriate interpolation methods, which approximate ideal reconstruction, have to be used to find the pixel values at new coordinates from the given ones. The known values are appropriately weighted and summed to get the interpolated value at intermediate locations. The weight corresponding to a pixel is a function of its distance from the interpolated pixel. The shorter the distance the heavier is the weight.

6.1.1 Nearest-Neighbor Interpolation

In the nearest-neighbor interpolation method, the interpolated pixel value is that of its nearest-neighbor. The index of the nearest- neighbor is found using the rounding operation. In rounding a decimal number, the number is replaced by the next greater integer if the fractional part is 0.5 or greater. Otherwise, it is replaced by the largest integer not greater than itself. Let the input data be $\{x(1) = 7, x(2) = 8\}$. Then,

$$x(1.3) = x(round(1.3)) = x(1) = 7 \quad \text{and} \quad x(1.8) = x(round(1.8)) = x(2) = 8$$

Consider the 2 × 2 image shown in Table 6.1 (left). Let us interpolate the middle values. Pixel values after interpolation along the rows are shown in the table (middle). By interpolating these values along the columns, we get the nearest-neighbor interpolated image, as shown in the table (right). While this method is the simplest, the interpolated image is very blurry.

6.1.2 Bilinear Interpolation

In linear interpolation, it is assumed that a straight line is drawn between two neighboring points, and the interpolated value is a linear combination of the distances of the neighbors and their values. In bilinear interpolation, linear interpolation is carried out in each of the two orthogonal coordinates of the image. As in the case of computation of the 2-D DFT by the row–column method, linear interpolation is car-

Table 6.2 Pixel values of a 4 × 4 image (*left*). Pixel values after linear interpolation along the rows (*right*)

2	1	4	2
3	1	2	1
1	2	1	4
4	1	2	3

2.00	1.50	1.00	2.50	4.00	3.00	2.00
3.00	2.00	1.00	1.50	2.00	1.50	1.00
1.00	1.50	2.00	1.50	1.00	2.50	4.00
4.00	2.50	1.00	1.50	2.00	2.50	3.00

ried out along any one of the two directions and the resulting partial result is linearly interpolated in the other direction. Up to four nearest-neighbors are involved in the interpolation. Consider the set of pixels

$$\begin{array}{ccc} \bullet x(m,n) & & \bullet x(m,n+1) \\ & \bullet x(m',n') & \\ \bullet x(m+1,n) & & \bullet x(m+1,n+1) \end{array}$$

The bilinear interpolated value $x(m', n')$ of a pixel at the location (m', n') of an image is given by the expression

$$\begin{aligned} x(m',n') = (1-c)((1-r)x(m,n) + (r)x(m+1,n)) \\ + (c)((1-r)x(m,n+1) + (r)x(m+1,n+1)) \end{aligned} \tag{6.1}$$

where $r = m' - m$ and $c = n' - n$ and the distance between pixel locations is one. The 2-D interpolation can be decomposed into a 1-D row interpolation followed by a 1-D column interpolation or vice versa. In Eq. (6.1), 1-D interpolation along the columns is carried out first followed by 1-D row interpolation. While the two 1-D interpolation operations are linear, their sequential application is nonlinear.

Consider the 4 × 4 image shown in Table 6.2 (left). Let us interpolate along the rows and find the values in the middle of the pixels. Then, $r = 0$ and $c = 0.5$. Equation (6.1) reduces to

$$\begin{aligned} x(m,n') &= (1-0)((1-0.5)x(m,n) + (0.5)x(m,n+1)) \\ &= 0.5(x(m,n) + x(m,n+1)) \end{aligned}$$

which is just the average of the adjacent pixel values. For example, the first interpolated value is $(2 + 1)/2 = 1.5$. The second interpolated value is $(1 + 4)/2 = 2.5$ and so on. The interpolated image along the rows is shown in Table 6.2 (right). Carrying out the interpolation of these values along the columns yields the interpolated image, shown in Table 6.3. This method is often used in practice, and the interpolated image is less blurry compared with the nearest-neighbor algorithm. Interpolation is determining the values of a function inside the range of known values. Therefore, we got 7 × 7 output for a 4 × 4 input. If we need a 8 × 8 output, extrapolation is

Table 6.3 Pixel values after bilinear interpolation

2.00	1.50	1.00	2.50	4.00	3.00	2.00
2.50	1.75	1.00	2.00	3.00	2.25	1.50
3.00	2.00	1.00	1.50	2.00	1.50	1.00
2.00	1.75	1.50	1.50	1.50	2.00	2.50
1.00	1.50	2.00	1.50	1.00	2.50	4.00
2.50	2.00	1.50	1.50	1.50	2.50	3.50
4.00	2.50	1.00	1.50	2.00	2.50	3.00

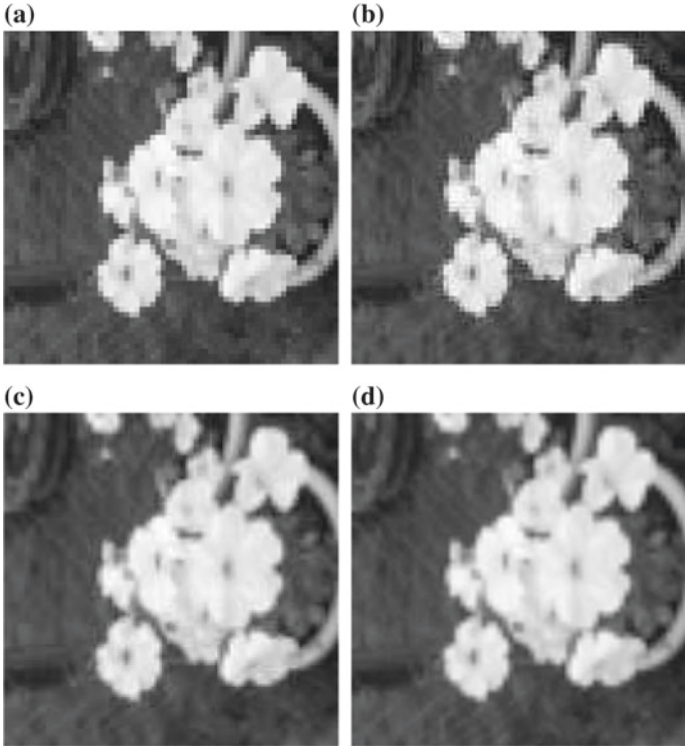


Fig. 6.1 **a** A 64 × 64 8-bit gray level image; **b** nearest-neighbor interpolated 127 × 127 image; **c** bilinear interpolated 127 × 127 image; **d** bilinear interpolated 253 × 253 image

required. Suitable border extensions (or extrapolation methods), such as that used for neighborhood operations, can be used.

A 64 × 64 8-bit gray level image is shown in Fig. 6.1a. Let us interpolate the image values at the middle and one-quarter values of the coordinates of the input image. Due to low spatial resolution, the image quality is not good. Figure 6.1b shows the nearest-neighbor 127 × 127 interpolated image. Although the size is almost doubled,

the blockiness is still visible. The bilinear interpolated 127 × 127 image, shown in Fig. 6.1c, looks smoother. Figure 6.1d shows the bilinear interpolated 253 × 253 image, and the image quality is still better. A polynomial passing through a set of neighboring points may give a more accurate interpolated values at an increased computational cost.

6.2 Affine Transform

Geometric transformations are often required in processing images. The image coordinates m and n of an image $x(m, n)$ are mapped to m' and n' such that $x'(m', n') = x(m, n)$. Typical transformations are translation, scaling, rotation, reflection and shear. All these operations are represented by the affine transformation characterized by

$$m' = am + bn + c$$
$$n' = dm + en + f$$

$$\begin{bmatrix} m' \\ n' \end{bmatrix} = \begin{bmatrix} a & b \\ d & e \end{bmatrix} \begin{bmatrix} m \\ n \end{bmatrix} + \begin{bmatrix} c \\ f \end{bmatrix} \tag{6.2}$$

The constants c and f effecting the translations or shifts can be merged to form a single transformation matrix. This form of the affine transform, called homogenous form, is

$$\begin{bmatrix} m' \\ n' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} m \\ n \\ 1 \end{bmatrix} \tag{6.3}$$

Appropriate values of the transformation matrix are to be used for each type of transformation. After the transformation of the coordinates, interpolation may be required. Affine transformation matrices for various transformations are shown in Table 6.4. The transforms can be combined using matrix multiplication. Some combinations are commutative and some are not. As multiplying a matrix by the identity matrix (with $a = e = 1$ and $b = c = d = f = 0$) leaves the matrix unchanged, an image remains the same by such a transformation.

6.2.1 Scaling

With a and e taking any positive value and $b = c = d = f = 0$, the scaling matrix is obtained, as shown in Table 6.4. As a special case of scaling, with $a = -1$ or $e = -1$ and $b = c = d = f = 0$, the reflection matrix is obtained.

Table 6.4 Affine transformation matrices

Type of transformation	Matrix	Coordinates relationship
Translation	$\begin{bmatrix} 1 & 0 & c \\ 0 & 1 & f \\ 0 & 0 & 1 \end{bmatrix}$	$m' = m + c$ $n' = n + f$
Scaling	$\begin{bmatrix} a & 0 & 0 \\ 0 & e & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$m' = am$ $n' = en$
Shear (along m-axis)	$\begin{bmatrix} 1 & b & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$m' = m + bn$ $n' = n$
Shear (along n-axis)	$\begin{bmatrix} 1 & 0 & 0 \\ d & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$m' = m$ $n' = dm + n$
Rotation, counterclockwise	$\begin{bmatrix} \cos(\beta) & -\sin(\beta) & 0 \\ \sin(\beta) & \cos(\beta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$m' = m\cos(\beta) - n\sin(\beta)$ $n' = m\sin(\beta) + n\cos(\beta)$

Consider the 256×256 gray level image shown in Fig. 6.2a. Let

$$a = 3/4,\ e = 1/2 \quad \text{then} \qquad m' = (3/4)m,\ n' = (1/2)n$$

The coordinates are multiplied by the respective scale factors. The transformation matrix and its inverse are

$$\begin{bmatrix} 3/4 & 0 & 0 \\ 0 & 1/2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 4/3 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The scaled 192×128 image is shown in Fig. 6.2b. Bilinear interpolation is used. For scaling factors less than 1, object size is reduced proportionally. For scaling factors greater than 1, object size is increased. For the same scale factors, consider the 4×4 image and its 3×2 scaled version using the nearest-neighbor interpolation.

$$\begin{bmatrix} 2 & 1 & 3 & 4 \\ 1 & 1 & 2 & 3 \\ 4 & 2 & 1 & 3 \\ 2 & 2 & 3 & 1 \end{bmatrix} \quad \begin{bmatrix} 2 & 3 \\ 1 & 2 \\ 2 & 3 \end{bmatrix}$$

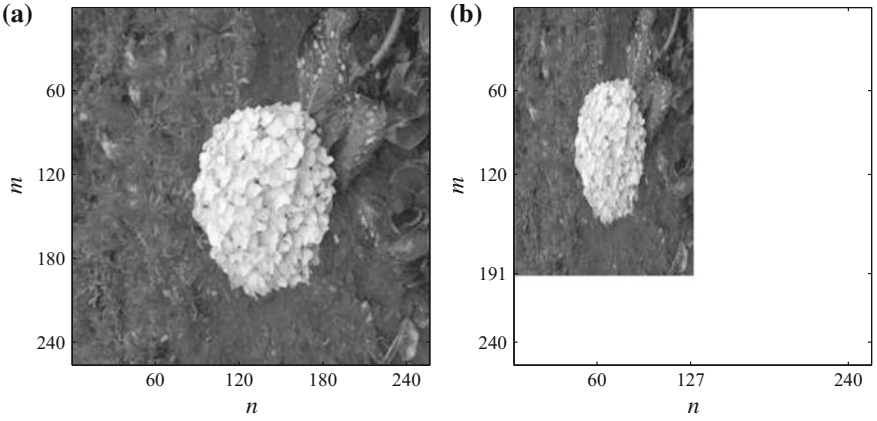


Fig. 6.2 **a** A 256 × 256 gray level image and **b** its 192 × 128 scaled version

The size of the output image will be $(3/4)4 \times (1/2)4 = 3 \times 2$. The coordinates are shown in the left matrix shown below. There are two ways find the scaled image. One is called the forward mapping and the other is called the backward mapping. The first method is to find the corresponding address in the output matrix and find the value of the image at that address. The second method, starting from the output matrix, is to find the corresponding address in the input matrix and find the value of the image at that address. The second method turns out to be the better of the two.

$$\begin{bmatrix} (0,0) & (0,1) \\ (1,0) & (1,1) \\ (2,0) & (2,1) \end{bmatrix} \quad \begin{bmatrix} (0.0,0) & (0.0,2) \\ (1.3,0) & (1.3,2) \\ (2.7,0) & (2.7,2) \end{bmatrix} \quad \begin{bmatrix} (0,0) & (0,2) \\ (1,0) & (1,2) \\ (3,0) & (3,2) \end{bmatrix}$$

Using the backward mapping (inverse of the transformation matrix), we get the middle matrix of the coordinates from that of the output. Bilinear interpolation can be used to find the corresponding values in the input matrix. Let us use the nearest-neighbor interpolation. We simply round the coordinates of the middle matrix to get the right matrix. The values corresponding to these coordinates in the input matrix are the output values. For example, $(2, 0)$ in the output matrix corresponds to $(3, 0)$ in the input matrix and the output value is 2.

6.2.2 Shear

In shearing, rows or columns are successively shifted uniformly with respect to one another. The object gets distorted by moving one side relative to another. With $a = e = 1$, $c = d = f = 0$, and b taking any value, shear occurs along the m-axis.

With $a = e = 1$, $c = b = f = 0$, and d taking any value, shear occurs along the n-axis.

Let

$$b = 0,\ d = 1 \qquad \text{then} \quad m' = m,\ n' = m + n$$

The transformation matrix and its inverse are

$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

A 4×4 image and its sheared version using the nearest-neighbor interpolation are

$$\begin{bmatrix} 2 & 1 & 3 & 4 \\ 1 & 1 & 2 & 3 \\ 4 & 2 & 1 & 3 \\ 2 & 2 & 3 & 1 \end{bmatrix} \qquad \begin{bmatrix} 2 & 1 & 3 & 4 & 0 & 0 & 0 \\ 0 & 1 & 1 & 2 & 3 & 0 & 0 \\ 0 & 0 & 4 & 2 & 1 & 3 & 0 \\ 0 & 0 & 0 & 2 & 2 & 3 & 1 \end{bmatrix}$$

The maximum value index n' takes is $3 + 3 = 6$. Therefore, the size of the output image will be 4×7. The coordinates are as shown in the matrix.

$$\begin{bmatrix} (0,0) & (0,1) & (0,2) & (0,3) & (0,4) & (0,5) & (0,6) \\ (1,0) & (1,1) & (1,2) & (1,3) & (1,4) & (1,5) & (1,6) \\ (2,0) & (2,1) & (2,2) & (2,3) & (2,4) & (2,5) & (2,6) \\ (3,0) & (3,1) & (3,2) & (3,3) & (3,4) & (3,5) & (3,6) \end{bmatrix}$$

Using the backward mapping (inverse of the transformation matrix), we get the matrix

$$\begin{bmatrix} (0,0) & (0,1) & (0,2) & (0,3) & (0,4) & (0,5) & (0,6) \\ (1,-1) & (1,0) & (1,1) & (1,2) & (1,3) & (1,4) & (1,5) \\ (2,-2) & (2,-1) & (2,0) & (2,1) & (2,2) & (2,3) & (2,4) \\ (3,-3) & (3,-2) & (3,-1) & (3,0) & (3,1) & (3,2) & (3,3) \end{bmatrix}$$

The values corresponding to these coordinates in the input matrix are the output values. For example, $(2, 0)$ in the input matrix corresponds to $(2, 2)$ in the output matrix and the output value is 4. Only those coordinates with the corresponding backward mapped coordinates located inside the input image have pixel values defined in the sheared image. All other pixel values of the 4×7 sheared image are undefined and, usually, assigned the value zero.

Consider the 256×256 gray level image shown in Fig. 6.3a. Let

$$b = 0,\ d = 0.5 \qquad \text{then} \quad m' = m,\ n' = 0.5m + n$$

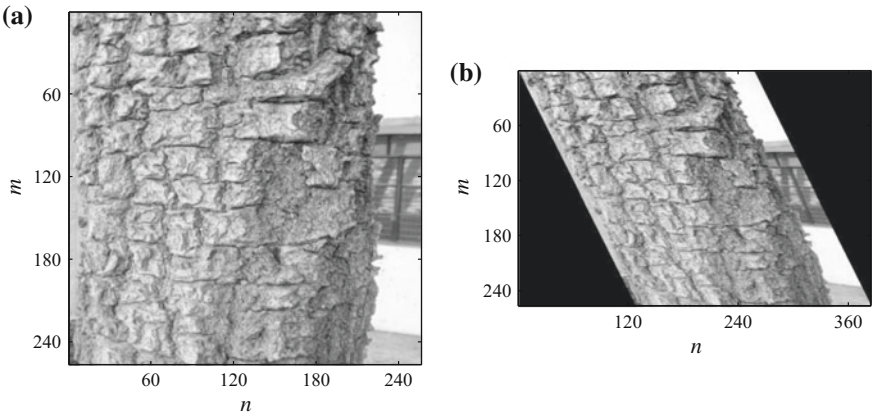


Fig. 6.3 **a** A 256 × 256 gray level image and **b** its 256 × 384 sheared version, $d = 0.5$

The transformation matrix and its inverse are

$$\begin{bmatrix} 1 & 0 & 0 \\ 0.5 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 0 \\ -0.5 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The sheared version, with $d = 0.5$, of the image is shown in Fig. 6.3b.

Let

$$b = 0.3,\ d = 0 \qquad \text{then} \quad m' = m + 0.3n,\ n' = n$$

The transformation matrix and its inverse are

$$\begin{bmatrix} 1 & 0.3 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & -0.3 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

A 4 × 4 image and its sheared version using the nearest-neighbor interpolation are

$$\begin{bmatrix} 2 & 1 & 3 & 4 \\ 1 & 1 & 2 & 3 \\ 4 & 2 & 1 & 3 \\ 2 & 2 & 3 & 1 \end{bmatrix} \quad \begin{bmatrix} 2 & 1 & 0 & 0 \\ 1 & 1 & 3 & 4 \\ 4 & 2 & 2 & 3 \\ 2 & 2 & 1 & 3 \\ 0 & 0 & 3 & 1 \end{bmatrix}$$

The maximum value index m' takes is $3 + 3(0.3) = 3.9$. Therefore, the size of the output image will be 5 × 4. The coordinates are as shown in the matrix.

$$\begin{bmatrix} (0,0) & (0,1) & (0,2) & (0,3) \\ (1,0) & (1,1) & (1,2) & (1,3) \\ (2,0) & (2,1) & (2,2) & (2,3) \\ (3,0) & (3,1) & (3,2) & (3,3) \\ (4,0) & (4,1) & (4,2) & (4,3) \end{bmatrix}$$

Using the backward mapping and then rounding, we get the matrices

$$\begin{bmatrix} (0,0) & (-0.3,1) & (-0.6,2) & (-0.9,3) \\ (1,0) & (0.7,1) & (0.4,2) & (0.1,3) \\ (2,0) & (1.7,1) & (1.4,2) & (1.1,3) \\ (3,0) & (2.7,1) & (2.4,2) & (2.1,3) \\ (4,0) & (3.7,1) & (3.4,2) & (3.1,3) \end{bmatrix} \quad \begin{bmatrix} (0,0) & (0,1) & (-1,2) & (-1,3) \\ (1,0) & (1,1) & (0,2) & (0,3) \\ (2,0) & (2,1) & (1,2) & (1,3) \\ (3,0) & (3,1) & (2,2) & (2,3) \\ (4,0) & (4,1) & (3,2) & (3,3) \end{bmatrix}$$

The values corresponding to these coordinates in the input matrix are the output values. For example, $(3, 3)$ in the input matrix corresponds to $(4, 3)$ in the output matrix and the output value is 1.

Consider the 256×256 gray level image shown in Fig. 6.3a. Let the transformation matrices be

$$\begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 1 & 0.5 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The 512×256 sheared version, with $b = 1$, of the image is shown in Fig. 6.4a. The 384×256 sheared version, with $b = 0.5$, of the image is shown in Fig. 6.4b.

6.2.3 *Translation*

With $a = e = 1$ and $b = d = 0$, and c and f taking any values is spatial shifting. The image gets shifted. Figure 6.5a, b shows a 256×256 gray level image and its shifted version. The transformation matrix is

$$\begin{bmatrix} 1 & 0 & 30 \\ 0 & 1 & 20 \\ 0 & 0 & 1 \end{bmatrix}$$

The value of a pixel in the input image at coordinates (m, n) occurs at $(m + 30,\ n + 20)$ in the shifted image. For example, the value at (0, 0) in (a) occurs at (30, 20) in (b). Shifting of an image is an often used operation. For example, shifting is required in implementing the convolution operation.

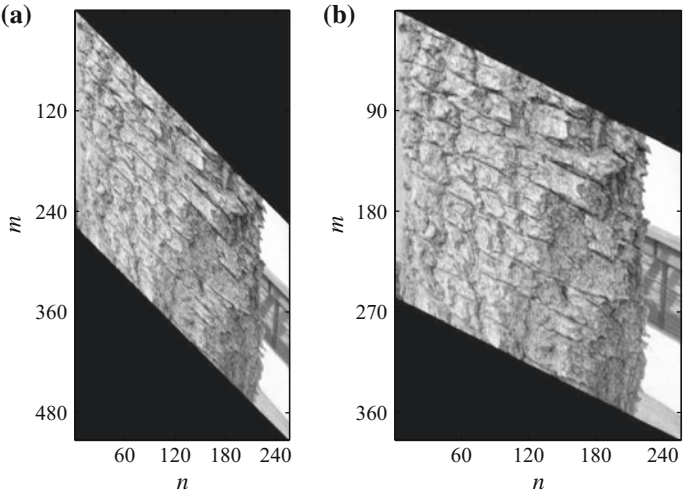


Fig. 6.4 **a** The 512×256 sheared version, $b = 1$, of the image in Fig. 6.3a, **b** the 384×256 sheared version, $b = 0.5$

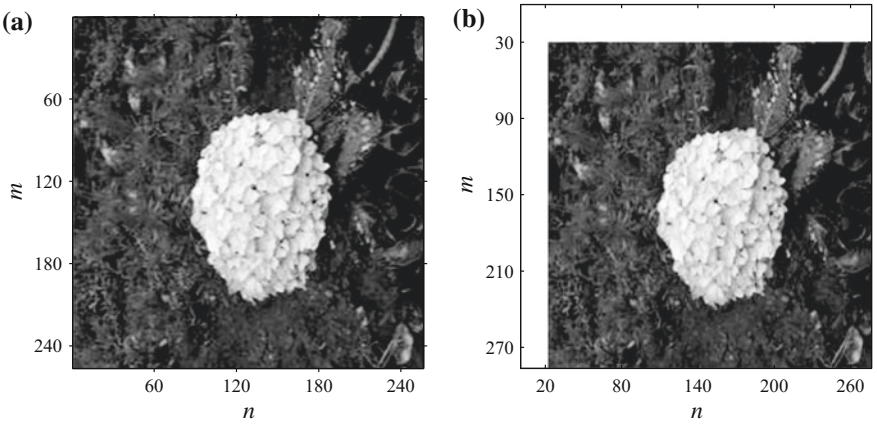


Fig. 6.5 **a** A 256×256 gray level image; **b** its shifted version

6.2.4 Rotation

In this operation, the input image is rotated by a specified angle about its center. The size of the output image is usually larger than that of the input. Consider the rotation, about the origin in the counterclockwise direction with the angle $\beta > 0$, of the vector with the coordinates (m, n) of its vertex A, shown in Fig. 6.6. The angle between the vector and the x-axis is θ. Coordinates (m, n) are given by

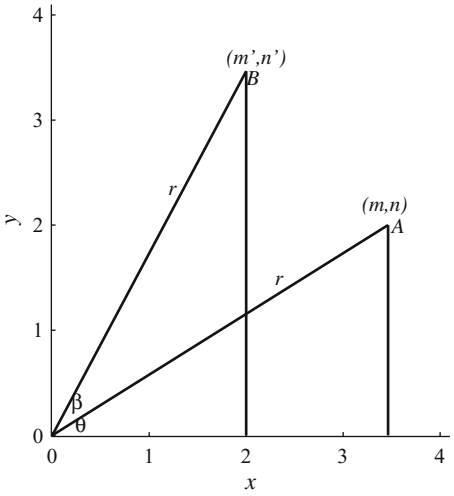


Fig. 6.6 Rotation of a vector

$$m = r\cos(\theta) \quad \text{and} \quad n = r\sin(\theta)$$

where r is the length of the vector (the distance between the origin and vertex A). Let this vector be rotated by β in the counterclockwise rotation. Now, the coordinates (m', n') of its vertex B are given by

$$m' = r\cos(\theta + \beta), \quad n' = r\sin(\theta + \beta)$$

$$m' = r\cos(\theta)\cos(\beta) - r\sin(\theta)\sin(\beta) = m\cos(\beta) - n\sin(\beta)$$

$$n' = r\sin(\theta)\cos(\beta) + r\cos(\theta)\sin(\beta) = m\sin(\beta) + n\cos(\beta)$$

In rotating the vector, its length r remains the same. Therefore, the equations governing counterclockwise rotation are

$$m' = m\cos(\beta) - n\sin(\beta) \tag{6.4}$$

$$n' = m\sin(\beta) + n\cos(\beta) \tag{6.5}$$

In matrix notation, we get

$$\begin{bmatrix} m' \\ n' \end{bmatrix} = \begin{bmatrix} \cos(\beta) & -\sin(\beta) \\ \sin(\beta) & \cos(\beta) \end{bmatrix} \begin{bmatrix} m \\ n \end{bmatrix}$$

For clockwise rotation, change the sign of the angle. That is, for transformation in the reverse order, we get

$$m = m' \cos(\beta) + n' \sin(\beta) \tag{6.6}$$
$$n = -m' \sin(\beta) + n' \cos(\beta) \tag{6.7}$$

Consider rotating the following 4×4 image, about its center.

$$\begin{bmatrix} 2 & 1 & 3 & 4 \\ 1 & 1 & 2 & 3 \\ 4 & 2 & 1 & 3 \\ 2 & 2 & 3 & 1 \end{bmatrix}$$

With $\beta = 0$, the equations reduce to $m' = m$ and $n' = n$, and there is no rotation. With $\beta = 180°$, the equations reduce to

$$m' = -m \quad \text{and} \quad n' = -n$$

The rotated image is

$$\begin{bmatrix} 1 & 3 & 2 & 2 \\ 3 & 1 & 2 & 4 \\ 3 & 2 & 1 & 1 \\ 4 & 3 & 1 & 2 \end{bmatrix}$$

which is just folding the input image about the y-axis and then folding the resulting image about the x-axis or vice versa.

With $\beta = 90°$, the equations reduce to

$$m' = -n \quad \text{and} \quad n' = m$$

The rotated image is

$$\begin{bmatrix} 4 & 3 & 3 & 1 \\ 3 & 2 & 1 & 3 \\ 1 & 1 & 2 & 2 \\ 2 & 1 & 4 & 2 \end{bmatrix}$$

which is just folding the input image about the y-axis and then taking the transpose of the result.

When we rotate an image by an angle, which is an integer multiple of 90°, the new image coordinates are trivially related to the original coordinates. The size of the image remains the same, and it is easy to find the rotated image. Rotation, by other angles, requires interpolation. A straightforward method, called forward mapping, is to use Eqs. (6.4) and (6.5) to compute the new coordinates and copy the pixel values to the new locations. This method has too many problems and, therefore, is

not practically useful. The second method, called backward mapping, is to select a set of required integer coordinates for (m', n'), use Eqs. (6.4) and (6.5) to find the corresponding (m, n) and copy the pixel values. However, coordinates (m, n) are usually real-valued, and interpolation is required to find the appropriate pixel values. For example, with $\beta = 45°$, we get the reverse transformation by substituting $\beta = -45°$ in Eqs. (6.4) and (6.5) and solving.

$$m = \frac{1}{\sqrt{2}}(m' - n') \quad \text{and} \quad n = \frac{1}{\sqrt{2}}(m' + n') \tag{6.8}$$

A 4×4 input image and its 5×5 rotated version (using the nearest-neighbor interpolation), respectively, are

$$\begin{bmatrix} 9 & 1 & 3 & 7 \\ 6 & 8 & 0 & 4 \\ 5 & 4 & 6 & 1 \\ 2 & 7 & 3 & 5 \end{bmatrix} \qquad \begin{bmatrix} 0 & 0 & 7 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 9 & 8 & 4 & 6 & 5 \\ 0 & 5 & 4 & 7 & 0 \\ 0 & 0 & 2 & 0 & 0 \end{bmatrix}$$

Remember that the rotation is about the center of the image, and the center of the input image is in the center of the square formed by the locations of the pixels $\{8, 0, 4, 6\}$. The central pixel of the rotated image, 4, is shown in boldface.

Assuming either m' or n' is equal to zero and the other variable assuming values $\{0, 1, 2, 3\}$, from Eq. (6.8), we obtain integer multiples of $\pm 1/\sqrt{2}$. For example,

$$\{(0, 1, 2, 3)/\sqrt{2}\} = \{0, 0.71, 1.41, 2.12\}$$

As the maximum distance from the center of the 4×4 input image to its border is 1.5, the limit of the address is 1.41. We conclude that the coordinates of the rotated image are restricted to the values $\{-2, -1, 0, 1, 2\}$, and the size of the rotated image is 5×5. The corresponding output image coordinates are

$$\begin{bmatrix} (2,-2) & (2,-1) & (2,0) & (2,1) & (2,2) \\ (1,-2) & (1,-1) & (1,0) & (1,1) & (1,2) \\ (0,-2) & (0,-1) & (0,0) & (0,1) & (0,2) \\ (-1,-2) & (-1,-1) & (-1,0) & (-1,1) & (-1,2) \\ (-2,-2) & (-2,-1) & (-2,0) & (-2,1) & (-2,2) \end{bmatrix}$$

Using Eq. (6.8), with m' and n' varying from -2 to 2 with increment 1, we get the backward mapped coordinates, rounded to 2 digits after the decimal point, as

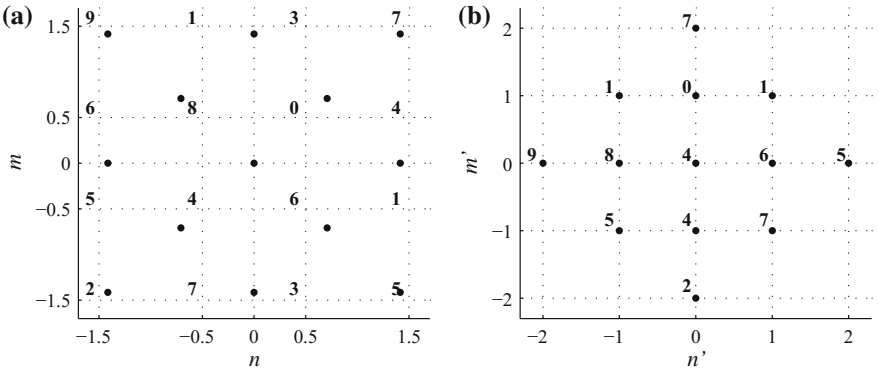


Fig. 6.7 Rotation of an image by 45° in the counterclockwise direction. **a** A 4 × 4 input image with reverse coordinates shown by dots; **b** its 5 × 5 rotated version

$$\begin{bmatrix} (2.83, 0.00) & (2.12, 0.71) & (1.41, 1.41) & (0.71, 2.12) & (0.00, 2.83) \\ (2.12, -0.71) & (1.41, 0.00) & (0.71, 0.71) & (0.00, 1.41) & (-0.71, 2.12) \\ (1.41, -1.41) & (0.71, -0.71) & (0.00, 0.00) & (-0.71, 0.71) & (-1.41, 1.41) \\ (0.71, -2.12) & (0.00, -1.41) & (-0.71, -0.71) & (-1.41, 0.00) & (-2.12, 0.71) \\ (0.00, -2.83) & (-0.71, -2.12) & (-1.41, -1.41) & (-2.12, -0.71) & (-2.83, 0.00) \end{bmatrix}$$

Only those coordinates with the corresponding backward mapped coordinates with magnitude less than or equal to 1.5 have pixel values defined in the rotated image. All other pixel values of the 5 × 5 rotated image are undefined and, usually, assigned the value zero. The 4 × 4 input image, with the backward mapped coordinates shown by dots, is shown in Fig. 6.7a. The 5 × 5 rotated image is shown in Fig. 6.7b. The pixel value nearest to the backward mapped coordinates in the input image is placed in the corresponding location in the rotated image. This method is called the nearest-neighbor interpolation algorithm. The coordinates (2, 0) in the output image backward map to (1.41, 1.41) in the input image. The nearest pixel value is 7 and that value is placed at (2, 0). The values along the main diagonal {9, 8, 6, 5} of the input image are placed on the horizontal axis through the origin. The values along the other diagonal {7, 0, 4, 2} are placed on the vertical axis through the origin. At coordinates (0, 0), the value 4 is chosen to be the nearest value.

In the bilinear interpolation algorithm, the 4 pixel values around the backward mapped coordinates are used to find the pixel value in the corresponding location in the rotated image. The coordinates (1, 0) in the output image backward map to (0.7071, 0.7071) in the input image. The 4 neighbors are (0.5, 0.5), (1.5, 0.5), (1.5, 1.5), and (0.5, 1.5) with pixel values 0, 3, 7, and 4, respectively. Interpolation value due to 0 and 3 is $3(0.7071 - 0.5) = 3(0.2071) = 0.6213$. Interpolation value due to 4 and 7 is $4(1 - 0.2071) + 7(0.2071) = 4.6213$. Interpolation value due to 0.6213 and 4.6213 is $0.6213(1 - 0.2071) + 4.6213(0.2071) = 1.4497$. The input image and the rotated images using nearest-neighbor and bilinear interpolation are, respectively,

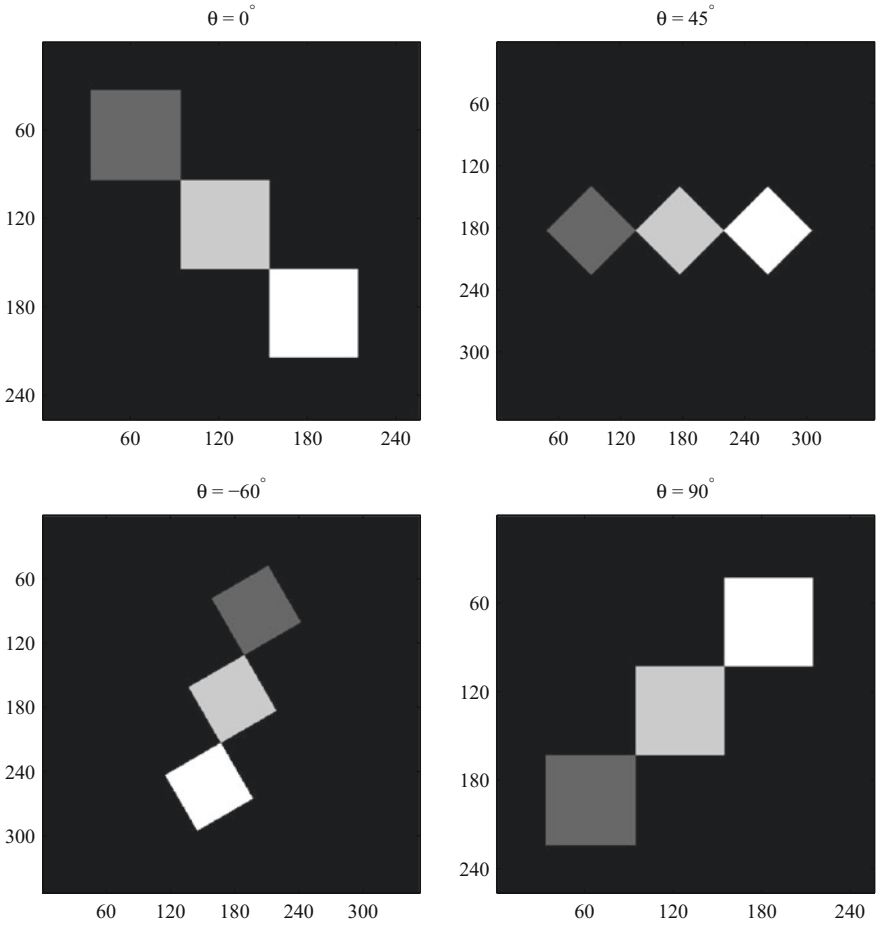


Fig. 6.8 Rotation of a 256 × 256 image by 0°, 45°, −60°, and 90° in the counterclockwise direction

$$\begin{bmatrix} 9 & 1 & 3 & 7 \\ 6 & 8 & 0 & 4 \\ 5 & 4 & 6 & 1 \\ 2 & 7 & 3 & 5 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 7 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 9 & 8 & 4 & 6 & 5 \\ 0 & 5 & 4 & 7 & 0 \\ 0 & 0 & 2 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 6.4 & 0 & 0 \\ 0 & 2.17 & 1.45 & 2.54 & 0 \\ 8.13 & 6.56 & 4.50 & 4.64 & 4.54 \\ 0 & 5.54 & 4.57 & 5.00 & 0 \\ 0 & 0 & 2.64 & 0 & 0 \end{bmatrix}$$

Other interpolation algorithms can also be used. For rotation other than about the center, translate the image to the origin, rotate and then translate back. Rotation of a 256 × 256 image by 0°, 45°, −60° and 90° is shown in Fig. 6.8.

Some properties of the affine transform are: (i) straight lines are mapped to straight lines, (ii) triangles are mapped to triangles, (iii) parallel lines remain parallel after transformation, (iv) rectangles are mapped to parallelograms, and (v) ratios are always preserved. For example, midpoints map to midpoints.

6.3 Correlation

Correlation operation is a similarity measure between two signals. It is a probabilistic relationship. The number of accidents occurred due to a driver during a period is, on the average, likely to be directly proportional to the amount of his consumption of alcohol before he drives. Correlation between the number of accidents and his alcohol consumption will be high if the number of accidents is high and vice versa. On the other hand, there will be little or no correlation between the number of accidents occurred due to a driver and the consumption of alcohol by another driver. While the use of convolution and correlation is different, both are essentially sum of products (with a small difference) and, therefore, their implementation is similar.

6.3.1 1-D Correlation

The cross-correlation of two signals $x(n)$ and $y(n)$ is defined as

$$r_{xy}(m) = \sum_{n=-\infty}^{\infty} x(n)y(n-m), \quad m = 0, \pm 1, \pm 2, \ldots$$

(An alternate cross-correlation definition is

$$r_{xy}(m) = \sum_{n=-\infty}^{\infty} x(n)y(n+m), \quad m = 0, \pm 1, \pm 2, \ldots$$

The outputs of the two definitions are time reversed version of each other). The correlation of $x(n)$ and $y(n)$ is a function of the delay time. For example, our hungriness is proportional to the delay time after our last meal. Therefore, the independent variable in the correlation function is the time-lag or time-delay variable m, which has the dimension of time (delay time). The independent time variable n indicates the running time. If the two signals are similar, then the values of $r_{xy}(m)$ will be large and vice versa. The correlation of $\{y(n), n = 0, 1\} = \{3, -2\}$ and $\{x(n), n = 0, 1, 2, 3\} = \{2, 1, 3, 4\}$ is shown in Fig. 6.9. The convolution operation without time reversal is the correlation operation. The correlation of a signal by itself is the autocorrelation. The autocorrelation of {3, 1, 2, 4} is {12, 10, 13, 30, 13, 10, 12}.

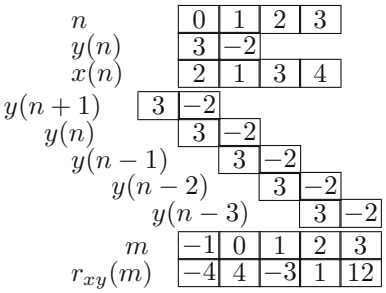


Fig. 6.9 1-D linear correlation

6.3.2 2-D Correlation

In the 2-D correlation, a 2-D window or template is moved over the image. Template is a pattern to be matched. The correlation of images $x(m, n)$ and $y(m, n)$ is defined as

$$r_{xy}(m, n) = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} x(k, l)y(k - m, l - n)$$

Consider the 3×3 template image $y(k, l)$ and the 4×4 image $x(k, l)$ shown in Fig. 6.10.

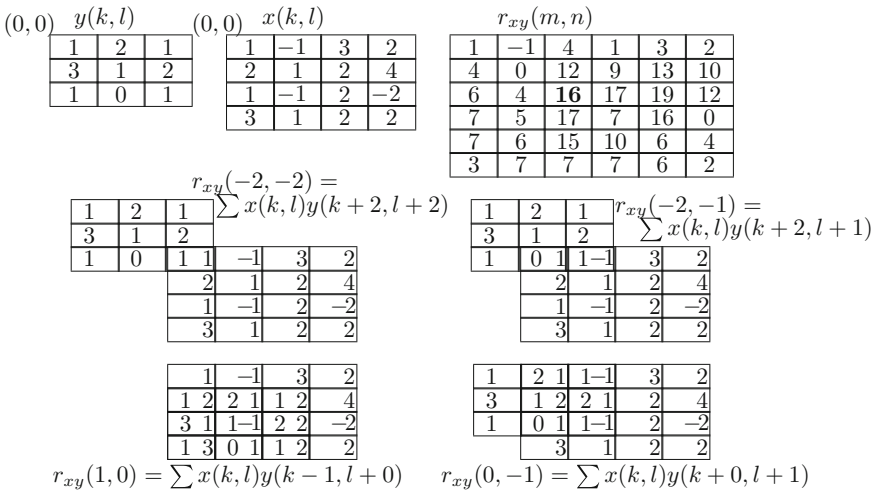


Fig. 6.10 2-D linear correlation

$$y(k,l) = \begin{bmatrix} 1 & 2 & 1 \\ 3 & 1 & 2 \\ 1 & 0 & 1 \end{bmatrix} \qquad x(k,l) = \begin{bmatrix} 1 & -1 & 3 & 2 \\ 2 & 1 & 2 & 4 \\ 1 & -1 & 2 & -2 \\ 3 & 1 & 2 & 2 \end{bmatrix}$$

Three operations, similar to those of the convolution, are repeatedly executed in carrying out the 2-D correlation.

1. Template $y(k,l)$ is shifted by (m,n) to get $y(k-m, l-n)$.
2. The products $x(k,l)y(k-m,l-n)$ of all the overlapping samples are found.
3. The sum of all the products yields the correlation output $r_{xy}(m,n)$ at (m,n).

Four examples of computing the correlation output are shown. For example, with a shift of $(k+2, l+2)$, there is only one overlapping pair (1, 1). The product of these numbers yields the output $r_{xy}(-2,-2) = 1$. $r_{xy}(0,0) = 16$ is shown in boldface. The process is repeated to get the complete correlation output $r_{xy}(m,n)$. It is assumed that the pixel values outside the defined region of the image are zero. This assumption may not be suitable. Usually, some suitable assumption, such as periodicity, symmetry or replication at the borders of the image, is made in processing images.

The autocorrelation of $x(k,l)$ is

$$x(k,l) = \begin{bmatrix} 2 & 2 & 3 \\ 3 & -1 & 2 \\ 1 & 2 & 1 \end{bmatrix} \qquad r_{xx}(m,n) = \begin{bmatrix} 2 & 6 & 9 & 8 & 3 \\ 7 & 7 & 13 & 6 & 11 \\ 13 & 9 & 37 & 9 & 13 \\ 11 & 6 & 13 & 7 & 7 \\ 3 & 8 & 9 & 6 & 2 \end{bmatrix}$$

The normalized cross-correlation (correlation coefficient) of images $x(m,n)$ and $y(m,n)$ is defined as

$$rn_{xy}(m,n) = \frac{\sum_{k=-\infty}^{\infty}\sum_{l=-\infty}^{\infty}(x(k,l)-\bar{x}_l)(y(k-m,l-n)-\bar{y})}{\sqrt{\sum_{k=-\infty}^{\infty}\sum_{l=-\infty}^{\infty}(x(k,l)-\bar{x}_l)^2 \sum_{k=-\infty}^{\infty}\sum_{l=-\infty}^{\infty}(y(k-m,l-n)-\bar{y})^2}}$$

As the mean is subtracted, all the values of the template cannot be the same. If the variance of the image over the overlapping portion with the template is zero, then the correlation coefficient is assigned the value zero. The range of values vary from -1 to 1. A high value indicates a good match between the template and the image. The major difference in this version of correlation is that the local image and template values are the difference between the given values and their local mean $\bar{x}_l$ and $\bar{y}$. That is, the fluctuating part of the values of the operands are analyzed. The numerator is cross-correlation with the means subtracted. The denominator is a normalizing factor. It is the square root of the product of the variances of the overlapping samples of the operands. For the same $y(k,l)$ and $x(k,l)$ as in the last example, let us compute the $rn_{xy}(m,n)$. Subtracting the mean, 1.3333 from $y(m,n)$, we get

$$ym(m,n) = \begin{bmatrix} 1 & 2 & 1 \\ 3 & 1 & 2 \\ 1 & 0 & 1 \end{bmatrix} - \begin{bmatrix} 1.3333 & 1.3333 & 1.3333 \\ 1.3333 & 1.3333 & 1.3333 \\ 1.3333 & 1.3333 & 1.3333 \end{bmatrix} = \begin{bmatrix} -0.3333 & 0.6667 & -0.3333 \\ 1.6667 & -0.3333 & 0.6667 \\ -0.3333 & -1.3333 & -0.3333 \end{bmatrix}$$

The variance of $ym(m,n)$ is 6.
Subtracting the mean, 1.1111 from part of $x(m,n)$ for a neighborhood, we get

$$xm(m,n) = \begin{bmatrix} 1 & -1 & 3 \\ 2 & 1 & 2 \\ 1 & -1 & 2 \end{bmatrix} - \begin{bmatrix} 1.1111 & 1.1111 & 1.1111 \\ 1.1111 & 1.1111 & 1.1111 \\ 1.1111 & 1.1111 & 1.1111 \end{bmatrix} = \begin{bmatrix} -0.1111 & -2.1111 & 1.8889 \\ 0.8889 & -0.1111 & 0.8889 \\ -0.1111 & -2.1111 & 0.8889 \end{bmatrix}$$

The variance of $xm(m,n)$ is 14.8889. The sum of pointwise product of $ym(m,n)$ and $xm(m,n)$ is 2.6667. Now, $2.6667/\sqrt{(6)(14.8889)} = 0.2821$. Let us try another neighborhood. Subtracting the mean, 1.2222 from part of $x(m,n)$, we get

$$xm(m,n) = \begin{bmatrix} 1 & 2 & 4 \\ -1 & 2 & -2 \\ 1 & 2 & 2 \end{bmatrix} - \begin{bmatrix} 1.2222 & 1.2222 & 1.2222 \\ 1.2222 & 1.2222 & 1.2222 \\ 1.2222 & 1.2222 & 1.2222 \end{bmatrix} = \begin{bmatrix} -0.2222 & 0.7778 & 2.7778 \\ -2.2222 & 0.7778 & -3.2222 \\ -0.2222 & 0.7778 & 0.7778 \end{bmatrix}$$

The variance of $xm(m,n)$ is 25.5556. The sum of pointwise product of $ym(m,n)$ and $xm(m,n)$ is -7.6667. Now, $-7.6667/\sqrt{(6)(25.5556)} = -0.6191$. The complete correlation coefficients, for the example, are

$$\begin{bmatrix} -0.1443 & -0.6170 & -0.3203 & -0.6121 & -0.5052 & -0.1443 \\ 0.0615 & -0.5862 & 0.0969 & -0.3479 & -0.1310 & 0.2979 \\ 0.0569 & -0.1826 & 0.2821 & 0.2610 & 0.2496 & 0.5533 \\ -0.0700 & -0.4811 & -0.0426 & -0.6191 & 0.1601 & -0.2128 \\ 0.1837 & 0.1543 & 0.6056 & 0.6508 & 0.2887 & 0.7217 \\ -0.1443 & 0.1837 & -0.0700 & 0.0259 & 0.1091 & -0.1443 \end{bmatrix}$$

Correlation coefficients are very effective in finding the location of a template in an image. Figure 6.11a shows a 256 × 256 8-bit image. We want to find the locations of the four bolts near the center of the wheel. The enlarged 17 × 13 8-bit template is shown in Fig. 6.11b. The four brightest points in Fig. 6.11c, which show the correlation coefficients image between the image and the template, clearly indicate the locations of the four bolts. Figure 6.11d shows the un-normalized correlation output between the image and the template, which does not point out the location of the bolts.

6.4 Image Registration

Image registration is important in applications such as the study of satellite and medical images. Different images of the same scene taken at different times and settings or by different equipments are aligned and represented in the same coordinate

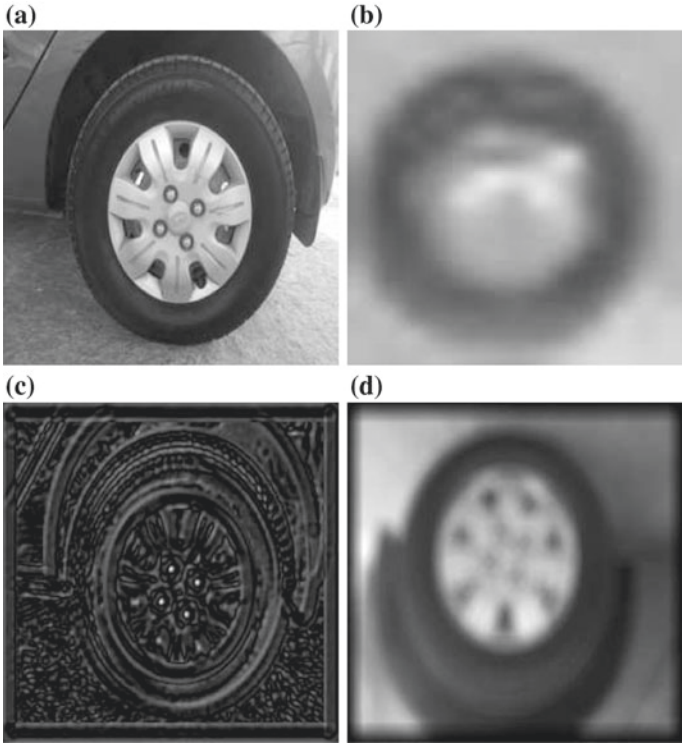


Fig. 6.11 **a** A 256 × 256 8-bit image; **b** an enlarged 17 × 13 8-bit template; **c** the correlation coefficients image; **d** the un-normalized correlation output

system to study the changes in the behavior of the objects in the scene. For example, the study of the growth of plants in remote sensing and tumor growth in medical analysis is important.

Image registration consists of the following four basic steps.

Feature detection Features, such as lines and corners, are to be detected.
Feature matching The detected features are to be matched with those of the reference image.
Selection of geometric transformation Suitable geometrical transformations are selected to align the images.
Application of the transformation The transformations are applied to obtain the aligned images.

The extent the corresponding features match is the measure of good registration.

Figure 6.12a shows a 256 × 256 8-bit image, which is a 60° counterclockwise rotated version of that in Fig. 6.11a. The features to be matched are the four bolts. The four brightest points in Fig. 6.12b, which show the correlation coefficients image between the image and the template, clearly indicate the locations of the four bolts.

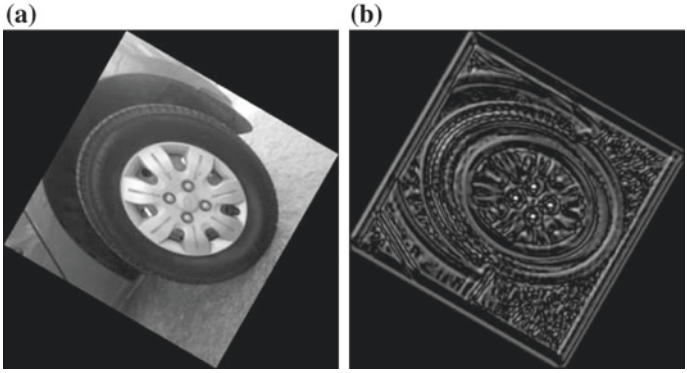


Fig. 6.12 **a** A 256 × 256 8-bit image; **b** the correlation coefficients image

The locations of the bolts in Figs. 6.11c and 6.12b, however, are different. The difference has to analyzed and affine transformation for a 60° clockwise rotation has to be selected and applied. Then, the images are aligned and a comparative study can be made. In practice, a combination of transformations may be required to align the images.

6.5 Summary

- Geometric operations, such as shifting and rotation, are required in image processing, in addition to the arithmetic operations of the pixel values.
- Geometric operations change the coordinates of the image.
- Due to the constraint that the image coordinates have to be finite integer values, geometric operations invariably require interpolation.
- Interpolation is estimating the values of a function inside the range of the given values.
- Interpolation is required for reconstruction of images from its samples. In addition, it is also required in operations such as image rotation, conversion of images from polar representation to rectangular coordinates and vice versa.
- Two of the often used interpolation methods are the nearest-neighbor and the bilinear.
- Operations such as translation, scaling, rotation, and shearing are often used in image processing. These operations change the coordinates of the image, but the pixel values are unaffected. They are geometrical transformations. They map points from one coordinate system into another.
- These operations, called the affine transformation, are formulated as matrix multiplication with suitable transformation matrices.

- Correlation operation, which is a similarity measure, is often used in processing images.
- Correlation operation can detect the presence of an object in an image and, if present, it gives the location of the object. From the implementation point of view, it is the same thing as convolution without the time-reversal step.
- Image registration is the alignment of images of a scene taken at different times, settings or with different equipments. Registration helps to study the behavior of the objects in the scene at different times and settings.
- The selected features of an image are computed and compared with those of the reference image and suitable geometrical transformations are used to align the images.

Exercises

6.1 Using nearest-neighbor interpolation, find the 7×7 interpolated version of the image $x(m, n)$.

(i)

$$\begin{bmatrix} 43 & 50 & 50 & 52 \\ 45 & 49 & 51 & 50 \\ 46 & 46 & 49 & 48 \\ 43 & 44 & 47 & 42 \end{bmatrix}$$

(ii)

$$\begin{bmatrix} 68 & 80 & 83 & 78 \\ 39 & 54 & 61 & 66 \\ 41 & 44 & 44 & 67 \\ 55 & 46 & 34 & 66 \end{bmatrix}$$

(iii)

$$\begin{bmatrix} 63 & 64 & 64 & 64 \\ 62 & 62 & 62 & 61 \\ 61 & 61 & 61 & 58 \\ 62 & 61 & 60 & 58 \end{bmatrix}$$

6.2 Using bilinear interpolation, find the 7×7 interpolated version of the image $x(m, n)$.

(i)

$$\begin{bmatrix} 34 & 51 & 56 & 53 \\ 38 & 53 & 57 & 54 \\ 40 & 52 & 56 & 52 \\ 39 & 48 & 52 & 49 \end{bmatrix}$$

*(ii)

$$\begin{bmatrix} 53 & 42 & 39 & 58 \\ 51 & 46 & 44 & 49 \\ 54 & 52 & 58 & 46 \\ 63 & 57 & 63 & 52 \end{bmatrix}$$

(iii)

$$\begin{bmatrix} 82 & 84 & 86 & 97 \\ 80 & 80 & 85 & 103 \\ 79 & 77 & 90 & 114 \\ 80 & 84 & 102 & 118 \end{bmatrix}$$

6.3 Using nearest-neighbor interpolation, find the scaled version of the image $x(m, n)$.
(i) $a = 0.5, e = 0.75$

$$\begin{bmatrix} 52 & 61 & 57 & 66 \\ 58 & 64 & 69 & 64 \\ 45 & 60 & 74 & 61 \\ 56 & 63 & 74 & 63 \end{bmatrix}$$

*(ii) $a = -0.5, e = -0.5$

$$\begin{bmatrix} 71 & 56 & 47 & 92 \\ 66 & 51 & 47 & 108 \\ 64 & 55 & 70 & 122 \\ 73 & 57 & 81 & 127 \end{bmatrix}$$

(iii) $a = 0.75, e = 0.25$

$$\begin{bmatrix} 48 & 53 & 46 & 62 \\ 54 & 54 & 54 & 80 \\ 64 & 53 & 59 & 78 \\ 57 & 46 & 56 & 55 \end{bmatrix}$$

6.4 Using nearest-neighbor interpolation, find the sheared version of the image $x(m, n)$.
(i) $b = 0, d = 1$

$$\begin{bmatrix} 17 & 20 & 26 & 25 \\ 18 & 23 & 30 & 24 \\ 17 & 24 & 32 & 27 \\ 20 & 28 & 30 & 32 \end{bmatrix}$$

(ii) $b = 0, d = 0.5$

$$\begin{bmatrix} 63 & 49 & 51 & 54 \\ 66 & 60 & 52 & 56 \\ 57 & 62 & 62 & 57 \\ 57 & 56 & 64 & 61 \end{bmatrix}$$

(iii) $b = 0, d = 0.3$

$$\begin{bmatrix} 179 & 178 & 179 & 184 \\ 177 & 178 & 179 & 189 \\ 176 & 177 & 180 & 193 \\ 174 & 175 & 184 & 190 \end{bmatrix}$$

6.5 Using nearest-neighbor interpolation, find the sheared version of the image $x(m, n)$.

(i) $b = 1, d = 0$

$$\begin{bmatrix} 41 & 36 & 123 & 151 \\ 27 & 10 & 79 & 136 \\ 17 & 17 & 33 & 91 \\ 17 & 30 & 17 & 70 \end{bmatrix}$$

(ii) $b = 0.7, d = 0$

$$\begin{bmatrix} 172 & 157 & 115 & 62 \\ 163 & 165 & 118 & 83 \\ 138 & 185 & 128 & 71 \\ 121 & 184 & 126 & 83 \end{bmatrix}$$

*(iii) $b = 0.3, d = 0$

$$\begin{bmatrix} 95 & 111 & 48 & 32 \\ 96 & 115 & 59 & 26 \\ 89 & 90 & 37 & 24 \\ 86 & 73 & 15 & 21 \end{bmatrix}$$

6.6 Using nearest-neighbor interpolation, find the rotated version of the image $x(m, n)$ in the counterclockwise direction. $\theta = 90°$, $\theta = 180°$ and $\theta = 45°$.

(i)

$$\begin{bmatrix} 95 & 47 & 65 & 55 \\ 74 & 60 & 60 & 47 \\ 105 & 103 & 67 & 46 \\ 103 & 78 & 67 & 58 \end{bmatrix}$$

(ii)

$$\begin{bmatrix} 74 & 81 & 67 & 75 \\ 77 & 77 & 77 & 83 \\ 58 & 69 & 69 & 80 \\ 46 & 61 & 69 & 82 \end{bmatrix}$$

(iii)

$$\begin{bmatrix} 66 & 77 & 79 & 64 \\ 56 & 68 & 61 & 69 \\ 43 & 51 & 49 & 66 \\ 39 & 45 & 43 & 55 \end{bmatrix}$$

6.7 Find the cross-correlation and the correlation coefficients of $x(m, n)$ and $h(m, n)$. Assume zero-padding at the borders.

$$h(m,n) = \begin{bmatrix} 1 & 1 & 1 \\ 3 & 1 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

*(i)

$$x(m,n) = \begin{bmatrix} 2 & 1 & 3 & 2 \\ 3 & 2 & 1 & 0 \\ 1 & 2 & 1 & 2 \\ 1 & 3 & 0 & 2 \end{bmatrix}$$

(ii)

$$x(m,n) = \begin{bmatrix} 3 & 1 & 1 & 2 \\ 0 & 2 & 1 & 2 \\ 1 & 1 & 1 & 2 \\ 1 & 2 & 0 & 2 \end{bmatrix}$$

(iii)

$$x(m,n) = \begin{bmatrix} 1 & 2 & 3 & 0 \\ 0 & 2 & 1 & 0 \\ 2 & 2 & 1 & 2 \\ 1 & 1 & 0 & 2 \end{bmatrix}$$

6.8 Find the autocorrelation of $x(m, n)$. Assume zero-padding at the borders.

(i)

$$x(m,n) = \begin{bmatrix} 2 & 1 & 3 & 2 \\ 2 & 1 & 1 & 0 \\ 1 & 0 & 1 & 2 \\ 1 & 1 & 0 & 2 \end{bmatrix}$$

(ii)

$$x(m,n) = \begin{bmatrix} 3 & 1 & 1 & 2 \\ 0 & 3 & 1 & 2 \\ 1 & 2 & 1 & 0 \\ 1 & 2 & 0 & 2 \end{bmatrix}$$

(iii)

$$x(m,n) = \begin{bmatrix} 3 & 2 & 1 & 0 \\ 0 & 2 & 1 & 0 \\ 2 & 0 & 1 & 2 \\ 3 & 1 & 0 & 2 \end{bmatrix}$$

Chapter 7
Image Reconstruction from Projections

Abstract The Radon transform is presented, which is important in computerized tomography in medical and industrial applications. This transform enables to produce the image of an object, without intrusion, using its projections at various directions. As this transform uses the normal form of a line, it is presented first. Then, the Radon transform and its properties are described next. The Fourier-slice theorem and the filtered back-projection of the images are presented. Finally, line detection using the Hough transform is given. Examples are included to illustrate the various concepts.

In such medical applications as computerized axial tomography and nondestructive testing of mechanical objects, the Radon transform, which is an advanced spectral method, plays an important role. Computerized tomography is in widespread use, and even small hospitals are equipped with such systems. Axial tomography is forming the image of the interior of an object, such as the human body, using the projections of the object along several directions. Projections are obtained, for example, by the absorption of the X-ray in passing through the object from the source to the detector. The source and the detector assembly are rotated around the object to get the projections at a finite set of angles. The advantage is that it facilitates nonintrusive medical diagnosis.

In image processing, appropriate representation of an image that is suitable for the specific task is important. While images occur mostly in the continuous form, we convert them to digital form for processing convenience. Further, the image is often represented in the transform domain. For example, in Fourier analysis, an image is represented as a linear combination of sinusoidal surfaces. In the Radon transform, an image is represented by its mappings with respect to a set of lines at various angles represented by polar coordinates. The values at various polar coordinates are the transform coefficients. An image can be represented using either the Cartesian coordinates or polar. The higher the frequency content of the image, the more is the number of samples required in either representation. In the limit, with infinite samples, either representation is the same as the image. Of course, in practice, a finite number of samples only can be used and it has to be ensured that the number of samples taken represents an image with adequate accuracy. When an image is represented in Radon transform form, we are able to form the image of the interior

D. Sundararajan, *Digital Image Processing*, DOI 10.1007/978-981-10-6113-4_7

of an object with out intrusion. In its implementation, we use the 1-D DFT and interpolation operations.

In this chapter, we study the formation of images using tomography. As projection (representation of an object on a plane as it is seen from a particular direction) is the basis of the Radon transform, we need to study the normal form of a line first. This form of line is also used in Hough transform to detect lines in images.

7.1 The Normal Form of a Line

In mathematics, a line is an object which has length but no breadth. The slope of a line is a measure of its steepness. We are quite familiar with the slope–intercept form of a line given by

$$y = mx + c \tag{7.1}$$

where m is the slope of the line, and c is its y-axis intercept. The equation is linear, as the graph of any linear equation is a line. The y-axis intercept is the point where the line intersects the y-axis and is obtained by setting $x = 0$ in Eq. (7.1). The x-axis intercept is the point where the line intersects the x-axis and is obtained by setting $y = 0$ in Eq. (7.1) and solving for x. The intercept form of a line is given by

$$\frac{x}{b} + \frac{y}{c} = 1$$

where b and c are, respectively, the x-axis and y-axis intercepts of the line. Solving for y, we get back the slope–intercept form

$$y = -\frac{c}{b}x + c$$

where $-c/b$ is the slope.

The polar coordinates are advantageous in some applications than the more common Cartesian coordinates. For example, complex number multiplication and division is easier in the polar form. A point with Cartesian coordinates (x, y) can be equivalently represented by its polar coordinates (s, θ). The angle between the positive side of the x-axis and the line joining the point with the origin is θ. The length of the line is s. The relationships between the two coordinates are given by

$$x = s\cos(\theta), \quad y = s\sin(\theta) \quad \text{and} \quad s = \sqrt{x^2 + y^2}, \quad \tan(\theta) = \frac{y}{x},\ x \neq 0$$

A line can also be represented using polar coordinates. The Radon and Hough transforms use the normal form a line. In this form, a line is expressed in terms of its perpendicular distance from the line to the origin and the angle subtended between the perpendicular line and the x-axis. Let L be a line, as shown in Fig. 7.1.

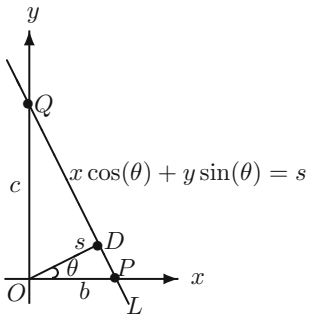


Fig. 7.1 The normal form of a line

The intercepts of the line with the x-axis and y-axis are b and c, respectively. Line OD is perpendicular to line L with length s and at angle θ from the x-axis. That is, $\angle ODP = 90°$.

Then,

$$b\cos(\theta) = s \quad \text{and} \quad c\sin(\theta) = s$$

Substituting $b = s/\cos(\theta)$ and $c = s/\sin(\theta)$ in the intercept form, the normal form of a line is given by

$$x\cos(\theta) + y\sin(\theta) = s \tag{7.2}$$

where s is always positive and $0 \leq \theta < 360°$. This form is preferred, since the lines are evenly distributed in the parameter space. All the points of a line are transformed to a single point in the (s, θ) plane. An alternate derivation of a line in the normal form is as follows. The coordinates of point D are $(s\cos(\theta), s\sin(\theta))$. The slope of the line OD is $\tan(\theta) = \sin(\theta)/\cos(\theta)$. The slope of the line perpendicular to OD is $-\cos(\theta)/\sin(\theta)$. Therefore, the equation of the line L is

$$\frac{y - s\sin(\theta)}{x - s\cos(\theta)} = -\frac{\cos(\theta)}{\sin(\theta)}$$

Simplifying the expression, we get Eq. (7.2).

Given a linear equation

$$\sqrt{3}x + y + 2 = 0,$$

let us put it in the normal form of a line. Shift the constant term to the other side and ensure that it is positive. We get

$$-\sqrt{3}x - y = 2$$

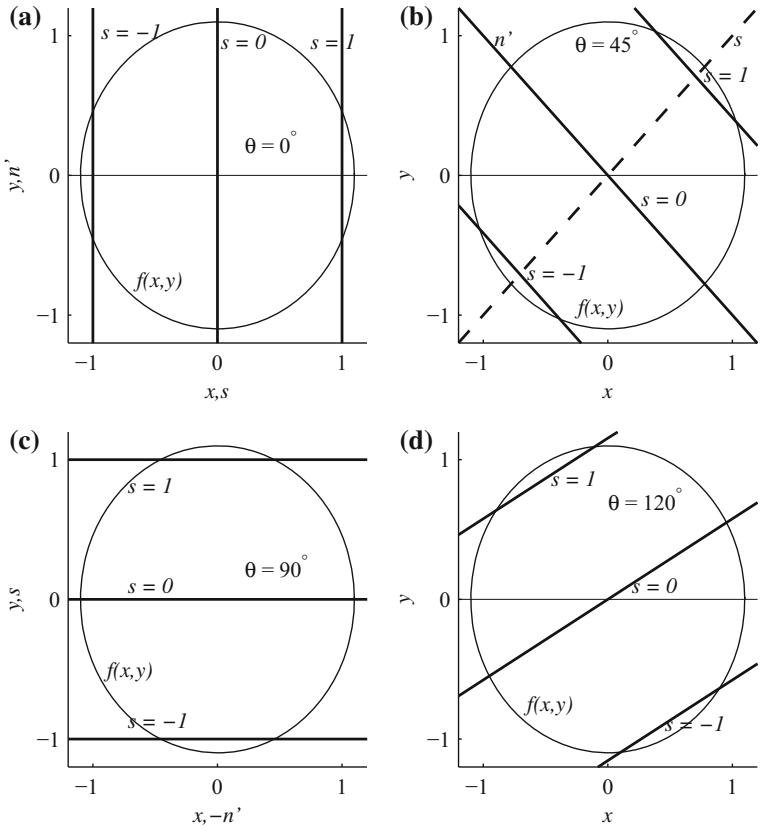


Fig. 7.2 Lines with s varying from -1 to 1 and **a** $\theta = 0°$; **b** $\theta = 45°$; **c** $\theta = 90°$; and **d** $\theta = 120°$

Since x and y have to be associated with $\cos(\theta)$ and $\sin(\theta)$, respectively, and

$$\cos^2(\theta) + \sin^2(\theta) = 1,$$

the coefficients have to be normalized. Divide both sides by the square root of the sum of the squares of the constants associated with x and y. Since $\sqrt{(-\sqrt{3})^2 + (-1)^2} = 2$, we get

$$-\frac{\sqrt{3}}{2}x - \frac{1}{2}y = \frac{2}{2} = 1 \quad \text{or} \quad x\cos(210°) + y\sin(210°) = 1$$

with $\theta = 210°$ and $s = 1$.

To get used to this representation, let us look at the graphs of the lines with varying values of s and θ. Figures 7.2a, b, c, d show lines with s varying from -1 to 1, and $\theta = 0°$, $\theta = 45°$, $\theta = 90°$, and $\theta = 120°$, respectively. The inside of the ellipse, $f(x, y)$, is the object whose projections along the lines are to be determined. The axis

along which s varies and a perpendicular to it form the rotated coordinate system (s, n'). The x-axis rotated by θ degrees in the counterclockwise direction is the s-axis.

7.2 The Radon Transform

The Radon transform gives the projection of an image along lines in the coordinate plane of the image. The Radon transform $R(s, \theta)$ of a continuous image $f(x, y)$ is defined as

$$R(s, \theta) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y)\delta(x\cos(\theta) + y\sin(\theta) - s)dx\,dy \quad -\infty < s < \infty,\ 0 \leq \theta < \pi \tag{7.3}$$

The definition gives line integrals of $f(x, y)$ along lines at various angles and distances in the (x, y) plane. The strength of the impulse function in the integral is 1 only along a line specified by the parameters s and θ. Elsewhere, the strength of the impulse is zero. Along this line, the line integral of $f(x, y)$ is determined. Remember that the strength of the impulse function $\delta(x)$ is 1 when $x = 0$ and 0 otherwise. The plot of $R(s, \theta)$ is called a sinogram. Similar to the Fourier spectrum, while the values of the sinogram are real, the image can be reconstructed from its sinogram. Using the coordinate transformation, the relations between the coordinate systems (x, y) and (s, n') (rotated) in Fig. 7.2 are given by

$$s = x\cos(\theta) + y\sin(\theta) \tag{7.4}$$
$$n' = -x\sin(\theta) + y\cos(\theta) \tag{7.5}$$

and

$$x = s\cos(\theta) - n'\sin(\theta) \tag{7.6}$$
$$y = s\sin(\theta) + n'\cos(\theta) \tag{7.7}$$

In the rotated coordinate system (s, n'), Eq. (7.3) becomes

$$R(s, \theta) = \int_{-\infty}^{\infty} f(s\cos(\theta) - n'\sin(\theta), s\sin(\theta) + n'\cos(\theta))\,dn' \tag{7.8}$$

In this form, the projection is easily determined as the projection direction coincides with the n'-axis. The other axis coincides with the s-axis. The Radon transform reduces to integration along the n' direction.

The back-projection (not the inverse Radon transform) of $R(s, \theta)$ is defined as

$$\hat{f}(x, y) = \int_0^{\pi} R(x\cos(\theta) + y\sin(\theta), \theta)\, d\theta \tag{7.9}$$

where $\hat{f}(x, y)$ is a blurred version of $f(x, y)$. The back-projection at any point requires projections from all angles. While it is possible to deblur $\hat{f}(x, y)$, it is not implemented in practice due to the availability of a better algorithm.

Example 7.1 Find the Radon transform of an object, which is a circular cylinder with radius r and height 1 located at the origin. The object is characterized by

$$f(x, y) = \begin{cases} 1 \text{ for } x^2 + y^2 \leq r^2 \\ 0 \text{ otherwise} \end{cases}$$

Solution
As the object is circular, its projection is same in all directions. Therefore, let us compute the projection at $\theta = 0°$. Form the definition, we get

$$R(s, \theta) = \int_{-\infty}^{\infty} f(s, n')\, dn'$$

The distance of the edge of the cylinder along a vertical line at distance s from the origin is

$$n' = \sqrt{r^2 - s^2}$$

Therefore, as the distances from the horizontal line to either edge of the cylinder are equal,

$$R(s, \theta) = 2\int_0^{\sqrt{r^2-s^2}} f(s, n')\, dn' = 2\int_0^{\sqrt{r^2-s^2}} 1\, dn' = \begin{cases} 2\sqrt{r^2 - s^2} & \text{for } |s| \leq r \\ 0 & \text{otherwise} \end{cases}$$

Figure 7.3a shows a 256×256 image and (b) shows its Radon transform. At the center of the circle, the projection has the maximum value $2r = 240$ as $s = 0$, reduces away from the center and it is zero on its circumference. In the figure, the transform changes from white to black, independent of the value of θ.

Example 7.2 Find the Radon transform of a 2-D delayed impulse $f(x, y) = \delta(x - x_0, y - y_0)$.

Solution
From the definition of the Radon transform and the sifting property of the impulse, we get

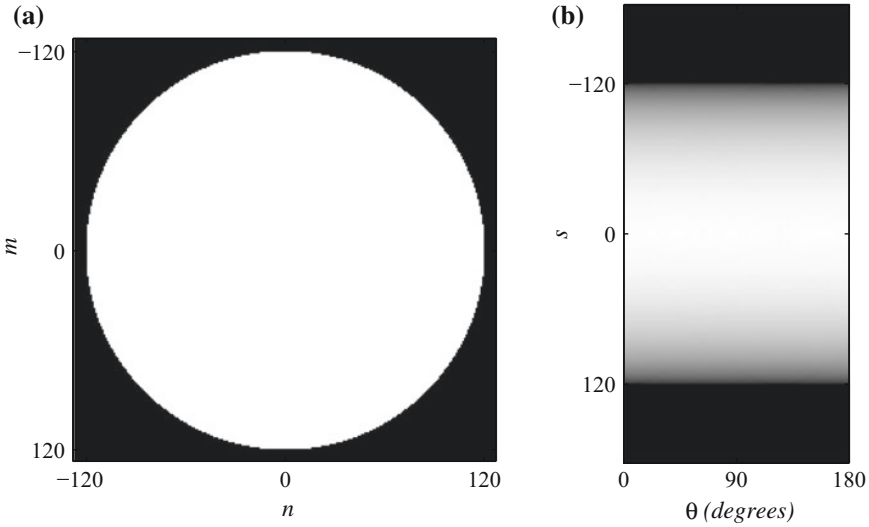


Fig. 7.3 **a** A 256 × 256 image and **b** its Radon transform

$$R(s,\theta) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \delta(x - x_0, y - y_0)\delta(x\cos(\theta) + y\sin(\theta) - s)dx\,dy$$
$$-\infty < s < \infty,\ 0 \le \theta < \pi = \delta(x_0\cos(\theta) + y_0\sin(\theta) - s)$$

As the strength of the impulse is concentrated only when its argument becomes zero, the Radon transform is given by

$$x_0\cos(\theta) + y_0\sin(\theta) - s = 0 \quad \text{or} \quad s = x_0\cos(\theta) + y_0\sin(\theta)$$

The point (x_0, y_0), where the impulse occurs in the spatial domain, can be described in polar coordinates as

$$x_0 = r\cos(\phi), \quad y_0 = r\sin(\phi) \quad \text{and} \quad r = \sqrt{x_0^2 + y_0^2}, \quad \tan(\phi) = \frac{y_0}{x_0},\ x_0 \neq 0$$

The Radon transform, in terms of r and ϕ, is given by

$$s = r\cos(\phi)\cos(\theta) + r\sin(\phi)\sin(\theta) = r\cos(\phi - \theta) = r\cos(\theta - \phi)$$

which is a sinusoid. Therefore, the Radon transform of an impulse is a sinusoid. Figure 7.4a shows an image with impulses and (b) shows its Radon transform. The horizontal and vertical axes are also superimposed in (a). In both the figures, we have dilated the image for clear display of the points and curves. Actually, the points and curves are 1-pixel wide. The Radon transforms of the four impulses

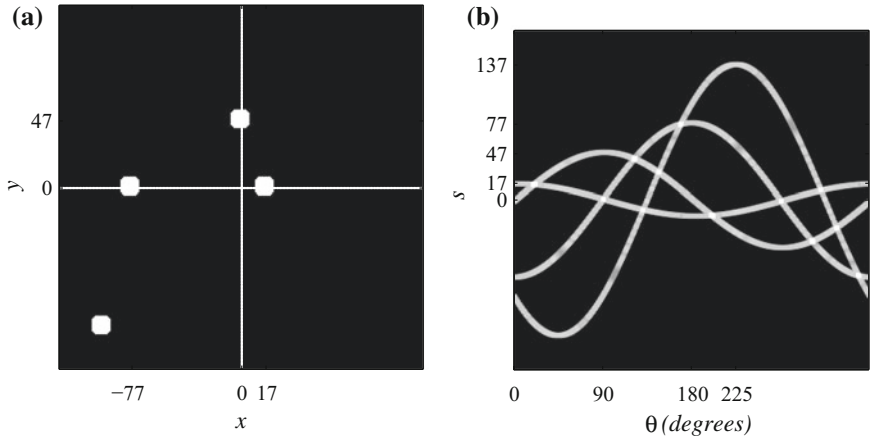


Fig. 7.4 **a** An image with 4 impulses and **b** its Radon transform

$$\delta(x-17, y), \quad \delta(x, y-47), \quad \delta(x+77, y), \quad \delta(x+97, y+97)$$

starting from the one in the east direction, with increasing r values, are

$$s = 17\cos(\theta - 0°), \quad s = 47\cos(\theta - 90°), \quad s = 77\cos(\theta - 180°), \quad s = 137\cos(\theta - 225°)$$

The last sinusoid reaches its positive peak last at $\theta = 225°$.

7.2.1 Properties of the Radon Transform

As the impulse function is part of the Radon transform definition, the properties of the Radon transform are mostly due to the properties of the impulse function.

$$R(s, \theta) = R(-s, \theta \pm 180°)$$

The Radon transform of $f(x, y)$ at $\theta + 180°$ is, from the definition,

$$\begin{aligned} R(s, \theta + 180°) &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x, y)\delta(x\cos(\theta + 180°) + y\sin(\theta + 180°) - s)dx\,dy \\ &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x, y)\delta(-x\cos(\theta) - y\sin(\theta) - s)dx\,dy \\ &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x, y)\delta(-(x\cos(\theta) + y\sin(\theta) + s))dx\,dy = R(-s, \theta) \end{aligned}$$

as the impulse is an even-symmetric signal, $\delta(-t) = \delta(t)$. The Radon transforms of the impulse functions shown in Fig. 7.4b illustrate this property.

$$f(x, y) \leftrightarrow R(s, \theta) \rightarrow f(x - x_0, y - y_0) \leftrightarrow R(s - x_0 \cos(\theta) - y_0 \sin(\theta), \theta)$$

The Radon transform of $f(x - x_0, y - y_0)$ is, from the definition,

$$\begin{aligned} &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x - x_0, y - y_0) \delta(x \cos(\theta) + y \sin(\theta) - s) dx\, dy \\ &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y) \delta((x - x_0) \cos(\theta) + (y - y_0) \sin(\theta) - s) dx\, dy \\ &= R(s - x_0 \cos(\theta) - y_0 \sin(\theta), \theta) \end{aligned}$$

due to the sifting property of the impulse function. Consider a line from (0, 0) to (0, 1). Its transform is $R(0, 0°) = 1$. Let the line gets shifted to coordinates (1, 0) to (1, 1). Its transform $R(1, 0°)$ is, in terms of that of the original line, $R(1-1, 0°) = R(0, 0°) = 1$.

$$f(x, y) \leftrightarrow R(s, \theta) \rightarrow f(kx, ky) \leftrightarrow \frac{1}{|k|} R(ks, \theta), \qquad k \neq 0$$

The Radon transform of $f(kx, ky)$ is, from the definition,

$$\begin{aligned} &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(kx, ky) \delta(kx \cos(\theta) + ky \sin(\theta) - s) dx\, dy \\ &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y) \frac{1}{|k|} \delta(x \cos(\theta) + y \sin(\theta) - \frac{s}{k}) dx\, dy \\ &= \frac{1}{|k|} R(ks, \theta) \end{aligned}$$

due to the scaling property of the impulse function. The transform of the scaled function is the scaled and compressed version of that of the original function. Consider a rectangular image with vertices at (0, 0), (0, 1), (2, 1), and (2, 0). The transform at angle zero degree is $R(s, 0) = 1$, $s = 0$ to 2. The compressed version of the rectangle with $k = 2$ has its vertices at (0, 0), (0, 1/2), (1, 1/2), and (1, 0). The transform at angle zero degree is $R(s, 0) = 1/2$, $s = 0$ to 1.

Further properties of the Radon transform are as follows:

1. The Radon transform is linear. That is, the transform of a linear combination of a set of images is the same linear combinations of their individual transforms.
2. An impulse in the spatial domain corresponds to a sine wave in the transform domain.
3. A line in the spatial domain corresponds to an impulse in the transform domain.
4. Rotation of an image by ϕ degrees shifts its transform by ϕ, $R(s, \theta + \phi)$. The transforms of two impulses located at right angles differ by a phase shift of 90° in Fig. 7.4b.

5. The Radon transform is periodic with period 2π, $R(s, \theta) = R(s, \theta + 2k\pi)$ with k an integer, as the sinusoids in the definition are periodic.
6. If $f(x, y) = 0$, $|x| > K$, $|y| > K$, then $R(s, \theta) = 0$, $|s| > \sqrt{2}K$.

7.2.2 The Discrete Approximation of the Radon Transform

In the discrete case, the lines are defined by

$$m\cos(\theta) + n\sin(\theta) = s$$

where θ is the angle between the m-axis and the perpendicular from the origin to the line, and $s \geq 0$ is the length of the perpendicular. The Radon transform is approximated for a $N \times N$ image $x(m, n)$ as

$$R(s, \theta) = \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} x(m, n)\delta(m\cos(\theta) + n\sin(\theta) - s) \tag{7.10}$$

where m, n, s, and θ are all discrete variables. The value of the impulse function in the summation is 1 only along a line specified by the parameters s and θ. Along this line, the pixel values are summed. Given a set of values for θ, the summation is carried out for lines, at each value of θ, with various s values. Remembering that Eq. (7.3) is a line integral, a numerical integration constant has to be taken into account in using Eq. (7.10). The reason is that increased number of samples, with the assumption that the sampling interval is one, will give different summation values for approximating the same integral.

In the rotated coordinate system (s, n'), Eq. (7.10) becomes

$$R(s, \theta) = \sum_{n'} x(s\cos(\theta) - n'\sin(\theta), s\sin(\theta) + n'\cos(\theta)) \tag{7.11}$$

We can verify the coordinate transformations using Fig. 7.2, with coordinates (x, y) replaced by (m, n), for some specific values. When $n' = 0$, clearly, $m = s\cos(\theta)$ and $n = s\sin(\theta)$. When $\theta = 0$, clearly, $m = s$ and $n = n'$. When $\theta = 90$, clearly, $m = -n'$ and $n = s$. After rotating $x(m, n)$ by a specified angle, the sum is evaluated along the columns of the rotated image. The Radon transform transforms an image in the spatial domain (m, n) to the (s, θ) domain. In the DFT, a transform coefficient $(a + jb)$ corresponds to a complex sinusoid in the time domain. Similarly, a Radon transform coefficient $R(s, \theta)$ corresponds to the mapping of an image along a line in the spatial domain.

The back-projection (not the inverse Radon transform) of $R(s, \theta)$ is defined as

$$\hat{x}(m,n) = \sum_{\theta} R(m\cos(\theta) + n\sin(\theta), \theta) \tag{7.12}$$

where $\hat{x}(m,n)$ is a discrete and blurred version of $x(m,n)$. The reconstructed image is found corresponding to each of the angles θ, and the resulting images are summed to get the final image.

Example 7.3 Find the Radon transform of the 2×2 image

$$x(m,n) = \begin{bmatrix} 1 & 4 \\ 2 & 5 \end{bmatrix}$$

Let the origin $(0,0)$ be at the bottom-left corner. Reconstruct the image by back-projection using the transform coefficients.
Solution
With $\theta = 0°$, the sum of the columns yields

$$R(0,0°) = 3 \quad \text{and} \quad R(1,0°) = 9$$

The average (DC) value of the image is 3. This value has to be subtracted from the image to find the transform at other angles except at any one angle. Making the average value 0, we get

$$x(m,n) = \begin{bmatrix} 1-3 & 4-3 \\ 2-3 & 5-3 \end{bmatrix} = \begin{bmatrix} -2 & 1 \\ -1 & 2 \end{bmatrix}$$

With $\theta = 90°$, the sum of the rows yields

$$R(0,90°) = 1 \quad \text{and} \quad R(1,90°) = -1$$

Let us reconstruct the image using Eq. (7.12). With $m = 0$, $n = 0$, and $\theta = 0°$, we get $x(0,0) = R(0,0°) = 3$. Proceeding similarly, we get the reconstructed image corresponding to $\theta = 0°$ as

$$x_0(m,n) = \begin{bmatrix} 3 & 9 \\ 3 & 9 \end{bmatrix}$$

The reconstructed image corresponding to $\theta = 90°$ is

$$x_{90}(m,n) = \begin{bmatrix} -1 & -1 \\ 1 & 1 \end{bmatrix}$$

The sum of the partially reconstructed images is the final image given by

$$x(m,n) = x_0(m,n) + x_{90}(m,n) = \begin{bmatrix} 3 & 9 \\ 3 & 9 \end{bmatrix} + \begin{bmatrix} -1 & -1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 8 \\ 4 & 10 \end{bmatrix}$$

which is the same as the input image multiplied by 2. A factor of 2 appears because we ignored the numerical integration constant 2. For example, the area under the samples $\{1, 2\}$, with width 1, is $(1 + 2)/2$ but not $(1 + 2) = 3$. All the 4 Radon transform values have to be divided by 2. That is,

$$\{R(0, 0^\circ) = \frac{3}{2},\ R(1, 0^\circ) = \frac{9}{2},\ R(0, 90^\circ) = \frac{1}{2},\ R(1, 90^\circ) = -\frac{1}{2}\}$$

In practical applications, a set of projections $R(s, \theta)$ is provided by the sensors. The problem is the inversion of the transform to reconstruct the image. The DFT is used, in practice, to find the reconstructed image. Let us find the relation between the Radon transform and the 2-D DFT spectrum of an image. The 2-D DFT $X(k, l)$ of a $N \times N$ image $x(m, n)$ is defined as

$$X(k, l) = \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} x(m, n) e^{-j\frac{2\pi}{N}(mk+nl)}, \quad k, l = 0, 1, \ldots, N-1.$$

Let the frequency index l be 0. Then,

$$X(k, 0) = \sum_{m=0}^{N-1} \{\sum_{n=0}^{N-1} x(m, n)\} e^{-j\frac{2\pi}{N}(mk)}, \quad k = 0, 1, \ldots, N-1.$$

The summation inside the braces is $R(s, 0^\circ)$. Therefore,

$$NR(s, 0^\circ) \leftrightarrow X(k, 0) \quad \text{and similarly} \quad NR(s, 90^\circ) \leftrightarrow X(0, l).$$

Example 7.4 Using the 2-D DFT of the image, find the Radon transform of the 2×2 image

$$x(m, n) = \begin{bmatrix} 1 & 4 \\ 2 & 5 \end{bmatrix}$$

Let the origin $(0, 0)$ be at the bottom-left corner.

Solution

Let us compute the 2-D DFT of $x(m, n)$ using the row–column method. The 1-D row DFT of the image and the column DFT of this partial transform are the 2-DFT and they are

$$\begin{bmatrix} 5 & -3 \\ 7 & -3 \end{bmatrix} \quad \begin{bmatrix} 2 & 0 \\ 12 & -6 \end{bmatrix}$$

The 2-D DFT is shown in Fig. 7.5. The 1-D IDFT of the first row coefficients $\{12, -6\}$ is $\{3, 9\}$. These are the $R(0, 0^\circ)$ and $R(1, 0^\circ)$ coefficients computed in Example (7.3) with $\theta = 0^\circ$. The 1-D IDFT of the first column coefficients $\{0, 2\}$ is $\{1, -1\}$. These are the $R(0, 90^\circ)$ and $R(1, 90^\circ)$ coefficients computed in Example 7.3 with $\theta = 90^\circ$. Remember that the DC coefficient $X(0, 0)$ can be included only in

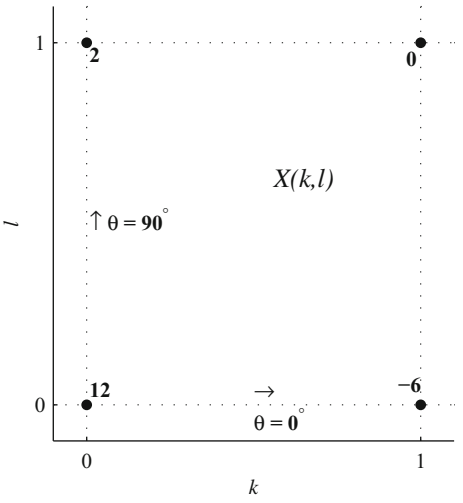


Fig. 7.5 The 2-DFT spectrum of the image $X(k, l)$

one computation. As we computed the 1-D 2-point IDFT using the 2×2 2-D DFT coefficients, we have to divide these coefficients by 2 to get the true Radon transform coefficients.

The conclusion from Example (7.4) is that the 1-D DFT of the Radon transform $R(s, \theta_k)$ in a certain direction is the 2-D DFT $X(s, \theta_k)$ of the image in the same direction with a scale factor.

Example 7.5 Using the 2-D DFT of the image, find the Radon transform of the 8×8 image

$$x(m, n) = \sin\left(\frac{2\pi}{8}n\right)$$

shown in Fig. 7.6a.

Solution
The image is a sinusoidal surface composed of a stack of 8 sine waves along the m-axis. Its DFT is located on the imaginary axis with nonzero values $X(0, 1) = -j32$ and $X(0, -1) = j32$, as shown in Fig. 7.6b in the center-zero format. The 8-point 1-D DFT spectrum with $\theta = 90°$ is

$$\{0, -j32, 0, 0, 0, 0, 0, j32\}$$

in the normal format. The IDFT of this spectrum

$$\frac{8}{\sqrt{2}}\{0, 1, \sqrt{2}, 1, 0, -1, -\sqrt{2}, -1\}$$

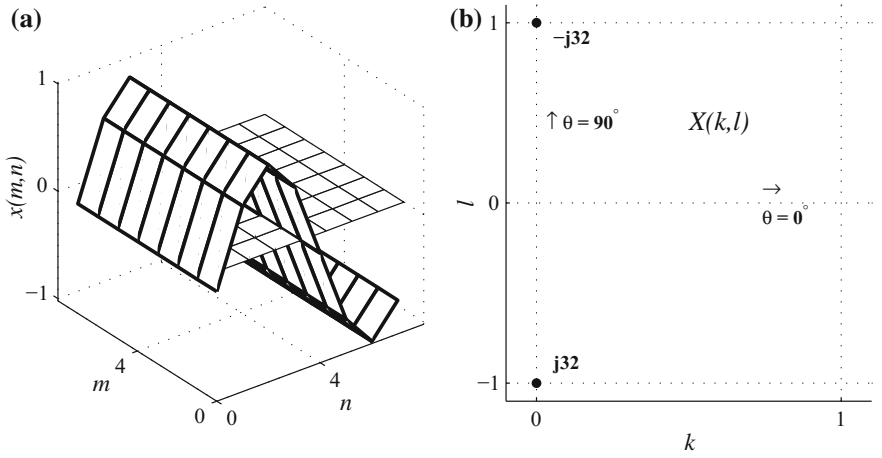


Fig. 7.6 **a** A 8×8 sinusoidal surface, $x(m, n) = \sin(\frac{2\pi}{8}n)$, and **b** its 2-D DFT spectrum, $X(k, l)$, in the center-zero format and the angle of the coefficients

is the set of Radon transform coefficients

$$\{R(0, 90^\circ), R(1, 90^\circ), R(2, 90^\circ), R(3, 90^\circ), R(-4, 90^\circ), R(-3, 90^\circ), R(-2, 90^\circ), R(-1, 90^\circ)\}$$

multiplied by 8. Let us reconstruct the image using Eq. (7.12), which reduces to $\hat{x}(m, n) = R(n)$.

In the examples presented so far, the DFT coefficients are located in the spectrum at angles $\theta = 0^\circ$ and $\theta = 90^\circ$. In these cases, the Cartesian and polar coordinates coincide. With other angles, the two systems of coordinates do not coincide and we need to interpolate the spectral values to find the spectrum in the polar coordinates. Remember that the input image and its spectrum are in Cartesian coordinates and we need the spectrum in polar coordinates to find the Radon transform at various angles. A polar plot is composed of radial lines and concentric circles. Interpolation was introduced in rotating an image in Chap. 6, and in this chapter, we use interpolation to find the spectral values in a polar coordinate spectrum from those in a Cartesian coordinate spectrum.

7.2.3 The Fourier-Slice Theorem

In the examples presented thus far, we computed the 2-D DFT of the image, found the DFT spectral values along the required angle, and computed the 1-D IDFT of these values to find the Radon transform at that angle. This is called the Fourier-slice theorem. As the Radon transform is defined as a continuous function, we have to use the Fourier transform (FT) version of the Fourier analysis in derivations. The results

are approximated in the practical implementation using the DFT with interpolation. The theorem states that the 1-D FT of the Radon transform $R(s, \theta)$ with respect to s of an image $f(x, y)$ is equal to the slice of the 2-D FT of the image at the same angle θ. This theorem relates the 1-D FT of the projections of an image to that of its 2-D FT and is the basis for the effective reconstruction of the image from its projections. The 1-D FT of a projection $R(s, \theta)$ with respect to s, for a given θ, is given by

$$R(j\omega, \theta) = \int_{-\infty}^{\infty} R(s, \theta) e^{-j\omega s} ds \tag{7.13}$$

Substituting for $R(s, \theta)$, we get

$$\begin{aligned} R(j\omega, \theta) &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(s\cos(\theta) - n' \sin(\theta), s\sin(\theta) + n'\cos(\theta)) e^{-j\omega s}\, ds\, dn' \\ &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y) e^{-j\omega(x\cos(\theta) + y\sin(\theta))} dx\, dy \end{aligned}$$

The last step is obtained using the coordinate transformation from (s, n') to (x, y). Letting $\omega_1 = \omega\cos(\theta)$ and $\omega_2 = \omega\sin(\theta)$, we get

$$R(j\omega, \theta) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y) e^{-j(\omega_1 x + \omega_2 y)} dx\, dy = F(j\omega_1, j\omega_2)\,|_{\omega_1 = \omega\cos(\theta), \omega_2 = \omega\sin(\theta)}$$

While the theory is perfect for continuous signals, due to the necessity for interpolation, the reconstructed images are only approximate. As it is always the case, in discrete signal analysis, sampling has to be adequate so that the error in processing the signal is within acceptable limits. In the Radon transform, increasing the number of values of the parameter s increases the number of samples. Increasing the number of values of the parameter θ increases the number of partially reconstructed images. These two parameters should be suitably selected for the given application.

We can get the spectral values on to a square grid from the polar values by interpolation, the 2-D IDFT of which yields the reconstructed image. However, it is difficult to interpolate the spectral values in the polar form. Another problem for the degradation of the reconstructed image is that the distance between sample points increases as the frequency increases from the origin along the radial lines, as shown in Fig. 7.7. Ideally, the spectrum should be expressed as a sum of wedges. The high-frequency components are not adequately represented, which results in blurring of the image. That is, the values of the high-frequency components must be boosted to reduce blurring. Therefore, this method of reconstruction is not practically used.

Example 7.6 Using the 2-D DFT of the image, find the Radon transform of the 8×8 image

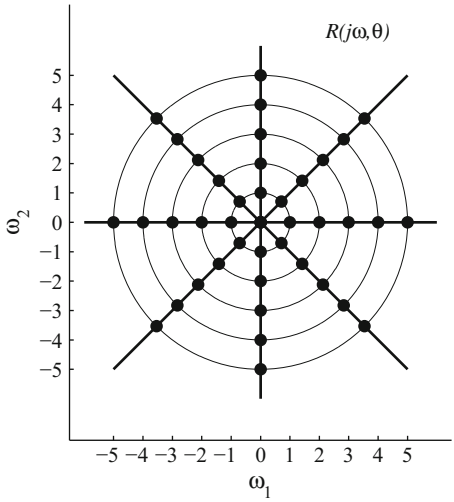


Fig. 7.7 The FT of the Radon transform in polar coordinates

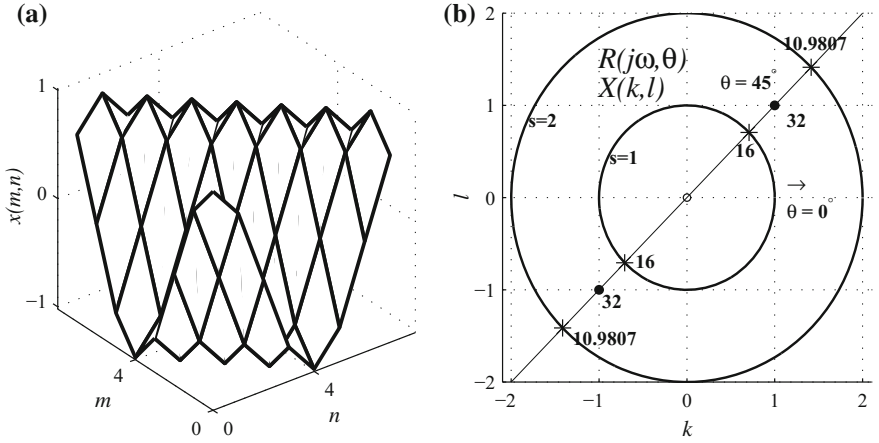


Fig. 7.8 **a** A 8×8 sinusoidal surface, $x(m, n) = \cos\left(\frac{2\pi}{8}(m+n)\right)$, and **b** its 2-D DFT spectrum (dots), $X(k, l)$, in the center-zero format, and in the polar form, $X(s, \theta)$ (asterisks)

$$x(m, n) = \cos\left(\frac{2\pi}{8}(m+n)\right)$$

shown in Fig. 7.8a.

Solution The image is a sinusoidal surface composed of a stack of a cosine waves along an axis with a shift in the other direction. The values of $x(m, n)$ are

$$x(m,n) = \frac{1}{\sqrt{2}} \begin{bmatrix} \sqrt{2} & 1 & 0 & -1 & -\sqrt{2} & -1 & 0 & 1 \\ 1 & 0 & -1 & -\sqrt{2} & -1 & 0 & 1 & \sqrt{2} \\ 0 & -1 & -\sqrt{2} & -1 & 0 & 1 & \sqrt{2} & 1 \\ -1 & -\sqrt{2} & -1 & 0 & 1 & \sqrt{2} & 1 & 0 \\ -\sqrt{2} & -1 & 0 & 1 & \sqrt{2} & 1 & 0 & -1 \\ -1 & 0 & 1 & \sqrt{2} & 1 & 0 & -1 & -\sqrt{2} \\ 0 & 1 & \sqrt{2} & 1 & 0 & -1 & -\sqrt{2} & -1 \\ 1 & \sqrt{2} & 1 & 0 & -1 & -\sqrt{2} & -1 & 0 \end{bmatrix}$$

The DFT of the image in the standard and center-zero formats, respectively, is

$$X(k,l) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 32 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 32 \end{bmatrix} \qquad X(k,l) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 32 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 32 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

By swapping the quadrants of the spectrum, we get one format from the other.

The DFT of the image has nonzero values only on the diagonal with $\theta = 45°$, as shown in Fig. 7.8b, in the center-zero format. The nonzero values are $X(1,1) = 32$ and $X(-1,-1) = 32$. From these coefficients, we have to find the corresponding polar spectral coefficients by interpolation. Using the linear interpolation formula Eq. (6.1), with $s = 1$, the interpolated value is $32/(\sqrt{2}\sqrt{2}) = 16$. With $s = 2$, the interpolated value is $32(2-\sqrt{2})(2-\sqrt{2}) = 10.9807$, as shown in Fig. 7.8b. With a 8×8 image in the center-zero format, the farthest coordinates from the center are $(4,4)$. In polar coordinates, the magnitude of the farthest coordinates is

$$4\cos(45°) + 4\sin(45°) = \frac{8}{\sqrt{2}} = 5.6569$$

With rounding, the value becomes 6. Therefore, the required s values for reconstruction are $s = -6, -5, \ldots, 5, 6$. The 13-point 1-D DFT spectrum with $\theta = 45°$ is

$$\{0, 16, 10.9807, 0, 0, 0, 0, 0, 0, 0, 0, 10.9807, 16\}$$

in the normal format. The IDFT of this spectrum, in the center-zero format,

$$\{-0.8942, -1.6389, -2.1374, -1.3435, 0.7993, 3.1392, 4.1509, 3.1392,$$

$$0.7993, -1.3435, -2.1374, -1.6389, -0.8942\}$$

is the set of Radon transform coefficients

$$R(s, 45^\circ), \quad s = -6, -5, \ldots, 5, 6$$

multiplied by 64/13. The Radon coefficients are

$$\{-0.1816, -0.3329, -0.4342, -0.2729, 0.1624, 0.6377, 0.8431,$$

$$0.6377, 0.1624, -0.2729, -0.4342, -0.3329, -0.1816\}$$

The values of the image can be reconstructed from the Radon transform coefficients using Eq. (7.12).

7.2.4 Reconstruction with Filtered Back-projections

The polar coordinate system is the mainstay of the Radon transform due to its inherent nature. Therefore, it is more appropriate to find the reconstructed image using the 2-D IFT in polar form. The 2-D IFT of the FT, $F(j\omega_1, j\omega_2)$, of an image $f(x, y)$ is given by

$$f(x, y) = \frac{1}{4\pi^2} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} F(j\omega_1, j\omega_2) e^{j(\omega_1 x + \omega_2 y)} d\omega_1 \, d\omega_2$$

Letting $\omega_1 = \omega \cos(\theta)$, $\omega_2 = \omega \sin(\theta)$, the differentials $d\omega_1 \, d\omega_2$ become $\omega d\omega \, d\theta$. Then, using the polar coordinates and the Fourier-slice theorem, we get

$$\begin{aligned} f(x, y) &= \frac{1}{4\pi^2} \int_0^{2\pi} \int_0^{\infty} F(j\omega \cos(\theta), j\omega \sin(\theta)) e^{j\omega(x \cos(\theta) + y \sin(\theta))} \omega d\omega \, d\theta \\ &= \frac{1}{4\pi^2} \int_0^{2\pi} \int_0^{\infty} R(j\omega, \theta) e^{j\omega(x \cos(\theta) + y \sin(\theta))} \omega d\omega \, d\theta \\ &= \frac{1}{4\pi^2} \int_0^{\pi} \int_{-\infty}^{\infty} R(j\omega, \theta) e^{j\omega(x \cos(\theta) + y \sin(\theta))} |\omega| d\omega \, d\theta \\ &= \frac{1}{4\pi^2} \int_0^{\pi} \left(\int_{-\infty}^{\infty} |\omega| R(j\omega, \theta) e^{j\omega s} ds \right) \Big|_{s = (x \cos(\theta) + y \sin(\theta))} \end{aligned}$$

Note that $R(j\omega, \theta)$ is symmetric.
The inverse Radon transform consists of two steps. The projection $R(s, \theta)$ is filtered first, and then, the back-projection is carried out. The filtering operation can be implemented in the spatial domain or frequency domain.

From Example (7.6), the ramp-filtered 13-point 1-D DFT spectrum with $\theta = 45^\circ$ is

$$\{0, 16, 21.9614, 0, 0, 0, 0, 0, 0, 0, 0, 21.9614, 16\}$$

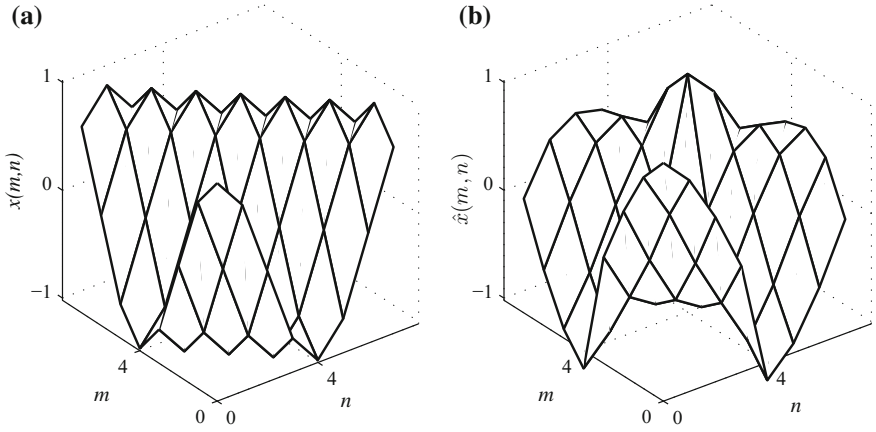


Fig. 7.9 **a** An image and **b** its filtered and back-projected reconstruction

in the normal format. The IDFT of this spectrum, multiplied by the scaling constant 13/64 and in the center-zero format, is

$$\{0.1222, -0.2915, -0.6910, -0.6061, 0.0407, 0.8326, 1.1863,$$

$$0.8326, 0.0407, -0.6061, -0.6910, -0.2915, 0.1222\}$$

is the set of filtered Radon transform coefficients

$$R(s, 45^\circ), \quad s = -6, -5, \ldots, 5, 6$$

The pixel values of the image can be reconstructed from the Radon transform coefficients using Eq. (7.12). Figures 7.9a, b show, respectively, a 8×8 image and its filtered back-projected reconstruction. We started with an image, found its Radon transform, and then reconstructed the given image. This process involved interpolation of the DFT coefficients. We went through this procedure in order to understand the Radon transform. If the image is available, then there is no necessity to use the Radon transform. In practice, as mentioned earlier, the Radon transform is given from practical measurements and the real problem is the reconstruction of the image from the given projections.

Figure 7.10 shows the block diagram of the inverse Radon transform with the filtering carried out in the s-domain. The Radon transform $R(s, \theta)$ in each direction is convolved with the impulse response of the filter and then back-projected.

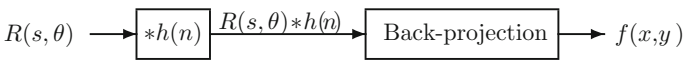


Fig. 7.10 The inverse Radon transform in the s-domain

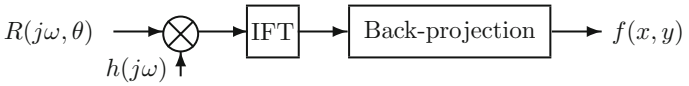


Fig. 7.11 The inverse Radon transform in the frequency domain

Figure 7.11 shows the block diagram of the inverse Radon transform with the filtering carried out in the frequency domain. As convolution becomes multiplication in the frequency domain, filtering is carried out by multiplying the FT of $R(s, \theta)$, $R(j\omega, \theta)$, with the frequency response of the filter. The IFT of the product is the filtered Radon transform. Then, back-projection reconstructs the image.

Figure 7.12a–d shows the filtered reconstruction of a cylinder using Radon transform in 4, 8, 16, and 32 directions, respectively. As the number of directions is increased, the reconstructed image becomes better. The process is very similar to the reconstruction of a waveform in Fourier analysis with more and more components. Remember the reconstruction of the square wave. In the Fourier analysis or Radon transform, the desired object is created by the interference from the components. At some part of the object, the interference is constructive and it is destructive at other parts, such that the object is reconstructed better and better with increased number of transform components. As in Fourier analysis, while infinite components are required in theory, a finite number of components are used for reconstruction so that the quality of the reconstructed image is adequate.

A procedure for computing the Radon transform is as follows.

1. Compute the 2-D DFT of the image.
2. Interpolate the spectral values to get the spectrum on polar coordinates, for all angles of interest.
3. Compute the 1-D IDFT of the spectral values at all angles to get the Radon transform.

A procedure for computing the inverse Radon transform is as follows.

1. Compute the 1-D DFT of each of the projections of the image.
2. Multiply each DFT by the ramp filter. Take into account the DC value of the spectrum.
3. Compute the 1-D IDFT of the spectral values at all angles to get the filtered Radon transform.
4. Obtain the filtered back-projected image using the back-projection definition, Eq. (7.12), for each angle of projection.
5. Sum all the filtered back-projected images to reconstruct the image.

While there are other procedures for computing the Radon transform and its inverse, the procedure given, using the DFT, is conceptually simpler to understand. The alternative of using a windowed ramp filter reduces the ringing in the reconstructed image, but results in blurring. While we computed the Radon transform

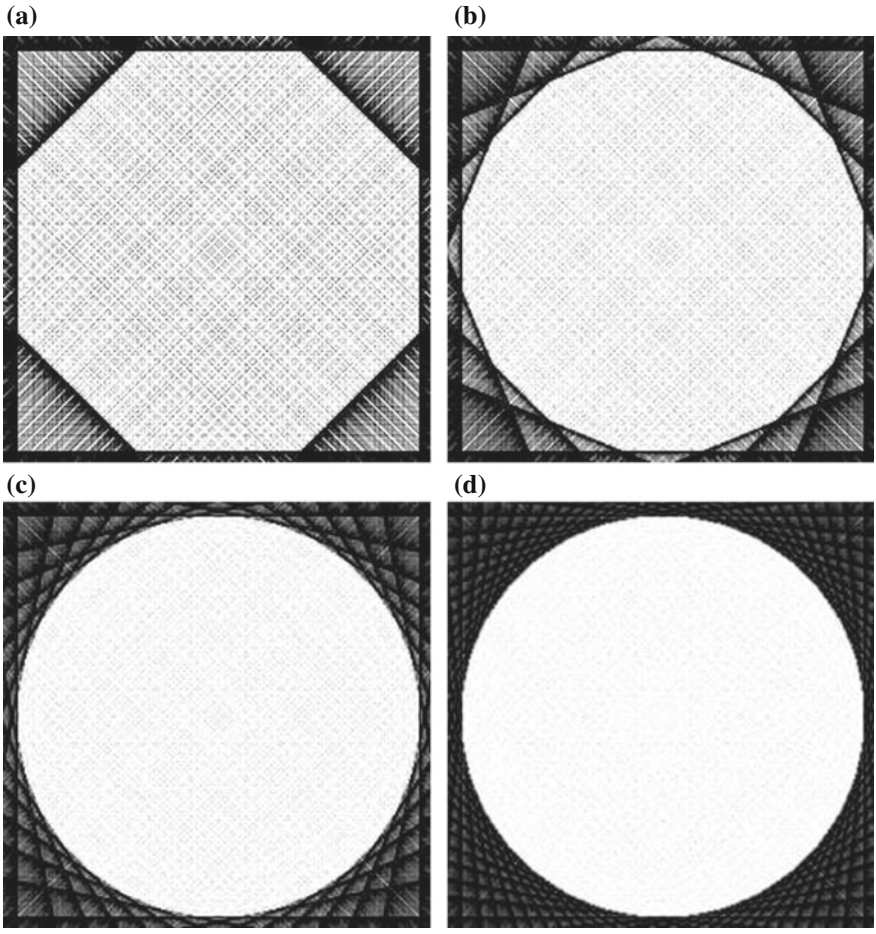


Fig. 7.12 (a-d) Reconstruction of a cylinder using Radon transform in 4, 8, 16, and 32 directions, respectively

with 0, 45, and 90° for simplicity in the examples, typically, angles vary from 0 to 179 ° with an increment of 1. The range of the s values is proportional to the size of image.

7.3 Hough Transform

In the normal form, a line is represented in the (s, θ) domain by a single point. In the Hough transform, this representation is used to detect lines in the image. The more the number of points in the image with a certain (s, θ), it is more likely that a line

characterized by the vector lies in the image. Therefore, it is finding the number of occurrences of all possible (s, θ) vectors and applying a threshold. This transform is more efficient to detect a line than by template matching. Further, its performance is better in the presence of noise and when a line is partially occluded. The Hough transform is mostly used to detect lines in images, although it can be extended to find curves of a specified shape.

Let the input be a $N \times N$ binary image. Let $s(k)$ and $\theta(k)$ be the arrays containing the discretized values of the parameters s and θ of the lines in the normal form. The distance parameter s varies from 0 to $\sqrt{((N-1)^2 + (N-1)^2)}$. The angle parameter θ varies from 0 to π. The steps of the Hough transform algorithm are as follows:

1. Select a set of points for the parameters (s, θ).
2. For each value of θ, compute the corresponding value of s using Eq. (7.2) for all nonzero pixels.
3. Create an accumulator matrix which accumulates the number of occurrences of each pair of (s, θ), as all the pixels with value 1 in the input image are analyzed.
4. Find the accumulator values those are greater than a given threshold.

The output is a set of lines characterized by the thresholded values of the accumulator matrix.

Example 7.7 Detect the line in the 4×4 binary image $x(m, n)$. The origin is at the top-left corner.

$$x(m,n) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Solution
Variable s varies from 0 to $\sqrt{3^2 + 3^2}$, and the maximum distance from the origin is $s = \sqrt{18} = 4.2426$. Let the discrete values of s be $s = \{0, 1, 2, 3, 4\}$. Let the discrete values of θ be $\theta = \{0, 45, 90, 135\}$ degrees. These values are to be set to suit the accuracy required. Initialize a 5×4 *acc* matrix to hold the number of votes for each parameter set. Compute

$$m\cos(\theta(k)) + n\sin(\theta(k)) = s$$

for all nonzero pixels in the image. Find $s(k)$ that is nearest to s. Increment $acc(s, \theta)$ by one for each occurrence. The set of parameters (s, θ) corresponding to the selected vote counts describe the lines. Each nonzero pixel is mapped into the following parameter space.

$$(s, \theta) = \begin{bmatrix} (0,0) & (0,45) & (0,90) & (0,135) \\ (1,0) & (1,45) & (1,90) & (1,135) \\ (2,0) & (2,45) & (2,90) & (2,135) \\ (3,0) & (3,45) & (3,90) & (3,135) \\ (4,0) & (4,45) & (4,90) & (4,135) \end{bmatrix}$$

The second row of the image only has nonzero pixels. For pixel $x(1, 0)$, we get the s values as

$$\begin{aligned} s &= 1\cos(0) + 0\sin(0) = 1 \\ s &= 1\cos(45) + 0\sin(45) = \frac{1}{\sqrt{2}} \approx 1 \\ s &= 1\cos(90) + 0\sin(90) = 0 \\ s &= 1\cos(135) + 0\sin(135) = -\frac{1}{\sqrt{2}} \end{aligned}$$

Note that the negative value $s = -1/\sqrt{2}$ is ignored. For pixel $x(1, 1)$, we get the s values as

$$\begin{aligned} s &= 1\cos(0) + 1\sin(0) = 1 \\ s &= 1\cos(45) + 1\sin(45) = \sqrt{2} \approx 1 \\ s &= 1\cos(90) + 1\sin(90) = 1 \\ s &= 1\cos(135) + 1\sin(135) = 0 \end{aligned}$$

For pixel $x(1, 2)$, we get the s values as

$$\begin{aligned} s &= 1\cos(0) + 2\sin(0) = 1 \\ s &= 1\cos(45) + 2\sin(45) = \frac{3}{\sqrt{2}} \approx 2 \\ s &= 1\cos(90) + 2\sin(90) = 2 \\ s &= 1\cos(135) + 2\sin(135) = \frac{1}{\sqrt{2}} \approx 1 \end{aligned}$$

For pixel $x(1, 3)$, we get the s values as

$$\begin{aligned} s &= 1\cos(0) + 3\sin(0) = 1 \\ s &= 1\cos(45) + 3\sin(45) = 2\sqrt{2} \approx 3 \\ s &= 1\cos(90) + 3\sin(90) = 3 \\ s &= 1\cos(135) + 3\sin(135) = \sqrt{2} \approx 1 \end{aligned}$$

The values of the *acc* matrix, after scanning all the pixels, are

$$acc(m,n) = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 4 & 2 & 1 & 2 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

The values in the *acc* matrix are subjected to a suitable threshold. Let the threshold be 3. Then, the entry $acc(1, 0) = 4$ in the *acc* matrix indicates a line at distance 1 from the top-left corner (origin) of the image at angle $0 + 90 = 90°$ from the m-axis. That is a horizontal line in the second row of the image.

An image and the its *acc* matrix are

$$x(m,n) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad acc(m,n) = \begin{bmatrix} 1 & 1 & 1 & 2 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Let the threshold be 2. Then, the entry $acc(0, 3) = 2$ in the *acc* matrix indicates a line at distance 0 from the top-left corner (origin) of the image at angle $135 + 90 = 225°$ from the m-axis. That is a line along the main diagonal of the image. An image and the its *acc* matrix are

$$x(m,n) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \qquad acc(m,n) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 0 \end{bmatrix}$$

Let the threshold be 2. Then, the entry $acc(4, 1) = 2$ in the *acc* matrix indicates a line at distance 4 from the top-left corner (origin) of the image at angle $45 + 90 = 135°$ from the m-axis. That is a line near the right bottom of the image. The end points of a line and its exact position can be found by having another accumulator that keeps track of the coordinates of the pixels for each entry in the *acc* matrix.

Typical applications of this transform include feature extraction, image recognition, and compression. This transform can also be extended to detect curves. However, the computational complexity also increases. Figure 7.13a shows a 256×256 gray level image.

The binary image showing the edge map is shown in Fig. 7.13b. The lines are 1-pixel wide. For a clear view, we have shown the dilated lines in (b). There are 4 horizontal and 4 vertical lines in the image. Using values $s = 0$ to $s = 361$ and $\theta = 0, 45, 90, 135$ degrees, a 362×4 accumulator is used to collect the votes. Figure 7.13c shows the number of votes versus distance s for $\theta = 0°$. It is obvious from the figure that width of the lines is about 60 and 120 pixels. The lines occur

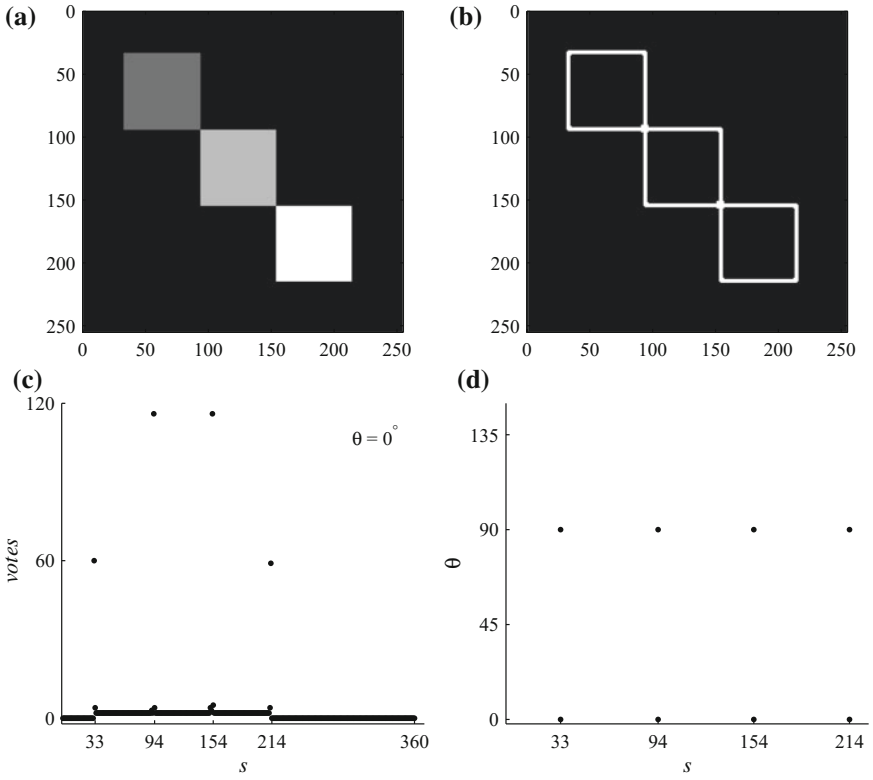


Fig. 7.13 **a** A 256×256 gray level image; **b** the edge map image; **c** number of votes versus distance s for $\theta = 0°$; **d** the Hough transform of the image

at $s = 33, 94, 154, 214$. For $\theta = 90°$, we get a similar statistics accounting for all the 8 lines. With a threshold 20, the (s, θ) vectors of the detected lines are shown in Fig. 7.13d.

7.4 Summary

- In the Radon transform, an image is represented by its mappings with respect to a set of lines at various angles represented by polar coordinates. The values at various polar coordinates are the transform coefficients.
- When an image is represented in Radon transform form, we are able to form the image of the interior of an object with out intrusion. In its implementation, we use the 1-D DFT and interpolation operations.
- The Radon transform uses the normal form of a line. In this form, a line is expressed in terms of its perpendicular distance, s, from the line to the origin and the angle, θ, subtended between the perpendicular line and the x-axis.

- The Radon transform transforms an image in the spatial domain (m, n) to the (s, θ) domain.
- The reconstructed image is found corresponding to each of the angles θ, and the resulting images are summed to get the final image, called the back-projection.
- In practical applications, a set of projections $R(s, \theta)$ is provided by the sensors. The problem is the inversion of the transform to reconstruct the image. The direct or indirect convolution and back-projection are used, in practice, to find the inverse Radon transform.
- Projections are obtained, for example, by the absorption of the X-ray in passing through the object from the source to the detector. The source and the detector assembly are rotated around the object to get the projections at a finite set of angles.
- Fourier-slice theorem states that the 1-D FT of the Radon transform $R(s, \theta)$ with respect to s of an image $f(x, y)$ is equal to the slice of the 2-D FT of the image at the same angle θ. This theorem relates the 1-D FT of the projections of an image to that of its 2-D FT and is the basis for the effective reconstruction of the image from its projections.
- Filtered back-projected image using the inverse Radon transform definition, for each angle of projection, is computed. Sum of all the filtered back-projected images yields the image of the object.
- Computerized axial tomography and nondestructive testing of mechanical objects are typical applications of the Radon transform.
- The Hough transform is mostly used to detect lines in images. This transform uses the normal form a line.

Exercises

7.1 Find the equation of the straight line, in the normal form, located at a distance s from the origin and the perpendicular to it makes an angle of θ degrees with the x-axis.
(i) $s = 3,\ \theta = 0°$.
(ii) $s = 1,\ \theta = 45°$.
* (iii) $s = 2,\ \theta = -60°$.
(iv) $s = 5,\ \theta = 315°$.
(v) $s = 0,\ \theta = 30°$.

7.2 Find the Radon transform of a circular cylinder with radius 6 and height 3 located at the origin.
The cylinder is characterized by

$$f(x, y) = \begin{cases} 3 \text{ for } x^2 + y^2 \leq 6^2 \\ 0 \text{ otherwise} \end{cases}$$

From the transform obtained, and using the Radon transform properties, find the Radon transform of

(i)

$$f(x, y) = \begin{cases} 3 \text{ for} x^2 + y^2 \leq 3^2 \\ 0 \text{ otherwise} \end{cases}$$

* (ii)

$$f(x, y) = \begin{cases} 3 \text{ for} (x-1)^2 + (y-2)^2 \leq 6^2 \\ 0 \text{ otherwise} \end{cases}$$

(iii)

$$f(x, y) = \begin{cases} 3 \text{ for } (3x)^2 + (3y)^2 \leq 6^2 \\ 0 \text{ otherwise} \end{cases}$$

7.3 Find the Radon transform of the shifted and scaled impulse.
(i) $\delta(x, y)$
(ii) $\delta(x-4, y-4)$
* (iii) $\delta(x-4, y+4)$
(iv) $\delta(x-1, y)$
(v) $\delta(x, y-1)$

7.4 Find the Radon transform of the line $f(x, y)$ characterized by the given equation. Using the result, find the transforms of $f(ax, ay)$ and $f(x-p, y-q)$ from the properties of the Radon transform. Find the equation of the lines $f(ax, ay)$ and $f(x-p, y-q)$ and determine the Radon transform directly. Verify that the results are the same as those obtained using the properties.
(i)

$$f(x, y) = 3x + 2y - 6 = 0, x \text{ is limited from 0 to 2}, a = 2, p = 2, q = 3$$

(ii)

$$f(x, y) = 4x + 2y - 8 = 0, x \text{ is limited from 0 to 2}, a = 3, p = 3, q = 2$$

(iii)

$$f(x, y) = x + y - 1 = 0, x \text{ is limited from 0 to 1}, a = -3, p = -3, q = 2$$

7.5 Find the Radon transform $R(s, \theta)$ of the image $x(m, n)$ and reconstruct the image from its transform by the back-projection method, using the DFT and the IDFT.
(i)

$$x(m, n) = \begin{bmatrix} 1 & 2 \\ 2 & 3 \end{bmatrix}$$

(ii)

$$x(m, n) = \begin{bmatrix} 3 & 1 \\ 2 & 0 \end{bmatrix}$$

(iii)

$$x(m,n) = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

(iv)

$$x(m,n) = \begin{bmatrix} 3 & 4 \\ 1 & 2 \end{bmatrix}$$

(v)

$$x(m,n) = \begin{bmatrix} 4 & 3 \\ 3 & 2 \end{bmatrix}$$

7.6 Detect the lines in the 4×4 binary image $x(m,n)$. Choose a suitable threshold.

***** (i)

$$x(m,n) = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

(ii)

$$x(m,n) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

(iii)

$$x(m,n) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \end{bmatrix}$$

(iv)

$$x(m,n) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

(v)

$$x(m,n) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Chapter 8
Morphological Image Processing

Abstract In processing the color and grayscale images, which occur mostly, their binary version is often used. In morphological processing of images, pixels are added or removed from the images. The structure and shape of the objects are analyzed so that they can be identified. The basic operations in this processing are binary convolution and correlation, that is based on logical operations rather than arithmetic operations. Dilation and erosion are the basic operations, and rest of the operations and algorithms are based on these operations. Morphological processing is also extended to gray-level images using the minimum and maximum operators.

After segmentation of an image, the shape of its objects has to be analyzed. Morphology is a study of form and structure. In image processing, it is used to analyze and modify geometric properties (shape) of an image by probing it with different forms. The fitness of these forms, called structuring elements, leads to quantitative measures those are useful in computer vision. The process is similar to linear convolution and correlation, except that logical operations AND (denoted by &), OR (denoted by |), and NOT (denoted by ~) are used (a logical neighborhood operation) instead of arithmetic operations. Pixels are added to an object or deleted from it. Border extension has to be defined, and windows (structuring elements) may have to be rotated by 180°. In linear convolution, the output is a linear combination of the pixels in the neighborhood. In median filtering, we use sorting and selection to find the output. In morphological image processing with binary images, we use the logical version of the convolution operation. In the convolution operation, with masks made of different types of impulse responses, we are able to process signals with different filters such as low pass, high pass. In a similar way, with different types of structuring elements (masks) and carrying out convolution with logical operators, we are able to perform various types of analysis of objects. While its primary use is with binary images, morphology is also extended to grayscale images.

D. Sundararajan, *Digital Image Processing*, DOI 10.1007/978-981-10-6113-4_8

8.1 Binary Morphological Operations

Dilation and erosion are the two basic operations in morphology. Various morphological operations are carried out by combining these two operations in a suitable way. Dilation expands objects in an image by adding pixels at the borders, while erosion removes pixels at the borders and shrinks the objects.

8.1.1 Dilation

The 1-D dilation operation is shown in Fig. 8.1a. The input sequence is

$$\{x(0) = 1, x(1) = 1, x(2) = 1, x(3) = 0, x(4) = 0, x(5) = 1\}$$

The window or mask or structuring element (similar to the impulse response in convolution) is

$$\{h(-1) = 1, h(0) = \mathbf{1}, h(1) = 0\}$$

The origin of the structuring element is shown in boldface. This origin must overlap the input pixel being processed. The time-reversed structuring element is

$$\{h(1) = 0, h(0) = \mathbf{1}, h(-1) = 1\}$$

The required pixels at the borders are assumed to be zero for dilation operation, as shown in the figure in dashed boxes. This assumption is used to avoid the border effect. The characterizing equation of the dilation operation is

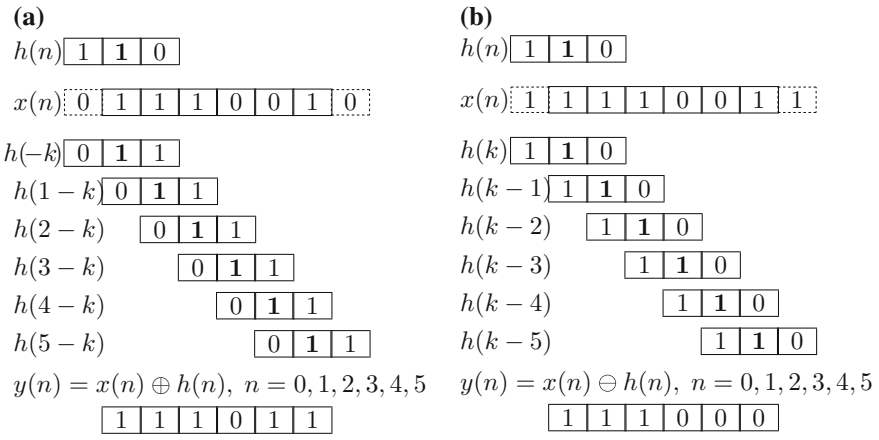


Fig. 8.1 **a** Dilation and **b** erosion

$$y(n) = (h(1)\ \&\ x(n-1)) \mid (h(0)\ \&\ x(n)) \mid (h(-1)\ \&\ x(n+1)),\ n = 0, 1, \ldots 5,$$

which is similar to the linear convolution operation with the arithmetic operations replaced by logical operations. In finding the output for each pixel, the neighborhood is defined by the 1's of the structuring element. The time-reversed structuring element is shifted to various positions and the output, with the same number of elements as that of the input, is found. For the example,

$$y(0) = (0\ \&\ 0) \mid (1\ \&\ 1) \mid (1\ \&\ 1) = 1$$

If there is at least one pair of 1's in the image and the mask at the corresponding positions, then the output is 1. Otherwise, the output is zero. The result is that the object is dilated or expanded. Small holes are filled, and the border becomes smoother.

The dilation of the binary image $x(m, n)$ and the window or mask $h(m, n)$ is defined as

$$y(m, n) = \mid_k \mid_l (h(k, l)\ \&\ x(m-k, n-l)) = x(m, n) \oplus h(m, n), \qquad (\forall k, l) h(k, l) = 1 \tag{8.1}$$

where m and n vary over the dimensions of the image, and k and l vary over the dimensions of the structuring element. Pixels corresponding to $h(k, l) = 1$ contribute to the output. A 3×3 window of an image is

$$\begin{bmatrix} x(m-1, n-1) & x(m-1, n) & x(m-1, n+1) \\ x(m, n-1) & x(m, n) & x(m, n+1) \\ x(m+1, n-1) & x(m+1, n) & x(m+1, n+1) \end{bmatrix}$$

and a 3×3 mask and its 180° rotated version are

$$h(m, n) = \begin{bmatrix} h(-1, -1) & h(-1, 0) & h(-1, 1) \\ h(0, -1) & h(0, 0) & h(0, 1) \\ h(1, -1) & h(1, 0) & h(1, 1) \end{bmatrix} \quad h(-m, -n) = \begin{bmatrix} h(1, 1) & h(1, 0) & h(1, -1) \\ h(0, 1) & h(0, 0) & h(0, -1) \\ h(-1, 1) & h(-1, 0) & h(-1, -1) \end{bmatrix}$$

Consider the 8×8 input image

$$x(m, n) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix},$$

the 3×3 structuring element and its 180° rotated version

$$h(m,n) = \begin{bmatrix} 1 & 1 & 0 \\ 0 & \mathbf{1} & 0 \\ 1 & 0 & 0 \end{bmatrix} \qquad h(-m,-n) = \begin{bmatrix} 0 & 0 & 1 \\ 0 & \mathbf{1} & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

The origin of the structuring element is shown in boldface. Assuming that the border pixels are zero-padded, the output of the dilation operation, obtained by sliding $h(-m,-n)$ over $x(m,n)$, is

$$y(m,n) = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \end{bmatrix}$$

The expression for the output is

$$y(m,n) = (x(m-1,n+1) \mid x(m,n) \mid x(m+1,n) \mid x(m+1,n+1)) = x(m,n) \oplus h(m,n)$$

Note that it is possible to decompose the 2-D structuring elements into two 1-D elements and obtain faster execution time, as in the case of 2-D convolution.

Figure 8.2a shows a 256×256 binary image and its dilated versions (b–d). The structuring elements used are

$$h4(m,n) = \begin{bmatrix} 0 & 1 & 0 \\ 1 & \mathbf{0} & 1 \\ 0 & 1 & 0 \end{bmatrix} \qquad h8(m,n) = \begin{bmatrix} 1 & 1 & 1 \\ 1 & \mathbf{1} & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

There are 8889 pixels with value 1, and the rest of the 65536 pixels are zero-valued in the image. The dilated output with $h4(m,n)$ is shown in Fig. 8.2b. The number of pixels with value 1 has increased to 9713. After 5 iterations of dilation with the same mask, the number of pixels with value 1 has increased to 12945, as shown in Fig. 8.2c. After 5 iterations of dilation with $h8(m,n)$, the number of pixels with value 1 has increased to 13965, as shown in Fig. 8.2d.

8.1.2 Erosion

The 1-D erosion operation is shown in Fig. 8.1b. The origin of the structuring element is shown in boldface. Note that there is no time-reversal required for erosion. The general expression defining 1-D erosion is given by

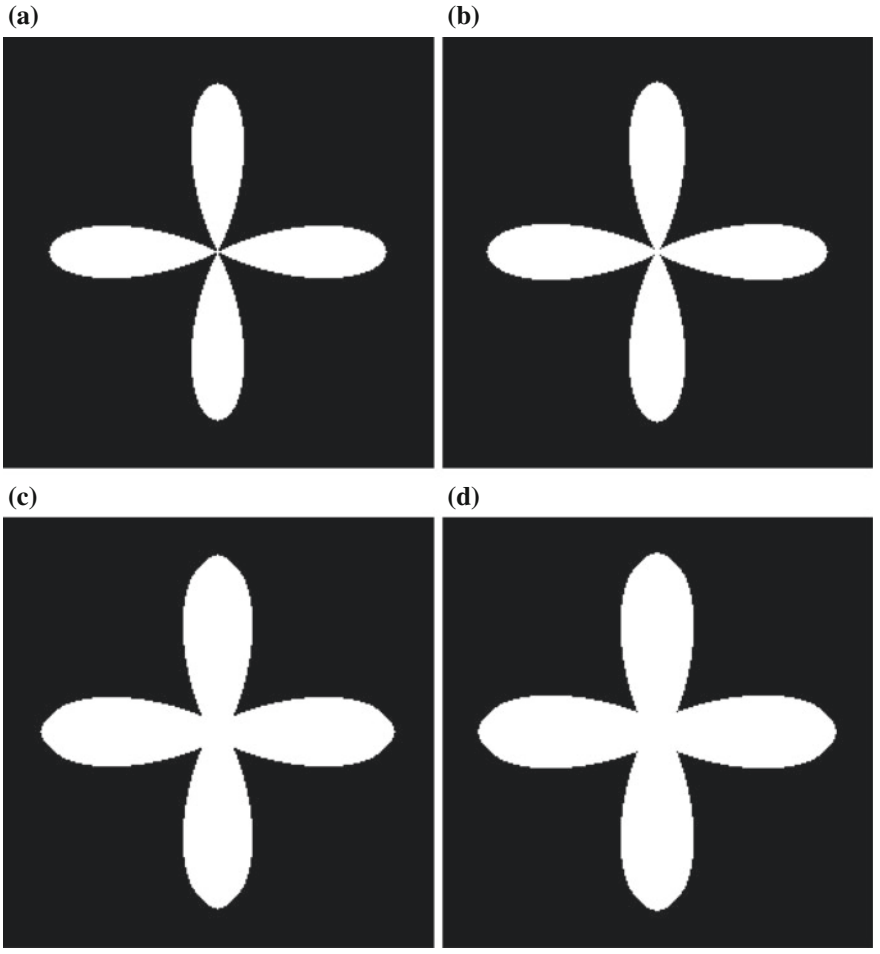


Fig. 8.2 **a** A 256×256 binary image; **b** the dilated output with $h4(m, n)$; **c** the dilated output with $h4(m, n)$ after 5 iterations; **d** the dilated output with $h8(m, n)$ after 5 iterations

$$y(n) = (h(-1)\ \&\ x(n-1))\ \&\ (h(0)\ \&\ x(n))\ \&\ (h(1)\ \&\ x(n+1)),\ \ n = 0, 1, \ldots 5$$

Only AND operations are used in the computation of erosion. For a specific $h(k)$, *the terms involving with* $h(k) = 1$ *only must be retained*. Then, the expression reduces to ANDing of all the corresponding pixels in the image. The required pixels at the borders are assumed to be 1 for erosion operation, which is similar to the linear correlation operation with the arithmetic operations replaced by logical operations. In finding the output for each pixel, the neighborhood is defined by the 1's of the structuring element. The structuring element is shifted to various positions and the

output, with the same number of elements as that of the input, is found. For the example,

$$y(0) = (1 \;\&\; 1) \;\&\; (1 \;\&\; 1) = (1 \;\&\; 1) = 1$$

If and only if all the 1's in the structuring element match up with those of the image at the corresponding positions, then the output is 1. Otherwise, the output is zero. The result is that the object is eroded or shrank. Objects may get disconnected or disappear.

The erosion of the binary image $x(m, n)$ and the structuring element $h(m, n)$ is defined as

$$y(m,n) = \;\&_k \;\&_l(h(k,l) \;\&\; x(m+k, n+l)) = x(m,n) \ominus h(m,n), \qquad (\forall k,l)h(k,l) = 1 \tag{8.2}$$

For a specific $h(k, l)$, *the terms with* $h(k, l) = 1$ *only must be retained*. Then, the expression reduces to ANDing of all the corresponding pixels in the image. The output of the erosion operation, for the same input $x(m, n)$ and structuring element $h(m, n)$ used for dilation, is obtained using

$$y(m,n) = (x(m+1, n-1) \;\&\; x(m,n) \;\&\; x(m-1,n) \;\&\; x(m-1,n-1)) = x(m,n) \ominus h(m,n)$$

The structuring element, input, and output are, respectively,

$$\begin{bmatrix} 1 & 1 & 0 \\ 0 & \mathbf{1} & 0 \\ 1 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \quad y(m,n) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

The dilation and erosion operations are shift-invariant and are not inverses of each other. Erosion is the complement of the dilation of the complement of the input with a 180° rotated mask. Dilation is the complement of the erosion of the complement of the input with a 180° rotated mask. That is,

$$x(m,n) \ominus h(m,n) = \tilde{z}(m,n), \;\; z(m,n) = \tilde{x}(m,n) \oplus h(-m,-n)$$
$$x(m,n) \oplus h(m,n) = \tilde{z}(m,n), \;\; z(m,n) = \tilde{x}(m,n) \ominus h(-m,-n)$$

Figure 8.3a shows a 256 × 256 binary image and its eroded versions (b–d). The structuring elements used are the same as that for dilation, $h4(m, n)$ and $h8(m, n)$. There are 8889 pixels with value 1, and the rest of the 65536 pixels are zero-valued in the image. The eroded output with $h4(m, n)$ is shown in Fig. 8.3b. The number of pixels with value 1 has decreased to 8077. After 5 iterations of erosion with the same mask, the number of pixels with value 1 has decreased to 5080, as shown in

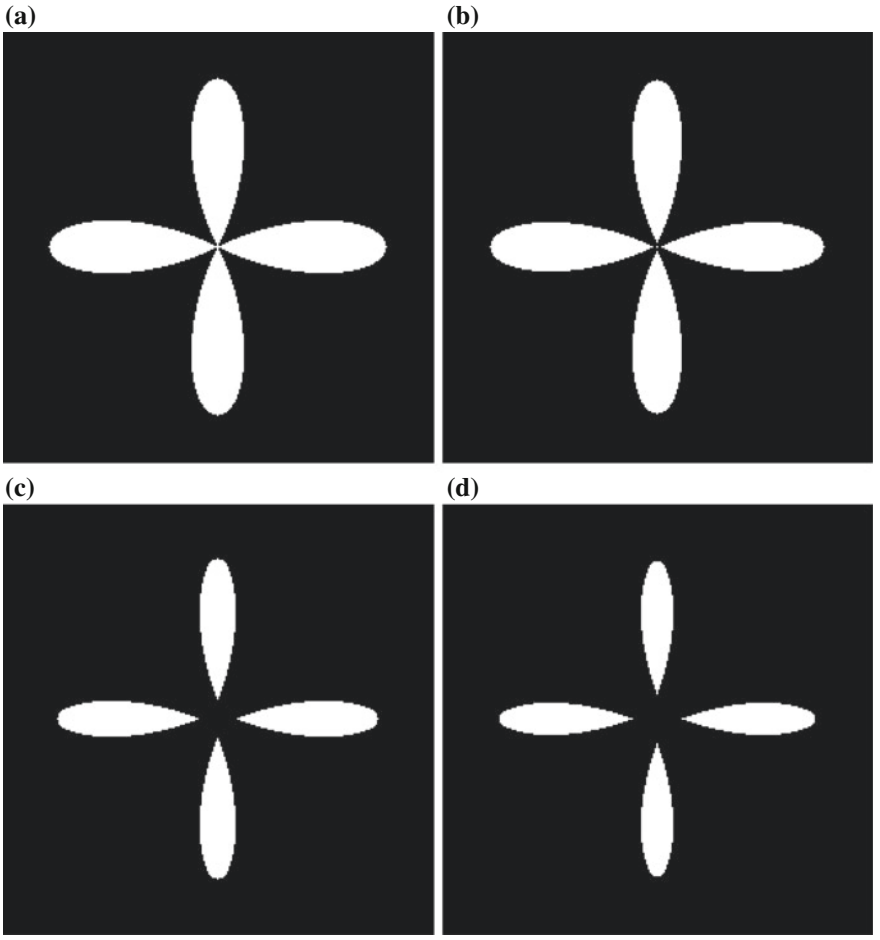


Fig. 8.3 **a** A 256×256 binary image; **b** the eroded output with $h4(m, n)$; **c** the eroded output with $h4(m, n)$ after 5 iterations; **d** the eroded output with $h8(m, n)$ after 5 iterations

Fig. 8.3c. After 5 iterations of erosion with $h8(m, n)$, the number of pixels with value 1 has decreased to 4268, as shown in Fig. 8.3d. The four components of the image are separated.

8.1.3 Opening and Closing

In these operations, the input image is subjected to both dilation and erosion. The difference is the order of these operations. The opening operation opens small gaps between touching objects in an image while the closing operation closes small gaps in an object.

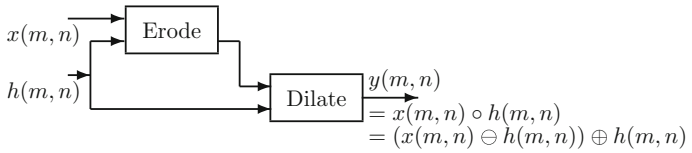


Fig. 8.4 Block diagram of the opening operation

Opening

Opening operation removes small regions of 1s. Dilation preceded by erosion is called the opening operation, defined as

$$y(m,n) = x(m,n) \circ h(m,n) = (x(m,n) \ominus h(m,n)) \oplus h(m,n)$$

The block diagram of the opening operation is shown in Fig. 8.4. The advantage of this operation is that while small objects are removed, the general shrinking of the object is avoided. Further, the object boundaries become smoother. The spatial content is also reduced.

The input image and the structuring element are the same as those used for dilation and erosion examples. The pixels on the border of the object are shown in boldface.

$$\begin{bmatrix} 1 & 1 & 0 \\ 0 & \mathbf{1} & 0 \\ 1 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 1 \\ 0 & \mathbf{1} & 0 \\ 0 & 1 & 1 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 & \mathbf{1} & \mathbf{1} & 0 & 0 \\ 0 & \mathbf{1} & 1 & \mathbf{1} & \mathbf{1} & 1 & \mathbf{1} & 0 \\ 0 & 0 & \mathbf{1} & 1 & 1 & 1 & \mathbf{1} & 0 \\ 0 & 0 & 0 & \mathbf{1} & 1 & 1 & \mathbf{1} & 0 \\ 0 & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 \end{bmatrix}$$

After erosion and, then, dilation yields,

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \quad y(m,n) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 1 & \mathbf{1} & 0 \\ 0 & 0 & 0 & 0 & \mathbf{1} & 1 & \mathbf{1} & 0 \\ 0 & 0 & 0 & \mathbf{1} & 1 & 1 & \mathbf{1} & 0 \\ 0 & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 \end{bmatrix}$$

- All the 1s in the object those are completely covered by the structuring element are preserved.
- All the 1s those can be reached by the structuring element, when it is placed at the 1s obtained in the erosion operation are also preserved.

This operation smooths the contour of an object by discarding pixels in the narrow portions. The erosion operation eliminates small objects, in addition to shrinking. The following dilation operation grows the objects back, but not the eliminated portions.

Figure 8.5a shows a 256 × 256 binary image of a grill. Structuring elements can be of arbitrary shapes and sizes to suit the purpose. Figure 8.5b shows the image subjected to opening operation by a structuring element, which is a straight line segment at 60° given by the matrix

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The portions of the image those fit in the structuring element are retained, and the rest are discarded. Figure 8.5c shows the output of the erosion operation alone. Figure 8.5d shows the image subjected to opening operation by a structuring element, which is a vertical line. Figure 8.5e shows the image subjected to opening operation by a structuring element, which is a disk given by the matrix

$$\begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \end{bmatrix}$$

Fig. 8.5f shows the image subjected to opening operation by a structuring element, which is a larger disk. The small disks in Fig. 9.5e have disappeared, and only the disk corresponding to the lock is retained.

Closing

Closing operation removes small regions of 0s. Dilation followed by erosion is called the closing operation, defined as

$$y(m,n) = x(m,n) \bullet h(m,n) = (x(m,n) \oplus h(m,n)) \ominus h(m,n)$$

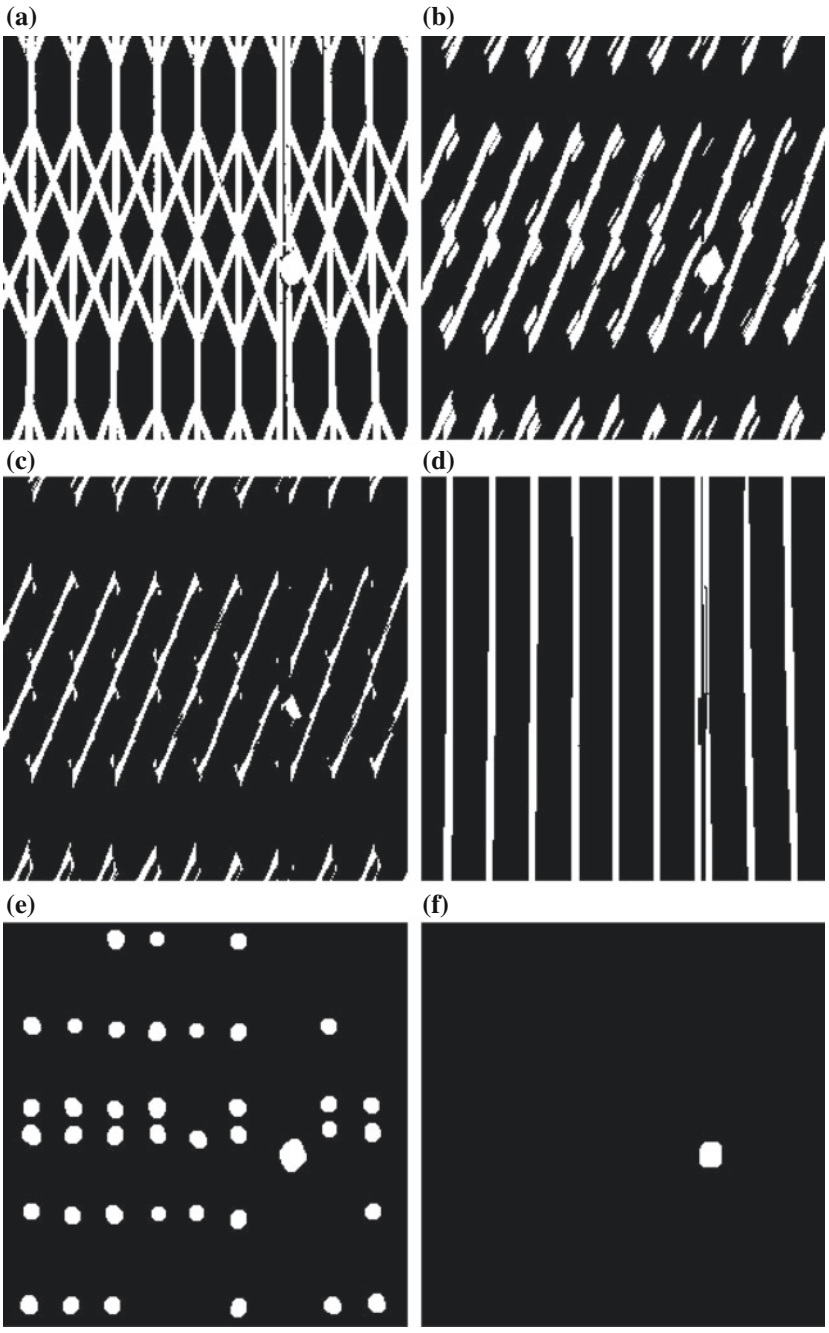


Fig. 8.5 **a** A 256 × 256 binary image; **b** the output of the opening operation by a structuring element, which is a straight line at 60°; **c** the output of the erosion operation alone; **d** the output of the opening operation by a structuring element, which is a vertical line; The outputs, **e** and **f**, by disk-shaped structuring elements

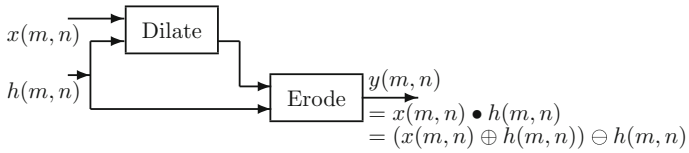


Fig. 8.6 Block diagram of the closing operation

The block diagram of the closing operation is shown in Fig. 8.6. The input image and the structuring element are the same as those used for dilation and erosion examples.

$$\begin{bmatrix} 1 & 1 & 0 \\ 0 & \mathbf{1} & 0 \\ 1 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 1 \\ 0 & \mathbf{1} & 0 \\ 0 & 1 & 1 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 & \mathbf{1} & \mathbf{1} & 0 & 0 \\ 0 & \mathbf{1} & 1 & \mathbf{1} & \mathbf{1} & 1 & \mathbf{1} & 0 \\ 0 & 0 & \mathbf{1} & 1 & 1 & 1 & \mathbf{1} & 0 \\ 0 & 0 & 0 & \mathbf{1} & 1 & 1 & \mathbf{1} & 0 \\ 0 & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 \end{bmatrix}$$

After dilation and, then, erosion yields,

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \end{bmatrix} \quad y(m,n) = \quad \begin{bmatrix} 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1} & 1 & 1 & \mathbf{1} & 0 & 0 \\ 0 & \mathbf{1} & 1 & 1 & 1 & 1 & \mathbf{1} & 0 \\ 0 & 0 & \mathbf{1} & 1 & 1 & 1 & \mathbf{1} & 0 \\ 0 & 0 & 0 & \mathbf{1} & 1 & 1 & \mathbf{1} & 0 \\ 0 & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 \end{bmatrix}$$

This operation can be expressed, in terms of opening, as

$$x(m,n) \bullet h(m,n) = \tilde{z}(m,n), \qquad z(m,n) = \tilde{x}(m,n) \circ h(-m,-n)$$

General expansion of the object is avoided. The spatial content is increased. The dilation operation changes the values of some pixels from 0 to 1 in narrow portions, in addition to expansion. The following erosion operation shrinks the object back, but leaves the 1s in the narrow portions. Further applications of opening and closing operations by the same structuring element have no effect. They are idempotent.

Figure 8.7a shows a 256 × 256 image of a set of flowers. Figure 8.7b, c show the image subjected to closing operation by structuring elements, which are 11 × 11 and 13 × 13 squares, respectively. The effect of closing is the tendency to fuse objects together by closing the gaps between them. A larger structuring element produces

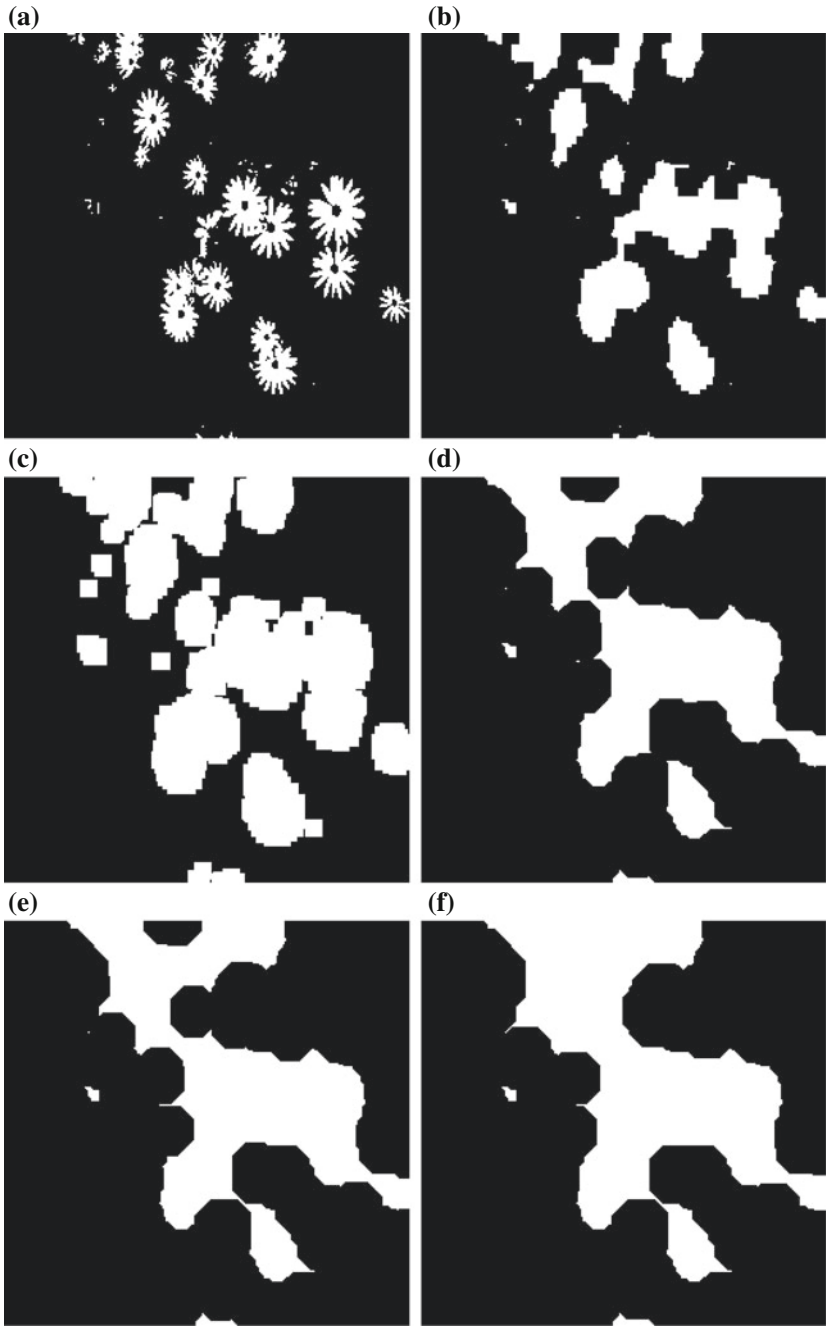


Fig. 8.7 **a** A 256 × 256 binary image of a set of flowers; the output of the closing operation by a structuring element which is **b** a 11 × 11 square; **c** a 13 × 13 square; **d** a 25 × 25 disk **e** a 29 × 29 disk; **e** a 33 × 33 disk

more fusion. Figure 8.7d, e, f show the image subjected to closing operation by structuring elements, which are 25 × 25, 29 × 29 and 33 × 33 disks, respectively. The borders of the object clearly show the nature of the square and disk structuring elements.

8.1.4 Hit-and-Miss Transformation

The erosion operation indicates the locations in an object wherever there is a match between object pixels and those of the structuring element, with no reference to the background of the image. The hit-and-miss transformation is used to detect specific objects in an image using a combination of two operators, in which the background of the image is also taken into account. This requires erosion with two nonoverlapping structuring elements. Let the input image be $x(m, n)$ and the structuring elements be $h(m, n) = \{h_h(m, n), h_{ms}(m, n)\}$. This transformation is given by

$$x(m, n) \circledast h(m, n) = (x(m, n) \ominus h_h(m, n)) \& (\tilde{x}(m, n) \ominus h_{ms}(m, n))$$

where $\tilde{x}(m, n)$ is the logical complement of $x(m, n)$. This expression is a logical AND of the erosion of $x(m, n)$ with $h_h(m, n)$ and the erosion of $\tilde{x}(m, n)$ with $h_{ms}(m, n)$. Figure 8.8 shows the block diagram of the hit-and-miss transformation. Note that there should be no overlap of elements of $h_h(m, n)$ and $h_{ms}(m, n)$. The output is a 1, when $h_h(m, n)$ matches the corresponding elements in $x(m, n)$ and $h_{ms}(m, n)$ matches the corresponding elements in $\tilde{x}(m, n)$. The occurrence of a shape can be detected. Let

$$h_h(m, n) = \begin{bmatrix} 1 & 1 & 0 \\ 1 & \mathbf{0} & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad h_{ms}(m, n) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & \mathbf{0} & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

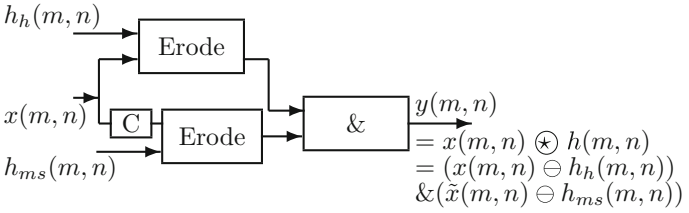


Fig. 8.8 Block diagram of the hit-and-miss transformation

The input $x(m,n)$ and its logical complement $\tilde{x}(m,n)$ are

$$x(m,n) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \quad \tilde{x}(m,n) = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \end{bmatrix}$$

The output of $(x(m,n) \ominus h_h(m,n)) = oh(m,n)$ checks only the hit pattern, and its output includes miss patterns also. The output of $(\tilde{x}(m,n) \ominus h_{ms}(m,n)) = oms(m,n)$ checks only the miss pattern and its output includes hit patterns also.

$$oh = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad oms = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}$$

The output, $y(m,n) = x(m,n) \circledast h(m,n)$, is the logical AND of oh and oms.

$$y(m,n) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The border extension for erosion is assumed to be padding with 1s.

Figure 8.9a shows a 256×256 binary image with 3 squares. It has 8 corners. Let us use the hit-and-miss transformation to find those corners. The 4 *hit* structuring elements $h_h(m,n)$ are

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & \mathbf{1} & 1 \\ 0 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 1 & 0 \\ 1 & \mathbf{1} & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 0 \\ 1 & \mathbf{1} & 0 \\ 0 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 0 \\ 0 & \mathbf{1} & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

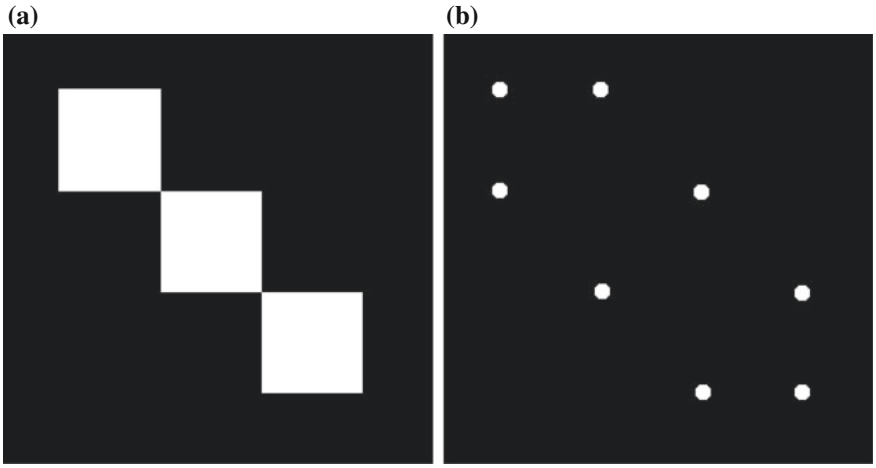


Fig. 8.9 **a** A 256 × 256 binary image; **b** the locations of its corners

The corresponding 4 $miss$ structuring elements $h_{ms}(m, n)$ are

$$\begin{bmatrix} 0 & 0 & 0 \\ 1 & \mathbf{0} & 0 \\ 1 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 0 \\ 0 & \mathbf{0} & 1 \\ 0 & 1 & 1 \end{bmatrix} \quad \begin{bmatrix} 0 & 1 & 1 \\ 0 & \mathbf{0} & 1 \\ 0 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 & 0 \\ 1 & \mathbf{0} & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

We apply the transformation using the 4 pairs of structuring elements, and the logical OR of the 4 outputs yields the exact locations of the corners shown in Fig. 8.9b. For easy visibility, we dilated the output (which is a single pixel at the location of each corner) so that the corner locations are marked by big disks.

8.1.5 Morphological Filtering

Let the image $x(m, n)$ be corrupted by impulse noise, the occurrence of random black and white pixels. The structuring element is 3 × 3 cross.

$$h(m, n) = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Then, $x(m, n) \ominus h(m, n)$ will reduce the noise due to white pixels in black areas but also enlarges the black pixels in white areas. We can reduce this effect by dilating twice.

$$((x(m, n) \ominus h(m, n)) \oplus h(m, n)) \oplus h(m, n)$$

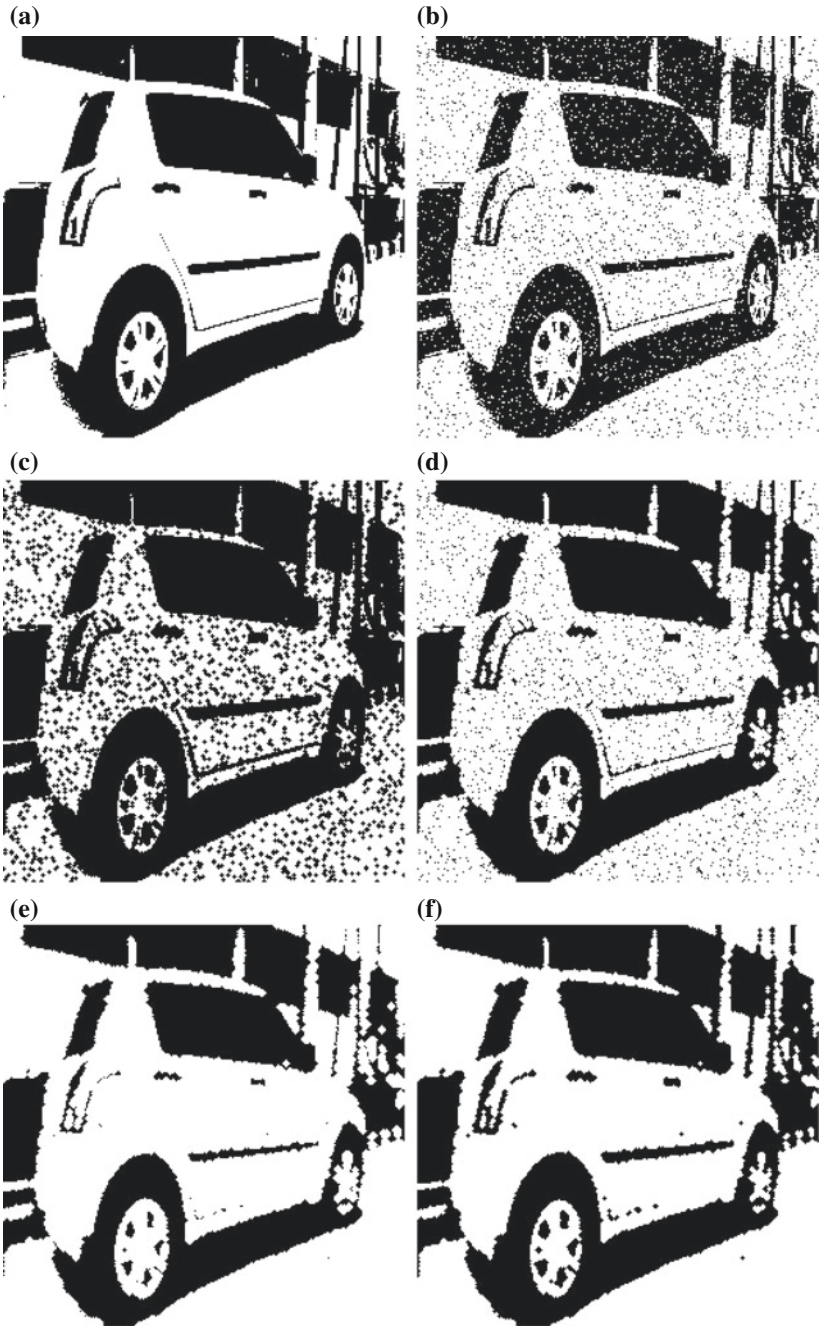


Fig. 8.10 **a** A 256 × 256 binary image; **b** its noisy version; **c** the image in **b** after erosion; **d** the image in **c** after dilation; **e** the image in **d** after dilation; **f** the image in **e** after erosion

Now, the noise is reduced but the black areas got shrunk. To restore the image properly, another erosion is required.

$$(((x(m,n) \ominus h(m,n)) \oplus h(m,n)) \oplus h(m,n)) \ominus h(m,n)$$

That is opening followed by closing

$$y = (x(m,n) \circ h(m,n)) \bullet h(m,n)$$

Figure 8.10a, b show a 256×256 binary image and its noisy version, respectively. The image in (b) after erosion, with enlarged black pixels, is shown in (c). The noise is reduced after two dilations of (c), as shown in (d) and (e). The filtered image, shown in (f), is obtained after the erosion of the image in (e). While the noise is effectively removed, some undesirable breaks in the image have appeared since no connectivity condition was imposed.

8.2 Binary Morphological Algorithms

8.2.1 Thinning

In thinning, an object without holes is reduced to a minimally connected stroke such that it is located equidistant from its nearest outer boundaries. An object with a hole is reduced to a minimally connected ring in the center of the object. The object is reduced to 1-pixel width without breaking and shortening.

The 3×3 neighborhood pixels of pixel p are

$$\begin{bmatrix} w(0,0) & w(0,1) & w(0,2) \\ w(1,0) & p & w(1,2) \\ w(2,0) & w(2,1) & w(2,2) \end{bmatrix}$$

Each iteration of this thinning algorithm, presented as MATLAB code, consists of two parts. In the first part, a pixel p is deleted only when the following three conditions are satisfied.

```
b1=0;b2=0;b3=0;b4=0;
if w(1,2) == 0 && ( w(0,2) == 1 || w(0,1) == 1)
  b1 = 1;
end
if w(0,1) == 0 && ( w(0,0) == 1 || w(1,0) == 1)
  b2 = 1;
end
if w(1,0) == 0 && ( w(2,0) == 1 || w(2,1) == 1)
  b3 = 1;
end
if w(2,1) == 0 && ( w(2,2) == 1 || w(1,2) == 1)
```

```
  b4 = 1;
end
c1 = b1 + b2 + b3 + b4 ; % Condition 1
s1 = (w(1,2) | w(0,2)) + (w(0,1) | w(0,0)) + (w(1,0) | w(2,0)) + (w(2,1) | w(2,2));
s2 = (w(0,2) | w(0,1)) + (w(0,0) | w(1,0)) + (w(2,0) | w(2,1)) + (w(2,2) | w(1,2));
c2 = min(s1,s2); % Condition 2
c3 = ( w(0,2) | w(0,1) | ~w(2,2) ) & w(1,2) ; % Condition 3
if c3 == 0 && c2>=2 && c2<=3 && c1 == 1
  p = 0 ; % setting the pixel value
end
```

In the second part, a pixel is deleted only when the first two conditions of the first part are satisfied along with the following third condition.

```
c3 = ( w(2,0) | w(2,1) | ~w(0,0) ) & w(1,0) ; % Condition 3
if c3 == 0 && c2>=2 && c2<=3 && c1 == 1
  p = 0 ; % setting the pixel value
end
```

The iterations continue until there is no change in the output from the previous iteration.

Let the input image $x(m, n)$ be

$$x(m,n) = \begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

It is assumed that the image is zero-padded on all sides.
Consider the pixel $x(0, 2)$. Its neighborhood is

$$\begin{bmatrix} w(0,0) & w(0,1) & w(0,2) \\ w(1,0) & p & w(1,2) \\ w(2,0) & w(2,1) & w(2,2) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

Now, $b1 = b2 = b3 = 0$, $b4 = 1$ and $c1 = 1$.
$s1 = 1 + 0 + 0 + 1 = 2$, $s2 = 0 + 0 + 0 + 1 = 1$ and the minimum of $s1$ and $s2$ is $c2 = 1$.
$c3 = (0|0|0)\&1 = 0$.
Now, the condition $c2 \geq 2$ fails and the pixel is not set to 0.
Consider the pixel $x(0, 3)$. Its neighborhood is

$$\begin{bmatrix} w(0,0) & w(0,1) & w(0,2) \\ w(1,0) & p & w(1,2) \\ w(2,0) & w(2,1) & w(2,2) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

Now, $b1 = b4 = b3 = 0$, $b2 = 1$ and $c1 = 1$.
$s1 = 1 + 0 + 1 + 1 = 3$, $s2 = 1 + 0 + 1 + 1 = 3$ and the minimum of $s1$ and $s2$ is $c2 = 3$.
$c3 = (0|0|0)\&1 = 0$.
Now, the condition for setting the pixel 0 is satisfied, and the pixel is set to 0. The output of the first part of the first iteration of the algorithm is

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Pixel $x(2, 4)$ is to be set 0, but it is not done in the first part of the algorithm. Its neighborhood is

$$\begin{bmatrix} w(0,0) & w(0,1) & w(0,2) \\ w(1,0) & p & w(1,2) \\ w(2,0) & w(2,1) & w(2,2) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

and $c3 = (1|1|0)\&1 = 1$.
In the second part of the algorithm, its neighborhood is

$$\begin{bmatrix} w(0,0) & w(0,1) & w(0,2) \\ w(1,0) & p & w(1,2) \\ w(2,0) & w(2,1) & w(2,2) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

$c1 = 1$, $c2 = 3$, and $c3 = (0|0|0)\&0 = 0$. Now, the condition for setting the pixel 0 is satisfied, and the pixel is set to 0. The thinned output of the input image is

$$y(m, n) = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure 8.11a shows a 256×256 binary image. The output of the thinning algorithm after 4, 7, and 11 iterations are shown in Fig. 8.11b–d. The number of 1s in the

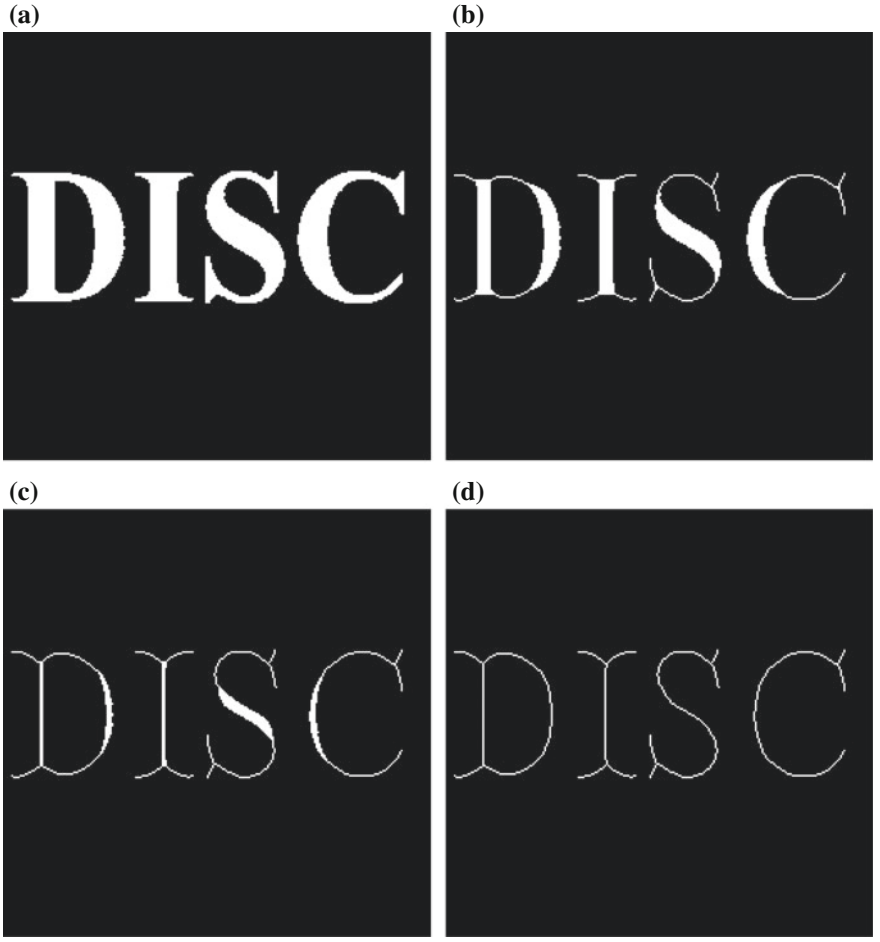


Fig. 8.11 **a** A 256 × 256 binary image; the output of the thinning algorithm after **b** 4 iterations, **c** 7 iterations, and **d** 11 iterations

input image is 7326 and those after 4, 7, and 11 iterations are 3125, 1235, and 692, respectively. The algorithm terminates after the 11th iteration.

8.2.2 Thickening

Thickening of an image can be carried out by:

- Complement the input image $x(m, n)$ to get $\tilde{x}(m, n)$.
- Thin $\tilde{x}(m, n)$ to get $\tilde{y}(m, n)$.
- Complement $\tilde{y}(m, n)$ to get the thickened input image $y(m, n)$.

Additional processing may be required after each iteration, in case of extraneous pixels appearing in the output.

Consider the input image and its complement

$$x(m,n) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \tilde{x}(m,n) = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

The thickened image after 1 and 2 iterations is, respectively

$$y1(m,n) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad y2(m,n) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The input image has 2 1s and it has become 5 and 13 after 1 and 2 iterations of thickening.

Figure 8.12a shows a 256×256 binary image. The output of the thickening operation after 1, 2, and 3 iterations are shown in Fig. 8.12b–d, respectively. The number of 0s in the input image is 2112. Due to thickening, this number increases to 4577, 6968, and 9342 in Fig. 8.12b–d.

8.2.3 Noise Removal

Isolated pixels with value 1 in a neighborhood of 0s are replaced by 0. Consider the mask

$$h(m,n) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Let the input image be $x(m,n)$. Then, pixels with value 1 in the output image are given by

$$\begin{aligned} y(m,n) = {} & (x(m,n)\&(x(m-1,n)|x(m-1,n-1)| \\ & x(m-1,n+1)|x(m,n-1)|x(m,n+1)|x(m+1,n)|x(m+1,n+1)|x(m+1,n-1))) \end{aligned}$$

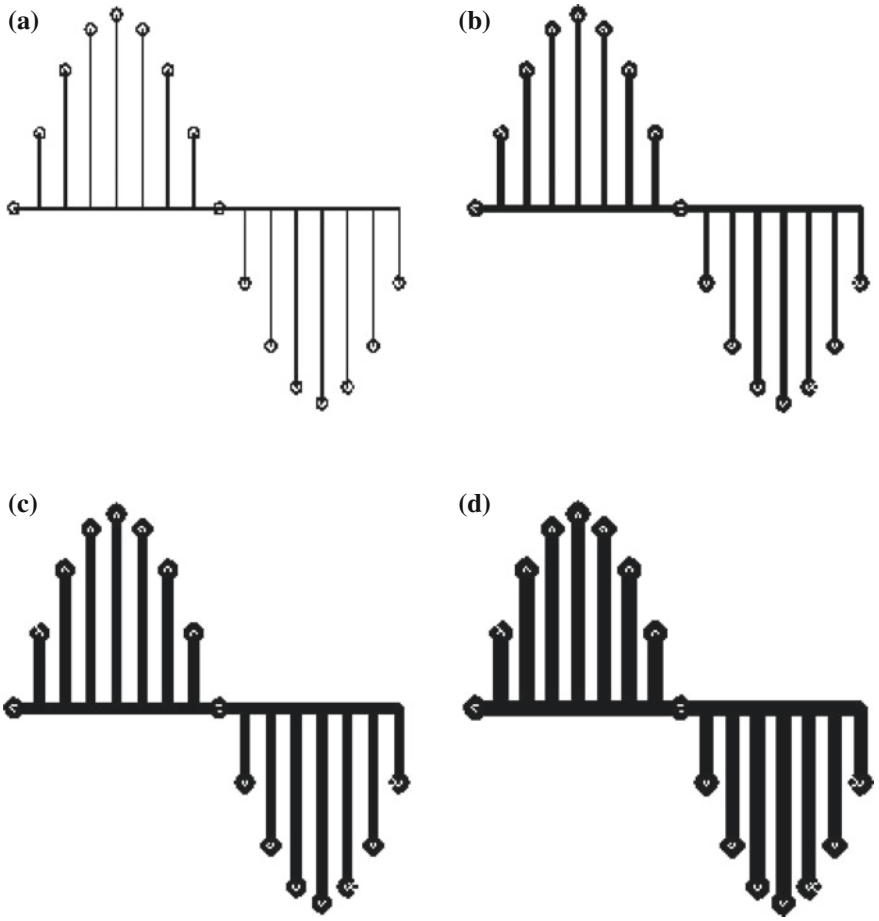


Fig. 8.12 **a** A 256 × 256 binary image; the output of the thickening operation after **b** 1 iteration, **c** 2 iterations, and **d** 3 iterations

For example, the input and output are

$$x(m,n) = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \quad y(m,n) = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Pixels $x(5, 0)$ and $x(2, 5)$ have been replaced by zeros. Border pixels are assumed to be 0s.

8.2.4 Skeletons

Skeleton is another description of a shape of an object. It is the axis, which is equidistant from the borders of a shape. It is a central outline of the object. One way to find the skeleton is to use the distance transform, presented in Chap. 10. The distance transform is an algorithm that approximates the Euclidean distance of each point in the image to a region faster.

Let the input image be $x(m, n)$ and the distance of the pixels from the region marked with 1s in the complement of $x(m, n)$, $\tilde{x}(m, n)$, be $D(m, n)$.

$$x(m,n) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad D(mn,) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 5 & 5 & 5 & 5 & 5 & 5 & 0 \\ 0 & 5 & 10 & 10 & 10 & 10 & 5 & 0 \\ 0 & 5 & 10 & 15 & 15 & 10 & 5 & 0 \\ 0 & 5 & 10 & 15 & 15 & 10 & 5 & 0 \\ 0 & 5 & 10 & 10 & 10 & 10 & 5 & 0 \\ 0 & 5 & 5 & 5 & 5 & 5 & 5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The distances shown in $D(m, n)$ are scaled by a factor of 5. For example, the distance of the pixel $\tilde{x}(4, 4)$ from its nearest pixel with value 1 is 3, which after scaling by 5 gives 15. The reason for complementing is that it is easier to comprehend the distance image.

We start with a matrix $skel(m, n)$, of the same size as the input image, with all zero entries. Each pixel in this matrix is replaced by a 1, if the corresponding value in $D(m, n)$ is greater or equal to the largest value of its 4 nearest neighbors. For example, consider the neighborhood of $D(1, 1)$.

$$\begin{bmatrix} & 0 & \\ 0 & 5 & 5 \\ & 5 & \end{bmatrix}$$

As $D(1, 1) = 5$ is greater or equal to the largest value of its 4 nearest neighbors, $skel(1, 1) = 1$. The skeleton of the input image is

$$skel(m,n) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure 8.13a shows a 256×256 binary image. Figure 8.13b shows its complement. Figure 8.13c shows its distance transform representation. The central axis is quite

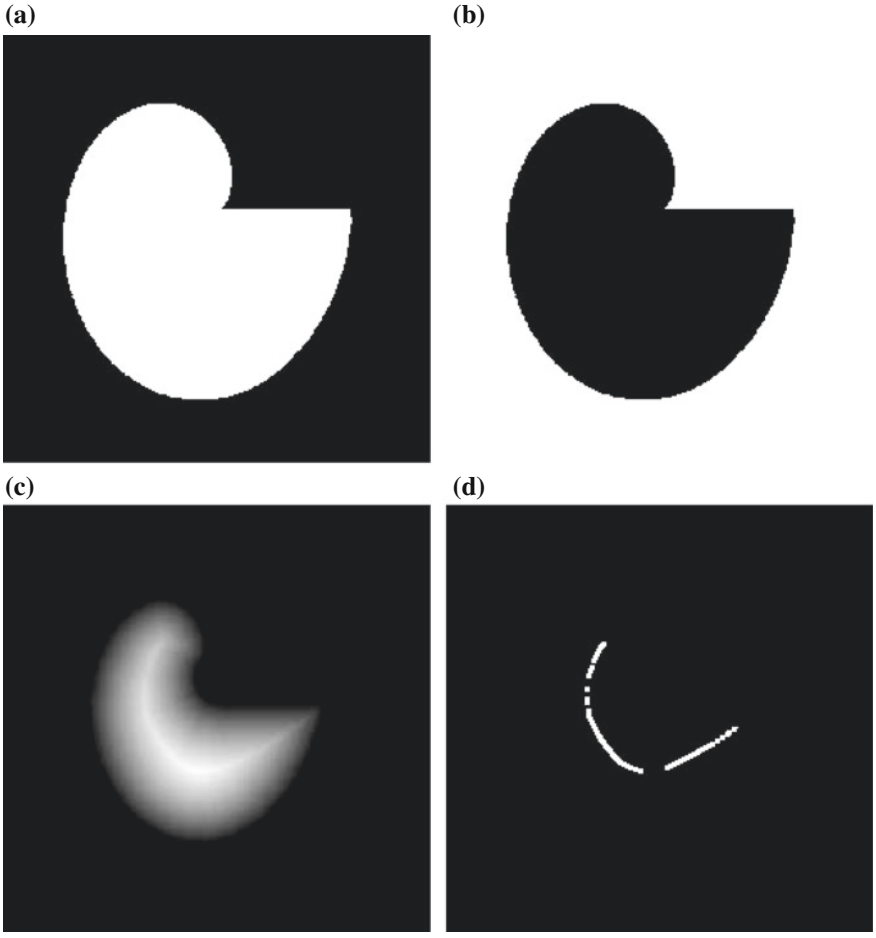


Fig. 8.13 **a** A 256 × 256 binary image; **b** its complement; **c** the distance transform representation; **d** the skeleton

clear. Figure 8.13d shows its skeleton. We have dilated the actual output once to get a thick line. The skeleton is not connected, as no connectivity check was made. Thinning algorithms yield skeletons with connectivity guaranteed.

8.2.5 Fill

Isolated pixels with value 0 in a neighborhood of 1s are replaced by 1. Consider the mask

$$h(m,n) = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Let the input image be $x(m,n)$. Then, pixels with value 1 in the output image is given by

$$y(m,n) = (x(m,n)|(x(m-1,n)\&x(m-1,n-1)\&x(m-1,n+1)\& \\ x(m,n-1)\&x(m,n+1)\&x(m+1,n)\&x(m+1,n+1)\&x(m+1,n-1)))$$

For example, the input and output are

$$x(m,n) = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \quad y(m,n) = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Pixel $x(1,1)$ has been replaced by 1. Pixels $x(4,4)$ and $x(5,5)$ will be replaced by 1s with a 4-connected neighborhood. Border pixels are assumed to be 0s.

8.2.6 Boundary Extraction

If we erode an image by a structuring element by one iteration, then the pixels in the border of the objects are set to zero, leaving the interior pixels unchanged. Now, if we subtract the output of erosion from the input, an image with object boundary is obtained. Consider the mask

$$h(m,n) = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Let the input image be $x(m,n)$. Then, pixels with value 1 in the eroded image are given by

$$y(m,n)=(x(m,n)\&(x(m-1,n)\&x(m-1,n-1)\&x(m-1,n+1)\& \\ x(m,n-1)\&x(m,n+1)\&x(m+1,n)\&x(m+1,n+1)\&x(m+1,n-1)))$$

For example, let

$$x(m,n) = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The output of erosion and the extracted border are, respectively,

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Border pixels are assumed to be 0s.

Figure 8.14a, b show, respectively, a 256 × 256 binary image and its boundary obtained using a 3 × 3 square structuring element.

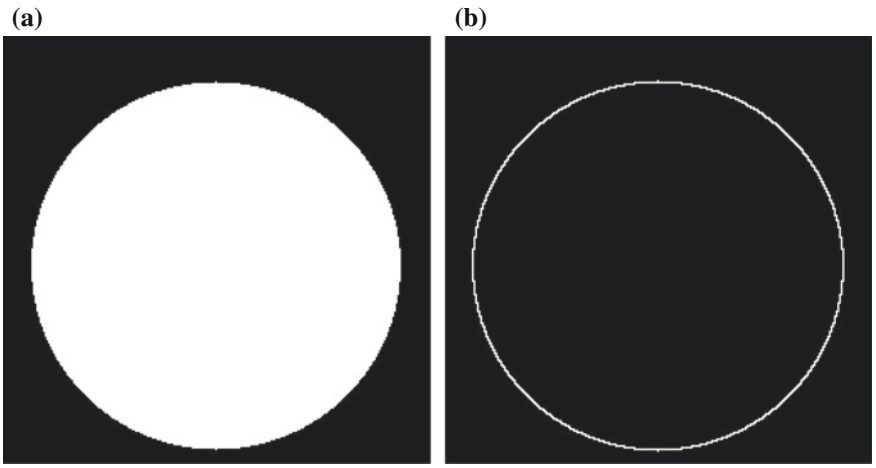


Fig. 8.14 **a** A 256 × 256 binary image and **b** its boundary

8.2.7 Region Filling

Given a region defined by its boundary and the location of a pixel within it, the interior of the region is to be filled. Let the image be $x(m, n)$. The algorithm is defined by

$$x_l(m, n) = (x_{l-1}(m, n) \oplus h(m, n)) \& \tilde{x}(m, n), \quad l = 2, 3, \ldots$$

where

$$h(m, n) = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

and $x_1(m, n)$ is a matrix of the same size as $x(m, n)$ with all entries zero except a 1 at the given location inside the region. Repeatedly, we keep dilating the current $x_l(m, n)$ with $h(m, n)$ and AND with the complement of the input image until there is no difference between two consecutive versions of $x_l(m, n)$. Without the AND operation, the dilation operation is uncontrolled and will fill up the entire image.

Let $x(m, n)$ be the given image and (3, 3) is the given starting location. Then,

$$x(m, n) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad x_1(m, n) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$x_2(m, n) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad x_3(m, n) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$x_4(m, n) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad x_5(m, n) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The OR of $x_5(m, n)$ and $x(m, n)$ gives the filled region.

8.2.8 Extraction of Connected Components

This algorithm is similar to the region filling algorithm except that the dilated image is combined with the input image (rather than its complement, as it is in region filling algorithm) by the AND operator. It is defined by

$$x_l(m,n) = (x_{l-1}(m,n) \oplus h(m,n)) \& x(m,n), \quad l = 2, 3, \ldots$$

Let the input image be

$$x(m,n) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The image has 2 connected components. Let us try the algorithm with a 3×3 square structuring element and initial location (3, 3). As all the 1s are reachable from the initial location, the algorithm finds the connected component, on the left, in one (dilation) iteration. Let the initial location be (6, 6). Then, the algorithm requires 3 iterations to find the connected component, on the right.

8.2.9 Convex Hull

If the straight line segment connecting any pair of points in an image lies within the image, then it is said to be convex. For example, a straight line, an ellipse, and a disk are all convex. Given an object, it is useful to find the minimal convex set, called the convex hull, containing the object. The concept of convex hull is useful in object description. The convex hull, $c(m,n)$, of an image, $x(m,n)$, can be found using the algorithm

$$c_k^l(m,n) = c_{k-1}^l(m,n) \circledast h^l(m,n) \mid x(m,n) \quad l = 0, 1, 2, 3 \quad \text{and } k = 1, 2, \ldots$$

where $c_0^l(m,n) = x(m,n)$ and $h^l(m,n)$ are the structuring elements used. When the algorithm converges, for each $h^l(m,n)$, $c_k^l(m,n) = c_{k-1}^l(m,n)$. Let the four outputs be $c0(m,n)$, $c1(m,n)$, $c2(m,n)$, and $c3(m,n)$. Then, the convex hull is

$$c(m,n) = (c0(m,n) \mid c1(m,n) \mid c2(m,n) \mid c3(m,n))$$

Let the four structuring elements $h(m, n)$ in four directions are

$$\begin{bmatrix} 1 & x & x \\ 1 & \mathbf{0} & x \\ 1 & x & x \end{bmatrix} \quad \begin{bmatrix} x & x & x \\ x & \mathbf{0} & x \\ 1 & 1 & 1 \end{bmatrix} \quad \begin{bmatrix} x & x & 1 \\ x & \mathbf{0} & 1 \\ x & x & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 & 1 \\ x & \mathbf{0} & x \\ x & x & x \end{bmatrix}$$

The structuring elements are 90° rotated versions. The entry x indicates don't care condition. If the image pixel corresponding to the center of the mask is zero and the other 3 corresponding pixels are 1s, then the pixel at the center is given the value 1. An image and its convex hull using this algorithm are

$$x(m,n) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad c(m,n) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

There are 14 1s in the input image and 21 1s in the convex set. The first mask does not match at any place in the image and contributes nothing to the output. The outputs are found for pixels within the object area only. The second mask makes $x(3, 5) = 1$. The third mask makes $x(5, 5) = 1$ in the first iteration and $x(5, 4) = 1$ in the second. The fourth mask makes $x(3, 4) = 1$ and $x(5, 5) = 1$ in the first iteration, $x(4, 3) = 1$ and $x(5, 4) = 1$ in the second iteration, $x(5, 3) = 1$, in the third iteration and $x(6, 4) = 1$ in the fourth iteration. The convex hull is the combination of these results by OR (|) operation. The algorithm may not give the minimal convex hull, for which a more complex algorithm is required.

8.2.10 Pruning

Pruning removes undesirable short spurs after operations such as thinning. Let us define an operation

$$x(m,n) \otimes w(m,n) = x(m,n) \;\&\; \tilde{z}(m,n), \qquad z(m,n) = (x(m,n) \circledast w(m,n))$$

with input image $x(m, n)$ and 8 masks

$$w1(m,n) = \begin{bmatrix} \times & 0 & 0 \\ 1 & \mathbf{1} & 0 \\ \times & 0 & 0 \end{bmatrix}, \quad w2(m,n) = \begin{bmatrix} \times & 1 & \times \\ 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad w3(m,n) = \begin{bmatrix} 0 & 0 & \times \\ 0 & \mathbf{1} & 1 \\ 0 & 0 & \times \end{bmatrix},$$

$$w4(m,n) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 \\ \times & 1 & \times \end{bmatrix}, w5(m,n) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad w6(m,n) = \begin{bmatrix} 0 & 0 & 1 \\ 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 \end{bmatrix},$$

$$w7(m,n) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad w8(m,n) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

The masks are 90° rotated versions.

At the pixel $x(m,n)$, the output of the hit-and-miss transformation with mask $w1(m,n)$ is

$$o1 = x(m,n-1)\&x(m,n)\&\tilde{x}(m,n+1)\&\tilde{x}(m+1,n)\&\tilde{x}(m+1,n+1) \\ \&\tilde{x}(m-1,n)\&\tilde{x}(m-1,n+1)$$

This operation outputs a 1 if the neighbors of $x(m,n)$ match $w1(m,n)$. Then,

$$x(m,n) \otimes w1(m,n) = x(m,n)\ \&\ \tilde{o1}$$

If $o1 = 1$, pixel $x(m,n)$ is assigned 0. The sequence of 8 masks are used in the given order to get the pruned output. That is,

$$x(m,n) \otimes \{w(m,n)\} = (((((((x(m,n) \otimes w1(m,n)) \otimes w2(m,n)) \otimes w3(m,n)) \otimes w4(m,n)) \\ \otimes w5(m,n)) \otimes w6(m,n)) \otimes w7(m,n)) \otimes w8(m,n)$$

Pruning is carried out the required number of times. Some end points may also be lost in the process. Therefore, all the end points are reconnected at the end from the list of deleted pixels.

8.3 Grayscale Morphology

Grayscale morphology is an extension of that of the binary images, the binary morphology. In erosion of binary images, the image becomes darker since the number of pixels with zero value increases. In dilation, the image becomes lighter. The same effect is achieved in gray-level images also. Instead of the logical operators, we use the minimum and maximum functions. The origin of the structuring element must be noted. In the case of dilation operation, the structuring element has to be rotated 180° about its origin. We present a commonly used version of grayscale morphology.

8.3.1 Dilation

The dilation of the gray-level image $x(m, n)$ and the window or mask or structuring element $h(m, n)$ is defined as

$$y(m, n) = \max_{k,l}\{x(m - k, n - l)\} = x(m, n) \oplus h(m, n) \tag{8.3}$$

where m and n vary over the dimensions of the image, and k and l vary over the dimensions of the structuring element. For a 3×3 neighborhood with the origin at the center,

$$\begin{aligned} y(m, n) = \max\{&(x(m - 1, n - 1), x(m - 1, n), x(m - 1, n + 1), x(m, n - 1), \\ &x(m, n), x(m, n + 1), x(m + 1, n - 1, x(m + 1, n), x(m + 1, n + 1)\}) = x(m, n) \oplus h(m, n) \end{aligned}$$

Consider the 8×8 input image and the 3×3 structuring element.

$$x(m, n) = \begin{bmatrix} 88 & 100 & 104 & 101 & 114 & 110 & 110 & 107 \\ 92 & 102 & 104 & 107 & 112 & 110 & 104 & 92 \\ 103 & 105 & 111 & 112 & 114 & 108 & 86 & 39 \\ 106 & 107 & 112 & 113 & 107 & 73 & 26 & 25 \\ 111 & 114 & 14 & 104 & 64 & 23 & 25 & 28 \\ 117 & 115 & 97 & 45 & 15 & 13 & 23 & 29 \\ 119 & 93 & 32 & 0 & 15 & 11 & 19 & 23 \\ 88 & 15 & 0 & 2 & 10 & 13 & 11 & 15 \end{bmatrix}$$

Assuming that the border pixels are replicated, the output of the dilation operation is

$$y(m, n) = \begin{bmatrix} 102 & 104 & 107 & 114 & 114 & 114 & 110 & 110 \\ 105 & 111 & 112 & 114 & 114 & 114 & 110 & 110 \\ 107 & 112 & 113 & 114 & 114 & 114 & 110 & 104 \\ 114 & 114 & 114 & 114 & 114 & 114 & 108 & 86 \\ 117 & 117 & 115 & 113 & 113 & 107 & 73 & 29 \\ 119 & 119 & 115 & 104 & 104 & 64 & 29 & 29 \\ 119 & 119 & 115 & 97 & 45 & 23 & 29 & 29 \\ 119 & 119 & 93 & 32 & 15 & 19 & 23 & 23 \end{bmatrix}$$

For example, $y(1, 1) = 111$, which is the maximum in the neighborhood of $x(1, 1) = 102$.

8.3.2 Erosion

The erosion of the gray-level image $x(m, n)$ and the window or mask or structuring element $h(m, n)$ is defined as

$$y(m, n) = \min_{k,l}\{x(m + k, n + l)\} = x(m, n) \ominus h(m, n) \tag{8.4}$$

where m and n vary over the dimensions of the image, and k and l vary over the dimensions of the structuring element. For a 3×3 neighborhood with the origin at the center,

$$\begin{aligned} y(m, n) = \min\{&(x(m-1, n-1), x(m-1, n), x(m-1, n+1), x(m, n-1), \\ &x(m, n), x(m, n+1), x(m+1, n-1, x(m+1, n), x(m+1, n+1)\}) = x(m, n) \ominus h(m, n) \end{aligned}$$

For the same image and the structuring element, assuming that the border pixels are replicated, the output of the erosion operation is

$$y(m, n) = \begin{bmatrix} 88 & 88 & 100 & 101 & 101 & 104 & 92 & 92 \\ 88 & 88 & 100 & 101 & 101 & 86 & 39 & 39 \\ 92 & 92 & 102 & 104 & 73 & 26 & 25 & 25 \\ 103 & 14 & 14 & 14 & 23 & 23 & 23 & 25 \\ 106 & 14 & 14 & 14 & 13 & 13 & 13 & 23 \\ 93 & 14 & 0 & 0 & 0 & 11 & 11 & 19 \\ 15 & 0 & 0 & 0 & 0 & 10 & 11 & 11 \\ 15 & 0 & 0 & 0 & 0 & 10 & 11 & 11 \end{bmatrix}$$

For example, $y(1, 1) = 88$, which is the minimum in the neighborhood of $x(1, 1) = 102$.

8.3.3 Opening and Closing

These operations are defined and used the same way as in binary morphology.

$$y(m, n) = x(m, n) \circ h(m, n) = (x(m, n) \ominus h(m, n)) \oplus h(m, n)$$

$$y(m, n) = x(m, n) \bullet h(m, n) = (x(m, n) \oplus h(m, n)) \ominus h(m, n)$$

For the same image and the structuring element, assuming that the border pixels are replicated, the output of the opening operation is

$$y(m,n) = \begin{bmatrix} 88 & 100 & 101 & 101 & 104 & 104 & 104 & 92 \\ 92 & 102 & 104 & 104 & 104 & 104 & 104 & 92 \\ 103 & 103 & 104 & 104 & 104 & 101 & 86 & 39 \\ 106 & 106 & 104 & 104 & 104 & 73 & 26 & 25 \\ 106 & 106 & 14 & 23 & 23 & 23 & 25 & 25 \\ 106 & 106 & 14 & 14 & 14 & 13 & 23 & 23 \\ 93 & 93 & 14 & 0 & 11 & 11 & 19 & 19 \\ 15 & 15 & 0 & 0 & 10 & 11 & 11 & 11 \end{bmatrix}$$

The output of the closing operation is

$$y(m,n) = \begin{bmatrix} 102 & 102 & 104 & 107 & 114 & 110 & 110 & 110 \\ 102 & 102 & 104 & 107 & 114 & 110 & 104 & 104 \\ 105 & 105 & 111 & 112 & 114 & 108 & 86 & 86 \\ 107 & 107 & 112 & 113 & 107 & 73 & 29 & 29 \\ 114 & 114 & 104 & 104 & 64 & 29 & 29 & 29 \\ 117 & 115 & 97 & 45 & 23 & 23 & 23 & 29 \\ 119 & 93 & 32 & 15 & 15 & 15 & 19 & 23 \\ 119 & 93 & 32 & 15 & 15 & 15 & 19 & 23 \end{bmatrix}$$

8.3.4 Top-Hat and Bottom-Hat Transformations

The top-hat transformation is defined as the subtraction of its opening from the image $x(m,n)$.

$$y(m,n) = x(m,n) - (x(m,n) \circ h(m,n))$$

For the same image and the structuring element, assuming that the border pixels are replicated, the output of the top-hat transformation is

$$y(m,n) = \begin{bmatrix} 0 & 0 & 3 & 0 & 10 & 6 & 6 & 15 \\ 0 & 0 & 0 & 3 & 8 & 6 & 0 & 0 \\ 0 & 2 & 7 & 8 & 10 & 7 & 0 & 0 \\ 0 & 1 & 8 & 9 & 3 & 0 & 0 & 0 \\ 5 & 8 & 0 & 81 & 41 & 0 & 0 & 3 \\ 11 & 9 & 83 & 31 & 1 & 0 & 0 & 6 \\ 26 & 0 & 18 & 0 & 4 & 0 & 0 & 4 \\ 73 & 0 & 0 & 2 & 0 & 2 & 0 & 4 \end{bmatrix}$$

One of the applications of this transformation is in shading correction.

The bottom-hat transformation is defined as the subtraction of the image $x(m,n)$ from its closing.

$$y(m,n) = (x(m,n) \bullet h(m,n)) - x(m,n)$$

For the example $x(m, n)$, the output of the bottom-hat transformation is

$$y(m,n) = \begin{bmatrix} 14 & 2 & 0 & 6 & 0 & 0 & 0 & 3 \\ 10 & 0 & 0 & 0 & 2 & 0 & 0 & 12 \\ 2 & 0 & 0 & 0 & 0 & 0 & 0 & 47 \\ 1 & 0 & 0 & 0 & 0 & 0 & 3 & 4 \\ 3 & 0 & 90 & 0 & 0 & 6 & 4 & 1 \\ 0 & 0 & 0 & 0 & 8 & 10 & 0 & 0 \\ 0 & 0 & 0 & 15 & 0 & 4 & 0 & 0 \\ 31 & 78 & 32 & 13 & 5 & 2 & 8 & 8 \end{bmatrix}$$

This transformation is often used in contrast enhancement.

8.3.5 Morphological Gradient

The morphological gradient of the image $x(m, n)$ is defined as

$$y(m,n) = (x(m,n) \oplus h(m,n)) - (x(m,n) \ominus h(m,n))$$

For the example $x(m, n)$,

$$y(m,n) = \begin{bmatrix} 14 & 16 & 7 & 13 & 13 & 10 & 18 & 18 \\ 17 & 23 & 12 & 13 & 13 & 28 & 71 & 71 \\ 15 & 20 & 11 & 10 & 41 & 88 & 85 & 79 \\ 11 & 100 & 100 & 100 & 91 & 91 & 85 & 61 \\ 11 & 103 & 101 & 99 & 100 & 94 & 60 & 6 \\ 26 & 105 & 115 & 104 & 104 & 53 & 18 & 10 \\ 104 & 119 & 115 & 97 & 45 & 13 & 18 & 18 \\ 104 & 119 & 93 & 32 & 15 & 9 & 12 & 12 \end{bmatrix}$$

In a smooth region, the difference is small, and it is large in the vicinity of an edge.

8.4 Summary

- Mathematical morphology is the study of form and shapes of objects.
- The basic operations, in the analysis of binary images, are in the form of convolution and correlation with the difference that arithmetic operations are replaced by logical operations. Similar to realizing filters with different frequency responses by using suitable impulse response in the convolution operation, various types of shape analysis can be carried out using different structuring elements (windows or masks of 1s and 0s) in morphology.

- Two operations, dilation and erosion, are fundamental in morphology. All other operations are essentially a combination of these two operations.
- Dilation enlarges an object while erosion shrinks it.
- Pattern matching, thinning, thickening, filtering, region filling, boundary extraction, and pruning are some of the tasks carried out in morphology.
- Morphology of gray-level images is an extension of that of binary images. Minimum and maximum functions replace the logical operations used in the morphology of binary images.
- Morphology is essential in segmentation, feature extraction, and description of images.

Exercises

8.1 Find the dilation of $x(m, n)$ and $h(m, n)$.

$$h(m,n) = \begin{bmatrix} 1 & 0 & 1 \\ 0 & \mathbf{1} & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

(i)

$$x(m,n) = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

(ii)

$$x(m,n) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

(iii)

$$x(m,n) = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

8.2 Find the erosion of $x(m, n)$ and $h(m, n)$.

$$h(m,n) = \begin{bmatrix} 0 & 1 & 0 \\ 1 & \mathbf{0} & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

(i)

$$x(m,n) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

(ii)

$$x(m,n) = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

(iii)

$$x(m,n) = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

8.3 Find the opening of $x(m,n)$ and $h(m,n)$.

$$h(m,n) = \begin{bmatrix} 0 & 1 & 0 \\ 1 & \mathbf{1} & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

(i)

$$x(m,n) = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

(ii)

$$x(m,n) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

(iii)

$$x(m,n) = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{bmatrix}$$

8.4 Find the closing of $x(m, n)$ and $h(m, n)$.

$$h(m,n) = \begin{bmatrix} 1 & 0 & 1 \\ 0 & \mathbf{1} & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

Verify the output using the equivalent expression in terms of the opening operation.
(i)

$$x(m,n) = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(ii)

$$x(m,n) = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

(iii)

$$x(m,n) = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

8.5 Find the hit-and-miss transformation of $x(m, n)$ and $h_h(m, n)$ and $h_{ms}(m, n)$.

$$h_h(m,n) = \begin{bmatrix} 1 & 0 & 1 \\ 0 & \mathbf{0} & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad h_{ms}(m,n) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & \mathbf{0} & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

***** (i)

$$x(m,n) = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

(ii)

$$x(m,n) = \begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}$$

(iii)

$$x(m,n) = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

8.6 Find the thinned version of $x(m,n)$.

(i)

$$x(m,n) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

* (ii)

$$x(m,n) = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}$$

(iii)

$$x(m,n) = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

8.7 Find the skeleton of $x(m,n)$.

(i)

$$x(m,n) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

(ii)

$$x(m,n) = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

* (iii)

$$x(m,n) = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}$$

8.8 Extract the boundary of $x(m, n)$.

(i)

$$x(m,n) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

* (ii)

$$x(m,n) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \end{bmatrix}$$

(iii)

$$x(m,n) = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix}$$

Chapter 9
Edge Detection

Abstract Edge detection is an important step in segmentation of the image and leads to object recognition. An edge is a line of interaction of two surfaces. Edges are detected using operators based on the first and second derivatives of the image. A high value or a zero-crossing of the response of the operators indicates an edge pixel. Derivatives are approximated by differences in digital images. Commonly used edge operators are presented with examples.

Almost all signals (e.g., alternating voltage waveform in a transmission line) are constantly changing. The instantaneous rate of change of a signal is its derivative. Figure 9.1 shows the sine waveform $x(t) = \sin(2\pi t/8)$, its first derivative $x'(t) = (2\pi/8)\cos(2\pi t/8)$, and its second derivative $x''(t) = -(2\pi/8)^2 \sin(2\pi t/8)$. The derivative is an important characterization of a signal (image). The rate of change of the sine waveform is maximum at $t = 0$, and its first derivative, the cosine waveform, reaches its positive peak value. The second derivative has a zero-crossing at that point. Another point to be noted is that the amplitude of the derivatives gets multiplied by the frequency values, which may amplify noise levels, as noise constitutes significant part of the high-frequency components. These are the key points in edge detection, which, in digital images, is approximating the derivatives, in two directions, by differences.

9.1 Edge Detection

An edge is a line of interaction of two surfaces. Edge pixels are characterized by the abrupt change of intensity with the neighboring pixels. The boundaries of objects in an image are identified by edges. Edges are useful for tasks such as segmentation, registration, and object identification. Edges provide a compact representation of objects than pixels. Edges are amplitude discontinuities between regions of an image. In the frequency domain, an edge is characterized by the high-frequency components of the spectrum of the image. Basically, edge detection constitutes highpass filtering of an image. In the Haar DWT, the high-frequency subband of the signal is extracted

D. Sundararajan, *Digital Image Processing*, DOI 10.1007/978-981-10-6113-4_9

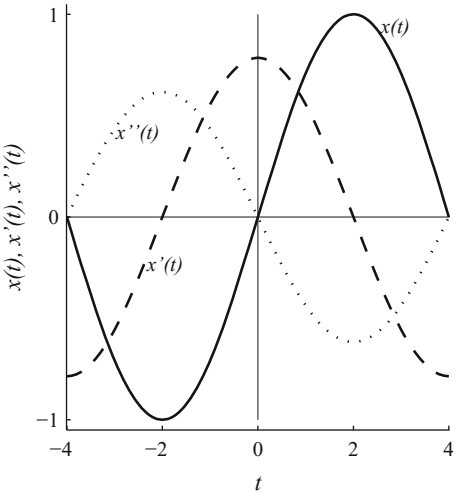


Fig. 9.1 Sine waveform $x(t) = \sin(2\pi t/8)$ and its first and second derivatives, $x'(t) = (2\pi/8)\cos(2\pi t/8)$, $x''(t) = -(2\pi/8)^2 \sin(2\pi t/8)$

by filtering the signal with a highpass filter. The impulse response of the filter is $\{-1, 1\}$. This filter finds the moving difference of a signal. This is the starting point of the theory of edge detectors.

It is assumed that the top-left corner of the image is the origin, as presented in Sect. 1.2. To find edges in a digital image, we use edge detectors. The gradient (a graded change in the magnitude) along the m-direction of an image $x(m, n)$ is

$$gm(m, n) = x(m + 1, n) - x(m, n)$$

The gradient along the n-direction is

$$gn(m, n) = x(m, n + 1) - x(m, n)$$

With these definitions, the gradients $gm(m, n)$ and $gn(m, n)$ are positive for an edge whose amplitude increases from top to bottom and left to right of the image, respectively. The directions of an edge and its gradient are perpendicular to each other. The operators are filters with masks

$$\left\{ \begin{array}{c} -1 \\ 1 \end{array} \right\} \text{ and } \{-1, 1\}$$

respectively. The frequency response is

$$H(e^{j\omega}) = -(1 - e^{-j\omega}) = -je^{-j\omega/2}\frac{1}{j}(e^{j\frac{\omega}{2}} - e^{-j\frac{\omega}{2}}) = -je^{-j\omega/2}2\sin\left(\frac{\omega}{2}\right)$$

which is a highpass filter and suppresses low-frequency components. The magnitude of the frequency response is the first quarter of a sine wave. The frequency response of $\{-1, 0, 1\}$, which is symmetrical, is

$$H(e^{j\omega}) = -(e^{j\omega} - e^{-j\omega}) = -j2\sin(\omega)$$

This is a bandpass filter. The magnitude of the frequency response is the first half of a sine wave.

Convolution of the input image with the impulse responses of filters yields the gradient image.

The 3×3 neighborhood is defined as

$$\begin{bmatrix} x(m-1,n-1) & x(m-1,n) & x(m-1,n+1) \\ x(m,n-1) & x(m,n) & x(m,n+1) \\ x(m+1,n-1) & x(m+1,n) & x(m+1,n+1) \end{bmatrix}$$

Table 9.1 shows the gradient operator masks of commonly used edge filters. Moving difference measures the change in intensity on only one side of the pixel. To make the measurement more symmetrical, we can find the gradient between the neighbors of the pixel and then take the average. This averaging is usually not shown or taken, since it affects uniformly across the image. Therefore, the moving difference filters become

$$gm(m,n) = x(m+1,n) - x(m-1,n)$$

and

$$gn(m,n) = x(m,n+1) - x(m,n-1)$$

The Prewitt operator averages intensity changes over six intervals. The Sobel operator gives twice the weight to the central pixels. The values of these masks are to be multiplied by the corresponding pixel values of the image pointwise and divided by the sum of the magnitudes of the elements of the mask. These operators compute the differences (highpass filtering in one direction) of local sums (lowpass filtering in another direction), which has the effect of reducing the noise.

Table 9.1 Typical gradient operator masks. The element at the origin is shown in boldface

Operator	Gradient, n-direction	Gradient, m-direction
Prewitt	$\begin{bmatrix} -1 & 0 & 1 \\ -1 & \mathbf{0} & 1 \\ -1 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} -1 & -1 & -1 \\ 0 & \mathbf{0} & 0 \\ 1 & 1 & 1 \end{bmatrix}$
Sobel	$\begin{bmatrix} -1 & 0 & 1 \\ -2 & \mathbf{0} & 2 \\ -1 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} -1 & -2 & -1 \\ 0 & \mathbf{0} & 0 \\ 1 & 2 & 1 \end{bmatrix}$

$$\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -1 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -1 \end{bmatrix}$$

This characteristic is also clear from the frequency response of the Sobel filters. Using the row and column filters, computational savings can be achieved. Figure 9.2a, b show, respectively, the magnitude of the frequency response of the two Sobel gradient filters. Figure 9.3 shows the image of the frequency response of the filter in (a). In these figures, the frequencies are normalized in the range -1 to 1, where 1 indicates π radians (or half the sampling frequency). The frequency response is symmetrical about the zero frequency. Filter in (a) has a lowpass frequency response along the l-direction. It has the peak value of 8 (the sum of the magnitudes of the

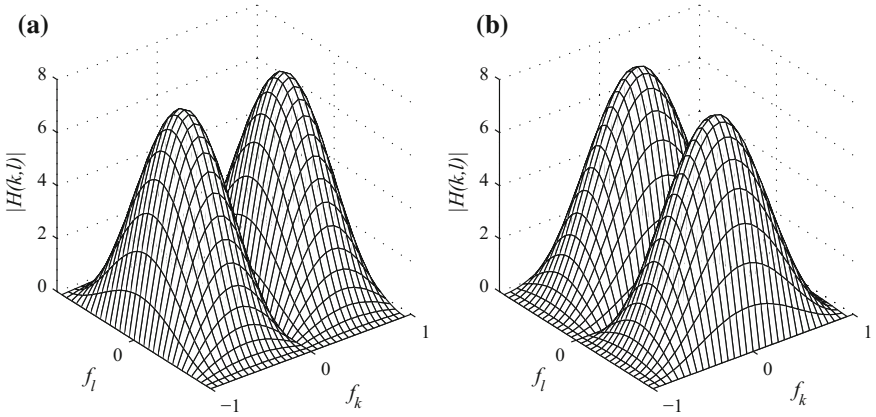


Fig. 9.2 Magnitude of the frequency responses of Sobel gradient filters

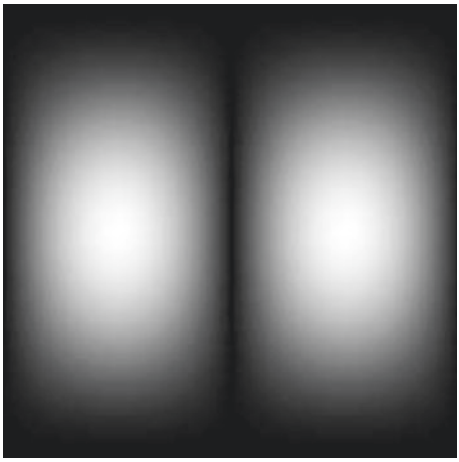


Fig. 9.3 Image of the magnitude of the frequency response of Sobel gradient filter

filter coefficients) at frequency 0 and tapers toward zero near frequency 1. It is the first half cycle of a cosine function with a DC offset. The filter has a bandpass response along the k-direction. The response is zero at frequency 0 and goes up toward high frequencies, and then, it goes down. It is the first half cycle of a sine function. Frequency response in (b) is the complement of that in (a).

For uniform regions of the image, the gradient is zero, as it should be. For the Sobel operator,

$$\begin{aligned} gm(m,n) = &(x(m+1,n-1) + 2x(m+1,n) + x(m+1,n+1)) \\ &- (x(m-1,n-1) + 2x(m-1,n) + x(m-1,n+1)) \end{aligned}$$

and

$$\begin{aligned} gn(m,n) = &(x(m-1,n+1) + 2x(m,n+1) + x(m+1,n+1)) \\ &- (x(m-1,n-1) + 2x(m,n-1) + x(m+1,n-1)) \end{aligned}$$

The magnitude of the gradient is

$$g(m,n) = \sqrt{(gm^2(m,n) + gn^2(m,n))}$$

and its direction, with respect to the row axis, is given by

$$\theta(m,n) = \tan^{-1}\left(\frac{gn(m,n)}{gm(m,n)}\right)$$

For computational simplicity, the magnitude of the gradient is often approximated as

$$g(m,n) = |(gm(m,n)| + |gn(m,n)|$$

Figure 9.4 shows the block diagram of edge detection using gradient operators. Edge gradients are computed using edge detectors in two orthogonal directions and pixels with gradient magnitude greater than a threshold are labeled as an edge pixel. Edge

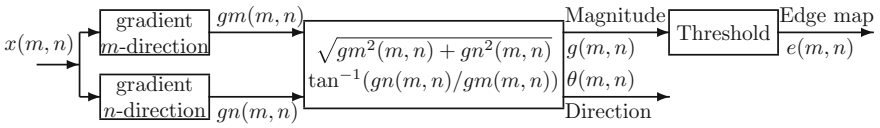


Fig. 9.4 Block diagram of edge detection using gradient operators

detection is similar to approximating a signal by Fourier analysis. The signal is correlated with cosine and sine basis signals, which are orthogonal, and the respective coefficients are computed. The magnitude and phase computed, using these coefficients, are the magnitude and phase of the corresponding frequency component. Components with very small magnitudes are ignored. Basically, both the processes involve correlation operation.

Consider the six sample neighborhoods of the image $x(m, n)$.

$$\begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

Using the Sobel operator, the gradient values and the angles, respectively, are

neighborhood	1	2	3	4	5	6
gm	4	0	3	4	-2	-3
gn	0	4	-3	-2	-4	-3
θ	0°	90°	$-45°$	$-26.5651°$	63.4349°	45°

In case 1, the gradient is in the vertical direction, which is parallel with the m-axis. Therefore, the angle is zero. The edge strength is also increasing along the m-axis. Therefore, the strength is positive. The edge strength is, obviously, zero along the n-axis. The other cases can be interpreted similarly.

Example 9.1 Let the 8×8 input image be

$$x(m, n) = \begin{bmatrix} 88 & 100 & 104 & 101 & 114 & 110 & 110 & 107 \\ 92 & 102 & 104 & 107 & 112 & 110 & 104 & 92 \\ 103 & 105 & 111 & 112 & 114 & 108 & 86 & 39 \\ 106 & 107 & 112 & 113 & 107 & 73 & 26 & 25 \\ 111 & 114 & 14 & 104 & 64 & 23 & 25 & 28 \\ 117 & 115 & 97 & 45 & 15 & 13 & 23 & 29 \\ 119 & 93 & 32 & 0 & 15 & 11 & 19 & 23 \\ 88 & 15 & 0 & 2 & 10 & 13 & 11 & 15 \end{bmatrix}$$

Find the edge output with the threshold $T = 40$ and $T = 30$ using the Sobel gradient operators.

Solution

The gradient values along the n-direction are

$$gn(m, n) = \begin{bmatrix} 6.0000 & 2.2500 & 3.6250 & 1.3750 & -6.0000 & -13.5000 \\ 4.2500 & 3.1250 & 1.1250 & -5.6250 & -18.1250 & -25.5000 \\ -9.6250 & 1.1250 & 5.3750 & -20.6250 & -28.6250 & -20.0000 \\ -26.0000 & -10.5000 & 1.6250 & -29.2500 & -18.8750 & -2.7500 \\ -28.0000 & -30.3750 & -16.3750 & -16.7500 & -2.3750 & 6.1250 \\ -35.2500 & -33.6250 & -13.2500 & 0.1250 & 2.1250 & 5.2500 \end{bmatrix}$$

The first value is obtained as

$$((104 + 2(104) + 111) - (88 + 2(92) + 103))/8 = 6$$

The divisor 8 is the sum of the magnitudes of the elements of the mask. The gradient values along the m-direction are

$$gm(m,n) = \begin{bmatrix} 4.0000 & 3.7500 & 3.6250 & 1.1250 & -3.5000 & -14.7500 \\ 4.0000 & 3.3750 & 1.8750 & -5.1250 & -19.6250 & -32.5000 \\ -8.8750 & -24.1250 & -20.3750 & -24.1250 & -35.1250 & -27.2500 \\ 1.5000 & -11.2500 & -30.3750 & -39.0000 & -26.8750 & -7.7500 \\ -2.0000 & -11.1250 & -29.8750 & -26.7500 & -9.8750 & -3.6250 \\ -40.7500 & -42.1250 & -23.5000 & -6.6250 & -2.1250 & -4.7500 \end{bmatrix}$$

The last value is obtained as

$$((13 + 2(11) + 15) - (13 + 2(23) + 29))/8 = -4.75$$

The magnitude values of the gradient, using the square-root form, are

$$g(m,n) = \begin{bmatrix} 7.2111 & 4.3732 & 5.1265 & 1.7766 & 6.9462 & 19.9953 \\ 5.8363 & 4.5996 & 2.1866 & 7.6096 & 26.7143 & 41.3098 \\ 13.0922 & 24.1512 & 21.0720 & 31.7397 & 45.3118 & 33.8018 \\ 26.0432 & 15.3887 & 30.4184 & 48.7500 & 32.8410 & 8.2234 \\ 28.0713 & 32.3482 & 34.0684 & 31.5614 & 10.1566 & 7.1173 \\ 53.8807 & 53.8995 & 26.9780 & 6.6262 & 3.0052 & 7.0799 \end{bmatrix}$$

The first value is computed as

$$\sqrt{4^2 + 6^2} = 7.2111$$

The rounded angles of the gradient, in degrees, are

$$\theta(m,n) = \begin{bmatrix} 56 & 31 & 45 & 51 & -120 & -138 \\ 47 & 43 & 31 & -132 & -137 & -142 \\ -133 & 177 & 165 & -139 & -141 & -144 \\ -87 & -137 & 177 & -143 & -145 & -160 \\ -94 & -110 & -151 & -148 & -166 & 121 \\ -139 & -141 & -151 & 179 & 135 & 132 \end{bmatrix}$$

The first value is computed as

$$\tan^{-1}(6/4) = 56.3099^\circ$$

With the threshold $T = 40$ and $T = 30$, the edge pixels are

$$e_{40}(m,n) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \quad e_{30}(m,n) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Usually, edges are thinned to make them 1-pixel wide. If the size of the edge output has to be same as that of the input image, then border pixels have to be defined. For example, the required pixels at the borders can be assigned the values of their neighbors. This method of border extension is called replication.

9.1.1 Edge Detection by Compass Gradient Operators

The operators, thus far, used gradients in two orthogonal directions to find the edge map of an image. For selected directions, typically at intervals of 45°, image gradients can be estimated using rotated versions of a mask. The maximum of the magnitude of the outputs of the different masks is the final gradient image.

Table 9.2 shows the typical masks of commonly used edge filters. Each mask of a specific operator is related by a 45° rotation to its neighbor. Only four of the masks are linearly independent. Convolution of the input image with the masks yields the gradient image. The direction of the gradient is the direction of the mask that produced the maximum value. With larger masks, a finer angular resolution can be obtained.

For the same six sample neighborhoods used earlier, the responses of the four compass Sobel operators are

	1	2	3	4	5	6
gS	4	0	3	4	−2	−3
gE	0	4	−3	−2	−4	−3
gSE	3	3	0	2	−4	−4
gSW	3	−3	4	4	2	0

Table 9.2 Typical masks of four compass gradient operators. The element at the origin is shown in boldface. The corresponding scale factor appears at the top. The direction of the gradient is indicated at the beginning of each row

	1/5	1/15	1/3	1/4
W	$\begin{bmatrix} 1 & 1 & -1 \\ 1 & \mathbf{-2} & -1 \\ 1 & 1 & -1 \end{bmatrix}$	$\begin{bmatrix} 5 & -3 & -3 \\ 5 & \mathbf{0} & -3 \\ 5 & -3 & -3 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & -1 \\ 1 & \mathbf{0} & -1 \\ 1 & 0 & -1 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & -1 \\ 2 & \mathbf{0} & -2 \\ 1 & 0 & -1 \end{bmatrix}$
SW	$\begin{bmatrix} 1 & -1 & -1 \\ 1 & \mathbf{-2} & -1 \\ 1 & 1 & 1 \end{bmatrix}$	$\begin{bmatrix} -3 & -3 & -3 \\ 5 & \mathbf{0} & -3 \\ 5 & 5 & -3 \end{bmatrix}$	$\begin{bmatrix} 0 & -1 & -1 \\ 1 & \mathbf{0} & -1 \\ 1 & 1 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & -1 & -2 \\ 1 & \mathbf{0} & -1 \\ 2 & 1 & 0 \end{bmatrix}$
S	$\begin{bmatrix} -1 & -1 & -1 \\ 1 & \mathbf{-2} & 1 \\ 1 & 1 & 1 \end{bmatrix}$	$\begin{bmatrix} -3 & -3 & -3 \\ -3 & \mathbf{0} & -3 \\ 5 & 5 & 5 \end{bmatrix}$	$\begin{bmatrix} -1 & -1 & -1 \\ 0 & \mathbf{0} & 0 \\ 1 & 1 & 1 \end{bmatrix}$	$\begin{bmatrix} -1 & -2 & -1 \\ 0 & \mathbf{0} & 0 \\ 1 & 2 & 1 \end{bmatrix}$
SE	$\begin{bmatrix} -1 & -1 & 1 \\ -1 & \mathbf{-2} & 1 \\ 1 & 1 & 1 \end{bmatrix}$	$\begin{bmatrix} -3 & -3 & -3 \\ -3 & \mathbf{0} & 5 \\ -3 & 5 & 5 \end{bmatrix}$	$\begin{bmatrix} -1 & -1 & 0 \\ -1 & \mathbf{0} & 1 \\ 0 & 1 & 1 \end{bmatrix}$	$\begin{bmatrix} -2 & -1 & 0 \\ -1 & \mathbf{0} & 1 \\ 0 & 1 & 2 \end{bmatrix}$
E	$\begin{bmatrix} -1 & 1 & 1 \\ -1 & \mathbf{-2} & 1 \\ -1 & 1 & 1 \end{bmatrix}$	$\begin{bmatrix} -3 & -3 & 5 \\ -3 & \mathbf{0} & 5 \\ -3 & -3 & 5 \end{bmatrix}$	$\begin{bmatrix} -1 & 0 & 1 \\ -1 & \mathbf{0} & 1 \\ -1 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} -1 & 0 & 1 \\ -2 & \mathbf{0} & 2 \\ -1 & 0 & 1 \end{bmatrix}$
NE	$\begin{bmatrix} 1 & 1 & 1 \\ -1 & \mathbf{-2} & 1 \\ -1 & -1 & 1 \end{bmatrix}$	$\begin{bmatrix} -3 & 5 & 5 \\ -3 & \mathbf{0} & 5 \\ -3 & -3 & -3 \end{bmatrix}$	$\begin{bmatrix} 0 & 1 & 1 \\ -1 & \mathbf{0} & 1 \\ -1 & -1 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 1 & 2 \\ -1 & \mathbf{0} & 1 \\ -2 & -1 & 0 \end{bmatrix}$
N	$\begin{bmatrix} 1 & 1 & 1 \\ 1 & \mathbf{-2} & 1 \\ -1 & -1 & -1 \end{bmatrix}$	$\begin{bmatrix} 5 & 5 & 5 \\ -3 & \mathbf{0} & -3 \\ -3 & -3 & -3 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 & 1 \\ 0 & \mathbf{0} & 0 \\ -1 & -1 & -1 \end{bmatrix}$	$\begin{bmatrix} 1 & 2 & 1 \\ 0 & \mathbf{0} & 0 \\ -1 & -2 & -1 \end{bmatrix}$
NW	$\begin{bmatrix} 1 & 1 & 1 \\ 1 & \mathbf{-2} & -1 \\ 1 & -1 & -1 \end{bmatrix}$	$\begin{bmatrix} 5 & 5 & -3 \\ 5 & \mathbf{0} & -3 \\ -3 & -3 & -3 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 & 0 \\ 1 & \mathbf{0} & -1 \\ 0 & -1 & -1 \end{bmatrix}$	$\begin{bmatrix} 2 & 1 & 0 \\ 1 & \mathbf{0} & -1 \\ 0 & -1 & -2 \end{bmatrix}$

When there is a clear cut maximum, the angle of the gradient is the angle of the corresponding mask. This is the case in columns 1, 2, 3, and 6. In the case of two maximum values, the exact angle can be computed using the gradients in the S and E directions, and the gradient value is assigned to the nearest mask. Cases 4 and 5 can be assigned to the masks at -45 and $45°$, respectively.

9.2 Canny Edge Detection Algorithm

Canny edge detection algorithm is based on three objectives.

1. The edges found should be true edges, and all the edges should be found. The probability of finding a good edge should be maximized and that of a false edge should be minimized. Achieving this objective requires a high signal-to-noise ratio.
2. The location of the edge found must be as close as possible to the exact location.
3. There should be no multiple responses for a single edge.

Finding the edges in an image using the Canny edge detection algorithm consists of the following four basic steps.

1. The input image is smoothed by a Gaussian filter to improve the SNR.
2. The gradient magnitude and angle images are formed using a gradient filter.
3. The edge image is thinned by using nonmaximum suppression of the gradient image.
4. By using two thresholds and connectivity constraint, the final edge image is formed.

Example 9.2 Let the 8×8 input image $x(m, n)$ be the same as that given in Example 9.1. Assuming replication of pixels at the borders, find the 8×8 edge map using the Canny edge detection algorithm. Let the standard deviation of the Gaussian filter be $\sigma = 1$. The Gaussian gradient filter is to be used for finding the gradients.

Solution
Step 1 Find the Gaussian smoothing and gradient filters.
The 2-D Gaussian filter is separable. The filter length L is given by the formula

$$L = 8\lceil\sigma\rceil = 8$$

The *ceiling* function rounds a number to the nearest integer greater than or equal to it. The impulse response of the 1-D Gaussian lowpass filter is given by

$$h(n) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{n^2}{2\sigma^2}}$$

where n varies from

$$-\frac{L-1}{2}, \ldots \frac{L-1}{2} = \{-3.5, -2.5, -1.5, -0.5, 0.5, 1.5, 2.5, 3.5\}$$

The impulse response of the 1-D Gaussian lowpass filter, for this example, is

$$h(n) = \{0.0009, 0.0175, 0.1295, 0.3521, 0.3521, 0.1295, 0.0175, 0.0009\}$$

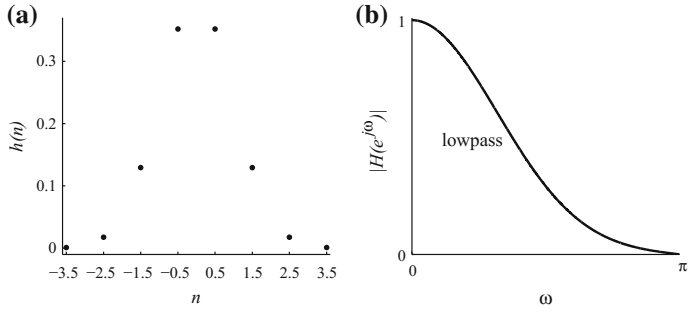


Fig. 9.5 **a** Impulse response of the 1-D Gaussian lowpass filter with $L = 8$ and $\sigma = 1$; **b** magnitude of its frequency response

Check normalization by ensuring that the sum of the values is 1. The values are even-symmetric. Figure 9.5a, b show, respectively, the impulse response and the magnitude of the frequency response of this lowpass filter.

The impulse response of the Gaussian gradient filter, for this example, is

$$hd(n) = \{0.0167, 0.0643, 0.1673, 0.1113, -0.1113, -0.1673, -0.0643, -0.0167\}$$

This sequence is obtained by computing differences of the values of $h(n)$. At the ends,

$$hd(-3.5) = (0.0175 - 0.0009) = 0.0167, hd(3.5) = (0.0009 - 0.0175) = -0.0167$$

The second, third, and fourth values are computed as

$$hd(2.5) = (0.1295 - 0.0009)/2 = 0.0643, hd(1.5) = (0.3521 - 0.0175)/2 = 0.1673,$$

$$hd(0.5) = (0.3521 - 0.1295)/2 = 0.1113$$

The values are odd-symmetric. The coefficients are normalized by dividing the values by the sum of the first half of the values, 0.3595. The normalized values are

$$hd(n) = \{0.0463, 0.1789, 0.4653, 0.3095, -0.3095, -0.4653, -0.1789, -0.0463\}$$

The sum of all the values is zero. Figure 9.6a, b show, respectively, the impulse response and the magnitude of the frequency response of this gradient filter.

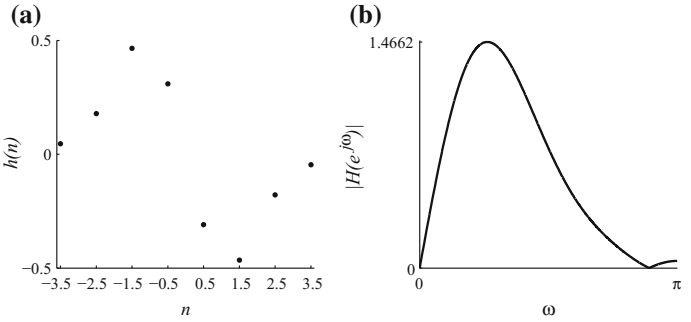


Fig. 9.6 **a** Impulse response of the 1-D Gaussian gradient filter; **b** magnitude of its frequency response

Step 2 The 8×8 input image is

$$x(m,n) = \begin{bmatrix} 88 & 100 & 104 & 101 & 114 & 110 & 110 & 107 \\ 92 & 102 & 104 & 107 & 112 & 110 & 104 & 92 \\ 103 & 105 & 111 & 112 & 114 & 108 & 86 & 39 \\ 106 & 107 & 112 & 113 & 107 & 73 & 26 & 25 \\ 111 & 114 & 14 & 104 & 64 & 23 & 25 & 28 \\ 117 & 115 & 97 & 45 & 15 & 13 & 23 & 29 \\ 119 & 93 & 32 & 0 & 15 & 11 & 19 & 23 \\ 88 & 15 & 0 & 2 & 10 & 13 & 11 & 15 \end{bmatrix}$$

Image $x(m,n)$ is convolved with the transpose of $h(n)$ to get the smoothed version, $xcl(m,n)$, of the image.

$$\begin{bmatrix} 91.6867 & 101.4867 & 104.9683 & 104.7502 & 113.1295 & 109.0165 & 103.2324 & 91.4051 \\ 97.4493 & 103.6297 & 105.9170 & 108.5433 & 111.4264 & 102.8939 & 86.9920 & 65.7040 \\ 103.6510 & 106.5541 & 97.4387 & 109.1937 & 102.9786 & 83.2139 & 58.5848 & 40.5729 \\ 108.7476 & 109.8222 & 73.7769 & 98.7010 & 79.2478 & 51.6998 & 34.3354 & 29.5898 \\ 112.9210 & 108.7355 & 59.7681 & 69.1888 & 45.8958 & 25.7829 & 24.5288 & 27.3254 \\ 112.4322 & 92.1835 & 49.2913 & 31.6881 & 22.3060 & 14.7257 & 20.1832 & 24.6257 \\ 103.0894 & 57.2298 & 24.1732 & 8.7501 & 13.4392 & 12.5235 & 15.6294 & 19.8665 \\ 92.5436 & 26.9420 & 5.8572 & 2.5837 & 10.7824 & 12.7497 & 12.2587 & 16.2929 \end{bmatrix}$$

For example, the convolution of the values in the fourth column of $x(m,n)$

$$\{101, 107, 112, 113, 104, 45, 0, 2\}$$

with $h(n)$ is $xcl(3,3) = 98.7010$. A quick and approximate check of this value is

$$(112 + 45)0.1 + (113 + 104)0.4 = 102.5$$

Step 3 The smoothed image $xcl(m, n)$ is convolved with $hd(n)$ to get the smoothed gradient $gh(m, n)$ along horizontal direction.

$$\begin{bmatrix} 12.5431 & 11.7946 & 8.9847 & 4.7762 & -4.8728 & -14.9135 & -16.3594 & -9.6601 \\ 8.4848 & 8.6223 & 4.9299 & -4.9615 & -21.6193 & -35.7217 & -34.0564 & -18.6762 \\ -1.0313 & -1.3096 & -3.7695 & -20.0463 & -42.8940 & -51.5689 & -39.7579 & -18.9002 \\ -19.1018 & -23.7512 & -20.1645 & -33.4650 & -50.0952 & -42.8899 & -23.8417 & -8.4642 \\ -36.9544 & -51.5308 & -46.0057 & -42.0521 & -36.5792 & -18.0210 & -3.6787 & 0.7167 \\ -54.2651 & -71.4931 & -59.7136 & -35.9351 & -15.2416 & 0.3622 & 6.0688 & 3.9456 \\ -71.9416 & -74.3582 & -45.4024 & -15.2671 & 0.4157 & 5.7410 & 6.3927 & 3.5829 \\ -80.5180 & -66.7048 & -26.5270 & -0.4152 & 6.4840 & 5.3481 & 4.5181 & 2.7661 \end{bmatrix}$$

For example, the convolution of the fourth row of $xcl(m, n)$

$$\{108.7476, 109.8222, 73.7769, 98.7010, 79.2478, 51.6998, 34.3354, 29.5898\}$$

with $hd(n)$ is $gh(3, 3) = -33.4650$. A quick and approximate check of this value is

$$0.2(34 - 110) + 0.5(51 - 74) + 0.3(79 - 99) = -32.7$$

Step 4 Image $x(m, n)$ is convolved with $h(n)$ to get the smoothed version, $xrl(m, n)$, of the image.

$$\begin{bmatrix} 94.5478 & 100.0169 & 103.5369 & 107.2654 & 110.0735 & 109.9113 & 108.5623 & 107.4473 \\ 97.3554 & 102.0548 & 105.7177 & 108.5940 & 109.0951 & 105.4261 & 98.9200 & 93.8872 \\ 104.9077 & 107.8836 & 110.7419 & 111.4065 & 106.5613 & 90.8975 & 65.8630 & 46.3625 \\ 107.2528 & 109.3599 & 110.1878 & 103.8641 & 83.8940 & 54.4910 & 33.0833 & 26.0425 \\ 99.3289 & 76.0975 & 67.0675 & 66.4990 & 48.1991 & 31.1661 & 26.9935 & 27.5552 \\ 112.3543 & 98.0500 & 69.2336 & 37.9184 & 20.9999 & 19.7808 & 24.5838 & 27.9302 \\ 96.4009 & 61.8850 & 27.6540 & 12.9377 & 12.6802 & 15.9353 & 19.8771 & 22.2646 \\ 49.3249 & 18.7441 & 5.7990 & 6.4543 & 10.0707 & 11.9990 & 13.2337 & 14.4425 \end{bmatrix}$$

For example, the convolution of the values in the fourth row of $x(m, n)$

$$\{106, 107, 112, 113, 107, 73, 26, 25\}$$

with $h(n)$ is $xl(3, 3) = 103.8641$.

Step 5 The smoothed image xrl is convolved with the transpose of $hd(n)$ to get the smoothed gradient $gv(m, n)$ along the vertical direction

$$\begin{bmatrix} 8.1835 & 4.8543 & 3.5275 & -0.1592 & -9.4872 & -23.7979 & -40.1333 & -50.8824 \\ 9.9288 & 1.7804 & -3.4646 & -11.2182 & -28.1610 & -48.5452 & -63.8324 & -70.5606 \\ 4.9156 & -13.7383 & -27.8064 & -38.6963 & -55.7961 & -66.2969 & -62.7429 & -55.3233 \\ -1.2537 & -25.8221 & -51.1524 & -67.5397 & -72.7382 & -60.8535 & -39.6482 & -25.2303 \\ -13.1870 & -35.1012 & -61.1333 & -74.6588 & -63.4017 & -39.9062 & -20.2757 & -11.0329 \\ -41.1412 & -58.2191 & -64.9127 & -57.9576 & -37.9925 & -21.3653 & -13.8481 & -11.4084 \\ -55.5250 & -64.7090 & -52.0752 & -31.9009 & -16.1342 & -10.2366 & -10.7183 & -11.5796 \\ -35.4956 & -36.9173 & -24.3557 & -11.4274 & -4.9359 & -4.1116 & -5.7590 & -6.6598 \end{bmatrix}$$

For example, the convolution of the fourth column of $xrl(m, n)$

$$\{[107.2654, 108.5940, 111.4065, 103.8641, 66.4990, 37.9184, 12.9377, 6.4543]\}$$

with the transpose of $hd(n)$ is $gv(3, 3) = -67.5397$.

Step 6 The magnitude gradient, $g(m, n) = \sqrt{gh^2(m, n) + gv^2(m, n)}$ is

$$g(m,n) = \begin{bmatrix} 14.9766 & 12.7545 & 9.6524 & 4.7788 & 10.6654 & 28.0847 & 43.3395 & 51.7912 \\ 13.0604 & 8.8042 & 6.0256 & 12.2664 & 35.5026 & 60.2717 & 72.3493 & 72.9904 \\ 5.0227 & 13.8006 & 28.0607 & 43.5804 & 70.3783 & 83.9918 & 74.2789 & 58.4627 \\ 19.1429 & 35.0842 & 54.9834 & 75.3759 & 88.3198 & 74.4493 & 46.2645 & 26.6122 \\ 39.2367 & 62.3499 & 76.5101 & 85.6873 & 73.1971 & 43.7865 & 20.6067 & 11.0562 \\ 68.0977 & 92.1994 & 88.2008 & 68.1940 & 40.9357 & 21.3684 & 15.1195 & 12.0714 \\ 90.8769 & 98.5718 & 69.0884 & 35.3659 & 16.1396 & 11.7366 & 12.4799 & 12.1212 \\ 87.9948 & 76.2392 & 36.0122 & 11.4350 & 8.1489 & 6.7459 & 7.3198 & 7.2115 \end{bmatrix}$$

For example, the first value is $\sqrt{8.1835^2 + 12.5431^2} = 14.9766$.

Step 7 The normalized magnitude gradient is obtained by dividing $g(m, n)$ by its maximum value, 98.5718.

$$g_n(m,n) = \begin{bmatrix} 0.1519 & 0.1294 & 0.0979 & 0.0485 & 0.1082 & 0.2849 & 0.4397 & 0.5254 \\ 0.1325 & 0.0893 & 0.0611 & 0.1244 & 0.3602 & 0.6114 & 0.7340 & 0.7405 \\ 0.0510 & 0.1400 & 0.2847 & 0.4421 & 0.7140 & 0.8521 & 0.7536 & 0.5931 \\ 0.1942 & 0.3559 & 0.5578 & 0.7647 & 0.8960 & 0.7553 & 0.4693 & 0.2700 \\ 0.3981 & 0.6325 & 0.7762 & 0.8693 & 0.7426 & 0.4442 & 0.2091 & 0.1122 \\ 0.6908 & 0.9354 & 0.8948 & 0.6918 & 0.4153 & 0.2168 & 0.1534 & 0.1225 \\ 0.9219 & 1.0000 & 0.7009 & 0.3588 & 0.1637 & 0.1191 & 0.1266 & 0.1230 \\ 0.8927 & 0.7734 & 0.3653 & 0.1160 & 0.0827 & 0.0684 & 0.0743 & 0.0732 \end{bmatrix}$$

Step 8 The high and low thresholds, respectively, are

$$\{0.7031, 0.2813\}$$

The high threshold is about 0.7 of the maximum value of $g_n(m, n)$, and the low threshold is 0.4 that of high. The thresholds can be set, by trial and error, for a particular image.

Step 9 The direction of the gradient is obtained by taking the inverse tangent of the ratio of the vertical and horizontal gradients, $gv(m, n)/gh(m, n)$. The rounded value of the angles, in degrees, are

$$\theta(m,n) = \begin{bmatrix} 33 & 22 & 21 & -1 & 62 & 57 & 67 & 79 \\ 49 & 11 & -35 & 66 & 52 & 53 & 61 & 75 \\ -78 & 84 & 82 & 62 & 52 & 52 & 57 & 71 \\ 3 & 47 & 68 & 63 & 55 & 54 & 58 & 71 \\ 19 & 34 & 53 & 60 & 60 & 65 & 79 & -86 \\ 37 & 39 & 47 & 58 & 68 & -89 & -66 & -70 \\ 37 & 41 & 48 & 64 & -88 & -60 & -59 & -72 \\ 23 & 28 & 42 & 87 & -37 & -37 & -51 & -67 \end{bmatrix}$$

For example, the first entry is obtained as

$$180(\tan^{-1}(\frac{8.1835}{12.5431}))/\pi = 33.1217 \approx 33$$

Step 10 The next step is nonmaximum suppression of the gradient values. That involves selecting the pixels with peak values at the top of the ridges in the edge map. That is to select only one pixel for an edge. Edges get thinned in this process. Thresholding of the normalized gradient image $g_n(m, n)$ with the value 0.7031 yields the edge map $e'(m, n)$. The final edge map $e(m, n)$ is derived from $e'(m, n)$. The pixels on the border rows and columns are not considered.

$$e'(m,n) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad e(m,n) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Three edge pixels are dropped in nonmaximum suppression. This process for the pixel $g_n(2, 4) = 0.7140$ is shown in Fig. 9.7. There are three edge pixels in that column with about the same orientation. Only one of them is adequate. The gradient direction, from the θ matrix, is 52° as shown by dashed line. The normal line, which is perpendicular to the gradient line, is also shown. For the pixel $g_n(2, 4) = 0.7140$ to be retained as an edge pixel, its value must be greater than those of its neighbors along

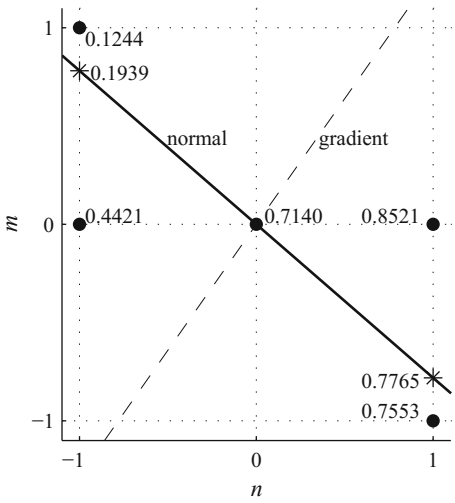


Fig. 9.7 Nonmaximum suppression of the edge pixel $g_n(2, 4) = 0.7140$

the normal line. For arbitrary angles of the gradient, the normal line intersects the coordinate axes between two pixels, as shown by asterisks in the figure. The values at the intersection are found using interpolation of the values of the neighboring pixels. For this pixel, the vertical distance of the point of intersection is given by

$$\tan(52 + 90) = 0.7813$$

and the point is marked by the top asterisk. The distance, -0.7813, is marked by the bottom asterisk. The values of the two neighboring pixels on the normal line are computed as

$$0.7813(0.1244) + (1 - 0.7813)0.4421 = 0.1939$$
$$0.7813(0.7553) + (1 - 0.7813)0.8521 = 0.7765$$

As the pixel value 0.7140 is less than the value of one of its neighbors, 0.7765, it is dropped from the list of edge pixels. A pixel will be retained only if its value is greater than those of its two adjacent neighbors. Similarly, two other pixels are also dropped. For $g_n(4, 4) = 0.7426$,

$$0.5774(0.7647) + (1 - 0.5774)0.4421 = 0.8089$$
$$0.5774(0.2168) + (1 - 0.5774)0.4442 = 0.3129$$

For $g_n(3, 3) = 0.7647$,

$$0.5095(0.2847) + (1 - 0.5095)0.5578 = 0.4186$$
$$0.5095(0.7426) + (1 - 0.5095)0.8960 = 0.8178$$

Step 11 The last step is hysteresis thresholding. In this step, an edge pixel with its value above the lower threshold and connected to a strong edge pixel is added to the edge pixels list. The application of the higher threshold results in edge contours with gaps. The application of the lower threshold results in several false edges. The idea is to use some of these pixels to fill up gaps resulted in the edges formed with the higher threshold, by searching for pixels in the direction of the edge. In the present example, all the selected edge pixels are connected. Therefore, there is no additional edge pixel found in the hysteresis thresholding step.

The 8×8 input image, with the edge pixels shown in boldface, is

$$x(m, n) = \begin{bmatrix} 88 & 100 & 104 & 101 & 114 & 110 & 110 & 107 \\ 92 & 102 & 104 & 107 & 112 & 110 & \mathbf{104} & 92 \\ 103 & 105 & 111 & 112 & 114 & \mathbf{108} & \mathbf{86} & 39 \\ 106 & 107 & 112 & 113 & \mathbf{107} & \mathbf{73} & 26 & 25 \\ 111 & 114 & \mathbf{14} & \mathbf{104} & 64 & 23 & 25 & 28 \\ 117 & \mathbf{115} & \mathbf{97} & 45 & 15 & 13 & 23 & 29 \\ 119 & \mathbf{93} & 32 & 0 & 15 & 11 & 19 & 23 \\ 88 & 15 & 0 & 2 & 10 & 13 & 11 & 15 \end{bmatrix}$$

An edge indicates an abrupt change in intensity. It is clear from the values of the matrix on both sides of the edge that the Canny edge detection algorithm is quite efficient, justifying its popularity at present.

9.3 Laplacian of Gaussian

While the gradient peaks are the indicators of edges using the first derivatives, zero-crossings are the indicators of edges using the second derivatives. A zero-crossing in one or more directions is an occurrence of zero-crossing. This type of masks seems to give better response when the edge transition is wider.

The 3×3 discrete Laplacian edge detector

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & \mathbf{-4} & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

is the same as that used in earlier chapters for image sharpening. One advantage of the Laplacian is that its response is same at all angles. However, it tends to amplify the noise components of the image significantly. Therefore, it is often used in combination with a Gaussian lowpass filter, called Laplacian of Gaussian (LoG). The steps involved in implementing the LoG filter are:

1. Reduce the noise by filtering the image with the Gaussian lowpass filter.
2. Apply the Laplacian to the smoothed image.
3. Detect the zero-crossings to find the edge image.

The 2-D Gaussian function is given by

$$g(x, y) = e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

Now, we have to find the Laplacian of this function. Taking the partial derivative with respect to x and y, we get

$$(-\frac{x}{\sigma^2})e^{-\frac{(x^2+y^2)}{2\sigma^2}} + (-\frac{y}{\sigma^2})e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

Taking the partial derivative with respect to x and y again, we get

$$\nabla^2 g(x, y) = (\frac{x^2}{\sigma^2} - 1)(\frac{1}{\sigma^2})e^{-\frac{(x^2+y^2)}{2\sigma^2}} + (\frac{y^2}{\sigma^2} - 1)(\frac{1}{\sigma^2})e^{-\frac{(x^2+y^2)}{2\sigma^2}} = \left(\frac{1}{\sigma^2}\right)\left(\frac{(x^2+y^2)}{\sigma^2} - 2\right)e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

The amplitude response of the filter is a function of σ. Zero-crossing occurs at $x^2 + y^2 = 2\sigma^2$, a circle with center at the origin and radius $\sqrt{2}\sigma$. This filter combines both smoothing and second-order differentiation.

The discrete version of the filter is obtained after some normalization. Let us find the coefficients of the 5×5 filter $h(m, n)$, $m = -2, -1, 0, 1, 2$, $n = -2, -1, 0, 1, 2$, and $\sigma = 0.5$. First, the coefficients of the lowpass filter

$$gl(m, n) = e^{-\frac{(m^2+n^2)}{2\sigma^2}}$$

are found.

$$g_l(m, n) = \begin{bmatrix} 0.0000 & 0.0000 & 0.0003 & 0.0000 & 0.0000 \\ 0.0000 & 0.0183 & 0.1353 & 0.0183 & 0.0000 \\ 0.0003 & 0.1353 & 1.0000 & 0.1353 & 0.0003 \\ 0.0000 & 0.0183 & 0.1353 & 0.0183 & 0.0000 \\ 0.0000 & 0.0000 & 0.0003 & 0.0000 & 0.0000 \end{bmatrix}$$

For example, with $m = 0$ and $n = 0$, the central value is 1. Next, the coefficients are normalized by dividing by their sum, which is 1.6163. The central value is $1/1.6163 = 0.6187$.

$$g_{ln}(m, n) = \begin{bmatrix} 0.0000 & 0.0000 & 0.0002 & 0.0000 & 0.0000 \\ 0.0000 & 0.0113 & 0.0837 & 0.0113 & 0.0000 \\ 0.0002 & 0.0837 & 0.6187 & 0.0837 & 0.0002 \\ 0.0000 & 0.0113 & 0.0837 & 0.0113 & 0.0000 \\ 0.0000 & 0.0000 & 0.0002 & 0.0000 & 0.0000 \end{bmatrix}$$

Now, the unnormalized LoG filter coefficients are found from the defining expression

$$g_{ln}(m, n)\left(\frac{1}{\sigma^2}\right)\left(\frac{(m^2 + n^2)}{\sigma^2} - 2\right)$$

$$h_{un}(m, n) = \begin{bmatrix} 0.0000 & 0.0020 & 0.0116 & 0.0020 & 0.0000 \\ 0.0020 & 0.2720 & 0.6698 & 0.2720 & 0.0020 \\ 0.0116 & 0.6698 & -4.9495 & 0.6698 & 0.0116 \\ 0.0020 & 0.2720 & 0.6698 & 0.2720 & 0.0020 \\ 0.0000 & 0.0020 & 0.0116 & 0.0020 & 0.0000 \end{bmatrix}$$

For example, the central coefficient is, with $m = 0$ and $n = 0$ and $\sigma = 0.5$, $-8(0.6187) = -4.9495$. Now, the coefficients of the filter $h(m, n)$ are found by subtracting the value obtained dividing the sum of all the elements of $h_{un}(m, n)$, which is -1.1196, by the product of the dimensions of the filter, which is $(5)(5) = 25$, (average) so that its sum is zero. That is, $-1.1196/25 = -0.0448$.

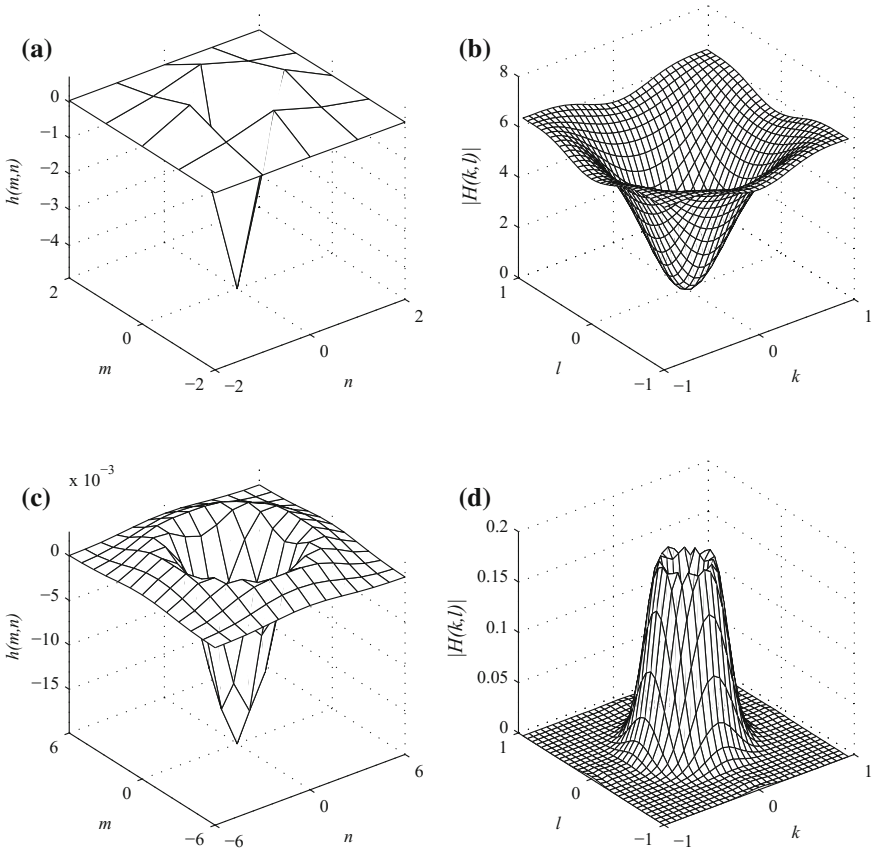


Fig. 9.8 **a** Impulse response of a 5×5 LoG filter, $\sigma = 0.5$; **b** its frequency response; **c** impulse response of a 13×13 LoG filter, $\sigma = 2$; **d** its frequency response

$$h(m,n) = \begin{bmatrix} 0.0448 & 0.0468 & 0.0564 & 0.0468 & 0.0448 \\ 0.0468 & 0.3167 & 0.7146 & 0.3167 & 0.0468 \\ 0.0564 & 0.7146 & -4.9048 & 0.7146 & 0.0564 \\ 0.0468 & 0.3167 & 0.7146 & 0.3167 & 0.0468 \\ 0.0448 & 0.0468 & 0.0564 & 0.0468 & 0.0448 \end{bmatrix}$$

For example, the central coefficient is $-4.9495 - (-0.0448) = -4.9047$. The difference 0.0001 from the actual coefficient is due to rounding. Figure 9.8a, b show, respectively, the impulse response of a 5×5 LoG filter with $\sigma = 0.5$ and its frequency response. Figure 9.8c, d show, respectively, the impulse response of a 13×13 LoG filter with $\sigma = 2$ and its frequency response. The frequency response is closer to that of a lowpass filter for large values of σ and to that of a highpass filter for small values of σ. In between, it is a bandpass filter, as shown in Fig. 9.8d. We have

used the 5×5 filter for the example. The filter size is, typically, $N \times N$, where $N = \lceil (3\sigma) \rceil 2 + 1$. A suitable value for σ has to be found by trial and error.

The same input used in Example 9.1, assuming replication at the borders, is convolved with $h(m, n)$ to get the filtered output

$$\begin{bmatrix}
28.8484 & -8.5289 & -7.0866 & 29.8151 & -28.0139 & -3.8529 & -21.5964 & -24.3066 \\
27.4631 & -5.8423 & 6.9058 & 3.8381 & -17.0194 & -24.5141 & -43.1364 & -35.9755 \\
-9.9300 & -4.8803 & -24.1955 & -22.2136 & -53.2700 & -83.7926 & -70.2963 & 114.4824 \\
0.6518 & -25.0575 & -90.9783 & -81.7078 & -101.2147 & -25.7595 & 133.1563 & 62.2465 \\
-7.5198 & -101.1569 & 421.1627 & -181.2069 & -15.2021 & 109.2424 & 29.5660 & 0.6081 \\
-40.8491 & -115.9491 & -176.7897 & 18.2622 & 101.7073 & 57.0825 & -0.8439 & -20.6883 \\
-120.7839 & -110.2583 & 77.2334 & 136.4755 & -2.6536 & 27.0237 & -6.0811 & -11.7889 \\
-53.3866 & 189.8142 & 110.3045 & 39.9276 & 1.9904 & -3.0470 & 19.7785 & 7.4534
\end{bmatrix}$$

The zero-crossings of the filtered output, subject to some threshold, are the locations of the edge pixels. For most images, it is found that zero-valued response is unlikely since the edges are mostly of ramp type, not step. A procedure to detect the zero-crossings is to ensure that both the following conditions hold.

1. The value of the pixel is negative and at least one of its four neighbors adjacent to it is positive.
2. The magnitude of the difference between the values of the two pixels is greater than a given threshold.

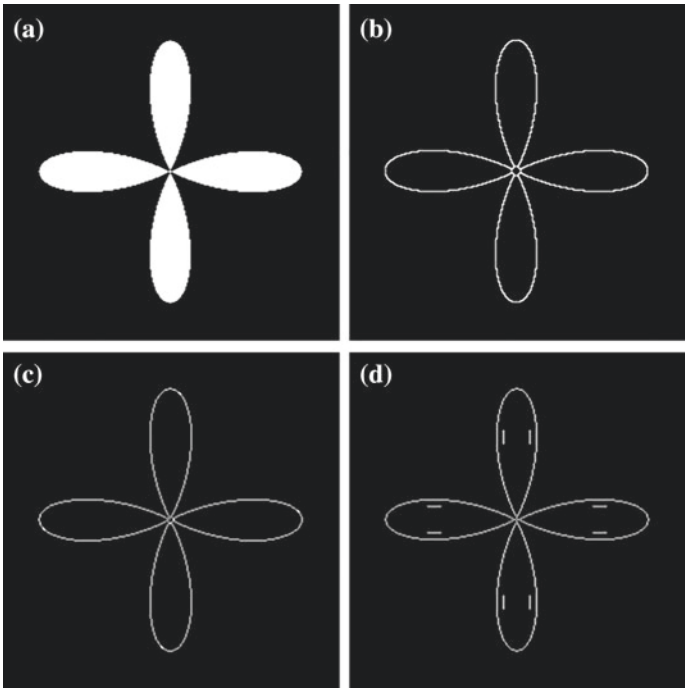


Fig. 9.9 **a** A 256×256 synthetic image; Edge maps by **b** Canny, **c** Sobel and **d** LoG methods

An additional checkup is required in case the pixel value is zero. In this case, at least one of the two adjacent pair of neighbors must have values with opposite sign. The threshold value becomes doubled.

With the threshold 41.4424, which is 0.75 times the average of the magnitude of the values of the filtered output, the edge map, for the example, is

$$e_{LoG}(m,n) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad e_{canny}(m,n) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Compare this edge map with that obtained by the Canny algorithm (right).

Figure 9.9a, b, c, d show, respectively, a 256 × 256 synthetic image and the edge map found by Canny, Sobel and LoG methods. Figure 9.10a, b, c, d show, respectively, a 256 × 256 image and the edge map found by Canny, Sobel, and LoG methods.

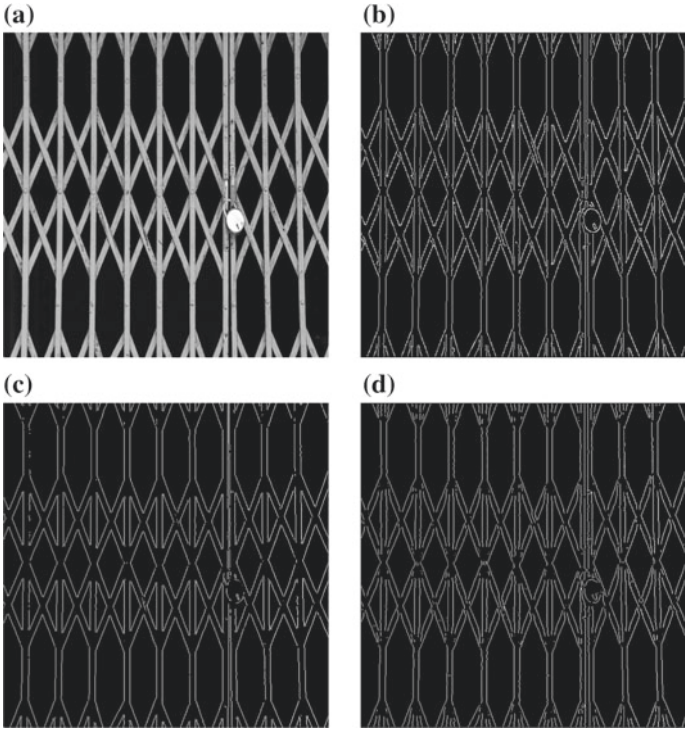


Fig. 9.10 **a** A 256 × 256 image; Edge maps by **b** Canny, **c** Sobel and **d** LoG methods

9.4 Summary

- The derivative (gradient) of a function, which is an indicator of its instantaneous rate of change, is one of its important characteristics.
- The gradient is approximated by difference in discrete signal analysis.
- In image processing, the gradient is used to find the edges (boundary between two regions of an image characterized by abrupt change in the gray level), which segment an image into its constituent components.
- Convolution and thresholding are the major operations in detecting edges in an image.
- The various edge detectors are different approximations of the gradient.
- One of the two principal ways to approximate the gradient is to find the peak of the first difference of the image.
- Another way to approximate the gradient is to find the zero-crossing of the second difference of the image.
- The thresholding of the gradient image yields the edge map.
- For noisy images, smoothing followed by edge detection is more effective.
- Thinning of the edges and using two thresholds yields an effective edge map.
- The choice of the edge detection method depends on the requirements of the application such as the accuracy, simplicity, computational complexity, and the nature of the input image.
- For simplicity, the Sobel edge detector is recommended. For accuracy, the Canny edge detection algorithm is recommended.

Exercises

9.1 Find the 6×6 filtered magnitude output and the edge map of $x(m, n)$ obtained using the Sobel filters. The threshold is 1.2 times the average of the magnitude of the values of the filtered output.

*(i)

$$x(m,n) = \begin{bmatrix} 227 & 179 & 45 & 10 & 14 & 15 & 14 & 12 \\ 227 & 227 & 177 & 18 & 15 & 17 & 15 & 14 \\ 227 & 227 & 225 & 56 & 8 & 15 & 18 & 15 \\ 227 & 227 & 227 & 98 & 9 & 17 & 18 & 16 \\ 227 & 227 & 227 & 141 & 7 & 17 & 18 & 17 \\ 227 & 227 & 227 & 194 & 26 & 9 & 14 & 14 \\ 227 & 227 & 227 & 214 & 92 & 0 & 11 & 11 \\ 227 & 227 & 227 & 212 & 184 & 64 & 5 & 14 \end{bmatrix}$$

(ii)

$$x(m,n) = \begin{bmatrix} 168 & 153 & 111 & 58 & 0 & 0 & 0 & 42 \\ 159 & 161 & 114 & 79 & 9 & 2 & 5 & 42 \\ 134 & 181 & 124 & 67 & 86 & 20 & 15 & 26 \\ 117 & 180 & 122 & 79 & 74 & 47 & 46 & 47 \\ 152 & 181 & 132 & 70 & 36 & 38 & 40 & 49 \\ 156 & 171 & 117 & 59 & 73 & 77 & 61 & 67 \\ 174 & 166 & 124 & 84 & 69 & 57 & 55 & 55 \\ 172 & 159 & 129 & 93 & 84 & 31 & 22 & 23 \end{bmatrix}$$

(iii)

$$x(m,n) = \begin{bmatrix} 13 & 10 & 13 & 9 & 7 & 5 & 4 & 2 \\ 6 & 10 & 12 & 10 & 6 & 6 & 2 & 0 \\ 19 & 19 & 12 & 11 & 6 & 3 & 2 & 2 \\ 19 & 19 & 18 & 18 & 10 & 5 & 2 & 3 \\ 19 & 18 & 16 & 17 & 14 & 10 & 7 & 6 \\ 17 & 14 & 15 & 15 & 10 & 9 & 8 & 6 \\ 13 & 10 & 11 & 13 & 12 & 9 & 8 & 3 \\ 14 & 9 & 9 & 9 & 10 & 10 & 8 & 5 \end{bmatrix}$$

9.2 Find the filtered output and the edge map of $x(m,n)$ obtained using a 5×5 LoG filter with $\sigma = 1$. The threshold is 0.75 times the average of the magnitude of the values of the filtered output. Assume replication at the borders.

(i)

$$x(m,n) = \begin{bmatrix} 25 & 17 & 22 & 8 & 118 & 186 & 136 & 133 \\ 15 & 38 & 18 & 13 & 152 & 147 & 131 & 150 \\ 33 & 32 & 14 & 9 & 115 & 165 & 143 & 173 \\ 34 & 14 & 14 & 12 & 21 & 64 & 41 & 179 \\ 15 & 14 & 7 & 18 & 106 & 137 & 92 & 195 \\ 15 & 24 & 39 & 156 & 188 & 194 & 191 & 197 \\ 12 & 24 & 129 & 204 & 204 & 206 & 208 & 195 \\ 0 & 7 & 145 & 206 & 206 & 209 & 211 & 200 \end{bmatrix}$$

*(ii)

$$x(m,n) = \begin{bmatrix} 147 & 163 & 179 & 186 & 191 & 194 & 197 & 157 \\ 160 & 175 & 182 & 184 & 184 & 186 & 162 & 50 \\ 141 & 163 & 170 & 175 & 174 & 133 & 38 & 3 \\ 91 & 127 & 135 & 124 & 85 & 16 & 0 & 7 \\ 113 & 126 & 121 & 117 & 18 & 0 & 1 & 10 \\ 136 & 135 & 125 & 151 & 99 & 54 & 8 & 9 \\ 148 & 150 & 159 & 161 & 149 & 106 & 89 & 20 \\ 142 & 164 & 178 & 181 & 168 & 113 & 120 & 91 \end{bmatrix}$$

(iii)

$$x(m,n) = \begin{bmatrix} 72 & 77 & 66 & 62 & 50 & 43 & 66 & 66 \\ 75 & 65 & 61 & 65 & 50 & 36 & 64 & 64 \\ 67 & 57 & 62 & 67 & 48 & 31 & 63 & 63 \\ 55 & 62 & 64 & 62 & 16 & 34 & 61 & 61 \\ 41 & 46 & 47 & 46 & 11 & 33 & 59 & 62 \\ 39 & 28 & 58 & 54 & 25 & 7 & 50 & 61 \\ 41 & 0 & 49 & 48 & 28 & 9 & 6 & 29 \\ 63 & 20 & 6 & 23 & 16 & 10 & 7 & 15 \end{bmatrix}$$

Chapter 10
Segmentation

Abstract Segmentation of an image is its partition into disjoint regions. Various segmentation methods are based on finding the interior of the region or its border. The border of a region can be found by edge detection. The interior of the region is determined by the distinct properties of the pixels comprising the region. Thresholding, region-based methods, and watershed are typical segmentation algorithms.

A typical image consists of regions, which are connected and exhibit distinct characteristics with respect to some measure, such as histogram bands and textural properties. With respect to these characteristics, it is possible to segment an image into regions. Image segmentation tries to identify and label the different objects and regions, the image is composed of. For example, the characters in a word have to be segmented before their identification. There are two basic methods of segmentation: (i) finding the boundary of a region and (ii) finding the interior of a region. Finding the boundary is typically based on the abrupt changes in gray levels. Finding the interior is based on the pixels having similar properties. If each region is composed of distinct band of gray levels, then, by appropriate thresholding, they can be segmented. It is assumed that each object is spatially connected.

A region can be defined by its interior part or border (edge). The homogeneity P of a region R is defined as

$$P(R) = \begin{cases} 1, \text{ if } f(R) \in H \\ 0, \text{ otherwise} \end{cases}$$

where f is function defining the homogeneity, and H is the predefined range of values of f. The function f can be defined in any suitable way. For example, it could be the standard deviation of the region, the mean, the difference between the largest and smallest values of pixels, co-occurrence matrices, or a gray-level threshold. Segmentation of an image is its partitioning into a set of N connected regions $R(n), n = 0, 1, \ldots, N-1$. Some of the points to be noted are:

D. Sundararajan, *Digital Image Processing*, DOI 10.1007/978-981-10-6113-4_10

1. The sum of all the regions is exactly equal to the image.
2. A pixel of the image belongs to only one of the regions.
3. The predicate of a region holds for all its pixels.
4. The predicates of adjacent regions must be different.

Edge detection, thresholding, region growing and splitting, and watershed segmentation are the basic approaches of segmentation.

10.1 Edge-Based Segmentation

Segmentation of a region by edge detection is based on abrupt change in the intensity of the pixels at the borders. The edges obtained are assumed to be the object boundaries and used to find the objects. Edge detection has been presented in the last chapter. Edges found, however, may not characterize a region accurately due to breaks in the edge contour because of noise and nonuniform illumination. Therefore, edge linking is usually followed by edge detection to make the boundary of the objects more useful. In this section, point and line detection is presented.

10.1.1 Point Detection

The mask, which is one of the versions of the Laplacian masks presented in Chap. 2 multiplied by -1, for point detection is

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & \mathbf{8} & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

The mask is applied to the image, and the output is thresholded to detect the isolated points (segmentation of points). Note that, for edge detection, the Laplacian response is analyzed for zero-crossings.

Figure 10.1a shows a 256×256 image. Figure 10.1b shows the locations of white dots in the three roses. The output of point detection, for each dot, is a single pixel. For clear visibility, a dilated version is shown in (b).

10.1.2 Line Detection

The masks for line detection at four directions (E-W, NW-SE, N-S, NE-SW) are, respectively,

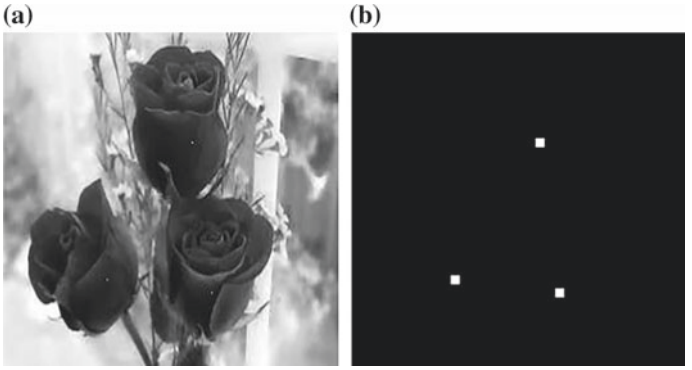


Fig. 10.1 **a** A 256 × 256 image; **b** image showing the locations of white dots in the three roses

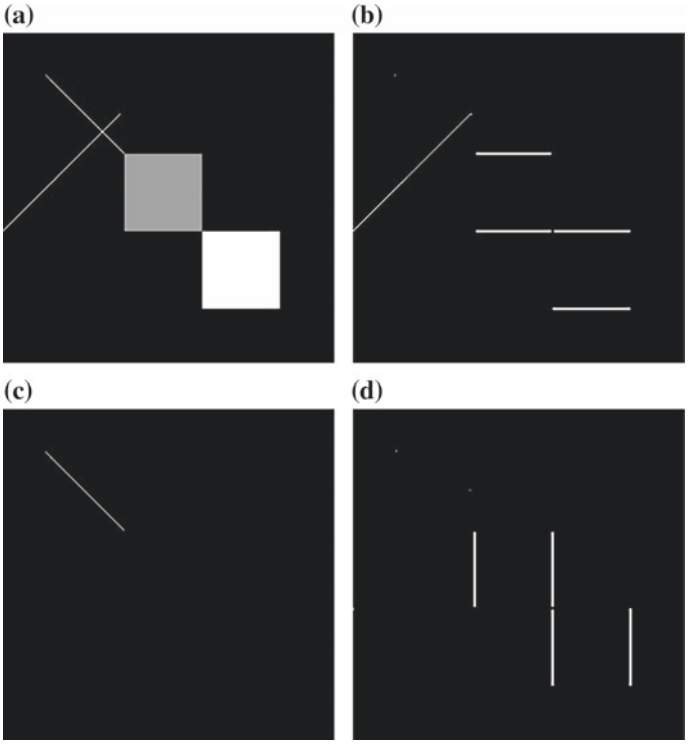


Fig. 10.2 **a** A 256 × 256 gray-level image; **b** line from the southwest to northeast direction and lines from east to west direction; **c** line from the northwest to southeast direction; **d** lines from north to south direction

$$\begin{bmatrix} -1 & -1 & -1 \\ 2 & \mathbf{2} & 2 \\ -1 & -1 & -1 \end{bmatrix} \quad \begin{bmatrix} 2 & -1 & -1 \\ -1 & \mathbf{2} & -1 \\ -1 & -1 & 2 \end{bmatrix} \quad \begin{bmatrix} -1 & 2 & -1 \\ -1 & \mathbf{2} & -1 \\ -1 & 2 & -1 \end{bmatrix} \quad \begin{bmatrix} -1 & -1 & 2 \\ -1 & \mathbf{2} & -1 \\ 2 & -1 & -1 \end{bmatrix}$$

The masks produce the strongest response for lines oriented in the corresponding directions. The pixels in the preferred directions are given a stronger weight. The sum of all the coefficients of each mask is zero, since no response should be produced in constant intensity regions of the image. For example, the first mask produces the strongest response for a line from east to west. With a constant background, the maximum response occurs when the middle-row coefficients are aligned with the line.

Figure 10.2b shows the detection of the line from the southwest to northeast direction and lines from east to west direction. Figure 10.2c shows the detection of the line from the northwest to southeast direction. Figure 10.2d shows the detection of the lines from north to south direction.

10.2 Threshold-Based Segmentation

The simplest approach to segmentation is to partition an image into regions based on different ranges of their pixel values. The histogram is an aid in this process. An image typically has 256 gray levels. If we are able to determine the threshold between an object and the background, then the region corresponding to the object is labeled by selecting the pixels those are in the object range of gray levels. The thresholding process of an image $x(m, n)$, yielding its segmented version $R(m, n)$, is given by

$$R(m,n) = \begin{cases} o, & \text{for } x(m,n) \geq T \\ b, & \text{otherwise} \end{cases}$$

where T is the threshold, and o and b represent the object and the background. Of course, there can be more than one threshold partitioning the image into several regions. For example,

$$R(m,n) = \begin{cases} o1, & \text{for } x(m,n) > T1 \\ o2, & \text{for } T0 < x(m,n) \leq T1 \\ b, & \text{otherwise} \end{cases}$$

Pixels in the gray-level range $x(m, n) > T1$ are labeled as object 1, and those in the range $T0 < x(m, n) \leq T1$ are labeled as object 2. The rest of the pixels are labeled as background.

Figure 10.3a and b show, respectively, a 256 × 256 gray level image and its histogram. The image is composed of the background, metal bars of the grill door, and the lock with pixel values around 5, 170, and 245, respectively. As can be seen

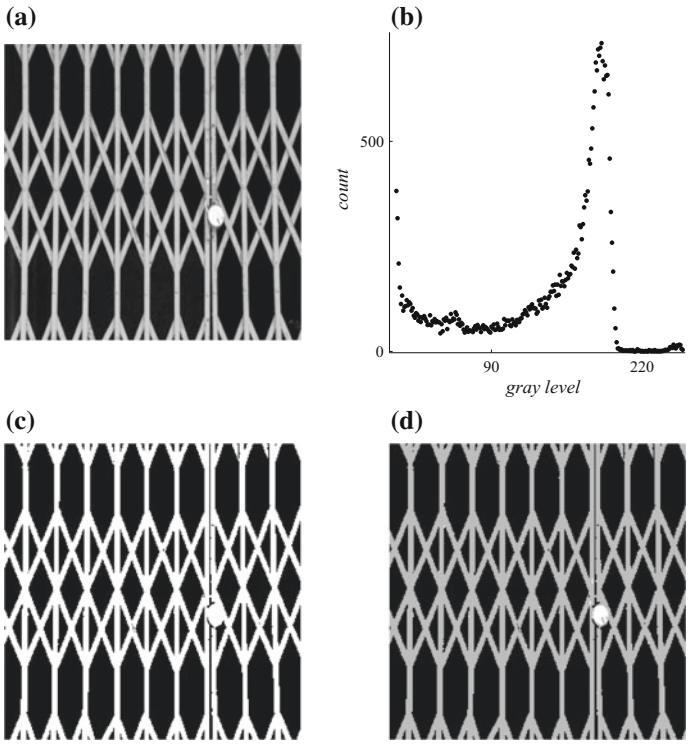


Fig. 10.3 **a** A 256 × 256 gray-level image; **b** its histogram; **c** segmenting the image with one threshold; **d** segmenting the image with two thresholds

from the histogram, most of the pixels belong to the background (not shown). With threshold 90, the segmentation of the image is shown in Fig. 10.3c. The background is black with gray level 0, and the rest is white with gray level 255. Normally, the regions in a segmented image are assigned integer labels. With one threshold, while we are able to isolate the background, we are not able to partition the lock from the bars. With two thresholds 90 and 220, the background, the bars, and the lock regions are segmented, as shown in Fig. 10.3d. The bars and the lock are assigned the gray levels 168 and 255, respectively.

Picking the lowest point in the valley between peaks of the histogram is a practical choice for the threshold. The histogram of the example image has clear valleys so that the selection of the thresholds and, hence, the segmentation of the image are easy. For any application of thresholding, the selection of the appropriate threshold is the key step. While algorithms have been developed for the determination of optimum threshold values, the manual approach, by trial and error, is the best in most of the cases. However, it may not be feasible if the number of images is very large, and algorithms are required for automatic determination of the threshold.

A simple algorithm to find the threshold is to iteratively separate the gray-level values into two groups based on an initial threshold, take the average of the averages of the two groups as the new threshold, and continue the process until the difference between the estimated threshold values of two consecutive iterations is within some limit. Consider the 4×4 image

$$\begin{bmatrix} 185 & 182 & 45 & 2 \\ 188 & 140 & 10 & 5 \\ 189 & 74 & 2 & 7 \\ 164 & 21 & 5 & 6 \end{bmatrix}$$

Let the initial threshold be the average gray level of the image, 76.5625. With this threshold, we get two groups of pixels

$$\begin{bmatrix} 185 & 182 \\ 188 & 140 \\ 189 & \\ 164 & \end{bmatrix} \quad \begin{bmatrix} & 45 & 2 \\ & 10 & 5 \\ 74 & 2 & 7 \\ 21 & 5 & 6 \end{bmatrix}$$

The first group has values greater than 76.5625 with the average 174.6667. The second group has the remaining values with the average 17.7. The average of the values, 96.1833, is the new threshold value. In the next iteration, we get the same value, and the algorithm terminates. The segmented image is

$$\begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

This simple algorithm works well for histograms with a reasonably clear valley between peaks.

10.2.1 Thresholding by Otsu's Method

This method maximizes the between-class variance, and it is based on the histogram of the image. Let the range of gray levels of the image be from 0 to $L-1$, and the normalized histogram values be $hn(k), k = 0, 1, \ldots L-1$. Let the threshold T be $k,\ k = 0, 1, \ldots, L-1$. Then, the pixels are placed in two groups $g1$ and $g2$ with the first group consists of pixels with gray levels in the range from 0 to k, and the other group consists of pixels with gray levels in the range from $k+1$ to $L-1$. The between-class variance, $\sigma_b^2(k)$, is given by

$$\sigma_b^2(k) = hc1(k)(ha1(k) - ha(L-1))^2 + hc2(k)(ha2(k) - ha(L-1))^2 \quad (10.1)$$

where the normalized cumulative histograms and the average intensities are defined as

$$hc1(k) = \sum_{i=0}^{k} hn(i), \quad hc2(k) = \sum_{i=k+1}^{L-1} hn(i) = 1 - hc1(k),$$

$$ha1(k) = \frac{1}{hc1(k)} \sum_{i=0}^{k} (i)\, hn(i), \quad ha2(k) = \frac{1}{hc2(k)} \sum_{i=k+1}^{L-1} (i)\, hn(i),$$

$$ha(k) = \sum_{i=0}^{k} (i)\, hn(i), \quad ha(L-1) = \sum_{i=0}^{L-1} (i)\, hn(i)$$

Since

$$hc2(k) = 1 - hc1(k), \quad ha1(k) = \frac{ha(k)}{hc1(k)} \quad \text{and} \quad ha2(k) = \frac{ha(L-1) - ha(k)}{1 - hc1(k)},$$

we get $\sigma_b^2(k)$ as

$$\sigma_b^2(k) = \frac{(ha(L-1)hc1(k) - ha(k))^2}{hc1(k)(1 - hc1(k))} \tag{10.2}$$

It is clear from Eq. 10.1 that the larger the difference between the means $ha1(k)$ and $ha2(k)$, the larger is the between-class variance. A measure of the separability of the image intensities, in the range 0–1, is given by

$$\frac{\sigma_b^2(k)}{\sigma^2}$$

where $\sigma^2 > 0$ is the image variance. If all the pixels have the same gray-level value, then only the variance is zero, and, in that case, it is not possible to separate the histogram. Equation 10.2 is convenient for computation, since it requires fewer variables to compute the variance.

Consider the 4×4 3-bit image

$$\begin{bmatrix} 2 & 7 & 6 & 6 \\ 5 & 6 & 5 & 5 \\ 6 & 5 & 5 & 6 \\ 7 & 6 & 4 & 5 \end{bmatrix}$$

The normalized histogram $hn(k)$, and its cumulative sum $hc1(k)$, the average intensity $ha(k)$, and the variance $\sigma_b^2(k)$ are shown, respectively, in the rows from 2 to 5 in the following table.

Since it is a 3-bit image, the gray-level range, shown in the first row, varies from 0 to 7. For example, $\sigma_b^2(2)$

Gray level, k	0	1	2	3	4	5	6	7
$hn(k)$	0	0	0.0625	0	0.0625	0.3750	0.3750	0.1250
$hc1(k)$	0	0	0.0625	0.0625	0.1250	0.5	0.8750	1
$ha(k)$	0	0	0.1250	0.1250	0.3750	2.2500	4.5000	5.3750
$\sigma_b^2(k)$	0	0	0.7594	0.7594	0.8058	0.7656	0.3772	0

$$\frac{((5.375)(0.0625) - 0.125)^2}{(0.0625(1 - 0.0625))} = 0.7594$$

The index of the maximum variance 0.8058 is 4, and it is the optimum threshold value T. If the maximum variance occurs more than once, then the average value of the indices is taken as the threshold. The measure of the separability is given by

$$\frac{0.8058}{\sum_{k=0}^{7}(k - 5.375)^2 hn(k)} = 0.5928$$

This measure varies from 0 (for an image with a single gray level) to 1 (for a 2-valued image with gray levels 0 and $L - 1$ only). Figure 10.4a shows a 256×256 gray-level

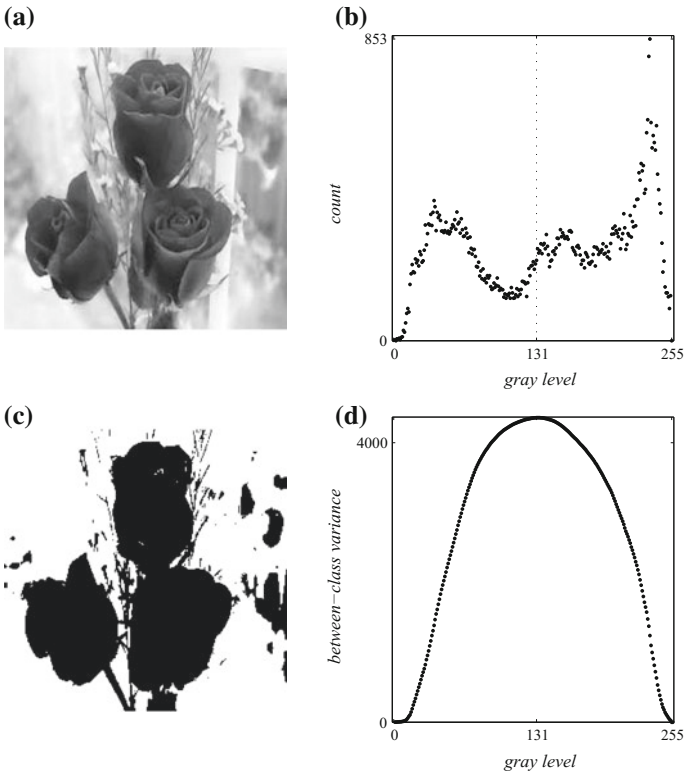


Fig. 10.4 **a** A 256×256 gray-level image; **b** its histogram; **c** segmented image; **d** the between-class variance

image, and (b) shows its histogram with the threshold 131. The segmented image is shown in (c). The between-class variance is shown in (d).

The two algorithms presented produce nearly the same threshold when the separability measure is high. In other cases, where thresholding is effective in segmentation, Otsu's method produces better results.

10.2.1.1 Segmenting Noisy Images

When an image is degraded due to noise, its histogram becomes somewhat flat, and segmentation becomes difficult. Figure 10.5a shows a 256 × 256 gray-level noisy image. The noise is Gaussian with mean zero and standard deviation 0.04. The valleys and peaks in the histogram of the original image have disappeared in the histogram, shown by dots in (b). If we apply the Otsu's method and segment, the resulting image is shown in (c). All the white dots in the black regions and vice versa are segmentation errors. Applying a lowpass 5 × 5 averaging filter reduces the noise,

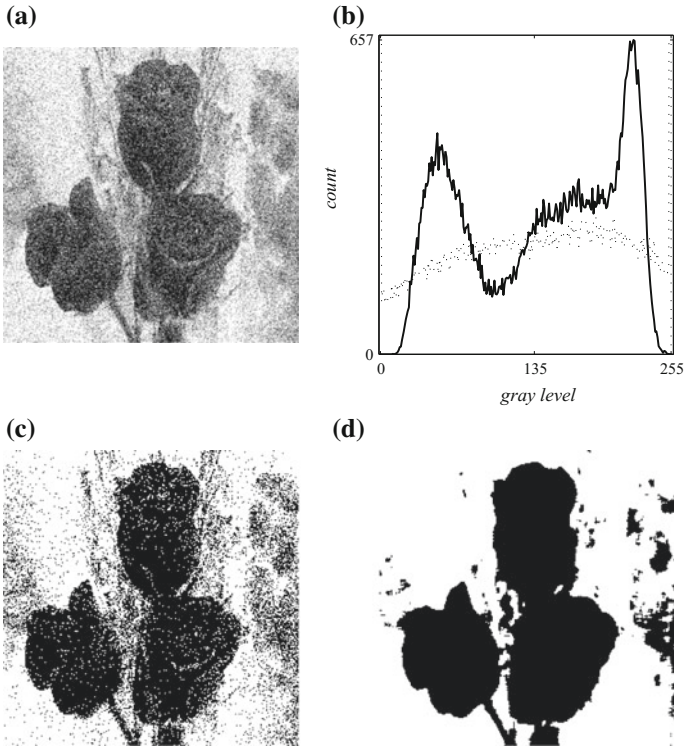


Fig. 10.5 **a** A 256 × 256 gray-level noisy image; **b** its histogram (*dots*) and the histogram (*line*) after noise reduction; **c** segmented noisy image; **d** segmented image after noise reduction

and the original histogram is almost restored, as shown by a line in (b). The result of thresholding is shown in (d). The thresholds of the noisy and filtered images are 134 and 129, respectively.

10.3 Region-Based Segmentation

In this type of segmentation, pixels are grouped into regions based on some similarity criteria and connectivity. For a pixel to be a part of a region, it should have the attributes characterizing the region and also should meet some connectivity constraints. Given a set of pixels, if we can identify at the least one 4-connected path between any pair of pixels, then it is a 4-connected region. Both 4- and 8-connectivities are often used in image-processing operations. An image can be considered as the union of disjoint regions, each region characterizing an object. A pixel belongs to any one of the regions of the image. The pixels in a region are connected. Typical attributes characterizing a region are:

- The average of the gray values of the pixels in a region is significantly different from that of the image.
- The standard deviation of the gray-level values of the pixels in the region is within a distinct range.
- Pixels in the region exhibit distinct textural properties.

10.3.1 Region Growing

In the region-growing method of segmentation, we start with a pixel, called a seed pixel, and start checking the similarity of the attributes of the pixels and the connectivity. All the pixels satisfying the criteria are collected, and they form the region. The process is continued until all the pixels of the image are assigned to some region.

Consider the 8×8 image $x(m, n)$.

$$\begin{bmatrix} 179 & 179 & 183 & 180 & 185 & 183 & 182 & 175 \\ 175 & 179 & 185 & 179 & 181 & 179 & 177 & 173 \\ 180 & 183 & 181 & 170 & 181 & 176 & 174 & 174 \\ 181 & 181 & 182 & 180 & 174 & 176 & 179 & 175 \\ 185 & 184 & 185 & 177 & 172 & 176 & 174 & 170 \\ 184 & 184 & 182 & 182 & 175 & 172 & 165 & 167 \\ 177 & 185 & 181 & 175 & 171 & 166 & 165 & 169 \\ 179 & 184 & 177 & 173 & 170 & 171 & 171 & 172 \end{bmatrix} \quad R_0(m, n) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Let the seed pixel be $x(2, 5) = 176$, shown in boldface. Let us use the 4-connectivity, and the region is to be made of pixels with gray levels less than or equal to 176. The

seed pixel $R_0(2, 5)$ is 1 in the region matrix $R_0(m, n)$. The pixels are examined row by row. In the first iteration, the four neighbors

$$\{x(2, 5), x(2, 7), x(1, 6), x(3, 6)\}$$

of $x(2, 6)$ are examined. Obviously, pixel $x(2, 6)$ belongs to the region and entered in the 8×8 $R_1(m, n)$ matrix with 1 at the corresponding coordinates $(2, 6)$. This entry is done only once for each pixel. Then, pixel $x(2, 7)$ is also found to be in the region. Out of bounds, pixels are to be ignored. That is, pixels in the defined image only are considered for connectivity determination. Then, the pixels in the next row are scanned. At the end of the first iteration, the region matrix $R_1(m, n)$ is

$$R_1(m, n) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}$$

After two more iterations, the final region map is obtained.

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

With 8-connectivity, the final region matrix is

$$R(m, n) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

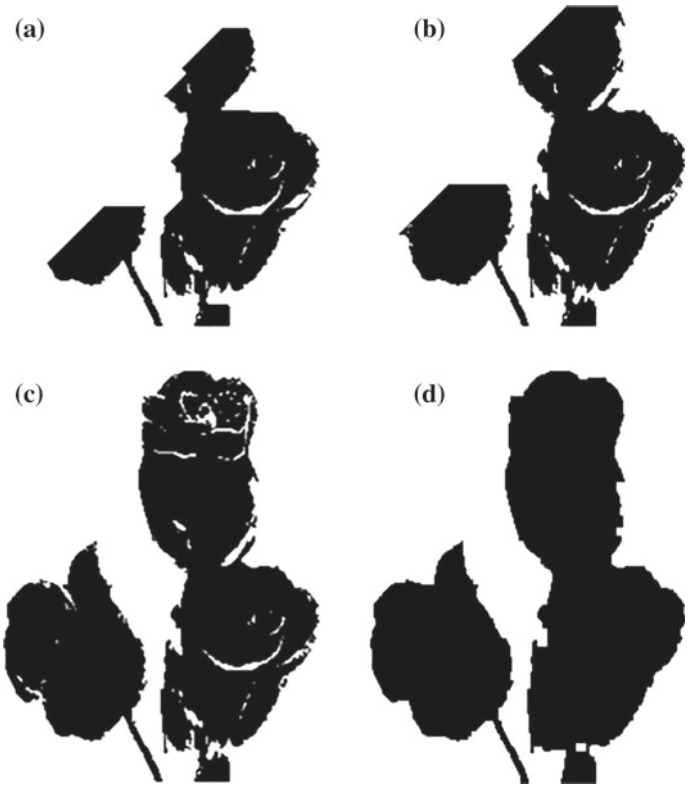


Fig. 10.6 **a** and **b** Partially segmented image during initial iterations; **c** the segmented image; **d** the image after morphological closing operation

One more pixel at coordinates (2, 3) is in the region compared with that of the 4-connectivity. The algorithm can also be used for growing a set of regions, each with a single seed pixel.

Figure 10.6a–d shows the segmentation of the image in Fig. 10.1a by region growing. The threshold is 89, and 8-connectivity condition is used. Figure 10.6a and b show partially segmented image during initial iterations. The segmented image, shown in Fig. 10.6c, has some small white patches. These can be removed by morphological closing operation, as shown in Fig. 10.6d.

The main limitation of this method is its dependence upon the location of the seeds. If the location of the seed is not available, then one procedure is to cluster the pixels of the regions, find the centroids, and use the coordinates of the pixels in the neighborhood of the centroids as the seed points. Figure 10.7 shows finding the location of the seeds by clustering. A 8×8 image with two regions, one marked by dots (4 pixels) and the other by crosses (9 pixels), is shown in the figure. Any

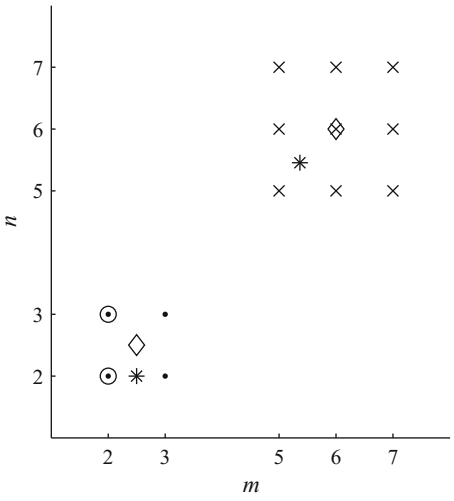


Fig. 10.7 Finding the location of the seeds by clustering

two points of the regions can be selected as initial centroids, for example (2, 2) and (2, 3), shown by circles in the figure. Now, points closest to each centroid are grouped together, and a new centroid is found. The points closest to (2, 2) are (2, 2) and (2, 3). They form the first group after the first iteration. The new centroid of this group is (2.5, 2), shown by asterisk. The rest of the 11 pixels form the second group, and their new centroid is (5.3636, 5.4545). In the next iteration, pixels are grouped properly with 4 in the first group and 9 in the second group, and final centroids (shown by the diamond symbol) are fixed. At this point of convergence, the sum of the distances of the pixels from their respective centroids is minimum. Even if the number of regions is given, some number of trials are recommended to fix the seed locations. If the number of regions is unknown, experimentation is the solution until the given segmentation problem is adequately solved.

10.3.2 Region Splitting and Merging

While region growing is a bottom-up approach, region splitting and merging is a top-down approach. The criteria of segmentation do not hold for the whole image, and we divide the image into subimages. This division is carried out recursively until the criteria are met and the merging of all these subimages as required in the region being formed. The data structure most suitable for this algorithm is quadtree. This is a tree in which each node, except the leaves, has four children. Each segmented region is represented by a leaf. This type of data structure is also used in image compression algorithms.

We start with a region image of the same size as the input image with all the pixel values equal to 1. The same image in the last example, after the first iteration of dividing it into four quadrants of size 4×4, yields the region map $R_1(m, n)$.

$$R_1(m,n) = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Each quadrant is checked with the constraint that the pixel value is less than or equal to 176, along with the 4-connectivity condition. Since there are no such pixels in the top-left quadrant of the image, all its entries are zeros in $R_1(m, n)$. Each of the other three quadrants are further divided into four quadrants of size 2×2. After examining the pixels, the region map, at the end of the second iteration, is

$$R_2(m,n) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

In the third iteration, the image is divided into subimages of size 1×1, and the rest of the pixels are labeled yielding the same region map as in region growing.

Figure 10.8a shows the segmentation of the image in Fig. 10.1a by region splitting and merging. The threshold is 89, and 8-connectivity condition is used. We start with a 256×256 region image with all pixels equal to zero. The image is divided into four quadrants each of size 128×128 in the first iteration. The pixels in each of the blocks are tested. Since none of the quadrants can be eliminated as a nonregion, we further divide each quadrant into blocks of size 64×64. Now, no pixel in the upper-left and bottom-right quadrants satisfies the segmentation conditions. Therefore, they are marked white as shown in (a). The process continues, and the elimination of block sizes 16×16 and 4×4 are shown, respectively, in (b) and (c). The segmented image is shown in (d).

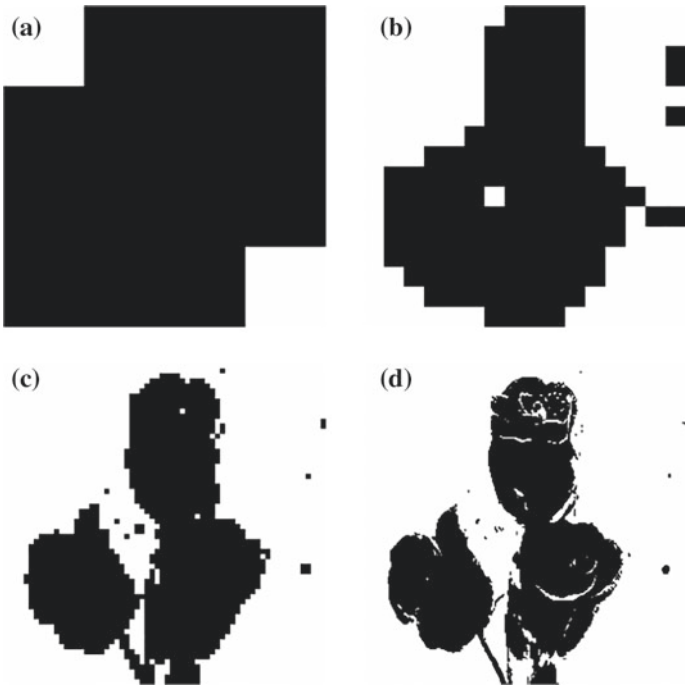


Fig. 10.8 Segmentation of a 256 × 256 image. **a**–**c** The image during iterations of the algorithm; **d** the segmented image

10.4 Watershed Algorithm

A watershed is a ridge that divides the catchment basins. The location of the watershed lines demarcates the basins. In image processing, an image is segmented using the boundaries by a process which resembles finding the watershed lines in a catchment area. The idea is to derive an image, from the input image, such that the catchment basins correspond to the constituent regions of the image. The distance between pixels is an important entity in watershed algorithms as well as in other image-processing tasks. Now, we present an algorithm to approximate the Euclidean distance.

10.4.1 The Distance Transform

It is often required to find the distance between pixels in images. The Euclidean distance between pixels $x(p, q)$ and $x(m, n)$ is defined as

$$d = \sqrt{(m-p)^2 + (n-q)^2} \tag{10.3}$$

For large number of pixels, the number of operations becomes very large. In addition, the operation is computationally intensive. The distance transform is an algorithm for computing the distances. Let the binary image $x(m, n)$ be

$$x(m,n) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The minimum distance between each pixel to its nearest pixel in the connected region defined by the 1s, using Eq. 10.3 is

$$DS(m,n) = \begin{bmatrix} 4.4721 & 3.6056 & 2.8284 & 2.2361 & 1.4142 & 1.0000 & 1.4142 & 2.2361 \\ 4.1231 & 3.1623 & 2.2361 & 1.4142 & 1.0000 & 0 & 1.0000 & 2.0000 \\ 4.0000 & 3.0000 & 2.0000 & 1.0000 & 0 & 0 & 1.0000 & 1.4142 \\ 3.6056 & 2.8284 & 2.2361 & 1.4142 & 1.0000 & 0 & 0 & 1.0000 \\ 3.1623 & 2.2361 & 1.4142 & 1.0000 & 1.0000 & 0 & 0 & 1.0000 \\ 3.0000 & 2.0000 & 1.0000 & 0 & 0 & 0 & 1.0000 & 1.4142 \\ 3.1623 & 2.2361 & 1.4142 & 1.0000 & 1.0000 & 0 & 1.0000 & 2.0000 \\ 3.6056 & 2.8284 & 2.2361 & 2.0000 & 1.4142 & 1.0000 & 1.4142 & 2.2361 \end{bmatrix}$$

The algorithm finds the approximate distances at a reduced computational complexity. The algorithm is essentially passing 2 masks over the image. The forward and backward masks are

$$h_f(m,n) = \begin{bmatrix} \infty & 1 & \infty \\ 1 & 0 & \infty \\ \infty & \infty & \infty \end{bmatrix} \quad \text{and} \quad h_b(m,n) = \begin{bmatrix} \infty & \infty & \infty \\ \infty & 0 & 1 \\ \infty & 1 & \infty \end{bmatrix}$$

The given image is sufficiently zero-padded, and all the zero-valued pixels are assigned a value of ∞, and the pixels with value 1 are assigned the value 0, giving a initial distance matrix DS as

$$DS(m,n) = \begin{bmatrix} \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & 0 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & 0 & 0 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & 0 & 0 & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & 0 & 0 & \infty & \infty \\ \infty & \infty & \infty & \infty & 0 & 0 & 0 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & 0 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty \end{bmatrix}$$

Now, the forward mask is passed over the matrix, starting from top-left corner of the image. The mask is moved from left to right and top to bottom. At each pixel, the pixels of the mask are added with the corresponding pixels of the image. The minimum value of this set replaces current value in the DS matrix. At each pixel location, the updated values of the DS matrix are to be used in the computation, not those of the initial DS matrix. For example, the pixels in the third row are updated as

$$\begin{bmatrix} \infty & 1 \\ 1 & 0 \end{bmatrix} + \begin{bmatrix} \infty & \infty \\ \infty & 0 \end{bmatrix} = \begin{bmatrix} \infty & \infty \\ \infty & 0 \end{bmatrix}$$

$$\begin{bmatrix} \infty & 1 \\ 1 & 0 \end{bmatrix} + \begin{bmatrix} \infty & \infty \\ 0 & \infty \end{bmatrix} = \begin{bmatrix} \infty & \infty \\ 1 & \infty \end{bmatrix}$$

$$\begin{bmatrix} \infty & 1 \\ 1 & 0 \end{bmatrix} + \begin{bmatrix} \infty & \infty \\ 1 & \infty \end{bmatrix} = \begin{bmatrix} \infty & \infty \\ 2 & \infty \end{bmatrix}$$

The result of the first pass is

$$DS(m,n) = \begin{bmatrix} \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & 0 & 1 & 2 & \infty \\ \infty & \infty & \infty & \infty & \infty & 0 & 0 & 1 & 2 & \infty \\ \infty & \infty & \infty & \infty & \infty & 1 & 0 & 0 & 1 & \infty \\ \infty & \infty & \infty & \infty & \infty & 2 & 0 & 0 & 1 & \infty \\ \infty & \infty & \infty & \infty & 0 & 0 & 0 & 1 & 2 & \infty \\ \infty & \infty & \infty & \infty & 1 & 1 & 0 & 1 & 2 & \infty \\ \infty & \infty & \infty & \infty & 2 & 2 & 1 & 2 & 3 & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty \end{bmatrix}$$

Remember that $\infty + 1 = \infty$. Now, the backward mask is passed over this matrix, starting from bottom-right corner of the image. The mask is moved from right to left and bottom to top. At each pixel, the pixels of the mask are added with the corresponding pixels of the image. The minimum value of this set replaces current value in the DS matrix. The result of the second pass is

$$DS(m,n) = \begin{bmatrix} 6 & 5 & 4 & 3 & 2 & 1 & 2 & 3 \\ 5 & 4 & 3 & 2 & 1 & 0 & 1 & 2 \\ 4 & 3 & 2 & 1 & 0 & 0 & 1 & 2 \\ 5 & 4 & 3 & 2 & 1 & 0 & 0 & 1 \\ 4 & 3 & 2 & 1 & 1 & 0 & 0 & 1 \\ 3 & 2 & 1 & 0 & 0 & 0 & 1 & 2 \\ 4 & 3 & 2 & 1 & 1 & 0 & 1 & 2 \\ 5 & 4 & 3 & 2 & 2 & 1 & 2 & 3 \end{bmatrix}$$

These distances are approximate. A more accurate result can be obtained using integer-valued masks and by increasing the mask size. Consider the forward and backward masks

$$h_f(m,n) = \begin{bmatrix} 4 & 3 & 4 \\ 3 & 0 & \infty \\ \infty & \infty & \infty \end{bmatrix} \quad \text{and} \quad h_b(m,n) = \begin{bmatrix} \infty & \infty & \infty \\ \infty & 0 & 3 \\ 4 & 3 & 4 \end{bmatrix}$$

With these masks, the first pass result is

$$DS(m,n) = \begin{bmatrix} \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & 0 & 3 & 6 & \infty \\ \infty & \infty & \infty & \infty & \infty & 0 & 0 & 3 & 6 & \infty \\ \infty & \infty & \infty & \infty & 4 & 3 & 0 & 0 & 3 & \infty \\ \infty & \infty & \infty & 8 & 7 & 4 & 0 & 0 & 3 & \infty \\ \infty & \infty & 12 & 11 & 0 & 0 & 0 & 3 & 4 & \infty \\ \infty & 16 & 15 & 4 & 3 & 3 & 0 & 3 & 6 & \infty \\ \infty & 19 & 8 & 7 & 6 & 4 & 3 & 4 & 7 & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty \end{bmatrix}$$

and the second pass result is

$$DS(m,n) = \begin{bmatrix} 14 & 11 & 8 & 7 & 4 & 3 & 4 & 7 \\ 13 & 10 & 7 & 4 & 3 & 0 & 3 & 6 \\ 12 & 9 & 6 & 3 & 0 & 0 & 3 & 4 \\ 11 & 8 & 7 & 4 & 3 & 0 & 0 & 3 \\ 10 & 7 & 4 & 3 & 3 & 0 & 0 & 3 \\ 9 & 6 & 3 & 0 & 0 & 0 & 3 & 4 \\ 10 & 7 & 4 & 3 & 3 & 0 & 3 & 6 \\ 11 & 8 & 7 & 6 & 4 & 3 & 4 & 7 \end{bmatrix}$$

These values have to be divided by 3, the value for pixel at distance 1. The first value $14/3 = 4.6667$ is quite close with the exact value 4.4721.

Consider the forward and backward masks

$$h_f(m,n) = \begin{bmatrix} \infty & 11 & \infty & 11 & \infty \\ 11 & 7 & 5 & 7 & 11 \\ \infty & 5 & 0 & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \end{bmatrix} \quad \text{and} \quad h_b(m,n) = \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 0 & 5 & \infty \\ 11 & 7 & 5 & 7 & 11 \\ \infty & 11 & \infty & 11 & \infty \end{bmatrix}$$

With these masks, the first pass result is

$$DS(m,n) = \begin{bmatrix} \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & 0 & 5 & 10 & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & 11 & 0 & 0 & 5 & 10 & \infty & \infty \\ \infty & \infty & \infty & 22 & 11 & 7 & 5 & 0 & 0 & 5 & \infty & \infty \\ \infty & \infty & 22 & 18 & 14 & 11 & 7 & 0 & 0 & 5 & \infty & \infty \\ \infty & \infty & 25 & 21 & 18 & 0 & 0 & 0 & 5 & 7 & \infty & \infty \\ \infty & \infty & 28 & 11 & 7 & 5 & 5 & 0 & 5 & 10 & \infty & \infty \\ \infty & \infty & 18 & 14 & 11 & 10 & 7 & 5 & 7 & 11 & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty \end{bmatrix}$$

and the second pass result is

$$DS(m,n) = \begin{bmatrix} 22 & 18 & 14 & 11 & 7 & 5 & 7 & 11 \\ 21 & 16 & 11 & 7 & 5 & 0 & 5 & 10 \\ 20 & 15 & 10 & 5 & 0 & 0 & 5 & 7 \\ 18 & 14 & 11 & 7 & 5 & 0 & 0 & 5 \\ 16 & 11 & 7 & 5 & 5 & 0 & 0 & 5 \\ 15 & 10 & 5 & 0 & 0 & 0 & 5 & 7 \\ 16 & 11 & 7 & 5 & 5 & 0 & 5 & 10 \\ 18 & 14 & 11 & 10 & 7 & 5 & 7 & 11 \end{bmatrix}$$

Dividing by 5, we get

$$DS(m,n) = \begin{bmatrix} 4.4 & 3.6 & 2.8 & 2.2 & 1.4 & 1.0 & 1.4 & 2.2 \\ 4.2 & 3.2 & 2.2 & 1.4 & 1.0 & 0 & 1.0 & 2.0 \\ 4.0 & 3.0 & 2.0 & 1.0 & 0 & 0 & 1.0 & 1.4 \\ 3.6 & 2.8 & 2.2 & 1.4 & 1.0 & 0 & 0 & 1.0 \\ 3.2 & 2.2 & 1.4 & 1.0 & 1.0 & 0 & 0 & 1.0 \\ 3.0 & 2.0 & 1.0 & 0 & 0 & 0 & 1.0 & 1.4 \\ 3.2 & 2.2 & 1.4 & 1.0 & 1.0 & 0 & 1.0 & 2.0 \\ 3.6 & 2.8 & 2.2 & 2.0 & 1.4 & 1.0 & 1.4 & 2.2 \end{bmatrix}$$

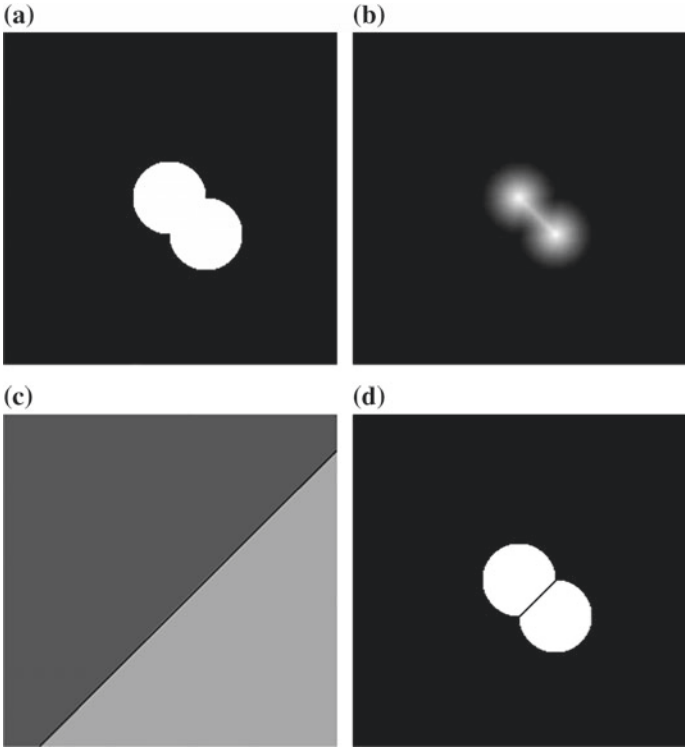


Fig. 10.9 **a** A 256 × 256 8-bit image; **b** distance transform of its complemented binary version; **c** labeled image indicating the two regions and the boundary between them; **d** segmented image

10.4.2 The Watershed Algorithm

Figure 10.9a shows a 256 × 256 8-bit image. The image is composed of two overlapping regions. The binary version of the image is derived first by suitable thresholding. Then, the binary image is complemented. The regions become black, and the background becomes white. The distance between each black pixel to its nearest white pixel is computed, and it is shown in Fig. 10.9b. The distance for each white pixel is zero, since the nearest white pixel is itself. Pixels in the center of the regions have the largest distances. The complement of this image makes the two regions similar to two catchment basins, and their minima indicate their bottom. Then, the watershed algorithm finds equidistant lines between the minima of the regions. In addition, the algorithm assigns label 0 to the boundary lines and other integers to the regions. For the example, Fig. 10.9c shows the labeled image with the two regions and the boundary line in between. The boundary line is superimposed on the input image yielding the segmented image, shown in Fig. 10.9d.

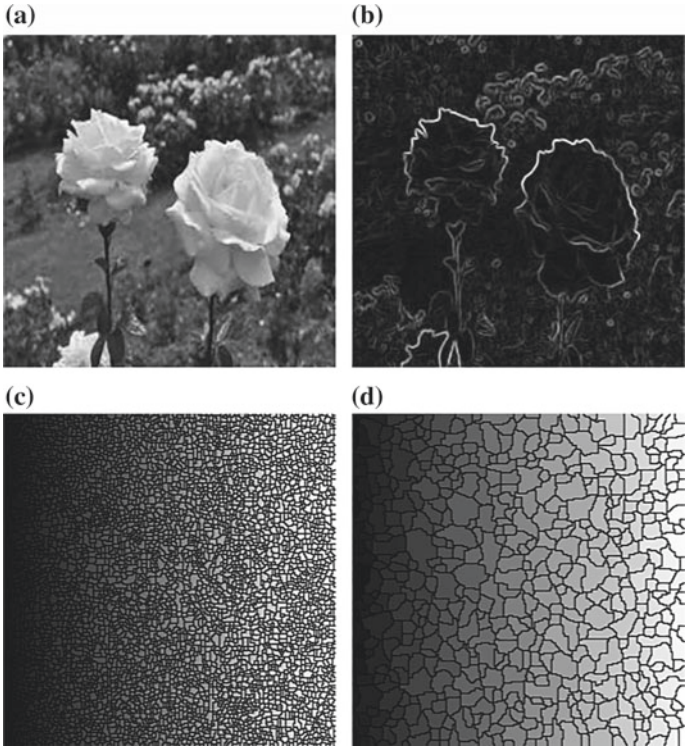


Fig. 10.10 **a** A 256 × 256 8-bit image; **b** its magnitude edge image; **c** watershed lines; **d** watershed lines with some morphological processing

The segmentation of the regions is perfect in the last example, since we used an idealized image. Good segmentation results can be achieved for practical images also, but with some additional processing. Consider the 256 × 256 8-bit image shown in Fig. 10.10a. In addition to the distance measure, the magnitude of the edge map can be used in the watershed algorithm. The gradient magnitude is high along the boundaries of regions and relatively low elsewhere. Either with the distance measure or with the gradient magnitude, due to noise and irrelevant objects, over-segmentation results. Figure 10.10b shows the gradient magnitude image obtained using the Sobel edge operator. Figure 10.10c and d show the watershed lines obtained without and with some morphological processing to smooth the gradient image. Even in the second case, there are too many watershed lines, and it is difficult to make a correspondence with the regions of interest.

Some suitable preprocessing using the characteristics of the regions mitigates the over-segmentation problem. Consider the 256×256 8-bit image shown in Fig. 10.11a. Let us say that the regions of interest are the three white flowers (one partly seen). With the threshold at 96, the binary image, shown in Fig. 10.11b, is obtained. Then, the binary image is subjected to morphological open and close operations, in that

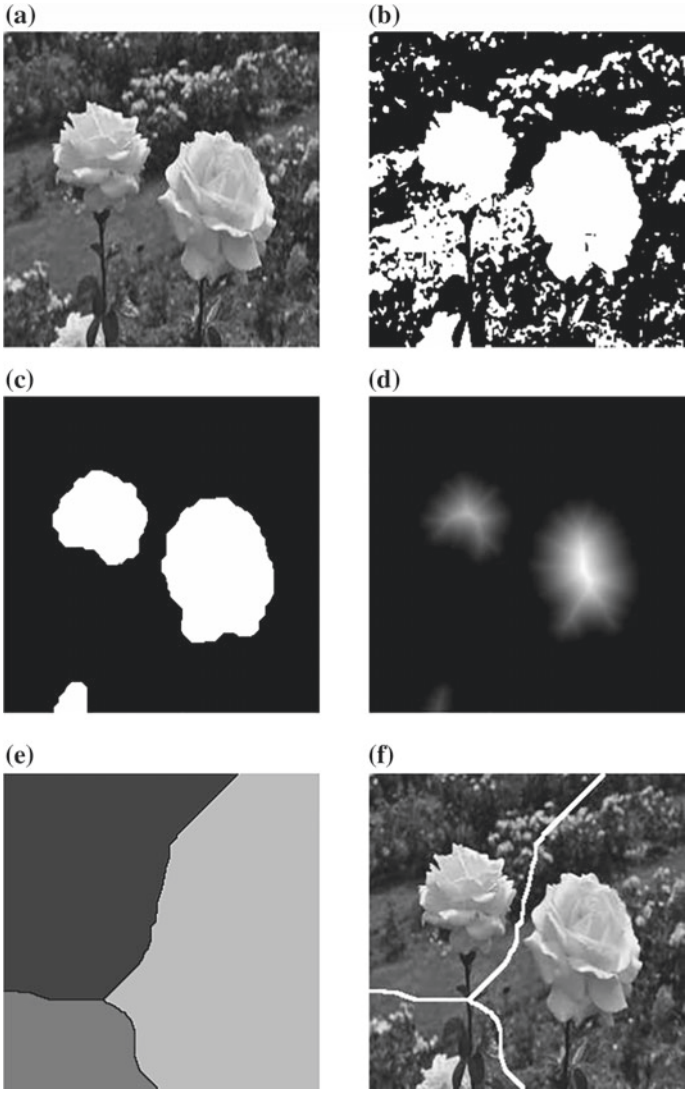


Fig. 10.11 **a** A 256 × 256 8-bit image; **b** its thresholded version; **c** image after morphological processing; **d** its distance transform; **e** labeled image indicating the three regions and the boundary lines between them; **f** segmented image

order, with the same structuring element, which is a disk of radius 9. The three objects are distinctly segmented, as shown in Fig. 10.11c. Figure 10.11d shows the distances. Figure 10.11e shows the labeled image with the three regions and the boundary lines between them. The boundary lines are first dilated (to make it clearly visible) and then superimposed on the input image yielding the segmented image, shown in Fig. 10.11f.

10.5 Summary

- Segmentation of an image is its partition into regions representing objects of interest.
- Segmentation provides a compact and abstract representation of an image that is necessary for the classification of the objects in the image.
- Selection of the method and the set of features that discriminate the regions are critical for effective segmentation.
- Segmentation methods either find the borders of the different regions or group the pixels in the regions. Each method has several variations.
- Edge detection is used to segment an image by finding the borders of the regions using the abrupt change in the intensity of the pixels along the borders.
- Thresholding method segments an image using the different ranges of the gray levels of the different regions.
- Region-growing method of segmentation, starting with a seed pixel, grows the region by grouping the pixels based on similarity of selected features and connectivity.
- Region splitting and merging method subdivides an image into some number of disjoint regions and then split and merge the regions as required to meet the similarity criteria and connectivity.
- In watershed segmentation method, an image is segmented based on the distance of the pixels of a region from other pixels.

Exercises

10.1 Using the averaging method, find the threshold of the 4×4 8-bit image. Let the initial value of the threshold be the average of the gray levels of the image. The iteration stops when the difference between two consecutive threshold values becomes less than 0.5.

$$\begin{bmatrix} 140 & 10 & 5 & 6 \\ 74 & 2 & 7 & 6 \\ 21 & 5 & 6 & 5 \\ 2 & 6 & 5 & 5 \end{bmatrix}$$

10.2 Using the averaging method, find the threshold of the 4×4 8-bit image. Let the initial value of the threshold be the average of the gray levels of the image. The iteration stops when the difference between two consecutive threshold values becomes less than 0.5.

$$\begin{bmatrix} 191 & 102 & 1 & 7 \\ 182 & 45 & 2 & 6 \\ 140 & 10 & 5 & 6 \\ 74 & 2 & 7 & 6 \end{bmatrix}$$

***10.3** Using the averaging method, find the threshold of the 4×4 8-bit image. Let the initial value of the threshold be the average of the gray levels of the image. The iteration stops when the difference between two consecutive threshold values becomes less than 0.5.

$$\begin{bmatrix} 184 & 188 & 72 & 2 \\ 188 & 163 & 22 & 5 \\ 191 & 102 & 1 & 7 \\ 182 & 45 & 2 & 6 \end{bmatrix}$$

10.4 Using Otsu's method, find the threshold of the 4×4 3-bit image. Find the separability index.

$$\begin{bmatrix} 5 & 2 & 6 & 5 \\ 2 & 5 & 6 & 6 \\ 2 & 7 & 6 & 6 \\ 5 & 6 & 5 & 5 \end{bmatrix}$$

10.5 Using Otsu's method, find the threshold of the 4×4 3-bit image. Find the separability index.

$$\begin{bmatrix} 4 & 0 & 2 & 6 \\ 3 & 6 & 5 & 6 \\ 6 & 1 & 7 & 6 \\ 5 & 2 & 6 & 5 \end{bmatrix}$$

***10.6** Using Otsu's method, find the threshold of the 4×4 3-bit image. Find the separability index.

$$\begin{bmatrix} 5 & 6 & 5 & 5 \\ 6 & 5 & 5 & 6 \\ 7 & 6 & 4 & 5 \\ 5 & 5 & 5 & 5 \end{bmatrix}$$

10.7 Consider the 8×8 image. The seed pixel is shown in boldface. Use the 4-connectivity to segment the image so that the region is to be made of pixels with gray levels less than or equal to 176.

$$\begin{bmatrix} 255 & 207 & 73 & 38 & 42 & 43 & 42 & 40 \\ 255 & 255 & 205 & 46 & 43 & 45 & 43 & 42 \\ 255 & 255 & 253 & 84 & 36 & 43 & 46 & 43 \\ 255 & 255 & 255 & 126 & 37 & 45 & 46 & 44 \\ 255 & 255 & 255 & 169 & 35 & 45 & 46 & 45 \\ 255 & 255 & 255 & 222 & 54 & 37 & 42 & 42 \\ 255 & 255 & 255 & 242 & 120 & 28 & 39 & 39 \\ 255 & 255 & 255 & 240 & 212 & 92 & 33 & 42 \end{bmatrix}$$

***10.8** Consider the 8×8 image. The seed pixel is shown in boldface. Use the 4-connectivity to segment the image so that the region is to be made of pixels with gray levels less than or equal to 76.

$$\begin{bmatrix} 172 & 157 & 115 & 62 & 4 & 4 & 4 & 46 \\ 163 & 165 & 118 & 83 & 13 & 6 & 9 & 46 \\ 138 & 185 & 128 & 71 & 90 & 24 & 19 & 30 \\ 121 & 184 & 126 & 83 & 78 & 51 & 50 & 51 \\ 156 & 185 & 136 & 74 & 40 & 42 & 44 & 53 \\ 160 & 175 & 121 & 63 & 77 & 81 & 65 & 71 \\ 178 & 170 & 128 & 88 & 73 & 61 & 59 & 59 \\ 176 & 163 & 133 & 97 & 88 & 35 & 26 & 27 \end{bmatrix}$$

10.9 Consider the 8×8 image. The seed pixel is shown in boldface. Use the 4-connectivity to segment the image so that the region is to be made of pixels with gray levels less than or equal to 56.

$$\begin{bmatrix} 65 & 62 & 65 & 61 & 59 & 57 & 56 & 54 \\ 58 & 62 & 64 & 62 & 58 & 58 & 54 & 52 \\ 71 & 71 & 64 & 63 & 58 & 55 & 54 & 54 \\ 71 & 71 & 70 & 70 & 62 & 57 & 54 & 55 \\ 71 & 70 & 68 & 69 & 66 & 62 & 59 & 58 \\ 69 & 66 & 67 & 67 & 62 & 61 & 60 & 58 \\ 65 & 62 & 63 & 65 & 64 & 61 & 60 & 55 \\ 66 & 61 & 61 & 61 & 62 & 62 & 60 & 57 \end{bmatrix}$$

***10.10** Consider the 8×8 image. Use the 8-connectivity to segment, using the split and merge algorithm, the image so that the region is to be made of pixels with gray levels less than or equal to 45.

$$\begin{bmatrix} 59 & 50 & 40 & 29 & 22 & 20 & 20 & 21 \\ 55 & 48 & 39 & 28 & 22 & 20 & 20 & 20 \\ 53 & 47 & 38 & 28 & 22 & 20 & 20 & 22 \\ 54 & 47 & 38 & 28 & 22 & 20 & 20 & 23 \\ 57 & 49 & 40 & 29 & 23 & 21 & 21 & 22 \\ 62 & 54 & 44 & 32 & 25 & 22 & 22 & 20 \\ 69 & 60 & 49 & 36 & 28 & 23 & 22 & 20 \\ 78 & 68 & 55 & 41 & 31 & 24 & 22 & 22 \end{bmatrix}$$

10.11 Consider the 8×8 image. Use the 8-connectivity to segment, using the split and merge algorithm, the image so that the region is to be made of pixels with gray levels less than or equal to 240.

$$\begin{bmatrix} 248 & 247 & 242 & 238 & 236 & 235 & 235 & 234 \\ 248 & 247 & 243 & 241 & 238 & 235 & 235 & 234 \\ 249 & 248 & 244 & 242 & 239 & 235 & 235 & 235 \\ 249 & 248 & 246 & 244 & 240 & 235 & 235 & 235 \\ 249 & 248 & 247 & 245 & 241 & 236 & 236 & 235 \\ 247 & 247 & 246 & 244 & 241 & 236 & 236 & 236 \\ 244 & 245 & 244 & 243 & 241 & 236 & 236 & 236 \\ 241 & 243 & 242 & 241 & 239 & 236 & 236 & 236 \end{bmatrix}$$

10.12 Consider the 8×8 image. Use the 8-connectivity to segment, using the split and merge algorithm, the image so that the region is to be made of pixels with gray levels less than or equal to 240.

$$\begin{bmatrix} 239 & 240 & 240 & 240 & 241 & 243 & 238 & 231 \\ 239 & 240 & 240 & 240 & 241 & 242 & 239 & 234 \\ 239 & 240 & 240 & 240 & 240 & 240 & 240 & 238 \\ 239 & 240 & 240 & 240 & 239 & 238 & 240 & 241 \\ 239 & 240 & 240 & 240 & 239 & 239 & 242 & 244 \\ 237 & 238 & 239 & 240 & 241 & 245 & 245 & 245 \\ 236 & 238 & 239 & 240 & 241 & 245 & 246 & 245 \\ 236 & 238 & 239 & 240 & 241 & 245 & 246 & 245 \end{bmatrix}$$

10.13 Find the seed points of the regions by clustering and finding the centroids. Initial seeds are $\{2, 2\}$ and $\{2, 3\}$.

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

***10.14** Find the seed points of the regions by clustering and finding the centroids. Initial seeds are {2, 2} and {2, 3}.

$$\begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}$$

10.15 Find the seed points of the regions by clustering and finding the centroids. Initial seeds are {2, 2} and {2, 3}.

$$\begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

10.16 Using the distance transform with masks

$$h_f(m,n) = \begin{bmatrix} \infty & 1 & \infty \\ 1 & 0 & \infty \\ \infty & \infty & \infty \end{bmatrix} \quad \text{and} \quad h_b(m,n) = \begin{bmatrix} \infty & \infty & \infty \\ \infty & 0 & 1 \\ \infty & 1 & \infty \end{bmatrix}$$

find the distance of the pixels of the 8×8 image.

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

10.17 Using the distance transform with masks

$$h_f(m,n) = \begin{bmatrix} 4 & 3 & 4 \\ 3 & 0 & \infty \\ \infty & \infty & \infty \end{bmatrix} \quad \text{and} \quad h_b(m,n) = \begin{bmatrix} \infty & \infty & \infty \\ \infty & 0 & 3 \\ 4 & 3 & 4 \end{bmatrix}$$

find the distance of the pixels of the 8×8 image.

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

***10.18** Using the distance transform with masks

$$h_f(m,n) = \begin{bmatrix} \infty & 11 & \infty & 11 & \infty \\ 11 & 7 & 5 & 7 & 11 \\ \infty & 5 & 0 & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \end{bmatrix} \quad \text{and} \quad h_b(m,n) = \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 0 & 5 & \infty \\ 11 & 7 & 5 & 7 & 11 \\ \infty & 11 & \infty & 11 & \infty \end{bmatrix}$$

find the distance of the pixels of the 8×8 image.

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Chapter 11
Object Description

Abstract The objects of a segmented image are represented by various ways, and their features are extracted to enable object recognition. The boundary of an object is represented by chain code, signature, and Fourier descriptor. The interior of a region is characterized by their geometrical features, moments, and textural features. Typical geometrical features are area, perimeter, compactness, and irregularity. Euler number is a topological descriptor. Moments are a sort of combination of geometrical features. Textural features are derived based on the histogram and co-occurrence matrices. Principal component analysis is based on decomposing an image into its components and leads to data reduction using linear algebra techniques.

Feature is a prominent attribute or distinct characteristic of an object. The chin, mouth, nose, eyes, and forehead are the features of a human face. Both the size and other features of these objects can be used to identify a particular face. Similarly, an image is composed of several objects. Each object is a collection of pixels, and it has to be described. The descriptor is a set of numbers characterizing the salient properties of the object. The descriptor is compared with that of the reference object for object recognition. A descriptor should have some desirable properties. A descriptor should completely characterize an object and, at the same time, it should be concise. A descriptor should be unique. That is, two objects with the same descriptor must have the same shape. Similar objects should have similar descriptors. A descriptor should be invariant with respect to scaling, rotation, and translation. Any prominent attribute or aspect of an object is a feature. Features are used both for segmentation and classification. Features should be evaluated and a figure-of-merit established for each feature. This is done by applying the features for segmentation and classification of artificial images with known features.

A region of an image is characterized by its internal or external features. Internal features are based on the pixels comprising the region. Typical features are area, perimeter, and compactness. External features are related to the boundary of the region.

D. Sundararajan, *Digital Image Processing*, DOI 10.1007/978-981-10-6113-4_11

11.1 Boundary Descriptors

A region is characterized by its boundary, and the form of the boundary is called the shape. A point is on the boundary if there is at the least one of its neighbors is outside the region and the point itself is in the region. One way of finding the boundary of an object is to start at a point on the boundary and keep following the boundary, either clockwise or anticlockwise, using connectivity and other conditions. Four pixels that share an edge (located on the same row or column), but not a corner, with a pixel are its 4-connected neighbors.

11.1.1 Chain Codes

The set of coordinates of all the boundary pixels of a region is its description. However, we are looking for efficient ways to represent the shape. One way is to use a code for each principal direction the trace of the boundary could move. Figure 11.1 shows the encoding scheme for chain coding with 8 directions. Each direction can be coded with 3 bits. If the boundary path moves toward east direction from a pixel to its neighbor, then the path is coded with digit 0. A path along west is coded with digit 4 and so on.

Figure 11.2 shows the chain coding of a boundary with 8 directional codes. Let us start with the top-left pixel on the boundary and traverse in the clockwise direction. The path moves along the east direction and, therefore, that link is coded with 0. Proceeding similarly, the chain code for the region is

$$\{0, 0, 7, 6, 5, 6, 5, 5, 6, 4, 4, 2, 2, 3, 2, 1, 1, 1\}$$

Despite the starting point, we can always end up with the same code for the same boundary by shifting the code (assuming that the digits of the code form an integer) so that its magnitude is the minimum. In the example, the magnitude of the code happens to be minimum. Chain code is invariant with respect to translation only. By

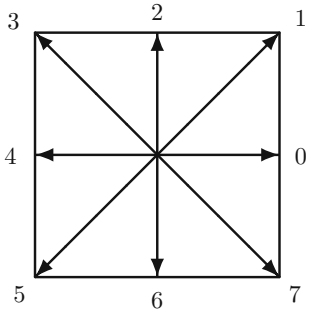


Fig. 11.1 Encoding scheme for 8-directional chain coding

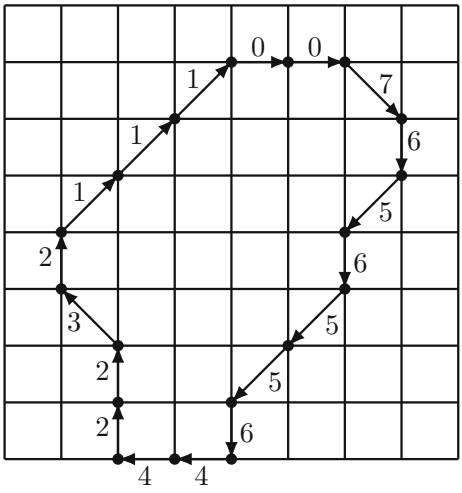


Fig. 11.2 Chain coding of a boundary with 8 directional codes

finding the differences of the adjacent codes, rotational invariance can be obtained. The difference is the number of changes in the direction between the adjacent codes in clockwise or counterclockwise direction. The code for the example, using the counterclockwise direction, after finding the differences is

$$\{7, 0, 7, 7, 7, 1, 7, 0, 1, 6, 0, 6, 0, 1, 7, 7, 0, 0\}$$

The first two codes are $\{0, 0\}$ pointing in the same direction, and, therefore, the second difference code is 0. The next two codes are $\{0, 7\}$. Starting from direction 0 and traversing in the counterclockwise direction, seven changes in the direction are required to get to direction 7. Therefore, the difference code is 7. The first element of the difference code, 7, is computed using the last and first codes $\{1, 0\}$. Rotational dependence is removed in the relative changes. Circularly shifting the code to the right two times, the code with the minimum magnitude is obtained.

$$\{0, 0, 7, 0, 7, 7, 7, 1, 7, 0, 1, 6, 0, 6, 0, 1, 7, 7\}$$

11.1.2 Signatures

A signature is a 1-D representation of a 2-D region by the radial distances of its boundary. The radial distances are computed from the centroid of the region. Then, the signature is plotted, distance versus angle. The signature of a closed boundary is a periodic function. This description is applicable to shapes for which the signature is a single-valued function. The area of a region $x(m, n)$ is given by

$$A = \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} x(m,n)$$

The coordinates of the centroid (center of mass), $(\bar{m}, \bar{n})$, of the region is defined as

$$\bar{m} = \frac{1}{A} \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} m x(m,n) \quad \text{and} \quad \bar{n} = \frac{1}{A} \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} n x(m,n)$$

For example, the centroid of a circle of radius R with the center at the origin is $(0, 0)$. Now, the radial distance at any angle is R and, therefore, the signature of this circle is a constant function $d(\theta) = R$.

Consider the 4×4 binary image $x(m, n)$

$$x(m,n) = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}$$

With the top-left corner the origin $(0, 0)$, the centroid of the boundary is at $(1.5, 2)$. The signature is

θ	0°	34°	63°	117°	146°	180°	−34°	−63°	−117°	−146°
$d(\theta)$	1.5	1.8028	1.1180	1.1180	1.8028	1.5	1.8028	1.1180	1.1180	1.8028

For example, the distance of the bottom-right pixel at coordinates (3,3) is

$$\sqrt{(1.5-3)^2 + (2-3)^2} = 1.8028$$

and the angle is 34°.

Consider the 100×78 binary image and its signature shown, respectively, in Fig. 11.3a and b.

The centroid is (52.6036, 37.0036), shown by a dot and the letter C in Figure (a). From C to the middle of the left side is angle −90°. From C to the middle of the bottom side is angle 0°. It is obvious that the distance from C is minimum at −90°. The distances at some angles are indicated by the symbol $*$ in (b).

11.1.3 Fourier Descriptors

Closed boundaries of an object in an image can be compactly represented using Fourier coefficients. The two coordinates of all the boundary pixels are represented in the transform domain. Starting from a point in the boundary with coordinates (m_0, n_0), followed by

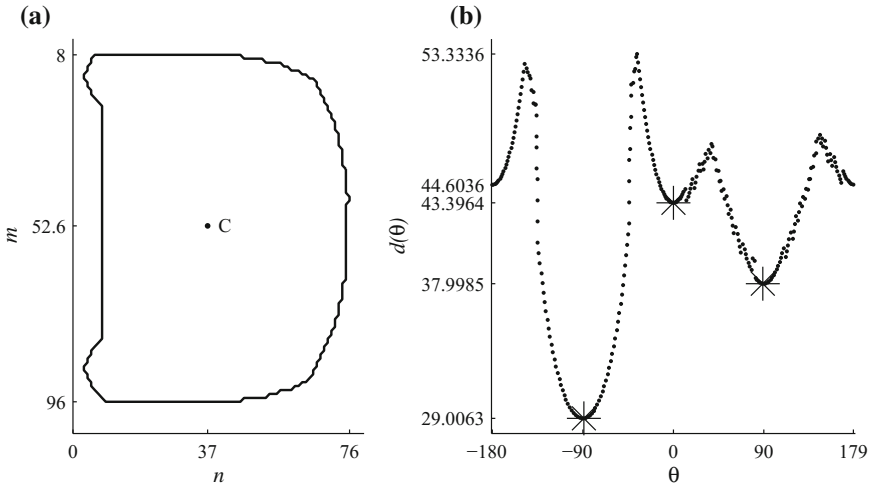


Fig. 11.3 **a** A boundary; **b** its signature

$$(m_l, n_l), \quad l = 1, 2, \ldots N-1$$

can be considered as a 1-D periodic complex data

$$d(l) = (m_l + jn_l), \quad l = 0, 1, \ldots N-1$$

of period N. The first and second coordinates represent, respectively, the real and imaginary parts of the complex data. Then, the DFT of $d(l)$, the set of N 1-D DFT coefficients, is the Fourier descriptor of the boundary with significant advantages. The 1-D N-point DFT of $d(l)$ is defined as

$$D(k) = \sum_{l=0}^{N-1} d(l) e^{-j\frac{2\pi}{N}kl}, \quad k = 0, 1, \ldots, N-1. \tag{11.1}$$

The 1-D IDFT of $D(k)$ gets back the coordinates.

$$d(l) = \frac{1}{N} \sum_{k=0}^{N-1} D(k) e^{j\frac{2\pi}{N}kl}, \quad l = 0, 1, \ldots, N-1. \tag{11.2}$$

Figure 11.4a and b show, respectively, a rectangular boundary and the normalized magnitude of part of its Fourier descriptor obtained using Eq. (11.1). The first 16 coefficients are shown by dots, and the last 16 are shown by crosses. The 512 boundary coordinates are encoded into complex form, and its DFT is computed. As the data is complex, its DFT does not have the conjugate symmetry associated with the DFT of

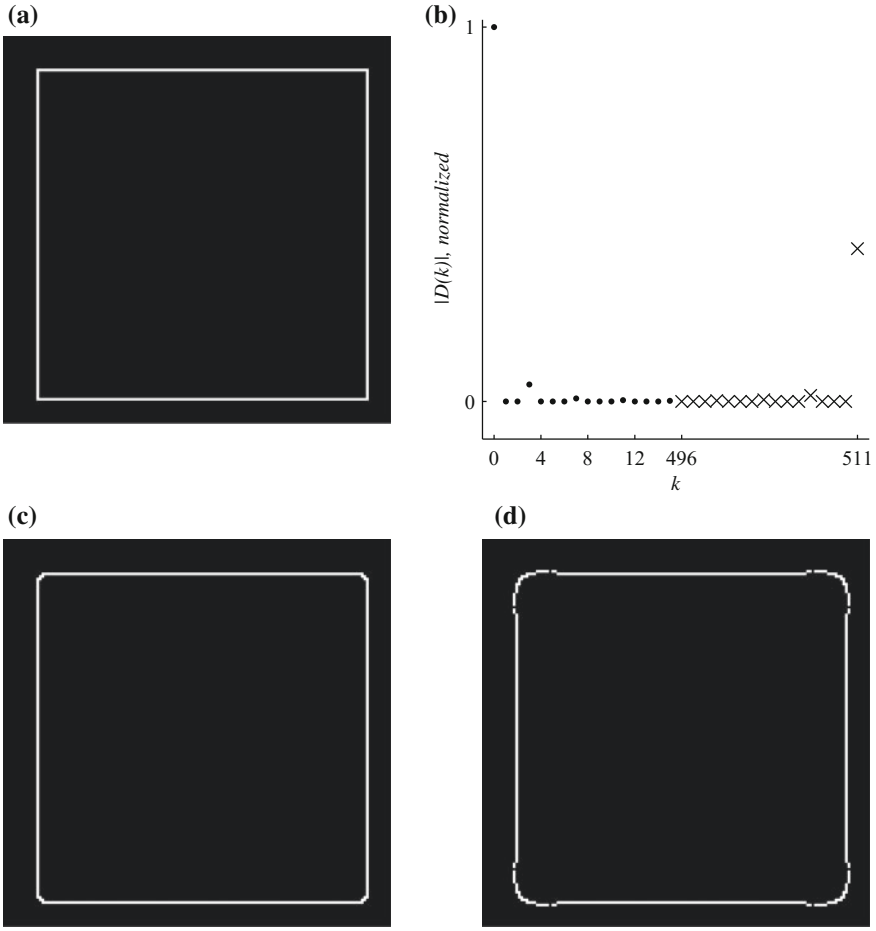


Fig. 11.4 **a** A rectangular boundary; **b** the magnitude of the largest 32 of the 512 DFT coefficients (normalized) of the boundary points; **c** and **d** the reconstructed boundaries using only the largest 48 and 32 DFT coefficients, respectively

real data. Figure 11.4c and d show the reconstructed boundary using only 48 and 32 of the largest of the 512 DFT coefficients, respectively. The boundary is quite close to the original.

Fourier analysis is decomposing an arbitrary waveform into its sinusoidal components. We always remember the well-known square wave reconstruction example. As more and more sinusoidal components are used to reconstruct the square wave, the more closer it becomes to its ideal form. In reconstructing a boundary from its Fourier descriptor, a similar process occurs but with a difference. Real sinusoidal waveforms are related to the complex exponential and its conjugate by the Euler's

formula. The Fourier descriptor does not have the complex-conjugate symmetry. A DFT coefficient is the coefficient of a complex exponential

$$e^{j\frac{2\pi}{N}kl}$$

in the time domain. The samples of a complex exponential occur on the unit circle. Therefore, the plot of the real and imaginary parts of a complex exponential is a circle. Its radius is proportional to the magnitude of its coefficient. In reconstructing a boundary from its Fourier descriptor, we use circles to approximate the boundary.

Figure 11.5a shows the circles corresponding to two of the largest DFT coefficients other than the DC. The boundary corresponding to the combined effect of the two DFT coefficients is also shown, which is an approximation of the actual boundary. The smaller circle corresponds to a higher frequency and appropriately adds negative and positive values to the larger circle, thereby making it closer to the actual boundary. Now, the addition of the DC component of the DFT, which represents the average of the two coordinates, fixes the center of the boundary, as shown in Fig. 11.5b. With just the DC and two other DFT coefficients, we are able to get a good approximation of the boundary. Therefore, a major advantage is that the shape can be adequately described by much fewer than the N coefficients. As noise is generally characterized by high frequencies, this leads to reduce the noise affecting the boundary points. Another advantage is that the 2-D shape is described by 1-D data.

A descriptor should be as insensitive as possible for scaling, translation, and rotation. Fourier descriptors of a boundary and its modified version are related due to the properties of the Fourier transform. Consider the 4×4 binary image $x(m, n)$ and its shifted version $x(m + 1, n + 1)$

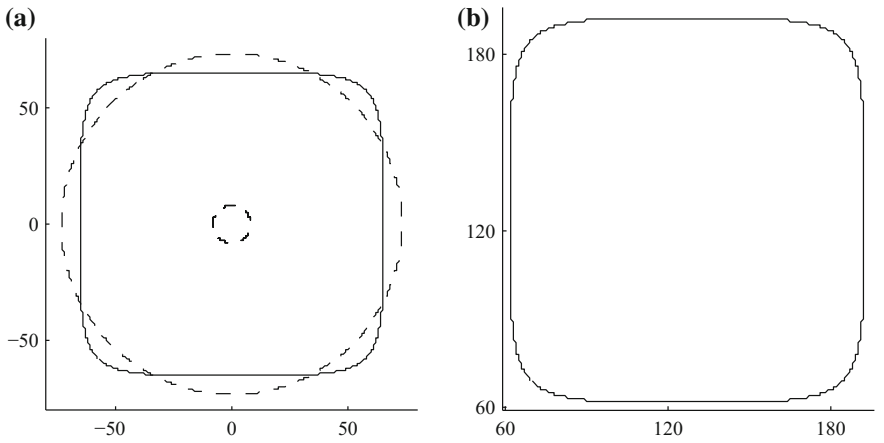


Fig. 11.5 **a** Two circles corresponding to two DFT coefficients and the boundary corresponding to their combined effect; **b** the reconstructed boundary using the DC and two other DFT coefficients only

$$x(m,n) = \begin{bmatrix} 0\,0\,0\,0 \\ 0\,1\,1\,1 \\ 0\,1\,0\,1 \\ 0\,1\,1\,1 \end{bmatrix} \quad x(m+1,n+1) = \begin{bmatrix} 1\,1\,1\,0 \\ 1\,0\,1\,0 \\ 1\,1\,1\,0 \\ 0\,0\,0\,0 \end{bmatrix}$$

The complex data formed from the boundary coordinates of $x(m,n)$ is

$$b(l) = \{1+j1, 2+j1, 3+j1, 3+j2, 3+j3, 2+j3, 1+j3, 1+j2\}$$

The DFT of $b(l)$ is

$$B(k) = \{16+j16, -6.8284-j6.8284, 0, 0, 0, -1.1716-j1.1716, 0, 0\}$$

The complex data formed from the boundary coordinates of $x(m+1,n+1)$ is

$$b(l) = \{0+j0, 1+j0, 2+j0, 2+j1, 2+j2, 1+j2, 0+j2, 0+j1\}$$

The DFT of $b(l)$ is

$$B(k) = \{8+j8, -6.8284-j6.8284, 0, 0, 0, -1.1716-j1.1716, 0, 0\}$$

Only, the DC components of the two DFTs differ and that difference indicates translation.

The complex data formed from the coordinates of the rotated boundary of $x(m,n)$, for example, is

$$c(l) = e^{j\frac{\pi}{4}}\{1+j1, 2+j1, 3+j1, 3+j2, 3+j3, 2+j3, 1+j3, 1+j2\}$$

The DFT of $c(l)$ is

$$C(k) = e^{j\frac{\pi}{4}}\{16+j16, -6.8284-j6.8284, 0, 0, 0, -1.1716-j1.1716, 0, 0\}$$

The starting-point shifted complex data formed from the coordinates of the boundary of $c(l)$, for example, is $c(l-1)$. The DFT of $c(l-1)$ is

$$e^{-jk\frac{2\pi}{8}}C(k) = e^{-jk\frac{2\pi}{8}}e^{j\frac{\pi}{4}}\{16+j16, -6.8284-j6.8284, 0, 0, 0, -1.1716-j1.1716, 0, 0\}$$

The scaled complex data formed from the coordinates of the boundary of $x(m,n)$, for example, is $Zb(l)$ and its transform is $ZB(k)$, due to the linearity property of the Fourier transform.

11.2 Regional Descriptors

The shape of a region has been identified and represented by a chain code, signature, or Fourier descriptor. Now, the region has to be described. Features are the characterizing attributes of the image and its objects. A feature vector is created for each object. Features are important in image segmentation and classification. Some features come from visual appearance of an image and other features are derived by operations such as taking the transform. Derived features include histograms and spectra of images. Certain patterns of gray-level values and the intensity of regions are visual features. There are several types of features. It is desirable that features have the invariance property with respect to scaling, translation, and rotation. This enables machine vision systems to identify the scaled, translated, and rotated versions of the same object. Further, the features should be robust despite the conditions affecting the quality of the image formation such as noise, poor spatial resolution, improper lighting, and other distortions.

11.2.1 Geometrical Features

Area The area of a connected region $x(m, n)$ of a binary image, measured in pixels, is defined as

$$A = \sum_m \sum_n x(m, n)$$

It is the number of pixels with value 1 in the region. It is invariant to rotation, except for a small error due to interpolation involved in rotation. Obviously, area changes with scaling.

Perimeter Let the coordinates of the perimeter of a region is given by $x(k)$ and $y(k)$. Then, the perimeter of the region is defined by

$$P = \sum_k \sqrt{(x(k) - x(k-1))^2 + (y(k) - y(k-1))^2}$$

It is the distance around the boundary of the region. As the pixels are located on a square grid, the terms in the summation are equal to 1 or $\sqrt{2}$ only, depending on the neighboring pixels located either on an axis or on a diagonal.

Compactness Compactness is defined, in terms of the perimeter and area, as

$$C = \frac{4\pi A}{P^2} = \frac{A}{P^2/(4\pi)}$$

For a circular region, which has the highest compactness, with radius r,

$$C = (4\pi(\pi r^2))/((2\pi r)(2\pi r)) = 1 \tag{11.3}$$

For a square, $C = \pi/4$. It is a measure of the area enclosing ability of the shape of the region. It is the ratio of the area of the region and the area of the circle with the same perimeter as that of the region. Let the object be a unit square. Its area is 1 and perimeter is 4. The radius of a circle with the same perimeter is $(4/(2\pi))$, and its area is $4/\pi$. The area of the circle is greater than that of the square. Two different shapes can have the same compactness. Therefore, compactness with other measures should be used for shape discrimination.

Irregularity Irregularity of a region is defined by

$$I = \frac{\pi \max_k (x(k) - \bar{x})^2 + (y(k) - \bar{y})^2)}{A}$$

where $(\bar{x}, \bar{y})$ are the averages of the coordinates of the region. This is a measure of the density of the region. The numerator defines the area of the smallest circle enclosing the region. For circular shapes, I is unity. For a square, it is $I = 0.5\pi$. There are also other geometric features.

Consider the 256×256 binary image, shown in Fig. 11.6a. The image is normally segmented and in binary form for feature extraction. That part involves segmentation and morphological operations. The area of the four objects in the image is

$$\{2460\text{D}, 1279\text{ I}, 1621\text{ S}, 1537\text{ C}\}$$

measured in number of pixels. For example, the width and height of the letter I are, respectively, 16 and 75. The area is approximately $(75)(16) = 1200$. All these values

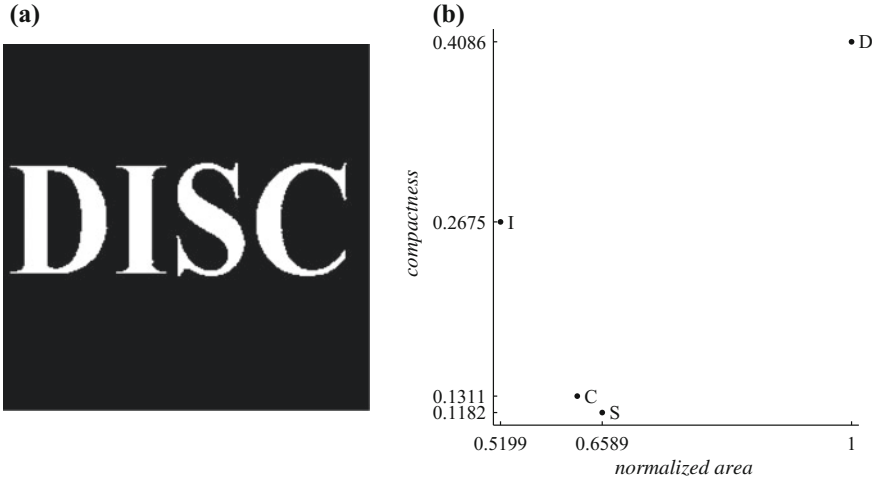


Fig. 11.6 **a** A 256×256 binary image; **b** its feature space profile (normalized area and compactness)

are divided by the largest area (2460 of D) to get the normalized area plotted. The perimeters of the four objects in the image are

$$\{275.0538\text{ D}, 245.1127\text{ I}, 415.1615\text{ S}, 383.8478\text{ C}\}$$

For example, the perimeter of the letter I is approximately $2(75 + 16) = 182$. The compactness is computed using Eq. (11.3). The compactness values of the four objects in the image are

$$\{0.4086\text{ D}, 0.2675\text{ I}, 0.1182\text{ S}, 0.1311\text{ C}\}$$

From the feature profile, shown in Fig. 11.6b, letters I and D can be easily identified. The other two letters are located close to each other in the profile, although still distinguishable. For large number of objects, more features are required for recognition. The accuracy of the features itself depends on good segmentation.

11.2.1.1 Euler Number

A topological descriptor of an image $x(m, n)$ is that which remains the same for all its versions of continuous one-to-one transformations (rubber-sheet distortions). This descriptor is not affected by rotation or stretching. The Euler number E is defined as the difference between the number of connected components and the number of holes. For the image in Fig. 11.6a, $E = 4 - 1 = 3$, since there are four objects and one hole.

11.2.2 Moments

Properties of images considering their behavior for very large and very small values of their independent variables are useful. Moments are global descriptors, similar to the Fourier descriptors, and provide the advantages of invariance, compactness, and reducing the effects of noise. They are a sort of combination of the geometrical features. The $(p + q)$th order moment of a $N \times N$ region $x(k, l)$ is defined as

$$m_{pq} = \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} k^p l^q x(k, l), \quad p = 0, 1, 2, \ldots, \quad q = 0, 1, 2, \ldots \tag{11.4}$$

Eq. (11.4) for various values of p and q are called as moment conditions, because of the similarity in form to moments encountered in mechanics. Some specific moments are

$$m_{00} = \sum_{k=0}^{N-1}\sum_{l=0}^{N-1} x(k,l), \quad m_{10} = \sum_{k=0}^{N-1}\sum_{l=0}^{N-1} kx(k,l), \quad m_{01} = \sum_{k=0}^{N-1}\sum_{l=0}^{N-1} lx(k,l)$$

The zero-order moment is the area. The other two first-order moments are this area multiplied by their distances of their center of gravity from the origin. The coordinates of the centroid (center of mass) of the region is defined as

$$\bar{k} = \frac{m_{10}}{m_{00}} \quad \text{and} \quad \bar{l} = \frac{m_{01}}{m_{00}}$$

The central moments, which are translation-invariant since centroids are part of their definition, are defined as

$$\mu_{pq} = \sum_{k=0}^{N-1}\sum_{l=0}^{N-1}(k-\bar{k})^p(l-\bar{l})^q x(k,l), \quad p = 0,1,2,\ldots, \quad q = 0,1,2,\ldots \tag{11.5}$$

The normalized central moments, which are invariant to translation, scaling, and rotation, are defined as

$$\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^{\gamma}}, \quad \gamma = \frac{(p+q)}{2} + 1, \quad (p+q) = 2,3,\ldots \tag{11.6}$$

The first seven normalized central moments are defined as

$$\begin{aligned}
\phi_1 &= \eta_{20} + \eta_{02} \\
\phi_2 &= (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2 \\
\phi_3 &= (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2 \\
\phi_4 &= (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2 \\
\phi_5 &= (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})((\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2) \\
&\quad + (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})(3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2) \\
\phi_6 &= (\eta_{20} - \eta_{02})((\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2) + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03}) \\
\phi_7 &= (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})((\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2) \\
&\quad + (3\eta_{12} - \eta_{30})(\eta_{21} + \eta_{03})(3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2)
\end{aligned}$$

Consider the 4×4 binary image with one region. Let us compute the first two normalized central moments ϕ_1 and ϕ_2.

$$x(k,l) = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

The coordinates of the top-left corner are (0, 0), those of the bottom-left corner are (3, 0), those of the top-right corner are (0, 3) and those of the bottom-right corner are (3, 3).

$$\{m_{00} = 6, \quad m_{10} = 6, \quad m_{01} = 11, \quad \bar{k} = 1, \quad \bar{l} = 1.8333\}$$

$$\mu_{11} = (-1)(-0.8333) + (-1)(0.1667) + (1)(0.1667) + (1)(1.1667) = 2,$$
$$\mu_{20} = 4, \quad \mu_{02} = 2.8333$$

$$\eta_{11} = 0.0556, \quad \eta_{20} = 0.1111, \quad \eta_{02} = 0.0787, \quad \phi_1 = 0.1898, \quad \phi_2 = 0.0134$$

Figure 11.7a shows a segmented binary image with four objects. There are some small holes in the objects. If we starting labeling, we may end up with more than four objects. Morphological close operation is used to get the image in Fig. 11.7b. Morphological operations are usually required in this stage to correct imperfect segmentation. The four components in the image are labeled and separated, as shown in Fig. 11.7c–f. The first two moments, ϕ_1 and ϕ_1, of the four objects are listed in Table 11.1. The three components, except the one at the bottom left, are translated, scaled, and rotated versions of the same object. The moments of these components in the first, third, and fourth columns are about the same. The moments of the other object, which is about the same size as the one at the top left, are totally different. Therefore, the three components are the images of the same object while the fourth one is different. For the images in Fig. 11.7b–f, the Euler numbers are $E = 4 - 6 = -2$, $E = 1 - 2 = -1$, $E = 1 - 0 = 1$, $E = 1 - 2 = -1$ and $E = 1 - 2 = -1$, respectively.

11.2.3 Textural Features

Texture is a pattern resembling a mosaic, made by a physical composition of an object using constituents of various sizes and shapes. Statistical features, taken over the whole image or in its neighborhoods, are used to characterize a texture.

11.2.3.1 Histogram-Based Features

Histogram is a representation of the distribution of the gray values of an image. In addition to its other uses in image analysis, useful features can also be derived from it. Let the L gray levels in the $N \times N$ image be

$$u = 0, 1, 2, \ldots L - 1$$

Let the number of occurrences of the gray levels be n_u. Then, the histogram of the image is given as

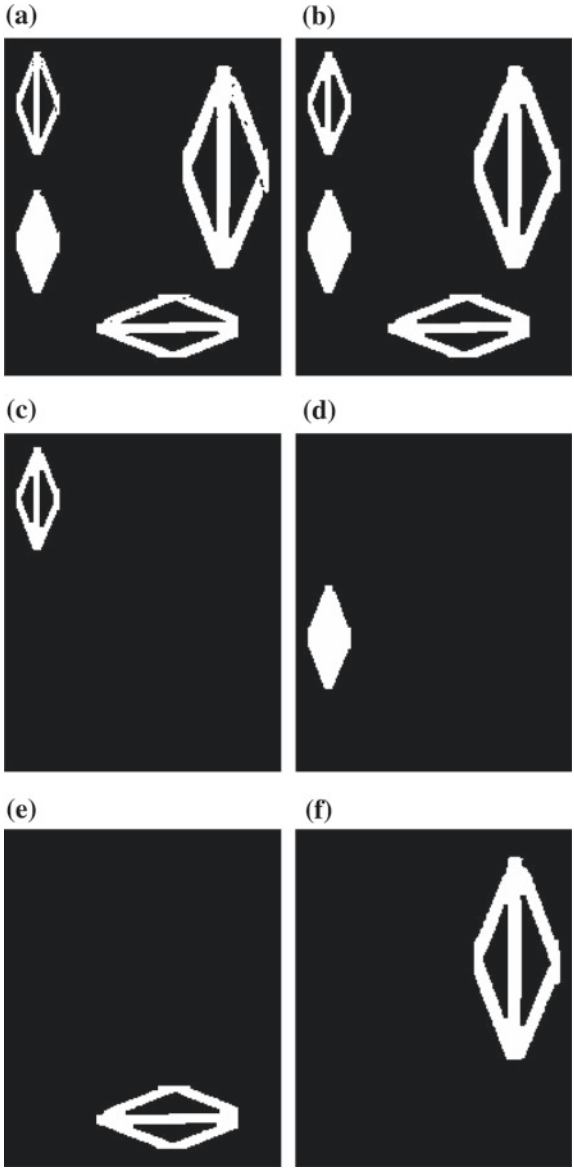


Fig. 11.7 **a** A 256 × 256 binary image; **b** the image after image close operation; **c–f** images of the four components in (**b**) separated

Table 11.1 First two moments of the four objects in Fig. 11.7b

ϕ_1	0.4519	0.2320	0.4554	0.4510
ϕ_2	0.1149	0.0269	0.1082	0.1116

$$his(u) = n_u$$

The normalized histogram of the image is

$$hisn(u) = p(u) = \frac{n_u}{N^2}$$

Consider the 4×4 8-bit image shown in Table 11.2.

The histogram of this image and its normalized version (with a precision of 2 digits) are shown in Table 11.3, which also is the probability $p(u)$ of the occurrences of the gray levels. The nonzero entries only are shown.

Mean The mean m, which is the average intensity, is given by

$$m = \sum_{u=0}^{L-1} up(u)$$

For the example, the mean is 35.3750 with $L = 256$.

Standard deviation The standard deviation σ, which is the average contrast and the 2nd moment, is defined as

$$\sigma = \sqrt{\sum_{u=0}^{L-1} (u-m)^2 p(u)}$$

Table 11.2 A 4×4 8-bit image

0	19	20	22
53	4	23	25
116	16	17	24
110	90	4	23

Table 11.3 The histogram and its normalized version of the image in Table 11.2

g_lev	0	4	16	17	19	20	22	23	24	25	53	90	110	116
his	1	2	1	1	1	1	1	2	1	1	1	1	1	1
hisn	0.06	0.13	0.06	0.06	0.06	0.06	0.06	0.13	0.06	0.06	0.06	0.06	0.06	0.06

For the example, $\sigma = 35.8153$. The square of the standard deviation is the variance.

Smoothness A measure of the smoothness of the texture is defined as

$$S = 1 - \frac{1}{1 + \sigma_n^2}$$

where σ_n^2 is a normalized version of the variance and

$$\sigma_n^2 = \frac{\sigma^2}{(L-1)^2}$$

For the example image,

$$S = 1 - \frac{1}{1 + (35.8153^2/255^2)} = 1 - \frac{1}{1.0197} = 0.0193$$

For regions with constant intensity, $S = 0$ and it increases with increasing value of σ toward the limit 1.

Skew The skew, which indicates the asymmetry of the histogram about the mean and the third moment, is defined as

$$Sk = \sum_{u=0}^{L-1} (u - m)^3 p(u)$$

Sk is zero for a symmetric histogram. It is positive for a right skew (spreads to the right) and negative for a left skew (spreads to the left). Sk is also normalized in the same way and the normalized value is 0.9341 (positive skew) for the example.

Uniformity This measure, uniformity of energy, is given by

$$U = \sum_{u=0}^{L-1} p^2(u)$$

U is maximum when all the intensity levels are the same, and it has a lower value otherwise. For the example image, $U = 0.0781$.

Entropy The entropy, which is measure of randomness, is given by

$$E = -\sum_{u=0}^{L-1} p(u) \log_2(p(u))$$

A lower value indicates a higher redundancy in the image data and should give a high compression ratio, when compressed. For the example image, $E = 3.75$.

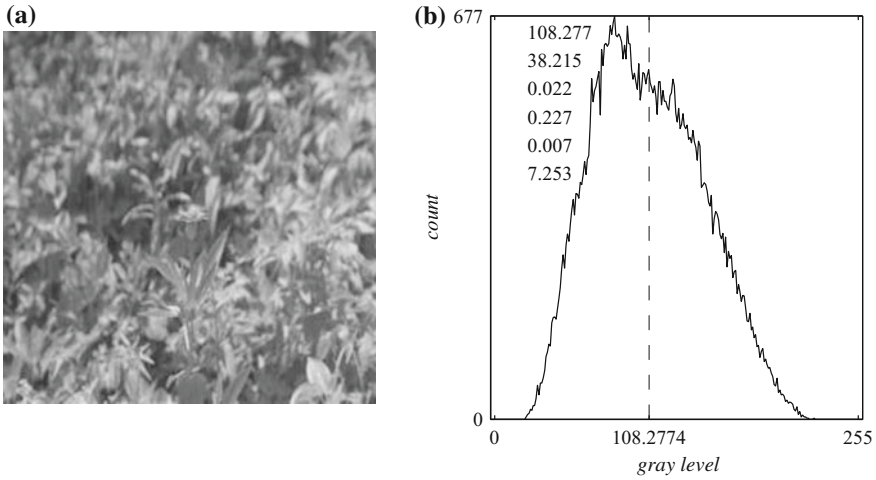


Fig. 11.8 **a** A 256 × 256 8-bit image; **b** its histogram with texture measures

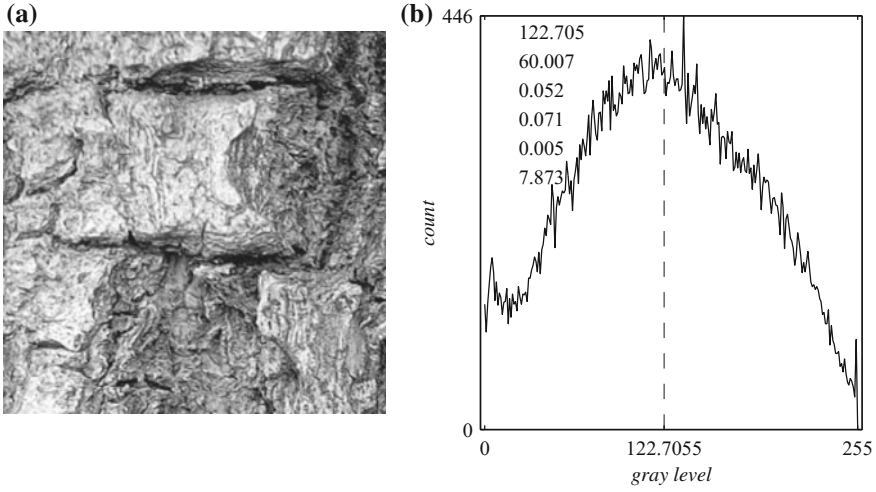


Fig. 11.9 **a** A 256 × 256 8-bit image; **b** its histogram with texture measures

Figures 11.8, 11.9, 11.10, and 11.11 show some images in the a part and the corresponding histograms with texture measures in the b part.

The six measures listed in each (b) part are mean, standard deviation, smoothness, skew, uniformity, and entropy in that order.

The entropy values of the images are {7.253, 7.873, 7.680, 7.843}. Obviously, the variation of the intensity is least random in the first image. Therefore, the contrast {38.215, 60.007, 51.867, 61.007} is also the least. With the smoothness measure {0.022, 0.052, 0.040, 0.054}, it is also the smoothest. It is more uniform with that

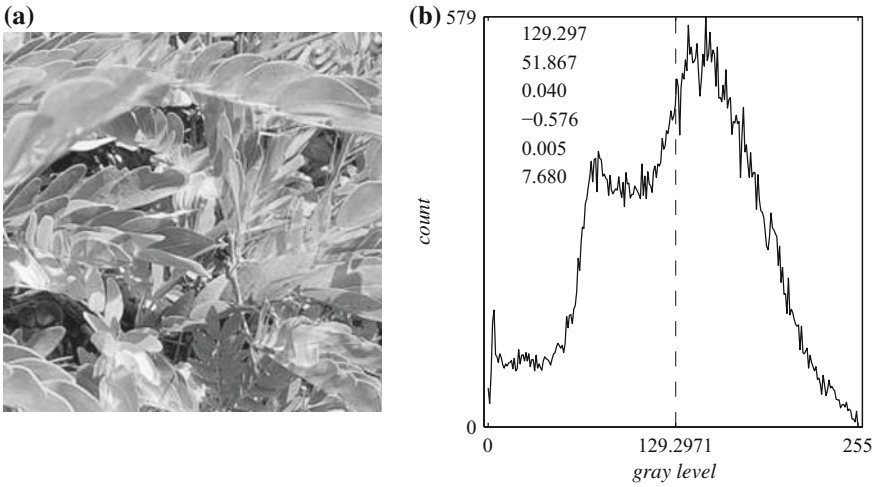


Fig. 11.10 **a** A 256 × 256 8-bit image; **b** its histogram with texture measures

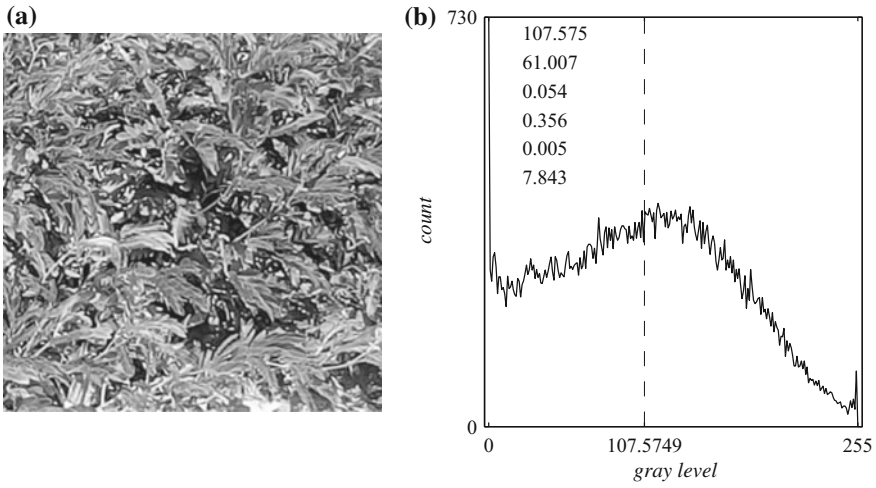


Fig. 11.11 **a** A 256 × 256 8-bit image; **b** its histogram with texture measures

measure {0.007, 0.005, 0.005, 0.005}. With respect to skewness, the third image has a negative skewness −0.576 indicating that its histogram has a longer left tail and a much shorter right tail about the mean. The two sides are demarcated by a dashed line. The other three images have positive skewness.

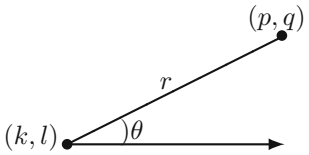


Fig. 11.12 Relationship between the locations of a pair of pixels

11.2.3.2 Co-occurrence Matrix-Based Features

A pair of pixels, with gray levels a and b, occurring with the same spatial relationship is co-occurrence. Due to the repetition of patterns in texture, co-occurrence is an important factor that should be taken in to account. Co-occurrence matrices carry information of the spatial relationships between pixels. Let the number of gray levels in $x(m, n)$ be L, $\{0, 1, \ldots, L-1\}$. A co-occurrence matrix $g(m, n)$ is a $L \times L$ matrix in which each element $g(m, n)$ represents the number of occurrences of a pair of pixels with intensities I_m and I_n, in a given spatial relationship in the image $x(m, n)$. It is the joint probability distribution of pairs of pixels. Let there be two pixels $x(k, l)$ and $x(p, q)$ at coordinates (k, l) and (p, q) with pixel values I_m and I_n, respectively, in the image $x(m, n)$. Further, let the radial distance between them be r and the angle between the line joining them and the horizontal axis be θ radians, as shown in Fig. 11.12. Due to discretization, only limited values of r and θ are possible. The probability $p(I_m, I_n)$ of the co-occurrences of the gray levels I_m and I_n is defined as

$$p(I_m, I_n) = \frac{n(I_m, I_n)}{M} = \frac{g(m, n)}{M}$$

where $n(I_m, I_n)$ is the number of occurrences with $x(k, l) = I_m$ and $x(p, q) = I_n$ and M is the total number of occurrences in $g(m, n)$, the co-occurrence matrix. If the pixel pairs are highly correlated, then the entries will be densely populated along the diagonal of the matrix. The finer the texture, the more uniform is the distribution of the values in the co-occurrence matrix. Coarse texture skews toward the diagonal.

Consider the 8×8 8-bit image $x(m, n)$

$$x(m, n) = \begin{bmatrix} 52 & 71 & 72 & 74 & 64 & 55 & 43 & 74 \\ 105 & 56 & 75 & 77 & 64 & 60 & 53 & 78 \\ 168 & 68 & 69 & 76 & 69 & 62 & 58 & 71 \\ 162 & 142 & 56 & 75 & 73 & 64 & 60 & 53 \\ 162 & 180 & 89 & 67 & 79 & 68 & 63 & 30 \\ 186 & 175 & 156 & 61 & 78 & 72 & 63 & 53 \\ 210 & 171 & 192 & 87 & 67 & 77 & 67 & 59 \\ 250 & 158 & 188 & 140 & 59 & 77 & 69 & 61 \end{bmatrix}$$

Since the number of gray levels is 256, the size of the co-occurrence matrix has to be 256×256. Using different spatial relationships, a series of matrices are used to analyze the texture in a single image. Therefore, in order to reduce the computational

complexity and the storage requirements, the intensity range of the image is usually quantized. Let us divide each pixel value in $x(m, n)$ by 32 and truncate the result. The intensity quantized image $x_q(m, n)$ is

$$x_q(m,n) = \begin{bmatrix} 1 & 2 & 2 & 2 & 2 & 1 & 1 & 2 \\ 3 & 1 & 2 & 2 & 2 & 1 & 1 & 2 \\ 5 & 2 & 2 & 2 & 2 & 1 & 1 & 2 \\ 5 & 4 & 1 & 2 & 2 & 2 & 1 & 1 \\ 5 & 5 & 2 & 2 & 2 & 2 & 1 & 0 \\ 5 & 5 & 4 & 1 & 2 & 2 & 1 & 1 \\ 6 & 5 & 6 & 2 & 2 & 2 & 2 & 1 \\ 7 & 4 & 5 & 4 & 1 & 2 & 2 & 1 \end{bmatrix}$$

Now, the co-occurrence matrix is of size 8×8, since the number of gray levels is limited to eight. Let the spatial relationship of a pair of pixels be $x(m, n)$ and $x(m, n + 1)$. That is, a pixel and its immediate right neighbor form the pair. With this spatial relationship and $x_q(m, n)$, we get the co-occurrence matrix $g(m, n)$ as

$$g(m,n) = \begin{array}{c} \\ \\ 0 \\ 1 \\ 2 \\ m\ 3 \\ \downarrow\ 4 \\ 5 \\ 6 \\ 7 \end{array} \begin{array}{c} \quad n \rightarrow \\ \begin{array}{cccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 5 & 8 & 0 & 0 & 0 & 0 & 0 \\ 0 & 8 & 18 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 2 & 0 & 3 & 2 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

In this matrix, each element is the number of occurrences of a left pixel with gray level m and a right pixel with gray level n in $x_q(m, n)$. Since there is no left pixel with value 0, the first row entries are all zeros. Since there is no right pixel with value seven, the last column entries are all zeros. Since there is only one occurrence of the pair of gray levels $(1, 0)$, ($x_q(4, 6) = 1$ and $x_q(4, 7) = 0$), $g(1, 0) = 1$. The number of occurrences of the pixel pair $(2, 2)$ is the highest at 18. Now, let the spatial relationship be $x(m, n)$ and $x(m+1, n+1)$. That is, a pixel and its immediate bottom-right (diagonal) neighbor form the pair. With this spatial relationship and $x_q(m, n)$, we get the co-occurrence matrix $g(m, n)$ as

$$g(m,n) = \begin{bmatrix} 0\,0 & 0\,0\,0\,0\,0\,0 \\ 1\,7 & 5\,0\,0\,0\,0\,0 \\ 0\,8 & 16\,0\,0\,0\,0\,0 \\ 0\,0 & 1\,0\,0\,0\,0\,0 \\ 0\,0 & 2\,0\,0\,0\,0\,0 \\ 0\,0 & 0\,0\,2\,4\,1\,0 \\ 0\,0 & 0\,0\,2\,0\,0\,0 \\ 0\,0 & 0\,0\,0\,0\,0\,0 \end{bmatrix}$$

Since there are seven occurrences of the pixel pair (1, 1) in the given diagonal spatial relationship, $g(1,1) = 7$. As many co-occurrence matrices as required have to be generated.

The joint probability matrix $p(m,n)$ is defined as

$$p(m,n) = \frac{g(m,n)}{M}, \quad M = \sum_m \sum_n g(m,n)$$

For the example, using the second $g(m,n)$ and with $M = 49$, we get

$$p(m,n) = \begin{bmatrix} 0 & 0 & 0\,0 & 0 & 0 & 0\,0 \\ 0.0204 & 0.1429 & 0.1020\,0 & 0 & 0 & 0\,0 \\ 0 & 0.1633 & 0.3265\,0 & 0 & 0 & 0\,0 \\ 0 & 0 & 0.0204\,0 & 0 & 0 & 0\,0 \\ 0 & 0 & 0.0408\,0 & 0 & 0 & 0\,0 \\ 0 & 0 & 0\,0\,0.0408 & 0.0816 & 0.0204 & 0 \\ 0 & 0 & 0\,0\,0.0408 & 0 & 0 & 0\,0 \\ 0 & 0 & 0\,0 & 0 & 0 & 0\,0 \end{bmatrix}$$

Some of the features are based on the energy distribution in this matrix.

Maximum probability
It is in the range 0 to 1 and indicates the maximum value of $p(m,n)$. For the example, it is $p(2,2) = 0.3265$.

Entropy

$$E = -\sum_{m=0}^{N-1} \sum_{n=0}^{N-1} p(m,n) \log_2 p(m,n)$$

For a $p(m,n)$ with all zero entries, $E = 0$. For a $p(m,n)$ with all entries equal, $E = 2\log_2 N$. For the example, $E = 2.8951$ with $N = 8$.

Contrast
The contrast in intensity of a pixel and its neighbor over the image is given by this measure. The range of values for C is from 0 to $(N-1)^2$.

$$C = \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} (m-n)^2 p(m,n)$$

For the example, $C = 0.6939$.

Energy (Uniformity)
This measure is an indicator of the energy. Its range is from 0 to 1.

$$U = \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} p^2(m,n)$$

For the example, $U = 0.1770$.
Homogeneity

$$H = \sum_{m=0}^{N-1} \sum_{N=0}^{N-1} \frac{p(m,n)}{1+|(m-n)|}$$

This measure indicates the closeness of the distribution of the values in the co-occurrence matrix to its diagonal and it is in the range 0 to 1. For the example, $H = 0.7619$.

Correlation

$$R = \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} \frac{(m-\bar{m})(n-\bar{n})p(m,n)}{\sigma_m \sigma_n}, \quad \sigma_m \neq 0, \sigma_n \neq 0$$

where

$$\bar{m} = \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} m p(m,n), \quad \bar{n} = \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} n p(m,n)$$

$$\sigma_m^2 = \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} (m-\bar{m})^2 p(m,n), \quad \sigma_n^2 = \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} (n-\bar{n})^2 p(m,n)$$

This measure, with range -1 to 1, indicates the similarity of a pixel to its neighbor over the entire image. For the example, $R = 0.8508$.

Figures 11.13, 11.14, 11.15, and 11.16 show, respectively, the corresponding co-occurrence matrix of the images in Figs. 11.8, 11.9, 11.10, and 11.11, the mesh plots in (a) parts and the images in (b) parts. A pixel and its immediate right neighbor form the pair. In Fig. 11.13, the number of co-occurrences is very high in the neighborhood of the main diagonal, indicating that a wide variation in the intensity levels but not many large jumps between adjacent pairs of pixels. In Fig. 11.14, the contrast is high and, therefore, there are less number of co-occurrences. Due to white patches in the image, the number of co-occurrences is only at high intensity values. In Fig. 11.15, the number of co-occurrences is moderately high in the middle and high at the end

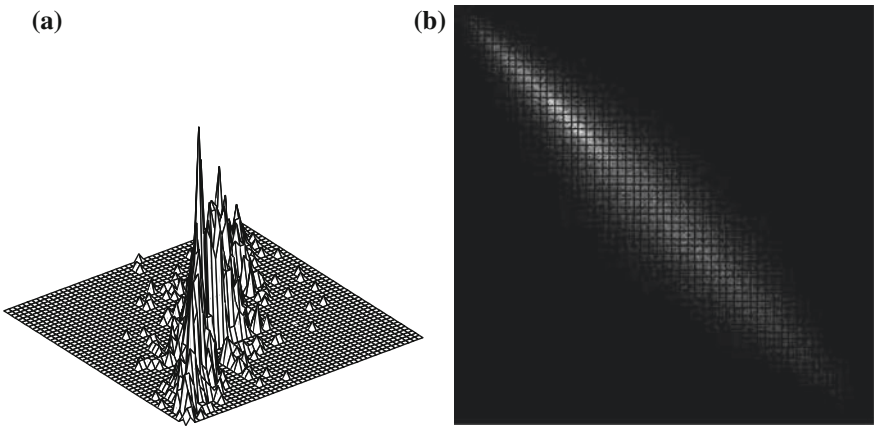


Fig. 11.13 Co-occurrence matrix **a** mesh plot; **b** image

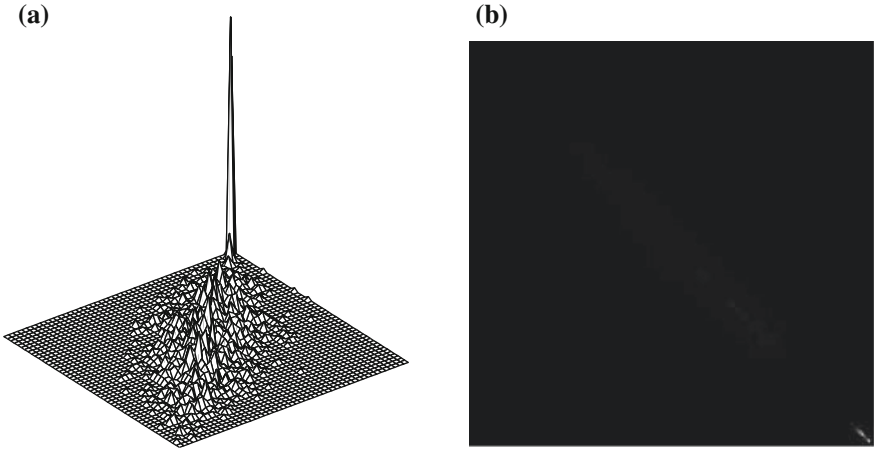


Fig. 11.14 Co-occurrence matrix **a** mesh plot; **b** image

due to the brightness of the image. In Fig. 11.16, due to the high contrast, the number of co-occurrences is small except at the low intensity values.

Table 11.4 shows the textural measures of the images in Figs. 11.8, 11.9, 11.10, and 11.11 based on the co-occurrence matrix. These measures can be used to differentiate the various types of textures.

11.2.3.3 Fourier Spectra Based Features

An image with texture typically has strong periodic spectral components. Figure 11.17a and b, show a 256 × 256 gray-level image and its DFT magnitude

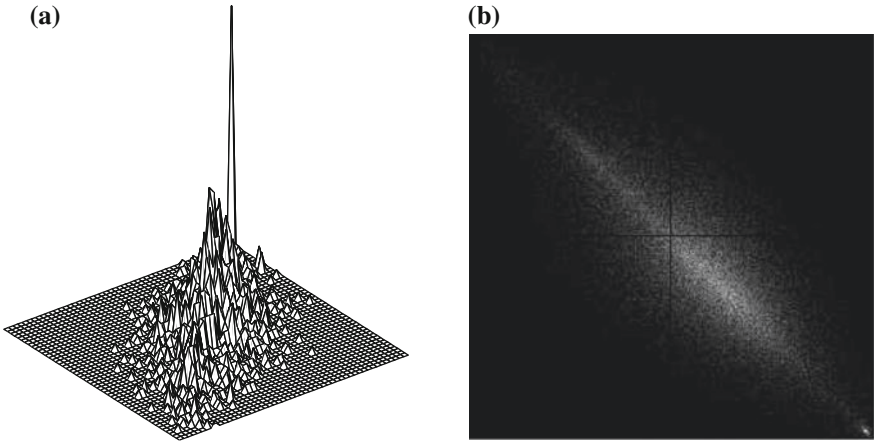


Fig. 11.15 Co-occurrence matrix **a** mesh plot; **b** image

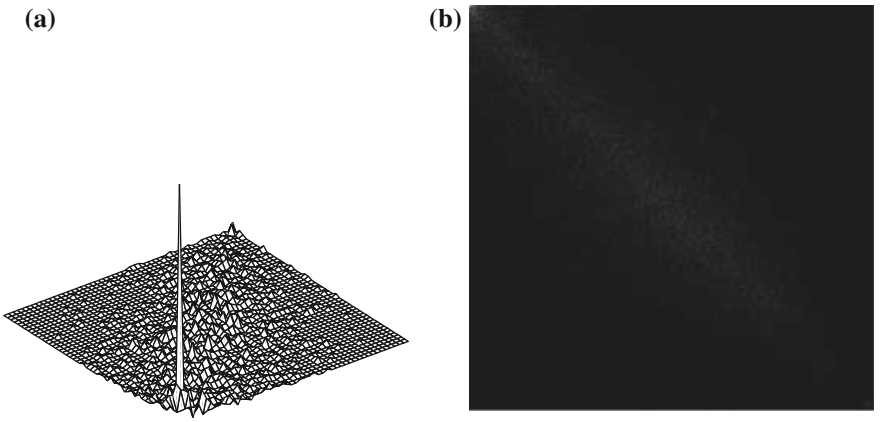


Fig. 11.16 Co-occurrence matrix **a** mesh plot; **b** image

Table 11.4 Texture measures based on co-occurrence matrix

Figure	Maximum p	E	H	U	C	R
11.8	0.0008	12.8600	0.1687	0.0002	380.6666	0.9188
11.9	0.0072	13.7058	0.1594	0.0002	756.0718	0.8703
11.10	0.0006	13.7221	0.1262	0.0001	838.6651	0.7726
11.11	0.0018	14.5118	0.0965	5.7966e − 05	2.2376e + 03	0.6996

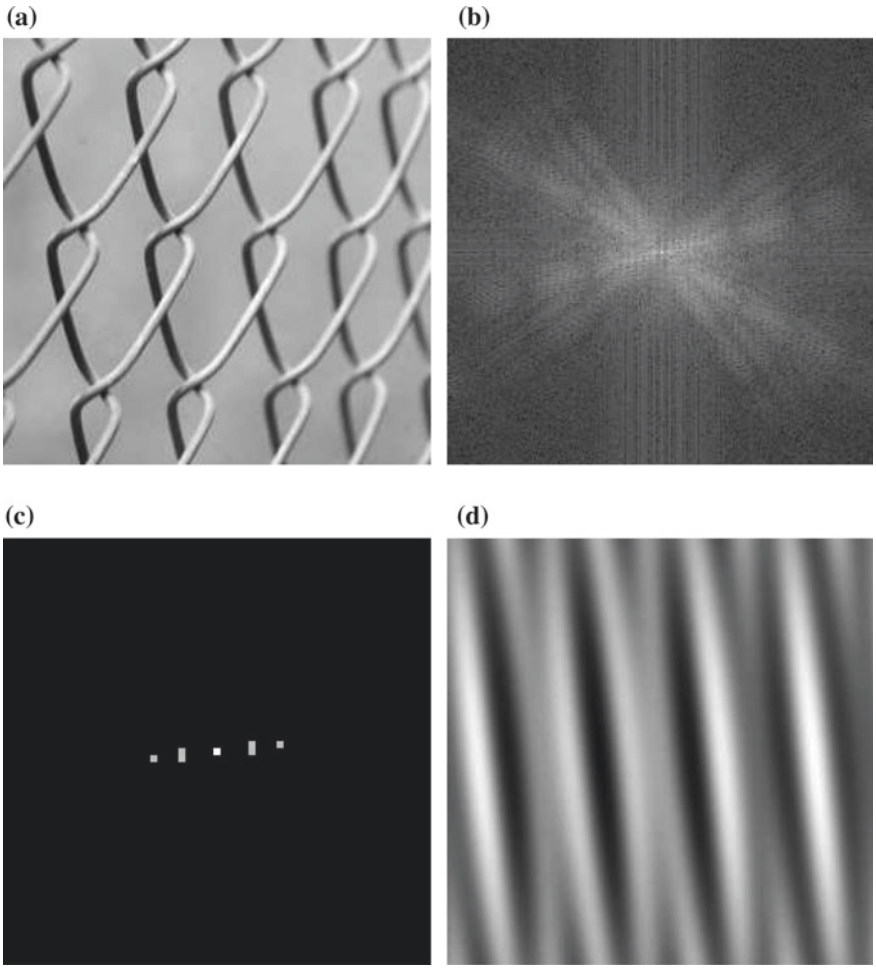


Fig. 11.17 **a** A 256 × 256 gray-level image; **b** its DFT magnitude spectrum in log scale; **c** a set of its largest DFT coefficients; **d** the reconstructed image from these coefficients

spectrum in log scale. It has dominant spectral components in two directions. A set of largest DFT coefficients is shown in Fig. 11.17c and the reconstructed image from these coefficients is shown in Fig. 11.17d. These coefficients are typically summed over sectors, rings, vertical and horizontal slits, as shown in Fig. 11.18. As the DFT coefficients are symmetric, the summation over half of the spectrum is sufficient. These sums are textural features. The DFT magnitude coefficients are shift invariant.

Fig. 11.18 Typical sector, ring, and slits over which the DFT coefficients are summed to represent texture

11.3 Principal Component Analysis

Signals may be random or deterministic. Fourier amplitude or power spectrum is a much simpler and effective representation of signals than their original form. A good approximation of the image, which is adequate for practical applications, can be obtained using a small set of coefficients. That is data reduction, which is a necessity in image processing. Principal component analysis (PCA) is another transformation, which serves a similar purpose, using linear algebra techniques. Matrix representation of data is transformed to its diagonal form. The data gets uncorrelated, and sufficient number of components can be used to approximate the data with a desired accuracy. Typical applications of PCA are in compression and approximation of the feature vectors. The more smoother the waveform, the lesser is the number of Fourier coefficients to represent the waveform. Similarly, the higher the correlation of the data, the more effective is the PCA in data reduction.

Let there be M vector variables, with each having N samples. The samples can be represented in column vectors. Then, the M variables form a $N \times M$ matrix

$$\boldsymbol{X} = [\boldsymbol{x}_0, \boldsymbol{x}_1 \cdots \boldsymbol{x}_{M-1}]$$

We want to find a matrix

$$\boldsymbol{Y} = [\boldsymbol{y}_0, \boldsymbol{y}_1 \cdots \boldsymbol{y}_{M-1}]$$

such that

$$\boldsymbol{Y} = \boldsymbol{X}\boldsymbol{R}$$

and the columns of $\boldsymbol{Y}$ are mutually orthogonal. Therefore,

$$\boldsymbol{Y}^T\boldsymbol{Y} = (\boldsymbol{X}\boldsymbol{R})^T(\boldsymbol{X}\boldsymbol{R}) = \boldsymbol{D}$$

where $\boldsymbol{D}$ is a diagonal matrix (a square matrix whose elements below and above the main diagonal are all zero). Using the property

$$(\boldsymbol{AB})^T = \boldsymbol{B}^T\boldsymbol{A}^T$$

and multiplying both sides by $1/(N-1)$, we get

$$\frac{1}{N-1}\boldsymbol{Y}^T\boldsymbol{Y} = \boldsymbol{R}^T\frac{(\boldsymbol{X}^T\boldsymbol{X})}{N-1}\boldsymbol{R} = \boldsymbol{D}$$

The factor

$$\frac{(\boldsymbol{X}^T\boldsymbol{X})}{N-1}$$

is the covariance matrix $\boldsymbol{C}$ of $\boldsymbol{X}$. Covariance is the generalization of the variance, presented in Chap. 2, for multiple variables. It is a measure of the linear dependence between two sets of data. From the diagonalization problem in linear algebra, $\boldsymbol{D}$ is composed of the eigenvalues of $\boldsymbol{C}$ and $\boldsymbol{R}$ is composed of the corresponding eigenvectors. The columns of $\boldsymbol{Y}$ are the principal components. The components are uncorrelated. The eigenvectors are the new coordinate system in which $\boldsymbol{Y}$ is represented. The orientation of the principal axis maximizes the overall variance of the data. The residual data is used to find the second principal axis and the process continues to find the all the principal axes, M. PCA is a coordinate transformation, similar to Fourier analysis.

Given two 2×2 images, let us find the corresponding PCA components and their covariance. Then, let us reconstruct the original images from the PCA components.

$$a(m,n) = \begin{bmatrix} 2 & 1 \\ 3 & 6 \end{bmatrix} \qquad b(m,n) = \begin{bmatrix} 3 & 2 \\ 1 & 2 \end{bmatrix}$$

The mean of the matrices are $am = 3$ and $bm = 2$. Subtracting the respective means from the matrices, we get

$$az(m,n) = \begin{bmatrix} -1 & -2 \\ 0 & 3 \end{bmatrix} \qquad bz(m,n) = \begin{bmatrix} 1 & 0 \\ -1 & 0 \end{bmatrix}$$

Converting $az(m,n)$ and $bz(m,n)$ into column vectors and concatenating, we get,

$$x(m,n) = \begin{bmatrix} -1 & 1 \\ -2 & 0 \\ 0 & -1 \\ 3 & 0 \end{bmatrix}$$

The covariance of this matrix is the scaled product of its transpose with itself.

$$C(m,n) = \frac{1}{3}\begin{bmatrix} -1 & -2 & 0 & 3 \\ 1 & 0 & -1 & 0 \end{bmatrix}\begin{bmatrix} -1 & 1 \\ -2 & 0 \\ 0 & -1 \\ 3 & 0 \end{bmatrix} = \frac{1}{3}\begin{bmatrix} 14 & -1 \\ -1 & 2 \end{bmatrix}$$

The covariance matrix is square with the dimensions equal to the number of images. This matrix is always symmetric (the matrix and its transpose are the same). In order to find the eigenvectors of this matrix, we have to find its eigenvalues. They are found by equating the determinant of $\boldsymbol{I} - \boldsymbol{C}$ to zero (its characteristic equation), where $\lambda \boldsymbol{I}$ is the identity matrix.

$$\begin{vmatrix} \lambda - \frac{14}{3} & \frac{1}{3} \\ \frac{1}{3} & \lambda - \frac{2}{3} \end{vmatrix} = 0 \quad \text{or} \quad \lambda^2 - \frac{16}{3}\lambda + \frac{27}{9} = 0 \quad \text{or} \quad (\lambda - 4.6943)(\lambda - 0.6391) = 0$$

The two eigenvalues are $\{4.6943, 0.6391\}$. For finding the eigenvectors, we use the equation

$$(\lambda \boldsymbol{I} - \boldsymbol{C})\boldsymbol{R} = 0$$

For $\lambda = 4.6943$, we get

$$\begin{bmatrix} 4.6943 - \frac{14}{3} & \frac{1}{3} \\ \frac{1}{3} & 4.6943 - \frac{2}{3} \end{bmatrix}\begin{bmatrix} R(0) \\ R(1) \end{bmatrix} = 0$$

For $\lambda = 0.6391$, we get

$$\begin{bmatrix} 0.6391 - \frac{14}{3} & \frac{1}{3} \\ \frac{1}{3} & 0.6391 - \frac{2}{3} \end{bmatrix}\begin{bmatrix} R(0) \\ R(1) \end{bmatrix} = 0$$

Solving the two sets of equations, we get the eigenvectors as

$$\boldsymbol{R} = \begin{bmatrix} -0.9966 & 0.0825 \\ 0.0825 & 0.9966 \end{bmatrix}$$

The first and second columns are, respectively, the eigenvectors corresponding to eigenvalues 4.6943 and 0.6391. The principal components are found as

$$\boldsymbol{Y} = \boldsymbol{X}\boldsymbol{R} = \begin{bmatrix} -1 & 1 \\ -2 & 0 \\ 0 & -1 \\ 3 & 0 \end{bmatrix}\begin{bmatrix} -0.9966 & 0.0825 \\ 0.0825 & 0.9966 \end{bmatrix} = \begin{bmatrix} 1.0791 & 0.9141 \\ 1.9932 & -0.1650 \\ -0.0825 & -0.9966 \\ -2.9898 & 0.2474 \end{bmatrix}$$

The covariance of the PCA component matrix is the scaled product of its transpose with itself.

$$C(m,n) = \frac{1}{3}\begin{bmatrix} 1.0791 & 1.9932 & -0.0825 & -2.9898 \\ 0.9141 & -0.1650 & -0.9966 & 0.2474 \end{bmatrix}\begin{bmatrix} 1.0791 & 0.9141 \\ 1.9932 & -0.1650 \\ -0.0825 & -0.9966 \\ -2.9898 & 0.2474 \end{bmatrix} = \begin{bmatrix} 4.6943 & 0 \\ 0 & 0.6391 \end{bmatrix}$$

Note that the components are uncorrelated (the two zero entries). The input can be reconstructed by

$$\boldsymbol{Y}\boldsymbol{R}^T + \begin{bmatrix} am & bm \\ am & bm \\ am & bm \\ am & bm \end{bmatrix} = \begin{bmatrix} 1.0791 & 0.9141 \\ 1.9932 & -0.1650 \\ -0.0825 & -0.9966 \\ -2.9898 & 0.2474 \end{bmatrix}\begin{bmatrix} -0.9966 & 0.0825 \\ 0.0825 & 0.9966 \end{bmatrix} + \begin{bmatrix} 3 & 2 \\ 3 & 2 \\ 3 & 2 \\ 3 & 2 \end{bmatrix}$$

$$= \begin{bmatrix} -1 & 1 \\ -2 & 0 \\ 0 & -1 \\ 3 & 0 \end{bmatrix} + \begin{bmatrix} 3 & 2 \\ 3 & 2 \\ 3 & 2 \\ 3 & 2 \end{bmatrix} = \begin{bmatrix} 2 & 3 \\ 1 & 2 \\ 3 & 1 \\ 6 & 2 \end{bmatrix}$$

The covariance matrix can be computed from the 2×2 matrices directly. Using the formula, for M number of $P \times Q$ images $x_i(m,n)$,

$$C(k,l) = \frac{1}{PQ-1}\sum_{m=0}^{P-1}\sum_{n=0}^{Q-1}(x_k(m,n)-\bar{x}_k)(x_l(m,n)-\bar{x}_l), \qquad k = 0,1,\ldots M-1, l = 0,1,\ldots M-1$$

Figure 11.19a shows a 256×256 RGB color image. Figure 11.19b shows the gray-level image obtained by averaging its RGB components. In the representation of multispectral images, the uncorrelated PCA components can be used to reduce the data. For example, the first principal component may be an adequate representation. Gray-level version is often used in image processing. The PCA representation is superior to averaging. Figure 11.20a, c, e show the R, G, and B components of a

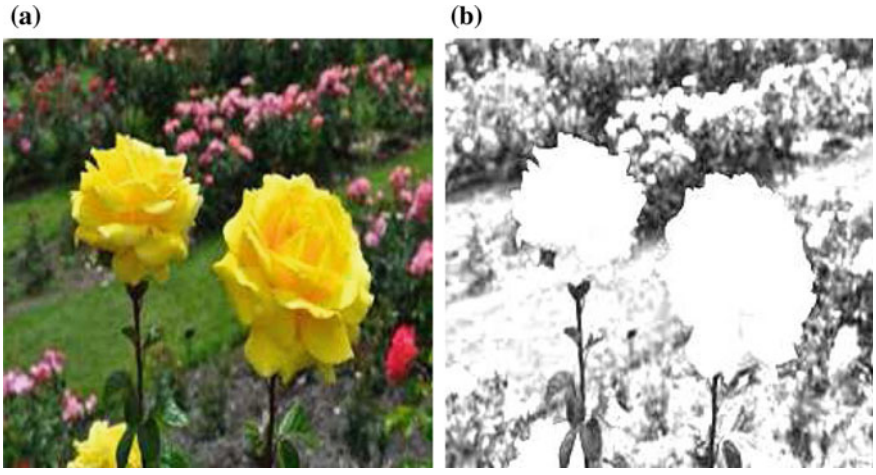


Fig. 11.19 **a** A 256×256 RGB color image; **b** *gray*-level image obtained by averaging its RGB components

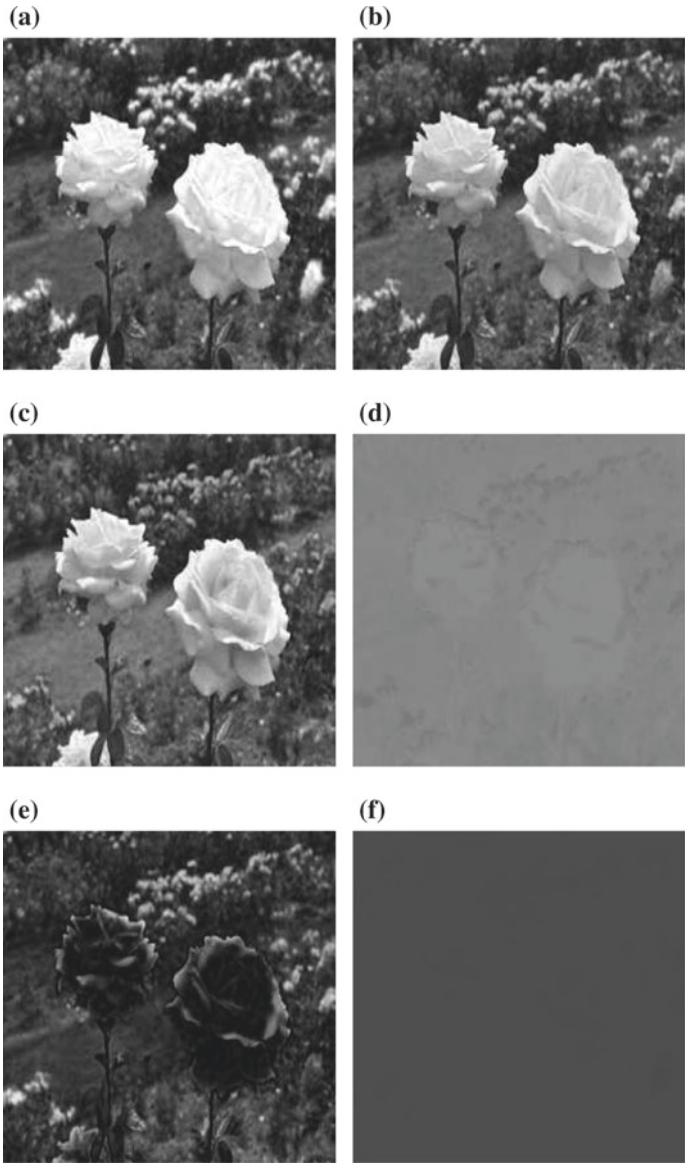


Fig. 11.20 **a, c, e** The R, G, and B components of a 256×256 color image; **b, d, f** the reconstructed images using the PCA components with variance decreasing, respectively

color image. Figure 11.20b, d, f show the PCA components of the color image. Figure 11.20b, with maximum variance, is a good gray-level version of the color image.

11.4 Summary

- Feature is any attribute of an object that characterizes it and can be used for segmentation and classification. Features are compact description of an image that is sufficient and convenient for further analysis.
- An object can be characterized by its shape (external) and the properties of the pixels (internal) comprising the object. External features characterize the shape and internal features characterize properties such as texture.
- Chain codes, signatures, and Fourier descriptors are typical shape descriptors.
- Area, perimeter, compactness, histograms, moments, Fourier coefficients, and Euler number are typical internal features of an object.
- It is desirable that descriptors and features are invariant with respect to translation, rotation, and scaling.
- Texture is a pattern resembling a mosaic, made by a physical composition of an object using constituents of various sizes and shapes. It is characterized by histogram and co-occurrence matrix-based features.
- Principal component analysis is a statistical and linear algebraic method that provides data reduction of images and their features, similar to that of the other transforms. It decomposes the principal constituent components of an image and the image can be adequately described by few components with high variances. The components are orthogonal and their covariance matrix is diagonal.

Exercises

11.1 Find the chain code for the image.
*(i)

$$\begin{bmatrix} 0&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0 \\ 0&1&1&0&0&0&0&0 \\ 0&1&0&1&0&0&0&0 \\ 0&1&0&0&1&0&0&0 \\ 0&1&0&0&0&1&0&0 \\ 0&1&1&1&1&1&1&0 \\ 0&0&0&0&0&0&0&0 \end{bmatrix}$$

(ii)

$$\begin{bmatrix} 0&0&0&0&0&0&0&0 \\ 0&0&1&1&1&0&0&0 \\ 0&1&0&0&0&1&0&0 \\ 0&1&0&0&0&1&0&0 \\ 0&1&0&0&0&1&0&0 \\ 0&0&1&1&1&0&0&0 \\ 0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0 \end{bmatrix}$$

(iii)

$$\begin{bmatrix} 0&0&0&0&0&0&0&0 \\ 0&1&1&1&1&1&0&0 \\ 0&1&0&0&0&1&0&0 \\ 0&1&0&0&0&1&0&0 \\ 0&1&0&0&0&1&0&0 \\ 0&0&1&1&1&0&0&0 \\ 0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0 \end{bmatrix}$$

11.2 Find the signature of the image.

(i)

$$\begin{bmatrix} 1&0&0&0 \\ 1&1&0&0 \\ 1&0&1&0 \\ 1&1&1&1 \end{bmatrix}$$

*(ii)

$$\begin{bmatrix} 1&1&1&1 \\ 1&0&0&1 \\ 1&0&0&1 \\ 1&1&1&1 \end{bmatrix}$$

(iii)

$$\begin{bmatrix} 0&1&1&0 \\ 1&0&0&1 \\ 1&0&0&1 \\ 0&1&1&0 \end{bmatrix}$$

11.3 Find the Fourier descriptor of the image. Reconstruct the image from the descriptor and verify that it is the same as the input image.

(i)

$$\begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}$$

(ii)

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

* (iii)

$$\begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

11.4 Find the area, perimeter, and compactness of the nonzero region in the 3×3 image.

(i)

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

(ii)

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

(iii)

$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

11.5 Find the Euler number of the 8-connected 4×4 image.

(i)

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

(ii)

$$\begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

(iii)

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

11.6 Find the first two normalized central moments ϕ_1 and ϕ_2 of the 4×4 image.

$$x(k,l) = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

11.7 Find the first two normalized central moments ϕ_1 and ϕ_2 of the 4×4 image.

$$x(k,l) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

***11.8** Find the first two normalized central moments ϕ_1 and ϕ_2 of the 4×4 image.

$$x(k,l) = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

***11.9** Find the histogram of the 4×4 8-bit image and derive its histogram-based features.

$$\begin{bmatrix} 108 & 81 & 69 & 92 \\ 114 & 105 & 72 & 101 \\ 117 & 73 & 67 & 92 \\ 105 & 0 & 65 & 101 \end{bmatrix}$$

11.10 Find the histogram of the 4×4 8-bit image and derive its histogram-based features.

$$\begin{bmatrix} 120 & 103 & 83 & 78 \\ 97 & 99 & 81 & 72 \\ 102 & 96 & 78 & 73 \\ 121 & 107 & 37 & 0 \end{bmatrix}$$

11.11 Find the histogram of the 4×4 8-bit image and derive its histogram-based features.

$$\begin{bmatrix} 10 & 133 & 175 & 170 \\ 0 & 61 & 171 & 170 \\ 3 & 15 & 131 & 172 \\ 1 & 3 & 70 & 167 \end{bmatrix}$$

***11.12** Find the co-occurrence matrix of the 8×8 3-bit image.

$$\begin{bmatrix} 5 & 2 & 2 & 2 & 2 & 1 & 1 & 2 \\ 5 & 4 & 1 & 2 & 2 & 2 & 1 & 1 \\ 5 & 5 & 2 & 2 & 2 & 2 & 1 & 1 \\ 5 & 5 & 4 & 1 & 2 & 2 & 1 & 1 \\ 6 & 5 & 6 & 2 & 2 & 2 & 2 & 1 \\ 6 & 4 & 5 & 4 & 1 & 2 & 2 & 1 \\ 5 & 5 & 5 & 5 & 2 & 2 & 2 & 1 \\ 5 & 4 & 5 & 5 & 3 & 2 & 2 & 2 \end{bmatrix}$$

Let the spatial relationship of a pair of pixels be $x(m, n)$ and $x(m, n + 1)$. That is, a pixel and its immediate right neighbor form the pair. Find the contrast, correlation, energy, and homogeneity.

11.13 Find the co-occurrence matrix of the 8×8 3-bit image.

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 2 & 2 & 5 \\ 1 & 0 & 0 & 0 & 1 & 2 & 2 & 5 \\ 1 & 0 & 0 & 0 & 1 & 2 & 2 & 4 \\ 1 & 0 & 0 & 0 & 1 & 2 & 2 & 4 \\ 2 & 1 & 0 & 0 & 1 & 2 & 2 & 3 \\ 2 & 1 & 0 & 0 & 1 & 1 & 1 & 3 \\ 2 & 1 & 0 & 0 & 1 & 1 & 1 & 3 \\ 2 & 1 & 0 & 0 & 1 & 1 & 1 & 3 \end{bmatrix}$$

Let the spatial relationship of a pair of pixels be $x(m, n)$ and $x(m, n + 1)$. That is, a pixel and its immediate right neighbor form the pair. Find the contrast, correlation, energy, and homogeneity.

11.14 Find the co-occurrence matrix of the 8×8 3-bit image.

$$\begin{bmatrix} 5 & 4 & 4 & 5 & 5 & 2 & 2 & 2 \\ 5 & 2 & 4 & 5 & 5 & 2 & 2 & 2 \\ 3 & 2 & 5 & 5 & 6 & 3 & 2 & 2 \\ 2 & 4 & 4 & 5 & 5 & 4 & 2 & 2 \\ 4 & 5 & 5 & 5 & 6 & 4 & 2 & 2 \\ 5 & 5 & 5 & 5 & 4 & 4 & 2 & 2 \\ 5 & 5 & 5 & 3 & 3 & 4 & 2 & 2 \\ 5 & 5 & 4 & 3 & 4 & 5 & 2 & 2 \end{bmatrix}$$

Let the spatial relationship of a pair of pixels be $x(m, n)$ and $x(m, n + 1)$. That is, a pixel and its immediate right neighbor form the pair. Find the contrast, correlation, energy, and homogeneity.

11.15 Given two 2×2 images, find the corresponding PCA components and their covariance. Then, reconstruct the original images from the PCA components.

$$a(m, n) = \begin{bmatrix} 2 & 1 \\ 3 & 4 \end{bmatrix} \qquad b(m, n) = \begin{bmatrix} 3 & 1 \\ 2 & 4 \end{bmatrix}$$

***11.16** Given two 2×2 images, find the corresponding PCA components and their covariance. Then, reconstruct the original images from the PCA components.

$$a(m, n) = \begin{bmatrix} 1 & 1 \\ 2 & 3 \end{bmatrix} \qquad b(m, n) = \begin{bmatrix} 3 & 1 \\ 2 & 4 \end{bmatrix}$$

11.17 Given two 2×2 images, find the corresponding PCA components and their covariance. Then, reconstruct the original images from the PCA components.

$$a(m, n) = \begin{bmatrix} 2 & 1 \\ 3 & 4 \end{bmatrix} \qquad b(m, n) = \begin{bmatrix} 2 & 2 \\ 1 & 3 \end{bmatrix}$$

Chapter 12
Object Classification

Abstract The different objects in an image have different characteristics represented by their features. Given a set of features of an object, comparing that with those in the database and assigning it to its proper class is classification. There are two main types of classification: (i) supervised classification and (ii) unsupervised classification. In supervised classification, features are specified a priori and objects are classified using them. Typical methods used are minimum distance, k-nearest neighbors, decision trees, and statistical (based on probability distribution models). The decision is prior. In unsupervised classification, we classify the objects by the constraints imposed by the features. The decision is posterior.

The different objects in an image have different characteristics represented by their features. Given a set of features of an object, comparing that with those in the database and assigning it to its proper class is classification. The task includes selection of the smallest set of features that can classify a set of objects with minimum effort and high reliability.

There are two main types of classification: (i) supervised classification and (ii) unsupervised classification. In supervised classification, features are specified a priori and objects are classified using them. Typical methods used are minimum distance, k-nearest neighbors, decision trees, and statistical (based on probability distribution models). The decision is prior. In unsupervised classification, we classify the objects by the constraints imposed by the features. Partition the data into groups by clustering. Unknown, but distinct set of feature classes are generated. The decision is posterior.

12.1 The k-Nearest Neighbors Classifier

In this method, the feature set of a test object is compared with the reference set, and the test object is assigned to the class whose features differ, with respect to some measure, by the least from that of the test object. In terms of distance, computing the distance between the k closest points in the reference sets of feature vectors

D. Sundararajan, *Digital Image Processing*, DOI 10.1007/978-981-10-6113-4_12

is the measure. The method is simple, capable of classifying overlapping classes and classes with complex structures. With $k = 1$, it becomes the minimum distance classifier.

Consider three classes each with three feature vectors each of length 2, as shown in Fig. 12.1. Each class is discriminated by different symbols. The two test data are {15, 5} and {9, 15}, shown by the pentagram symbol. The feature vectors are also shown in Table 12.1. The problem is to assign the test data to the most appropriate class. The distances between the test data {15, 5} and the other nine feature vectors are

$$\{11.0454, 10.1980, 9.2195, 5.6569, 7.2111, 8.0623, 11.0000, 11.1803, 9.0554\}$$

For example, the first distance is

$$d = \sqrt{(16-15)^2 + (16-5)^2} = 11.0454$$

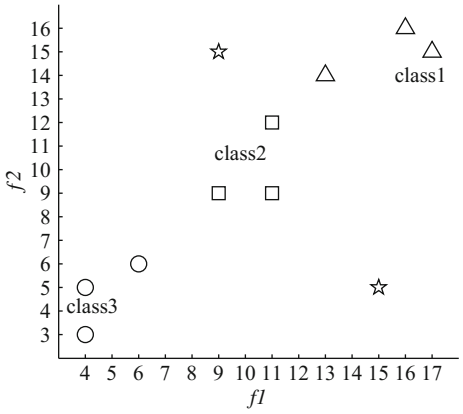


Fig. 12.1 k-nearest neighbors classifier

Table 12.1 Three classes each with three feature vectors of length 2

Class	Feature 1	Feature 2
Class1	16	16
Class1	17	15
Class1	13	14
Class2	11	9
Class2	9	9
Class2	11	12
Class3	4	5
Class3	4	3
Class3	6	6

The sorted distances are

$$\{5.6569, 7.2111, 8.0623, 9.0554, 9.2195, 10.1980, 11.0000, 11.0454, 11.1803\}$$

Sometimes, in order to reduce computation, an alternative definition of distance, called the city-block distance, is used. It is computed as

$$d = |(16 - 15)| + |(16 - 5)| = 12$$

It is the walking distance along the horizontal and vertical directions only, assuming a 4-neighborhood. The shortest three distances are

$$\{5.6569, 7.2111, 8.0623\}$$

All these three distances correspond to class2 and, therefore, the test data is assigned to class2. This result is obvious from Fig. 12.1. In the case that there is no maximally represented class, the test sample can be assigned to the class of the nearest neighbor.

The distances between the test data {9, 15} and the other nine feature vectors are

$$\{7.0711, 8.0000, 4.1231, 6.3246, 6.0000, 3.6056, 11.1803, 13.0000, 9.4868\}$$

The sorted distances are

$$\{3.6056, 4.1231, 6.0000, 6.3246, 7.0711, 8.0000, 9.4868, 11.1803, 13.0000\}$$

The shortest three distances are

$$\{3.6056, 4.1231, 6.0000\}$$

Out of these three, two of the distances correspond to class2 and, therefore, the test data are assigned to class2. The selection of more than one neighbor smooths the result, and it is less likely that classification is affected by noisy outlier points.

12.2 The Minimum-Distance-to-Mean Classifier

In this type of classification, the means of a set of features of a class characterize the class. A new sample is assigned to a class if the distance between the means of the reference set of that class and that of the new sample is minimum. There are other distance measures apart from the familiar Euclidean distance.

Let there be three classes each with mean vectors of length 2, as shown in Table 12.2. The mean vectors, $\{f1, f2\}$, of description of the three classes of objects are: (i) {6400, 320} (ii) {2500, 5000} and (iii) {500, 1000}. Now, the problem is to find the boundary discriminant functions. Consider the vectors of class1 and class2,

Table 12.2 Three classes each with mean vectors of length 2

Class	Feature 1	Feature 2
Class1	6400	320
Class2	2500	5000
Class3	500	1000

{6400, 320} and {2500, 5000}. First, we find the slope of the line passing through these points. The slope m of a line passing through points $(x1, y1)$ and $(x2, y2)$ is defined by

$$m = \frac{y2 - y1}{x2 - x1}$$

Therefore, the slope of the line

$$m = \frac{5000 - 320}{2500 - 6400} = \frac{4680}{-3900} = -\frac{6}{5}$$

The midpoint of this line is

$$(6400 - (6400 - 2500)/2, 320 + (5000 - 320)/2) = (6400 - (3900/2), 320 + (4680/2)) = (4450, 2660)$$

The line perpendicular to this line characterizes the boundary discriminant function for class1 and class2. The slope of a perpendicular line to a line with slope m is $-1/m$. Therefore, the slope of this line passing through the point $= (4450, 2660)$ is $= \frac{5}{6}$. The point-slope form of a line with slope m and passing through $(x1, y1)$ is

$$\frac{y - y1}{x - x1} = m \quad \text{or} \quad y = mx - mx1 + y1$$

The boundary discriminant function is

$$\frac{f2 - 2660}{f1 - 4450} = \frac{5}{6} \quad \text{or} \quad f2 = \frac{5}{6}f1 - \frac{5}{6}(4450) + 2660 = \frac{5}{6}f1 - 1048.3$$

This is line $l12$ as shown in Fig. 12.2. The function

$$\frac{5}{6}f1 - f2 - 1048.3$$

evaluates to negative if a test vector is closer to class2. A positive value indicates that it is closer to class1. Similarly, lines $l13$ and $l23$ can be found, respectively, as

$$f2 = 8.6765f1 - 29274 \quad \text{and} \quad f2 = -0.5f1 + 3750$$

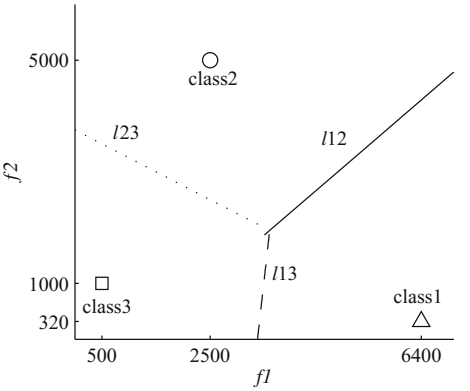


Fig. 12.2 Minimum-distance-to-mean classifier

While the procedure is simple enough, a general approach to the formulation of the problem is required for the classification with a large set of features.

12.2.1 Decision-Theoretic Methods

In this approach, the classification of an object is based on discriminant functions. Let the feature vector $\boldsymbol{x}$ of three classes be $\boldsymbol{m}_1$, $\boldsymbol{m}_2$, and $\boldsymbol{m}_3$. Three discriminant functions, $d_1(\boldsymbol{x})$, $d_2(\boldsymbol{x})$, and $d_3(\boldsymbol{x})$, have to be found such that the discriminant function corresponding to an unclassified feature vector will yield a value that is greater than those of the functions to which it does not belong. In the case of two or more functions evaluating to the same value, the decision is arbitrary or based on some additional factors. This formulation is another form of the minimum distance classifier.

Let the elements of a test vector be

$$\boldsymbol{x} = \{x_1, x_2, \ldots, x_M\}$$

Let the mean vector of the N classes be

$$\{\boldsymbol{m}_1, \boldsymbol{m}_2, \ldots, \boldsymbol{m}_N\}$$

Then,

$$d_n(\boldsymbol{x}) = \boldsymbol{x}\boldsymbol{m}_n^T - 0.5\boldsymbol{m}_n\boldsymbol{m}_n^T, \quad n = 1, 2, \ldots, N$$

Let us get the discriminant functions for the last example. For the first feature vector {6400, 320},

$$\begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} 6400 \\ 320 \end{bmatrix} - 0.5 \begin{bmatrix} 6400 & 320 \end{bmatrix} \begin{bmatrix} 6400 \\ 320 \end{bmatrix}$$

Simplifying, we get

$$d_1(\boldsymbol{x}) = 6400x_1 + 320x_2 - 20531200$$

Similarly, for class2, we get

$$d_2(\boldsymbol{x}) = 2500x_1 + 5000x_2 - 15625000$$

For class3, we get

$$d_3(\boldsymbol{x}) = 500x_1 + 1000x_2 - 625000$$

For the first feature vector, these three functions yield

$$\{20531200, 1975000, 2895000\}$$

As expected, the first function has the greatest value. Similarly, for the second feature vector, these three functions yield

$$\{-2931200, 15625000, 5625000\}$$

For the third feature vector, these three functions yield

$$\{-17011200, -9375000, 625000\}$$

The difference between two discriminant functions is the boundary discriminant function for them. For example, the boundary discriminant function for class1 and class2 is

$$\begin{aligned} d_{12}(\boldsymbol{x}) &= d_1(\boldsymbol{x}) - d_2(\boldsymbol{x}) \\ &= (6400x_1 + 320x_2 - 20531200) - (2500x_1 + 5000x_2 - 15625000) \\ &= 3900x_1 - 4680x_2 - 4906200 \end{aligned}$$

which is the same thing as that we got earlier by coordinate geometry. Classification based on the mean works well if the means are well spread out relative to the spread of the members of each class.

12.3 Decision Tree Classification

In this approach, the feature space is split into unique regions sequentially. A decision is arrived with out testing all classes and, therefore, it is advantages when the number of classes is large. Further, the convergence of this algorithm is guaranteed irrespective of the nature of the feature space.

Table 12.3 Objects and their features

Object	A	B	C	D	E
Holes	1	2	0	1	0
End points	2	0	2	0	3

Let us consider the problem of character recognition, a common application of image processing. In this problem, the tasks required may be removing noise, segmentation by thresholding, thinning, finding the end points and holes, comparing with the feature vectors, and recognizing the characters. For simplicity, let the text to be analyzed consists of the five uppercase alphabets {A, B, C, D, E}. The character set is predetermined and each character is discrete. A more complex problem may need more processing steps and more number of feature vectors with increased complexity of the algorithms. However, the basic steps in solving a problem remain the same.

Two sets of features describing the characters are shown in Table 12.3. This classifier isolates each object in a sequential manner. The first step is to sort the entries in each row of the feature vectors in ascending order. The maximum difference of adjacent entries in the first row is 1. It is 2 in the second row. A threshold, that is the average of the two adjacent entries (0, 2) that produced the maximum difference, is set. Using this threshold $f2 = (0 + 2)/2 = 1$, the rows are partitioned as

$$\left[\begin{array}{cc|ccc} 0 & 0 & 1 & 1 & 2 \\ 0 & 0 & 2 & 2 & 3 \end{array}\right]$$

This partitioning continues until each partition is just one column. Now, the order of the first row is restored and we get

$$\left[\begin{array}{cc|ccc} 2 & 1 & 1 & 0 & 0 \\ 0 & 0 & 2 & 2 & 3 \end{array}\right]$$

Now, the left side partition includes the letters B and D, which can be partitioned with a threshold $f1 = (1 + 2)/2 = 1.5$. The unsorted and sorted feature vectors for the other three characters are

$$\left[\begin{array}{ccc} 1 & 0 & 0 \\ 2 & 2 & 3 \end{array}\right] \quad \left[\begin{array}{cc|c} 0 & 0 & 1 \\ 2 & 2 & 3 \end{array}\right]$$

The letter A can be isolated with a threshold $f1 = 0.5$. Letters C and E can be isolated with a threshold $f2 = 2.5$. The decision tree classifier and its flowchart are shown in Figs. 12.3 and 12.4, respectively. The advantage of this method is that decisions are made without testing all the feature vectors, which is desirable for solving a problem with a large number of feature vectors. The algorithm always converges.

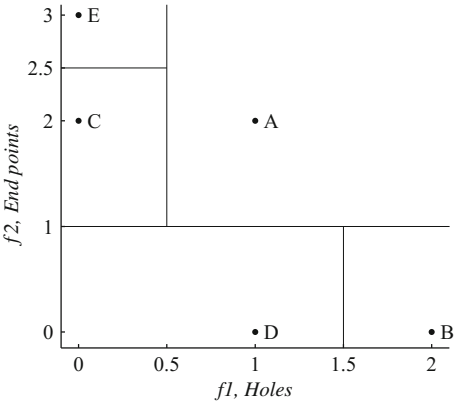


Fig. 12.3 Decision tree classifier

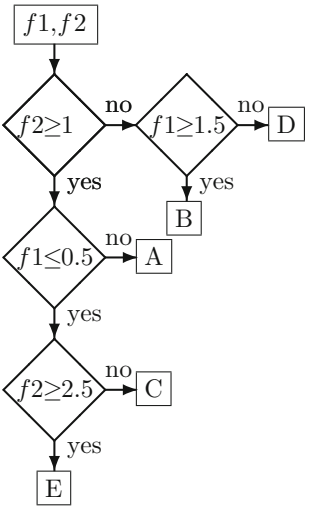


Fig. 12.4 Flowchart of the decision tree algorithm

12.4 Bayesian Classification

Samples originated from one class may lie closer to another class. In this case, the minimum distance classifier, which relies on the mean of class only, cannot do effective classification. The Bayesian approach to statistical methods of classification is based not only on the set of samples but also on the pertinent prior information. The Bayesian approach provides discriminant or decision functions, which maximize the number of correct classifications and minimize the incorrect ones. Let there be N features $\boldsymbol{x} = \{x_1, x_2, \ldots, x_N\}^T$ representing the M classes of objects, $\{\omega_1, \omega_2, \ldots, \omega_M\}$. Let the a priori probability of an arbitrary object belongs to class ω_i be $\{p(\omega_1), p(\omega_2), \ldots, p(\omega_M)\}$. Let the density distribution of all the objects be

$p(\boldsymbol{x})$. Let the conditional density distribution of all the objects belonging to class ω_i be $p(\boldsymbol{x}/\omega_i)$. Using the Bayes' theorem, the decision rule is

$$\text{if} \quad p(\boldsymbol{x}/\omega_i)p(\omega_i) > p(\boldsymbol{x}/\omega_j)p(\omega_j) \quad \text{for all} \quad i \neq j, \quad \text{then assign } \boldsymbol{x} \text{ to } \omega_i$$

Since it is difficult to estimate the actual $p(\boldsymbol{x}/\omega_i)$, in practice, the Gaussian (normal) density function is often assumed.

For normal distribution,

$$p(\boldsymbol{x}/\omega_i) = \frac{1}{(2\pi)^{(N/2)}|\boldsymbol{C}_i|^{0.5}} e^{-0.5(\boldsymbol{x}-\boldsymbol{m}_i)^T \boldsymbol{C}_i^{-1}(\boldsymbol{x}-\boldsymbol{m}_i)}$$

where the mean $\boldsymbol{m}_i$ and the covariance matrix $\boldsymbol{C}_i$ are approximated as

$$\boldsymbol{m}_i = \frac{1}{N_i}\sum_{x \in \omega_i} \boldsymbol{x} \quad \text{and} \quad \boldsymbol{C}_i = \frac{1}{N_i - 1}(\boldsymbol{x_i} - \boldsymbol{m_i})^T(\boldsymbol{x_i} - \boldsymbol{m_i})$$

The determinant of $\boldsymbol{C}_i$ is $|\boldsymbol{C}_i|$. Since $p(\boldsymbol{x}/\omega_i)$ is in exponential form, the decision rule

$$d_i(\boldsymbol{x}) = p(\boldsymbol{x}/\omega_i)p(\omega_i)$$

is changed to the form

$$d_i(\boldsymbol{x}) = \log_e(p(\boldsymbol{x}/\omega_i)p(\omega_i)) = \log_e(p(\boldsymbol{x}/\omega_i)) + \log_e(p(\omega_i))$$

for convenience of manipulation. This change in form does not alter the numerical order of the decision functions required for classification.

Substituting the exponential expression for $p(\boldsymbol{x}/\omega_i)$, we get the Bayes decision function

$$d_i(\boldsymbol{x}) = \log_e(p(\omega_i)) - 0.5\log_e(|\boldsymbol{C}_i|) - 0.5((\boldsymbol{x} - \boldsymbol{m}_i)^T \boldsymbol{C}_i^{-1}(\boldsymbol{x} - \boldsymbol{m}_i)), \quad i = 1, 2, \ldots, M \tag{12.1}$$

As the term $-(N/2)\log_e(2\pi)$ does not affect the numerical order of the decision functions, it is dropped. If all the covariance matrices are the same, then

$$d_i(\boldsymbol{x}) = \log_e(p(\omega_i)) + \boldsymbol{x}^T \boldsymbol{C}^{-1}\boldsymbol{m}_i - 0.5\boldsymbol{m}_i^T \boldsymbol{C}^{-1}\boldsymbol{m}_i, \quad i = 1, 2, \ldots, M$$

Further, if $\boldsymbol{C}$ is the identity matrix and $p(\omega_i) = 1/M,\ i = 1, 2, \ldots, M$, then

$$d_i(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{m}_i - 0.5\boldsymbol{m}_i^T \boldsymbol{m}_i, \quad i = 1, 2, \ldots, M$$

which is the same for the decision function of the minimum distance classifier.

Expanding the last term in Eq. 12.1

$$((\boldsymbol{x} - \boldsymbol{m}_i)^T \boldsymbol{C}_i^{-1}(\boldsymbol{x} - \boldsymbol{m}_i)),$$

for a class2 problem with $i = 1$, we get

$$\begin{bmatrix} (x_1 - m_1) & (x_2 - m_2) \end{bmatrix} \begin{bmatrix} ci_1(0,0) & ci_1(0,1) \\ ci_1(1,0) & ci_1(1,1) \end{bmatrix} \begin{bmatrix} (x_1 - m_1) \\ (x_2 - m_2) \end{bmatrix}$$

where the middle matrix is the coefficients of the inverse of the covariance matrix for class1. Simplifying, we get

$$ci_1(0,0)x_1^2 + ci_1(1,1)x_2^2 + (ci_1(0,1) + ci_1(1,0))x_1x_2 - (2ci_1(0,0)m_1 + (ci_1(0,1) + ci_1(1,0))m_2)x_1$$

$$-(2ci_1(1,1)m_2 + (ci_1(0,1) + ci_1(1,0))m_1)x_2 + ci_1(0,0)m_1^2 + ci_1(1,1)m_2^2 + (ci_1(0,1) + ci_1(1,0))m_1m_2$$

Figure 12.5a shows samples of training data of two classes, with their mean indicated by the square symbol. The class1 and class2 data are

$$c1(m,n) = \begin{bmatrix} 0.1795 & -1.7891 \\ 0.9715 & -1.4020 \\ 0.8436 & -1.0719 \\ 1.5955 & -0.1995 \\ 2.0372 & -0.8081 \\ 2.0523 & -0.4025 \\ 0.1807 & -1.4430 \\ 1.0734 & -0.5768 \end{bmatrix} \quad c2(m,n) = \begin{bmatrix} -2.5402 & -0.5013 \\ -0.9540 & 4.3240 \\ -0.2186 & 0.1294 \\ 0.5565 & 2.0579 \\ 1.1838 & 0.7279 \\ -0.8785 & 2.2567 \\ -3.1094 & -0.0817 \\ -2.0498 & -0.9831 \end{bmatrix}$$

The mean of class1 is $\{1.1167, -0.9616\}$ and that of class2 is $\{-1.0013, 0.9912\}$. Subtracting the respective mean from $c1(m,n)$ and $c2(m,n)$, we get

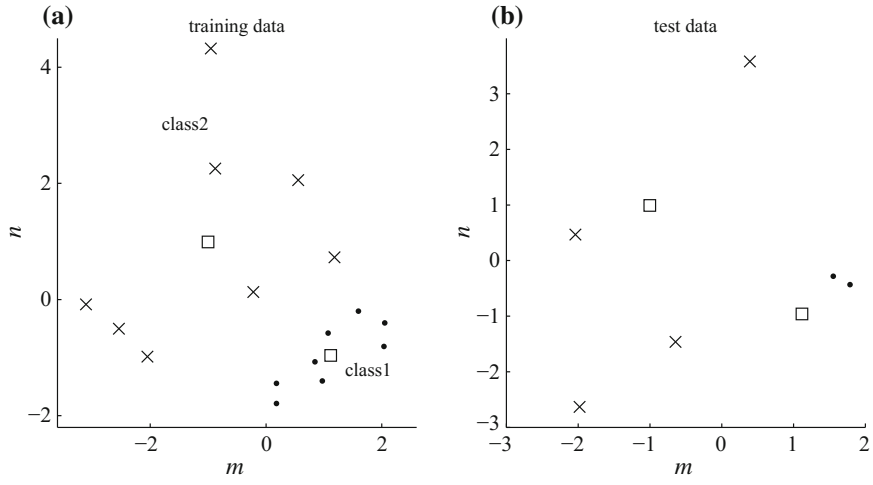


Fig. 12.5 Bayes classifier. **a** Training data; **b** test samples

$$cm1(m,n) = \begin{bmatrix} -0.9372 & -0.8275 \\ -0.1452 & -0.4404 \\ -0.2731 & -0.1103 \\ 0.4788 & 0.7621 \\ 0.9205 & 0.1535 \\ 0.9356 & 0.5591 \\ -0.9360 & -0.4814 \\ -0.0433 & 0.3848 \end{bmatrix} \quad cm2(m,n) = \begin{bmatrix} -1.5389 & -1.4925 \\ 0.0473 & 3.3328 \\ 0.7827 & -0.8618 \\ 1.5578 & 1.0667 \\ 2.1851 & -0.2633 \\ 0.1228 & 1.2655 \\ -2.1081 & -1.0729 \\ -1.0485 & -1.9743 \end{bmatrix}$$

The covariance matrices of class1 (dots) and class2 (crosses), respectively, are

$$\boldsymbol{C}_1 = \frac{cm1(m,n)^T cm1(m,n)}{7} = \begin{bmatrix} 0.5434 & 0.3333 \\ 0.3333 & 0.3125 \end{bmatrix}$$

$$\boldsymbol{C}_2 = \frac{cm2(m,n)^T cm2(m,n)}{7} = \begin{bmatrix} 2.2490 & 1.0505 \\ 1.0505 & 3.1336 \end{bmatrix}$$

The covariance matrices are symmetric. The diagonal elements are the variances of the class vector, and the off-diagonal elements are covariances. The determinants of the matrices are 0.0588 and 5.9440. The inverses are

$$\begin{bmatrix} 5.3180 & -5.6709 \\ -5.6709 & 9.2470 \end{bmatrix} \quad \begin{bmatrix} 0.5272 & -0.1767 \\ -0.1767 & 0.3784 \end{bmatrix}$$

Let $p(\omega_1) = 0.9$ and $p(\omega_2) = 0.1$. The decision function for class1, $d_1(\boldsymbol{x})$, is

$$\log_e(0.9) - 0.5\log_e(0.0588)$$

$$-0.5\begin{bmatrix} (x_1 - 1.1167) & (x_2 - (-0.9616)) \end{bmatrix}\begin{bmatrix} 5.3180 & -5.6709 \\ -5.6709 & 9.2470 \end{bmatrix}\begin{bmatrix} (x_1 - 1.1167) \\ (x_2 - (-0.9616)) \end{bmatrix}$$

$$= -2.6590x_1^2 - 4.6235x_2^2 + 5.6709(x_1 + x_2) + 11.3919x_1 - 15.2249x_2 - 12.3692$$

For the test data, shown in Fig. 12.5b,

$$\begin{bmatrix} 1.7884 & -0.4345 \\ 1.5550 & -0.2817 \\ 0.3931 & 3.5806 \\ -0.6417 & -1.4612 \\ -2.0366 & 0.4679 \\ -1.9784 & -2.6335 \end{bmatrix}$$

the decision function yields

$$\{0.8353, 0.3535, -114.1108, -3.0821, -60.1389, -7.7393\}$$

The decision function for class2, $d_2(\boldsymbol{x})$, is

$$-0.2636x_1^2 - 0.1892x_2^2 + 0.1767(x_1 + x_2) - 0.7030x_1 + 0.5520x_2 - 3.8193$$

For the test data, the decision function yields

$$\{-6.3326, -5.7979, -4.3366, -4.5215, -3.4324, -5.3051\}$$

Comparing the decision function values, the fourth values indicate that the sample belongs to class1 instead of class2. There is only one incorrect classification. Let $p(\omega_1) = 0.1$ and $p(\omega_2) = 0.9$. The decision function for class1, $d_1(\boldsymbol{x})$, is

$$-2.6590x_1^2 - 4.6235x_2^2 + 5.6709(x_1 + x_2) + 11.3919x_1 - 15.2249x_2 - 14.5665$$

For the test data, the decision function yields

$$\{-1.3620, -1.8437, -116.3081, -5.2794, -62.3361, -9.9366\}$$

The decision function for class2, $d_2(\boldsymbol{x})$, is

$$-0.2636x_1^2 - 0.1892x_2^2 + 0.1767(x_1 + x_2) - 0.7030x_1 + 0.5520x_2 - 1.6221$$

For the test data, the decision function yields

$$\{-4.1354, -3.6007, -2.1394, -2.3243, -1.2352, -3.1079\}$$

The classification is correct. With changes in $p(\omega_i)$, the boundary between classes of data gets shifted.

12.5 *k*-Means Clustering

A cluster is a number of similar things gathered together. The image is analyzed to find clusters in the feature space. Once clusters have been found, then decision boundaries may be constructed to classify the objects similar to supervised classifiers. The problem is to find the number of clusters, k, based on some given features.

Consider the image shown in Fig. 12.6a. It consists of four objects each one with distinct gray levels. The background is white. In general, no knowledge of the number of objects is available. While the objects can be segmented using the histogram, let us try the clustering approach.

- Let us divide the gray levels into two ranges, 0–128 and 129–254. The gray level of the background is 255. Gather the pixels into two clusters using the two ranges

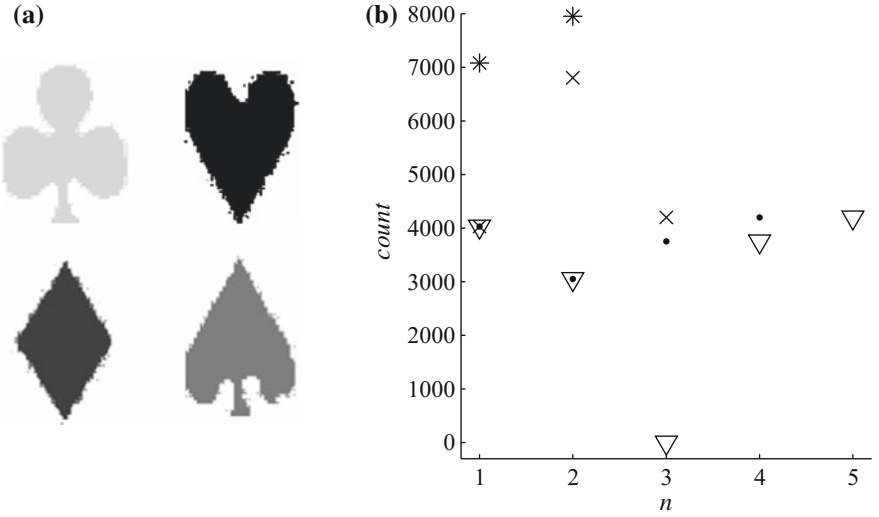


Fig. 12.6 **a** A 256 × 256 gray-level image; **b** *k*-means clustering of the four objects

of the gray levels. The number of pixels, {7077, 7951}, in the two clusters is shown in Fig. 12.6b by the symbol ∗.

- Let us divide the gray levels into three ranges, 0–85, 86–170, and 171–254. Gather the pixels into three clusters using the three ranges of the gray levels. The number of pixels, {4028, 6803, 4197}, in the three clusters is shown in Fig. 12.6b by the symbol ×.
- Gather the pixels into four clusters using the four ranges of the gray levels, 0–64, 65–128, 129–192, and 193–254. The number of pixels, {4028, 3049, 3754, 4197}, in the four clusters is shown in Fig. 12.6b by dots.
- Gather the pixels into five clusters using the five ranges of the gray levels, 0 to 51, 52 to 102, 103 to 153, 154 to 204, and 205 to 254. The number of pixels, {4028, 3049, 0, 3754, 4197}, in the five clusters is shown in Fig. 12.6b by the symbol ▽.

With four and five clusters, we get the same number of pixels in four clusters and zero pixels in the fifth indicating the convergence of the algorithm. While the clusters are identified, in order to classify each object as club, spade, heart, or diamond, we have to use known features as in supervised algorithms. This simple example is presented just to illustrate the basics of clustering algorithms. The flowchart of a typical clustering algorithm is shown in Fig. 12.7. The input image is decomposed into some number of clusters based on some similarity measures. Then, a number of iterations of the algorithm are executed with increased number of clusters until the algorithm converges or for some number of fixed iterations. After termination, the output, using additional features, contains the required classification.

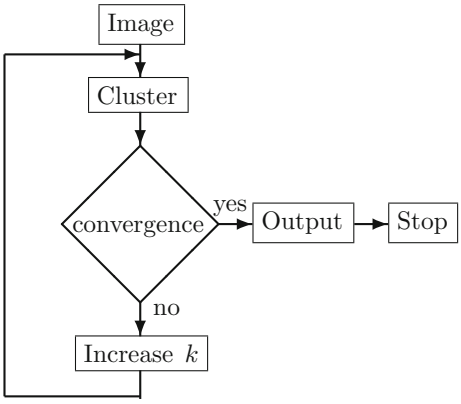


Fig. 12.7 Flowchart of the k-means clustering algorithm

12.6 Summary

- Given a set of features of an object, comparing that with those in the database and assigning it to its proper class is classification.
- There are two main types of classification: (i) supervised classification and (ii) unsupervised classification.
- In supervised classification, typical methods used are minimum distance, k-nearest neighbors, decision trees, and statistical (based on probability distribution models). The decision is prior.
- In unsupervised classification, the data is partitioned into groups by clustering. Unknown, but distinct set of feature classes are generated. The decision is posterior.
- In the k-nearest neighbors classifier method, the feature set of a test object is compared with the reference set, and the test object is assigned to the class whose features differ, with respect to some measure, by the least from that of the test object.
- In the minimum-distance-to-mean classifier, the means of a set of features of a class characterize the class. A new sample is assigned to a class if the distance between the means of the reference set of that class and that of the new sample is minimum.
- The classification of an object is based on discriminant functions. Discriminant functions have to be found such that the discriminant function corresponding to an unclassified feature vector will yield a value that is greater than those of the functions to which it does not belong.
- In the decision tree classification approach, the feature space is split into unique regions sequentially. A decision is arrived with out testing all classes.
- The Bayesian approach to statistical methods of classification is based not only on the set of samples but also on the pertinent prior information.

- In k-means clustering approach, the image is analyzed to find clusters in the feature space. Once clusters have been found, then the decision boundaries may be constructed to classify the objects similar to supervised classifiers.

Exercises

***12.1** The feature vectors of three classes are given. Using the 3-nearest neighbors classifier, classify the test vector {16, 16}.

Class	Feature 1	Feature 2
Class1	14	16
Class1	14	14
Class1	13	17
Class2	18	14
Class2	17	16
Class2	17	15
Class3	14	17
Class3	17	19
Class3	15	17

12.2 The feature vectors of three classes are given. Using the 3-nearest neighbors classifier, classify the test vector {16, 17}.

Class	Feature 1	Feature 2
Class1	15	16
Class1	16	19
Class1	15	17
Class2	17	14
Class2	18	15
Class2	17	15
Class3	14	17
Class3	17	18
Class3	14	16

12.3 The feature vectors of three classes are given. Using the 3-nearest neighbors classifier, classify the test vector {17, 16}.

Class	Feature 1	Feature 2
Class1	15	17
Class1	16	20
Class1	14	14
Class2	20	13
Class2	17	14
Class2	18	15
Class3	16	16
Class3	13	18
Class3	16	19

12.4 Let the mean feature vectors of three classes be $\{10, 10\}$, $\{-10, -10\}$, and $\{10, -10\}$. Let the test vectors be $\{12, -10\}$, $\{8, 9\}$, and $\{-9, -11\}$. Find the three discriminant functions of the three classes. Classify the test vectors.

***12.5** Let the mean feature vectors of three classes be $\{22, -20\}$, $\{20, 20\}$, and $\{-18, -19\}$. Let the test vectors be $\{23, -20\}$, $\{21, 18\}$, and $\{-18, -19\}$. Find the three discriminant functions of the three classes. Classify the test vectors.

12.6 Let the mean feature vectors of three classes be $\{15, -17\}$, $\{17, 15\}$, and $\{-16, 16\}$. Let the test vectors be $\{18, -16\}$, $\{18, 14\}$, and $\{-16, 16\}$. Find the three discriminant functions of the three classes. Classify the test vectors.

12.7 Draw the decision tree flowchart for classifying the five objects.

Object	F	O	U	R	S
Area	750	964	647	1084	658
Form	132	484	108	233	116

***12.8** Draw the decision tree flowchart for classifying the five objects.

Object	5	6	7	8	9
Area	1199	1245	822	1478	1229
Form	168	327	205	445	327

12.9 Draw the decision tree flowchart for classifying the five objects.

Object	0	1	2	3	4
Area	1419	880	1090	1023	1118
Form	576	273	183	176	402

12.10 Samples of training data of two classes, $c1(m,n)$ and $c2(m,n)$, are

$$c1(m,n) = \begin{bmatrix} 0.7124 & -1.0637 \\ 1.0219 & -1.3503 \\ 1.0487 & -0.6465 \\ 1.7837 & -0.5444 \\ 2.4486 & -0.3089 \\ 1.4430 & -0.6230 \\ 0.8010 & -0.9970 \\ 1.5931 & -1.0655 \end{bmatrix} \qquad c2(m,n) = \begin{bmatrix} -1.2296 & 0.7281 \\ -1.2066 & 0.6124 \\ -1.7524 & 3.1638 \\ 1.3789 & 0.6478 \\ -2.2385 & -0.5050 \\ -1.6842 & 3.2676 \\ -2.0069 & 2.7461 \\ -2.6606 & 0.6753 \end{bmatrix}$$

Classify the test samples $t(m,n)$ using Bayes classification.

$$t(m,n) = \begin{bmatrix} 0.4804 & -1.2375 \\ 1.0979 & -0.8694 \\ -1.7471 & 0.1364 \\ -0.8781 & 0.2799 \\ -0.9416 & 0.6194 \\ -2.9676 & 1.2279 \end{bmatrix}$$

with
(i) $p(\omega_1) = 0.9$ and $p(\omega_2) = 0.1$.
(ii) $p(\omega_1) = 0.1$ and $p(\omega_2) = 0.9$.

12.11 Samples of training data of two classes, $c1(m,n)$ and $c2(m,n)$, are

$$c1(m,n) = \begin{bmatrix} -1.3587 & -1.7484 \\ 1.5513 & -0.9966 \\ -1.0799 & -1.7673 \\ -1.2003 & -1.9427 \\ 1.0758 & -0.6775 \\ 0.1002 & -1.2125 \\ 1.3904 & -0.5126 \\ 1.6422 & -0.7607 \end{bmatrix} \qquad c2(m,n) = \begin{bmatrix} -1.2082 & -0.4353 \\ 0.4252 & 0.3338 \\ -4.0033 & 1.1938 \\ -1.7136 & 0.5872 \\ -2.7969 & 1.4268 \\ -1.5411 & 1.5656 \\ -0.0826 & -0.3152 \\ 0.1678 & 0.7499 \end{bmatrix}$$

Classify the test samples $t(m,n)$ using Bayes classification.

$$t(m,n) = \begin{bmatrix} -2.5374 & -2.4336 \\ 1.7707 & -0.8282 \\ -1.7326 & 2.1029 \\ -3.5297 & 1.9491 \\ -1.3760 & -0.9096 \\ -2.2018 & 1.1494 \end{bmatrix}$$

with
$p(\omega_1) = 0.5$ and $p(\omega_2) = 0.5$.

***12.12** Samples of training data of two classes, $c1(m, n)$ and $c2(m, n)$, are

$$c1(m,n) = \begin{bmatrix} 2.6140 & -0.5545 \\ 0.5164 & -1.0168 \\ 0.9973 & -1.4846 \\ 1.8727 & -0.5267 \\ 1.1421 & -0.6542 \\ 2.3328 & -0.3331 \\ 1.9811 & -0.2564 \\ 1.2766 & -0.1643 \end{bmatrix} \qquad c2(m,n) = \begin{bmatrix} 0.3066 & 1.8237 \\ -0.6225 & -0.4234 \\ -0.0926 & 1.0912 \\ -0.3983 & 1.8489 \\ -2.8593 & -0.9255 \\ -1.5889 & 1.4916 \\ 0.7320 & 0.7428 \\ -1.0616 & -0.3287 \end{bmatrix}$$

Classify the test samples $t(m, n)$ using Bayes classification.

$$t(m,n) = \begin{bmatrix} 1.5646 & -1.3982 \\ 0.2250 & -1.5204 \\ -1.8283 & -1.3793 \\ -2.0365 & 0.0497 \\ -2.6668 & 0.3246 \\ -0.3692 & 0.8388 \end{bmatrix}$$

with
$p(\omega_1) = 0.4$ and $p(\omega_2) = 0.6$.

Chapter 13
Image Compression

Abstract Image compression is as essential as its processing, since images require large amounts of memory and, in their original form, it is difficult to transmit and store them. There are two types of image compression: (i) lossless and (ii) lossy. In lossless compression, the original image can be reconstructed exactly from its compressed version. Lossy compression is based on the fact that the magnitude of the frequency components of typical images decreases with increasing frequency. Emphasis is given to using the DWT, since it is a part of the current image compression standard.

Image compression is essential due to the widespread use of the Internet and other media. It makes the transmission and storage of images efficient. For example, most of the images naturally occur in a continuous form. However, images are converted to digital form and processed for efficiency. Due to the sampling theorem, a finite number of samples over a finite time interval are adequate to represent a signal accurately. In a similar way, most images carry information with some redundancy. They have to be stored and transmitted in a compressed form for efficiency. In lossy compression, it has to be ensured that image fidelity is maintained to the required level. Conversion of images in one form to another, to suit the requirement of processing, is common in image processing. Along with analog-to-digital conversion and transformation such as Fourier, image compression is a part of image processing. In the conversion from one form to another, it has to be ensured that image quality is maintained to the required level.

Same information can be represented in different ways. For example, number 7 can be coded in ASCII form using 8 bits or in binary form (111) with 3 bits. Both forms are required in different applications. For efficient storage and transmission of images in particular, the compressed form is mandatory. The number of bits required to represent typical binary, gray, and color images is given in Chap. 2. Let us say we have to transmit a number from 0 to 7 and, most of the time, it is in the range 0–3. Then, with the assumption that a number in the range 4–7 only is coded using 3 bits and the other numbers coded with 2 bits, we get efficient transmission and storage.

D. Sundararajan, *Digital Image Processing*, DOI 10.1007/978-981-10-6113-4_13

The objective in image compression is that the data has to be reduced as much as possible while retaining the information the image carries.

There are two basic types of compression. In lossless compression, the redundancy in the image is reduced so that the image can be reconstructed exactly. This type of compression gives relatively small compression ratios but is essential for compression of documents such as legal and medical records. In lossy compression, exact reconstruction is not possible but a higher compression ratio is obtained. The degradation of the image is limited to acceptable levels.

There are three basic types of redundancies in image data. Typically, 256 gray levels are used in representing an image. In the case that a smaller number of gray levels are adequate to represent an image, a reduced number of gray levels and, hence, a reduced number of bits can be used. Further, not all the gray levels occur with the same probability. Instead of using fixed number of bits to represent all the gray levels, we can use less number of bits to gray levels occurring frequently. Coding redundancy exists in using a fixed number of bits to represent all the gray levels, which can be exploited to compress images.

Most of the time, the gray levels of adjacent pixels are same or nearly the same. This type is called interpixel redundancy. As pixel values are correlated, the value of a pixel can be predicted from that of its neighbor with reasonable accuracy. This process requires a smaller number of bits to represent an image than fixed-length coding.

Certain information, such as the background, is not important for visual purposes. Therefore, those pixels can be coded using a smaller number of bits. The response of a human eye with respect to quantization levels and frequency is limited. This type of redundancy is due to irrelevant data.

Some of the measures of the effectiveness of the compression algorithms are defined. The compression ratio is defined as

$$C = \frac{n}{n_c}$$

where n is the bits per pixel in the uncompressed image and n_c is that in the compressed image. A higher C indicates better compression.

The root-mean-square error ϵ is defined, for a $N \times N$ image $x(m, n)$, as

$$\epsilon = \sqrt{\frac{1}{N^2} \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} (\hat{x}(m, n) - x(m, n))^2}$$

where $\hat{x}(m, n)$ is the reconstructed image from its compressed version. The error ϵ must be zero for lossless compression and as low as possible for lossy compression.

Another quality measure of the reconstructed image is the signal-to-noise ratio, expressed in decibels and defined as

$$\text{SNR} = 10\log_{10}\left(\frac{\sum_{m=0}^{N-1}\sum_{n=0}^{N-1}\hat{x}^2(m,n)}{\sum_{m=0}^{N-1}\sum_{n=0}^{N-1}(x(m,n)-\hat{x}(m,n))^2}\right)$$

Compute the signal-to-noise ratio in decibels of the input $x(n)$ and its approximation $\hat{x}(n)$.

$$x(n) = \{10, 12\}, \quad \hat{x}(n) = \{10.1, 11.8\}$$

SNR = 36.835 dB.

13.1 Lossless Compression

For data, such as legal or medical, to be reconstructed exactly as the input, from its compressed version, lossless compression is used.

13.1.1 Huffman Coding

Instead of using the same number of bits for all the values of the image to be compressed, the idea is to represent highly probable values by a small number of bits and lowly probable values by relatively more number of bits. The steps of the Huffman coding algorithm are:

1. Find the probabilities of the occurrence of each value of the image.
2. Starting from the lowest probability, pair the two of the lowest probabilities to get a binary tree.
3. Assign 0 to leaf to the left side of a branch and 1 to that in the right side or vice versa.
4. Repeat steps 2 and 3, until all the values are placed.
5. Read the codes from the root of the tree to the leaf node where the value is placed.

This coding is popular and optimum, when each character is coded individually, giving the highest compression ratio.

Consider coding the 4×4 image

13	14	2	14
8	2	2	8
15	15	2	15
15	8	13	2

A list of unique values of the image and the relative frequencies is created, as shown in Table 13.1. Then, a binary tree is constructed with the unique values as its leaves and arranged with the probabilities sorted. The higher the frequency of a value, the more closer is its location to the root. A right leaf is assigned the bit 0, and the left leaf is assigned the bit 1. This choice is arbitrary and could be reversed. The position of a value in the tree indicates its code. We simply read the bits from the root to the location of the value. The higher the frequency of a value, the fewer the bits used to encode it. Consider the binary tree shown in Fig. 13.1a. A two-leaf tree is constructed from the two lowest frequencies (2/16 and 2/16). The sum 4/16 is placed at the node connecting the two leaves. Now, a left leaf is created with frequency 3/16. A two-leaf tree is constructed from the two frequencies (3/16 and 4/16). The sum 7/16 is placed at the node connecting the two leaves. One more left leaf with frequency 4/16 completes the right side of the tree. The sum 11/16 is placed at the node connecting the two leaves. The most frequently occurring number 2 is placed in the leaf that is located to the left of the root. Therefore, its code is 1. Number 15 is placed in the first left leaf to the right side of the root. Therefore, its code is 01. The code for the example image is

{0001 0000 1 0000 001 1 1 001 01 01 1 01 01 001 0001 1}

Table 13.1 Assignment of Huffman codes

Character	Frequency	Relative frequency	Code A	Total bits	Code B	Total bits
2	5	$\frac{5}{16}$	1	5	11	10
8	3	$\frac{3}{16}$	001	9	01	6
13	2	$\frac{2}{16}$	0001	8	001	6
14	2	$\frac{2}{16}$	0000	8	000	6
15	4	$\frac{4}{16}$	01	8	10	8

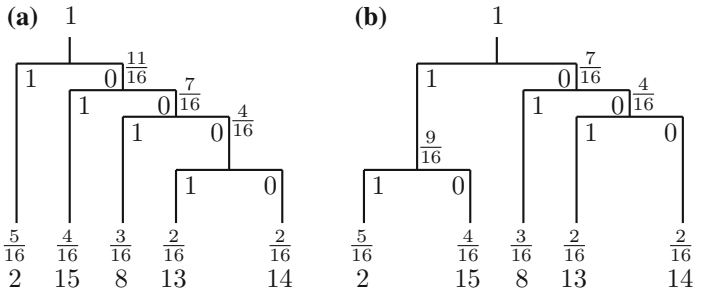


Fig. 13.1 Huffman coding trees

The total number of bits to code the image is $5+9+8+8+8 = 38$ bits compared with $(16)(8) = 128$ bits without coding. Therefore, the average number of bits required to code a character is reduced to $38/16 = 2.375$ bits from 8. For coding a large number of characters, the Huffman coding algorithm is not trivial. If precomputed tables are available, near optimal coding can be achieved for a class of images.

The decoding is carried out just by looking at the first few characters until a code corresponding to a value is found. The first 4 bits are 0001 and it corresponds to 13. The next 4 bits are 0000 and it corresponds to 14. Next, we look at 1 and it corresponds to 2. It goes on until all the bits are decoded.

An alternate binary tree is shown in Fig. 13.1b. Total number of bits to code the image is $10+6+6+6+8 = 36$ bits. Therefore, the average number of bits required to code a character is reduced to $36/16 = 2.25$ bits from 8.

13.1.1.1 Entropy

A measure of the effectiveness of compression algorithms is the average information of the image, called the entropy. It is the theoretical limit of the minimum number of bits per pixel (bpp) for compressing an image by a lossless compression algorithm. The total number of bits used to represent an image divided by the total number of pixels is bpp. Let the number of distinct values in an image be N and the frequency of their occurrence be

$$f_1, f_2, \ldots, f_N$$

Entropy is defined as

$$E = \sum_{k=1}^{N} p(f_k) \log_2 \left(\frac{1}{p(f_k)} \right)$$

where $p(f_k)$ is the probability of occurrence of f_k. The term $\log_2(1/p(f_k))$ gives the number of bits required to represent $1/p(f_k)$. By multiplying this factor with $p(f_k)$, we get the bpp required to code f_k. The sum of bpp for all the distinct values yields the bpp to code the image. This equation can be equivalently written as

$$E = -\sum_{k=1}^{N} p(f_k) \log_2(p(f_k))$$

Let $N = 4$ be number of values in a set with each value being distinct. The probability of their occurrence is

$$\{1, 1, 1, 1\}/4$$

Then,

$$E = -(4)\frac{1}{4}\log_2(\frac{1}{4}) = (4)\frac{1}{4}\log_2(4) = 2$$

With each value distinct, the number of bits required is 2 (which is required in binary encoding of the numbers). Therefore, there is no compression. The variation of the probabilities of the occurrences of pixel values of an image makes compression possible. With all the values the same, $E = 0$. For all other cases, entropy varies between 0 and $\log_2(N)$. In typical images, a considerable amount of repetition of values occur. For the example 4×4 image,

$$E = -(\frac{5}{16}\log_2(\frac{5}{16}) + \frac{3}{16}\log_2(\frac{3}{16}) + (2)\frac{2}{16}\log_2(\frac{2}{16}) + \frac{4}{16}\log_2(\frac{4}{16})) = 2.2272$$

The actual value of 2.25 bpp is quite close with the ideal value 2.2272.

13.1.2 Run-Length Encoding

A gray-level image can be decomposed into a set of binary images. Each binary image can be considered as sequences of alternating strings of 1 s and 0s. A sequence of 0 s and 1 s can be encoded by the number of their occurrences. Consider the 8×8 binary image

0	1	1	1	0	0	1	0
0	1	1	1	0	1	1	0
0	1	1	0	0	0	1	0
1	1	1	1	0	0	1	0
0	1	1	1	0	0	1	0
0	1	1	1	1	1	1	0
1	1	1	1	0	0	1	1
0	0	0	1	0	0	1	0

One method starts with the number of zeros. Each line is coded separately. The first line is encoded as (13211), since the first bit is zero, followed by 3 1s, 2 0s, 1 1s, and 1 0s. The other lines are encoded as (13121), (12311), (04211), (13211), (161), (0422), and (31211).

In another method, the starting position of sequences of 1s and their lengths are recorded. That is, for each sequence of 1s, a pair of numbers is used to encode. The first line is encoded as (2371), since the first sequence of 1s starts at second position with length 3 and the next sequence starts at seventh position with length 1. The other lines are encoded as (2362), (2271), (1471), (2371), (26), (1472), and (4171).

Gray-level images can be encoded by breaking them into bit planes first and then encoding each of the planes individually. For example,

$$\begin{bmatrix} 1 & 5 & 12 & 5 \\ 14 & 7 & 11 & 9 \\ 1 & 4 & 15 & 6 \\ 7 & 12 & 7 & 11 \end{bmatrix} = 2^3 \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} + 2^2 \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} + 2 \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix} + \begin{bmatrix} 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{bmatrix}$$

13.1.3 Lossless Predictive Coding

Consider the pixel values of a 4×4 portion of a 8-bit gray-level image

$$\begin{bmatrix} 246 & 243 & 243 & 242 \\ 245 & 243 & 242 & 241 \\ 245 & 242 & 242 & 241 \\ 245 & 242 & 242 & 241 \end{bmatrix}$$

The maximum difference of adjacent pixel values is 3. This pattern is typical in significant part of an image. The difference between adjacent pixel values in an image is typically in the neighborhood of zero. Therefore, if the pixels are represented by the difference (the new information) between the actual and predicted values, then the redundancy in the image is increased and the entropy is reduced. The number of bits per pixel can be reduced using the difference values of neighboring pixels.

Figure 13.2a shows an encoder. It consists of a FIR filter with one coefficient, $h(0)$. A filter can have more coefficients. The rounded output of the filter is the predicted value. The rounding operation returns the nearest integer of a number. This value is subtracted from the current pixel value to get the predicted code. The code is passed through symbol encoders, such as Huffman encoder, to get the compressed signal. The prediction error is the difference between the current input pixel $x(n)$ and the rounded output of the predictor, $\hat{x}(n)$.

$$e(n) = x(n) - \hat{x}(n)$$

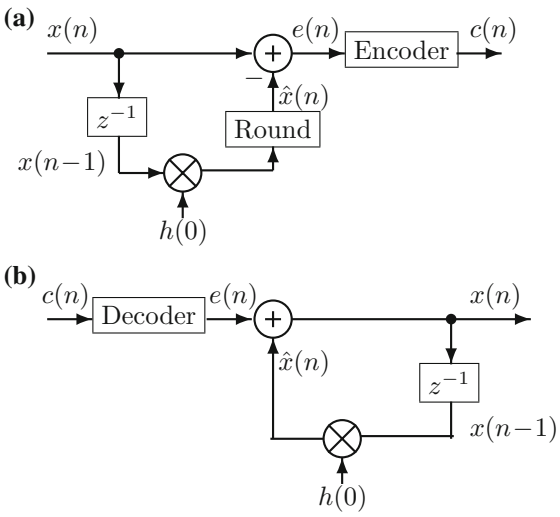


Fig. 13.2 Lossless predictive coding

where

$$\hat{x}(n) = \text{round}\left(\sum_{k=0}^{N-1} h(k)x(n-k+1)\right)$$

The filter coefficients are $h(k)$. The decoder, shown in Fig. 13.2b, decodes the coded image to its original form.

$$x(n) = e(n) + \hat{x}(n)$$

Each row of the image is coded separately.

$$\hat{x}(m,n) = \text{round}\left(\sum_{k=0}^{N-1} h(k)x(m,n-k+1)\right)$$

As the output depends on some number of past input pixels, the process does not start from the first pixel.

Consider the 4×4 image

51	50	52	47
53	44	39	48
40	63	131	212
114	128	154	155

Let us use a filter with one coefficient with value 1. Then,

$$\hat{x}(m,n) = \text{round}\,(x(m,n-1)) \quad \text{and} \quad e(m,n) = x(m,n) - \hat{x}(m,n) = x(m,n) - x(m,n-1)$$

except for the first pixels of the rows. The predictive coding representation of the image is

51	−1	2	−5
53	−9	−5	9
40	23	68	81
114	14	26	1

The first column values remain the same. For the first row, the second, third, and fourth column values are $50 - 51 = -1$, $52 - 50 = 2$, and $47 - 52 = -5$, respectively. The values of the other three rows are found similarly. For decompressing of the first row, the second, third, and fourth column values are $51 - 1 = 50$, $50 + 2 = 52$, and $52 - 5 = 47$, respectively.

In the input image, there are 16 symbols each with probability 1/16. Therefore, the entropy is

$$-(16(1/16)\log 2(1/16)) = 4$$

In the coded image, there are 15 symbols (-5 repeats twice), 14 with probability 1/16 and one with 2/16. Therefore, the entropy is

$$-(14(1/16)\log 2(1/16) + (2/16)\log 2(2/16)) = 31/8 = 3.8750$$

To code the difference, we need one bit more, and usually, the independent symbols in the coded image are more than that of the input image. However, due to the density of the histogram at the center, the coded image has a lower entropy, as presented in the next example.

The histogram of a 256 × 256 image, shown in Fig. 13.3a, is fairly spread out. The histogram of its predictive coding representation using a filter with just one coefficient with value 1 is shown in Fig. 13.3b. Although the range of values is larger than that of the image, the histogram is densely populated in the neighborhood of zero. Therefore, the entropy of coded image, 6.2297, is smaller than that, 7.4371, of the input image. The lower entropy will result in a smaller bpp when the coded image is passed through a symbol encoder.

13.1.4 Arithmetic Coding

In arithmetic coding, a group of symbols is coded rather than each one individually as in Huffman coding. The advantages are that the coding is more efficient for longer sequences and no table is required. As it is more complex than the Huffman coding, we have given MATLAB programs to describe it. Further, although it is more efficient for longer sequences, we use short sequences in the examples for ease of understanding the algorithm.

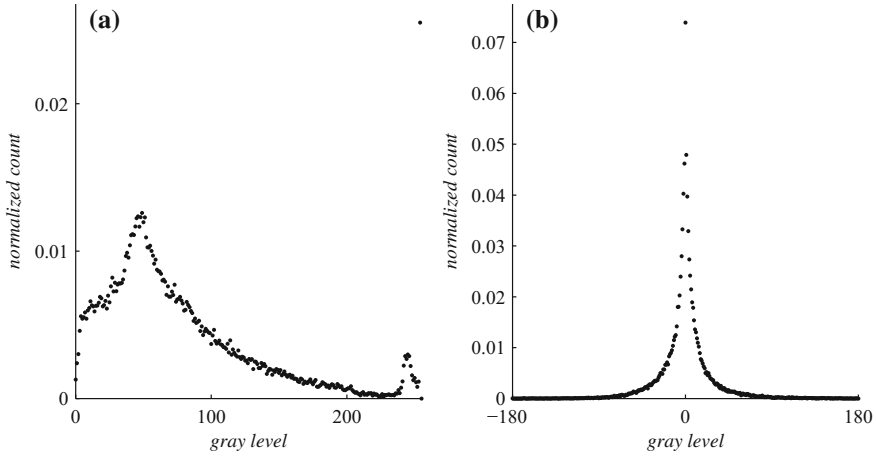


Fig. 13.3 **a** Histogram of an image and **b** the histogram of its representation by the linear prediction error

Assume that we have to code {22, 23, 32, 33}. Let the probability of occurrence of the four numbers be equal. Then, a possible code is {00, 01, 10, 11}. Of course, with longer numbers and the probability of occurrence varying, the number of bits to code increases. In order to make it practically efficient with finite precision, rescaling has to be done during the coding process. Further, rounding operation is required to make the code accurate. Rescaling and coding are the key steps in the algorithm.

As in Huffman coding, the basic principle is to assign a code to each symbol the length of which is inversely proportional to its probability of occurrence. The assignment of the code is based on the cumulative distribution function of the symbols. Histogram equalization is also based on the cumulative distribution function. A higher probability of a gray level gets a longer range of gray levels in the equalized histogram. In arithmetic coding, a higher probability of a symbol gets a shorter code. With infinite precision, it is possible to assign a distinct code to each distinct sequence of symbols. However, the number of bits to represent a number is limited in digital image processing. For example, the range of numbers is 0–255 with 8 bits. In order to code efficiently with finite precision, arithmetic coding relies on rescaling of the numbers. For example, to examine a decimal number 125 (01111101 in binary), with 8-bit representation, we test its MSB. As that bit is 0, it is determined that it is in the lower-half of the range 0–255. Once that bit is stored, we can rescale the number 125 to get 11111010 and determine that 125 is in the upper-half of the range 0–127 by testing its MSB.

A main program and two functions are given. The actual codes can be downloaded from the Web site of the book. The main program calls an encoding function to get the code for the given sequence. Then, a decoding function is called with the generated code to get back the input sequence. Just a few runs of the program are enough to get used to the basics of this coding.

```
% Program to find the arithmetic code of a sequence.
% set x, the sequence of integers and
% count, the number of times each integer occurs.
% outputs the binary code and the wordlength used.
% decodes the binary code to get back the input
clear
x=[9 9 1 9 9 4 8 1] % input sequence
lseq=length(x); % length of the sequence
y=unique(x) % list of unique symbols
count=[10 2 5 9] % number of times each symbol occurs
c_count=cumsum([0 count]); % cumulative count
[code,N]=a_encode(x,y,c_count) % calling encoding function
xr=a_decode(N,y,c_count,code,lseq) % calling decoding function
```

```
% encoding function to generate the binary code
function [code_b,N] = a_encode(seq,sym,c_count)
total=c_count(end); % total count
N=round(log2(4*total)); % word length
```

```
msb=N; % position of the msb
msbm1=N-1; % position of the second msb
l=0; % initial lower limit of the interval
u=2^N-1; % initial upper limit of the interval
flag=0; % flag
cod=[]; % initialize code accumulator
for k=1:length(seq) % one loop for each symbol
  temp=l;
  num=seq(k); % read the new number
  index=find( sym == num); % find the index of the symbol
  l=l+floor((u-l+1)*c_count(index)/total); % lower limit
  u=temp+floor((u-temp+1)*c_count(index+1)/total)-1; % upper limit
  lb=de2bi(l,N); % convert l to binary
  ub=de2bi(u,N); % convert u to binary
  while (lb(msb) == ub(msb) | (lb(msbm1) == 1 & ub(msbm1) == 0))
% loop if lb(msb) = ub(msb) or lb(msbm1) == 1 & ub(msbm1) == 0
    if (lb(msb) == ub(msb)) % if msbs of l and u are equal
      b=ub(msb); % msb of ub is the code bit
      cod=[cod b]; % accumulate the code
      lb=circshift(lb,[0,1]);lb(1)=0;
% left shift lb by one position and then set lsb to 0
      ub=circshift(ub,[0,1]);ub(1)=1;
% left shift ub by one position and then set lsb to 1
      l=bi2de(lb); % convert lb to decimal
      u=bi2de(ub); % convert ub to decimal
      while flag > 0
        cod=[cod ~b];
% code bit is the complement of the last code bit
        flag=flag - 1; % decrement the flag
      end
    end % if
  % if  lb(msbm1) == 1 & ub(msbm1) == 0 holds
    if (lb(msbm1) == 1 & ub(msbm1) == 0)
      lb(msbm1)=0; % complement lb(msbm1)
      ub(msbm1)=1; % complement ub(msbm1)
      lb=circshift(lb,[0,1]);lb(1)=0;
      ub=circshift(ub,[0,1]);ub(1)=1;
      l=bi2de(lb);
      u=bi2de(ub);
      flag=flag + 1; % increment the flag
    end
  end % while
end % for
if flag ==  appending the tag to the code
  code_b= [cod lb(N:-1:1)];
else
  code_b= [cod lb(N) ones(1,flag) lb(N-1:-1:1)];
end
code_b=code_b(length(code_b):-1:1);



% decoding function to reconstruct the input sequence
```

```
function [decod] =a_decode(N,sym,c_count,code_b,lseq)
%
msb=N; % position of the msb
msbm1=N-1; % position of the second msb
total=c_count(end); % length of the sequence
tag=bi2de(code_b(end-N+1:end)); % most significant N bits
l=0; % initial lower limit
u=2^N-1; % initial upper limit
decod=[]; % symbol accumulator
ps=1;
for pp=1:lseq
  tem=floor(((tag-l+1)*total-1)/(u-l+1));
  k=1;
  while (tem >= c_count(k)); % symbol decode
    k=k+1;
  end
  x=sym(k-1); % decoded symbol
  decod=[decod sym(k-1)]; % accumulate the symbols
  temp=l;
  l=l+floor((u-l+1)*c_count(k-1)/total);
  u=temp+floor((u-temp+1)*c_count(k)/total)-1;
  lb=de2bi(l,N);
  ub=de2bi(u,N);
  while (lb(msb) == ub(msb) | (lb(msbm1) == 1 & ub(msbm1) == 0))
    if (lb(msb) == ub(msb))
      tb=de2bi(tag,N);
      lb=circshift(lb,[0,1]);lb(1)=0;
      ub=circshift(ub,[0,1]);ub(1)=1;
      tb=circshift(tb,[0,1]);tb(1)=code_b(end-N+1-ps);
% left shift the tag by 1 bit and read the next bit
% into the lsb from the binary code
      ps=ps+1;
      l=bi2de(lb);
      u=bi2de(ub);
      tag=bi2de(tb);
    end % if end
    if (lb(msbm1) == 1 & ub(msbm1) == 0)
      lb(msbm1)=0; % complement second msb of lb
      ub(msbm1)=1; % complement second msb of ub
      lb=circshift(lb,[0,1]);lb(1)=0;
      ub=circshift(ub,[0,1]);ub(1)=1;
      l=bi2de(lb);
      u=bi2de(ub);
      tb=de2bi(tag,N);
      tb=circshift(tb,[0,1]);tb(1)=code_b(end-N+1-ps);
      ps=ps+1;
      tb(msb)=~tb(msb);
      tag=bi2de(tb);
    end % 2nd if end
  end % while end
end % for end
```

Example 13.1 Depending on the probability of occurrence of the symbols, the word length has to be fixed. Let us say that our sequence length is 1. Let there be three symbols $\{1, 2, 3\}$ and number of occurrences of the symbols, respectively, be $\{2, 1, 1\}$ in a sequence of length 4. The cumulative count is defined as $\{c\_count(n), n = 1, 2, 3, 4\} = \{0, 2, 3, 4\}$ in a sequence of length 4. The interval for the representation of the input sequence has to be four times that of the total count $total = 4$. The interval has to be at least $4 \times 4 = 16$, and the wordlength is 4. That is, $2^4 = 16$. Wordlength has to be a power of 2. The index of the MSB is 4, that of the second MSB is 3, and that of the LSB is 1. The initial lower limit of the interval is $l = 0$, and the upper limit is $u = 2^4 - 1 = 15$.

Encoding
Let the symbol to be coded is 3. This number is read, and u and l are updated.

$$l1 = l + \left\lfloor \frac{(u - l + 1) \times c\_count(3)}{total} \right\rfloor = 0 + \left\lfloor \frac{16 \times 3}{4} \right\rfloor = 12 = (1100)_2$$

$$u1 = 0 + \left\lfloor \frac{(u - l + 1) \times c\_count(4)}{total} \right\rfloor - 1 = 0 + \left\lfloor \frac{16 \times 4}{4} \right\rfloor - 1 = 15 = (1111)_2$$

If both the MSBs are same, that bit is stored in $code$. Therefore, $code = \{1\}$. Now, both $u1$ and $l1$ are shifted left by 1 bit and the LSBs are set 1 and 0, respectively.

$$l1 = 8 = (1000)_2, \qquad u1 = 15 = (1111)_2$$

Since the MSBs are same, $code = \{11\}$. Shifting $u1$ and $l1$, we get

$$l1 = 0 = (0000)_2, \qquad u1 = 15 = (1111)_2$$

With these values, we exit from the $while$ loop and also exit from the for loop since there is no more symbol to be read. The last value of $l1$ is appended to the $code$ so that the final $code$ is $\{110000\}$.

Decoding
In encoding, we generate the code bits from the symbol with the lower and upper limits of the intervals. In decoding, we generate the symbol from the code bits with the same set of lower and upper limits of the intervals. It is just the reverse process. To decode, we read the first 4 bits of the code into $tag = (1100)_2 = 12$. Using this and the lower and upper limits (0 and 15), we compute the cumulative count of the symbol as

$$tem = \left\lfloor \frac{(tag - l + 1) \times total - 1}{u - l + 1} \right\rfloor = \left\lfloor \frac{13 \times 4 - 1}{16} \right\rfloor = 3 = (0011)_2$$

Starting from the first value, we find a value in c_count so that it is greater than or equal to tem in the $while$ loop. The index of this value is 3. The decoded symbol, 3,

is with this index in sym list. Now, we update the limits.

$$l1 = 0 + \left\lfloor \frac{16 \times 3}{4} \right\rfloor = 12 = (1100)_2$$

$$u1 = 0 + \left\lfloor \frac{16 \times 4}{4} \right\rfloor - 1 = 15 = (1111)_2$$

As both MSBs are equal, $u1, l1$, and tag are shifted and the LSB of the tag is replaced by the next bit in the $code$ to get

$$l1 = 8 = (1000)_2, u1 = 15 = (1111)_2, tag = 8 = (1000)_2$$

Since the MSB and LSB are equal, we go back to the $while$ loop and update $u1, l1$, and tag.

$$l1 = 0 = (0000)_2, u1 = 15 = (1111)_2, tag = 0 = (0000)_2$$

Now, we go back to the for loop, and since there is no more symbol to be decoded, the execution is finished. ■

Example 13.2 The input sequence is

$$\{9, 9, 1, 9, 9, 4, 8, 1\}$$

There are four symbols $\{1, 4, 8, 9\}$. The number of occurrences of the symbols, in that order, is given as $\{10, 2, 5, 9\}$ in a sequence of length 26. The cumulative count is $\{0, 10, 12, 17, 26\}$. The range of the interval has to be at least $4 \times 26 = 104$, and the wordlength is 7. That is, $2^7 = 128$. Wordlength has to be a power of 2. The index of the MSB is 7, that of the second MSB is 6, and that of the LSB is 1. The initial lower limit of the interval is $l = 0$, and the upper limit is $u = 2^7 - 1 = 127$.

Encoding
loop 1. symbol 9

$$l1 = 0 + \left\lfloor \frac{128 \times 17}{26} \right\rfloor = 83 = (1010011)_2$$

$$u1 = 0 + \left\lfloor \frac{128 \times 26}{26} \right\rfloor - 1 = 127 = (1111111)_2$$

$$cod = \{1\}$$

$$l1 = 38 = (0100110)_2, \qquad u1 = 127 = (1111111)_2$$

loop 2. symbol 9

$$l2 = 38 + \left\lfloor \frac{90 \times 17}{26} \right\rfloor = 96 = (1100000)_2$$

$$u2 = 38 + \left\lfloor \frac{90 \times 26}{26} \right\rfloor - 1 = 127 = (1111111)_2$$

$$cod = \{11\}$$

$$l2 = 64 = (1000000)_2, \qquad u2 = 127 = (1111111)_2$$

$$cod = \{111\}$$

$$l2 = 0 = (0000000)_2, \qquad u2 = 127 = (1111111)_2$$

loop 3. symbol 1

$$l3 = 0 + \left\lfloor \frac{128 \times 0}{26} \right\rfloor = 0 = (0000000)_2$$

$$u3 = 0 + \left\lfloor \frac{128 \times 10}{26} \right\rfloor - 1 = 48 = (0110000)_2$$

$$cod = \{1110\}$$

$$l3 = 0 = (0000000)_2, \qquad u3 = 97 = (1100001)_2$$

loop 4. symbol 9

$$l4 = 0 + \left\lfloor \frac{98 \times 17}{26} \right\rfloor = 64 = (1000000)_2$$

$$u4 = 0 + \left\lfloor \frac{98 \times 26}{26} \right\rfloor - 1 = 97 = (1100001)_2$$

$$cod = \{11101\}$$

$$l4 = 0 = (0000000)_2, \qquad u4 = 67 = (1000011)_2$$

loop 5. symbol 9

$$l5 = 0 + \left\lfloor \frac{68 \times 17}{26} \right\rfloor = 44 = (0101100)_2$$

$$u5 = 0 + \left\lfloor \frac{68 \times 26}{26} \right\rfloor - 1 = 67 = (1000011)_2$$

$$l5 = 24 = (0011000)_2, \qquad u5 = 71 = (1000111)_2$$

$$flag = 1$$

loop 6. symbol 4

$$l6 = 24 + \left\lfloor \frac{48 \times 10}{26} \right\rfloor = 42 = (0101010)_2$$

$$u6 = 24 + \left\lfloor \frac{48 \times 12}{26} \right\rfloor - 1 = 45 = (0101101)_2$$

$$cod = \{111010\}$$

$$l6 = 84 = (1010100)_2, \qquad u6 = 91 = (1011011)_2$$

$$cod = \{1110101\}$$

$$cod = \{11101011\}$$

$$l6 = 40 = (0101000)_2, \qquad u6 = 55 = (0110111)_2$$

$$cod = \{111010110\}$$

$$l6 = 80 = (1010000)_2, \qquad u6 = 111 = (1101111)_2$$

$$cod = \{1110101101\}$$

$$l6 = 32 = (0100000)_2, \qquad u6 = 95 = (1011111)_2$$

$$l6 = 0 = (0000000)_2, \qquad u6 = 127 = (1111111)_2$$

$$flag = 1$$

loop 7. symbol 8

$$l7 = 0 + \left\lfloor \frac{128 \times 12}{26} \right\rfloor = 59 = (0111011)_2$$

$$u7 = 0 + \left\lfloor \frac{128 \times 17}{26} \right\rfloor - 1 = 82 = (1010010)_2$$

$$l7 = 54 = (0110110)_2, \qquad u7 = 101 = (1100101)_2$$

$$flag = 2$$

loop 8. symbol 1

$$l8 = 54 + \left\lfloor \frac{48 \times 0}{26} \right\rfloor = 54 = (0110110)_2$$

$$u8 = 54 + \left\lfloor \frac{48 \times 10}{26} \right\rfloor - 1 = 71 = (1000111)_2$$

$$l8 = 44 = (0101100)_2, \qquad u8 = 79 = (1001111)_2$$

$$flag = 3$$

$$l8 = 24 = (0011000)_2, \qquad u8 = 95 = (1011111)_2$$

$$flag = 4$$

The lower limit is $l8 = 0011000$. Since $flag = 4$, we insert 4 1s after the MSB of $l8$ to get the tag 01111011000 of 11 bits. This tag is appended to the $cod = 1110101101$ to get the final code of 21 bits, for the input sequence, as

$$\{111010110101111011000\}$$

Decoding

The first 7 bits of the tag is $(1110101)_2 = 117$. The initial limits are $l = 0$ and $u = 127$.

loop 1

$$tem = \left\lfloor \frac{118 \times 26 - 1}{128} \right\rfloor = 23, \qquad decod = \{9\}$$

$$l1 = 0 + \left\lfloor \frac{128 \times 17}{26} \right\rfloor = 83 = (1010011)_2$$

$$u1 = 0 + \left\lfloor \frac{128 \times 26}{26} \right\rfloor - 1 = 127 = (1111111)_2$$

$l1 = 38 = (0100110)_2, \qquad u1 = 127 = (1111111)_2, \qquad tag = 107 = (1101011)_2$

loop 2

$$tem = \left\lfloor \frac{70 \times 26 - 1}{90} \right\rfloor = 20, \qquad decod = \{9, 9\}$$

$$l2 = 38 + \left\lfloor \frac{90 \times 17}{26} \right\rfloor = 96 = (1100000)_2$$

$$u2 = 38 + \left\lfloor \frac{90 \times 26}{26} \right\rfloor - 1 = 127 = (1111111)_2$$

$$l2 = 64 = (1000000)_2, \qquad u2 = 127 = (1111111)_2, \qquad tag = 86 = (1010110)_2$$

$$l2 = 0 = (0000000)_2, \qquad u2 = 127 = (1111111)_2, \qquad tag = 45 = (0101101)_2$$

loop 3

$$tem = \left\lfloor \frac{46 \times 26 - 1}{128} \right\rfloor = 9, \qquad decod = \{9, 9, 1\}$$

$$l3 = 0 + \left\lfloor \frac{128 \times 0}{26} \right\rfloor = 0 = (0000000)_2$$

$$u3 = 0 + \left\lfloor \frac{128 \times 10}{26} \right\rfloor - 1 = 48 = (0110000)_2$$

$$l3 = 0 = (0000000)_2, \qquad u3 = 97 = (1100001)_2, \qquad tag = 90 = (1011010)_2$$

loop 4

$$tem = \left\lfloor \frac{91 \times 26 - 1}{98} \right\rfloor = 24, \qquad decod = \{9, 9, 1, 9\}$$

$$l4 = 0 + \left\lfloor \frac{98 \times 17}{26} \right\rfloor = 64 = (1000000)_2$$

$$u4 = 0 + \left\lfloor \frac{98 \times 26}{26} \right\rfloor - 1 = 97 = (1100001)_2$$

$$l4 = 0 = (0000000)_2, \qquad u4 = 67 = (1000011)_2, \qquad tag = 53 = (0110101)_2$$

loop 5

$$tem = \left\lfloor \frac{54 \times 26 - 1}{68} \right\rfloor = 20, \qquad decod = \{9, 9, 1, 9, 9\}$$

$$l5 = 0 + \left\lfloor \frac{68 \times 17}{26} \right\rfloor = 44 = (0101100)_2$$

$$u5 = 0 + \left\lfloor \frac{68 \times 26}{26} \right\rfloor - 1 = 67 = (1000011)_2$$

$$l5 = 24 = (0011000)_2, \quad u5 = 71 = (1000111)_2, \quad tag = 43 = (0101011)_2$$

loop 6

$$tem = \left\lfloor \frac{20 \times 26 - 1}{48} \right\rfloor = 10, \quad decod = \{9, 9, 1, 9, 9, 4\}$$

$$l6 = 24 + \left\lfloor \frac{48 \times 10}{26} \right\rfloor = 42 = (0101010)_2$$

$$u6 = 24 + \left\lfloor \frac{48 \times 12}{26} \right\rfloor - 1 = 45 = (0101101)_2$$

$$l6 = 84 = (1010100)_2, \quad u6 = 91 = (1011011)_2, \quad tag = 87 = (1010111)_2$$

$$l6 = 40 = (0101000)_2, \quad u6 = 55 = (0110111)_2, \quad tag = 47 = (0101111)_2$$

$$l6 = 80 = (1010000)_2, \quad u6 = 111 = (1101111)_2, \quad tag = 94 = (1011110)_2$$

$$l6 = 32 = (0100000)_2, \quad u6 = 95 = (1011111)_2, \quad tag = 61 = (0111101)_2$$

$$l6 = 0 = (0000000)_2, \quad u6 = 127 = (1111111)_2, \quad tag = 59 = (0111011)_2$$

loop 7

$$tem = \left\lfloor \frac{60 \times 26 - 1}{128} \right\rfloor = 12, \quad decod = \{9, 9, 1, 9, 9, 4, 8\}$$

$$l7 = 0 + \left\lfloor \frac{128 \times 12}{26} \right\rfloor = 59 = (0111011)_2$$

$$u7 = 0 + \left\lfloor \frac{128 \times 17}{26} \right\rfloor - 1 = 82 = (1010010)_2$$

$$l7 = 54 = (0110110)_2, \quad u7 = 101 = (1100101)_2, \quad tag = 54 = (0110110)_2$$

loop 8

$$tem = \left\lfloor \frac{1 \times 26 - 1}{48} \right\rfloor = 0, \qquad decod = \{9, 9, 1, 9, 9, 4, 8, 1\}$$

$$l8 = 54 + \left\lfloor \frac{48 \times 0}{26} \right\rfloor = 54 = (0110110)_2$$

$$u8 = 54 + \left\lfloor \frac{48 \times 10}{26} \right\rfloor - 1 = 71 = (1000111)_2$$

$$l8 = 44 = (0101100)_2, \qquad u8 = 79 = (1001111)_2, \qquad tag = 44 = (0101100)_2$$

$$l8 = 24 = (0011000)_2, \qquad u8 = 95 = (1011111)_2, \qquad tag = 24 = (0011000)_2$$

■

13.2 Transform-Domain Compression

While, in theory, the range of the frequencies of the components of a signal may be infinite, the frequencies which physical systems can generate are of finite order. Further, the magnitude of the high-frequency components decreases with increasing frequency. We have seen, in earlier chapters, the spectrum of several images in the center-zero format. In all cases, the magnitude is highest at low frequencies and lowest at high frequencies. The point is that the rate of convergence of the spectral coefficients of practical signals and images is quite high. This characteristic makes the compression of images practical. That is, the magnitudes of the low-frequency components can be coded with more number of bits and those of the high-frequency components can be coded with less number of bits. This is the basis of transform compression.

For a long time, images were decomposed into their individual frequency components using versions of Fourier analysis and compressed. The computational complexity of this approach is $O(N \log_2 N)$. Now, the DWT is part of the current compression standard and is widely used in practice. The point is that, for compression and some other applications, the decomposition of an image into their subband components turns out to be better. That is, the basis signals of the DWT are finite and composed of groups of frequency components of the spectrum. Decomposing signals into their individual components requires a complexity of $O(N \log_2 N)$, while the decomposition in terms of basis signals composed of groups of frequency components requires a complexity of $O(N)$. Further, the decomposition using the DWT reduces the storage requirement. In signal compression, the compressed version along with its storage format is required. Remember that, the signal has to be decompressed after storage or transmission. In terms of storage requirements also, the DWT

is better. For these reasons, the DWT is the most suitable transform for image compression. Therefore, the DWT is briefly presented in the next section. By taking the transform of an image, the image data gets uncorrelated. Then, the components those are negligible or unimportant can be coded with less number of bits or discarded altogether.

13.2.1 The Discrete Wavelet Transform

A signal is expressed as a linear combination of sinusoidal signals in Fourier analysis. In the DWT decomposition, a signal is expressed as a linear combination of finite duration basis signals, composed of continuous groups of frequency components (subbands) of the spectrum. For example, a signal is a combination of low- and high-frequency components. In the DWT, these components are separated similar to the separation of the individual frequency components in Fourier analysis. This decomposition is obtained by lowpass and highpass filtering of the signal. While Fourier analysis is of fundamental importance in signal and system analysis, the DWT is advantageous in analyzing nonstationary signals and in multiresolution analysis. One advantage is that the basis functions are local enabling the detection of the instant of occurrence of an event. Further, the basis functions are of different lengths giving multiresolution characteristic. The computational complexity of the DWT is $O(N)$.

The matrix formulation of the DWT is similar to that of the DFT with some differences. Consider the 4-point DWT of the sequence

$$\{x(0), x(1), x(2), x(3)\}$$

The 1-level (scale 1) 4-point Haar DWT is

$$\begin{bmatrix} X_\phi(1,0) \\ X_\phi(1,1) \\ X_\psi(1,0) \\ X_\psi(1,1) \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{bmatrix} \quad \text{or} \quad \boldsymbol{X} = \boldsymbol{W}_{4,1}\boldsymbol{x}$$

The basis functions are nonzero only for subintervals of the input sequence. This feature enables the shifting of a basis function leading to time-frequency representation of the input signal. The coefficients $X_\phi(1,0)$ and $X_\phi(1,1)$, indicated by ϕ and called the approximation coefficients, are obtained by averaging pairs of input signals. The coefficients $X_\psi(1,0)$ and $X_\psi(1,1)$, indicated by ψ and called the detail coefficients, are obtained by differencing pairs of input signals. The averaging operation is lowpass filtering, and differencing is highpass filtering. The spectrum of a sequence is split into a low-frequency subband and a high-frequency subband. The spectral range is decomposed into

$$(0 - \frac{\pi}{2}) \text{ and } (\frac{\pi}{2} - \pi) \text{ radians}$$

and the signal is decomposed into two components. The corresponding DWT coefficients are

$$\{X_\phi(1,0), X_\phi(1,1)\}, \{X_\psi(1,0), X_\psi(1,1)\}$$

For example, if $X_\phi(1,0)$ is zero, then there is no signal component in the frequency range $(0-\frac{\pi}{2})$ during the first two samples. If $X_\phi(1,0)$ is small or zero, the probability is high that $X_\psi(1,0)$ is also small or zero, as they represent the same time-domain samples of the signal. This results in less storage requirement for storing the locations of the coefficients in the compressed signal, making compression using the DWT more efficient. The spectral range of the original signal is 0 to π radians. The signal samples themselves are the approximation coefficients, and no detail coefficients are required. This decomposition of a signal into subband components is advantageous in applications such as discontinuity detection, denoising, and compression. With longer basis functions, the transform coefficients are obtained by weighted averaging and differencing. Further, the signal can be decomposed into a suitable number of subband components.

The inverse DWT (IDWT), with $\boldsymbol{W}_{4,1}^{-1} = \boldsymbol{W}_{4,1}^{T}$, is

$$\begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} X_\phi(1,0) \\ X_\phi(1,1) \\ X_\psi(1,0) \\ X_\psi(1,1) \end{bmatrix} \quad \text{or} \quad \boldsymbol{x} = \boldsymbol{W}_{4,1}^{-1}\boldsymbol{X}$$

Example 13.3 Using the Haar transform matrix, find the 1-level (scale 1) DWT of $x(n)$. Verify that $x(n)$ is reconstructed by computing the IDWT. Verify Parseval's theorem.

$$\{x(0) = 1, x(1) = -3, x(2) = 2, x(3) = 4\}$$

Solution

$$\begin{bmatrix} X_\phi(1,0) \\ X_\phi(1,1) \\ X_\psi(1,0) \\ X_\psi(1,1) \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ -3 \\ 2 \\ 4 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} -2 \\ 6 \\ 4 \\ -2 \end{bmatrix}$$

As is the case in the DFT, the output of taking the transform is a set of coefficients. Summing the product of these coefficients with the corresponding basis functions yields the input back.

$$\begin{array}{llll} \frac{1}{\sqrt{2}}(1 & 1 & 0 & 0) \; \frac{1}{\sqrt{2}}(-2)+ \\ \frac{1}{\sqrt{2}}(0 & 0 & 1 & 1) \; \frac{1}{\sqrt{2}}(6)+ \\ \frac{1}{\sqrt{2}}(1 & -1 & 0 & 0) \; \frac{1}{\sqrt{2}}(4)+ \\ \frac{1}{\sqrt{2}}(0 & 0 & 1 & -1) \; \frac{1}{\sqrt{2}}(-2) = \\ \hline 1 & -3 & 2 & 4 \end{array}$$

Formally, the IDWT gets back the input.

$$\begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & -1 \end{bmatrix} \frac{1}{\sqrt{2}} \begin{bmatrix} -2 \\ 6 \\ 4 \\ -2 \end{bmatrix} = \begin{bmatrix} 1 \\ -3 \\ 2 \\ 4 \end{bmatrix}$$

From Parseval's theorem, we get

$$1^2 + (-3)^2 + 2^2 + 4^2 = 30 = (\frac{1}{\sqrt{2}})^2((-2)^2 + 6^2 + 4^2 + (-2)^2$$

■

13.2.1.1 Two-Channel Haar Filter Bank

While the DWT is introduced in matrix form, in practical implementations, convolution operation is used for efficiency. The basic operation in DWT is filtering (convolution) of the signal to decompose it into subband components and downsample (as the bandwidth of the components becomes smaller). A two-channel Haar analysis and synthesis filter bank is shown in Fig. 13.4. In the analysis filter bank, the input $x(n)$ is passed through a lowpass filter in the upper channel and a highpass filter in the lower channel. The convolution of the input with the respective filter coefficients is downsampled by a factor of 2 (the odd-indexed values are discarded) to get the outputs, $X_\phi(k)$ and $X_\psi(k)$, of the analysis filter bank. Coefficients $X_\phi(k)$ and $X_\psi(k)$ represent, respectively, the low- and high-frequency components of the

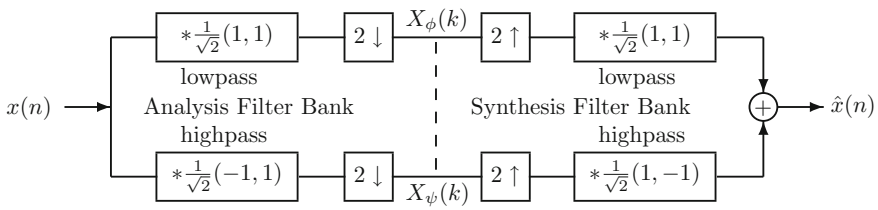


Fig. 13.4 Two-channel Haar analysis and synthesis filter bank

input signal in terms of the respective basis signals. As frequency range of the components is one-half of the input, downsampling operation eliminates the redundancy in the representation of the signal.

In the synthesis filter bank, coefficients $X_\phi(k)$ and $X_\psi(k)$ are upsampled by a factor of 2 (a zero-valued coefficient is inserted between every two adjacent coefficients). The sampling rate has to be increased to that of the input signal. Then, the upsampled signals are passed through a lowpass filter in the upper channel and a highpass filter in the lower channel to filter out the unnecessary frequency components arose due to upsampling. The upsampling and filtering operations constitute the interpolation of the signal. This interpolation is required in order to get the original sampling rate of the signal from the downsampled version. The sum of the two interpolated signals is the reconstructed input signal. The input and output of the filter bank are the same as that obtained using matrix formulation in Example (13.3). By recursively decomposing the approximation (low frequency) component at each stage, a signal can be decomposed into a set of components. The number of components can be 2 to $1 + \log_2(N)$. As in the case of the DFT, the length of the input sequence is assumed to be a power of 2.

13.2.1.2 2-Level Haar DWT

Consider the computation of a 2-level 4-point DWT using a two-stage two-channel Haar analysis filter bank, shown in Fig. 13.5. The 4-point input is $\{1, -3, 2, 4\}$, which is also considered as approximation of the input at scale 2, $X_\phi(2, k)$. The approximation coefficients at scale 1, $X_\phi(1, k)$, are computed by convolving the input $x(k)$ with the lowpass filter impulse response $\left\{\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right\}$ and then downsampling by a factor of 2. Note that the convolution output has five values. Out of these five values, only middle three values correspond to cases where both the impulse response values overlap with the given input. The first and the third values of the three middle values constitute the approximation output $X_\phi(1, k)$. These values correspond to those obtained from the definition (Example 13.3). Similarly, the detail coefficients at scale 1, $X_\psi(1, k)$, are computed by convolving the input $x(k)$ with the highpass filter impulse response

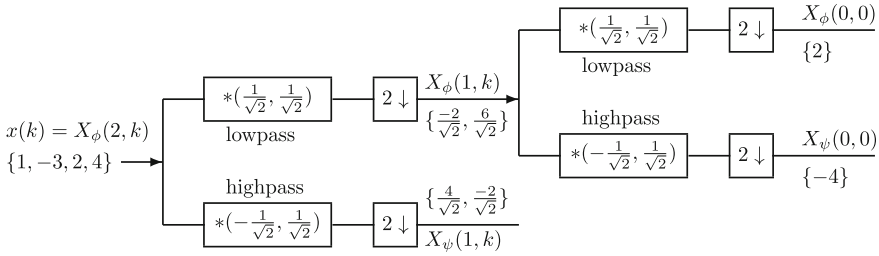


Fig. 13.5 Computation of a 2-level 4-point DWT using a two-stage two-channel Haar analysis filter bank

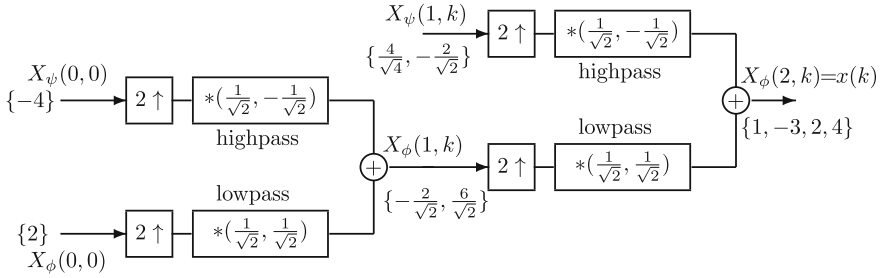


Fig. 13.6 Computation of a 2-level 4-point IDWT using a two-stage two-channel Haar synthesis filter bank

$\left\{-\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right\}$ and then downsampling by a factor of 2. The four coefficients obtained are the ones required to reconstruct the input. The approximation output $X_\phi(1,k)$ of the first stage again goes through the same process with half the values to produce $X_\phi(0,0)$ and $X_\psi(0,0)$ at the end of the second-stage analysis filter bank.

The computation of a 2-level 4-point IDWT using a two-stage two-channel Haar synthesis filter bank is shown in Fig. 13.6. In the synthesis filter bank, convolution is preceded by upsampling. The outputs of the convolution in both the channels are added to reconstruct the signal at a higher scale. An upsampler inserts zeros after each sample so that its output contains double the number of samples in the input. Coefficients $X_\phi(0,0) = 2$ and $X_\psi(0,0) = -4$ are upsampled to yield $\{2,0\}$ and $\{-4,0\}$, respectively. These samples are convolved, respectively, with impulse responses $\left\{\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right\}$ and $\left\{\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}}\right\}$ to produce $\left\{\frac{2}{\sqrt{2}}, \frac{2}{\sqrt{2}}\right\}$ and $\left\{\frac{-4}{\sqrt{2}}, \frac{4}{\sqrt{2}}\right\}$. Adding the last two sequences yields $\left\{\frac{-2}{\sqrt{2}}, \frac{6}{\sqrt{2}}\right\}$. A similar process in the second filter bank reconstructs the input to the analysis filter bank.

13.2.2 Haar 2-D DWT

Computation of a 1-level 4×4 2-D DWT using a two-stage analysis filter bank is shown in Fig. 13.7. Coefficients $\boldsymbol{X}_\phi$ are obtained by applying lowpass filtering and downsampling to each row of the 2-D data $\boldsymbol{x}$ followed by applying lowpass filtering and downsampling to each column of the resulting data. Coefficients $\boldsymbol{X}_\psi^H$ are obtained by applying highpass filtering and downsampling to each row of the 2-D data $\boldsymbol{x}$ followed by applying lowpass filtering and downsampling to each column of the resulting data. Coefficients $\boldsymbol{X}_\psi^V$ are obtained by applying lowpass filtering and downsampling to each row of the 2-D data $\boldsymbol{x}$ followed by applying highpass filtering and downsampling to each column of the resulting data. Coefficients $\boldsymbol{X}_\psi^D$ are obtained by applying highpass filtering and downsampling to each row of the 2-D data $\boldsymbol{x}$ followed by applying highpass filtering and downsampling to each column

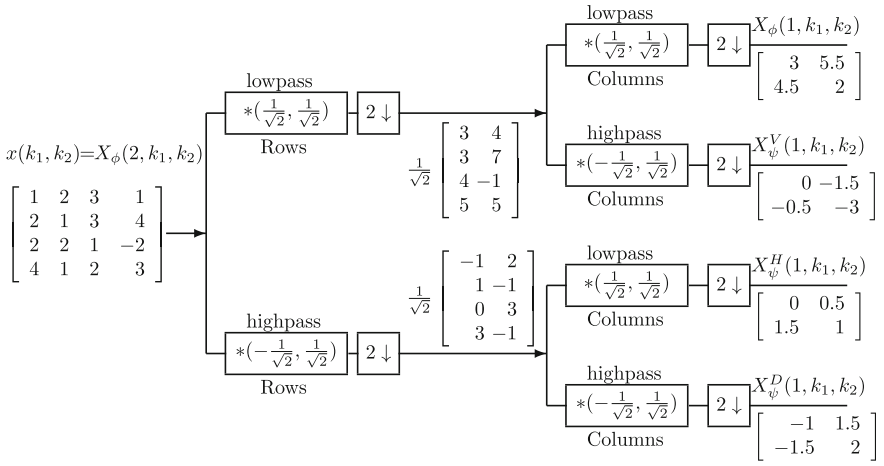


Fig. 13.7 Computation of a 1-level 4 × 4 2-D Haar DWT using a two-stage filter bank

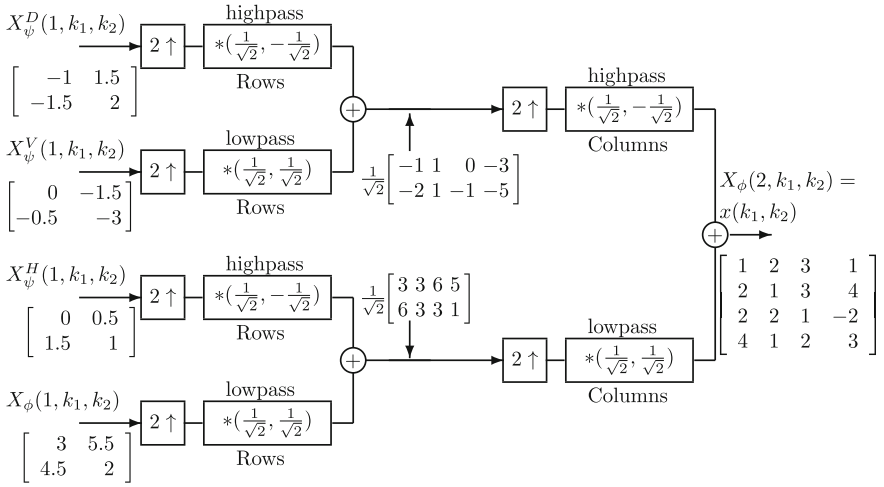


Fig. 13.8 Computation of a 1-level 4 × 4 2-D IDWT using a two-stage filter bank

of the resulting data. Computation of a 1-level 4 × 4 2-D IDWT using a two-stage synthesis filter bank is shown in Fig. 13.8. The order of the computation can be changed. That is, columns can be processed first followed by processing the rows of the resulting data.

13.2.3 Image Compression with Haar Filters

Consider the compression of the 4×4 8-bit image

172	188	189	186
178	187	189	192
188	190	196	197
191	193	197	199

The image is level shifted by subtracting 128 (0.5 times the maximum gray-level value) from each pixel value. The resulting image is

44	60	61	58
50	59	61	64
60	62	68	69
63	65	69	71

This ensures that the DWT coefficients are more evenly distributed around zero and quantization will be more effective. The row DWT (left) and the 2-D Haar DWT (right) of the level-shifted image are

$$\frac{1}{\sqrt{2}}\left[\begin{array}{cc||cc} 104 & 119 & -16 & 3 \\ 109 & 125 & -9 & -3 \\ 122 & 137 & -2 & -1 \\ 128 & 140 & -2 & -2 \end{array}\right] \qquad \left[\begin{array}{cc||cc} 106.5 & 122.0 & -12.5 & 0 \\ 125.0 & 138.5 & -2.0 & -1.5 \\ \hline -2.5 & -3.0 & -3.5 & 3.0 \\ -3.0 & -1.5 & 0 & 0.5 \end{array}\right]$$

The column DWT of the row DWT is the 2-D DWT. The transform coefficients have to be quantized. The quantized levels are assumed to be -64 to 63. The maximum value of the coefficients is $X(1, 1) = 138.5$. Therefore, the quantization step is $138.5/63 = 2.1984$. The resulting quantized coefficient matrix is

$$\left[\begin{array}{cc||cc} 48 & 55 & -5 & 0 \\ 56 & 63 & 0 & 0 \\ \hline -1 & -1 & -1 & 1 \\ -1 & 0 & 0 & 0 \end{array}\right]$$

A high quantization step will result in more compression, but the quality of the reconstructed image will be low. The quantized 4×4 matrix is converted to a 1×16 vector by arranging the values in the zigzag order. Zigzag scanning helps in getting a high run-length of zeros. The approximation coefficients are followed by the H, V, and D detail coefficients. We get the vector

$$\{48, 55, 56, 63, \quad -5, 0, 0, 0, \quad -1, -1, -1, 0, \quad -1, 1, 0, 0\}$$

A Huffman code dictionary is

```
[ 1]    '0  0  1  1'
[48]    '0  0  1  0'
[55]    '0  0  0  0  1'
[56]    '0  0  0  0  0'
[63]    '0  0  0  1  1'
[-5]    '0  0  0  1  0'
[-1]    '0  1'
[ 0]    '1'
```

The Huffman code of the image is

$$\{0010\ 00001\ 00000\ 00011\ 00010\ 1\ 1\ 1\ 01\ 01\ 01\ 1\ 01\ 0011\ 1\ 1\}$$

Small space between the codes is given for easy readability. It can be verified that the code corresponds to the 1-D vector. The compressed image requires 42 bits and that of the input image is $16 \times 8 = 128$. The compression ratio C is defined as the ratio of the bpp of the given image and that of its compressed version. Therefore, the compression ratio is $C = 128/42 = 3.0476$.
The bits per pixel is $bpp = 8/3.0476 = 2.6250$.
For reconstructing the image, the code is decoded using the dictionary and the resulting 1-D vector is converted to the 4×4 matrix. The values are multiplied by the quantization step, 2.1984, to get

105.5238	120.9127	−10.9921	0
123.1111	138.5000	0	0
−2.1984	−2.1984	−2.1984	2.1984
−2.1984	0	0	0

The IDWT of these values is also computed by the row–column method using the IDWT transform matrix. The row IDWT (right) and the 2-D Haar IDWT (left) are

66.8440	82.3892	85.4982	85.4982
88.6072	88.6072	97.9343	97.9343
−3.1090	0	0	−3.1090
−1.5545	−1.5545	0	0

45.0675	58.2579	60.4563	58.2579
49.4643	58.2579	60.4563	62.6548
60.4563	60.4563	69.2500	69.2500
62.6548	62.6548	69.2500	69.2500

These values are level shifted by adding 128 to get the reconstructed image.

173.0675	186.2579	188.4563	186.2579
177.4643	186.2579	188.4563	190.6548
188.4563	188.4563	197.2500	197.2500
190.6548	190.6548	197.2500	197.2500

SNR is 45.2924 dB.

13.3 Image Compression with Biorthogonal Filters

While the Haar DWT filters are practically useful (also the simplest, shortest and easiest to understand), a set of other DWT filters, with different characteristics, is also often used in practical applications. We just present one of them. For any DWT application, the filter selection is important. The basic principle of the DWT remains the same (decomposition of a signal into its subband components) for any filter, and the computational procedures (matrix formulation or filter bank approach) are also the same. The other filters can be considered as generalizations of the Haar filters. The DWT filters carry out the same lowpass filtering operation as a Gaussian filter does, but they have to meet certain constraints as DWT filters and their design procedure is different.

For compression purposes, a standard, called JPEG 2000, is in force. JPEG stands for Joint Photographic Experts Group. In this standard, the use of the DWT for image compression is a main feature. Two DWT filters are recommended for image compression. As linear-phase characteristics are important in image processing, both of them are linear-phase filters. The 5/3 filter is recommended for lossless compression, and the 9/7 filter is recommended for lossy compression. As lossy compression is more often used, we concentrate on that.

13.3.1 CDF 9/7 Filter

We list the impulse responses of all the four CDF 9/7 filters with a precision of six digits.
Lowpass analysis filter

$$\begin{aligned} l(0) &= 0.852699 \\ l(1) = l(-1) &= 0.377403 \\ l(2) = l(-2) &= -0.110624 \\ l(3) = l(-3) &= -0.023850 \\ l(4) = l(-4) &= 0.037829 \end{aligned}$$

Highpass analysis filter

$$\begin{aligned} h(-2) = h(4) = \tilde{l}(3) &= -0.064539 \\ h(-1) = h(3) = -\tilde{l}(2) &= \quad 0.040689 \\ h(0) = h(2) = \tilde{l}(1) &= \quad 0.418092 \\ h(1) = -\tilde{l}(0) &= -0.788486 \end{aligned}$$

Lowpass synthesis filter

$$\begin{aligned} \tilde{l}_0 &= \quad 0.788486 \\ \tilde{l}_{-1} = \tilde{l}_1 &= \quad 0.418092 \\ \tilde{l}_{-2} = \tilde{l}_2 &= -0.040689 \\ \tilde{l}_{-3} = \tilde{l}_3 &= -0.064539 \end{aligned}$$

Highpass synthesis filter

$$\begin{aligned} \tilde{h}_{-3} = \tilde{h}_5 = -l(4) &= -0.037829 \\ \tilde{h}_{-2} = \tilde{h}_4 = l(3) &= -0.023850 \\ \tilde{h}_{-1} = \tilde{h}_3 = -l(2) &= \quad 0.110624 \\ \tilde{h}_0 = \tilde{h}_2 = l(1) &= \quad 0.377403 \\ \tilde{h}_1 = -l(0) &= -0.852699 \end{aligned}$$

The magnitude of the frequency responses of the 9/7 analysis filters is shown in Fig. 13.9. Compared with the Haar filters, these filters have much sharper frequency responses.

Example 13.4 Compute the 1-level DWT of the input $x(n)$ using the CDF 9/7 filter by the convolution approach. Assume whole-point symmetry of the data. Compute the IDWT of the DWT coefficients and verify that the input $x(n)$ is reconstructed.

$$\boldsymbol{x} = \{1, -2, 3, 4, 2, -3, 1, 1, 3, -2, 2, 4, 4, 2, -1, 1\}$$

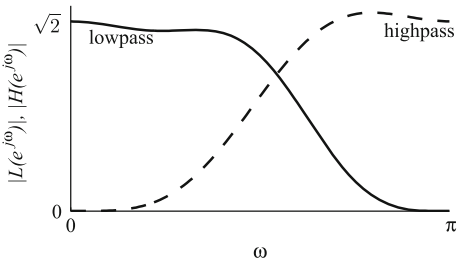


Fig. 13.9 Magnitude of the frequency responses of the CDF 9/7 analysis filters

Solution
DWT: The input, with the whole-point symmetry extension, is

$$\boldsymbol{x}_w = \{2, 4, 3, -2, \quad 1, -2, 3, 4, 2, -3, 1, 1, 3, -2, 2, 4, 4, 2, -1, 1, \quad -1, 2, 4, 4\}$$

The convolution of $\boldsymbol{x}_w$ with the analysis lowpass filter impulse response,

$$\{l_{-4}, l_{-3}, l_{-2}, l_{-1}, l_0, l_1, l_2, l_3, l_4\} = \{0.0378, -0.0238, -0.1106, 0.3774, 0.8527, 0.3774, -0.1106, -0.0238, 0.0378\}$$

after downsampling, is

$$\{X_\phi(3,0), X_\phi(3,1), X_\phi(3,2), X_\phi(3,3), X_\phi(3,4), X_\phi(3,5), X_\phi(3,6), X_\phi(3,7)\} = \{-1.3601, 3.2516, 1.8155, -0.3138, 2.0519, 1.6143, 5.6641, 0.0315\}$$

With periodic extension, the lowpass output is

$$\{1.8155, 3.2516, \ -1.3601, 3.2516, 1.8155, -0.3138, 2.0519, 1.6143, 5.6641, 0.0315, \ 0.0315, 5.6641\}$$

The convolution of $\boldsymbol{x}_w$ with the analysis highpass filter impulse response,

$$\{h_{-2}, h_{-1}, h_0, h_1, h_2, h_3, h_4\} = \{-0.0645, 0.0407, 0.4181, -0.7885, 0.4181, 0.0407, -0.0645\}$$

after downsampling, is

$$\{X_\psi(3,0), X_\psi(3,1), X_\psi(3,2), X_\psi(3,3) X_\psi(3,4), X_\psi(3,5), X_\psi(3,6), X_\psi(3,7)\} = \{3.0080, -1.3960, 3.4359, 0.4223, 3.5482, -0.7745, -0.1838, -1.9782\}$$

With periodic extension, the highpass output is

$$\{-1.396, 3.008, \ 3.008, -1.396, 3.4359, 0.4223, 3.5482, -0.7745, -0.1838, -1.9782, \ -0.1838, -0.7745\}$$

IDWT: The convolution of the lowpass output alternately with the even- and odd-indexed coefficients

$$\{-0.0407, 0.7885, -0.0407\}$$

and

$$\{-0.0645, 0.4181, 0.4181, -0.0645\}$$

of the synthesis lowpass filter impulse response

$$\tilde{\boldsymbol{l}} = \{-0.0645, -0.0407, 0.4181, 0.7885, 0.4181, -0.0407, -0.0645\}$$

yields the even- and odd-indexed values of the output of the lowpass channel.

$$\{-1.3371, 0.4638, 2.5453, 2.2265, 1.3119, 0.2856, -0.4048, 0.5054,$$

$$1.5650, 1.1875, 0.9589, 2.9086, 4.3991, 2.2751, -0.2069, -0.7048\}$$

The convolution of the highpass output alternately with the even- and odd-indexed coefficients

$$\{-0.0238, 0.3774, 0.3774, -0.0238, \}$$

and

$$\{-0.0378, 0.1106, -0.8527, 0.1106, -0.0378\}$$

of the synthesis highpass filter impulse response

$$\tilde{\boldsymbol{h}} = \{-0.0378, -0.0238, 0.1106, 0.3774, -0.8527, 0.3774, 0.1106, -0.0238, -0.0378\}$$

yields the even- and odd-indexed values of the output of the highpass channel.

$$\{2.3371, -2.4638, 0.45471.7735, 0.6881, -3.2856, 1.4048, 0.4946,$$

$$1.4350, -3.1875, 1.0411, 1.0914, -0.3991, -0.2751, -0.7931, 1.7048\}$$

Adding the last two output sequences, we get back the reconstructed input

$$\{1, -2, 3, 4, 2, -3, 1, 1, 3, -2, 2, 4, 4, 2, -1, 1\}.$$ ■

In this example, the compression of a 16×16 8-bit gray-level image using the CDF 9/7 filter for lossy compression is presented. Consider the 16×16 image, shown in Table 13.2, with the pixels represented by 8 bits. The 0–255 gray-level range of the image is changed to -28 to 127 by subtracting 128 from each pixel value. This process, called level-shifting, spreads the gray levels more evenly around zero, and the quantization becomes more effective. The resulting image is shown in Table 13.3.

The 1-level 2-D DWT is computed by the row–column method using the 9/7 filter, assuming whole-point symmetry extension at the borders. The resulting transform representation of the image is shown in Tables 13.4, 13.5, 13.6, and 13.7.

The range and precision of the DWT values widely vary. The DWT coefficients have to be quantized. Quantization is restricting the values of a function to a finite set of possible values. Let us say the quantization levels be integer values from -64 to 63. That is, the quantized image values are restricted to

$$\{-64, -63, -62, \ldots, -1, 0, 1, \ldots, 61, 62, 63\}$$

Table 13.2 16×16 8-bit image

61	67	59	43	43	49	49	48	48	56	49	48	50	49	48	50
67	59	62	46	45	44	51	51	48	52	54	51	48	50	49	50
69	52	62	47	48	45	50	50	53	60	53	48	53	48	53	49
71	56	62	55	48	48	47	53	52	56	51	50	48	50	51	52
72	61	56	60	50	51	48	50	52	50	53	49	47	51	50	51
68	64	56	58	56	51	51	50	50	51	54	51	48	48	50	49
64	69	59	56	57	51	51	51	49	49	52	55	51	48	51	49
60	74	62	57	56	52	51	52	50	48	50	54	51	49	50	49
67	73	70	56	59	55	53	51	53	50	46	50	53	45	38	48
85	65	72	57	56	59	51	52	54	53	46	44	41	64	132	212
99	63	62	67	55	61	58	54	53	47	61	48	112	228	255	255
84	76	56	66	63	58	60	55	38	154	234	225	255	255	255	255
96	87	58	65	66	60	59	49	129	249	255	255	255	255	255	255
166	94	72	70	69	62	49	111	255	255	255	255	255	255	255	255
219	155	89	72	72	60	53	212	255	255	255	255	255	255	255	255
228	238	179	102	75	101	195	255	255	255	255	255	255	255	255	255

Table 13.3 Level-shifted image

−67	−61	−69	−85	−85	−79	−79	−80	−80	−72	−79	−80	−78	−79	−80	−78
−61	−69	−66	−82	−83	−84	−77	−77	−80	−76	−74	−77	−80	−78	−79	−78
−59	−76	−66	−81	−80	−83	−78	−78	−75	−68	−75	−80	−75	−80	−75	−79
−57	−72	−66	−73	−80	−80	−81	−75	−76	−72	−77	−78	−80	−78	−77	−76
−56	−67	−72	−68	−78	−77	−80	−78	−76	−78	−75	−79	−81	−77	−78	−77
−60	−64	−72	−70	−72	−77	−77	−78	−78	−77	−74	−77	−80	−80	−78	−79
−64	−59	−69	−72	−71	−77	−77	−77	−79	−79	−76	−73	−77	−80	−77	−79
−68	−54	−66	−71	−72	−76	−77	−76	−78	−80	−78	−74	−77	−79	−78	−79
−61	−55	−58	−72	−69	−73	−75	−77	−75	−78	−82	−78	−75	−83	−90	−80
−43	−63	−56	−71	−72	−69	−77	−76	−74	−75	−82	−84	−87	−64	4	84
−29	−65	−66	−61	−73	−67	−70	−74	−75	−81	−67	−80	−16	100	127	127
−44	−52	−72	−62	−65	−70	−68	−73	−90	26	106	97	127	127	127	127
−32	−41	−70	−63	−62	−68	−69	−79	1	121	127	127	127	127	127	127
38	−34	−56	−58	−59	−66	−79	−17	127	127	127	127	127	127	127	127
91	27	−39	−56	−56	−68	−75	84	127	127	127	127	127	127	127	127
100	110	51	−26	−53	−27	67	127	127	127	127	127	127	127	127	127

Table 13.4 2-D DWT approximation coefficients of the image using 9/7 filter, assuming whole-point symmetry extension

−127.3448	−139.9055	−169.7991	−155.9316	−158.6128	−152.6405	−158.5150	−158.1505
−133.4291	−141.8051	−163.5655	−157.7568	−149.1364	−148.9220	−156.4869	−154.4819
−121.4208	−140.2536	−149.7902	−157.5888	−153.2156	−152.5162	−160.3273	−155.2975
−121.4762	−135.7732	−145.3589	−153.8353	−157.3353	−150.6995	−150.5305	−147.9851
−115.7834	−121.1063	−143.6174	−152.0599	−146.9519	−155.2393	−165.5578	−145.9830
−87.8762	−131.3340	−136.5491	−136.5590	−155.0885	−89.5567	−1.2265	220.7238
−60.8548	−133.4541	−118.9716	−153.8616	20.6111	270.8120	262.0152	246.5985
144.7163	−34.7725	−106.7292	−44.1569	254.1556	236.3062	247.4466	253.9437

Table 13.5 2-D DWT horizontal detail coefficients of the image using 9/7 filter, assuming whole-point symmetry extension

−2.9799	7.6479	−1.6731	−0.2939	−4.0619	0.8056	−1.9453	−2.5407
14.5210	6.2178	2.7970	−0.3321	−5.9503	3.1955	2.8303	2.3185
5.2183	−6.7116	−1.7966	−0.9849	1.4458	0.4469	−2.5127	−0.9890
−8.0599	1.3487	3.4332	−1.8456	2.0106	−2.2757	1.1864	4.8338
−3.4114	9.5148	−2.0237	2.6907	−2.5267	−6.0214	12.7566	−31.1115
16.7760	−8.4134	−2.7546	−10.1540	1.3611	28.3373	−21.5006	−6.5359
0.2730	−5.9260	−4.9717	37.9919	−42.0348	13.3518	3.4031	−0.7926
5.6580	4.6383	−1.5196	−32.9030	16.1861	−0.1892	−1.3045	0.6151

Table 13.6 2-D DWT vertical detail coefficients of the image using 9/7 filter, assuming whole-point symmetry extension

−0.9432	−0.9770	0.5242	−1.1217	3.1915	−1.5871	1.1461	−0.0318
0.4405	−2.7077	0.4483	0.2741	−0.8797	0.8123	0.3435	−0.3217
0.3747	1.2473	−1.4410	−0.8110	−0.1582	−1.1254	1.3535	1.6255
0.6065	0.6022	1.2402	0.0866	0.8957	−1.2081	−6.8537	−17.9384
0.2776	−1.5503	1.1796	3.7991	−8.8033	−8.7175	45.8253	8.1111
1.3479	−1.0804	0.8881	−10.2223	33.1293	−72.0737	−64.1080	12.2863
12.5249	14.2337	−4.3143	21.8700	−34.7945	21.4219	9.8220	0.0960
−26.6349	−95.1025	0.4736	−120.1604	19.4085	−5.2811	0.5968	−0.0000

Table 13.7 2-D DWT diagonal detail coefficients of the image using 9/7 filter, assuming whole-point symmetry extension

−1.6493	1.0956	−2.3437	1.8100	−4.3731	1.7987	2.4691	0.6713
−0.2431	−0.2099	1.2240	2.4962	1.3431	2.3551	0.7319	1.4349
−0.5937	−1.0636	−1.5184	−0.8815	1.1241	−1.2761	−0.5882	0.1502
3.1337	3.0249	0.3671	1.3077	−1.5401	−0.8745	3.6515	−7.0732
−3.5100	−4.5996	3.1019	−2.2649	4.9736	15.7175	−30.4008	47.7066
7.7589	−0.9004	−6.4132	28.0413	−4.0175	−5.6723	−13.7155	5.0818
−20.6581	2.9835	8.6363	−29.4636	−20.8988	2.6410	2.2256	−1.0495
22.9774	−8.0826	−21.1060	−25.0058	12.6283	−1.0181	0.0000	0.0000

For example, the value 61.6 is represented by 62, the nearest of the allowable quantization level. We round the values and assign them to the nearest levels. As there are 128 levels, a wordlength of 7 bits is sufficient. Quantization is a stage in compression where one has to trade off between accuracy and compression ratio. Quantization results in loss of information. It is an irreversible process, since all the values in the range of a quantization level are assigned the same value.

Table 13.8 Quantized image

−30	−33	−40	−36	−37	−36	−37	−37	−1	2	0	0	−1	0	0	−1
−31	−33	−38	−37	−35	−35	−36	−36	3	1	1	0	−1	1	1	1
−28	−33	−35	−37	−36	−35	−37	−36	1	−2	0	0	0	0	−1	0
−28	−32	−34	−36	−37	−35	−35	−34	−2	0	1	0	0	−1	0	1
−27	−28	−33	−35	−34	−36	−39	−34	−1	2	0	1	−1	−1	3	−7
−20	−31	−32	−32	−36	−21	0	51	4	−2	−1	−2	0	7	−5	−2
−14	−31	−28	−36	5	63	61	57	0	−1	−1	9	−10	3	1	0
34	−8	−25	−10	59	55	58	59	1	1	0	−8	4	0	0	0
0	0	0	0	1	0	0	0	0	0	−1	0	−1	0	1	0
0	−1	0	0	0	0	0	0	0	0	0	1	0	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	−2	−4	1	1	0	0	0	0	1	−2
0	0	0	1	−2	−2	11	2	−1	−1	1	−1	1	4	−7	11
0	0	0	−2	8	−17	−15	3	2	0	−1	7	−1	−1	−3	1
3	3	−1	5	−8	5	2	0	−5	1	2	−7	−5	1	1	0
−6	−22	0	−28	5	−1	0	0	5	−2	−5	−6	3	0	0	0

The maximum value of the coefficients is $X(6, 5) = 270.8120$. Therefore, the quantization step is $270.8120/63 = 4.2986$. The resulting quantized coefficient matrix is shown in Table 13.8. No thresholding is done in this example. If thresholding is applied, the number of independent values reduces and the compression ratio will increase at the cost of more degradation of the reconstructed image.

The quantized coefficients are scanned in a zigzag pattern and represented by a 1-D vector. Then, the corresponding Huffman code is generated. The average bpp is 4.0313.

13.3.1.1 Image Reconstruction

For reconstructing the image, the Huffman code is decoded using the dictionary and the resulting 1-D vector is converted to 16×16 matrix. The values are multiplied by the quantization step, 4.2986, to get the 2-D DWT of the reconstructed level-shifted image, shown in Table 13.9. The 2-D IDWT yields the values of level-shifted reconstructed image, shown in Table 13.10. These values are level shifted by adding 128 to get the reconstructed image, shown in Table 13.11.

Figure 13.10a shows a 256×256 8-bit image. Figure 13.10b–d shows, respectively, the reconstructed images with quantization step sizes 30.1062, 60.2123, and 120.4247. The compression ratios are 4.1173, 5.0486, and 6.2006, respectively.

Table 13.9 2-D DWT of the reconstructed level-shifted image

−129	−142	−172	−155	−159	−155	−159	−159	−4	9	0	0	−4	0	0	−4
−133	−142	−163	−159	−150	−150	−155	−155	13	4	4	0	−4	4	4	4
−120	−142	−150	−159	−155	−150	−159	−155	4	−9	0	0	0	0	−4	0
−120	−138	−146	−155	−159	−150	−150	−146	−9	0	4	0	0	−4	0	4
−116	−120	−142	−150	−146	−155	−168	−146	−4	9	0	4	−4	−4	13	−30
−86	−133	−138	−138	−155	−90	0	219	17	−9	−4	−9	0	30	−21	−9
−60	−133	−120	−155	21	271	262	245	0	−4	−4	39	−43	13	4	0
146	−34	−107	−43	254	236	249	254	4	4	0	−34	17	0	0	0
0	0	0	0	4	0	0	0	0	0	−4	0	−4	0	4	0
0	−4	0	0	0	0	0	0	0	0	0	4	0	4	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	−9	−17	4	4	0	0	0	0	4	−9
0	0	0	4	−9	−9	47	9	−4	−4	4	−4	4	17	−30	47
0	0	0	−9	34	−73	−64	13	9	0	−4	30	−4	−4	−13	4
13	13	−4	21	−34	21	9	0	−21	4	9	−30	−21	4	4	0
−26	−95	0	−120	21	−4	0	0	21	−9	−21	−26	13	0	0	0

Table 13.10 Level-shifted reconstructed image

−67	−62	−69	−86	−87	−79	−78	−78	−80	−72	−80	−78	−79	−83	−80	−76
−64	−68	−68	−83	−82	−86	−76	−79	−80	−78	−76	−79	−78	−77	−79	−79
−59	−76	−67	−80	−81	−83	−79	−78	−75	−69	−75	−81	−73	−81	−74	−80
−57	−71	−66	−71	−79	−82	−81	−75	−77	−74	−76	−76	−79	−78	−76	−79
−56	−67	−74	−68	−78	−78	−79	−80	−77	−75	−74	−79	−81	−75	−79	−78
−60	−63	−74	−70	−74	−78	−78	−80	−79	−77	−75	−75	−80	−77	−77	−77
−63	−60	−71	−73	−71	−78	−76	−79	−80	−78	−76	−72	−78	−80	−77	−77
−69	−53	−67	−69	−71	−76	−75	−77	−78	−78	−80	−74	−77	−78	−78	−81
−62	−55	−57	−71	−68	−74	−73	−76	−75	−77	−81	−80	−75	−84	−89	−80
−42	−64	−57	−71	−72	−67	−76	−79	−73	−74	−82	−85	−88	−65	3	84
−28	−65	−66	−62	−74	−67	−70	−75	−74	−80	−67	−82	−14	100	126	128
−43	−51	−74	−63	−66	−69	−71	−74	−91	27	106	98	128	127	126	125
−32	−41	−69	−65	−62	−69	−68	−80	2	122	126	126	127	125	127	126
38	−33	−55	−57	−60	−67	−78	−17	128	126	127	129	127	128	126	127
91	29	−40	−56	−56	−69	−75	86	127	127	128	127	129	126	127	127
100	110	51	−26	−52	−28	68	128	125	125	127	128	129	126	127	127

Table 13.11 Reconstructed image

61	66	59	42	41	49	50	50	48	56	48	50	49	45	48	52
64	60	60	45	46	42	52	49	48	50	52	49	50	51	49	49
69	52	61	48	47	45	49	50	53	59	53	47	55	47	54	48
71	57	62	57	49	46	47	53	51	54	52	52	49	50	52	49
72	61	54	60	50	50	49	48	51	53	54	49	47	53	49	50
68	65	54	58	54	50	50	48	49	51	53	53	48	51	51	51
65	68	57	55	57	50	52	49	48	50	52	56	50	48	51	51
59	75	61	59	57	52	53	51	50	50	48	54	51	50	50	47
66	73	71	57	60	54	55	52	53	51	47	48	53	44	39	48
86	64	71	57	56	61	52	49	55	54	46	43	40	63	131	212
100	63	62	66	54	61	58	53	54	48	61	46	114	228	254	256
85	77	54	65	62	59	57	54	37	155	234	226	256	255	254	253
96	87	59	63	66	59	60	48	130	250	254	254	255	253	255	254
166	95	73	71	68	61	50	111	256	254	255	257	255	256	254	255
219	157	88	72	72	59	53	214	255	255	256	255	257	254	255	255
228	238	179	102	76	100	196	256	253	253	255	256	257	254	255	255

13.4 Summary

- Image compression is essential for the efficient storage and transmission of images due to their enormous amount of data.
- In lossless compression, the image is compressed so that it can be decompressed to its original version exactly. While this type of compression is mandatory for legal and medical records, the compression ratio achieved is relatively less.
- In lossy compression, a higher compression is achieved at the loss of some fidelity. However, this type of compression is more often used as some degradation is acceptable for most purposes.
- In both types of compression, coding is one of the stages, where compression is achieved. The pixel values with a higher probability are coded with less number of bits and vice versa.
- In lossy compression, the DWT of the image is computed. The magnitude of the subband components of image varies. The transform values with low magnitudes can be coded with less number of bits and those with negligible magnitudes can be discarded altogether. In lossy compression coding, quantization and thresholding affect the compression.
- The DWT is a generalization of the Fourier analysis. The basis signals are of finite duration, composed of a continuous group of frequency components of the spectrum.

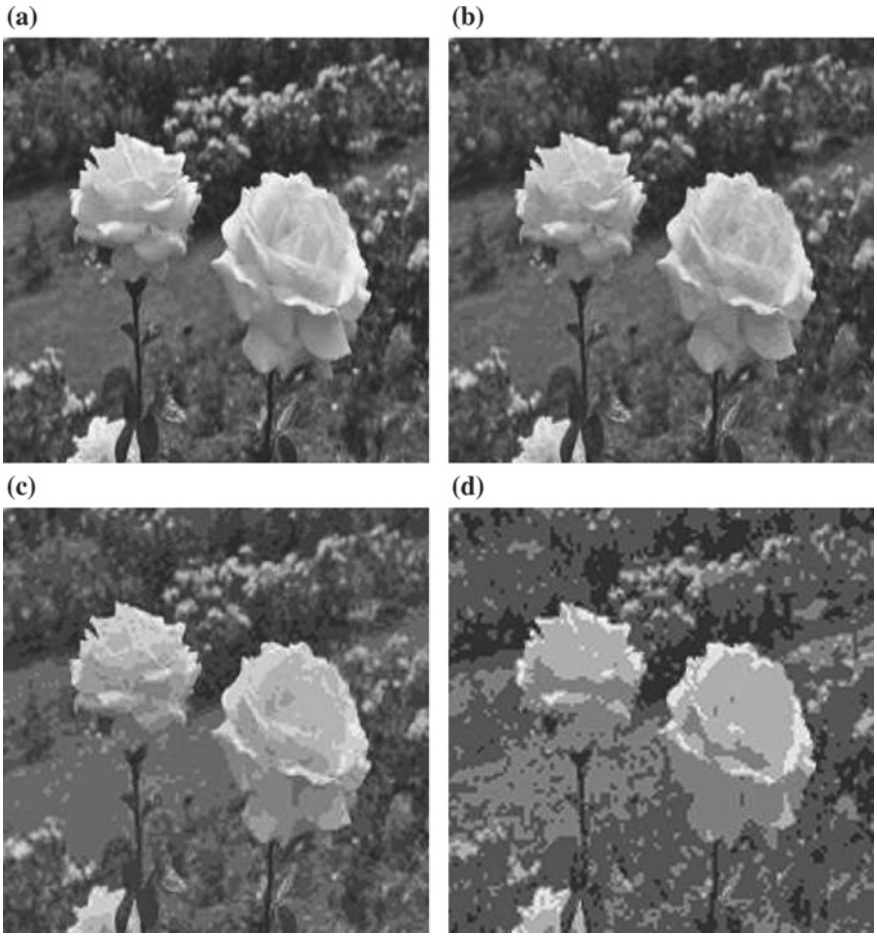


Fig. 13.10 **a** 256×256 8-bit image; **b–d** reconstructed images with quantization step sizes 30.1062, 60.2123, and 120.4247

- The DWT decomposes an image into subband components rather than individual frequency components as in Fourier analysis. DWT is implemented using filter banks, which are composed of samplers and filters. A variety of filters are available.
- The basic step in the implementation of the DWT is convolution followed by decimation.
- The 2-D DWT is decomposable and has fast algorithms for its computation.
- The 5/3 and 9/7 DWT filters are recommended for image compression in JPEG 2000 standard. Both these filters are linear-phase filters.
- As the DWT is able to provide time-frequency analysis of an image, it is inherently suitable for nonstationary images.

- Two of the major applications of the DWT in image processing are image compression and denoising.

Exercises

13.1 Given a 4×4 image, compute the entropy. Find the Huffman code representation of its unique symbols and the bpp.

* (i)

$$\begin{bmatrix} 144 & 113 & 121 & 107 \\ 144 & 110 & 121 & 103 \\ 129 & 109 & 120 & 99 \\ 116 & 108 & 121 & 103 \end{bmatrix}$$

(ii)

$$\begin{bmatrix} 209 & 190 & 179 & 179 \\ 143 & 136 & 132 & 129 \\ 131 & 130 & 125 & 117 \\ 113 & 109 & 118 & 143 \end{bmatrix}$$

(iii)

$$\begin{bmatrix} 85 & 91 & 91 & 89 \\ 79 & 83 & 88 & 87 \\ 90 & 86 & 86 & 90 \\ 97 & 93 & 88 & 90 \end{bmatrix}$$

13.2 Given a 4×4 image, decompose it into 4 bit planes and represent each of them by run-length coding. Use both the methods.

(i)

$$\begin{bmatrix} 6 & 6 & 15 & 14 \\ 8 & 8 & 4 & 11 \\ 9 & 9 & 10 & 12 \\ 10 & 14 & 13 & 1 \end{bmatrix}$$

* (ii)

$$\begin{bmatrix} 11 & 9 & 15 & 14 \\ 13 & 7 & 6 & 7 \\ 6 & 5 & 5 & 5 \\ 11 & 4 & 4 & 2 \end{bmatrix}$$

(iii)

$$\begin{bmatrix} 13 & 9 & 12 & 10 \\ 13 & 13 & 12 & 9 \\ 13 & 13 & 12 & 9 \\ 11 & 13 & 12 & 10 \end{bmatrix}$$

13.3 Given a 4×4 image, find the linear predictive code. Find the entropies of the input and the code.
(i)

$$\begin{bmatrix} 15 & 16 & 20 & 20 \\ 15 & 15 & 19 & 22 \\ 15 & 16 & 19 & 20 \\ 15 & 17 & 19 & 16 \end{bmatrix}$$

(ii)

$$\begin{bmatrix} 158 & 157 & 154 & 149 \\ 168 & 153 & 157 & 149 \\ 170 & 152 & 157 & 149 \\ 166 & 153 & 157 & 142 \end{bmatrix}$$

* (iii)

$$\begin{bmatrix} 106 & 103 & 98 & 99 \\ 121 & 122 & 108 & 93 \\ 102 & 102 & 100 & 99 \\ 100 & 101 & 102 & 96 \end{bmatrix}$$

13.4 Given a 1-digit sequence, find its arithmetic code. Reconstruct the sequence from the code. Let there be three symbols $\{1, 2, 3\}$ and number of occurrences of the symbols, respectively, be $\{2, 1, 1\}$ in a sequence of length 4.
(i) $\{1\}$
(ii) $\{2\}$

13.5 Given a sequence $x(n)$, find the 1-level DWT coefficients using the $9/7$ filter. Assume whole-point symmetry at the borders. Verify that the reconstructed signal is the same as the input.
* (i)

$$\{1, -4, 1, 3, 3, 1, 3, 0, 2, 2, 3, 1, -5, 2, 0, 3\}$$

(ii)

$$\{1, 2, 1, 1, 3, 4, 0, 3, 1, 3, 2, -1, 0, 1, 4, -3\}$$

(iii)

$$\{-2, 0, 2, -2, 1, 0, 3, 1, -2, 1, 2, -1, 2, 0, -2, 1\}$$

13.6 Given a 4×4 image, find its compressed version using the 1-level Haar DWT and the Huffman code. What is the bpp and SNR.
(i)

$$\begin{bmatrix} 170 & 168 & 164 & 173 \\ 179 & 167 & 167 & 167 \\ 184 & 179 & 173 & 166 \\ 183 & 179 & 184 & 173 \end{bmatrix}$$

* (ii)

$$\begin{bmatrix} 172 & 173 & 170 & 171 \\ 171 & 176 & 173 & 172 \\ 174 & 178 & 172 & 170 \\ 176 & 175 & 171 & 170 \end{bmatrix}$$

(iii)

$$\begin{bmatrix} 162 & 163 & 163 & 161 \\ 162 & 164 & 161 & 161 \\ 163 & 165 & 164 & 162 \\ 167 & 161 & 162 & 164 \end{bmatrix}$$

Chapter 14
Color Image Processing

Abstract Human vision is more sensitive to color than gray levels. Therefore, color image processing is important, although it requires more memory to store and longer execution times to process. There are different color models, and each one is suitable for some application. In the RGB model, a color image is expressed in terms of the intensities of its red, green, and blue components. In the HSI model, the intensity component is separated from the color components. This model can use the algorithms for gray level images. Some of the processing are based on those of gray level images, and some are exclusive to color images.

The human visual system is more sensitive to color and edges than to gray level. There are two types of color images, full-color and pseudocolor. The first type is obtained by color sensors, and the second type is obtained by assigning colors to gray level images. Most of the processing methods of gray level images are applicable to color image processing either directly or with some modifications. The visible spectrum is composed of different colors. It is the reflectivity of the object that determines the color human beings perceive. For example, an object that reflects all colors equally well is perceived as white. On the other hand, objects which absorb some colors and reflect others exhibit color.

The pixel value of a color image is vector-valued. For example, the intensity values of its red, green, and blue components form a vector. Therefore, three 2-D matrices are required to represent a color image. Obviously, the storage and processing requirements of a color image are three times that of a gray scale image. For each color, with 8-bit representation, intensity values zero or 255 implies that the color is absent or fully present, respectively. For example, the vector (0, 0, 0) represents black and (255, 255, 255) represents white. If all the components of all the vectors of an image are equal, then it becomes a gray scale image.

D. Sundararajan, *Digital Image Processing*, DOI 10.1007/978-981-10-6113-4_14

14.1 Color Models

There are infinite colors. To specify a color, we need a color model. There are infinite points in a plane. But all of them are specified by their x- and y-coordinates. In Fourier analysis, an arbitrary signal is specified by sinusoidal signals of various frequencies. Infinite places on earth are specified by their longitudes and latitudes. Similarly, any color can be specified by a set of basis colors. Similar to the availability of various transforms suitable for various applications, various color models are available to suit various color image processing tasks. RGB model is mostly used for image acquisition and display. CMY and CYMK models are used in color printing. The HSI model is suitable for image processing operations since it decouples the color component from the intensity value of the image.

14.1.1 The RGB Model

In the RGB color model, any color can be specified as a linear combination of the three primary colors, red, green, and blue. Computers and televisions use this model for color display. Figure 14.1 shows the wavelengths of the three primary colors, as set by a standard. However, the wavelengths have to vary around the specified values to produce all colors. Figure 14.2 shows the variation of the blue, green, and red color intensities with 8 bits in the first row. The variation of the cyan, magenta, and yellow color intensities with 8 bits is shown in the second row. Each point in the RGB color cube, shown as an image in Fig. 14.3 and as a line figure in Fig. 14.4, is one of the infinite colors. The three primary colors form 3 of the 8 corners of the color cube. Black and white form 2 other corners. The dotted line joining the black and white corners is the gray level line, along which the gray level varies from black to white. Obviously, the contributions of the 3 primary colors on this line are equal. The points on the dotted line between black and white have equal values of the primary colors. They are shades of gray. The other 3 corners are the secondary colors, cyan, magenta, and yellow.

A digital color image is characterized by 3 2-D matrices, one for each primary color, of equal size in the RGB model. The values of these 3 matrices are combined to produce the image for display. Each of the element in the 3 matrices is typically represented by 8 bits. A color pixel is characterized by $3 \times 8 = 24$ bits. Therefore,

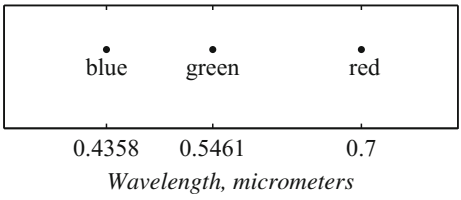


Fig. 14.1 The wavelengths of the 3 primary colors *blue*, *green*, and *red* in the visible spectrum

Fig. 14.2 The variation of the *red*, *green*, and *blue* color intensities with 8 bits (*first row*). The variation of the *cyan*, *magenta*, and *yellow* color intensities with 8 bits

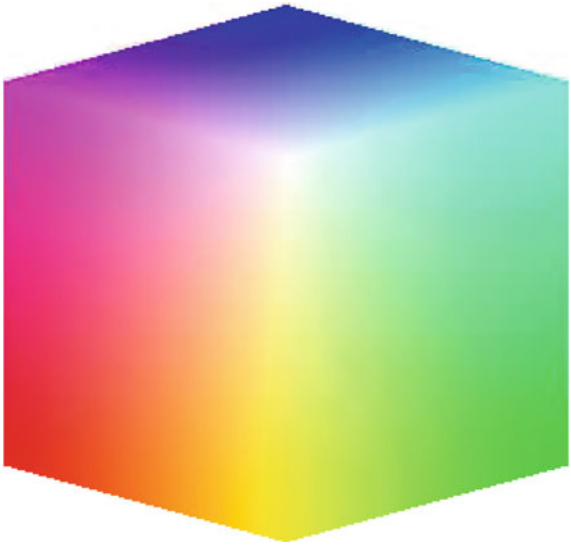


Fig. 14.3 The RGB color cube

the total number of colors possible is $2^{24} = 16777216$. In the line figure, the black and blue color edge appears first, whereas the yellow and white color edge appears first in the color cube.

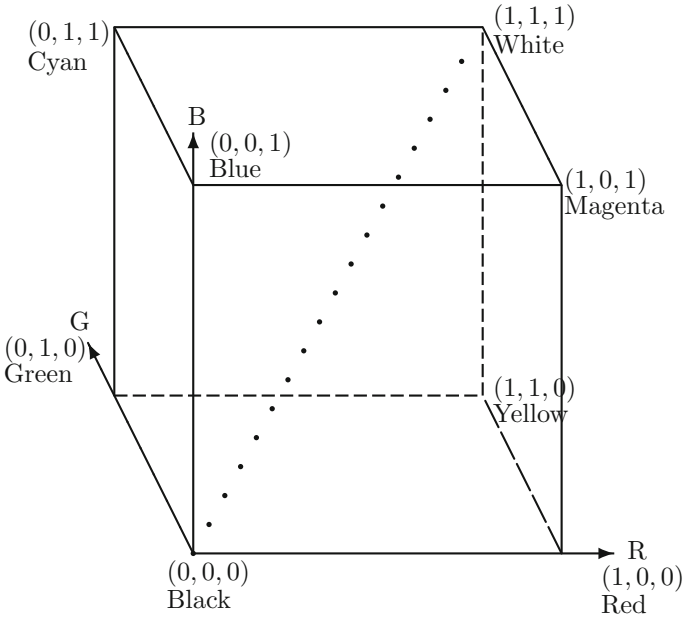


Fig. 14.4 The RGB color model

Figure 14.5a–d shows a 256×256 color image and the intensities of its red, green, and blue components. The red, green, and blue component pixel values at coordinates $(73 : 76, 173 : 176)$, respectively, are

$$xr = \begin{bmatrix} 245 & 246 & 246 & 248 \\ 246 & 247 & 246 & 247 \\ 246 & 246 & 245 & 248 \\ 248 & 247 & 246 & 248 \end{bmatrix} \quad xg = \begin{bmatrix} 191 & 196 & 193 & 190 \\ 192 & 197 & 192 & 189 \\ 192 & 197 & 192 & 189 \\ 192 & 198 & 191 & 188 \end{bmatrix}$$

$$xb = \begin{bmatrix} 222 & 225 & 223 & 223 \\ 223 & 226 & 223 & 222 \\ 223 & 225 & 222 & 222 \\ 223 & 224 & 221 & 224 \end{bmatrix}$$

In this neighborhood (about the center of the top-right quadrant), the image is primarily white, and therefore, the intensities of all the three components are almost equal and high.

Fig. 14.5 **a** A 256 × 256 color image; **b–d** the intensity images of the its *red*, *green*, and *blue* components, respectively

The red, green, and blue component pixel values at coordinates (63 : 66, 89 : 92), respectively, are

$$xr = \begin{bmatrix} 251 & 252 & 252 & 248 \\ 253 & 251 & 252 & 250 \\ 253 & 252 & 252 & 250 \\ 252 & 250 & 250 & 247 \end{bmatrix} \quad xg = \begin{bmatrix} 109 & 114 & 116 & 112 \\ 118 & 122 & 124 & 120 \\ 125 & 129 & 130 & 127 \\ 132 & 137 & 139 & 134 \end{bmatrix}$$

$$xb = \begin{bmatrix} 88 & 84 & 89 & 90 \\ 75 & 67 & 63 & 65 \\ 55 & 41 & 37 & 42 \\ 27 & 9 & 5 & 6 \end{bmatrix}$$

In this neighborhood, the image is primarily red, and therefore, the intensity of the red component is high. The green component has average intensity, and the intensity of the blue component is low.

Secondary colors are obtained by adding two of the three primary colors. While full-color representation yields high-quality images, in practice, it is found that 256 colors are adequate for most purposes, reducing the execution time and storage requirements.

14.1.2 The XYZ Color Model

While the RGB model can generate color corresponding to any wavelength in the visible spectrum, it is found that the values of some of the components become negative. As physical realization of negative color sources is not possible, we are left with two options. One option is to ignore those colors which require negative component values. As a second option, a color model, called XYZ, is defined. The conversion between the RGB and XYZ color models is given by the following equations.

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.411 & 0.342 & 0.178 \\ 0.222 & 0.707 & 0.071 \\ 0.020 & 0.130 & 0.939 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 3.063 & -1.393 & -0.476 \\ -0.969 & 1.876 & 0.042 \\ 0.068 & -0.229 & 1.069 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

The Y component corresponds to luminance or perceived brightness of the color. The values $\{X, Y, Z\}$ are called tristimulus values and can be normalized by dividing with $(X + Y + Z)$. Alternate transform matrices are possible.

14.1.3 The CMY and CMYK Color Models

Most color printers and copiers use these models. The CMY (cyan, magenta, and yellow) model can be obtained from the RGB model using the relationship, assuming color values have been normalized to the range 0–1,

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

The model is shown in Fig. 14.6. Note that cyan subtracts (absorbs) red component, and therefore, when white light is reflected from an object with cyan color, the red

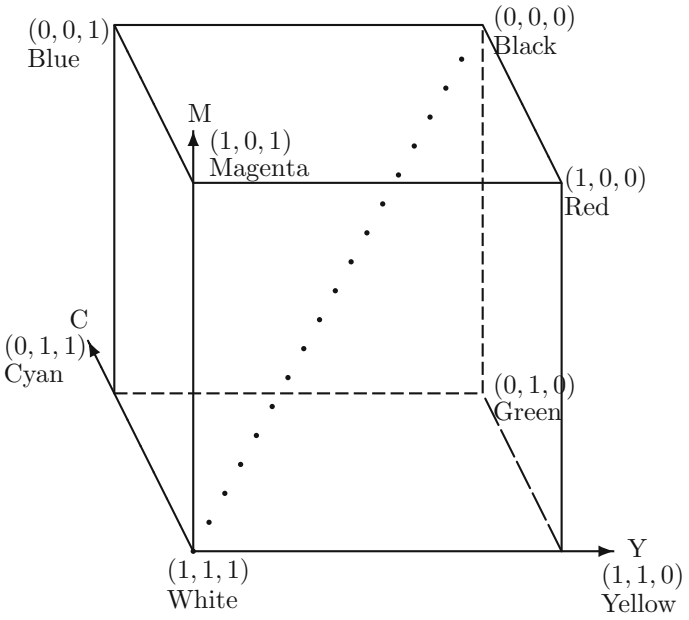


Fig. 14.6 The CMY color model

component will be zero. Similarly, magenta and yellow surfaces do not reflect green and blue, respectively. The combination of these colors, in equal proportion, does not produce a proper black color (essential for printing), as expected. A black component K is added as the fourth component to get a proper black color.

Figure 14.7a and b shows, respectively, a 256 × 256 color image in RGB and CMY formats. The whitish area of the RGB image has become darker in the CMY image and vice versa. The pinkish area has become greenish, as the red and blue component intensities are high compared with that of green. The yellowish area has become bluish. The cyan, magenta, and yellow component pixel values at coordinates (73 : 76, 173 : 176), respectively, are

$$xc = \begin{bmatrix} 10 & 9 & 9 & 7 \\ 9 & 8 & 9 & 8 \\ 9 & 9 & 10 & 7 \\ 7 & 8 & 9 & 7 \end{bmatrix} \quad xm = \begin{bmatrix} 64 & 59 & 62 & 65 \\ 63 & 58 & 63 & 66 \\ 63 & 58 & 63 & 66 \\ 63 & 57 & 64 & 67 \end{bmatrix} \quad xy = \begin{bmatrix} 33 & 30 & 32 & 32 \\ 32 & 29 & 32 & 33 \\ 32 & 30 & 33 & 33 \\ 32 & 31 & 34 & 31 \end{bmatrix}$$

The intensities of the RGB components in this neighborhood have been given earlier. It can be verified that these values are obtained by subtracting the RGB values from 255.

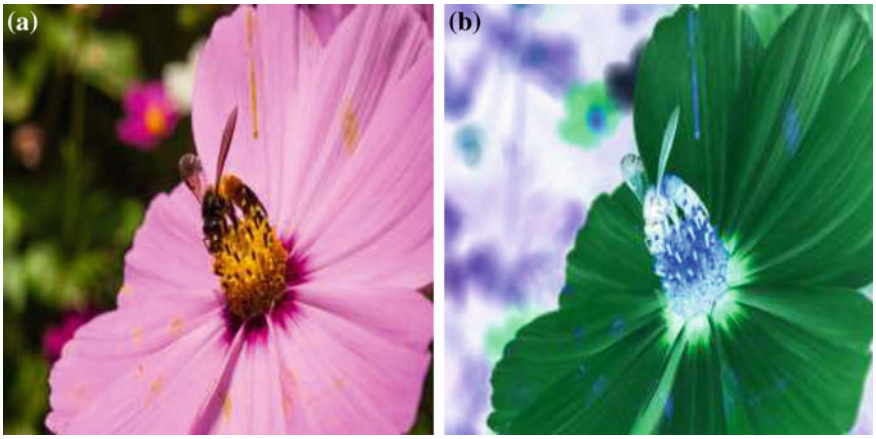


Fig. 14.7 **a** A 256 × 256 color image in RGB format; **b** the image in CMY format

The cyan, magenta, and yellow component pixel values at coordinates (63 : 66, 89 : 92), respectively, are

$$xc = \begin{bmatrix} 4 & 3 & 3 & 7 \\ 2 & 4 & 3 & 5 \\ 2 & 3 & 3 & 5 \\ 3 & 5 & 5 & 8 \end{bmatrix} \quad xm = \begin{bmatrix} 146 & 141 & 139 & 143 \\ 137 & 133 & 131 & 135 \\ 130 & 126 & 125 & 128 \\ 123 & 118 & 116 & 121 \end{bmatrix}$$

$$xy = \begin{bmatrix} 167 & 171 & 166 & 165 \\ 180 & 188 & 192 & 190 \\ 200 & 214 & 218 & 213 \\ 228 & 246 & 250 & 249 \end{bmatrix}$$

14.1.4 The HSI Color Model

In the HSI (hue, saturation, and intensity) model, the intensity component is decoupled from the color information, making it highly suitable for developing image processing algorithms. Humans also describe a color using these components rather than in terms of red, green, and blue components. The significance of the 3 components are as follows:

Hue	The true color attribute identifies colors red, green, yellow, etc.
Saturation	It indicates the amount of white color mixed (color purity). More white in the color will result in a low saturation value.
Intensity	It is a measure of brightness. The intensity of a dark color is low.

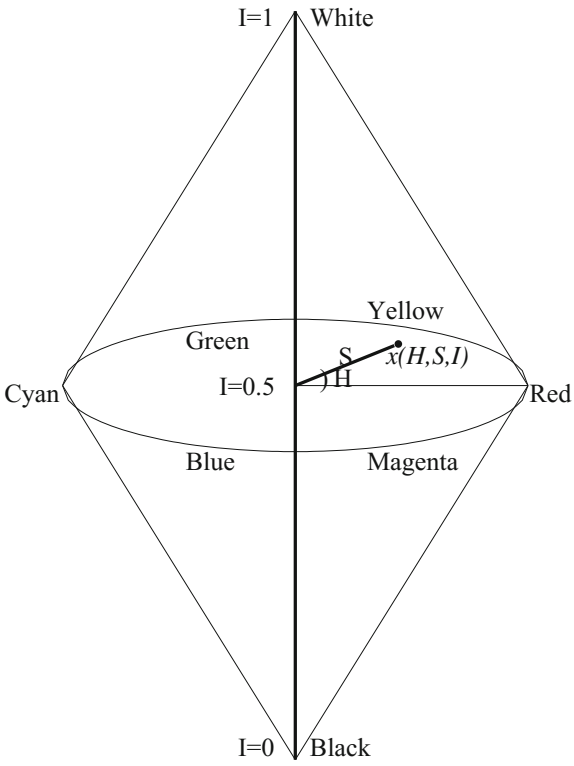


Fig. 14.8 The HSI color model

The HSI color model is shown in Fig. 14.8. This is a perception-based color model. The conversion of a RGB image to a HSI image is governed by the following equations.

$$H = \begin{cases} \theta, & \text{for } B \leq G \\ 360 - \theta, & \text{for } B > G \end{cases}, \qquad \theta = \cos^{-1}\left(\frac{0.5((R-G)+(R-B))}{\sqrt{(R-G)^2+(R-B)(G-B)}}\right)$$

$$S = 1 - \frac{3(\min(R,G,B))}{(R+G+B)}, \qquad I = \frac{(R+G+B)}{3}$$

The primary colors are separated by 120°. This model is derived by making the RGB color cube stand on its black corner with intensity value zero. Then, the white corner, with intensity value one, is at the top. The two corners are joined by the vertical intensity line, which gives the intensity component of a pixel. The intensity value I of any pixel $x(H, S, I)$ is given by the intersection of this line with a plane containing the pixel and perpendicular to the intensity line. The intensity is the average of those of the 3 components.

The red color is set as the reference for measuring the hue H of a pixel. The reference line is from the center of the figure to the red color corner. The color of a

Table 14.1 HSI model values for images with pure primary colors, black, white and pure secondary colors with intensity varying from 0 to 1

Color	RGB values	H	S	I
Red	[1 0 0]	0	1	0–1/3
Green	[0 1 0]	120°/360	1	0–1/3
Blue	[0 0 1]	240°/360	1	0–1/3
Black	[0 0 0]	0	0	0
White	[1 1 1]	0	0	1
Cyan	[0 1 1]	180°/360	1	0–2/3
Magenta	[1 0 1]	300°/360	1	0–2/3
Yellow	[1 1 0]	60°/360	1	0–2/3

pixel $x(H, S, I)$ H is the angle, measured in the anticlockwise direction, between this reference line and the line joining the pixel and the center of the figure. Therefore, $H = 0°$ for red color, and it is measured along the circumference of the circle.

The saturation component S of a pixel is the length of the line between the center of the figure and the pixel (radial distance). It indicates the purity of the color. If the color is achromatic, then $S = 0$. For a pure color, $S = 1$. This value is dependent on the number of colors contributing to the color perception. The higher the number, the lower is the value of S. The smallest value of the RGB components determines the amount of white color possible. Table 14.1 shows HSI model values for images with pure primary colors, black, white and pure secondary colors with intensity varying from 0 to 1. The values in the table can be verified using the defining equations.

The conversion of a HSI image to a RGB image is governed by the following equations.

RG sector ($0° \leq H < 120°$) :

$$B = I(1 - S)$$

$$R = I\left(1 + \frac{S\cos(H)}{\cos(60° - H)}\right)$$

$$G = 3I - (R + B)$$

GB sector ($120° \leq H < 240°$) :

$$H = H - 120°$$

$$R = I(1 - S)$$

$$G = I\left(1 + \frac{S\cos(H)}{\cos(60° - H)}\right)$$

$$B = 3I - (R + G)$$

Fig. 14.9 **a** A 256 × 256 color image; **b–d** the intensity images of the its HSI components, H, S and I in that order

BR sector ($240° \leq H \leq 360°$) :

$$H = H - 240°$$

$$G = I(1 - S)$$

$$B = I\left(1 + \frac{S\cos(H)}{\cos(60° - H)}\right)$$

$$R = 3I - (B + G)$$

Figure 14.9a–d shows a 256 × 256 color image and the intensities of its HSI components, H, S and I in that order. The light red color in most of the area contains the RGB components almost in equal proportion. But the red component has the

maximum intensity. The H value is a function of the maximum intensity of the 3 color components. Therefore, the H value has to be around 0. Further, the blue component intensity is greater than that of the green component. Therefore, the H values are high, and the component image is almost white. In reddish and yellowish areas, the color is almost pure, and therefore, the saturation value is high, and the S component is almost white. In the I component image, the dark color areas are dark and bright color areas are bright. That is, the intensity is proportional to the average intensity of the components. The H, S, and I component pixel values at coordinates (73 : 76, 173 : 176), respectively, are

$$\begin{bmatrix} 0.9031 & 0.9020 & 0.9046 & 0.9040 \\ 0.9031 & 0.9020 & 0.9031 & 0.9040 \\ 0.9031 & 0.9036 & 0.9046 & 0.9058 \\ 0.9068 & 0.9110 & 0.9083 & 0.8984 \end{bmatrix} \begin{bmatrix} 0.1292 & 0.1184 & 0.1254 & 0.1377 \\ 0.1286 & 0.1179 & 0.1286 & 0.1383 \\ 0.1286 & 0.1153 & 0.1259 & 0.1396 \\ 0.1312 & 0.1121 & 0.1292 & 0.1455 \end{bmatrix}$$

$$\begin{bmatrix} 0.8601 & 0.8719 & 0.8654 & 0.8641 \\ 0.8641 & 0.8758 & 0.8641 & 0.8601 \\ 0.8641 & 0.8732 & 0.8614 & 0.8614 \\ 0.8667 & 0.8745 & 0.8601 & 0.8627 \end{bmatrix}$$

The H, S, and I component pixel values at coordinates (63 : 66, 89 : 92), respectively, are

$$\begin{bmatrix} 0.0189 & 0.0268 & 0.0247 & 0.0205 \\ 0.0372 & 0.0470 & 0.0512 & 0.0467 \\ 0.0567 & 0.0681 & 0.0710 & 0.0666 \\ 0.0772 & 0.0891 & 0.0920 & 0.0891 \end{bmatrix} \begin{bmatrix} 0.4107 & 0.4400 & 0.4158 & 0.4000 \\ 0.4955 & 0.5432 & 0.5695 & 0.5517 \\ 0.6189 & 0.7085 & 0.7351 & 0.6993 \\ 0.8029 & 0.9318 & 0.9619 & 0.9535 \end{bmatrix}$$

$$\begin{bmatrix} 0.5856 & 0.5882 & 0.5974 & 0.5882 \\ 0.5830 & 0.5752 & 0.5739 & 0.5686 \\ 0.5660 & 0.5516 & 0.5477 & 0.5477 \\ 0.5373 & 0.5176 & 0.5150 & 0.5059 \end{bmatrix}$$

From the values of the corresponding RGB components given earlier, the last value in the H component is

$$\cos^{-1}((0.5((247-134)+(247-6)))/\sqrt{(247-134)^2+(247-6)(134-6)}) = 0.5595$$

After normalizing, the value becomes $0.5595/(2\pi) = 0.0891$. The last value in the S component is $1-(3\times 6)/(247+134+6) = 0.9535$. The last value in the I component is $(247+134+6)/(3\times 255) = 0.5059$.

The color images in RGB format can be processed either using the vector-valued pixels or using the basis color components individually. However, it is found that processing of images using some other formats is also desirable. Humans view an image in terms of luminance and chrominance. Luminance is a measure of the brightness and contrast of a pixel. Chrominance is the difference, at the same brightness, between a reference color and a color.

14.1.5 The NTSC Color Model

This format is used for television in some countries. The advantage is that it is suitable for both color and monochrome television. The conversion between the formats can be carried using a transformation and its inverse. The luminance (intensity) is represented by the Y component, and I and Q carry color information jointly, hue and saturation.

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.274 & -0.322 \\ 0.211 & -0.523 & 0.312 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

For a gray scale image with no color, as the RGB components are equal, the first row of the transformation matrix adds to 1 and the other two add to zero. In finding the Y component, more weight is given to the green component in order to match the response of the human visual system. The inverse transformation is

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.0 & 0.956 & 0.621 \\ 1.0 & -0.272 & -0.647 \\ 1.0 & -1.106 & 1.703 \end{bmatrix} \begin{bmatrix} Y \\ I \\ Q \end{bmatrix}$$

Figure 14.10a–d show a 256×256 color image in NTSC format and its Y, I and Q components, respectively. The Y component is a gray level version of the color image. The Y, I, and Q component pixel values at coordinates $(73:76, 173:176)$, respectively, are

$$\begin{bmatrix} 0.8262 & 0.8402 & 0.8324 & 0.8278 \\ 0.8301 & 0.8441 & 0.8301 & 0.8239 \\ 0.8301 & 0.8425 & 0.8285 & 0.8251 \\ 0.8325 & 0.8455 & 0.8269 & 0.8237 \end{bmatrix} \begin{bmatrix} 0.0871 & 0.0803 & 0.0860 & 0.0939 \\ 0.0871 & 0.0803 & 0.0871 & 0.0939 \\ 0.0871 & 0.0792 & 0.0860 & 0.0963 \\ 0.0918 & 0.0817 & 0.0907 & 0.0948 \end{bmatrix}$$

$$\begin{bmatrix} 0.0826 & 0.0769 & 0.0806 & 0.0884 \\ 0.0826 & 0.0769 & 0.0826 & 0.0884 \\ 0.0826 & 0.0748 & 0.0806 & 0.0892 \\ 0.0843 & 0.0724 & 0.0823 & 0.0937 \end{bmatrix}$$

The Y values are close to 1, since the neighborhood is almost white. The transformation can be verified from the intensities of the RGB components in this neighborhood given earlier. For example, the last values in the matrices are obtained as

$$\begin{bmatrix} 0.8237 \\ 0.0948 \\ 0.0937 \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.274 & -0.322 \\ 0.211 & -0.523 & 0.312 \end{bmatrix} \frac{1}{255} \begin{bmatrix} 248 \\ 188 \\ 224 \end{bmatrix}$$

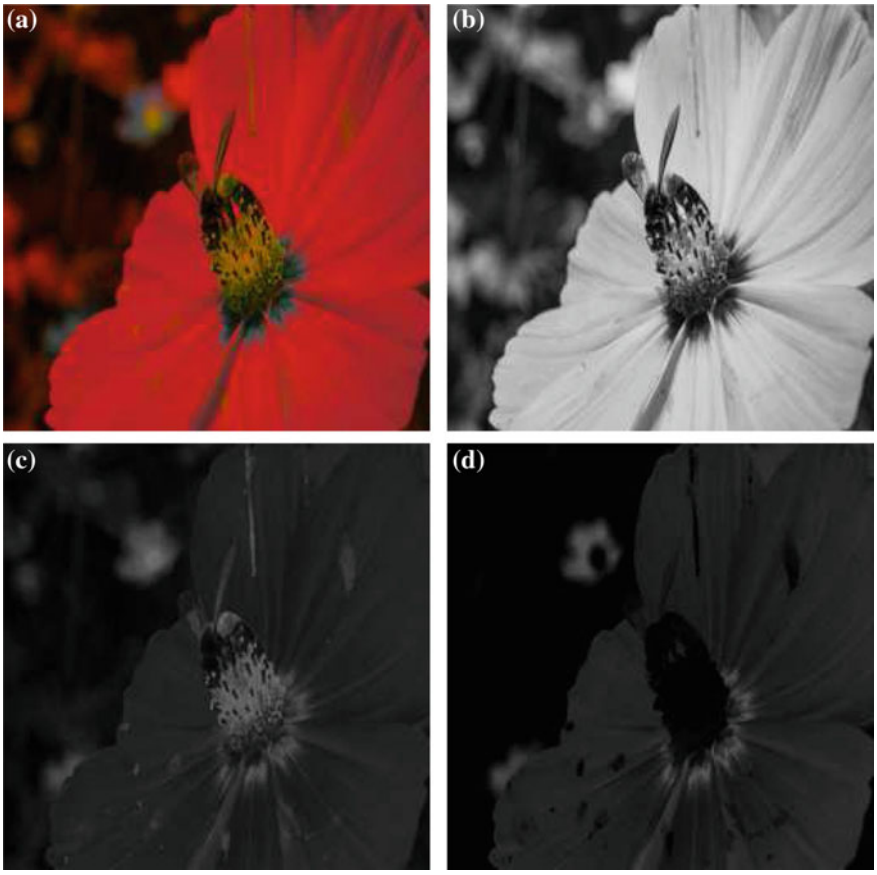


Fig. 14.10 **a** A 256 × 256 color image in NTSC format; **b** the Y component; **c** the I component; **d** the Q component

14.1.6 The YCbCr Color Model

The YCbCr model is mostly used in digital video. The YCbCr model is a format in which *Y* represents the intensity and Cb and Cr represent the chrominance. Cb component is the difference between blue component and a reference value. Cr component is the difference between red component and a reference value. The energy of an image is more evenly distributed among its three components in the RGB format. In the YCbCr format, the intensity carries most of the energy. Therefore, the chrominance component can be effectively compressed requiring reduced storage requirements.

The luminance is defined as a weighted average of that of the three components. This is due to the response of the human eye for different colors. Let the intensity values of an image from 0 to 255 be scaled to 0–1 obtained by dividing by 255. The conversion between the formats can be carried using a transformation and its inverse.

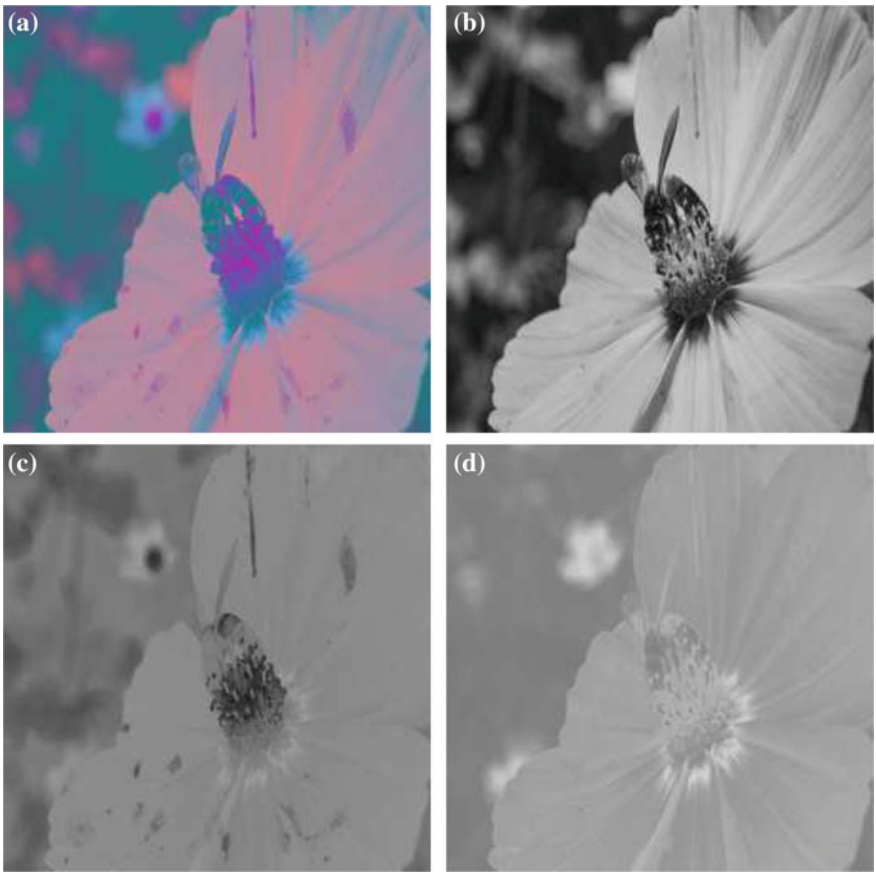


Fig. 14.11 **a** A 256 × 256 color image in YCbCr format; **b** the Y component; **c** the Cb component; **d** the Cr component

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix} + \begin{bmatrix} 65.481 & 128.553 & 24.966 \\ -37.797 & -74.203 & 112.000 \\ 112.000 & -93.786 & -18.214 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

In this formula, let the RGB input values be 0–1. Then, output Y varies from 16 to 235. Outputs Cb and Cr vary from 16 to 240. Scaling the output by 255, we get the outputs in the range 0–1. The inverse transformation is

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 0.0046 & 0.0000 & 0.0063 \\ 0.0046 & -0.0015 & -0.0032 \\ 0.0046 & 0.0079 & 0.0000 \end{bmatrix} \left[\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} - \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix} \right]$$

Figure 14.11a–d shows a 256 × 256 color image in YCbCr format and its Y, Cb, and Cr components, respectively. The Y, Cb, and Cr component pixel values at coordinates (73 : 76, 173 : 176), respectively, are

$$\begin{bmatrix} 0.7723 & 0.7843 & 0.7776 & 0.7737 \\ 0.7757 & 0.7877 & 0.7757 & 0.7704 \\ 0.7757 & 0.7863 & 0.7743 & 0.7714 \\ 0.7777 & 0.7889 & 0.7729 & 0.7702 \end{bmatrix} \begin{bmatrix} 0.5240 & 0.5228 & 0.5228 & 0.5251 \\ 0.5240 & 0.5228 & 0.5240 & 0.5251 \\ 0.5240 & 0.5217 & 0.5228 & 0.5245 \\ 0.5228 & 0.5183 & 0.5217 & 0.5291 \end{bmatrix}$$
$$\begin{bmatrix} 0.5863 & 0.5800 & 0.5848 & 0.5926 \\ 0.5863 & 0.5800 & 0.5863 & 0.5926 \\ 0.5863 & 0.5785 & 0.5848 & 0.5943 \\ 0.5897 & 0.5791 & 0.5883 & 0.5952 \end{bmatrix}$$

The transformation can be verified from the intensities of the RGB components in this neighborhood given earlier. For example, the last values in the matrices are obtained as

$$\begin{bmatrix} 196.3907 \\ 134.9184 \\ 151.7816 \end{bmatrix} = \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix} + \begin{bmatrix} 65.481 & 128.553 & 24.966 \\ -37.797 & -74.203 & 112.000 \\ 112.000 & -93.786 & -18.214 \end{bmatrix} \frac{1}{255} \begin{bmatrix} 248 \\ 188 \\ 224 \end{bmatrix}$$

Normalizing the output values, dividing by 255, we get the values 0.7702, 0.5291, and 0.5952.

14.2 Pseudocoloring

Pseudocoloring is often used to color gray level images for easier visual interpretation of their aspects. One of the methods used is intensity slicing.

14.2.1 Intensity Slicing

The histogram of the image is computed. Then, each range of the gray levels is assigned a color. The set of colors to be assigned is called the color map. A color map is a matrix with 3 columns, and each row shows the RGB values from 0 to 1. The number of rows of the color map is the number of partitions of the histogram.

Consider the 256×256 gray level image and its histogram shown in Fig. 14.12a and b. The background of the image is white with gray level 255. The 4 objects are club, heart, diamond, and spade suits of playing cards with gray levels 0, 64, 128, 192, respectively. The histogram shows that there are 2022, 2049, 1564, and 1909 pixels with these gray level values. Obviously, the diamond suit is the smallest, and the heart suit is the largest. There are 57992 pixels in the background, which is not shown in the histogram. The pixel counts add up to

$$256 \times 256 = 65536 = 2022 + 2049 + 1564 + 1909 + 57992$$

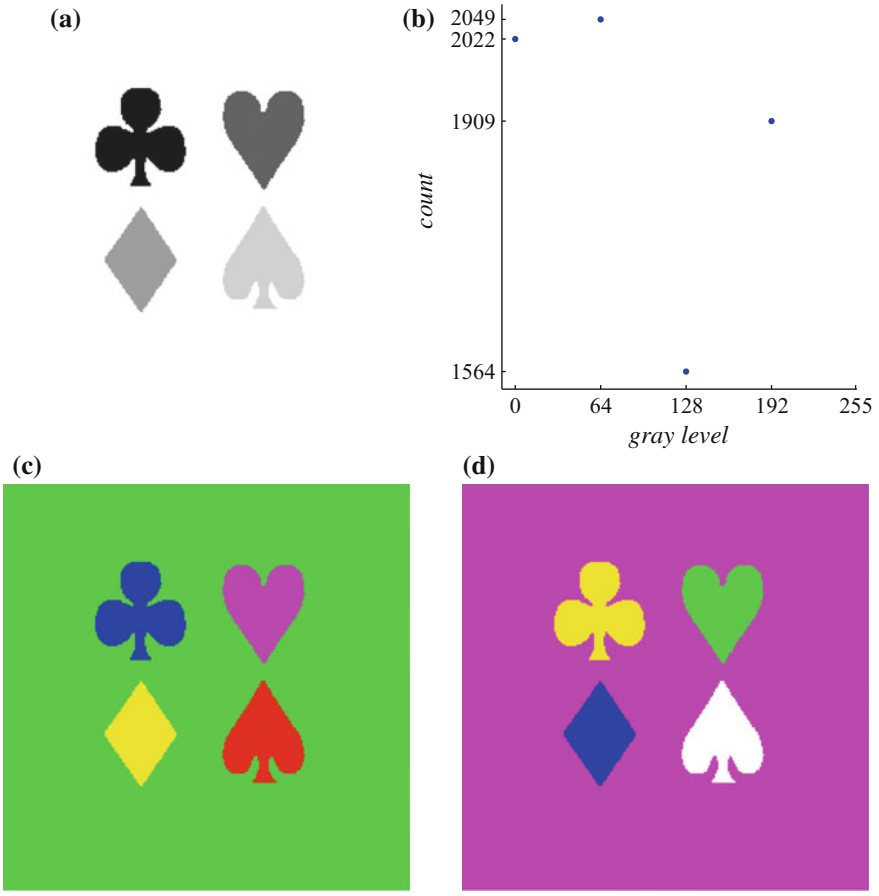


Fig. 14.12 **a** A gray level image; **b** its histogram; **c**, **d** coloring the image with two different color maps

We have to assign colors to each object. Let the two color maps be

$$color\_map1 = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \qquad color\_map2 = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}$$

The 5 ranges of the histogram values and their color assignment are shown in Table 14.2. Figure 14.12c and d shows the pseudocolored images with color map 1 and 2, respectively. The background color of the first one is green and that of the second is magenta. Typically, 256 colors are used, which is adequate for most appli-

Table 14.2 Two color maps

Gray level range	0	1–64	65–128	129–192	255
*color_map*1	Blue	Magenta	Yellow	Red	Green
*color_map*2	Yellow	Green	Blue	White	Magenta

cations and requires much less storage than full-color representation using 24 bits for each pixel. Another way to pseudocolor images is to specify suitable functions to yield the RGB values for each gray level.

14.3 Color Image Processing

There are two basic ways a color image can be processed.

- The RGB image can be processed using pixels with vector values, or each of its color components can be processed separately and the partial results combined.
- The RGB image can be converted to perception-based models, in which the intensity component is separated from the color components, and the intensity component is processed using algorithms for gray level images. The processed intensity component is recombined with the color components to obtain the processed color image.

The suitable approach is to be chosen depending upon the processing requirements. The block diagram of the approaches is shown in Fig. 14.13a–c.

14.3.1 Image Complement

The complement of an image is its negative version. It is obtained by reversing the shades of gray or colors. With the intensity range normalized to 0–1, the complement of a color or gray level image $x(m, n)$ is obtained by subtracting its values from 1, $1-x(m, n)$. The complement of a binary image is its logical complement. Sometimes, complementing a part of the intensity range is also carried out. Complements are useful to highlight certain features. In complementing a color image, each component is individually complemented. Figure 14.14a and b shows a RGB image and its complement. The flowers are yellow, and the values of the R and G components are about equal with that of the blue component is close to zero. Therefore, the flowers look blue in the complement. The dark areas in the image have become white in its

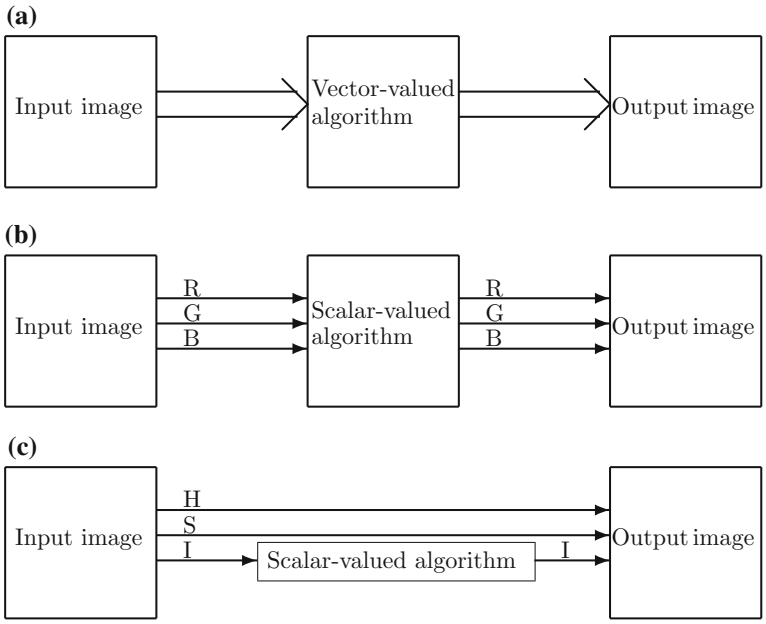


Fig. 14.13 Color image processing

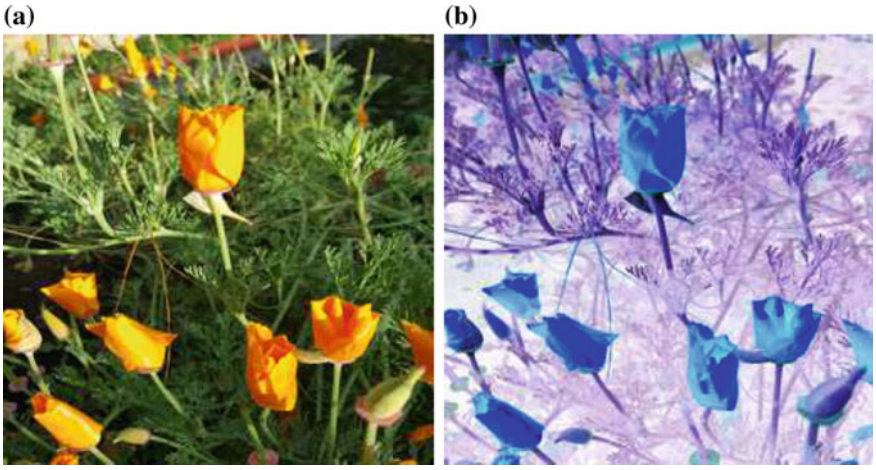


Fig. 14.14 **a** A RGB image and **b** its complement

complement. In other areas of the image, the values of the R and G components are about equal with that of the blue component about half of that. Therefore, these areas look light blue in the complement.

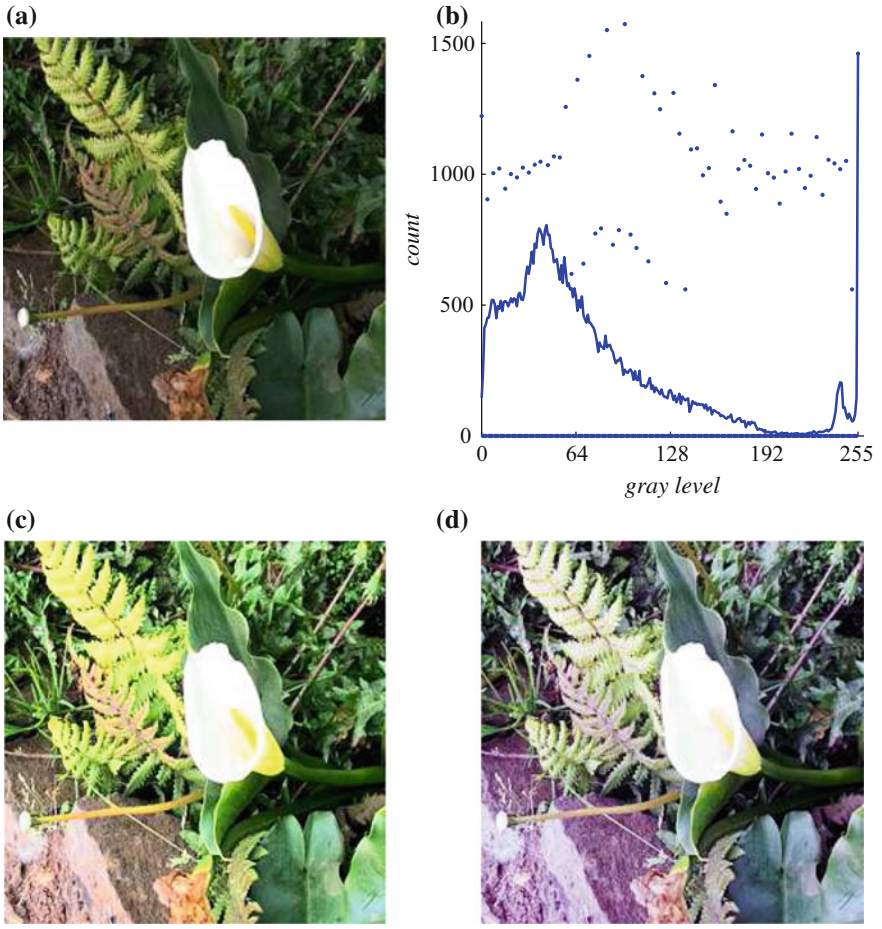


Fig. 14.15 **a** A RGB image; **b** the histogram of its intensity component and that of its equalized version; **c** the histogram-equalized image obtained after adjusting the intensity component alone; **d** the histogram-equalized image after adjusting all the 3 color components individually

14.3.2 Contrast Enhancement

The contrast can be enhanced by histogram equalization. Figure 14.15a shows a RGB image. The image is converted to HSI type. The histogram of its intensity component is shown by a continuous line in Fig. 14.15b. The equalized histogram is also shown by dots in Fig. 14.15b. The image is reconstructed with the modified intensity component and the unchanged color components. It is shown in Fig. 14.15c and is brighter than the original. However, while the colors remain unchanged, their intensity looks somewhat dimmed. The histogram-equalized image can be improved by increasing the saturation component slightly. Figure 14.15d shows the histogram-

Fig. 14.16 **a** A RGB image; **b** the averaged image using its RGB components; **c** the averaged image using its HSI intensity component; **d** the averaged image using all its HSI components

equalized image obtained by equalizing its 3 components separately. Changes in color are noticeable. Therefore, for this type of processing, modifying the intensity component with further adjustments seems better.

14.3.3 Lowpass Filtering

Figure 14.16a and b show a 256×256 image and its blurred version using a 11×11 averaging filter. The filter is applied to each of its 3 RGB components separately, and then, the image is reconstructed (Fig. 14.16b) using the filtered components. The image gets blurred, as averaging is lowpass filtering. Next, the filter is applied to its intensity component of its HSI version, and then, the image is reconstructed using the filtered intensity component and the unchanged color components (Fig. 14.16c).

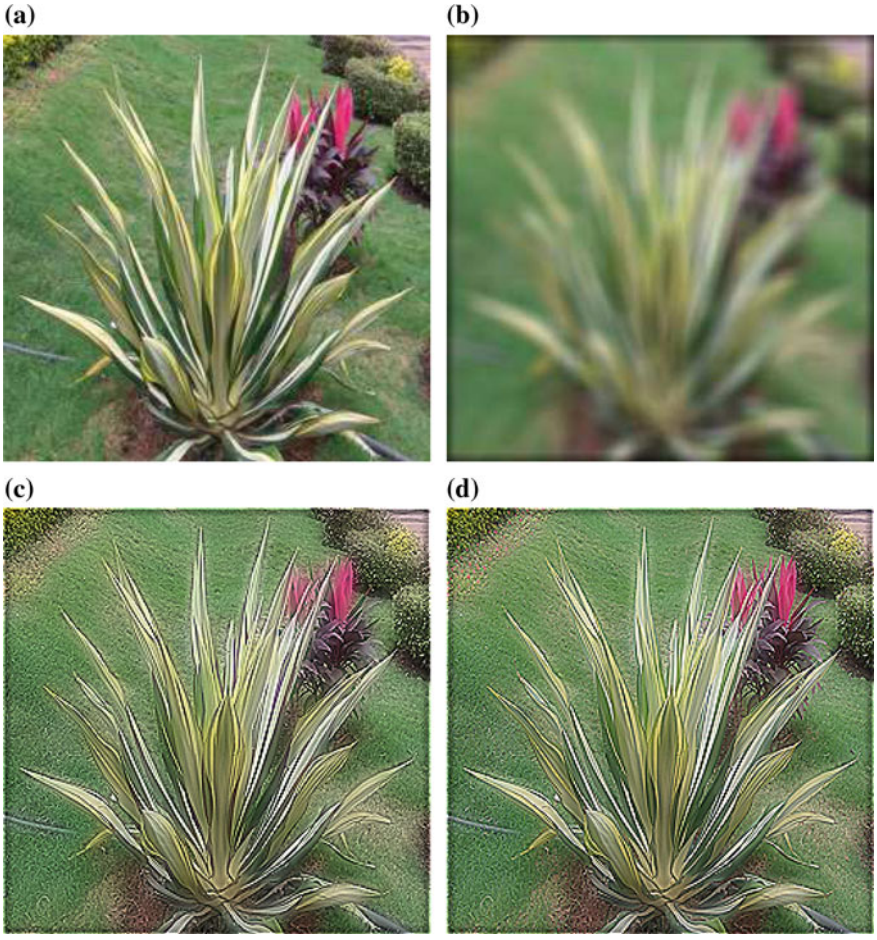


Fig. 14.17 **a** A RGB image; **b** the blurred image using its RGB components; **c** enhanced image obtained by highpass filtering of its HSI intensity component alone; **d** enhanced image obtained by highpass filtering of its RGB components

In this case, the image gets less blurred and the color composition is also changed. Next, the filter is applied to each of its 3 HSI components separately, and then, the image is reconstructed (Fig. 14.16d) using the filtered components. Due to averaging of the color components, it is clearly seen that we get new colors. The conclusion is that processing the 3 RGB components separately seems better.

14.3.4 Highpass Filtering

Figure 14.17a and b shows a 256×256 image $x(m, n)$ and its blurred version $xb(m, n)$ using a 11×11 averaging filter. The HSI intensity component of the blurred image

$xb(m, n)$ is convolved with the Laplacian mask $h(m, n)$ to get the highpass filtered image $xf(m, n)$.

$$h(m, n) = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

The corresponding enhanced image is obtained using the equation

$$xenh(m, n) = xb(m, n) - xf(m, n),$$

is shown in Fig. 14.17c. The enhanced image obtained by highpass filtering of its RGB components separately is shown in Fig. 14.17d, in which the sharpening is better. Filtering with a Laplacian mask results in negative values of the pixels in the filtered image. It has to be rescaled for proper display.

14.3.5 Median Filtering

Figure 14.18a and b shows a 256×256 image $x(m, n)$ and its noisy version $xn(m, n)$ with salt-and-pepper noise. The RGB components are separately median filtered, and the results are combined (Fig. 14.18c). The noise is almost removed. Figure 14.18d shows the result of filtering its HSI intensity component only. As the noise spreads to all the components in RGB to HSI conversion, only part of the noise is removed.

14.3.6 Edge Detection

In finding the edges in gray level images, typically, the gradients are found in two directions and the square root of the sum of their squares is the magnitude of the gradient. In digital image processing, the derivatives are approximated by differences of gray level values. Different operators are available to approximate the gradients using the convolution operation. The result of applying the operators is subjected to a threshold in finding the edge map of an image.

In finding the edges in color images, we can follow the same procedure for each of the 3 RGB components. The 3 outputs are added and then subjected to a threshold in finding the edge map of a color image. While the results are good, it turns out that using vector-valued algorithms yields a better edge map. Let the partial derivatives of the RGB components along the two directions, at each pixel, be

$$\left\{\frac{\partial R}{\partial x}, \frac{\partial G}{\partial x}, \frac{\partial B}{\partial x}\right\} \quad \text{and} \quad \left\{\frac{\partial R}{\partial y}, \frac{\partial G}{\partial y}, \frac{\partial B}{\partial y}\right\}$$

Fig. 14.18 **a** A RGB image; **b** the image with salt-and-pepper noise; **c** median filtering of its RGB components; **d** median filtering of its HSI intensity component only

The partial derivatives are approximated using gradient operators, such as Sobel. Then,

$$gxx = \left(\frac{\partial R}{\partial x}\right)^2 + \left(\frac{\partial G}{\partial x}\right)^2 + \left(\frac{\partial B}{\partial x}\right)^2, \qquad gyy = \left(\frac{\partial R}{\partial y}\right)^2 + \left(\frac{\partial G}{\partial y}\right)^2 + \left(\frac{\partial B}{\partial y}\right)^2,$$

$$gxy = \frac{\partial R}{\partial x}\frac{\partial R}{\partial y} + \frac{\partial G}{\partial x}\frac{\partial G}{\partial y} + \frac{\partial B}{\partial x}\frac{\partial B}{\partial y}$$

The angle of the gradient is given by

$$\theta 1 = 0.5 \tan^{-1}\left(\frac{2gxy}{gxx - gyy}\right), \qquad \theta 2 = \theta 1 + \frac{\pi}{2}$$

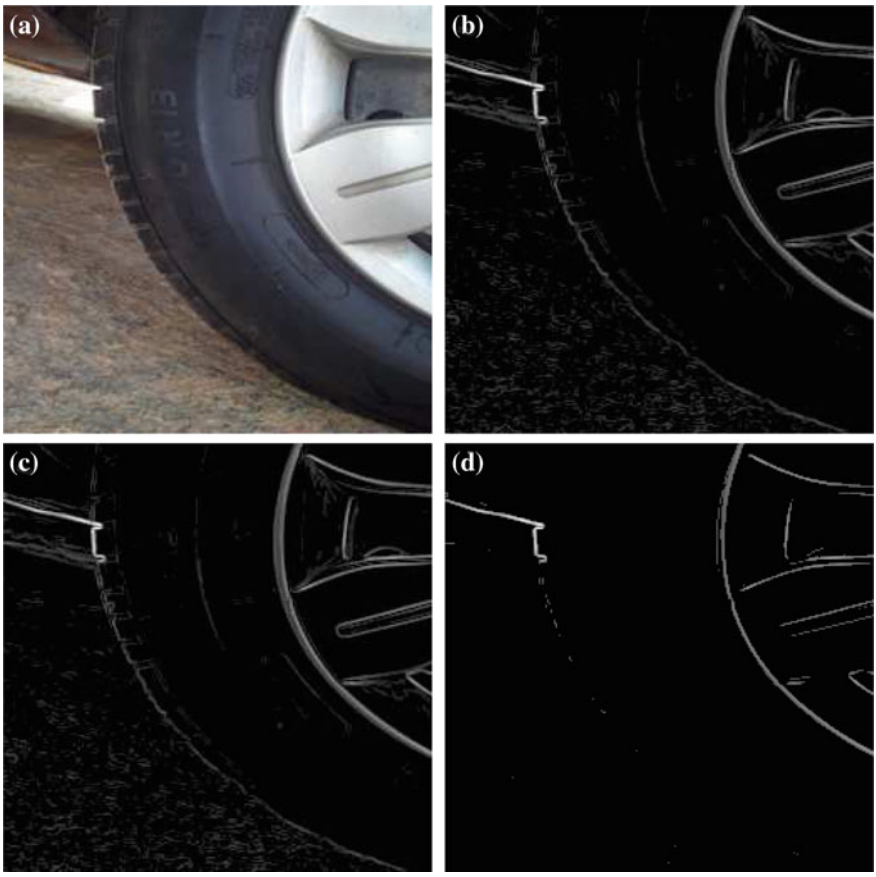


Fig. 14.19 **a** A 256 × 256 image; **b** edge map by vector-valued algorithm with threshold 0.1 in the intensity range 0–1; **c** edge map by processing RGB components separately with the same threshold; **d** edge map by vector-valued algorithm with threshold 0.3

The magnitude of the gradient in the direction of $\theta 1$ and $\theta 2$ is computed using the expression

$$g = \sqrt{0.5((gxx + gyy) + (gxx - gyy)\cos(2\theta) + 2gxy\sin(2\theta))}$$

The maximum of the two values is taken as the magnitude of the gradient, which is then thresholded.

Figure 14.19a shows a 256 × 256 RGB image. Its edge maps, obtained by vector-valued algorithm, using the Sobel operator, with thresholds 0.1 and 0.3 in the intensity range 0–1, are shown, respectively, in Fig. 14.19b and d. Figure 14.19c shows the edge map obtained by processing RGB components separately using the Sobel mask with the threshold 0.1.

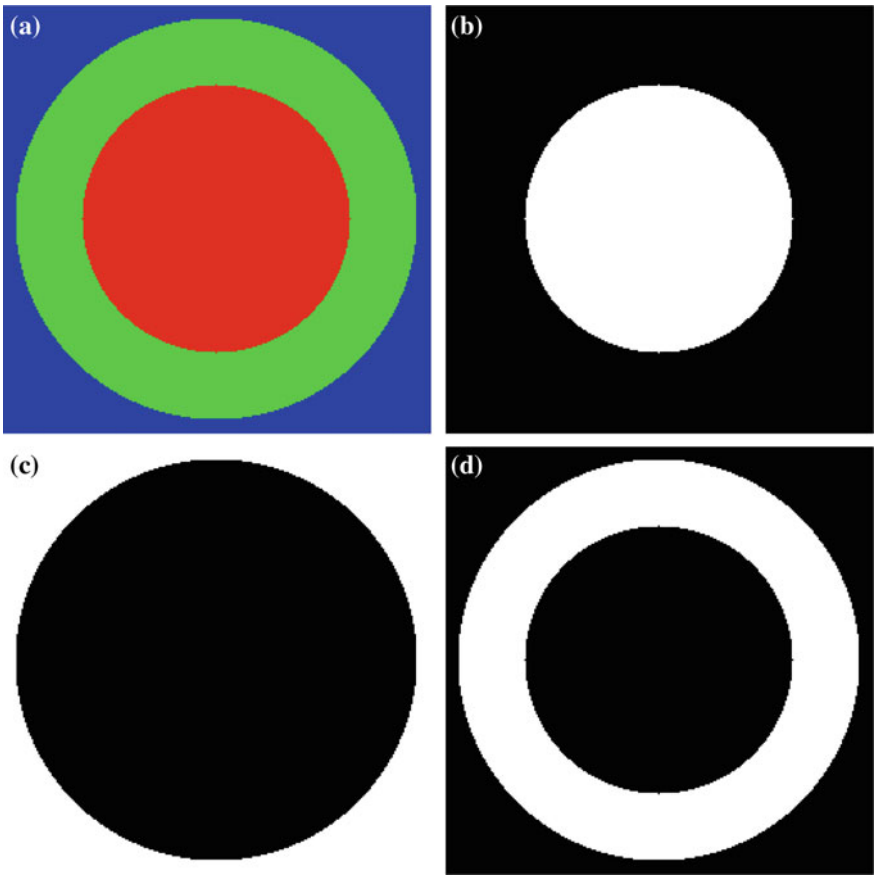


Fig. 14.20 **a** A 256 × 256 image; **b** segmentation of the *red disk*; **c** segmentation of the *blue areas*; **d** segmentation of the *green ring*

14.3.7 Segmentation

Let the average color of the region, to be segmented, of the RGB image $x(m, n)$ be $\{ar, ag, ab\}$. Then, the square root of the sum of the Euclidean distance between the reference and image pixel color components is computed, and then, it is subjected to a threshold. The distances are computed using the equation.

$$D(m, n) = \sqrt{(xr(m, n) - ar)^2 + (xg(m, n) - ag)^2 + (xb(m, n) - ab)^2)}$$

The pixels with distances above the threshold are not in the segment and are assigned the value zero. The other pixels are assigned the value 1.

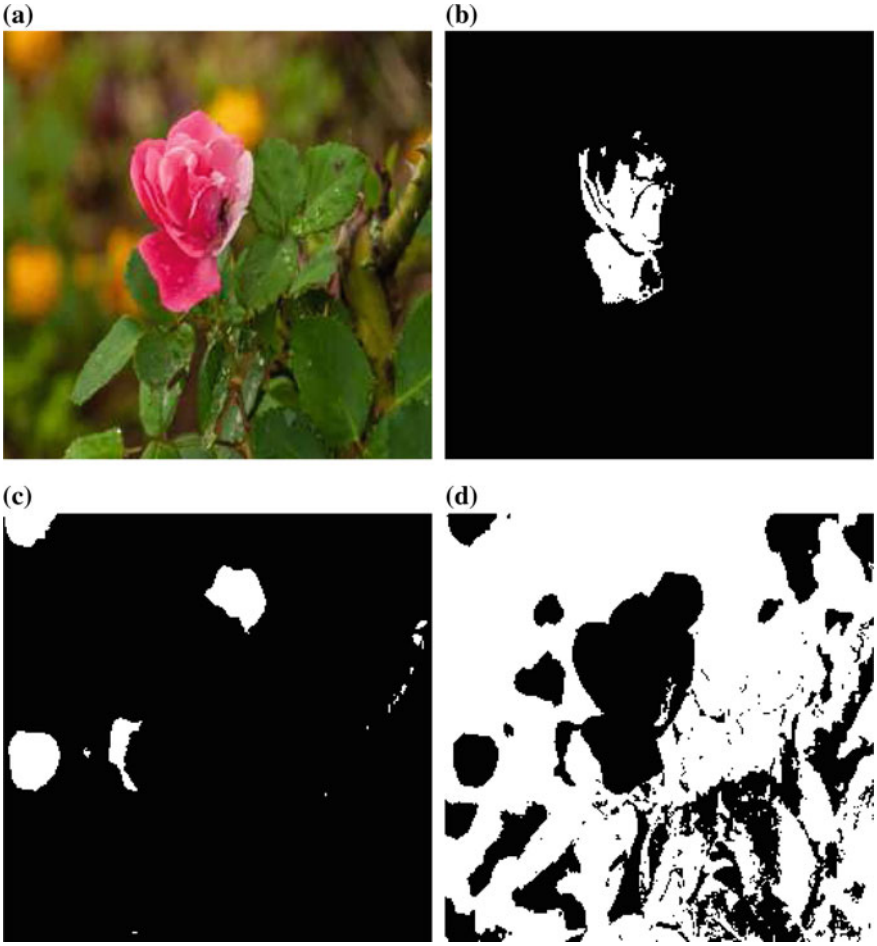


Fig. 14.21 **a** A 256 × 256 image; **b** segmentation of the *red flower*; **c** segmentation of the *yellow areas*; **d** segmentation of the *green areas*

Consider the synthetic image shown in Fig. 14.20a. Let us try to segment the red disk. The average value is $(1, 0, 0)$. The distance of all the red pixels is zero and that of the other colors will be $\sqrt{2}$. With a threshold value less than $\sqrt{2}$, the red disk is segmented as shown in Fig. 14.20b. The other two color segments, shown in Fig. 14.20c and d, are isolated similarly with reference color vectors $(0, 0, 1)$ and $(0, 1, 0)$.

Figure 14.21a shows a 256×256 image. Figure 14.21b–d shows the segmentation of the red flower and the yellow and green areas, respectively.

14.4 Summary

- Since most naturally occurring images are color images and human vision system is more sensitive to color than to gray levels, color images are important.
- A color is typically composed of 3 components.
- The pixels of a color image are vector-valued.
- Although the pixels are vector-valued requiring more processing time and storage, color images are more powerful in aiding in the visualization of the features of an image.
- In one type of representation, the intensity value of the pixels is decoupled from the color components.
- In another type of representation, each component carries both the intensity and color values.
- A commonly used color model is to compose the color using red, green, and blue components. A gray color is composed of equal amounts of the 3 color components.
- In another model, cyan, magenta, and yellow colors are used as basis colors.
- In the hue, saturation, and intensity color model, the intensity of a pixel is decoupled from its color components.
- Different color models are suitable for different applications.
- In addition to full-color images, pseudocolor images are also widely used. They require less storage and adequate for some applications.
- Operations such as enhancement, edge detection, and segmentation can be carried out with color images.
- There are three types of algorithms used to process the color images.
- In one type of algorithms, the images are processed using vector-valued algorithms.
- In another type of algorithms, the images are processed using their intensity component only.
- In yet another type of algorithms, the 3 color components are processed separately.

Exercises

14.1 Given the RGB components of a 4×4 color image, find the CMY components of its CMY model.
(i)

$$\begin{bmatrix} 229 & 226 & 238 & 214 \\ 239 & 238 & 225 & 233 \\ 252 & 247 & 222 & 242 \\ 214 & 213 & 240 & 224 \end{bmatrix} \begin{bmatrix} 215 & 212 & 231 & 213 \\ 222 & 225 & 214 & 224 \\ 242 & 235 & 209 & 229 \\ 201 & 196 & 225 & 211 \end{bmatrix} \begin{bmatrix} 71 & 41 & 73 & 72 \\ 90 & 70 & 74 & 93 \\ 91 & 76 & 53 & 89 \\ 50 & 33 & 69 & 79 \end{bmatrix}$$

(ii)

$$\begin{bmatrix} 171 & 228 & 231 & 229 \\ 180 & 225 & 231 & 221 \\ 192 & 225 & 229 & 213 \\ 204 & 221 & 229 & 216 \end{bmatrix} \begin{bmatrix} 133 & 54 & 64 & 44 \\ 130 & 51 & 63 & 42 \\ 99 & 48 & 63 & 40 \\ 71 & 50 & 65 & 44 \end{bmatrix} \begin{bmatrix} 125 & 120 & 130 & 113 \\ 125 & 115 & 129 & 105 \\ 119 & 114 & 127 & 102 \\ 116 & 114 & 128 & 109 \end{bmatrix}$$

(iii)

$$\begin{bmatrix} 64 & 68 & 67 & 66 \\ 61 & 59 & 65 & 65 \\ 68 & 64 & 66 & 69 \\ 74 & 71 & 68 & 69 \end{bmatrix} \begin{bmatrix} 102 & 107 & 108 & 108 \\ 93 & 100 & 106 & 104 \\ 107 & 102 & 102 & 106 \\ 115 & 110 & 103 & 107 \end{bmatrix} \begin{bmatrix} 55 & 64 & 63 & 54 \\ 55 & 56 & 63 & 59 \\ 65 & 59 & 58 & 62 \\ 69 & 66 & 60 & 56 \end{bmatrix}$$

14.2 Given the RGB components of a 4×4 color image, find the HSI components of its HSI model.
∗(i)

$$\begin{bmatrix} 232 & 234 & 235 & 235 \\ 235 & 235 & 237 & 235 \\ 236 & 236 & 234 & 234 \\ 236 & 237 & 234 & 236 \end{bmatrix} \begin{bmatrix} 49 & 48 & 57 & 71 \\ 52 & 55 & 61 & 78 \\ 53 & 61 & 61 & 84 \\ 57 & 64 & 69 & 92 \end{bmatrix} \begin{bmatrix} 162 & 169 & 174 & 188 \\ 166 & 175 & 181 & 194 \\ 169 & 180 & 185 & 200 \\ 172 & 184 & 191 & 206 \end{bmatrix}$$

(ii)

$$\begin{bmatrix} 210 & 148 & 149 & 155 \\ 201 & 152 & 146 & 150 \\ 190 & 152 & 140 & 144 \\ 180 & 157 & 134 & 137 \end{bmatrix} \begin{bmatrix} 103 & 37 & 29 & 34 \\ 64 & 37 & 28 & 34 \\ 48 & 35 & 27 & 34 \\ 37 & 34 & 28 & 31 \end{bmatrix} \begin{bmatrix} 137 & 111 & 120 & 125 \\ 113 & 115 & 119 & 124 \\ 125 & 120 & 115 & 122 \\ 135 & 131 & 110 & 115 \end{bmatrix}$$

(iii)

$$\begin{bmatrix} 98 & 94 & 86 & 82 \\ 94 & 90 & 83 & 80 \\ 88 & 85 & 80 & 78 \\ 80 & 79 & 76 & 74 \end{bmatrix} \begin{bmatrix} 131 & 127 & 120 & 116 \\ 124 & 122 & 116 & 112 \\ 116 & 115 & 111 & 108 \\ 109 & 106 & 103 & 103 \end{bmatrix} \begin{bmatrix} 82 & 80 & 77 & 74 \\ 80 & 80 & 78 & 74 \\ 75 & 75 & 74 & 71 \\ 69 & 70 & 69 & 67 \end{bmatrix}$$

14.3 Given the RGB components of a 4×4 color image, find the YIQ components of its NTSC model.
(i)

$$\begin{bmatrix} 236 & 239 & 242 & 247 \\ 236 & 239 & 244 & 249 \\ 235 & 240 & 245 & 250 \\ 236 & 241 & 245 & 249 \end{bmatrix} \begin{bmatrix} 193 & 190 & 187 & 186 \\ 190 & 187 & 185 & 184 \\ 189 & 185 & 184 & 182 \\ 188 & 184 & 182 & 180 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 2 & 2 \\ 0 & 3 & 6 & 6 \\ 2 & 6 & 9 & 9 \end{bmatrix}$$

∗(ii)

$$\begin{bmatrix} 209 & 210 & 232 & 250 \\ 206 & 212 & 235 & 248 \\ 208 & 213 & 233 & 238 \\ 212 & 212 & 220 & 219 \end{bmatrix} \begin{bmatrix} 32 & 28 & 37 & 43 \\ 29 & 29 & 40 & 42 \\ 31 & 33 & 43 & 39 \\ 35 & 38 & 39 & 32 \end{bmatrix} \begin{bmatrix} 51 & 54 & 66 & 76 \\ 46 & 51 & 65 & 69 \\ 47 & 49 & 61 & 61 \\ 50 & 49 & 53 & 48 \end{bmatrix}$$

(iii)

$$\begin{bmatrix} 222 & 223 & 227 & 231 \\ 226 & 227 & 232 & 235 \\ 233 & 234 & 235 & 237 \\ 238 & 239 & 238 & 238 \end{bmatrix} \begin{bmatrix} 185 & 185 & 186 & 189 \\ 183 & 183 & 183 & 186 \\ 179 & 179 & 179 & 180 \\ 175 & 174 & 174 & 174 \end{bmatrix} \begin{bmatrix} 3 & 1 & 2 & 5 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

14.4 Given the RGB components of a 4×4 color image, find the YCbCr components of its YCbCr model.

∗(i)

$$\begin{bmatrix} 142 & 142 & 150 & 149 \\ 144 & 144 & 143 & 163 \\ 145 & 144 & 152 & 157 \\ 147 & 153 & 155 & 126 \end{bmatrix} \begin{bmatrix} 15 & 15 & 24 & 24 \\ 17 & 17 & 12 & 35 \\ 18 & 18 & 16 & 24 \\ 18 & 21 & 20 & 18 \end{bmatrix} \begin{bmatrix} 44 & 44 & 49 & 48 \\ 46 & 47 & 37 & 59 \\ 47 & 46 & 43 & 51 \\ 48 & 49 & 42 & 28 \end{bmatrix}$$

(ii)

$$\begin{bmatrix} 227 & 230 & 230 & 229 \\ 224 & 215 & 214 & 215 \\ 208 & 207 & 208 & 209 \\ 191 & 197 & 198 & 195 \end{bmatrix} \begin{bmatrix} 0 & 6 & 5 & 3 \\ 4 & 0 & 0 & 0 \\ 2 & 2 & 2 & 1 \\ 0 & 6 & 6 & 3 \end{bmatrix} \begin{bmatrix} 27 & 32 & 32 & 32 \\ 30 & 20 & 19 & 22 \\ 19 & 15 & 14 & 16 \\ 13 & 15 & 14 & 13 \end{bmatrix}$$

(iii)

$$\begin{bmatrix} 226 & 226 & 226 & 225 \\ 226 & 226 & 225 & 223 \\ 226 & 226 & 225 & 222 \\ 226 & 226 & 225 & 223 \end{bmatrix} \begin{bmatrix} 239 & 239 & 238 & 235 \\ 239 & 239 & 238 & 235 \\ 239 & 239 & 238 & 235 \\ 239 & 239 & 238 & 236 \end{bmatrix} \begin{bmatrix} 219 & 221 & 221 & 220 \\ 219 & 221 & 220 & 218 \\ 219 & 221 & 220 & 218 \\ 219 & 221 & 221 & 219 \end{bmatrix}$$

14.5 A 8-bit gray level image is to be converted to a color image by pseudocoloring. What is the color map for the given color assignment.

(i)

Gray level range	0	1–64	65–128	129–192	255
color_map	Cyan	Green	Blue	magenta	Red

∗(ii)

Gray level range	0	1–64	65–128	129–192	255
color_map	Yellow	Red	magenta	Cyan	Green

(iii)

Gray level range	0	1–64	65–128	129–192	255
color_map	magenta	Yellow	Red	Cyan	Blue

14.6 Given the RGB components of a 4×4 color image, find its complement.
(i)

$$\begin{bmatrix} 76 & 69 & 93 & 147 \\ 129 & 88 & 77 & 163 \\ 67 & 114 & 142 & 163 \\ 100 & 92 & 160 & 165 \end{bmatrix} \begin{bmatrix} 96 & 90 & 121 & 179 \\ 153 & 113 & 102 & 180 \\ 99 & 142 & 169 & 191 \\ 132 & 122 & 188 & 186 \end{bmatrix} \begin{bmatrix} 50 & 47 & 58 & 82 \\ 78 & 59 & 49 & 100 \\ 34 & 69 & 96 & 111 \\ 57 & 54 & 109 & 104 \end{bmatrix}$$

(ii)

$$\begin{bmatrix} 69 & 114 & 59 & 41 \\ 78 & 82 & 141 & 56 \\ 138 & 47 & 81 & 106 \\ 128 & 131 & 52 & 79 \end{bmatrix} \begin{bmatrix} 81 & 131 & 82 & 65 \\ 90 & 99 & 167 & 82 \\ 157 & 61 & 104 & 136 \\ 153 & 150 & 70 & 106 \end{bmatrix} \begin{bmatrix} 42 & 73 & 40 & 27 \\ 49 & 48 & 93 & 37 \\ 93 & 30 & 53 & 68 \\ 83 & 88 & 35 & 51 \end{bmatrix}$$

(iii)

$$\begin{bmatrix} 128 & 164 & 139 & 142 \\ 174 & 199 & 178 & 163 \\ 179 & 204 & 204 & 169 \\ 101 & 174 & 202 & 149 \end{bmatrix} \begin{bmatrix} 160 & 190 & 169 & 171 \\ 197 & 215 & 199 & 189 \\ 200 & 217 & 218 & 193 \\ 124 & 189 & 216 & 175 \end{bmatrix} \begin{bmatrix} 74 & 93 & 71 & 88 \\ 102 & 105 & 99 & 105 \\ 90 & 100 & 109 & 100 \\ 42 & 70 & 104 & 82 \end{bmatrix}$$

14.7 Given the RGB components of a 4×4 color image with intensity values varying from 0 to 255, find the histogram-equalized version of its intensity component of its HSI model.
∗(i)

$$\begin{bmatrix} 81 & 80 & 81 & 90 \\ 78 & 73 & 84 & 100 \\ 75 & 79 & 93 & 93 \\ 74 & 84 & 102 & 91 \end{bmatrix} \begin{bmatrix} 108 & 107 & 107 & 116 \\ 105 & 99 & 110 & 129 \\ 101 & 106 & 120 & 122 \\ 100 & 110 & 130 & 120 \end{bmatrix} \begin{bmatrix} 38 & 36 & 37 & 45 \\ 37 & 31 & 41 & 55 \\ 35 & 38 & 50 & 49 \\ 34 & 43 & 59 & 49 \end{bmatrix}$$

(ii)

$$\begin{bmatrix} 58 & 110 & 89 & 25 \\ 61 & 106 & 89 & 81 \\ 64 & 83 & 130 & 157 \\ 62 & 88 & 173 & 162 \end{bmatrix} \begin{bmatrix} 79 & 129 & 112 & 39 \\ 82 & 124 & 111 & 83 \\ 86 & 100 & 130 & 139 \\ 85 & 92 & 149 & 134 \end{bmatrix} \begin{bmatrix} 30 & 73 & 45 & 7 \\ 29 & 66 & 49 & 45 \\ 27 & 43 & 70 & 90 \\ 27 & 39 & 97 & 94 \end{bmatrix}$$

(iii)

$$\begin{bmatrix} 114 & 127 & 144 & 114 \\ 129 & 136 & 131 & 84 \\ 144 & 137 & 124 & 76 \\ 143 & 139 & 145 & 120 \end{bmatrix} \begin{bmatrix} 84 & 99 & 139 & 128 \\ 106 & 120 & 135 & 95 \\ 127 & 135 & 133 & 86 \\ 143 & 148 & 155 & 130 \end{bmatrix} \begin{bmatrix} 69 & 68 & 71 & 55 \\ 98 & 71 & 57 & 39 \\ 111 & 80 & 64 & 34 \\ 89 & 90 & 96 & 76 \end{bmatrix}$$

Appendix A
Computation of the DFT

Abstract An algorithm for fast computation of the discrete Fourier transform is described. It is not an exaggeration to state that a single most important reason for the existence and continuing growth of digital signal and image processing is due to this algorithm. The algorithm is based on the divide-and-conquer strategy of developing fast algorithms. The DFT decomposes an arbitrary time-domain waveform in terms of sinusoidal waveforms. Using the half-wave symmetry of periodical signals, an arbitrary waveform can be decomposed into two components by add–subtract operation. One component is composed of the even-indexed frequency components, and the other is composed of odd-indexed ones. With a frequency shift of the latter component, the original problem becomes decomposed into two components of half the size.

A.1 The DFT Problem Formulation

Any finite-valued periodic sequence $x(n)$ of period N can be expressed by a linear combination of N complex exponentials. That is,

$$x(n) = X(0)e^{j0\frac{2\pi}{N}n} + X(1)e^{j1\frac{2\pi}{N}n} + X(2)e^{j2\frac{2\pi}{8}n} + \cdots + X(N-1)e^{j(N-1)\frac{2\pi}{N}n}$$

While the N values of $x(n)$ and the complex exponentials are known, the task is to separate the frequency components. That is to determine the values $X(k),\ k = 0, 1, \ldots, N-1$ so that the equation is satisfied. A finite sequence of values is assumed to be periodic in DFT computation. Assume that period N is an integral power of 2. In practice, this constraint is not so severe as most of the signals to be analyzed are aperiodic and zero-padding can be used to extend their length so that the number of samples is equal to 2^M for some positive integer M. In most applications, therefore, it is assumed that the number of samples is a power of two. A real sinusoid, at a given frequency, is characterized by its amplitude and phase. The mathematically equivalent complex exponential is characterized, at a given frequency, by its single

D. Sundararajan, *Digital Image Processing*, DOI 10.1007/978-981-10-6113-4

complex amplitude. Although most practical signals are real, for obtaining the highest efficiency as well as ease of use, it is a necessity to formulate the DFT algorithms using complex exponentials as basis functions, rather than real sinusoids.

A.2 Half-Wave Symmetry of Periodic Waveforms

The easiest way to understand the basics of DFT algorithms is through the half-wave symmetry of periodic waveforms. Any periodic sequence $x(n)$ of period N can be decomposed into its even and odd half-wave symmetric components $x_{eh}(n)$ and $x_{oh}(n)$, respectively. That is $x(n) = x_{eh}(n) + x_{oh}(n)$, where

$$x_{eh}(n) = \frac{1}{2}\left(x(n) + x\left(n \pm \frac{N}{2}\right)\right) \quad \text{and} \quad x_{oh}(n) = \frac{1}{2}\left(x(n) - x\left(n \pm \frac{N}{2}\right)\right)$$

The sequence values of the even half-wave symmetric waveform $x_{eh}(n)$ over any half period are the same over the preceding or succeeding half period

$$x_{eh}\left(n \pm \frac{N}{2}\right) = x_{eh}(n)$$

That is, the fundamental period of $x_{eh}(n)$ is $N/2$. The sequence values of the odd half-wave symmetric waveform $x_{oh}(n)$ over any half period are the negatives of those over the preceding or succeeding half period

$$x_{oh}\left(n \pm \frac{N}{2}\right) = -x_{oh}(n)$$

Therefore, $N/2$ values of each of $x_{eh}(n)$ and $x_{oh}(n)$ are adequate to uniquely represent them and, thereby, representing the N values of one period of $x(n)$.

A.3 The DFT and the Half-Wave Symmetry

Sequence $x_{eh}(n)$ contributes the even-indexed frequency components to the DFT representation of $x(n)$, and $x_{oh}(n)$ contributes the odd-indexed frequency components, as, from the DFT definition,

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j\frac{2\pi}{N}kn}, \quad k = 0, 1, \ldots, N-1$$

$$= \begin{cases} \sum_{n=0}^{(N/2)-1} \left(x(n) + x\left(n \pm \frac{N}{2}\right) \right) e^{-j\frac{2\pi}{N}kn} = \sum_{n=0}^{(N/2)-1} 2x_{eh}(n) e^{-j\frac{2\pi}{N}kn}, & k \text{ even} \\ \sum_{n=0}^{(N/2)-1} \left(x(n) - x\left(n \pm \frac{N}{2}\right) \right) e^{-j\frac{2\pi}{N}kn} = \sum_{n=0}^{(N/2)-1} 2x_{oh}(n) e^{-j\frac{2\pi}{N}kn}, & k \text{ odd} \end{cases}$$

It is by the repeated decomposition of a waveform into its even and odd half-wave symmetric components, along with frequency shifting, and using the temporal redundancy (in time) of these components, we extract the frequency coefficients of its constituent sinusoids. *The frequency components are eventually isolated, and they are reduced to low frequency components with frequency indices either zero or one, but with their coefficient values unchanged. Then, the first sample value of each component is its coefficient value.* The decomposition operation involves taking two values, finding their sum and difference, and storing the resulting two values. Therefore, as the sum and difference of a and b is $a \pm b$ and the plus–minus operation is basic to the algorithms, the DFT algorithms are called PM DFT algorithms. Further, as two values are input to the basic operation resulting in two values, it is found that the most efficient data structure for the algorithms is an array of two element vectors.

A.4 The PM DIF DFT Algorithm

Given a waveform $x(n)$ composed of N frequency components (let us assume $N = 8$),

$$x(n) = X(0)e^{j0\frac{2\pi}{8}n} + X(1)e^{j1\frac{2\pi}{8}n} + X(2)e^{j2\frac{2\pi}{8}n} + X(3)e^{j3\frac{2\pi}{8}n} + X(4)e^{j4\frac{2\pi}{8}n} + X(5)e^{j5\frac{2\pi}{8}n} + X(6)e^{j6\frac{2\pi}{8}n} + X(7)e^{j7\frac{2\pi}{8}n}, \ n = 0, 1, \ldots, 7$$

the first step is to decompose $x(n)$ into its even and odd half-wave symmetric components $x_{eh}(n)$ and $x_{oh}(n)$, respectively. The decomposition results in

$$a(n) = \{a_0(n), a_1(n)\} = 2\{x_{eh}(n), x_{oh}(n)\}$$
$$= 2\{X(0)e^{j0\frac{2\pi}{8}n} + X(2)e^{j2\frac{2\pi}{8}n} + X(4)e^{j4\frac{2\pi}{8}n} + X(6)e^{j6\frac{2\pi}{8}n},$$
$$X(1)e^{j1\frac{2\pi}{8}n} + X(3)e^{j3\frac{2\pi}{8}n} + X(5)e^{j5\frac{2\pi}{8}n} + X(7)e^{j7\frac{2\pi}{8}n}\}, \ n = 0, 1, 2, 3$$

The division operation by two, required in finding the symmetric components, is not carried out and hence the factor two appears in the result. Since a half-wave symmetric component is defined by its values over half the period, the values over half the period is sufficient for further processing. Therefore, the components are found

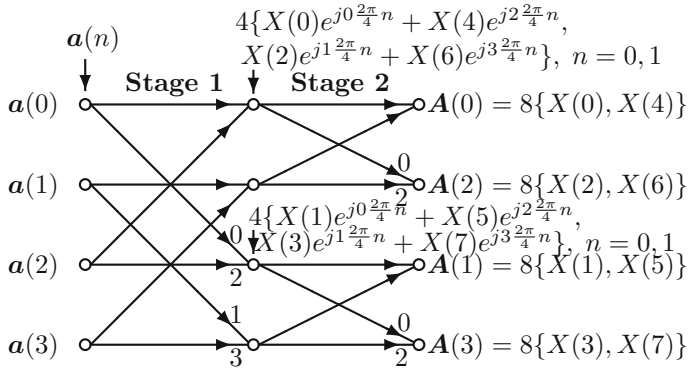


Fig. A.1 The signal-flow graph of the PM DIF DFT algorithm, with $N = 8$, showing the decomposition of a waveform

only for $n = 0, 1, 2$, and 3. We reformulate the problem of separating the frequency components of an arbitrary $x(n)$ into that of its even and odd half-wave symmetric components $x_{eh}(n)$ and $x_{oh}(n)$. We form the data structure $a(n) = \{a_0(n), a_1(n)\} = 2\{x_{eh}(n), x_{oh}(n)\},\ n = 0, 1, \ldots (N/2) - 1$, an array of two element vectors. For $N = 8$, we get an array of length four, with each element of the array being a pair of ordered complex numbers. This array is stored in the nodes at the beginning of the signal-flow graph of the algorithm shown in Fig. A.1.

A signal-flow graph is an interconnection of nodes and branches. The direction of signal flow along a branch is indicated by an arrow. A node, shown by unfilled circles, stores two values. In addition, except the first set of nodes at the beginning of the graph, each node finds the sum and difference of the two values supplied by the two incoming branches. An upper node is a node where a branch with a positive slope terminates. This type of nodes receive the first element of the vectors of the nodes from which their incoming branches originate. A lower node is a node where a branch with a negative slope terminates. This type of nodes receive the second element of the vectors of the nodes from which their incoming branches originate. A value passing along a branch is multiplied by $e^{-j\frac{2\pi}{8}n}$, where n is an integer that appears near the arrow of the branch. No integer near an arrow implies that the branch simply passes the value to its connecting node.

The even half-wave symmetric component can be expressed as

$$\begin{aligned} &2(X(0)e^{j0\frac{2\pi}{8}n} + X(2)e^{j2\frac{2\pi}{8}n} + X(4)e^{j4\frac{2\pi}{8}n} + X(6)e^{j6\frac{2\pi}{8}n}) \\ &= 2(X(0)e^{j0\frac{2\pi}{4}n} + X(2)e^{j1\frac{2\pi}{4}n} + X(4)e^{j2\frac{2\pi}{4}n} + X(6)e^{j3\frac{2\pi}{4}n}),\ n = 0, 1, 2, 3 \end{aligned}$$

This is a waveform composed of $\frac{N}{2} = 4$ frequency components with frequency coefficients

$$2\{X(0), X(2), X(4), X(6)\}$$

The odd half-wave symmetric component is multiplied by the exponential $e^{-j\frac{2\pi}{8}n}$ to get

$$\begin{aligned}
&2(X(1)e^{j1\frac{2\pi}{8}n} + X(3)e^{j3\frac{2\pi}{8}n} + X(5)e^{j5\frac{2\pi}{8}n} + X(7)e^{j7\frac{2\pi}{8}n})e^{-j\frac{2\pi}{8}n} \\
&= 2(X(1)e^{j0\frac{2\pi}{8}n} + X(3)e^{j2\frac{2\pi}{8}n} + X(5)e^{j4\frac{2\pi}{8}n} + X(7)e^{j6\frac{2\pi}{8}n}) \\
&= 2(X(1)e^{j0\frac{2\pi}{4}n} + X(3)e^{j1\frac{2\pi}{4}n} + X(5)e^{j2\frac{2\pi}{4}n} + X(7)e^{j3\frac{2\pi}{4}n}),\ n = 0, 1, 2, 3
\end{aligned}$$

This is a waveform composed of $\frac{N}{2} = 4$ frequency components with frequency coefficients

$$2\{X(1), X(3), X(5), X(7)\}$$

Note that multiplication by $e^{-j\frac{2\pi}{8}n}$ (called twiddle factor) is the frequency shifting operation, which shifts the spectrum to the left by one sample interval. The coefficients of the spectral components $X(k)$ are not changed. The multiplication operation is indicated by the numbers (the value of n in $e^{-j\frac{2\pi}{8}n}$) close to the first set of arrows near the two bottom nodes in the signal-flow graph.

We have reduced the problem of decomposing a waveform composed of N frequency components into two problems, each of decomposing a waveform composed of $\frac{N}{2}$ frequency components. Now, we repeat the same process to these two waveforms. Decomposing the two waveforms into their even and odd half-wave symmetric components, we get

$$4\{X_0 e^{j0\frac{2\pi}{4}n} + X(4)e^{j2\frac{2\pi}{4}n}, X(2)e^{j1\frac{2\pi}{4}n} + X(6)e^{j3\frac{2\pi}{4}n}\},\ n = 0, 1 \tag{A.1}$$

$$4\{X(1)e^{j0\frac{2\pi}{4}n} + X(5)e^{j2\frac{2\pi}{4}n}, X(3)e^{j1\frac{2\pi}{4}n} + X(7)e^{j3\frac{2\pi}{4}n}\},\ n = 0, 1 \tag{A.2}$$

These vector arrays are stored in the nodes at the middle of the signal-flow graph of the algorithm shown in Fig. A.1. The nodes, except the first set, carry out an add–subtract operation, in addition to providing storage.

The even half-wave symmetric component of the waveform given by Eq. (A.1) can be expressed as

$$\begin{aligned}
&4(X_0 e^{j0\frac{2\pi}{4}n} + X(4)e^{j2\frac{2\pi}{4}n}) \\
&= 4(X_0 e^{j0\frac{2\pi}{2}n} + X(4)e^{j1\frac{2\pi}{2}n}),\ n = 0, 1
\end{aligned}$$

This is a waveform composed of $\frac{N}{4} = 2$ frequency components with frequency coefficients $4\{X(0), X(4)\}$. The coefficients $\boldsymbol{A}(0) = \{A_0(0), A_1(0)\} = 8\{X(0), X(4)\}$ are obtained by simply adding and subtracting the two sample values. These coefficients are stored in the top node at the end of the signal-flow graph shown in Fig. A.1.

The odd half-wave symmetric component of the waveform given by Eq. (A.1) is multiplied by the exponential $e^{-j\frac{2\pi}{8}(2n)} = e^{-j\frac{2\pi}{4}n}$ to get

$$\begin{aligned} &4(X(2)e^{j1\frac{2\pi}{4}n} + X(6)e^{j3\frac{2\pi}{4}n})e^{-j\frac{2\pi}{4}n} \\ &= 4(X(2)e^{j0\frac{2\pi}{4}n} + X(6)e^{j2\frac{2\pi}{4}n}) \\ &= 4(X(2)e^{j0\frac{2\pi}{2}n} + X(6)e^{j1\frac{2\pi}{2}n}),\ n = 0, 1 \end{aligned}$$

This is a waveform composed of $\frac{N}{4} = 2$ frequency components with frequency coefficients $4\{X(2), X(6)\}$. The coefficients $\boldsymbol{A}(2) = \{A_0(2), A_1(2)\} = 8\{X(2), X(6)\}$ are obtained by simply adding and subtracting the two sample values. These coefficients are stored in the second node from top at the end of the signal-flow graph shown in Fig. A.1.

The even half-wave symmetric component of the waveform defined by Eq. (A.2) can be expressed as

$$\begin{aligned} &4(X(1)e^{j0\frac{2\pi}{4}n} + X(5)e^{j2\frac{2\pi}{4}n}) \\ &= 4(X(1)e^{j0\frac{2\pi}{2}n} + X(5)e^{j1\frac{2\pi}{2}n}),\ n = 0, 1 \end{aligned}$$

This is a waveform composed of $\frac{N}{4} = 2$ frequency components with frequency coefficients $4\{X(1), X(5)\}$. The coefficients $\boldsymbol{A}(1) = \{A_0(1), A_1(1)\} = 8\{X(1), X(5)\}$ are obtained by simply adding and subtracting the two sample values. These coefficients are stored in the third node from top at the end of the signal-flow graph shown in Fig. A.1.

The odd half-wave symmetric component of the waveform defined by Eq. (A.2) is multiplied by the exponential $e^{-j\frac{2\pi}{4}n}$ to get

$$\begin{aligned} &4(X(3)e^{j1\frac{2\pi}{4}n} + X(7)e^{j3\frac{2\pi}{4}n})e^{-j\frac{2\pi}{4}(n)} \\ &= 4(X(3)e^{j0\frac{2\pi}{4}n} + X(7)e^{j2\frac{2\pi}{4}n}) \\ &= 4(X(3)e^{j0\frac{2\pi}{2}n} + X(7)e^{j1\frac{2\pi}{2}n}),\ n = 0, 1 \end{aligned}$$

This is a waveform composed of $\frac{N}{4} = 2$ frequency components with frequency coefficients $4\{X(3), X(7)\}$. The coefficients $\boldsymbol{A}(3) = \{A_0(3), A_1(3)\} = 8\{X(3), X(7)\}$ are obtained by simply adding and subtracting the two sample values. These coefficients are stored in the fourth node from top at the end of the signal-flow graph shown in Fig. A.1.

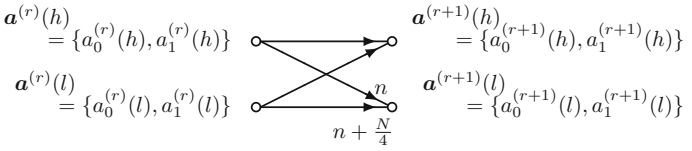


Fig. A.2 The signal-flow graph of the butterfly of the PM DIF DFT algorithm, where $0 \leq n < \frac{N}{4}$. A twiddle factor $W_N^n = e^{-j\frac{2\pi}{N}n}$ is indicated only by its variable part of the exponent, n

The output vectors $\{\boldsymbol{A}(0), \boldsymbol{A}(1), \boldsymbol{A}(2), \boldsymbol{A}(3)\}$ appear in bit-reversed order. The binary number representation of the frequency indices $\{0, 1, 2, 3\}$ is $\{00, 01, 10, 11\}$. By reversing the order of bits, we get the bit-reversed order $\{00, 10, 01, 11\}$ in binary form and $\{0, 2, 1, 3\}$ in decimal form. The bit-reversed order occurs at the output because of the repeated splitting of the frequency components into groups consisting of odd- and even-indexed frequency indices over the stages of the algorithm.

Each stage of the algorithm requires N complex additions and $N/2$ complex multiplications, where N is the sequence length. There are $(\log_2 N) - 1$ stages. In addition, the initial vector formation requires N complex additions. Therefore, the algorithm reduces the computational complexity to $N \log_2 N$ compared from that of N^2 required for the direct computation of the N-point DFT.

The algorithm is so regular that one can easily get the signal-flow graph for any value of N that is an integral power of two. The signal-flow graph algorithm is basically an interconnection of butterflies (a computational structure), shown in Fig. A.2. The defining equations of a butterfly at the rth stage are given by

$$\begin{aligned}
a_0^{(r+1)}(h) &= a_0^{(r)}(h) + a_0^{(r)}(l) \\
a_1^{(r+1)}(h) &= a_0^{(r)}(h) - a_0^{(r)}(l) \\
a_0^{(r+1)}(l) &= W_N^n a_1^{(r)}(h) + W_N^{n+\frac{N}{4}} a_1^{(r)}(l) \\
a_1^{(r+1)}(l) &= W_N^n a_1^{(r)}(h) - W_N^{n+\frac{N}{4}} a_1^{(r)}(l),
\end{aligned}$$

where $W_N^n = e^{-j\frac{2\pi}{N}n}$. There are $(\log_2 N) - 1$ stages, each with $N/4$ butterflies. With $N = 8$, therefore, we see four butterflies in Fig. A.1.

The extraction of the coefficient $e^{j\frac{\pi}{3}} = \frac{1}{2} + j\frac{\sqrt{3}}{2}$, multiplied by 8, $(4 + j4\sqrt{3})$, of the waveform $x(n) = e^{j(2\frac{2\pi}{8}n + \frac{\pi}{3})}$, is shown in Fig. A.3.

The extraction of the coefficient $3e^{j\frac{\pi}{6}}$, multiplied by 8, of $x(n) = 3e^{j(7\frac{2\pi}{8}n + \frac{\pi}{6})}$, is shown in Fig. A.4.

Input values	Vector formation	Stage 1 output	Stage 2 output
$x(0)=\frac{1}{2}+j\frac{\sqrt{3}}{2}$ $x(4)=\frac{1}{2}+j\frac{\sqrt{3}}{2}$	$a_0(0)=1+j\sqrt{3}$ $a_1(0)=0$	0 $2+j2\sqrt{3}$	$X(0)=A_0(0)=0$ $X(4)=A_1(0)=0$
$x(1)=-\frac{\sqrt{3}}{2}+j\frac{1}{2}$ $x(5)=-\frac{\sqrt{3}}{2}+j\frac{1}{2}$	$a_0(1)=-\sqrt{3}+j1$ $a_1(1)=0$	0 $-2\sqrt{3}+j2$	$X(2)=A_0(2)=4+j4\sqrt{3}$ $X(6)=A_1(2)=0$
$x(2)=-\frac{1}{2}-j\frac{\sqrt{3}}{2}$ $x(6)=-\frac{1}{2}-j\frac{\sqrt{3}}{2}$	$a_0(2)=-1-j\sqrt{3}$ $a_1(2)=0$	0 0	$X(1)=A_0(1)=0$ $X(5)=A_1(1)=0$
$x(3)=\frac{\sqrt{3}}{2}-j\frac{1}{2}$ $x(7)=\frac{\sqrt{3}}{2}-j\frac{1}{2}$	$a_0(3)=\sqrt{3}-j1$ $a_1(3)=0$	0 0	$X(3)=A_0(3)=0$ $X(7)=A_1(3)=0$

Fig. A.3 The trace of the PM DIF DFT algorithm, with $N=8$, in extracting the coefficient $e^{j\frac{\pi}{3}}$, multiplied by 8, of $x(n)=e^{j(2\frac{2\pi}{8}n+\frac{\pi}{3})}$

Input values	Vector formation	Stage 1 output	Stage 2 output
$x(0)=\frac{3\sqrt{3}}{2}+j\frac{3}{2}$ $x(4)=-\frac{3\sqrt{3}}{2}-j\frac{3}{2}$	$a_0(0)=0$ $a_1(0)=3\sqrt{3}+j3$	0 0	$X(0)=A_0(0)=0$ $X(4)=A_1(0)=0$
$x(1)=\frac{3(\sqrt{6}+\sqrt{2})}{4}$ $\quad -j\frac{3(\sqrt{6}-\sqrt{2})}{4}$ $x(5)=-\frac{3(\sqrt{6}+\sqrt{2})}{4}$ $\quad +j\frac{3(\sqrt{6}-\sqrt{2})}{4}$	$a_0(1)=0$ $a_1(1)=\frac{3(\sqrt{6}+\sqrt{2})}{2}$ $\quad -j\frac{3(\sqrt{6}-\sqrt{2})}{2}$	0 0	$X(2)=A_0(2)=0$ $X(6)=A_1(2)=0$
$x(2)=\frac{3}{2}-j\frac{3\sqrt{3}}{2}$ $x(6)=-\frac{3}{2}+j\frac{3\sqrt{3}}{2}$	$a_0(2)=0$ $a_1(2)=3-j3\sqrt{3}$	0 $6\sqrt{3}+j6$	$X(1)=A_0(1)=0$ $X(5)=A_1(1)=0$
$x(3)=-\frac{3(\sqrt{6}-\sqrt{2})}{4}$ $\quad -j\frac{3(\sqrt{6}+\sqrt{2})}{4}$ $x(7)=\frac{3(\sqrt{6}-\sqrt{2})}{4}$ $\quad +j\frac{3(\sqrt{6}+\sqrt{2})}{4}$	$a_0(3)=0$ $a_1(3)=-\frac{3(\sqrt{6}-\sqrt{2})}{2}$ $-j\frac{3(\sqrt{6}+\sqrt{2})}{2}$	0 $6-j6\sqrt{3}$	$X(3)=A_0(3)=0$ $X(7)=A_1(3)=12\sqrt{3}+j12$

Fig. A.4 The trace of the PM DIF DFT algorithm, with $N=8$, in extracting the coefficient $3e^{j\frac{\pi}{6}}$, multiplied by 8, of $x(n)=3e^{j(7\frac{2\pi}{8}n+\frac{\pi}{6})}$

A.5 The PM DIT DFT Algorithm

We have given the physical explanation of the decomposition of waveforms in the DIF DFT algorithm. In a decimation-in-frequency (DIF) algorithm, the transform sequence, $X(k)$, is successively divided into smaller subsequences. For example, in the beginning of the first stage, the computation of an N-point DFT is decomposed into two problems: (i) computing the $(N/2)$ even-indexed $X(k)$ and (ii) computing the $(N/2)$ odd-indexed $X(k)$. In a decimation-in-time (DIT) algorithm, the data sequence, $x(n)$, is successively divided into smaller subsequences. For example, in the beginning of the last stage, the computation of an N-point DFT is decomposed into two problems: (i) computing the $(N/2)$-point DFT of even-indexed $x(n)$ and (ii) computing the $(N/2)$-point DFT of odd-indexed $x(n)$. The DIT DFT algorithm is based on zero-padding, time-shifting, and spectral redundancy. For understanding, the DIF DFT algorithms are easier. However, the DIT algorithms are used more often, as taking care of the data scrambling problem occurring at the beginning of the algorithm is relatively easier. The DIT DFT algorithms can be considered as the algorithms obtained by transposing the signal-flow graph of the corresponding DIF algorithms, that is by reversing the direction of (signal flow) all the arrows and interchanging the input and the output. The computational complexity and the storage requirements can be reduced by a factor of 2 for computing the DFT of real data. It is the reduction of the computational complexity from N^2 to $N \log_2 N$ is the most important factor in the development of digital signal and image processing applications.

Bibliography

1. Gonzalez, R. C., & Woods, R. E. (2008). *Digital image processing*. NJ: Pearson Education Inc.
2. Gonzalez, R. C., Woods, R. E., & Eddins, S. L. (2010). *Digital image processing using Matlab*. USA: McGraw Hill.
3. Jain, A. K. (1989). *Fundamental of digital image processing*. NJ: Prentice-Hall Inc.
4. Pratt, W. K. (2013). *Introduction to digital image processing*. NY: CRC Press.
5. Sayood, K. (2006). *Introduction to data compression*. USA: Elsevier.
6. Sundararajan, D. (2001). *Discrete Fourier transform, theory, algorithms, and applications*. Singapore: World Scientific.
7. Sundararajan, D. (2008). *Signals and systems—A practical approach*. Singapore: Wiley.
8. Sundararajan, D. (2015). *Discrete wavelet transform, a signal processing approach*. Singapore: Wiley.
9. The Mathworks. (2017). *Matlab image processing tool box user's guide*. USA: The Mathworks, Inc.
10. The Mathworks. (2017). *Matlab signal processing tool box user's guide*. USA: The Mathworks, Inc.
11. The Mathworks. (2017). *Matlab wavelet tool box user's guide*. USA: The Mathworks, Inc.

D. Sundararajan, *Digital Image Processing*, DOI 10.1007/978-981-10-6113-4

Answers to Selected Exercises

Chapter 1

1.3(i).

$$\begin{bmatrix} 218 & 255 & 218 & 128 & 37 & 0 & 37 & 127 \\ 255 & 218 & 128 & 37 & 0 & 37 & 127 & 218 \\ 218 & 128 & 37 & 0 & 37 & 127 & 218 & 255 \\ 128 & 37 & 0 & 37 & 127 & 218 & 255 & 218 \\ 37 & 0 & 37 & 127 & 218 & 255 & 218 & 128 \\ 0 & 37 & 127 & 218 & 255 & 218 & 128 & 37 \\ 37 & 127 & 218 & 255 & 218 & 128 & 37 & 0 \\ 127 & 218 & 255 & 218 & 128 & 37 & 0 & 37 \end{bmatrix}$$

1.7(i). Aliasing

$$x(m, n) = \cos(\frac{2\pi}{32}4m + \frac{2\pi}{32}2n + \frac{\pi}{6})$$

Chapter 2

2.4.(i).

$$\begin{bmatrix} 14 & 14 & 3 & 3 \\ 14 & 14 & 14 & 3 \\ 15 & 14 & 14 & 14 \\ 14 & 14 & 14 & 14 \end{bmatrix}$$

2.5.(i).

$$\begin{bmatrix} 5 & 5 & 3 & 3 \\ 5 & 5 & 5 & 3 \\ 5 & 5 & 5 & 5 \\ 5 & 5 & 5 & 5 \end{bmatrix}$$

D. Sundararajan, *Digital Image Processing*, DOI 10.1007/978-981-10-6113-4

2.10.

$$y(m,n) = \begin{bmatrix} 202.7682 & 197.0167 & 193.2693 & 192.9102 \\ 214.7263 & 209.1150 & 202.5467 & 208.1208 \\ 221.8699 & 212.8023 & 215.4462 & 224.2604 \\ 222.0558 & 209.8174 & 224.0159 & 229.2589 \end{bmatrix}$$

2.16.

$$y(m,n) = \begin{bmatrix} 184 & 201 & 241 & 267 \\ 143 & 209 & 272 & 252 \\ 161 & 205 & 260 & 29 \\ 158 & 255 & 224 & 166 \end{bmatrix}$$

Chapter 3

3.1.(i). The samples are

$$\{4.5000, -3.5981, 1.5000, 1.5981\}$$

The DFT coefficients are

$$4\{1, 0.75 + j0.75\sqrt{3}, 2, 0.75 - j0.75\sqrt{3}\}$$

The least squares errors are 34, 34.0400, 34.0400.

3.3.(i). The image is

$$\begin{bmatrix} 3.0000 & 1.7321 & 1.0000 & -1.7321 \\ 1.7321 & 1.0000 & -1.7321 & 3.0000 \\ 1.0000 & -1.7321 & 3.0000 & 1.7321 \\ -1.7321 & 3.0000 & 1.7321 & 1.0000 \end{bmatrix}$$

The DFT coefficients are

$$\begin{bmatrix} 16 & 0 & 0 & 0 \\ 0 & 8 - j13.8564 & 0 & 0 \\ 0 & 0 & 16 & 0 \\ 0 & 0 & 0 & 8 + j13.8564 \end{bmatrix}$$

The least squares errors are 48, 48.16, 48.16.

3.4.(i).

$$= 1000 \begin{bmatrix} 1.6140 & 0.1970 - j0.1670 & -0.0200 & 0.1970 + j0.1670 \\ 0.1480 + j0.0440 & 0.0670 + j0.0630 & -0.1480 + j0.2140 & 0.0010 - j0.1210 \\ 0.0300 & -0.1050 + j0.0030 & -0.1520 & -0.1050 - j0.0030 \\ 0.1480 - j0.0440 & 0.0010 + j0.1210 & -0.1480 - j0.2140 & 0.0670 - j0.0630 \end{bmatrix}$$

The signal power is 188384. The magnitude in log scale and center-zero format is

$$\begin{bmatrix} 2.1847 & 2.0255 & 1.4914 & 2.0255 \\ 2.4170 & 1.9683 & 2.1915 & 2.0864 \\ 1.3222 & 2.4137 & 3.2082 & 2.4137 \\ 2.4170 & 2.0864 & 2.1915 & 1.9683 \end{bmatrix}$$

3.6.(i).

$$y(m,n) = \begin{bmatrix} 31 & 23 & 8 & 7 \\ 17 & 5 & 13 & 17 \\ 8 & 9 & 29 & 26 \\ 18 & 26 & 18 & 9 \end{bmatrix}$$

$$r_{hx}(m,n) = \begin{bmatrix} 31 & 14 & 11 & 19 \\ 5 & 8 & 22 & 18 \\ 4 & 20 & 30 & 12 \\ 28 & 21 & 11 & 10 \end{bmatrix} \qquad r_{xh}(m,n) = \begin{bmatrix} 31 & 19 & 11 & 14 \\ 28 & 10 & 11 & 21 \\ 4 & 12 & 30 & 20 \\ 5 & 18 & 22 & 8 \end{bmatrix}$$

$$r_{xx}(m,n) = \begin{bmatrix} 56 & 38 & 26 & 38 \\ 40 & 32 & 29 & 34 \\ 38 & 36 & 38 & 36 \\ 40 & 34 & 29 & 32 \end{bmatrix}$$

Chapter 4
4.1.(iii).

$$y(n) = \{-12, 7, -9, 2\}$$

4.2.(i).

$$y(m,n) = \begin{bmatrix} 2 & 3 & 4 & -3 \\ 5 & -4 & 12 & 5 \\ 4 & 4 & -5 & 1 \\ 1 & 4 & 5 & 2 \end{bmatrix}$$

4.3.(ii).

$$y(m,n) = \begin{bmatrix} 1.7958 & 1.3504 & 1.8327 & 2.0064 \\ 1.5006 & 1.0751 & 1.4203 & 1.8115 \\ 0.6812 & 0.3663 & 0.1055 & 0.4098 \\ 0.8445 & 0.6601 & 0.5600 & 0.5797 \end{bmatrix}$$

4.5.(i).

$$y(m,n) = \begin{bmatrix} 18 & 19 & 11 & -9 \\ 8 & -3 & 6 & -7 \\ -20 & 32 & 3 & 9 \\ 17 & -22 & 0 & -6 \end{bmatrix}$$

Chapter 5
5.2.

$$\hat{x}(n) = \{0.0082, 0.6993, 0.9808, 0.6877, -0.0082, -0.6993, -0.9808, -0.6877\}$$

Chapter 6
6.2.(ii).

$$\begin{bmatrix} 53.00 & 47.50 & 42.00 & 40.50 & 39.00 & 48.50 & 58.00 \\ 52.00 & 48.00 & 44.00 & 42.75 & 41.50 & 47.50 & 53.50 \\ 51.00 & 48.50 & 46.00 & 45.00 & 44.00 & 46.50 & 49.00 \\ 52.50 & 50.75 & 49.00 & 50.00 & 51.00 & 49.25 & 47.50 \\ 54.00 & 53.00 & 52.00 & 55.00 & 58.00 & 52.00 & 46.00 \\ 58.50 & 56.50 & 54.50 & 57.50 & 60.50 & 54.75 & 49.00 \\ 63.00 & 60.00 & 57.00 & 60.00 & 63.00 & 57.50 & 52.00 \end{bmatrix}$$

6.3.(ii).

$$\begin{bmatrix} 127 & 57 \\ 108 & 51 \end{bmatrix}$$

6.5.(iii).

$$\begin{bmatrix} 95 & 111 & 0 & 0 \\ 96 & 115 & 48 & 32 \\ 89 & 90 & 59 & 26 \\ 86 & 73 & 37 & 24 \\ 0 & 0 & 15 & 21 \end{bmatrix}$$

6.7.(i).

$$\begin{bmatrix} 2 & 5 & 7 & 9 & 7 & 2 \\ 3 & 10 & 15 & 10 & 12 & 6 \\ 3 & 10 & 23 & 19 & 13 & 4 \\ 4 & 11 & 18 & 15 & 10 & 8 \\ 1 & 4 & 10 & 14 & 5 & 8 \\ 1 & 4 & 4 & 5 & 2 & 2 \end{bmatrix} \quad \begin{bmatrix} -0.1000 & 0.2828 & -0.0447 & 0.2236 & 0.1180 & -0.1000 \\ -0.4128 & 0.0286 & 0.0408 & -0.1342 & 0.6261 & 0.8000 \\ -0.5814 & -0.4727 & 0.6200 & 0.3213 & 0.2683 & -0.1512 \\ -0.3124 & -0.4491 & 0.1315 & -0.1315 & 0.4619 & 0.5292 \\ -0.4914 & -0.6252 & 0.0316 & 0.2286 & -0.1890 & 0.5292 \\ -0.1000 & -0.1315 & -0.1315 & -0.1474 & -0.1000 & -0.1000 \end{bmatrix}$$

Chapter 7

7.1.(iii).

$$\frac{x}{2} - \frac{\sqrt{3}y}{2} = 2$$

7.2.(ii).

$$R(s,\theta) = 6\sqrt{6^2 - (s - \cos(\theta) - 2\sin(\theta))^2}$$

7.3.(iii).

$$s = \sqrt{32}\cos(-45^\circ - \theta)$$

7.6.(i).

$$acc(m,n) = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 2 & 2 & 0 & 2 \\ 1 & 1 & 4 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

The entry $acc(2,2) = 4$ in the acc matrix indicates a line at distance 2 from the top-left corner (origin) of the image at angle $90 + 90 = 180^\circ$ from the m-axis.

Chapter 8

8.5.(i).

$$x(m,n) = \begin{bmatrix} 0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&1 \\ 1&0&0&0&0&0&0&1 \\ 1&0&0&1&0&0&0&0 \\ 0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&1&1&1 \end{bmatrix}$$

8.6.(ii).

$$x(m,n) = \begin{bmatrix} 0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0 \\ 0&1&1&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0 \\ 0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&0&0 \end{bmatrix}$$

8.7.(iii).

$$x(m,n) = \begin{bmatrix} 1&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0\\ 0&0&1&0&0&0&0&0\\ 0&0&0&1&0&0&0&0\\ 0&0&0&0&1&0&0&0\\ 0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&1 \end{bmatrix}$$

8.8.(ii).

$$x(m,n) = \begin{bmatrix} 0&0&0&0&0&0&0&0\\ 1&0&0&0&0&0&0&0\\ 1&1&0&0&0&0&0&0\\ 1&1&1&0&0&0&0&0\\ 1&0&1&1&0&0&0&0\\ 1&0&0&1&1&0&0&0\\ 1&0&0&0&1&1&0&0\\ 1&1&1&1&1&1&1&0 \end{bmatrix}$$

Chapter 9
9.1.(i).

$$\begin{bmatrix} 49.5025 & 110.4451 & 78.8531 & 6.3738 & 1.2748 & 1.7766\\ 9.1992 & 87.9275 & 104.8967 & 22.1923 & 3.6443 & 1.1180\\ 0.3536 & 65.3292 & 111.1987 & 42.2334 & 4.8894 & 0.5590\\ 0 & 43.4403 & 110.5075 & 66.0350 & 2.4044 & 2.3049\\ 0 & 22.5534 & 98.9326 & 92.8995 & 12.9735 & 5.3180\\ 0 & 9.5197 & 68.6740 & 106.8324 & 54.7280 & 5.4458 \end{bmatrix}$$

$$e(m,n) = \begin{bmatrix} 1&1&1&0&0&0\\ 0&1&1&0&0&0\\ 0&1&1&0&0&0\\ 0&0&1&1&0&0\\ 0&0&1&1&0&0\\ 0&0&1&1&1&0 \end{bmatrix}$$

9.2.(ii).

$$\begin{bmatrix} 5.0105 & -0.2727 & -5.3430 & -5.7242 & -10.2161 & -28.1357 & -33.8425 & -15.9979 \\ -8.9723 & -13.5013 & -16.3388 & -18.7095 & -28.2793 & -38.4562 & -18.6954 & 20.0468 \\ 0.5121 & -9.8865 & -17.8006 & -24.2760 & -27.9287 & -12.6175 & 21.2665 & 43.7632 \\ 24.1600 & 9.4774 & -1.3458 & -0.2453 & 18.7616 & 38.9715 & 39.5796 & 24.7139 \\ 14.9449 & 10.6197 & 4.6543 & 8.7492 & 40.4500 & 53.8614 & 35.7797 & 13.2239 \\ -2.4414 & 2.3624 & -2.5966 & -13.3771 & 3.0772 & 20.0810 & 23.4170 & 23.2587 \\ -3.6860 & -3.4955 & -11.0425 & -25.8184 & -23.2475 & -10.5215 & -2.0928 & 14.6576 \\ 2.5690 & -5.4928 & -15.0516 & -24.8636 & -19.8852 & -8.3275 & -14.2503 & -9.5180 \end{bmatrix}$$

$$e(m,n) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Chapter 10

10.3. 85, 94.5714.

10.6.

Gray level, k	0	1	2	3	4	5	6	7
$hn(k)$	0	0	0	0	0.0625	0.6250	0.2500	0.0625
$hc1(k)$	0	0	0	0	0.0625	0.6875	0.9375	1
$ha(k)$	0	0	0	0	0.25	3.3750	4.8750	5.3125
$\sigma_b^2(k)$	0	0	0	0	0.1148	0.3580	0.1898	0

The threshold and the separability index are 5 and 0.7702.

10.8.

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}$$

10.10.

$$\begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

10.14.

2, 2, 2, 3

2.5000, 2.0000, 4.4286, 5.2857

2.1818, 3.0000, 5.6429, 6.1429

2.4000, 3.4000, 6.70006.8000

2.5000, 3.5000, 7.0000, 7.0000

10.18.

$$\begin{bmatrix} 3.0 & 2.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 3.2 & 2.2 & 1.4 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 3.6 & 2.8 & 2.2 & 1.4 & 1.0 & 0.0 & 0.0 & 0.0 \\ 4.2 & 3.6 & 2.8 & 2.2 & 1.4 & 1.0 & 0.0 & 0.0 \\ 5.0 & 4.2 & 3.6 & 2.8 & 2.0 & 1.0 & 0.0 & 0.0 \\ 5.6 & 5.0 & 4.0 & 3.0 & 2.0 & 1.0 & 0.0 & 0.0 \\ 6.2 & 5.2 & 4.2 & 3.2 & 2.2 & 1.4 & 1.0 & 0.0 \\ 6.4 & 5.4 & 4.4 & 3.6 & 2.8 & 2.0 & 1.0 & 0.0 \end{bmatrix}$$

Chapter 11

11.1.(i).

{7, 7, 7, 7, 7, 4, 4, 4, 4, 4, 2, 2, 2, 2, 2}

11.2.(ii). Centroid {1.5, 1.5}

θ	-135	-162	162	135	108	72	45	18	-18	-45	-72	-108
$d(\theta)$	2.121	1.581	1.581	2.121	1.581	1.581	2.121	1.581	1.581	2.121	1.581	1.581

11.3.(iii).

$$\{2 + j0, 1 + j1, 0 + j2, 1 + j3, 2 + j3, 3 + j3, 3 + j2, 3 + j, 3 + j0\}$$

Fourier descriptor

$$\{18 + j15, 1.3803 - j0.5371, 2.1665 + j0.0717, 0 + j0, -0.0271 - j0.5967,$$

$$-1.1577 + j0.3974, 0 + j0, 1.2449 - j1.1700, -3.6070 - j13.1652\}$$

11.8.

$$\{m_{00} = 7, \quad m_{10} = 7, \quad m_{01} = 12, \quad \bar{k} = 1, \quad \bar{l} = 1.7143\}$$

$$\mu_{11} = 0, \quad \mu_{20} = 4, \quad \mu_{02} = 3.4286$$

$$\eta_{11} = 0, \eta_{20} = 0.0816, \eta_{02} = 0.07$$

$$\phi_1 = 0.1516, \quad \phi_2 = 0.0001$$

11.9.

$$mean = 85.1250, std = 27.7689, Sk = -0.5252, S = 0.0117, U = 0.0859, E = 3.6250$$

11.12.

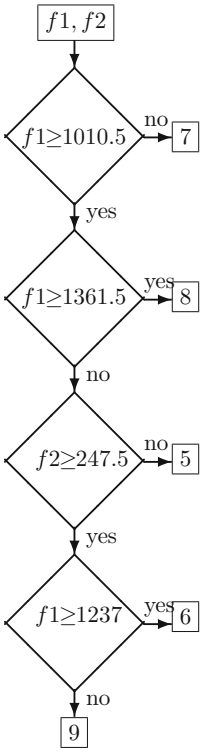
$$\begin{bmatrix} 4 & 4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 7 & 17 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 1 & 4 & 0 & 6 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0.0714 & 0.0714 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.1250 & 0.3036 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.0179 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.0536 & 0 & 0 & 0 & 0 & 0 & 0.0357 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.0536 & 0 & 0.0179 & 0.0714 & 0 & 0.1071 & 0.0179 \\ 0 & 0.0179 & 0 & 0 & 0.0179 & 0 & 0.0179 & 0 \end{bmatrix}$$

$$Contrast : 3.8929, Correlation : 0.7238, Energy : 0.1435, Homogeneity : 0.6710$$

11.16.

$$\begin{bmatrix} -0.0126 & 0.9013 \\ 1.6669 & -0.1846 \\ 0.2841 & -0.4814 \\ -1.9383 & -0.2352 \end{bmatrix}$$



12.8 Flowchart of the decision tree algorithm

Chapter 12
12.1. The sorted distances are

$$\{1, 1.4142, 1.4142, 2.0000, 2.2361, 2.8284, 2.8284, 3.1623, 3.1623\}$$

Test vector belongs to class 2.
12.5.

$$d_1(x) = 22x_1 - 20x_2 - 442$$

$$d_2(x) = 20x_1 + 20x_2 - 400$$

$$d_3(x) = -18x_1 - 19x_2 - 342.5$$

For the first feature vector, these three functions yield

$$\{464, -340, -458\}$$

For the second feature vector, these three functions yield

$$\{-340, 380, -1140\}$$

and, for the third feature vector, these three functions yield

$$1000\{-0.3765, -1.0625, 0.3425\}$$

12.8. Figure
12.12. With $p(\omega_1) = 0.4$ and $p(\omega_2) = 0.6$, the decision function for class1 $d_1(x)$ is

$$-1.4270x_1^2 - 3.8686x_2^2 + 2.6807(x_1 + x_2) + 6.2149x_1 - 9.0933x_2 - 7.3479$$

For the test data, the decision function yields

$$\{-1.8305, -2.0560, -11.5379, -26.6557, -39.7503, -21.0165\}$$

The decision function for class2 $d_2(x)$ is

$$-0.5588x_1^2 - 0.6123x_2^2 + 0.6542(x_1 + x_2) - 1.2152x_1 + 1.2711x_2 - 1.3883$$

For the test data, the decision function yields

$$\{-9.0630, -5.2618, -2.3027, -1.2355, -2.3397, -0.5830\}$$

Chapter 13
13.1.(i).

```
[ 99]    '1  0  1  1'
[103]    '1  0  0'
[107]    '1  0  1  0'
[108]    '0  1  0  1'
[109]    '0  1  0  0'
[110]    '0  1  1  1'
[113]    '0  1  1  0'
[116]    '0  0  0  0  1'
[120]    '0  0  0  0  0'
[121]    '1  1'
[129]    '0  0  0  1'
[144]    '0  0  1'
```

$$bpp = 3.5, \qquad entropy = 3.4528$$

13.2.(ii).

$$\begin{bmatrix} 11 & 9 & 15 & 14 \\ 13 & 7 & 6 & 7 \\ 6 & 5 & 5 & 5 \\ 11 & 4 & 4 & 2 \end{bmatrix} = 2^3 \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} + 2^2 \begin{bmatrix} 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} + 2 \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

{04, 013, 4, 013 22, 04, 04, 121 0112, 13, 013, 0121 031, 0211, 13, 013}

{14, 11, 10, 11 32, 14, 14, 22 1132, 23, 11, 1141 13, 1241, 23, 11}

13.3.(iii).

$$\begin{bmatrix} 106 & -3 & -5 & 1 \\ 121 & 1 & -14 & -15 \\ 102 & 0 & -2 & -1 \\ 100 & 1 & 1 & -6 \end{bmatrix}$$

{3.4528, 3.5}

13.5.(i).

{−2.3039, 0.2557, 3.8341, 2.4144, 1.6731, 4.0880, −3.5067, 2.2929,

3.6913, −1.0733, 1.6485, 1.8253, 0.6833, −1.5910, −3.6983, −1.5573}

13.6.(ii).

```
[57]    '0  1  0  1'
[59]    '0  1  0  0'
[63]    '0  1  1  1'
[56]    '0  1  1  0'
[ 1]    '0  0  1'
[-1]    '0  0  0'
[ 0]    '1'
```

$$bpp = 2.375, SNR = 46.9179$$

Chapter 14

14.2.(i).

$$\begin{bmatrix} 0.8952 & 0.8893 & 0.8881 & 0.8782 \\ 0.8942 & 0.8864 & 0.8837 & 0.8738 \\ 0.8923 & 0.8841 & 0.8776 & 0.8680 \\ 0.8907 & 0.8817 & 0.8737 & 0.8650 \end{bmatrix} \begin{bmatrix} 0.6682 & 0.6807 & 0.6330 & 0.5688 \\ 0.6556 & 0.6452 & 0.6180 & 0.5385 \\ 0.6528 & 0.6164 & 0.6187 & 0.5135 \\ 0.6323 & 0.6041 & 0.5810 & 0.4831 \end{bmatrix}$$

$$\begin{bmatrix} 0.5791 & 0.5895 & 0.6092 & 0.6458 \\ 0.5922 & 0.6078 & 0.6261 & 0.6627 \\ 0.5987 & 0.6235 & 0.6275 & 0.6771 \\ 0.6078 & 0.6340 & 0.6458 & 0.6980 \end{bmatrix}$$

14.3.(ii).

$$\begin{bmatrix} 0.3415 & 0.3348 & 0.3867 & 0.4260 \\ 0.3288 & 0.3381 & 0.3966 & 0.4183 \\ 0.3362 & 0.3476 & 0.3994 & 0.3961 \\ 0.3515 & 0.3579 & 0.3714 & 0.3519 \end{bmatrix} \begin{bmatrix} 0.3897 & 0.3926 & 0.4192 & 0.4422 \\ 0.3922 & 0.3999 & 0.4242 & 0.4474 \\ 0.3935 & 0.4005 & 0.4213 & 0.4373 \\ 0.3947 & 0.3928 & 0.4054 & 0.4169 \end{bmatrix}$$

$$\begin{bmatrix} 0.1700 & 0.1827 & 0.1971 & 0.2120 \\ 0.1676 & 0.1786 & 0.1923 & 0.2038 \\ 0.1663 & 0.1688 & 0.1796 & 0.1919 \\ 0.1651 & 0.1577 & 0.1672 & 0.1746 \end{bmatrix}$$

14.4.(i).

$$\begin{bmatrix} 64 & 64 & 71 & 71 \\ 66 & 66 & 62 & 81 \\ 67 & 67 & 67 & 73 \\ 68 & 71 & 70 & 60 \end{bmatrix} \begin{bmatrix} 122 & 122 & 120 & 120 \\ 122 & 122 & 120 & 120 \\ 122 & 122 & 120 & 120 \\ 122 & 121 & 118 & 116 \end{bmatrix} \begin{bmatrix} 182 & 182 & 182 & 181 \\ 182 & 182 & 184 & 183 \\ 182 & 181 & 186 & 184 \\ 183 & 184 & 186 & 175 \end{bmatrix}$$

14.5.(ii).

$$color\_map1 = \begin{bmatrix} 1\ 1\ 0 \\ 1\ 0\ 0 \\ 1\ 0\ 1 \\ 0\ 1\ 1 \\ 0\ 1\ 0 \end{bmatrix}$$

14.7.(i). The intensity component and its equalized version are

$$\begin{bmatrix} 76 & 74 & 75 & 84 \\ 73 & 68 & 78 & 95 \\ 70 & 74 & 88 & 88 \\ 69 & 79 & 97 & 87 \end{bmatrix} \begin{bmatrix} 128 & 96 & 112 & 175 \\ 64 & 16 & 143 & 239 \\ 48 & 96 & 223 & 223 \\ 32 & 159 & 255 & 191 \end{bmatrix}$$

Index

D. Sundararajan, *Digital Image Processing*, DOI 10.1007/978-981-10-6113-4