Digital Image Processing

Asst. Prof. Dr. Hammam Alshazly

Faculty of Computers and Artificial Intelligence,
South Valley National University

This book presents in-depth the fundamental methods and applications of digital image processing.

November 2024

Table of Contents

CHAPTER 1
INTRODUCTION

1.1 Overview

In our increasingly visual world, digital images have become ubiquitous - from the photos we capture on our smartphones to the high-resolution satellite imagery used for scientific research and commercial applications. As our ability to generate, collect, and store digital images has grown exponentially, the need to understand how to process, analyze, and extract meaningful information from these visual data sources has become more important than ever.

Digital image processing is an interdisciplinary field that sits at the intersection of computer science, electrical engineering, and applied mathematics. It involves the development and application of algorithms and techniques to perform a wide range of operations on digital images, such as enhancement, filtering, restoration, segmentation, and compression.

The applications of digital image processing are vast and diverse, spanning fields as varied as medical imaging, remote sensing, surveillance, autonomous vehicles, and artistic image manipulation. By applying sophisticated algorithms to digital image data, we can unlock powerful capabilities, such as the ability to automate the detection of tumors in medical scans, monitor crop health from satellite imagery, or enable self-driving cars to navigate safely.

In this book, we will take a deep dive into the fundamental principles and techniques of digital image processing. We start by exploring the basics of digital image representation and the various file formats, and coordinate systems used to store and work with visual data. From there, we move on to cover core image processing operations, such as filtering, enhancement, image compression, and image segmentation.

So let's get started on our journey into the world of digital image processing!

1.2 What Is Digital Image Processing?

An image may be defined as a two-dimensional function, $f(x,y)$, where x and y are spatial (plane) coordinates, and the amplitude of f at any pair of coordinates (x,y) is called the *intensity* or *gray level* of the image at that point. When x,y, and the intensity values of f are all finite, discrete quantities, we call the image a *digital image*. The field of digital image processing refers to processing digital images by means of a digital computer. Note that a digital image is composed of a finite number of elements, each of which has a particular location and value. These elements are called *picture elements*, *image elements, pels, and pixels*. Pixel is the term used most widely to denote the elements of a digital image.

Types of Images

1. **Binary Image**: The binary image, as its name suggests, contains only two pixel elements, i.e., 0 and 1, where 0 refers to black and 1 refers to white. This image is also known as Monochrome.
2. **Black and White Image**: The image which consists of only black and white color is called a BLACK AND WHITE IMAGE.
3. **8 bit COLOR FORMAT**: It is the most famous image format. It has 256 different shades of colors in it and is commonly known as

Grayscale Image. In this format, 0 stands for Black, 255 stands for White, and 127 stands for Gray.

4. **16 bit COLOR FORMAT**: It is a color image format. It has 65,536 different colors in it. It is also known as High Color Format. In this format, the distribution of color is not the same as that of Grayscale images.

A 16 bit format is actually divided into three further formats: Red, Green, and Blue. That famous RGB format.

The Image as a Matrix

As we know, images are represented in rows and columns. Mathematically, an image can be represented as a function $f(x,y)$, where (x,y) are the pixel coordinates. The image is of size $M \times N$, and can be expressed as a matrix:

$$f(x,y) = \begin{bmatrix} f(0,0) & f(0,1) & \cdots & f(0,N-1) \\ f(1,0) & f(1,1) & \cdots & f(1,N-1) \\ \vdots & \vdots & \ddots & \vdots \\ f(M-1,0) & f(M-1,1) & \cdots & f(M-1,N-1) \end{bmatrix} \tag{1.1}$$

where $\mathbf{f}(\mathbf{x},\mathbf{y})$ is the image matrix with M rows and N columns. Every element of this matrix is called image element, picture element, or pixel.

1.3 Steps in Digital Image Processing

It is helpful to divide the material covered in the following chapters into the two broad categories: methods whose input and output are images, and methods whose inputs may be images but whose outputs are attributes extracted from those images. This organization is summarized in Fig. 1.1. The diagram does not imply that every process is applied to an image. Rather, the intention is to convey an idea of all the methodologies that can be applied to images for different purposes and possibly with different objectives. The discussion in this section may be viewed as a brief overview of the material in the remainder of the book.

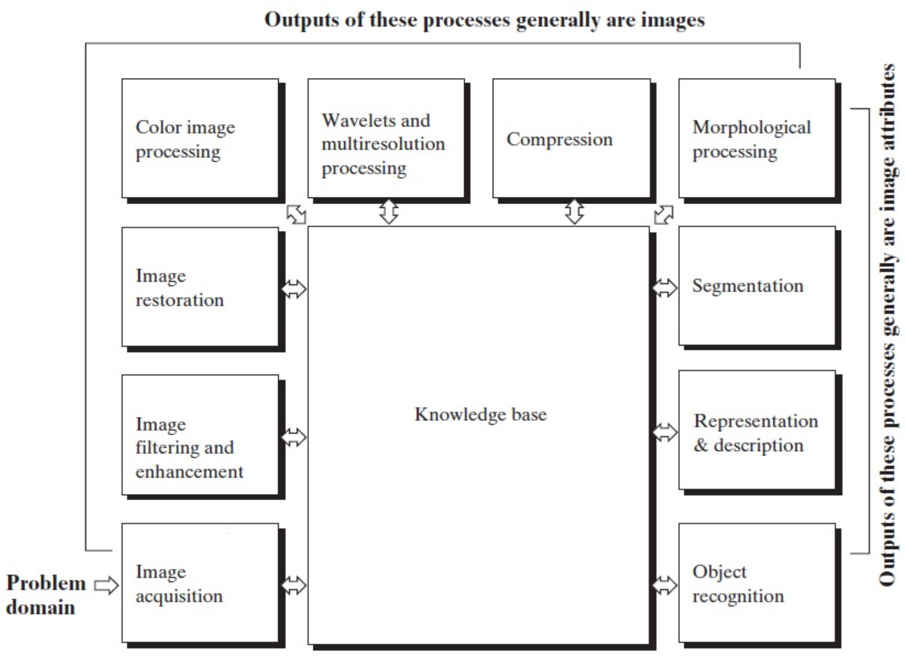

Outputs of these processes generally are images
Color image processing
Wavelets and multiresolution processing
Compression
Morphological processing
Image restoration
Segmentation
Knowledge base
Image filtering and enhancement
Representation & description
Problem domain
Image acquisition
Object recognition
Outputs of these processes generally are image attributes

Figure 1.1: Fundamental steps in digital image processing.

Image acquisition is the first process in Fig. 1.1. Note that acquisition could be as simple as being given an image that is already in digital form. Generally, the image acquisition stage involves preprocessing, such as scaling.

Image enhancement is the process of manipulating an image so that the result is more suitable than the original for a specific application. The word specific is important here, because it establishes at the outset that enhancement techniques are problem oriented. Thus, for example, a method that is quite useful for enhancing X-ray images may not be the best approach for enhancing satellite images taken in the infrared band of the electromagnetic spectrum.

There is no general "theory" of image enhancement. When an image is processed for visual interpretation, the viewer is the ultimate judge of how well a particular method works. Enhancement techniques are so varied, and use so many different image processing approaches, that it is difficult to assemble a meaningful body of techniques suitable for enhancement in one chapter without extensive background development. For this reason, and also because beginners in the field of image processing generally find enhancement applications visually appealing, interesting, and relatively simple to understand, we use image enhancement as examples when introducing new concepts in the following chapters. The material in the next two chapters span many of the methods used traditionally for image enhancement.

Image restoration is an area that also deals with improving the appearance of an image. It is the process of recovering an image from a degraded version—usually a blurred and noisy image. It is a fundamental problem in image processing, and it also provides a testbed for more general inverse problems. However, unlike enhancement, which is subjective, image restoration is objective, in the sense that restoration techniques tend to be based on mathematical or probabilistic models of image degradation. Enhancement, on the other hand, is based on human subjective preferences regarding what constitutes a "good" enhancement result.

Color image processing is an area that has been gaining in importance because of the significant increase in the use of digital images over the Internet. The topics cover a number of fundamental concepts in color models and basic color processing in a digital domain. Color is used as the basis for extracting features of interest in an image.

Wavelets are the foundation for representing images in various degrees of resolution. In particular, this material is used in for image data compression and for pyramidal representation, in which images are subdivided successively into smaller regions. Wavelets based transform are mathematical tools which are used to extract information from images. A significant benefit it has over

Fourier transforms is temporal resolution which signifies that it can captures both frequency and location information of the images

Compression, as the name implies, deals with techniques for reducing the storage required to save an image, or the bandwidth required to transmit it. Although storage technology has improved significantly over the past decade, the same cannot be said for transmission capacity.This is true particularly in uses of the Internet, which are characterized by significant pictorial content. Image compression is familiar (perhaps inadvertently) to most users of computers in the form of image file extensions, such as the jpg file extension used in the JPEG (Joint Photographic Experts Group) image compression standard.

Morphological processing deals with tools for extracting image components that are useful in the representation and description of shape. The material in this chapter begins a transition from processes that output images to processes that output image attributes.

Segmentation procedures partition an image into its constituent parts or objects. In general, autonomous segmentation is one of the most difficult tasks in digital image processing. A rugged segmentation procedure brings the process a long way toward successful solution of imaging problems that require objects to be identified individually. On the other hand, weak or erratic segmentation algorithms almost always guarantee eventual failure. In general, the more accurate the segmentation, the more likely recognition is to succeed.

Representation and description almost always follow the output of a segmentation stage, which usually is raw pixel data, constituting either the boundary of a region (i.e., the set of pixels separating one image region from another) or all the points in the region itself. In either case, converting the data to a form suitable for computer processing is necessary. The first decision that must be made is whether the data should be represented as a boundary or as a complete region. Boundary representation is appropriate when the focus is on external shape characteristics, such as corners and inflections. Regional representation is appropriate when the focus is on internal properties, such as texture or skeletal shape. In some applications, these representations complement each other. Choosing a representation is only part of the solution for transforming raw data into a form suitable for subsequent computer processing.A method must also be specified for describing the data so that features of interest are highlighted. *Description*, also called *feature selection*, deals with extracting attributes that result in some quantitative information of interest or are basic for differentiating one class of objects from another.

Recognition is the process that assigns a label (e.g., “vehicle”) to an object based on its descriptors. Image recognition is the process of identifying an object or a feature in an image or video. It is used in many applications like defect detection, medical imaging, and security surveillance. Image recognition is a crucial technique in many applications, and is the main driver in deep learning applications like: Visual Inspection, Image Classification, Automated Driving, and Robotics(e.g., Image recognition can be used by robots to identify objects and enhance autonomous navigation by identifying locations or objects on their path).

So far we have said nothing about the need for prior knowledge or about the interaction between the knowledge base and the processing modules in Fig. 1.1. Knowledge about a problem domain is coded into an image processing system in the form of a knowledge database. This knowledge may be as simple as detailing regions of an image where the information of interest is known to be located, thus limiting the search that has to be conducted in seeking that information. The knowledge base also can be quite complex, such as an interrelated list of all major possible defects in a materials inspection problem or an image database containing high-resolution satellite images of a region in connection with change-detection applications. In addition to guiding the operation of each processing module, the knowledge base also controls the interaction between modules. This distinction is made in Fig. 1.1 by the use of double-headed arrows between the processing modules and the knowledge base, as opposed to single-headed arrows linking the processing modules.

1.4 Advantages and Disadvantages

The following points highlight the balance between the capabilities and challenges associated with digital image processing.

Advantages of Digital Image Processing:

1. **Improved image quality:** Digital image processing algorithms can improve the visual quality of images, making them clearer, sharper, and more informative.
2. **Automated image-based tasks:** Digital image processing can automate many image-based tasks, such as object recognition, pattern detection, and measurement.
3. **Increased efficiency:** Digital image processing algorithms can process images much faster than humans, making it possible to analyze large amounts of data in a short amount of time.
4. **Increased accuracy:** Digital image processing algorithms can provide more accurate results than humans, especially for tasks that require precise measurements or quantitative analysis.

Disadvantages of Digital Image Processing:

1. **High computational cost:** Some digital image processing algorithms are computationally intensive and require significant computational resources.
2. **Limited interpretability:** Some digital image processing algorithms may produce results that are difficult for humans to interpret, especially for complex or sophisticated algorithms.

3. **Dependence on quality of input:** The quality of the output of digital image processing algorithms is highly dependent on the quality of the input images. Poor quality input images can result in poor quality output.
4. **Limitations of algorithms:** Digital image processing algorithms have limitations, such as the difficulty of recognizing objects in cluttered or poorly lit scenes, or the inability to recognize objects with significant deformations or occlusions.

1.5 Summary

The main purpose of the material presented in this chapter is to provide a sense of perspective about the origins of digital image processing and, more important, about current and future areas of application of this technology. Although the coverage of these topics in this chapter was necessarily incomplete due to space limitations, it should have left you with a clear impression of the breadth and practical scope of digital image processing. As we proceed in the following chapters with the development of image processing theory and applications, numerous examples are provided to keep a clear focus on the utility and promise of these techniques. Upon concluding the study of the final chapter, a reader of this book will have arrived at a level of understanding that is the foundation for most of the work currently underway in this field.

CHAPTER 2
INTENSITY TRANSFORMATION AND SPATIAL FILTERING

2.1 Background

2.1.1 Intensity Transformations

All the image processing techniques discussed in this section are implemented in the *spatial domain*, which is simply the plane containing the pixels of an image. The spatial domain techniques operate directly on the pixels of an image as opposed, for example, to the frequency domain in which operations are performed on the Fourier transform of an image, rather than on the image itself. As you will learn in progressing through the book, some image processing tasks are easier or more meaningful to implement in the spatial domain while others are best suited for other approaches. Generally, spatial domain techniques are more efficient computationally and require less processing resources to implement.

The spatial domain processes we discuss in this chapter can be denoted by the expression

$$g(x,y) = T[f(x,y)] \tag{2.1}$$

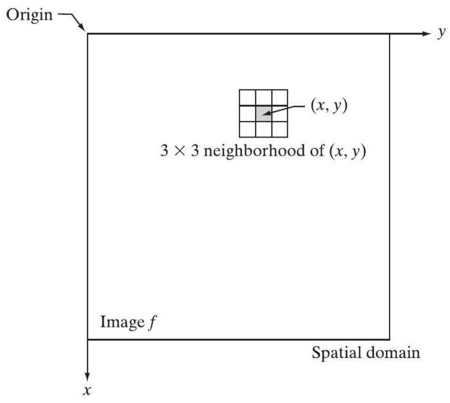


Figure 2.1: A 3×3 neighborhood about a point (x,y) in an image in the spatial domain. The neighborhood is moved from pixel to pixel in the image to generate an output image.

where $f(x,y)$ is the input image, $g(x,y)$ is the output image, and T is an operator on f defined over a neighborhood of point (x,y). The operator can apply to a single image (our principal focus in this chapter) or to a set of images, such as performing the pixel-by-pixel sum of a sequence of images for noise reduction. The point (x,y) shown is an arbitrary location in the image, and the small region shown containing the point is a neighborhood of (x,y). Typically, the neighborhood is rectangular, centered on (x,y), and much smaller in size than the image. Other neighborhood shapes, such as digital approximations to circles, are used sometimes, but rectangular shapes are by far the most prevalent because they are much easier to implement computationally.

The process that Fig. 2.1 illustrates consists of moving the origin of the neighborhood from pixel to pixel and applying the operator T to the pixels in the neighborhood to yield the output at that location. Thus, for any specific location (x,y), the value of the output image g at those coordinates is equal to the result of applying T to the neighborhood with origin at (x,y) in f. For example, suppose that the neighborhood is a square of size 3×3, and that operator T is defined as "compute the average intensity of the neighborhood."

Consider an arbitrary location in an image, say $(100, 150)$. Assuming that the origin of the neighborhood is at its center, the result, $g(100, 150)$, at that location is computed as the sum of $f(100, 150)$ and its 8-neighbors, divided by 9 (i.e., the average intensity of the pixels encompassed by the neighborhood). The origin of the neighborhood is then moved to the next location and the procedure is repeated to generate the next value of the output image g. Typically, the process starts at the top left of the input image and proceeds pixel by pixel in a horizontal scan, one row at a time. When the origin of the neighborhood is at the border of the image, part of the neighborhood will reside outside the image. The procedure is either to ignore the outside neighbors in the computations specified by T, or to pad the image with a border of 0 s or some other specified intensity values. The thickness of the padded border depends on the size of the neighborhood.

The procedure just described is called *spatial filtering*, in which the neighborhood, along with a predefined operation, is called a *spatial filter* (also referred to as a *spatial mask*, *kernel*, *template*, or *window*). The type of operation performed in the neighborhood determines the nature of the filtering process.

The smallest possible neighborhood is of size 1×1. In this case, g depends only on the value of f at a single point (x, y) and T in Eq. (2.1) becomes an intensity (also called *gray-level or mapping*) *transformation function* of the form

$$s = T(r) \tag{2.2}$$

where, for simplicity in notation, s and r are variables denoting, respectively, the intensity of g and f at any point (x, y). For example, if $T(r)$ has the form in Fig. 2.2(a), the effect of applying the transformation to every pixel of f to generate the corresponding pixels in g would be to produce an image of higher contrast than the original by darkening the intensity levels below k and brightening the levels above k. In this technique, sometimes called *contrast* *stretching*, values of r lower than k are compressed by the transformation function into a narrow range of s, toward black. The opposite is true for values of r higher than k. Observe how an intensity value r_0 is mapped to obtain the corresponding value s_0. In the limiting case shown in Fig. 2.2(b), $T(r)$ produces a two-level (binary) image. A mapping of this form is called a thresholding function. Some fairly simple, yet powerful, processing approaches can be formulated with intensity transformation functions. In this chapter, we use intensity transformations principally for image enhancement.

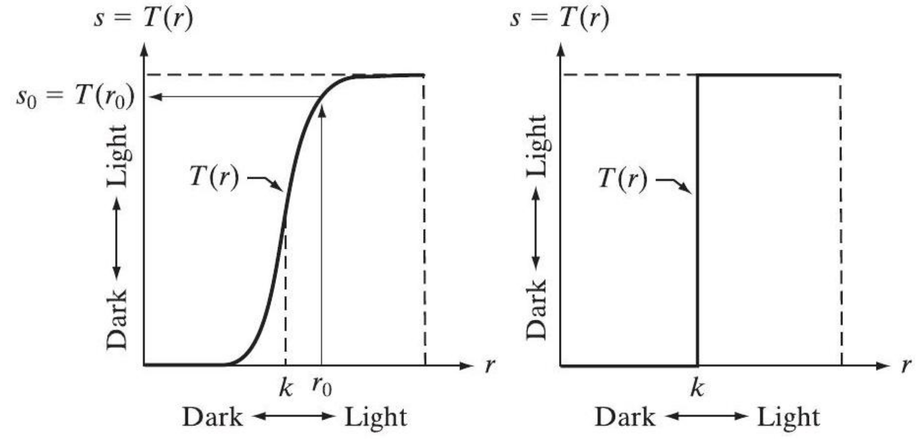


Figure 2.2: Intensity transformation functions. (a) Contrast stretching function. (b) Thresholding function.

2.1.2 About the Examples in This Chapter

Although intensity transformations and spatial filtering span a broad range of applications, most of the examples in this chapter are applications to image enhancement. Enhancement is the process of manipulating an image so that the result is more suitable than the original for a specific application. The word specific is important here because it establishes at the outset that enhancement techniques are problem oriented. Thus, for example, a method that is quite useful for enhancing X-ray images may not be the best approach for enhancing satellite images taken in the infrared band of the electromagnetic spectrum. There is no general "theory" of image enhancement. When an image is processed for visual interpretation, the viewer is the ultimate judge of how well a particular method works. When dealing with machine perception, a given technique is easier to quantify. For example, in an automated character-recognition system, the most appropriate enhancement method is the one that results in the best recognition rate, leaving aside other considerations such as computational requirements of one method over another.

2.2 Basic Intensity Transformation Functions

Intensity transformations are among the simplest of all image processing techniques. The values of pixels, before and after processing, will be denoted by r and s, respectively. As indicated in the previous section, these values are related by an expression of the form $s = T(r)$, where T is a transformation that maps a pixel value r into a pixel value s. Because we are dealing with digital quantities, values of a transformation function typically are stored in a one-dimensional array and the mappings from r to s are implemented via table lookups. For an 8 -bit environment, a lookup table containing the values of T will have 256 entries.

As an introduction to intensity transformations, consider Fig. 2.3, which shows three basic types of functions used frequently for image enhancement: linear (negative and identity transformations), logarithmic (log and inverse-log transformations), and power-law (nth power and nth root transformations). The identity function is the trivial case in which output intensities are identical to input intensities. It is included in the graph only for completeness.

2.2.1 Image Negatives

The negative of an image with intensity levels in the range $[0, L-1]$ is obtained by using the negative transformation shown in Fig. 2.3, which is given by the expression

$$s = L - 1 - r \tag{2.3}$$

Reversing the intensity levels of an image in this manner produces the equivalent of a photographic negative. This type of processing is particularly suited for enhancing white or gray detail embedded in dark regions of an image, especially when the black areas are dominant in size. Figure 2.4 shows an example.

2.2.2 Log Transformations

The general form of the log transformation in Fig. 2.3 is

$$s = c \log(1 + r) \tag{2.4}$$

where c is a constant, and it is assumed that $r \geq 0$. The shape of the log curve in Fig. 2.3 shows that this transformation maps a narrow range of low intensity values in the input into a wider range of output levels. The opposite is true of higher values of input levels. We use a transformation of this type to expand the values of dark pixels in an image while compressing the higher-level values. The opposite is true of the inverse log transformation.

Any curve having the general shape of the log functions shown in Fig. 2.3 would accomplish this spreading/compressing of intensity levels in an image, but the power-law transformations discussed in the next section are much more versatile for this purpose. The log function has the important characteristic that it compresses the dynamic range of images with large variations in pixel values. A classic illustration of an application in which

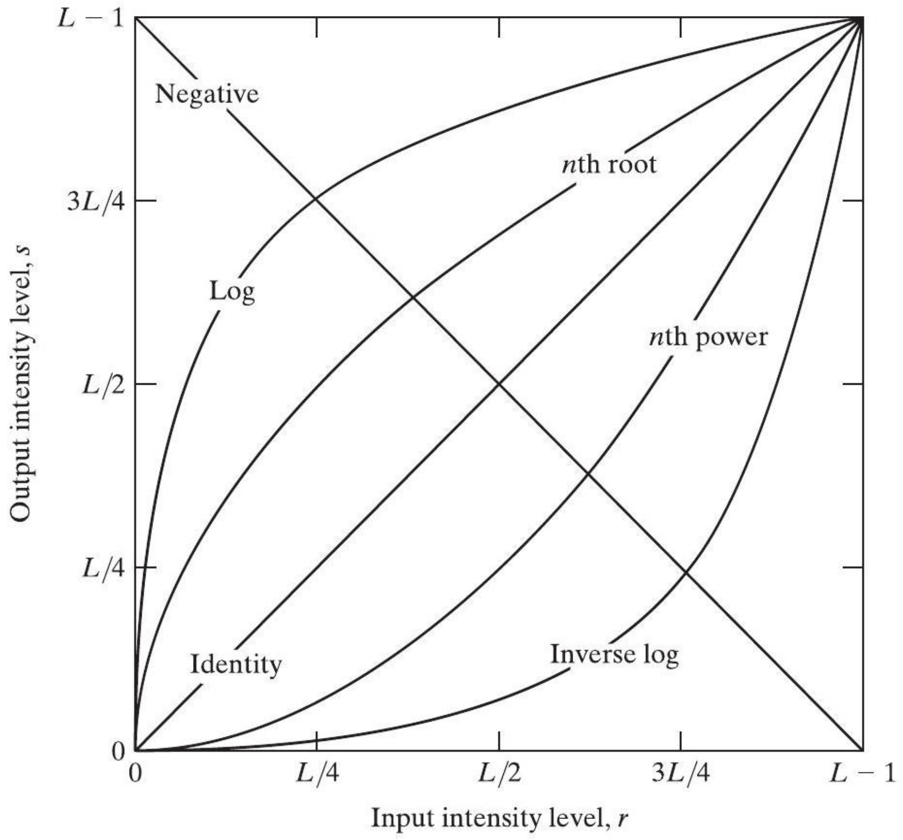


Figure 2.3: Some basic intensity transformation functions. All curves were scaled to fit in the range shown.

‘

pixel values have a large dynamic range is the Fourier spectrum. At the moment, we are concerned only with the image characteristics of spectra. It is not unusual to encounter spectrum values that range from 0 to 10^6 or higher. While processing numbers such as these presents no problems for a computer, image display systems generally will not be able to reproduce faithfully such a wide range of intensity values. The net effect is that a significant degree of intensity detail can be lost in the display of a typical Fourier spectrum.

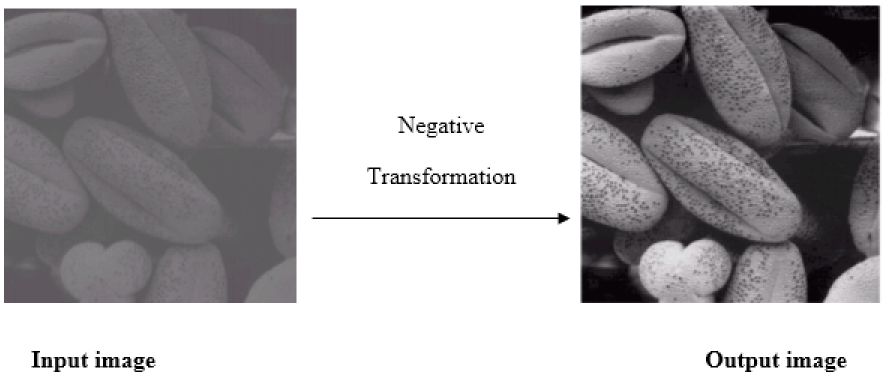


Figure 2.4: An example of a negative image obtained using the negative transformation in Eq. (2.3).

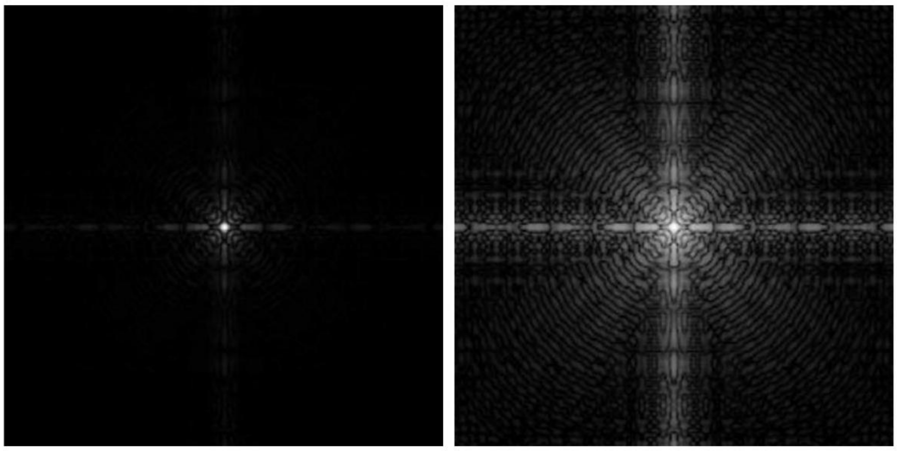


Figure 2.5: (a) Fourier spectrum. (b) Result of applying the log transformation in Eq.2.4 with c = 1.

As an illustration of log transformations, Fig. 2.5(a) shows a Fourier spectrum with values in the range 0 to 1.5×10^6. When these values are scaled linearly for display in an 8 -bit system, the brightest pixels will dominate the display, at the expense of lower (and just as important) values of the spectrum. The effect of this dominance is illustrated vividly by the relatively small area of the image in Fig. 2.5(a) that is not perceived as black. If, instead of displaying the values in this manner, we first apply Eq. (2.4) (with $c = 1$ in this case) to the spectrum values, then the range of values of the result becomes 0 to 6.2 , which is more manageable. Figure 2.5(b) shows the result of scaling this new range linearly and displaying the spectrum in the same 8-bit display. The wealth of detail visible in this image as compared to an unmodified display of the spectrum is evident from these pictures. Most of the Fourier spectra seen in image processing publications have been scaled in just this manner.

2.2.3 Power-Law (Gamma) Transformations

Power-law transformations have the basic form

$$s = cr^{\gamma} \tag{2.5}$$

where c and γ are positive constants. Sometimes Eq. (2.5) is written as $s = c(r+\varepsilon)^{\gamma}$ to account for an offset (that is, a measurable output when the input is zero). However, offsets typically are an issue of display calibration and as a result they are normally ignored in Eq. (2.5). Plots of s versus r for various values of γ are shown in Fig. 2.6. As in the case of the log transformation, power-law curves with fractional values of γ map a narrow range of dark input values into a wider range of output values, with the opposite being true for higher values of input levels. Unlike the log function, however, we notice here a family of possible transformation curves obtained simply by varying γ. As expected, we see in Fig. that curves generated with values of $\gamma > 1$ have exactly the opposite effect as those generated with values of $\gamma < 1$. Finally, we note that Eq. (2.5) reduces to the identity transformation when $c = \gamma = 1$.

A variety of devices used for image capture, printing, and display respond according to a power law. By convention, the exponent in the power-law

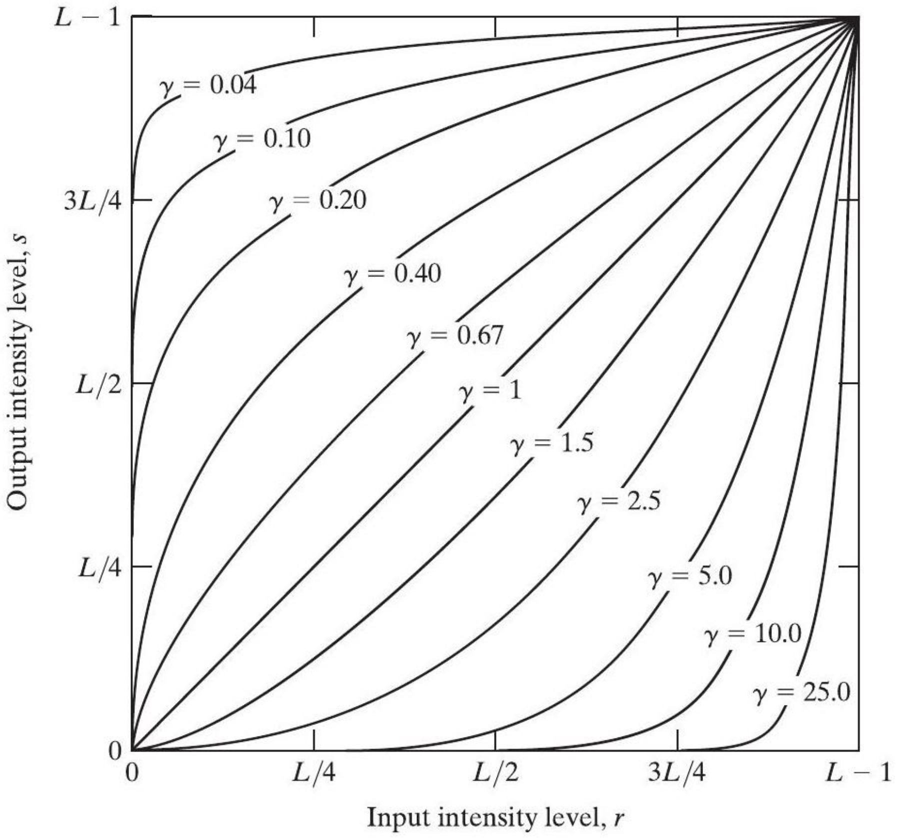


Figure 2.6: Plots of the equation $s = cr^{\gamma}$ for various values of γ (c=1 in all cases). All curves were scaled to fit in the range shown.

equation is referred to as *gamma* [hence our use of this symbol in Eq. (2.5)]. The process used to correct these power-law response phenomena is called *gamma correction*. For example, cathode ray tube (CRT) devices have an intensity-to-voltage response that is a power function, with exponents varying from approximately 1.8 to 2.5 . With reference to the curve for $\gamma = 2.5$ in Fig. 2.6, we see that such display systems would tend to produce images that are darker than intended. This effect is illustrated in Fig. 2.7. Figure 2.7(a) shows a simple intensity-ramp image input into a monitor. As expected, the output of the monitor appears darker than the input, as Fig. 2.7(b) shows.

Gamma correction in this case is straightforward. All we need to do is preprocess the input image before inputting it into the monitor by performing the transformation $s = r^{1/2.5} = r^{0.4}$. The result is shown in Fig. 2.7(c). When input into the same monitor, this gamma-corrected input produces an output that is close in appearance to the original image, as Fig. 2.7(d) shows. A similar analysis would apply to other imaging devices such as scanners and printers. The only difference would be the device-dependent value of gamma.

Gamma correction is important if displaying an image accurately on a computer screen is of concern. Images that are not corrected properly can look either bleached out, or, what is more likely, too dark. Trying to reproduce colors accurately also requires some knowledge of gamma correction because varying the value of gamma changes not only the intensity, but also the ratios of red to green to blue in a color image. Gamma correction has become increasingly important in the past few years, as the use of digital images for commercial purposes over the Internet has increased. It is not unusual that images created for a popular Web site will be viewed by millions of people, the majority of whom will have different monitors and/or monitor settings. Some computer systems even have partial gamma correction built in. Also, current image standards do not contain the value of gamma with which an image was created, thus complicating the issue further. Given these constraints, a reasonable approach when storing images in a Web site is to preprocess the images with a gamma that represents an "average" of the types of monitors and computer systems that one expects in the open market at any given point in time.

In addition to gamma correction, power-law transformations are useful for general-purpose contrast manipulation. Figure 2.8(a) shows a magnetic resonance image (MRI) of an upper thoracic human spine with a fracture dislocation and spinal cord impingement. The fracture is visible near the vertical center of the spine, approximately one-fourth of the way down from the top of the picture. Because the given image is predominantly dark, an

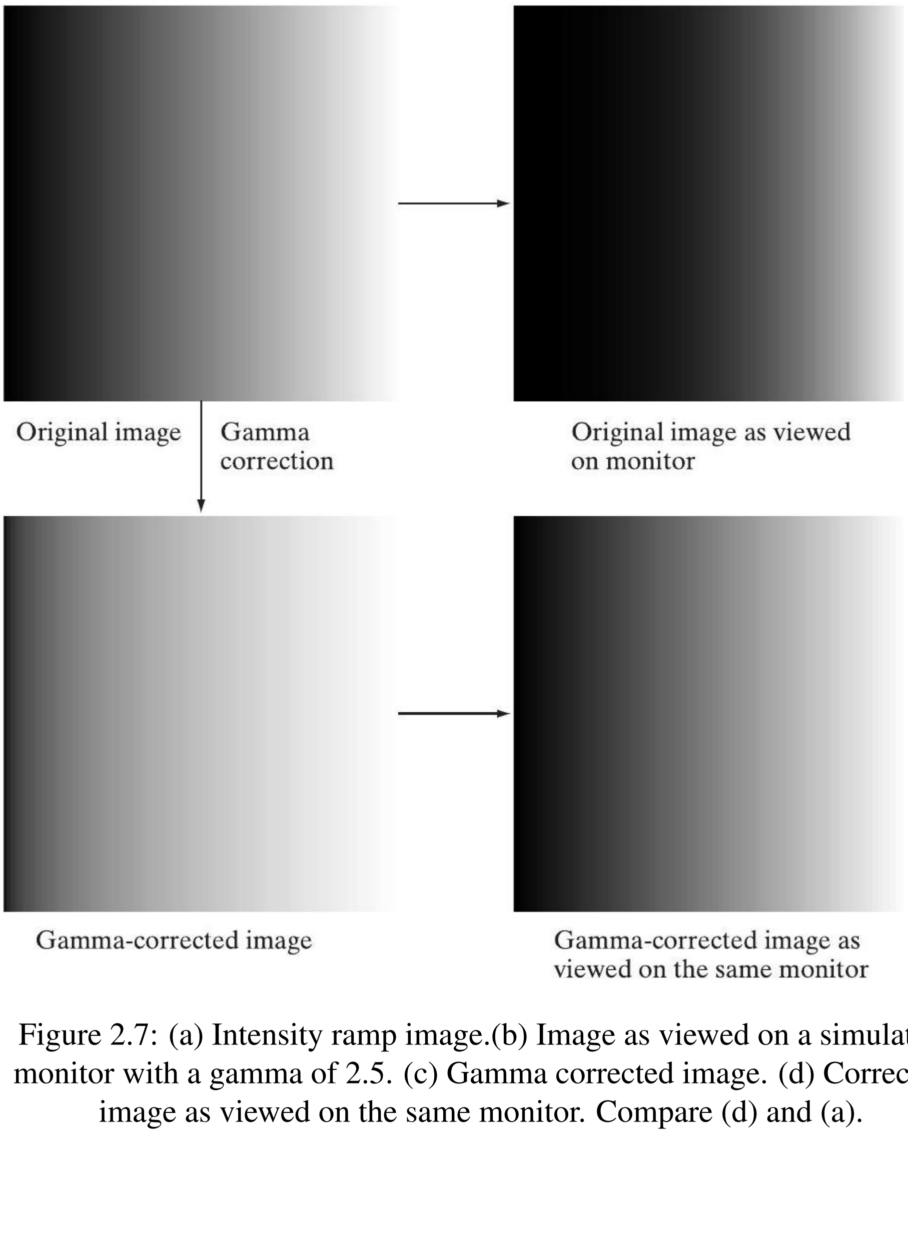

Original image
Gamma correction
Original image as viewed on monitor
Gamma-corrected image
Gamma-corrected image as viewed on the same monitor

Figure 2.7: (a) Intensity ramp image.(b) Image as viewed on a simulated monitor with a gamma of 2.5. (c) Gamma corrected image. (d) Corrected image as viewed on the same monitor. Compare (d) and (a).

expansion of intensity levels is desirable. This can be accomplished with a power-law transformation with a fractional exponent. The other images shown in the figure were obtained by processing Fig. 2.8(a) with the power-law transformation function of Eq. (2.5). The values of gamma corresponding to images (b) through (d) are $0.6, 0.4$, and 0.3 , respectively (the value of c was 1 in all cases). We note that, as gamma decreased from 0.6 to 0.4 , more detail became visible. A further decrease of gamma to 0.3 enhanced a little more detail in the background, but began to reduce contrast to the point where the image started to have a very slight "washed-out" appearance, especially in the background. By comparing all results, we see that the best enhancement in terms of contrast and discernable detail was obtained with $\gamma = 0.4$. A value of $\gamma = 0.3$ is an approximate limit below which contrast in this particular image would be reduced to an unacceptable level.

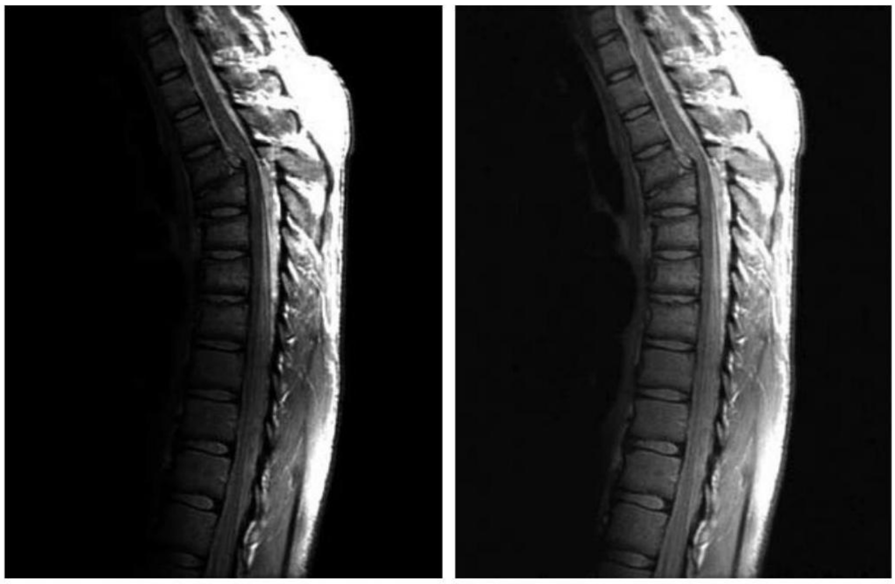


Figure 2.8: (a) Magnetic resonance image (MRI) of a fractured human spine. (b)–(d) Results of applying the transformation in Eq. 2.5

Figure 2.9(a) shows the opposite problem of Fig.2.8(a). The image to be processed now has a washed-out appearance, indicating that a compression of intensity levels is desirable. This can be accomplished with Eq. (2.5) using values of γ greater than 1. The results of processing Fig. 2.9(a) with $\gamma = 3.0, 4.0$, and 5.0 are shown in Fig. 2.9(b) through (d). Suitable results were obtained with gamma values of 3.0 and 4.0 , the latter having a slightly more appealing appearance because it has higher contrast. The result obtained with $\gamma = 5.0$ has areas that are too dark, in which some detail is lost. The dark region to the left of the main road in the upper left quadrant is an example of such an area.

2.2.4 Piecewise-Linear Transformation Functions

A complementary approach to the methods discussed in the previous three sections is to use piecewise linear functions. The principal advantage of piecewise linear functions over the types of functions we have discussed thus far is that the form of piecewise functions can be arbitrarily complex. In fact, as you will see shortly, a practical implementation of some important transformations can be formulated only as piecewise functions. The principal disadvantage of piecewise functions is that their specification requires considerably more user input.

Contrast Stretching

One of the simplest piecewise linear functions is a contrast-stretching transformation. Low-contrast images can result from poor illumination, lack of dynamic range in the imaging sensor, or even the wrong setting of a lens aperture during image acquisition. Contrast stretching is a process that expands the range of intensity levels in an image so that it spans the full intensity range of the recording medium or display device.

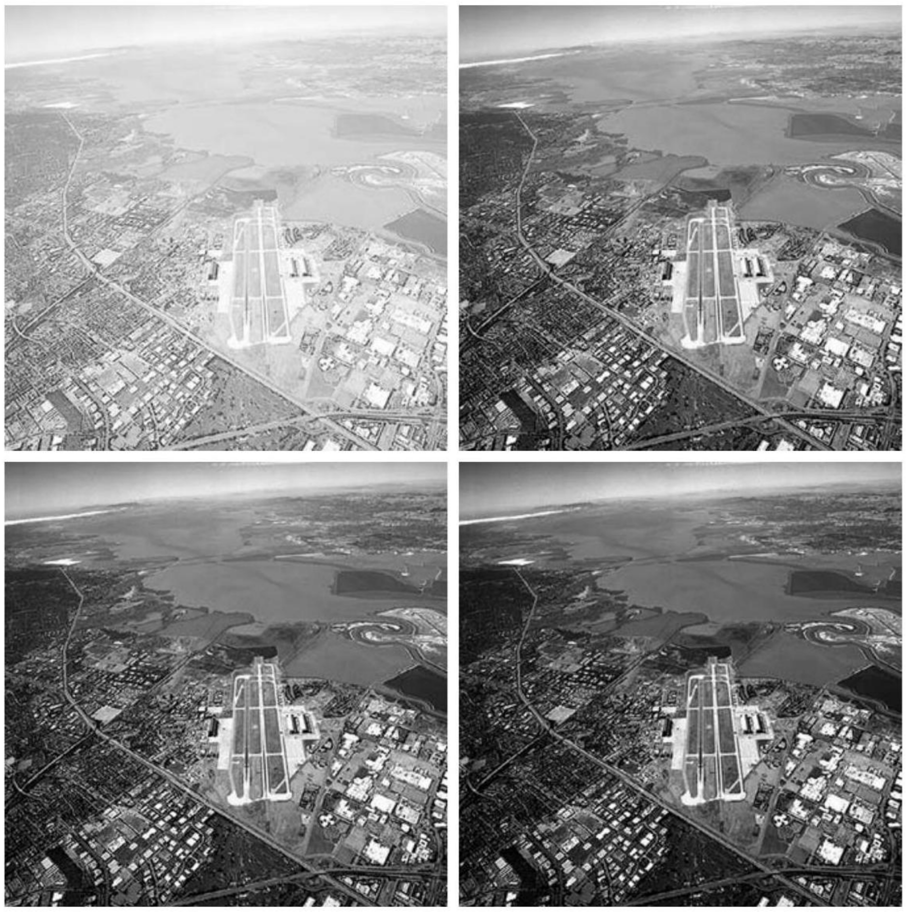


Figure 2.9: (a) Aerial image. (b)–(d) Results of applying the transformation in Eq. 2.5

Figure 2.10(a) shows a typical transformation used for contrast stretching. The locations of points (r_1, s_1) and (r_2, s_2) control the shape of the transformation function. If $r_1 = s_1$ and $r_2 = s_2$, the transformation is a linear function that produces no changes in intensity levels. If $r_1 = r_2, s_1 = 0$ and $s_2 = L - 1$, the transformation becomes a thresholding function that creates a binary image. Intermediate values of (r_1, s_1) and (r_2, s_2) produce various degrees of spread in the intensity levels of the output image, thus affecting its contrast. In general, $r_1 \leq r_2$ and $s_1 \leq s_2$ is assumed so that the function is single valued and monotonically increasing. This condition preserves the order of intensity levels, thus preventing the creation of intensity artifacts in the processed image.

Figure 2.10(b) shows an 8-bit image with low contrast. Figure 2.10(c) shows the result of contrast stretching, obtained by setting $(r_1, s_1) = (r_{\min}, 0)$ and $(r_2, s_2) = (r_{\max}, L-1)$, where $r_{\min}$ and $r_{\max}$ denote the minimum and maximum intensity levels in the image, respectively. Thus, the transformation function stretched the levels linearly from their original range to the full range $[0, L-1]$. Finally, Fig.2.10(d) shows the result of using the thresholding function defined previously, with $(r_1, s_1) = (m, 0)$ and $(r_2, s_2) = (m, L-1)$, where m is the mean intensity level in the image. The original image on which these results are based is a scanning electron microscope image of pollen, magnified approximately 700 times.

Intensity-level slicing

Highlighting a specific range of intensities in an image often is of interest. Applications include enhancing features such as masses of water in satellite imagery and enhancing flaws in X-ray images. The process, often called intensity-level slicing, can be implemented in several ways, but most are variations of two basic themes. One approach is to display in one value (say, white) all the values in the range of interest and in another (say, black) all other intensities. This transformation, shown in Fig. 2.11(a), produces a binary image. The second approach, based on the transformation in Fig. 2.11(b), brightens (or darkens) the desired range of intensities but leaves all other intensity levels in the image unchanged.

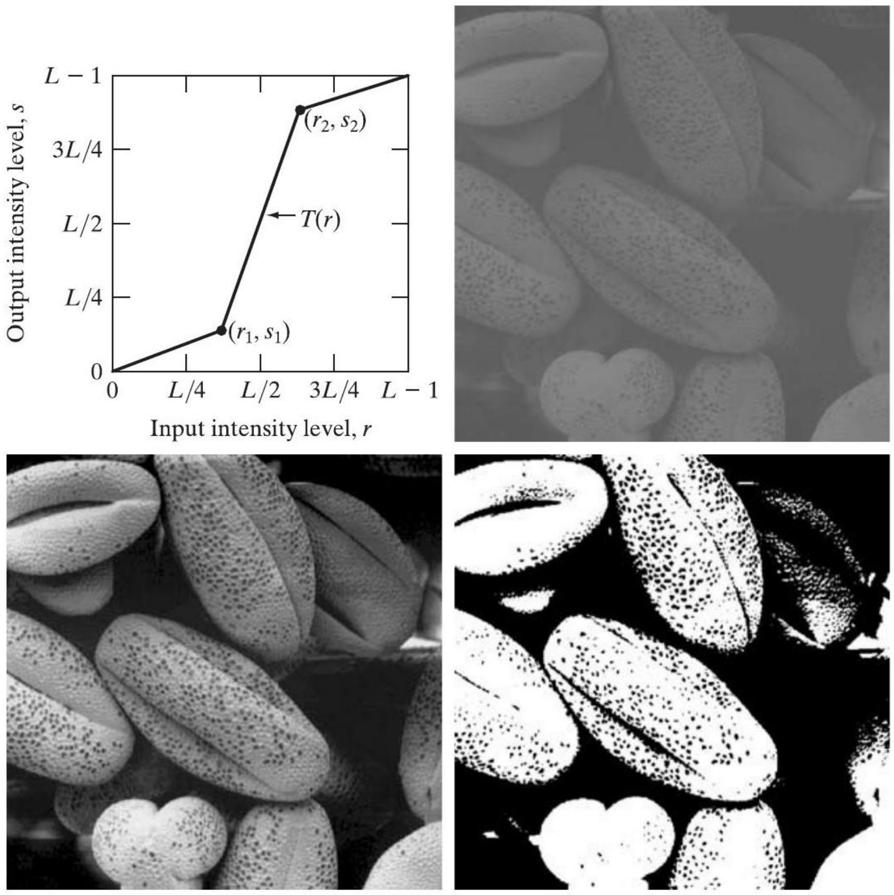


Figure 2.10: Contrast stretching. (a) Form of transformation function. (b) A low-contrast image. (c) Result of contrast stretching. (d) Result of thresholding.

Bit-plane slicing

Pixels are digital numbers composed of bits. For example, the intensity of each pixel in a 256 -level gray-scale image is composed of 8 bits (i.e., one byte). Instead of highlighting intensity-level ranges, we could highlight the contribution made to total image appearance by specific bits. As Fig. 2.12 illustrates, an 8-bit image may be considered as being composed of eight 1-bit

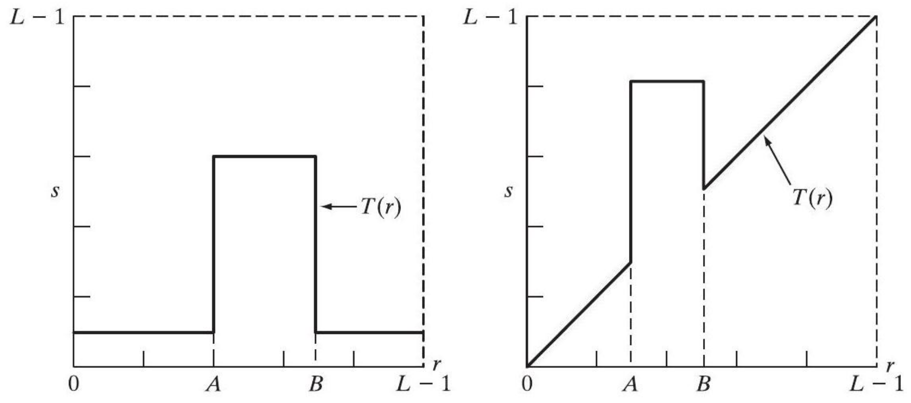


Figure 2.11: (a) This transformation highlights intensity range [A, B] and reduces all other intensities to a lower level. (b) This transformation highlights range [A, B] and preserves all other intensity levels.

planes, with plane 1 containing the lowest-order bit of all pixels in the image and plane 8 all the highest-order bits.

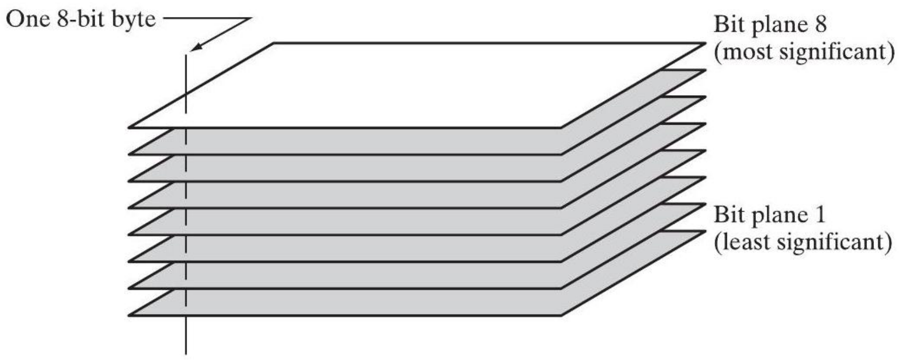


Figure 2.12: Bit-plane representation of an 8-bit image.

Figure 2.13(a) shows an 8-bit gray-scale image and Figs. 2.13(b) through (i) are its eight 1-bit planes, with Fig. 2.13(b) corresponding to the lowest-order bit. Observe that the four higher-order bit planes, especially the last two, contain a significant amount of the visually significant data. The lower-order planes contribute to more subtle intensity details in the image. The original image has a gray border whose intensity is 194 . Notice that the corresponding borders of some of the bit planes are black (0), while others are white (1). To see why, consider a pixel in, say, the middle of the lower border of Fig. 2.13(a). The corresponding pixels in the bit planes, starting with the highest-order plane, have values 1 1 0 0 0 0 1 0, which is the binary representation of decimal 194.The value of any pixel in the original image can be similarly reconstructed from its corresponding binary-valued pixels in the bit planes.

In terms of intensity transformation functions, it is not difficult to show that the binary image for the 8th bit plane of an 8-bit image can be obtained by processing the input image with a thresholding intensity transformation function that maps all intensities between 0 and 127 to 0 and maps all levels between 128 and 255 to 1. The binary image in Fig. 2.13(i) was obtained in just this manner. Decomposing an image into its bit planes is useful for

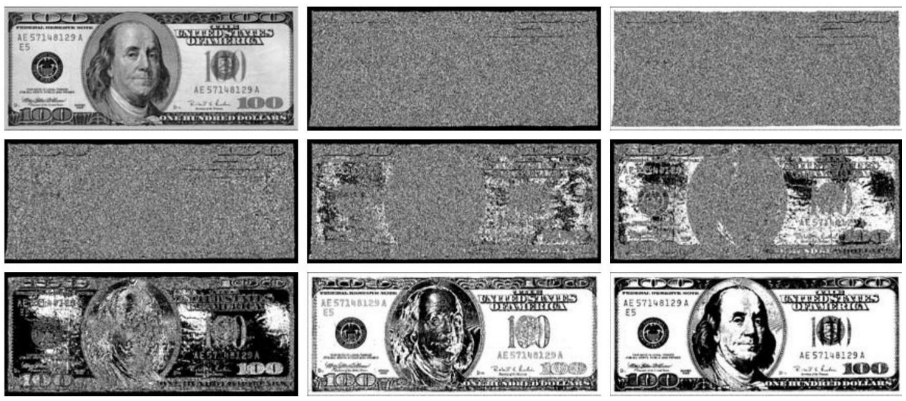


Figure 2.13: (a) An 8-bit gray-scale image of size 500 $\times$ 1192 pixels. (b) through (i) Bit planes 1 through 8, with bit plane 1 corresponding to the least significant bit. Each bit plane is a binary image.

analyzing the relative importance of each bit in the image, a process that aids in determining the adequacy of the number of bits used to quantize the image. Also, this type of decomposition is useful for image compression, in which fewer than all planes are used in reconstructing an image. For example, Fig. 2.14(a) shows an image reconstructed using bit planes 8 and 7. The reconstruction is done by multiplying the pixels of the nth plane by the constant 2^{n-1}. This is nothing more than converting the nth significant binary bit to decimal. Each plane used is multiplied by the corresponding constant, and all planes used are added to obtain the gray scale image. Thus, to obtain Fig. 2.14(a), we multiplied bit plane 8 by 128, bit plane 7 by 64, and added the two planes. Although the main features of the original image were restored, the reconstructed image appears flat, especially in the background. This is not surprising because two planes can produce only four distinct intensity levels. Adding plane 6 to the reconstruction helped the situation, as Fig. 2.14(b) shows. Note that the background of this image has perceptible false contouring. This effect is reduced significantly by adding the 5th plane to the reconstruction, as Fig. 2.14(c) illustrates. Using more planes in the reconstruction would not contribute significantly to the appearance of this image. Thus, we conclude that storing the four highest-order bit planes would allow us to reconstruct the original image in acceptable detail. Storing these four planes instead of the original image requires 50% less storage (ignoring memory architecture issues).

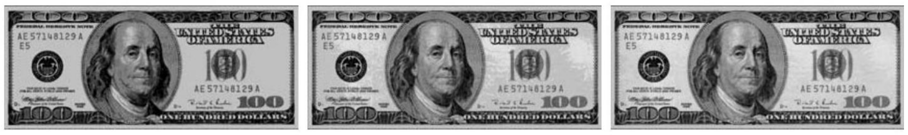


Figure 2.14: Images reconstructed using (a) bit planes 8 and 7; (b) bit planes 8, 7, and 6; and (c) bit planes 8, 7, 6, and 5. Compare (c) with Fig. 2.13(a).

2.3 Histogram Processing

The *histogram* of a digital image with intensity levels in the range $[0, L-1]$ is a discrete function $h(r_k) = n_k$, where r_k is the kt th intensity value and n_k is the number of pixels in the image with intensity r_k. It is common practice to normalize a histogram by dividing each of its components by the total number of pixels in the image, denoted by the product MN, where, as usual, M and N are the row and column dimensions of the image. Thus, a normalized histogram is given by $p(r_k) = r_k/MN$, for $k = 0, 1, 2, \ldots, L-1$. Loosely speaking, $p(r_k)$ is an estimate of the probability of occurrence of intensity level r_k in an image. The sum of all components of a normalized histogram is equal to 1.

Histograms are the basis for numerous spatial domain processing techniques. Histogram manipulation can be used for image enhancement, as shown in this section. In addition to providing useful image statistics, we shall see in subsequent chapters that the information inherent in histograms also is quite useful in other image processing applications, such as image compression and segmentation. Histograms are simple to calculate in software and also lend themselves to economic hardware implementations, thus making them a popular tool for real-time image processing.

As an introduction to histogram processing for intensity transformations, consider Fig. 2.15, which is the pollen image of Fig. 2.10 shown in four basic intensity characteristics: dark, light, low contrast, and high contrast. The right side of the figure shows the histograms corresponding to these images. The horizontal axis of each histogram plot corresponds to intensity values, r_k. The vertical axis corresponds to values of $h(r_k) = n_k$ or $p(r_k) = n_k/MN$ if the values are normalized. Thus, histograms may be viewed graphically simply as plots of $h(r_k) = n_k$ versus r_k or $p(r_k) = n_k/MN$ versus r_k.

We note in the dark image that the components of the histogram are concentrated on the low (dark) side of the intensity scale. Similarly, the components of the histogram of the light image are biased toward the high side of the scale. An image with low contrast has a narrow histogram located typically toward the middle of the intensity scale. For a monochrome image this implies a dull, washed-out gray look. Finally, we see that the components of the histogram in the high-contrast image cover a wide range of the intensity

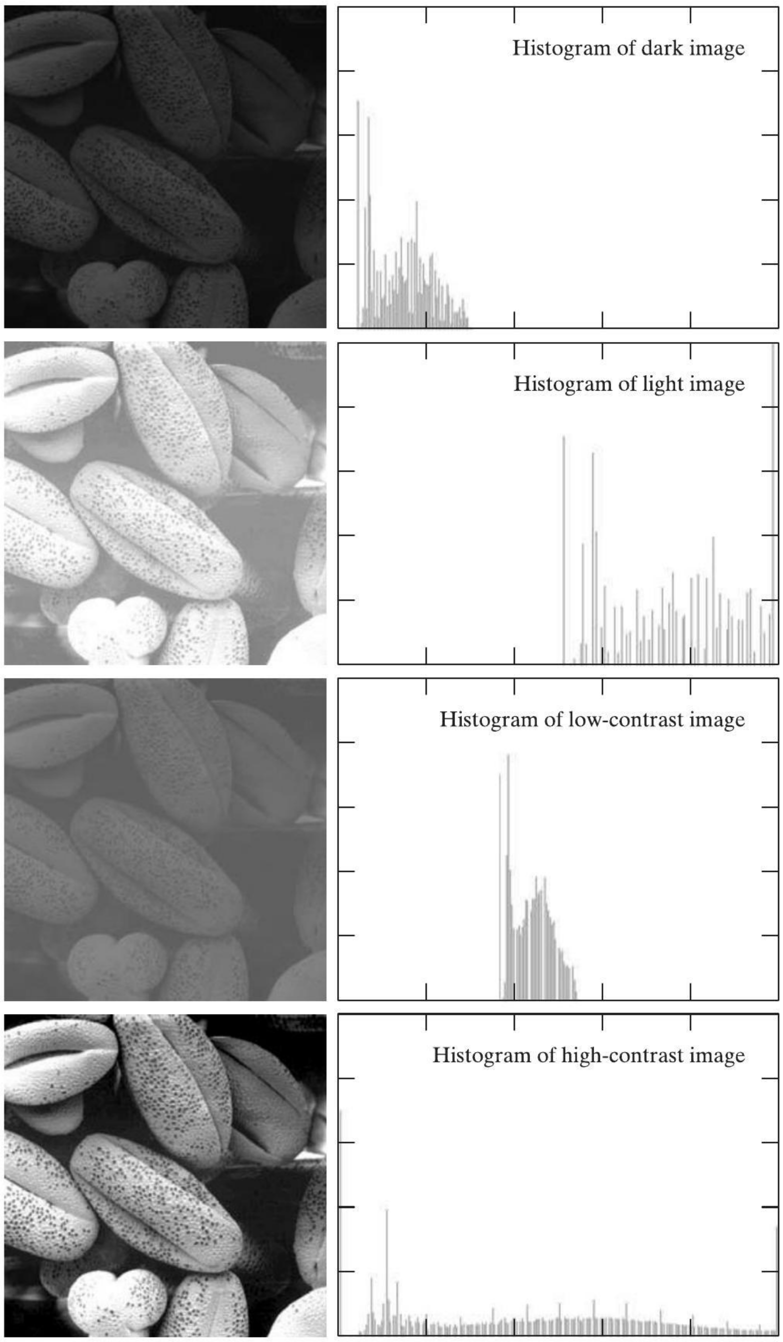


Figure 2.15: Four basic image types: dark, light, low contrast, high contrast, and their corresponding histograms.

scale and, further, that the distribution of pixels is not too far from uniform, with very few vertical lines being much higher than the others. Intuitively, it is reasonable to conclude that an image whose pixels tend to occupy the entire range of possible intensity levels and, in addition, tend to be distributed uniformly, will have an appearance of high contrast and will exhibit a large variety of gray tones. The net effect will be an image that shows a great deal of gray-level detail and has high dynamic range. It will be shown shortly that it is possible to develop a transformation function that can automatically achieve this effect, based only on information available in the histogram of the input image.

2.3.1 Histogram Equalization

Consider for a moment continuous intensity values and let the variable r denote the intensities of an image to be processed. As usual, we assume that r is in the range $[0, L-1]$, with $r = 0$ representing black and $r = L-1$ representing white. For r satisfying these conditions, we focus attention on transformations (intensity mappings) of the form

$$s = T(r) \quad 0 \leq r \leq L-1 \tag{2.6}$$

that produce an output intensity level s for every pixel in the input image having intensity r. We assume that:
(a) $T(r)$ is a monotonically increasing function in the interval $0 \leq r \leq L-1$; and
(b) $0 \leq T(r) \leq L-1$ for $0 \leq r \leq L-1$.

In some formulations to be discussed later, we use the inverse

$$r = T^{-1}(s) \quad 0 \leq s \leq L-1 \tag{2.7}$$

in which case we change condition (a) to
$(\mathbf{a}')T(r)$ is a strictly monotonically increasing function in the interval $0 \leq r \leq L-1$.

The requirement in condition (a) that $T(r)$ be monotonically increasing guarantees that output intensity values will never be less than corresponding input values, thus preventing artifacts created by reversals of intensity. Condition (b) guarantees that the range of output intensities is the same as the input. Finally, condition (a$'$) guarantees that the mappings from s back to r will be one-to-one, thus preventing ambiguities. Figure 2.16(a) shows a function that satisfies conditions (a) and (b). Here, we see that it is possible for multiple values to map to a single value and still satisfy these two conditions. That is, a monotonic transformation function performs a one-to-one or many-to-one mapping. This is perfectly fine when mapping from r to s. However, Fig.2.16(a) presents a problem if we wanted to recover the values of r uniquely from the mapped values (inverse mapping can be visualized by reversing the direction of the arrows). This would be possible for the inverse mapping of s_k in Fig. 2.16(a), but the inverse mapping of s_q is a *range* of values, which, of course, prevents us in general from recovering the original value of r that resulted in s_q. As Fig. 2.16(b) shows, requiring that $T(r)$ be strictly monotonic guarantees that the inverse mappings will be *single valued* (i.e., the mapping is one-to-one in both directions). This is a theoretical requirement that allows us to derive some important histogram processing techniques later in this chapter. Because in practice we deal with integer intensity values, we are forced to round all results to their nearest integer values. Therefore, when strict monotonicity is not satisfied, we address the problem of a non-unique inverse transformation by looking for the closest integer matches.

The intensity levels in an image may be viewed as random variables in the interval $[0, L-1]$. A fundamental descriptor of a random variable is its probability density function (PDF). Let $p_r(r)$ and $p_s(s)$ denote the PDFs of

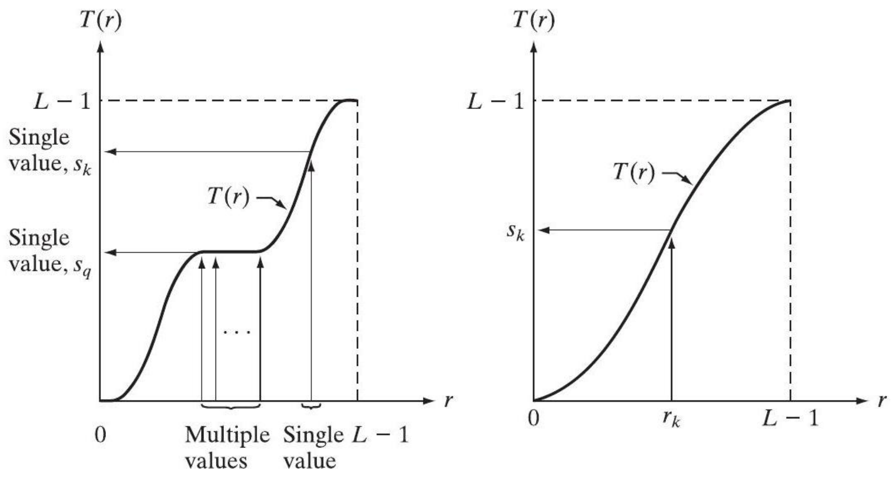


Figure 2.16: (a) Monotonically increasing function, showing how multiple values can map to a single value. (b) Strictly monotonically increasing function. This is a one-to-one mapping, both ways.

r and s, respectively, where the subscripts on p are used to indicate that p_r and p_s are different functions in general. A fundamental result from basic probability theory is that if $p_r(r)$ and $T(r)$ are known, and $T(r)$ is continuous and differentiable over the range of values of interest, then the PDF of the transformed (mapped) variable s can be obtained using the simple formula

$$p_s(s) = p_r(r)\left|\frac{dr}{ds}\right| \tag{2.8}$$

Thus, we see that the PDF of the output intensity variable, s, is determined by the PDF of the input intensities and the transformation function used [recall that r and s are related by $T(r)$].

A transformation function of particular importance in image processing has the form

$$s = T(r) = (L-1)\int_0^r p_r(w)dw \tag{2.9}$$

where w is a dummy variable of integration. The right side of this equation is recognized as the cumulative distribution function (CDF) of random variable r. Because PDFs always are positive, and recalling that the integral of a function is the area under the function, it follows that the transformation function of Eq. (2.9) satisfies condition (a) because the area under the function cannot decrease as r increases. When the upper limit in this equation is $r = (L-1)$, the integral evaluates to 1 (the area under a PDF curve always is 1), so the maximum value of s is $(L-1)$ and condition (b) is satisfied also.

To find the $p_s(s)$ corresponding to the transformation just discussed, we use Eq. 2.8. We know from Leibniz's rule in basic calculus that the derivative of a definite integral with respect to its upper limit is the integrand evaluated at the limit. That is,

$$\frac{ds}{dr} = \frac{dT(r)}{dr} = (L-1)\frac{d}{dr}\left[\int_0^r p_r(w)dw\right] = (L-1)p_r(r) \qquad (2.10)$$

Substituting this result for dr/ds in Eq. (2.8), and keeping in mind that all probability values are positive, yields

$$p_s(s) = p_r(r)\left|\frac{dr}{ds}\right| = p_r(r)\left|\frac{1}{(L-1)p_r(r)}\right| = \frac{1}{L-1} \quad 0 \le s \le L-1 \quad (2.11)$$

We recognize the form of $p_s(s)$ in the last line of this equation as a *uniform* probability density function. Simply stated, we have demonstrated that performing the intensity transformation in Eq. (2.9) yields a random variable, s, characterized by a uniform PDF. It is important to note from this equation that $T(r)$ depends on $p_r(r)$ but, as Eq. (2.11) shows, the resulting $p_s(s)$ always is uniform, *independently* of the form of $p_r(r)$. Figure 2.17 illustrates these concepts.

For discrete values, we deal with probabilities (histogram values) and summations instead of probability density functions and integrals. As men-

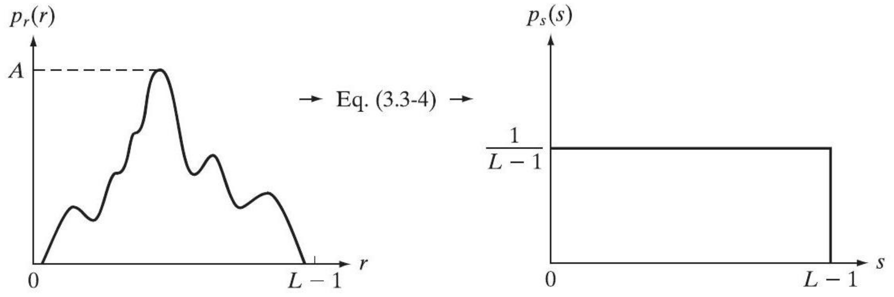


Figure 2.17: (a) An arbitrary PDF. (b) Result of applying the transformation in Eq. (2.9) to all intensity levels, r. The resulting intensities, s, have a uniform PDF, independently of the form of the PDF of the r 's.

tioned earlier, the probability of occurrence of intensity level r_k in a digital image is approximated by

$$p_r(r_k) = \frac{n_k}{MN} \quad k = 0, 1, 2, \ldots, L-1 \tag{2.12}$$

where MN is the total number of pixels in the image, n_k is the number of pixels that have intensity r_k, and L is the number of possible intensity levels in the image (e.g., 256 for an 8-bit image). As noted in the beginning of this section, a plot of $p_r(r_k)$ versus r_k is commonly referred to as a *histogram.*

The discrete form of the transformation in Eq. (2.9) is

$$s_k = T(r_k) = (L-1)\sum_{j=0}^{k} p_r(r_j) = \frac{(L-1)}{MN}\sum_{j=0}^{k} n_j \quad k = 0, 1, 2, \ldots, L-1 \tag{2.13}$$

Thus, a processed (output) image is obtained by mapping each pixel in the input image with intensity r_k into a corresponding pixel with level s_k in the output image, using Eq. (2.13). The transformation (mapping) $T(r_k)$ in this equation is called a *histogram equalization* or *histogram linearization* transformation. It is not difficult to show that this transformation satisfies conditions (a) and (b) stated previously in this section.

Example: Histogram Equalization.

The left column in Fig. 2.18 shows the four images from Fig. 2.15, and the center column shows the result of performing histogram equalization on each of these images. The first three results from top to bottom show significant improvement. As expected, histogram equalization did not have much effect on the fourth image because the intensities of this image already span the full intensity scale.

The third column in Fig. 2.18 shows the histograms of the equalized images. It is of interest to note that, while all these histograms are different, the histogram equalized images themselves are visually very similar. This is not unexpected because the basic difference between the images on the left column is one of contrast, not content. In other words, because the images have the same content, the increase in contrast resulting from histogram equalization was enough to render any intensity differences in the equalized images visually indistinguishable. Given the significant contrast differences between the original images, this example illustrates the power of histogram equalization as an adaptive contrast enhancement tool.

2.3.2 Histogram Matching (Specification)

As indicated in the preceding discussion, histogram equalization automatically determines a transformation function that seeks to produce an output image that has a uniform histogram. When automatic enhancement is desired, this is a good approach because the results from this technique are predictable and the method is simple to implement. We show in this section that there are applications in which attempting to base enhancement on a uniform histogram is not the best approach. In particular, it is useful sometimes to be able to specify the shape of the histogram that we wish the processed image to have. The method used to generate a processed image that has a specified histogram is called *histogram matching* or *histogram specification*.

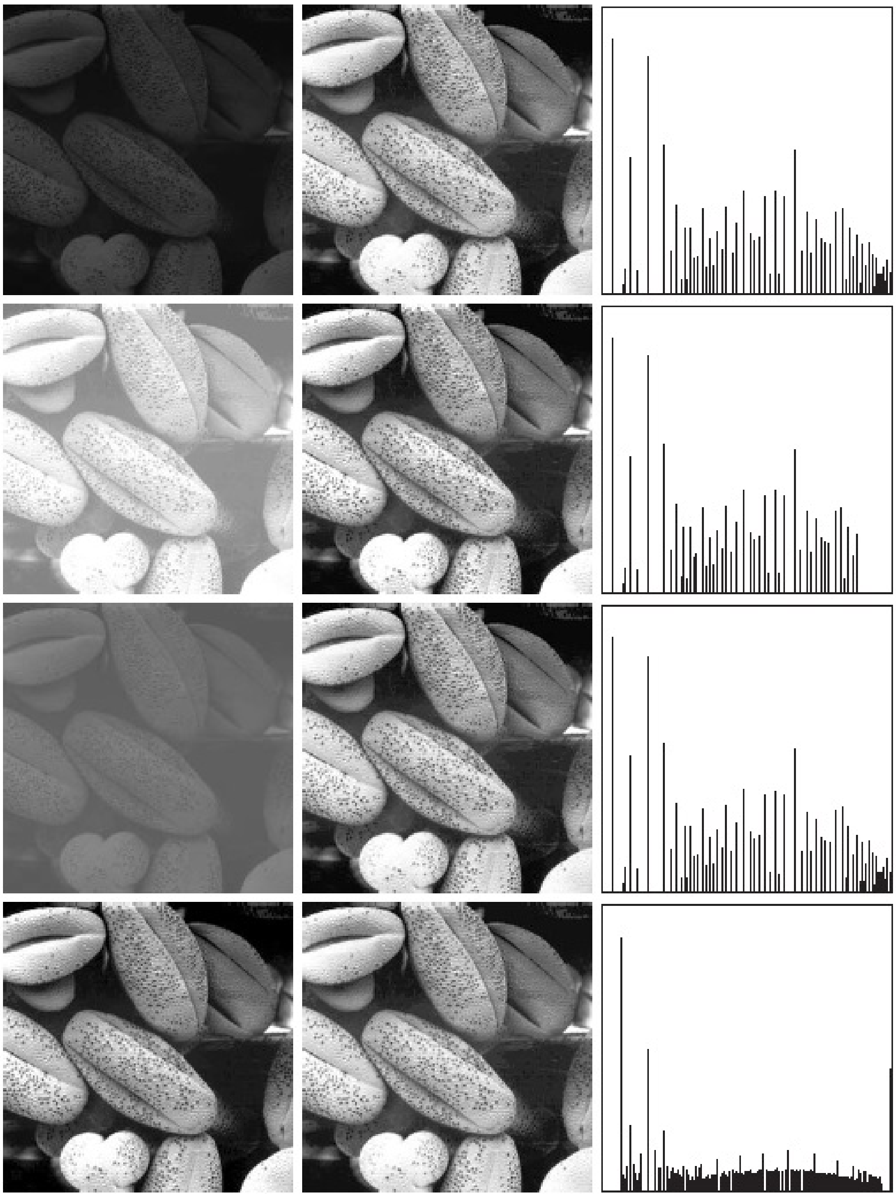


Figure 2.18: Left column: images from Fig. 2.15. Center column: corresponding histogram equalized images. Right column: histograms of the images in the center column.

Let s_k be a discrete random variable with the property

$$s_k = T\left(r_k\right) = (L-1)\sum_{j=0}^{k} p_r\left(r_j\right) = \frac{(L-1)}{MN}\sum_{j=0}^{k} n_j \quad k = 0,1,2,\ldots,L-1 \tag{2.14}$$

where, MN is the total number of pixels in the image, n_j is the number of pixels that have intensity value r_j, and L is the total number of possible intensity levels in the image. Similarly, given a specific value of s_k,

Suppose that we define a random variable z with the property

$$G\left(z_q\right) = (L-1)\sum_{i=0}^{q} p_z\left(z_i\right) \tag{2.15}$$

for a value of q, so that

$$G\left(z_q\right) = s_k \tag{2.16}$$

where $p_z\left(z_i\right)$, is the i th value of the specified histogram. As before, we find the desired value z_q by obtaining the inverse transformation:

$$z_q = G^{-1}\left(s_k\right) \tag{2.17}$$

In other words, this operation gives a value of z for each value of s; thus, it performs a *mapping* from s to z.

In practice, we do not need to compute the inverse of G. Because we deal with intensity levels that are integers (e.g., 0 to 255 for an 8 -bit image), it is a simple matter to compute all the possible values of G using Eq. (2.15) for $q = 0,1,2,\ldots,L-1$. These values are scaled and rounded to their nearest integer values spanning the range $[0, L-1]$. The values are stored in a table. Then, given a particular value of s_k, we look for the closest match in the values stored in the table. If, for example, the 64th entry in the table is the closest to s_k, then $q = 63$ (recall that we start counting at 0) and z_{63} is the best solution to Eq. (2.16). Thus, the given value s_k would be associated with z_{63} (i.e., that specific value of s_k would map to z_{63}). Because the z s are intensities used as the basis for specifying the histogram $p_z(z)$, it follows that $z_0 = 0$, $z_1 = 1, \ldots, z_{L-1} = L-1$, so z_{63} would have the intensity value 63 . By repeating this procedure, we would find the mapping of each value of s_k to the value of z_q that is the closest solution to Eq. (2.16). These mappings are the solution to the histogram-specification problem.

Recalling that the s_k s are the values of the histogram-equalized image, we may summarize the histogram-specification procedure as follows:

1. Compute the histogram $p_r(r)$ of the given image, and use it to find the histogram equalization transformation in Eq. (2.14). Round the resulting values, s_k, to the integer range $[0, L-1]$.
2. Compute all values of the transformation function G using the Eq. (2.15) for $q = 0, 1, 2, \ldots, L-1$, where $p_z(z_i)$ are the values of the specified histogram. Round the values of G to integers in the range $[0, L-1]$. Store the values of G in a table.
3. For every value of $s_k, k = 0, 1, 2, \ldots, L-1$, use the stored values of G from step 2 to find the corresponding value of z_q so that $G(z_q)$ is closest to s_k and store these mappings from s to z. When more than one value of z_q satisfies the given s_k (i.e., the mapping is not unique), choose the smallest value by convention.
4. Form the histogram-specified image by first histogram-equalizing the input image and then mapping every equalized pixel value, s_k, of this image to the corresponding value z_q in the histogram-specified image using the mappings found in step 3. As in the continuous case, the intermediate step of equalizing the input image is conceptual. It can be skipped by combining the two transformation functions, T and G^{-1}.

2.4 Fundamentals of Spatial Filtering

In this section, we introduce several basic concepts underlying the use of spatial filters for image processing. Spatial filtering is one of the principal tools used in this field for a broad spectrum of applications, so it is highly advisable that you develop a solid understanding of these concepts. The examples in this section deal mostly with the use of spatial filters for image enhancement.

The name *filter* is borrowed from frequency domain processing, which is the topic of the next chapter, where "filtering" refers to accepting (passing) or rejecting certain frequency components. For example, a filter that passes low frequencies is called a *lowpass* filter.The net effect produced by a lowpass filter is to blur (smooth) an image.We can accomplish a similar smoothing directly on the image itself by using spatial filters (also called *spatial masks*, *kernels*, *templates*, and *windows*). In fact, there is a one-to-one correspondence between linear spatial filters and filters in the frequency domain. However, spatial filters offer considerably more versatility because, as you will see later, they can be used also for nonlinear filtering, something we cannot do in the frequency domain.

2.4.1 The Mechanics of Spatial Filtering

In Fig. 2.1, we explained briefly that a spatial filter consists of (1) a *neighborhood*, (typically a small rectangle), and (2) a *predefined operation* that is performed on the image pixels encompassed by the neighborhood. Filtering creates a new pixel with coordinates equal to the coordinates of the center of the neighborhood, and whose value is the result of the filtering operation. A processed (filtered) image is generated as the center of the filter visits each pixel in the input image. If the operation performed on the image pixels is linear, then the filter is called a *linear spatial filter*. Otherwise, the filter is *nonlinear*. We focus attention first on linear filters and then illustrate some simple nonlinear filters.

Figure 2.19 illustrates the mechanics of linear spatial filtering using a 3×3 neighborhood. At any point (x, y) in the image, the response, $g(x, y)$, of the filter is the sum of products of the filter coefficients and the image pixels

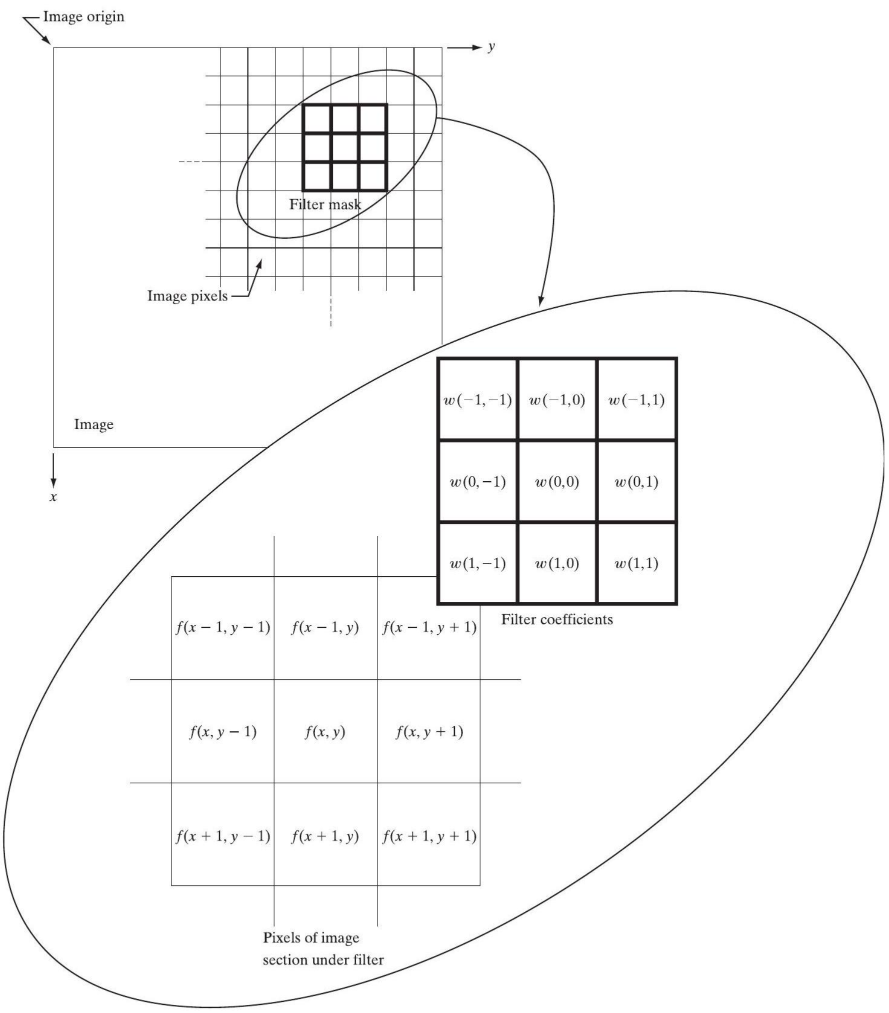


Figure 2.19: The mechanics of linear spatial filtering using a 3×3 filter mask. The form chosen to denote the coordinates of the filter mask coefficients simplifies writing expressions for linear filtering.

encompassed by the filter:

$$\begin{aligned} g(x,y) = {} & w(-1,-1)f(x-1,y-1) + w(-1,0)f(x-1,y) + \dots \\ & + w(0,0)f(x,y) + \dots + w(1,1)f(x+1,y+1) \end{aligned}$$

Observe that the center coefficient of the filter, $w(0,0)$, aligns with the pixel at location (x,y). For a mask of size $m \times n$, we assume that $m = 2a+1$ and $n = 2b+1$, where a and b are positive integers. This means that our focus in the following discussion is on filters of odd size, with the smallest being of size 3×3. In general, linear spatial filtering of an image of size $M \times N$ with a filter of size $m \times n$ is given by the expression:

$$g(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x+s, y+t)$$

where x and y are varied so that each pixel in w visits every pixel in f.

2.4.2 Spatial Correlation and Convolution

There are two closely related concepts that must be understood clearly when performing linear spatial filtering. One is *correlation* and the other is *convolution*. Correlation is the process of moving a filter mask over the image and computing the sum of products at each location, exactly as explained in the previous section. The mechanics of convolution are the same, except that the filter is first rotated by 180°. The best way to explain the differences between the two concepts is by example. We begin with a 1-D illustration.

Figure 2.20(a) shows a 1-D function, f, and a filter, w , and Fig. 2.20(b) shows the starting position to perform correlation. The first thing we note is that there are parts of the functions that do not overlap. The solution to this problem is to pad f with enough 0 s on either side to allow each pixel in w to visit every pixel in f. If the filter is of size m, we need $m-10$ s on either side of f. Figure 2.20(c) shows a properly padded function. The first value of correlation is the sum of products of f and w for the initial position shown in Fig. 2.20(c) (the sum of products is 0). This corresponds to a displacement $x = 0$. To obtain the second value of correlation, we shift w one pixel location to the right (a displacement of $x = 1$) and compute the sum of products. The result again is 0 . In fact, the first nonzero result is when $x = 3$, in which case the 8 in w overlaps the 1 in f and the result of correlation is 8. Proceeding in this manner, we obtain the full correlation result in Fig. 2.20(g). Note that it took 12 values of x (i.e., $x = 0, 1, 2, \ldots, 11$) to fully slide w past f so that each pixel in w visited every pixel in f. Often, we like to work with correlation arrays that are the same size as f, in which case we crop the full correlation to the size of the original function, as Fig. 2.20(h) shows.

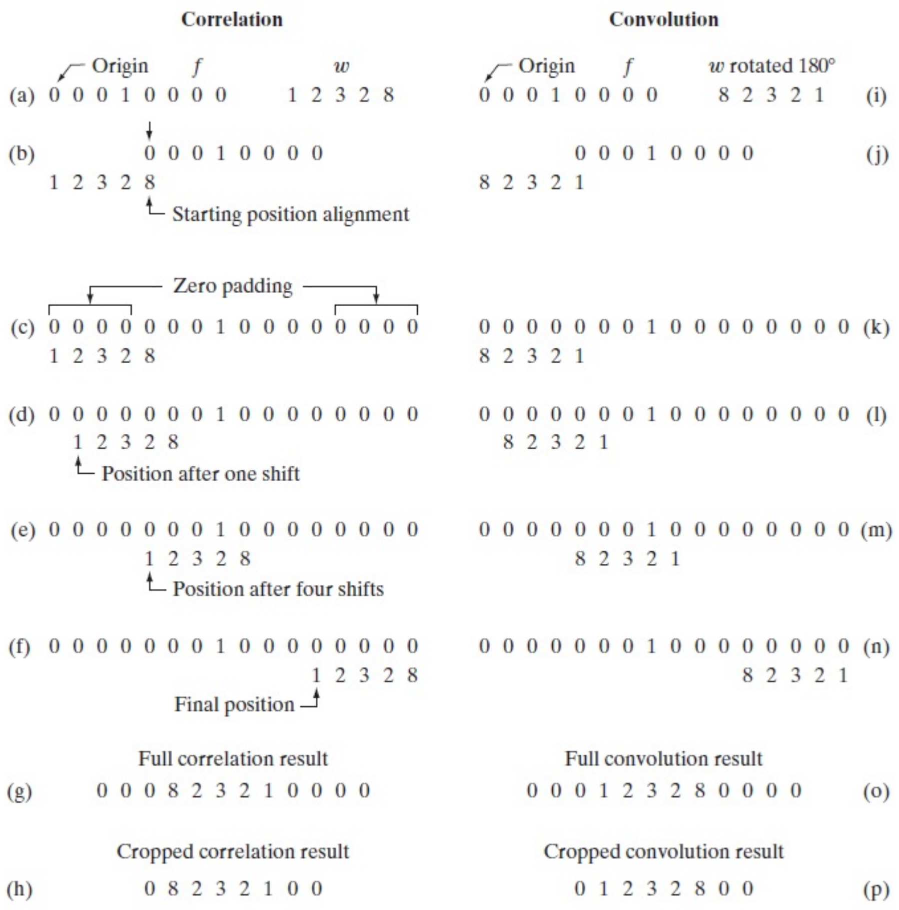

Correlation
Convolution
Origin f w
(a) 0 0 0 1 0 0 0 0 1 2 3 2 8
Origin f w rotated 180°
0 0 0 1 0 0 0 0 8 2 3 2 1 (i)
(b) 0 0 0 1 0 0 0 0
1 2 3 2 8
Starting position alignment
0 0 0 1 0 0 0 0 (j)
8 2 3 2 1
Zero padding
(c) 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
1 2 3 2 8
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 (k)
8 2 3 2 1
(d) 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
1 2 3 2 8
Position after one shift
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 (l)
8 2 3 2 1
(e) 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
1 2 3 2 8
Position after four shifts
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 (m)
8 2 3 2 1
(f) 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
1 2 3 2 8
Final position
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 (n)
8 2 3 2 1
Full correlation result
(g) 0 0 0 8 2 3 2 1 0 0 0 0
Full convolution result
0 0 0 1 2 3 2 8 0 0 0 0 (o)
Cropped correlation result
(h) 0 8 2 3 2 1 0 0
Cropped convolution result
0 1 2 3 2 8 0 0 (p)

Figure 2.20: Illustration of 1-D correlation and convolution of a filter with a discrete unit impulse. Note that correlation and convolution are functions of *displacement*.

There are two important points to note from the discussion in the preceding paragraph. First, correlation is a function of *displacement* of the filter. In other words, the first value of correlation corresponds to zero displacement of the filter, the second corresponds to one unit displacement, and so on. The second thing to notice is that correlating a filter w with a function that contains all 0 s and a single 1 yields a result that is a copy of w, but *rotated* by 180°. We call a function that contains a single 1 with the rest being 0 s a discrete unit impulse. So we conclude that correlation of a function with a discrete unit impulse yields a rotated version of the function at the location of the impulse.

The concept of convolution is a cornerstone of linear system theory. We saw in the previous paragraph that correlation yields a copy of the function also, but rotated by 180°. Therefore, if we *pre-rotate* the filter and perform the same sliding sum of products operation, we should be able to obtain the desired result. As the right column in Fig. 2.20 shows, this indeed is the case. Thus, we see that to perform convolution all we do is rotate one function by 180° and perform the same operations as in correlation. As it turns out, it makes no difference which of the two functions we rotate.

The preceding concepts extend easily to images, as Fig. 2.21 shows. For a filter of size $m \times n$, we pad the image with a minimum of $m-1$ rows of 0s at the top and bottom and $n-1$ columns of 0s on the left and right. In this case, m and n are equal to 3, so we pad f with two rows of 0s above and below and two columns of 0s to the left and right, as Fig. 2.21(b) shows. Figure 2.21(c) shows the initial position of the filter mask for performing correlation, and Fig. 2.21(d) shows the full correlation result. Figure 2.21(e) shows the corresponding cropped result. Note again that the result is rotated by 180°. For convolution, we pre-rotate the mask as before and repeat the sliding sum of products just explained. Figures 2.21 (f) through (h) show the result. You see again that convolution of a function with an impulse copies the function

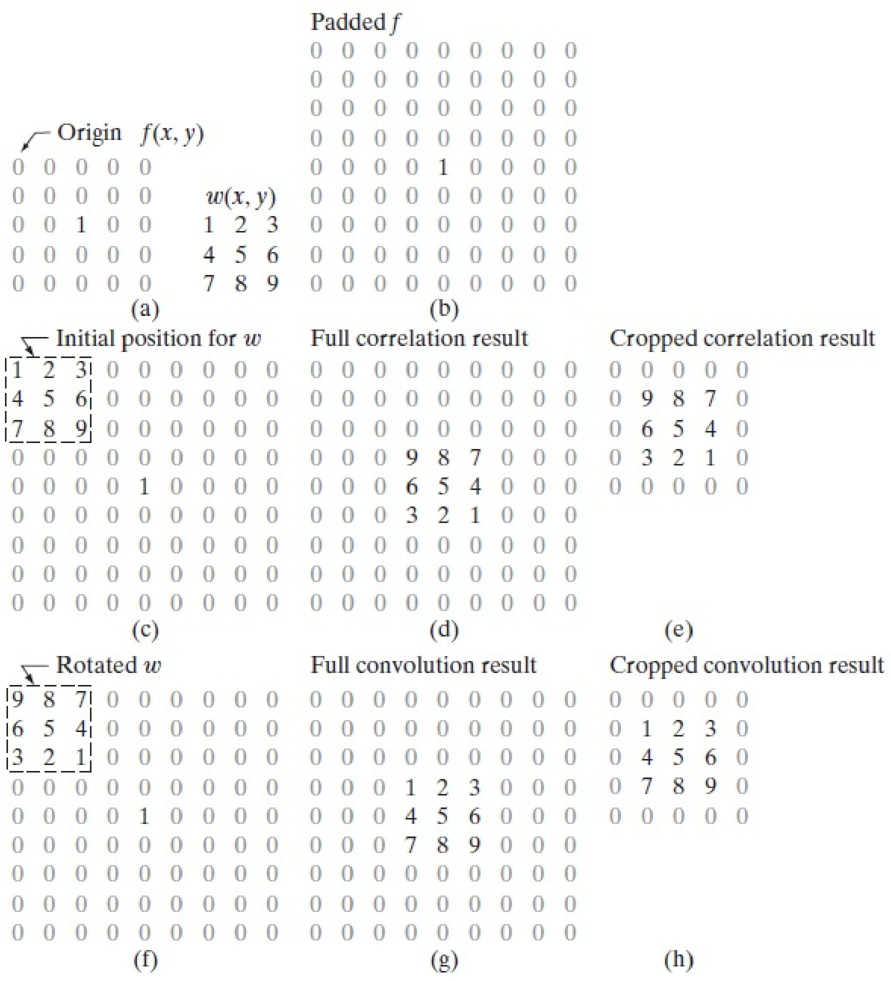


Figure 2.21: Correlation (middle row) and convolution (last row) of a 2-D filter with a 2-D discrete, unit impulse. The 0s are shown in gray to simplify visual analysis.

at the location of the impulse. It should be clear that, if the filter mask is symmetric, correlation and convolution yield the same result.

If, instead of containing a single 1, image f in Fig. 2.21 had contained a region identically equal to w, the value of the correlation function (after normalization) would have been maximum when w was centered on that region of f.

Summarizing the preceding discussion in equation form, we have that the correlation of a filter $w(x,y)$ of size $m \times n$ with an image $f(x,y)$, denoted as $w(x,y) \star f(x,y)$, is given by the equation listed at the end of the last section, which we repeat here for convenience:

$$w(x,y) \star f(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x+s, y+t) \tag{2.18}$$

This equation is evaluated for all values of the displacement variables x and y so that all elements of w visit every pixel in f, where we assume that f has been padded appropriately. As explained earlier, $a = (m-1)/2, b = (n-1)/2$, and we assume for notational convenience that m and n are odd integers.

In a similar manner, the convolution of $w(x,y)$ and $f(x,y)$, denoted by $w(x,y) \star f(x,y)$, is given by the expression

$$w(x,y) \star f(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x-s, y-t) \tag{2.19}$$

where the minus signs on the right flip f (i.e., rotate it by 180°). Flipping and shifting f instead of w is done for notational simplicity and also to follow convention. The result is the same. As with correlation, this equation is evaluated for all values of the displacement variables x and y so that every element of w visits every pixel in f, which we assume has been padded appropriately. You should expand Eq. (2.19) for a 3×3 mask and convince yourself that the result using this equation is identical to the example in Fig. 2.21. In practice, we frequently work with an algorithm that implements Eq. (2.18). If we want to perform correlation, we input w into the algorithm; for convolution, we input w rotated by 180°. The reverse is true if an algorithm that implements Eq. (2.19) is available instead.

2.4.3 Vector Representation of Linear Filtering

When interest lies in the characteristic response, R, of a mask either for correlation or convolution, it is convenient sometimes to write the sum of products as

$$R = w_1 z_1 + w_2 z_2 + \ldots + w_{mn} z_{mn} = \sum_{k=1}^{mn} w_k z_k = \mathbf{w}^T \mathbf{z} \tag{2.20}$$

where the w s are the coefficients of an $m \times n$ filter and the z s are the corresponding image intensities encompassed by the filter. If we are interested in using Eq. (2.20) for correlation, we use the mask as given. To use the same equation for convolution, we simply rotate the mask by 180°, as explained in the last section. It is implied that Eq. (2.20) holds for a particular pair of coordinates (x, y). You will see in the next section why this notation is convenient for explaining the characteristics of a given linear filter.

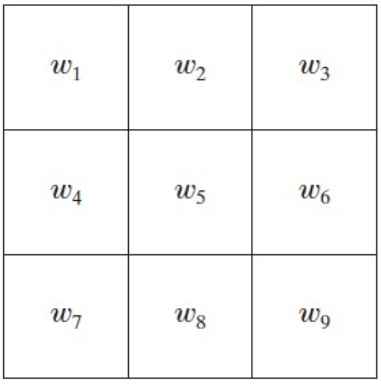


Figure 2.22: Another representation of a general 3×3 filter mask.

As an example, Fig. 2.22 shows a general 3×3 mask with coefficients labeled as above. In this case, Eq. (2.20) becomes

$$R = w_1 z_1 + w_2 z_2 + \ldots + w_9 z_9 = \sum_{k=1}^{9} w_k z_k = \mathbf{w}^T \mathbf{z} \tag{2.21}$$

where **w** and **z** are 9-dimensional vectors formed from the coefficients of the mask and the image intensities encompassed by the mask, respectively.

2.4.4 Generating Spatial Filter Masks

Generating an $m \times n$ *linear* spatial filter requires that we specify mn mask coefficients. In turn, these coefficients are selected based on what the filter is supposed to do, keeping in mind that all we can do with linear filtering is to implement a sum of products. For example, suppose that we want to replace the pixels in an image by the average intensity of a 3×3 neighborhood centered on those pixels. The average value at any location (x, y) in the image is the sum of the nine intensity values in the 3×3 neighborhood centered on (x, y) divided by 9 . Letting $z_i, i = 1, 2, \ldots, 9$, denote these intensities, the average is

$$R = \frac{1}{9}\sum_{i=1}^{9} z_i$$

But this is the same as Eq. (2.21) with coefficient values $w_i = 1/9$. In other words, a linear filtering operation with a 3×3 mask whose coefficients are $1/9$ implements the desired averaging. As we discuss in the next section, this operation results in image smoothing. We discuss in the following sections a number of other filter masks based on this basic approach.

In some applications, we have a continuous function of two variables, and the objective is to obtain a spatial filter mask based on that function. For example, a Gaussian function of two variables has the basic form

$$h(x, y) = e^{-\frac{x^2+y^2}{2\sigma^2}}$$

where σ is the standard deviation and, as usual, we assume that coordinates x and y are integers. To generate, say, a 3×3 filter mask from this function, we sample it about its center. Thus, $w_1 = h(-1, -1), w_2 = h(-1, 0), \ldots$,

$w_9 = h(1,1)$. An $m \times n$ filter mask is generated in a similar manner. Recall that a 2-D Gaussian function has a bell shape, and that the standard deviation controls the "tightness" of the bell.

Generating a *nonlinear* filter requires that we specify the size of a neighborhood and the operation(s) to be performed on the image pixels contained in the neighborhood. For example, recalling that the max operation is nonlinear, a 5×5 max filter centered at an arbitrary point (x,y) of an image obtains the maximum intensity value of the 25 pixels and assigns that value to location (x,y) in the processed image. Nonlinear filters are quite powerful, and in some applications can perform functions that are beyond the capabilities of linear filters.

2.5 Smoothing Spatial Filters

Smoothing filters are used for blurring and for noise reduction. Blurring is used in preprocessing tasks, such as removal of small details from an image prior to (large) object extraction, and bridging of small gaps in lines or curves. Noise reduction can be accomplished by blurring with a linear filter and also by nonlinear filtering.

2.5.1 Smoothing Linear Filters

The output (response) of a smoothing, linear spatial filter is simply the average of the pixels contained in the neighborhood of the filter mask. These filters sometimes are called *averaging filters*. As mentioned in the previous section, they also are referred to a *lowpass filters*.

The idea behind smoothing filters is straightforward. By replacing the value of every pixel in an image by the average of the intensity levels in the neighborhood defined by the filter mask, this process results in an image with reduced “sharp” transitions in intensities. Because random noise typically consists of sharp transitions in intensity levels, the most obvious application of smoothing is noise reduction. However, edges (which almost always are desirable features of an image) also are characterized by sharp intensity transitions, so averaging filters have the undesirable side effect that they blur edges.Another application of this type of process includes the smoothing of false contours that result from using an insufficient number of intensity levels. A major use of averaging filters is in the reduction of “irrelevant” detail in an image. By “irrelevant” we mean pixel regions that are small with respect to the size of the filter mask.

Figure 2.23 shows two 3×3 smoothing filters. Use of the first filter yields the standard average of the pixels under the mask. This can best be seen by substituting the coefficients of the mask into Eq. (2.21):

$$R = \frac{1}{9} \sum_{i=1}^{9} z_i$$

which is the average of the intensity levels of the pixels in the 3×3 neighborhood defined by the mask, as discussed earlier. Note that, instead of being $1/9$, the coefficients of the filter are all 1 s . The idea here is that it is computationally more efficient to have coefficients valued 1. At the end of the filtering process the entire image is divided by 9 . An $m \times n$ mask would have a normalizing constant equal to $1/mn$. A spatial averaging filter in which all coefficients are equal sometimes is called a *box filter*.

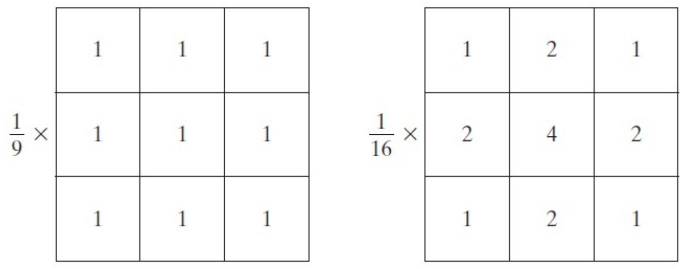


Figure 2.23: Two 3×3 smoothing (averaging) filter masks.The constant multiplier in front of each mask is equal to 1 divided by the sum of the values of its coefficients, as is required to compute an average.

The second mask in Fig. 2.23 is a little more interesting. This mask yields a so called *weighted average*, terminology used to indicate that pixels are multiplied by different coefficients, thus giving more importance (weight) to some pixels at the expense of others. In the mask shown in Fig. 2.23(b) the pixel at the center of the mask is multiplied by a higher value than any other, thus giving this pixel more importance in the calculation of the average. The other pixels are inversely weighted as a function of their distance from the center of the mask. The diagonal terms are further away from the center than the orthogonal neighbors (by a factor of $\sqrt{2}$) and, thus, are weighed less than the immediate neighbors of the center pixel. The basic strategy behind weighing the center point the highest and then reducing the value of the coefficients as a function of increasing distance from the origin is simply an attempt to reduce blurring in the smoothing process. We could have chosen other weights to accomplish the same general objective. However, the sum of all the coefficients in the mask of Fig. 2.23(b) is equal to 16, an attractive feature for computer implementation because it is an integer power of 2 . In practice, it is difficult in general to see differences between images smoothed by using either of the masks in Fig. 2.23, or similar arrangements, because the area spanned by these masks at any one location in an image is so small.

With reference to Eq. (2.183.4-1), the general implementation for filtering an $M \times N$ image with a weighted averaging filter of size $m \times n$ (m and n odd) is given by the expression

$$g(x,y) = \frac{\sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x+s,y+t)}{\sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t)} \tag{2.22}$$

The parameters in this equation are as defined in Eq. (2.183.4-1). As before, it is understood that the complete filtered image is obtained by applying Eq. (2.22) for $x = 0,1,2,\ldots,M-1$ and $y = 0,1,2,\ldots,N-1$. The denominator in Eq. (2.22) is simply the sum of the mask coefficients and, therefore, it is a constant that needs to be computed only once.

2.5.2 Order-Statistic (Nonlinear) Filters

Order-statistic filters are nonlinear spatial filters whose response is based on ordering (ranking) the pixels contained in the image area encompassed by the filter, and then replacing the value of the center pixel with the value determined by the ranking result. The best-known filter in this category is the *median filter*, which, as its name implies, replaces the value of a pixel by the median of the intensity values in the neighborhood of that pixel (the original value of the pixel is included in the computation of the median). Median filters are quite popular because, for certain types of random noise, they provide excellent noise-reduction capabilities, with considerably less blurring than linear smoothing filters of similar size. Median filters are particularly effective in the presence of *impulse noise*, also called *salt-and-pepper noise* because of its appearance as white and black dots superimposed on an image.

The median, ξ, of a set of values is such that half the values in the set are less than or equal to ξ, and half are greater than or equal to ξ. In order to perform median filtering at a point in an image, we first sort the values of the pixel in the neighborhood, determine their median, and assign that value to the corresponding pixel in the filtered image. For example, in a 3×3 neighborhood the median is the 5th largest value, in a 5×5 neighborhood it is the 13th largest value, and so on. When several values in a neighborhood are the same, all equal values are grouped. For example, suppose that a 3×3 neighborhood has values $(10, 20, 20, 20, 15, 20, 20, 25, 100)$. These values are sorted as $(10, 15, 20, 20, 20, 20, 20, 25, 100)$, which results in a median of 20 . Thus, the principal function of median filters is to force points with distinct intensity levels to be more like their neighbors. In fact, isolated clusters of pixels that are light or dark with respect to their neighbors, and whose area is less than $m^2/2$ (one-half the filter area), are eliminated by an $m \times m$ median filter. In this case "eliminated" means forced to the median intensity of the neighbors. Larger clusters are affected considerably less.

Although the median filter is by far the most useful order-statistic filter in image processing, it is by no means the only one. The median represents the 50th percentile of a ranked set of numbers, but recall from basic statistics that ranking lends itself to many other possibilities. For example, using the 100th percentile results in the so-called max filter, which is useful for finding the brightest points in an image. The response of a 3×3 *max filter* is given by $R = \max\{z_k \mid k = 1, 2, \ldots, 9\}$. The $0th$ percentile filter is the *min filter*, used for the opposite purpose.

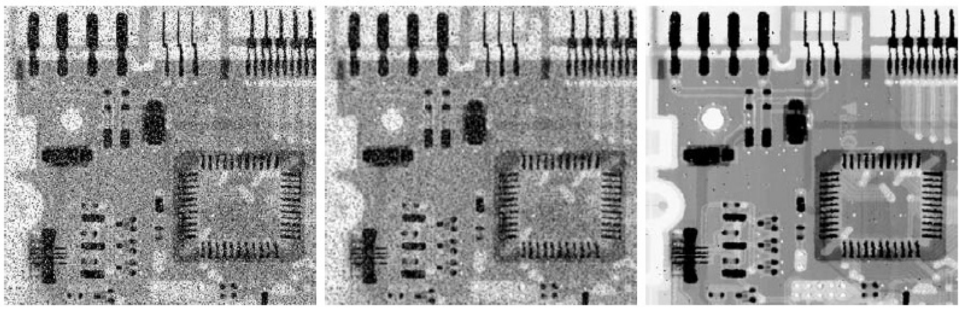


Figure 2.24: (a) X-ray image of circuit board corrupted by salt-and-pepper noise. (b) Noise reduction with a 3×3 averaging mask. (c) Noise reduction with a 3×3 median filter.

Figure 2.24(a) shows an X-ray image of a circuit board heavily corrupted by salt-and-pepper noise.To illustrate the point about the superiority of median filtering over average filtering in situations such as this, we show in Fig. 2.24(b) the result of processing the noisy image with a 3×3 neighborhood averaging mask, and in Fig. 2.24(c) the result of using a 3×3 median filter. The averaging filter blurred the image and its noise reduction performance was poor. The superiority in all respects of median over average filtering in this case is quite evident. In general, median filtering is much better suited than averaging for the removal of salt-and-pepper noise.

2.6 Sharpening Spatial Filters

The principal objective of sharpening is to highlight transitions in intensity. Uses of image sharpening vary and include applications ranging from electronic printing and medical imaging to industrial inspection and autonomous guidance in military systems. In the last section, we saw that image blurring could be accomplished in the spatial domain by pixel averaging in a neighborhood. Because averaging is analogous to integration, it is logical to conclude that sharpening can be accomplished by spatial differentiation. This section deals with various ways of defining and implementing operators for sharpening by digital differentiation. Fundamentally, the strength of the response of a derivative operator is proportional to the degree of intensity discontinuity of the image at the point at which the operator is applied. Thus, image differentiation enhances edges and other discontinuities (such as noise) and deemphasizes areas with slowly varying intensities.

2.6.1 Foundation

In the two sections that follow, we consider in some detail sharpening filters that are based on first- and second-order derivatives, respectively. Before proceeding with that discussion, however, we stop to look at some of the fundamental properties of these derivatives in a digital context. To simplify the explanation, we focus attention initially on one-dimensional derivatives. In particular, we are interested in the behavior of these derivatives in areas of constant intensity, at the onset and end of discontinuities (step and ramp discontinuities), and along intensity ramps. These types of discontinuities can be used to model noise points, lines, and edges in an image. The behavior of derivatives during transitions into and out of these image features also is of interest.

The derivatives of a digital function are defined in terms of differences. There are various ways to define these differences. However, we require that any definition we use for a *first derivative* (1) must be zero in areas of constant intensity; (2) must be nonzero at the onset of an intensity step or ramp; and (3) must be nonzero along ramps. Similarly, any definition of a *second derivative* (1) must be zero in constant areas; (2) must be nonzero at the onset and end of an intensity step or ramp; and (3) must be zero along ramps of constant slope. Because we are dealing with digital quantities whose values are finite, the maximum possible intensity change also is finite, and the shortest distance over which that change can occur is between adjacent pixels.

A basic definition of the first-order derivative of a one-dimensional function $f(x)$ is the difference

$$\frac{\partial f}{\partial x} = f(x+1) - f(x) \tag{2.23}$$

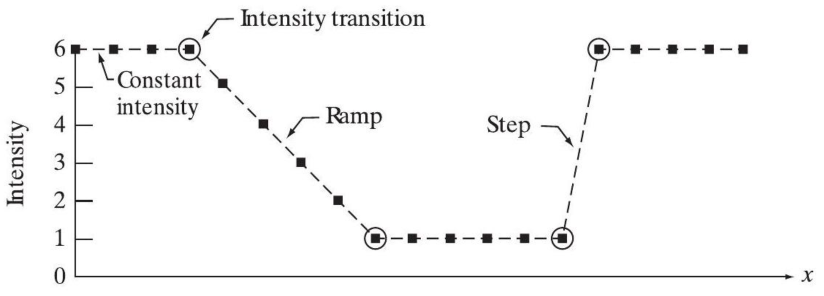
We used a partial derivative here in order to keep the notation the same as when we consider an image function of two variables, $f(x,y)$, at which time we will be dealing with partial derivatives along the two spatial axes. Use of a partial derivative in the present discussion does not affect in any way the nature of what we are trying to accomplish. Clearly, $\partial f/\partial x = df/dx$ when there is only one variable in the function; the same is true for the second derivative.

We define the second-order derivative of $f(x)$ as the difference

$$\frac{\partial^2 f}{\partial x^2} = f(x+1) + f(x-1) - 2f(x) \tag{2.24}$$

It is easily verified that these two definitions satisfy the conditions stated above. To illustrate this, and to examine the similarities and differences between first- and second-order derivatives of a digital function, consider the example in Fig. 2.25.

Figure 2.25(b) (center of the figure) shows a section of a scan line (intensity profile). The values inside the small squares are the intensity values in the scan line, which are plotted as black dots above it in Fig. 2.25(a). The



Scan line	6	6	6	6	5	4	3	2	1	1	1	1	1	1	6	6	6	6	6	→ x
1st derivative		0	0	−1	−1	−1	−1	−1	0	0	0	0	0	5	0	0	0	0		
2nd derivative		0	0	−1	0	0	0	0	1	0	0	0	0	5	−5	0	0	0		

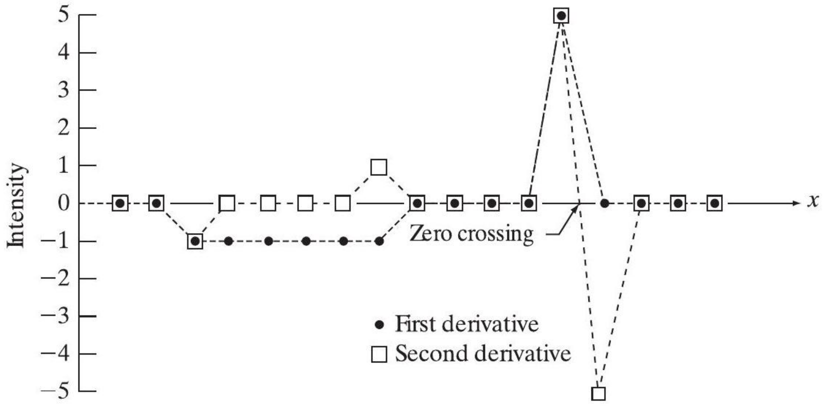


Figure 2.25: Illustration of the first and second derivatives of a 1-D digital function representing a section of a horizontal intensity profile from an image. In (a) and (c) data points are joined by dashed lines as a visualization aid.

dashed line connecting the dots is included to aid visualization. As the figure shows, the scan line contains an intensity ramp, three sections of constant intensity, and an intensity step. The circles indicate the onset or end of intensity transitions. The first- and second-order derivatives computed using the two preceding definitions are included below the scan line in Fig. 2.25(b), and are plotted in Fig. 2.25(c). When computing the first derivative at a location x, we subtract the value of the function at that location from the next point. So this is a "look-ahead" operation. Similarly, to compute the second derivative at *x*, we use the previous and the next points in the computation. To avoid a situation in which the previous or next points are outside the range of the scan line, we show derivative computations in Fig. 2.25 from the second through the penultimate points in the sequence.

Let us consider the properties of the first and second derivatives as we traverse the profile from left to right. First, we encounter an area of constant intensity and, as Figs. 2.25(b) and (c) show, both derivatives are zero there, so condition (1) is satisfied for both. Next, we encounter an intensity ramp followed by a step, and we note that the first-order derivative is nonzero at the onset of the ramp and the step; similarly, the second derivative is nonzero at the onset and end of both the ramp and the step; therefore, property (2) is satisfied for both derivatives. Finally, we see that property (3) is satisfied also for both derivatives because the first derivative is nonzero and the second is zero along the ramp. Note that the sign of the second derivative changes at the onset and end of a step or ramp. In fact, we see in Fig. 2.25(c) that in a step transition a line joining these two values crosses the horizontal axis midway between the two extremes. This zero crossing property is quite useful for locating edges.

Edges in digital images often are ramp-like transitions in intensity, in which case the first derivative of the image would result in thick edges because the derivative is nonzero along a ramp. On the other hand, the second derivative would produce a double edge one pixel thick, separated by zeros. From this, we conclude that the second derivative enhances fine detail much better than the first derivative, a property that is ideally suited for sharpening images. Also, as you will learn later in this section, second derivatives are much easier to implement than first derivatives, so we focus our attention initially on second derivatives.

2.6.2 Using the Second Derivative for Image Sharpening-The Laplacian

In this section we consider the implementation of 2-D, second-order derivatives and their use for image sharpening. The approach basically consists of defining a discrete formulation of the second-order derivative and then constructing a filter mask based on that formulation. We are interested in *isotropic* filters, whose response is independent of the direction of the discontinuities in the image to which the filter is applied. In other words, isotropic filters are *rotation invariant*, in the sense that rotating the image and then applying the filter gives the same result as applying the filter to the image first and then rotating the result.

It can be shown that the simplest isotropic derivative operator is the Laplacian, which, for a function (image) $f(x,y)$ of two variables, is defined as

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \tag{2.25}$$

Because derivatives of any order are linear operations, the Laplacian is a linear operator. To express this equation in discrete form, we use the definition in Eq. (2.24), keeping in mind that we have to carry a second variable. In the x-direction, we have

$$\frac{\partial^2 f}{\partial x^2} = f(x+1,y) + f(x-1,y) - 2f(x,y) \tag{2.26}$$

and, similarly, in the y-direction we have

$$\frac{\partial^2 f}{\partial y^2} = f(x,y+1) + f(x,y-1) - 2f(x,y) \tag{2.27}$$

Therefore, it follows from the preceding three equations that the discrete Laplacian of two variables is

$$\nabla^2 f(x,y) = f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1) - 4f(x,y) \tag{2.28}$$

This equation can be implemented using the filter mask in Fig. 2.26 (a), which gives an isotropic result for rotations in increments of 90°. The mechanics of implementation are for linear smoothing filters. We simply are using different coefficients here.

The diagonal directions can be incorporated in the definition of the digital Laplacian by adding two more terms to Eq. (2.28), one for each of the two diagonal directions. The form of each new term is the same as either Eq. (2.26) or (2.27), but the coordinates are along the diagonals. Because each diagonal term also contains a $-2f(x,y)$ term, the total subtracted from the difference terms now would be $-8f(x,y)$. Figure 2.26(b) shows the filter mask used to implement this new definition. This mask yields isotropic results in increments of 45°. You are likely to see in practice the Laplacian masks in Figs. 2.26(c) and (d). They are obtained from definitions of the second derivatives that are the negatives of the ones we used in Eqs. (2.26) and (2.27). As such, they yield equivalent results, but the difference in sign must be kept in mind when combining (by addition or subtraction) a Laplacian-filtered image with another image.

Because the Laplacian is a derivative operator, its use highlights intensity discontinuities in an image and deemphasizes regions with slowly varying intensity levels. This will tend to produce images that have grayish edge lines and other discontinuities, all superimposed on a dark, featureless background. Background features can be "recovered" while still preserving the sharpening effect of the Laplacian simply by adding the Laplacian image to the original. As noted in the previous paragraph, it is important to keep in mind which definition of the Laplacian is used. If the definition used has a negative center coefficient, then we *subtract*, rather than add, the Laplacian image to obtain

0	1	0
1	−4	1
0	1	0

1	1	1
1	−8	1
1	1	1

0	−1	0
−1	4	−1
0	−1	0

−1	−1	−1
−1	8	−1
−1	−1	−1

Figure 2.26: (a) Filter mask used to implement Eq. 2.28 (b) Mask used to implement an extension of this equation that includes the diagonal terms. (c) and (d) Two other implementations of the Laplacian found frequently in practice.

a sharpened result. Thus, the basic way in which we use the Laplacian for image sharpening is

$$g(x,y) = f(x,y) + c\left[\nabla^2 f(x,y)\right] \tag{2.29}$$

where $f(x,y)$ and $g(x,y)$ are the input and sharpened images, respectively. The constant is $c = -1$ if the Laplacian filters in Fig. 3.37(a) or (b) are used, and $c = 1$ if either of the other two filters is used.

2.6.3 Using First-Order Derivatives for (Nonlinear) Image Sharpening-The Gradient

First derivatives in image processing are implemented using the magnitude of the gradient. For a function $f(x,y)$, the gradient of f at coordinates (x,y) is defined as the two-dimensional column *vector*

$$\nabla f \equiv \text{grad}(f) \equiv \begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} \tag{2.30}$$

This vector has the important geometrical property that it points in the direction of the greatest rate of change of f at location (x,y).

The *magnitude (length)* of vector ∇f, denoted as $M(x,y)$, where

$$M(x,y) = \text{mag}(\nabla f) = \sqrt{g_x^2 + g_y^2} \tag{2.31}$$

is the *value* at (x,y) of the rate of change in the direction of the gradient vector. Note that $M(x,y)$ is an image of the same size as the original, created when x and y are allowed to vary over all pixel locations in f. It is common practice to refer to this image as the *gradient image* (or simply as the *gradient* when the meaning is clear).

Because the components of the gradient vector are derivatives, they are linear operators. However, the magnitude of this vector is not because of the squaring and square root operations. On the other hand, the partial derivatives in Eq. (2.30) are not rotation invariant (isotropic), but the magnitude of the gradient vector is. In some implementations, it is more suitable computationally to approximate the squares and square root operations by absolute values:

$$M(x,y) \approx |g_x| + |g_y| \tag{2.32}$$

This expression still preserves the relative changes in intensity, but the isotropic property is lost in general. However, as in the case of the Laplacian, the isotropic properties of the discrete gradient defined in the following paragraph are preserved only for a limited number of rotational increments that depend on the filter masks used to approximate the derivatives. As it turns out, the most popular masks used to approximate the gradient are isotropic at multiples of 90°. These results are independent of whether we use Eq. (2.31) or (2.32), so nothing of significance is lost in using the latter equation if we choose to do so.

As in the case of the Laplacian, we now define discrete approximations to the preceding equations and from there formulate the appropriate filter masks. In order to simplify the discussion that follows, we will use the notation in Fig. 2.27(a) to denote the intensities of image points in a 3×3 region. For example, the center point, z_5, denotes $f(x,y)$ at an arbitrary location, $(x,y); z_1$ denotes $f(x-1,y-1)$; and so on. The simplest approximations to a first-order derivative that satisfy the conditions stated in that section are $g_x = (z_8 - z_5)$ and $g_y = (z_6 - z_5)$. Two other definitions proposed by Roberts in the early development of digital image processing use cross differences:

$$g_x = (z_9 - z_5) \quad \text{and} \quad g_y = (z_8 - z_6) \tag{2.33}$$

If we use Eqs. (2.31) and (2.33), we compute the gradient image as

$$M(x,y) = \left[(z_9 - z_5)^2 + (z_8 - z_6)^2\right]^{1/2} \tag{2.34}$$

If we use Eqs.(2.32) and (2.33), then

$$M(x,y) \approx |z_9 - z_5| + |z_8 - z_6| \tag{2.35}$$

z_1	z_2	z_3
z_4	z_5	z_6
z_7	z_8	z_9

−1	0
0	1

0	−1
1	0

−1	−2	−1
0	0	0
1	2	1

−1	0	1
−2	0	2
−1	0	1

Figure 2.27: A 3×3 region of an image (the z s are intensity values). (b)-(c) Roberts cross gradient operators. (d)-(e) Sobel operators. All the mask coefficients sum to zero, as expected of a derivative operator.

where it is understood that x and y vary over the dimensions of the image in the manner described earlier. The partial derivative terms needed in equation (2.33) can be implemented using the two linear filter masks in Figs. 2.27(b) and (c). These masks are referred to as the *Roberts cross-gradient operators*.

Masks of even sizes are awkward to implement because they do not have a center of symmetry. The smallest filter masks in which we are interested are of size 3×3. Approximations to g_x and g_y using a 3×3 neighborhood centered on z_5 are as follows:

$$g_x = \frac{\partial f}{\partial x} = (z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3) \tag{2.36}$$

and

$$g_y = \frac{\partial f}{\partial y} = (z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7) \tag{2.37}$$

These equations can be implemented using the masks in Figs. 2.27(d) and (e). The difference between the third and first rows of the 3×3 image region implemented by the mask in Fig. 2.27(d) approximates the partial derivative in the x-direction, and the difference between the third and first columns in the other mask approximates the derivative in the y-direction. After computing the partial derivatives with these masks, we obtain the magnitude of the gradient as before. For example, substituting g_x and g_y into Eq. (2.32) yields

$$\begin{aligned} M(x,y) \approx & \mid (z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3) \mid \\ & + \left|(z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7)\right| \end{aligned} \tag{2.38}$$

The masks in Figs. 2.27(d) and (e) are called the *Sobel operators*. The idea behind using a weight value of 2 in the center coefficient is to achieve some smoothing by giving more importance to the center point. Note that the coefficients in all the masks shown in Fig. 2.27 sum to 0, indicating that they would give a response of 0 in an area of constant intensity, as is expected of a derivative operator.

As mentioned earlier, the computations of g_x and g_y are linear operations because they involve derivatives and, therefore, can be implemented as a sum of products using the spatial masks in Fig. 2.27. The nonlinear aspect of sharpening with the gradient is the computation of $M(x,y)$ involving squaring and square roots, or the use of absolute values, all of which are nonlinear operations. These operations are performed *after* the linear process that yields g_x and g_y.

CHAPTER 3

IMAGE COMPRESSION

Image compression, the art and science of reducing the amount of data required to represent an image, is one of the most useful and commercially successful technologies in the field of digital image processing. The number of images that are compressed and decompressed daily is staggering, and the compressions and decompressions themselves are virtually invisible to the user. Anyone who owns a digital camera, surfs the web, or watches the latest Hollywood movies on *Digital Video Disks* (DVDs) benefits from the algorithms and standards discussed in this chapter.

To better understand the need for compact image representations, consider the amount of data required to represent a two-hour *standard definition* (SD) *television* movie using $720 \times 480 \times 24$ bit pixel arrays. A digital movie (or video) is a sequence of video frames in which each frame is a full-color still image. Because video players must display the frames sequentially at rates near 30 fps (frames per second), SD digital video data must be accessed at

$$30\frac{\text{frames}}{\text{sec}} \times (720 \times 480)\frac{\text{pixels}}{\text{frame}} \times 3\frac{\text{bytes}}{\text{pixel}} = 31,104,000 \text{ bytes }/\text{sec}$$

and a two-hour movie consists of

$$31,104,000\frac{\text{bytes}}{\text{sec}} \times \left(60^2\right)\frac{\text{sec}}{\text{hr}} \times 2\text{hrs} \cong 2.24 \times 10^{11} \text{ bytes}$$

or 224 GB (gigabytes) of data. Twenty-seven 8.5 GB dual-layer DVDs (assuming conventional 12 cm disks) are needed to store it. To put a two-hour movie on a single DVD, each frame must be compressed-on average-by a factor of 26.3. The compression must be even higher for *high definition* (HD) *television*, where image resolutions reach $1920 \times 1080 \times 24$ bits/image.

Web page images and high-resolution digital camera photos also are compressed routinely to save storage space and reduce transmission time. For example, residential Internet connections deliver data at speeds ranging from 56 Kbps (kilobits per second) via conventional phone lines to more than 12 Mbps (megabits per second) for broadband. The time required to transmit a small $128 \times 128 \times 24$ bit full-color image over this range of speeds is from 7.0 to 0.03 seconds. Compression can reduce transmission time by a factor of 2 to 10 or more. In the same way, the number of uncompressed full-color images that an 8 -megapixel digital camera can store on a 1-GB flash memory card [about forty-one 24 MB (megabyte) images] can be similarly increased. In addition to these applications, image compression plays an important role in many other areas, including televideo conferencing, remote sensing, document and medical imaging, and facsimile transmission (FAX). An increasing number of applications depend on the efficient manipulation, storage, and transmission of binary, gray-scale, and color images.

In this chapter, we introduce the theory and practice of digital image compression. We examine the most frequently used compression techniques and describe the industry standards that make them useful. The material is introductory in nature and applicable to both still image and video applications.

3.1 Fundamentals

The term *data compression* refers to the process of reducing the amount of data required to represent a given quantity of information. In this definition, *data* and *information* are not the same thing; data are the means by which information is conveyed. Because various amounts of data can be used to represent the same amount of information, representations that contain irrelevant or repeated information are said to contain *redundant data*. If we let b and b' denote the number of bits (or information-carrying units) in two representations of the same information, the *relative data redundancy* R of the representation with b bits is

$$R = 1 - \frac{1}{C} \tag{3.1}$$

where C, commonly called the *compression ratio*, is defined as

$$C = \frac{b}{b'} \tag{3.2}$$

If $C = 10$ (sometimes written 10:1), for instance, the larger representation has 10 bits of data for every 1 bit of data in the smaller representation.

The corresponding relative data redundancy of the larger representation is 0.9 ($R = 0.9$), indicating that 90% of its data is redundant.

In the context of digital image compression, b in Eq. (3.2) usually is the number of bits needed to represent an image as a 2-D array of intensity values. The 2-D intensity arrays are the preferred formats for human viewing and interpretation-and the standard by which all other representations are judged. When it comes to compact image representation, however, these formats are far from optimal. Two-dimensional intensity arrays suffer from three principal types of data redundancies that can be identified and exploited:

1. *Coding redundancy.* A code is a system of symbols (letters, numbers, bits, and the like) used to represent a body of information or set of

events. Each piece of information or event is assigned a sequence of code symbols, called a code word. The number of symbols in each code word is its length. The 8 -bit codes that are used to represent the intensities in most 2-D intensity arrays contain more bits than are needed to represent the intensities.

2. *Spatial and temporal redundancy.* Because the pixels of most 2-D intensity arrays are correlated spatially (i.e., each pixel is similar to or dependent on neighboring pixels), information is unnecessarily replicated in the representations of the correlated pixels. In a video sequence, temporally correlated pixels (i.e., those similar to or dependent on pixels in nearby frames) also duplicate information.
3. *Irrelevant information.* Most 2-D intensity arrays contain information that is ignored by the human visual system and/or extraneous to the intended use of the image. It is redundant in the sense that it is not used.

The computer-generated images in Figs. 3.1(a) through (c) exhibit each of these fundamental redundancies. As will be seen in the next three sections, compression is achieved when one or more redundancy is reduced or eliminated.

3.1.1 Coding Redundancy

In Chapter 2, we developed techniques for image enhancement by histogram processing, assuming that the intensity values of an image are random quantities. In this section, we use a similar formulation to introduce optimal information coding.

Assume that a discrete random variable r_k in the interval $[0, L-1]$ is used to represent the intensities of an $M \times N$ image and that each r_k occurs with probability $p_r(r_k)$, which is expressed as:

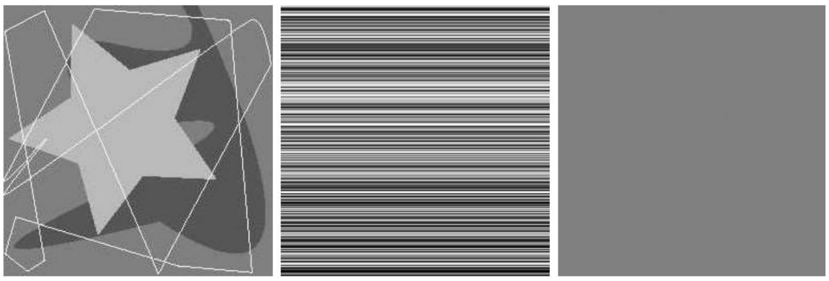


Figure 3.1: Computer generated $256 \times 256 \times 8$ bit images with (a) coding redundancy, (b) spatial redundancy, and (c) irrelevant information. (Each was designed to demonstrate one principal redundancy but may exhibit others as well.)

$$p_r(r_k) = \frac{n_k}{MN} \quad k = 0, 1, 2, \ldots, L-1 \tag{3.3}$$

where L is the number of intensity values, and n_k is the number of times that the k th intensity appears in the image. If the number of bits used to represent each value of r_k is $l(r_k)$, then the average number of bits required to represent each pixel is

$$L_{\text{avg}} = \sum_{k=0}^{L-1} l(r_k)\, p_r(r_k) \tag{3.4}$$

That is, the average length of the code words assigned to the various intensity values is found by summing the products of the number of bits used to represent each intensity and the probability that the intensity occurs. The total number of bits required to represent an $M \times N$ image is MNL_{avg}. If the intensities are represented using a natural m-bit fixed-length code, the right-hand side of Eq. (3.4) reduces to m bits. That is, $L_{\text{avg}} = m$ when m is substituted for $l(r_k)$. The constant m can be taken outside the summation, leaving only the sum of the $p_r(r_k)$ for $0 \leq k \leq L-1$, which, of course, equals 1 .

Example: A simple illustration of variable-length coding

The computer-generated image in Fig.3.1(a) has the intensity distribution shown in the second column of Table 3.1. If a natural 8-bit binary code (denoted as code 1 in Table 3.1) is used to represent its 4 possible intensities, L_{avg} - the average number of bits for code 1 -is 8 bits, because $l_1(r_k) = 8$ bits for all r_k.

Table 3.1: Example of variable-length coding.

r_k	$p_r(r_k)$	Code 1	$l_l(r_k)$	Code 2	$l_2(r_k)$
$r_{87} = 87$	0.25	01010111	8	01	2
$r_{128} = 128$	0.47	10000000	8	1	1
$r_{186} = 186$	0.25	11000100	8	000	3
$r_{255} = 255$	0.03	11111111	8	001	3
r_k for $k \neq 87, 128, 186, 255$	0	-	8	-	0

On the other hand, if the scheme designated as code 2 in Table 3.1 is used, the average length of the encoded pixels is, in accordance with Eq. (3.4),

$$L_{\text{avg}} = 0.25(2) + 0.47(1) + 0.25(3) + 0.03(3) = 1.81 \text{ bits}$$

The total number of bits needed to represent the entire image is $MNL_{\text{avg}} = 256 \times 256 \times 1.81$ or 118,621. From Eqs. (3.2) and (3.1), the resulting compression and corresponding relative redundancy are

$$C = \frac{256 \times 256 \times 8}{118,621} = \frac{8}{1.81} \approx 4.42$$

and

$$R = 1 - \frac{1}{4.42} = 0.774$$

respectively. Thus 77.4% of the data in the original 8-bit 2-D intensity array is redundant.

The compression achieved by code 2 results from assigning fewer bits to the more probable intensity values than to the less probable ones. In the resulting *variable-length code*, r_{128}-the image's most probable intensity-is assigned the 1 -bit code word 1 [of length $l_2(r_{128}) = 1$], while r_{255}-its least probable occurring intensity - is assigned the 3 -bit code word 001 [of length $l_2(r_{255}) = 3$]. Note that the best fixed-length code that can be assigned to the intensities of the image in Fig. 3.1(a) is the natural 2 -bit counting sequence $\{00, 01, 10, 11\}$, but the resulting compression is only $8/2$ or 4 : 1-about 10% less than the 4.42:1 compression of the variable-length code.

As the preceding example shows, *coding redundancy* is present when the codes assigned to a set of events (such as intensity values) do not take full advantage of the probabilities of the events. Coding redundancy is almost always present when the intensities of an image are represented using a natural binary code. The reason is that most images are composed of objects that have a regular and somewhat predictable morphology (shape) and reflectance, and are sampled so that the objects being depicted are much larger than the picture elements. The natural consequence is that, for most images, certain intensities are more probable than others (that is, the histograms of most images are not uniform). A natural binary encoding assigns the same number of bits to both the most and least probable values, failing to minimize Eq. (3.4) and resulting in coding redundancy.

3.1.2 Spatial and Temporal Redundancy

Consider the computer-generated collection of constant intensity lines in Fig. 3.1(b). In the corresponding 2-D intensity array:

1. All 256 intensities are equally probable. As Fig. 3.2 shows, the histogram of the image is uniform.
2. Because the intensity of each line was selected randomly, its pixels are independent of one another in the vertical direction.
3. Because the pixels along each line are identical, they are maximally correlated (completely dependent on one another) in the horizontal direction.

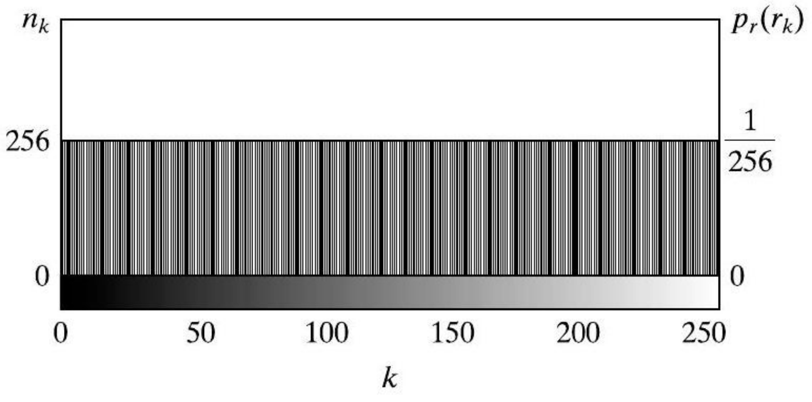


Figure 3.2: The intensity histogram of the image in Fig. 3.1(b).

The first observation tells us that the image in Fig. 3.1(b) — when represented as a conventional 8 -bit intensity array-cannot be compressed by variablelength coding alone. Unlike the image of Fig. 3.1(a), whose histogram was not uniform, a fixed-length 8 -bit code in this case minimizes Eq. (3.4). Observations 2 and 3 reveal a significant spatial redundancy that can be eliminated, for instance, by representing the image in Fig. 3.1(b) as a sequence of *run-length pairs*, where each run-length pair specifies the start of a new intensity and the number of consecutive pixels that have that intensity. A run-length based representation compresses the original 2-D, 8-bit intensity array by $(256 \times 256 \times 8)/[(256+256) \times 8]$ or 128 : 1. Each 256 -pixel line of the original representation is replaced by a single 8 -bit intensity value and length 256 in the run-length representation.

In most images, pixels are correlated spatially (in both x and y) and in time (when the image is part of a video sequence). Because most pixel intensities can be predicted reasonably well from neighboring intensities, the information carried by a single pixel is small. Much of its visual contribution is redundant in the sense that it can be inferred from its neighbors. To reduce the redundancy associated with spatially and temporally correlated pixels, a 2-D intensity array must be transformed into a more efficient but usually "non-

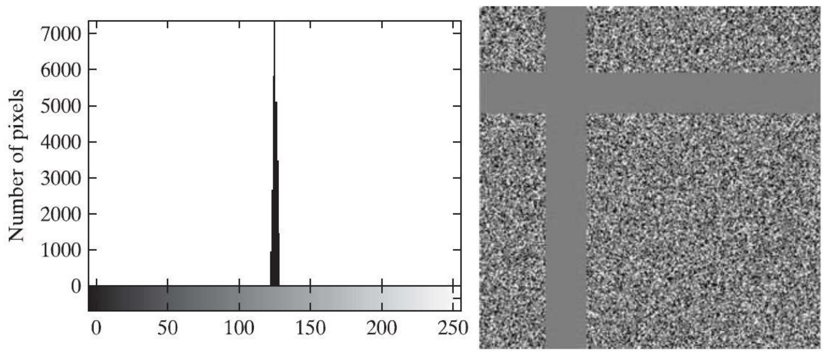


Figure 3.3: (a) Histogram of the image in Fig. 3.1(c) and (b) a histogram equalized version of the image.

visual" representation. For example, run-lengths or the differences between adjacent pixels can be used. Transformations of this type are called *mappings*. A mapping is said to be 3.1 if the pixels of the original 2-D intensity array can be reconstructed without error from the transformed data set; otherwise the mapping is said to be *irreversible*.

3.1.3 Irrelevant Information

One of the simplest ways to compress a set of data is to remove superfluous data from the set. In the context of digital image compression, information that is ignored by the human visual system or is extraneous to the intended use of an image are obvious candidates for omission. Thus, the computer-generated image in Fig. 3.1(c), because it appears to be a homogeneous field of gray, can

be represented by its average intensity alone-a single 8 -bit value. The original $256 \times 256 \times 8$ bit intensity array is reduced to a single byte; and the resulting compression is $(256 \times 256 \times 8)/8$ or $65,536 : 1$. Of course, the original $256 \times 256 \times 8$ bit image must be recreated to view and/or analyze it - but there would be little or no perceived decrease in reconstructed image quality.

Figure 3.3(a) shows the histogram of the image in Fig. 3.1(c). Note that there are several intensity values (intensities 125 through 131) actually present. The human visual system averages these intensities, perceives only the average value, and ignores the small changes in intensity that are present in this case. Figure 3.3 (b), a histogram equalized version of the image in Fig. 3.1(c), makes the intensity changes visible and reveals two previously undetected regions of constant intensity-one oriented vertically and the other horizontally. If the image in Fig. 3.1(c) is represented by its average value alone, this "invisible" structure (i.e., the constant intensity regions) and the random intensity variations surrounding them-real information-is lost. Whether or not this information should be preserved is application dependent. If the information is important, as it might be in a medical application (like digital X-ray archival), it should not be omitted; otherwise, the information is redundant and can be excluded for the sake of compression performance.

We conclude the section by noting that the redundancy examined here is fundamentally different from the redundancies discussed in Sections 3.1.1 and 3.1.2. Its elimination is possible because the information itself is not essential for normal visual processing and/or the intended use of the image. Because its omission results in a loss of quantitative information, its removal is commonly referred to as *quantization.* This terminology is consistent with normal use of the word, which generally means the mapping of a broad range of input values to a limited number of output values. Because information is lost, quantization is an irreversible operation.

3.1.4 Measuring Image Information

In the previous sections, we introduced several ways to reduce the amount of data used to represent an image. The question that naturally arises is this: How few bits are actually needed to represent the information in an image? That is, is there a minimum amount of data that is sufficient to describe an image without losing information? *Information theory* provides the mathematical framework to answer this and related questions. Its fundamental premise is that the generation of information can be modeled as a probabilistic process that can be measured in a manner that agrees with intuition. In accordance with this supposition, a random event E with probability $P(E)$ is said to contain

$$I(E) = \log \frac{1}{P(E)} = -\log P(E) \tag{3.5}$$

units of information. If $P(E) = 1$ (that is, the event always occurs), $I(E) = 0$ and no information is attributed to it. Because no uncertainty is associated with the event, no information would be transferred by communicating that the event has occurred [it *always* occurs if $P(E) = 1$].

The base of the logarithm in Eq. (3.5) determines the unit used to measure information. If the base m logarithm is used, the measurement is said to be in m-ary units. If the base 2 is selected, the unit of information is the bit. Note that if $P(E) = \frac{1}{2}, I(E) = -\log_2 \frac{1}{2}$, or 1 bit. That is, 1 bit is the amount of information conveyed when one of two possible equally likely events occurs. A simple example is flipping a coin and communicating the result.

Given a source of statistically independent random events from a discrete set of possible events $\{a_1, a_2, \ldots, a_J\}$ with associated probabilities $\{P(a_1), P(a_2), \ldots, P(a_J)\}$, the average information per source output, called the *entropy* of the source, is

$$H = -\sum_{j=1}^{J} P\left(a_j\right) \log P\left(a_j\right) \tag{3.6}$$

The a_j in this equation are called source symbols. Because they are statistically independent, the source itself is called a *zero-memory source*.

If an image is considered to be the output of an imaginary zero-memory "intensity source," we can use the histogram of the observed image to estimate the symbol probabilities of the source. Then the intensity source's entropy becomes

$$\tilde{H} = -\sum_{k=0}^{L-1} p_r(r_k)\log_2 p_r(r_k) \tag{3.7}$$

where variables L, r_k, and $p_r(r_k)$ are as defined earlier. Because the base 2 logarithm is used, Eq. (3.7) is the average information per intensity output of the imaginary intensity source in bits. It is not possible to code the *intensity values* of the imaginary source (and thus the sample image) with fewer than $\widetilde{H}$ bits/pixel.

Example: Image entropy estimates

The entropy of the image in Fig. 3.1(a) can be estimated by substituting the intensity probabilities from Table 3.1 into Eq. (3.7):

$$\begin{aligned}\tilde{H} &= -[0.25\log_2 0.25 + 0.47\log_2 0.47 + 0.25\log_2 0.25 + 0.03\log_2 0.03]\\ &\approx -[0.25(-2) + 0.47(-1.09) + 0.25(-2) + 0.03(-5.06)]\\ &\approx 1.6614 \text{ bits/pixel}\end{aligned}$$

In a similar manner, the entropies of the images in Fig. 3.1(b) and (c) can be shown to be 8 bits/pixel and 1.566 bits/pixel, respectively. Note that the image in Fig. 3.1(a) appears to have the most visual information, but has almost the lowest computed entropy -1.66 bits/pixel. The image in Fig. 3.1(b) has almost five times the entropy of the image in (a), but appears to have about the same (or less) visual information; and the image in Fig. 3.1(c), which seems to have little or no information, has almost the same entropy as the image in (a). The obvious conclusion is that the amount of entropy and thus information in an image is far from intuitive.

3.1.5 Fidelity Criteria

In Section 3.1.3, it was noted that the removal of "irrelevant visual" information involves a loss of real or quantitative image information. Because information is lost, a means of quantifying the nature of the loss is needed. Two types of criteria can be used for such an assessment: (1) objective fidelity criteria and (2) subjective fidelity criteria.

When information loss can be expressed as a mathematical function of the input and output of a compression process, it is said to be based on an *objective fidelity criterion*. An example is the *root-mean-square* (rms) *error* between two images. Let $f(x,y)$ be an input image and $\hat{f}(x,y)$ be an approximation of $f(x,y)$ that results from compressing and subsequently decompressing the input. For any value of x and y, the error $e(x,y)$ between $f(x,y)$ and $\hat{f}(x,y)$ is

$$e(x,y) = \hat{f}(x,y) - f(x,y) \tag{3.8}$$

so that the total error between the two images is

$$\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}[\hat{f}(x,y) - f(x,y)]$$

where the images are of size $M \times N$. The *root-mean-square error*, e_{rms}, between $f(x,y)$ and $\hat{f}(x,y)$ is then the square root of the squared error averaged over the $M \times N$ array, or

$$e_{\text{rms}} = \left[\frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}[\hat{f}(x,y) - f(x,y)]^2\right]^{1/2} \tag{3.9}$$

If $\hat{f}(x,y)$ is considered [by a simple rearrangement of the terms in Eq. (3.8)] to be the sum of the original image $f(x,y)$ and an error or "noise" signal $e(x,y)$, the mean-square signal-to-noise ratio of the output image, denoted SNR_{ms}, can be defined as:

$$\mathrm{SNR}_{\mathrm{ms}} = \frac{\sum_{x=0}^{M-1} \sum_{y=0}^{N-1} \hat{f}(x,y)^2}{\sum_{x=0}^{M-1} \sum_{y=0}^{N-1} [\hat{f}(x,y) - f(x,y)]^2} \tag{3.10}$$

The rms value of the signal-to-noise ratio, denoted $\mathrm{SNR}_{\mathrm{rms}}$, is obtained by taking the square root of Eq. (3.10).

While objective fidelity criteria offer a simple and convenient way to evaluate information loss, decompressed images are ultimately viewed by humans. So, measuring image quality by the subjective evaluations of people is often more appropriate. This can be done by presenting a decompressed image to a cross section of viewers and averaging their evaluations. The evaluations may be made using an absolute rating scale or by means of side-by-side comparisons of $f(x,y)$ and $\hat{f}(x,y)$. Table 3.2 shows one possible absolute rating scale. Side-by-side comparisons can be done with a scale such as $\{-3,-2,-1,0,1,2,3\}$ to represent the subjective evaluations {*much worse, worse, slightly worse, the same, slightly better, better, much better* }, respectively. In either case, the evaluations are based on *subjective fidelity criteria*.

Figure 3.4 shows three different approximations of the image in Fig. 3.1(a). Using Eq. (3.9) with Fig. 3.1(a) for $f(x,y)$ and the images in Figs. 3.4 (a) through (c) as $\hat{f}(x,y)$, the computed rms errors are $5.17, 15.67$, and 14.17 intensity levels, respectively. In terms of rms error - an objective fidelity criterionthe three images in Fig. 3.4 are ranked in order of decreasing quality as $\{(a),(c),(b)\}$.

Example: Image quality comparisons:

Figures 3.4(a) and (b) are typical of images that have been compressed and subsequently reconstructed. Both retain the essential information of the original image-like the spatial and intensity characteristics of its objects. And their rms errors correspond roughly to perceived quality. Figure 3.4(a), which is practically as good as the original image, has the lowest rms error, while Fig.

Table 3.2: Rating scale of the Television Allocations Study Organization.

Value	Rating	Description
1	Excellent	An image of extremely high quality, as good as you could desire.
2	Fine	An image of high quality, providing enjoyable viewing. Interference is not objectionable.
3	Passable	An image of acceptable quality. Interference is not objectionable.
4	Marginal	An image of poor quality; you wish you could improve it. Interference is somewhat objectionable.
6	Inferior	A very poor image, but you could watch it. Objectionable interference is definitely present.
		An image so bad that you could not watch it.

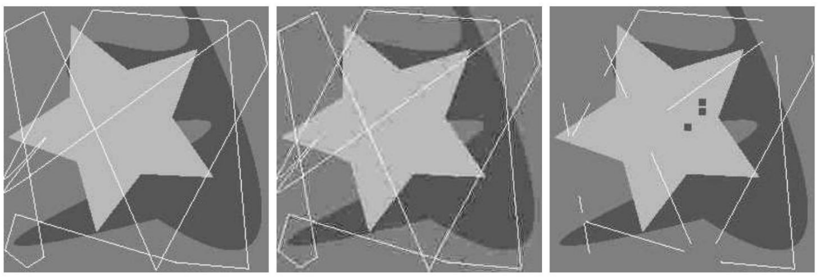

a b c

Figure 3.4: Three approximations of the image in Fig. 3.1(a).

3.4(b) has more error but noticeable degradation at the boundaries between objects. This is exactly as one would expect.

Figure 3.4(c) is an artificially generated image that demonstrates the limitations of objective fidelity criteria. Note that the image is missing large sections of several important lines (i.e., visual information), and has small dark squares (i.e., artifacts) in the upper right quadrant. The visual content of the image is misleading and certainly not as accurate as the image in (b), but it has less rms error -14.17 versus 15.67 intensity values. A subjective evaluation of the three images using Table 3.2 might yield an excellent rating

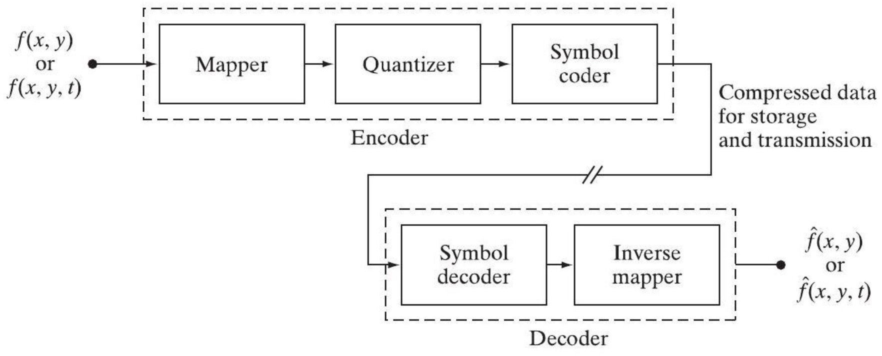


Figure 3.5: Functional block diagram of a general image compression system.

for (a), a passable or marginal rating for (b), and an inferior of unusable rating for (c). The rms error measure, on the other hand, ranks (c) ahead of (b).

3.1.6 Image Compression Models

As Fig. 3.5 shows, an image compression system is composed of two distinct functional components: an *encoder* and a *decoder*. The encoder performs compression, and the decoder performs the complementary operation of decompression. Both operations can be performed in software, as is the case in Web browsers and many commercial image editing programs, or in a combination of hardware and firmware, as in commercial DVD players. A *codec* is a device or program that is capable of both encoding and decoding.

Input image $f(x,\ldots)$ is fed into the encoder, which creates a compressed representation of the input. This representation is stored for later use, or transmitted for storage and use at a remote location. When the compressed representation is presented to its complementary decoder, a reconstructed output image $\hat{f}(x,\ldots)$ is generated. In still-image applications, the encoded input and decoder output are $f(x,y)$ and $\hat{f}(x,y)$, respectively; in video applications, they are $f(x,y,t)$ and $\hat{f}(x,y,t)$, where discrete parameter t specifies time. In general, $\hat{f}(x,\ldots)$ may or may not be an exact replica of $f(x,\ldots)$. If it is, the compression system is called error free, lossless, or information preserving. If not, the reconstructed output image is distorted and the compression system is referred to as lossy.

The encoding or compression process

The encoder of Fig. 3.5 is designed to remove the redundancies through a series of three independent operations. In the first stage of the encoding process, a *mapper* transforms $f(x,\dots)$ into a (usually nonvisual) format designed to reduce spatial and temporal redundancy. This operation generally is reversible and may or may not reduce directly the amount of data required to represent the image. Run-length coding is an example of a mapping that normally yields compression in the first step of the encoding process. The mapping of an image into a set of less correlated transform coefficients is an example of the opposite case (the coefficients must be further processed to achieve compression). In video applications, the mapper uses previous (and in some cases future) video frames to facilitate the removal of temporal redundancy.

The *quantizer* in Fig. 3.5 reduces the accuracy of the mapper's output in accordance with a pre-established fidelity criterion. The goal is to keep irrelevant information out of the compressed representation. This operation is irreversible. It must be omitted when error-free compression is desired. In video applications, the *bit rate* of the encoded output is often measured (in bits/second) and used to adjust the operation of the quantizer so that a predetermined average output rate is maintained. Thus, the visual quality of the output can vary from frame to frame as a function of image content.

In the third and final stage of the encoding process, the *symbol coder* of Fig. 3.5 generates a fixed- or variable-length code to represent the quantizer output and maps the output in accordance with the code. In many cases, a variable-length code is used. The shortest code words are assigned to the most frequently occurring quantizer output values-thus minimizing coding redundancy. This operation is reversible. Upon its completion, the input image has been processed for the removal of each of the three redundancies described earlier.

The decoding or decompression process

The decoder of Fig. 3.5 contains only two components: a *symbol decoder* and an *inverse mapper*. They perform, in reverse order, the inverse operations of the encoder's symbol encoder and mapper. Because quantization results in irreversible information loss, an inverse quantizer block is not included in the general decoder model. In video applications, decoded output frames are maintained in an internal frame store (not shown) and used to reinsert the temporal redundancy that was removed at the encoder.

3.1.7 Image Formats and Compression Standards

In the context of digital imaging, an image file format is a standard way to organize and store image data. It defines how the data is arranged and the type of compression-if any-that is used. An image container is similar to a file format but handles multiple types of image data. Image compression standards, on the other hand, define procedures for compressing and decompressing images - that is, for reducing the amount of data needed to represent an image. These standards are the underpinning of the widespread acceptance of image compression technology.

Figure 3.6 lists the most important image compression standards, file formats, and containers in use today, grouped by the type of image handled. The entries in black are international standards sanctioned by the International Standards Organization (ISO), the International Electrotechnical Commission (IEC), and/or the International Telecommunications Union (ITU-T)-a United Nations (UN) organization that was once called the Consultative Committee of the International Telephone and Telegraph (CCITT). Two video compression

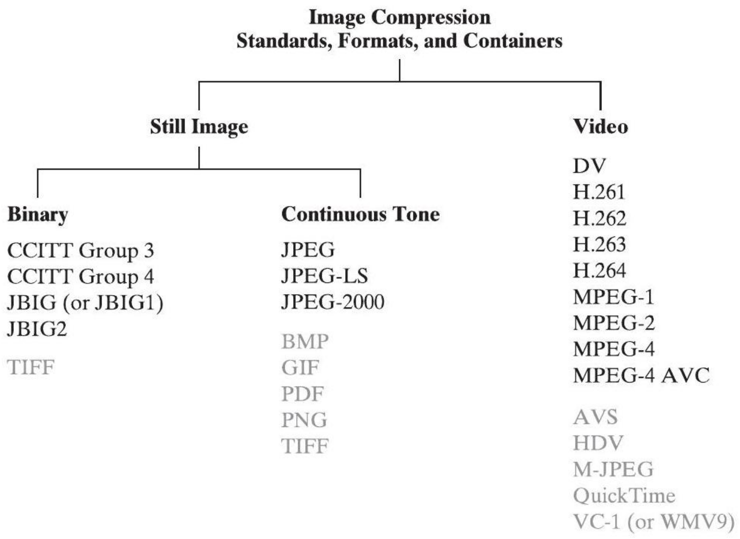


Figure 3.6: Some popular image compression standards, file formats, and containers. Internationally sanctioned entries are shown in black; all others are grayed.

standards, VC-1 by the Society of Motion Pictures and Television Engineers (SMPTE) and AVS by the Chinese Ministry of Information Industry (MII), are also included. Note that they are shown in gray, which is used in Fig. 3.6 to denote entries that are not sanctioned by an international standards organization.

3.2 Some Basic Compression Methods

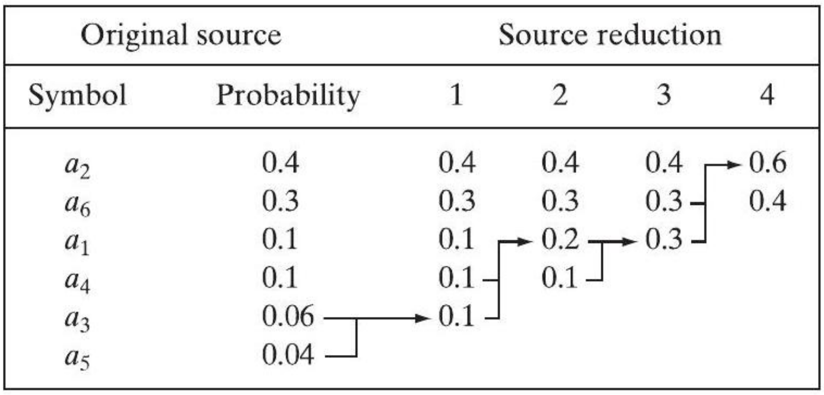
In this section, we describe the principal lossy and error-free compression methods in use today. Our focus is on methods that have proven useful in mainstream binary, continuous-tone still images, and video compression standards. The standards themselves are used to demonstrate the methods presented.

3.2.1 Huffman Coding

One of the most popular techniques for removing coding redundancy is due to Huffman. When coding the symbols of an information source individually, Huffman coding yields the smallest possible number of code symbols per source symbol. In practice, the source symbols may be either the intensities of an image or the output of an intensity mapping operation (pixel differences, run lengths, and so on).

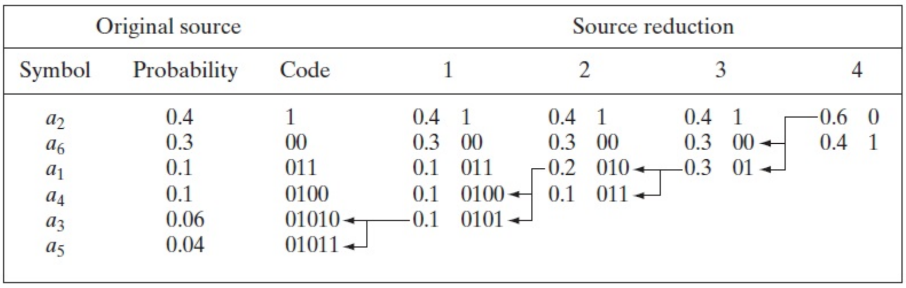
The first step in Huffman's approach is to create a series of source reductions by ordering the probabilities of the symbols under consideration and combining the lowest probability symbols into a single symbol that replaces them in the next source reduction. Figure 3.7 illustrates this process for binary coding (K-ary Huffman codes can also be constructed). At the far left, a hypothetical set of source symbols and their probabilities are ordered from top to bottom in terms of decreasing probability values. To form the first source reduction, the bottom two probabilities, 0.06 and 0.04 , are combined to form a "compound symbol" with probability 0.1 . This compound symbol and its associated probability are placed in the first source reduction column so that the probabilities of the reduced source also are ordered from the most to the least probable. This process is then repeated until a reduced source with two symbols (at the far right) is reached.

The second step in Huffman's procedure is to code each reduced source, starting with the smallest source and working back to the original source. The minimal length binary code for a two-symbol source, of course, are the symbols 0 and 1. As Fig. 3.8 shows, these symbols are assigned to the two symbols on the right (the assignment is arbitrary; reversing the order of the 0 and 1 would work just as well). As the reduced source symbol with probability 0.6 was generated by combining two symbols in the reduced source to its left, the 0 used to code it is now assigned to both of these symbols, and a 0 and 1 are arbitrarily appended to each to distinguish them from each other. This operation is then repeated for each reduced source until the original source is



Original source		Source reduction			
Symbol	Probability	1	2	3	4
a_2	0.4	0.4	0.4	0.4	0.6
a_6	0.3	0.3	0.3	0.3	0.4
a_1	0.1	0.1	0.2	0.3	
a_4	0.1	0.1	0.1		
a_3	0.06	0.1			
a_5	0.04				

Figure 3.7: Huffman source reductions.



Original source			Source reduction							
Symbol	Probability	Code	1		2		3		4	
a_2	0.4	1	0.4	1	0.4	1	0.4	1	0.6	0
a_6	0.3	00	0.3	00	0.3	00	0.3	00	0.4	1
a_1	0.1	011	0.1	011	0.2	010	0.3	01		
a_4	0.1	0100	0.1	0100	0.1	011				
a_3	0.06	01010	0.1	0101						
a_5	0.04	01011								

Figure 3.8: Huffman code assignment procedure.

reached. The final code appears at the far left in Fig. 8.8. The average length of this code is

$$\begin{aligned} L_{\text{avg}} &= (0.4)(1) + (0.3)(2) + (0.1)(3) + (0.1)(4) + (0.06)(5) + (0.04)(5) \\ &= 2.2\text{bits}/\ \text{pixel} \end{aligned}$$

and the entropy of the source is 2.14 bits/symbol.

Huffman's procedure creates the optimal code for a set of symbols and probabilities subject to the constraint that the symbols be coded one at a time. After the code has been created, coding and/or error-free decoding is accomplished in a simple lookup table manner. The code itself is an instantaneous uniquely decodable block code. It is called a block code because each source symbol is mapped into a fixed sequence of code symbols. It is instantaneous because each code word in a string of code symbols can be decoded without referencing succeeding symbols. It is uniquely decodable because any string of code symbols can be decoded in only one way. Thus, any string of Huffman encoded symbols can be decoded by examining the individual symbols of the string in a left-to-right manner. For the binary code of Fig. 3.8, a left-to-right scan of the encoded string 010100111100 reveals that the first valid code word is 01010 , which is the code for symbol a_3. The next valid code is 011 , which corresponds to symbol a_1. Continuing in this manner reveals the completely decoded message to be $a_3a_1a_2a_2a_6$.

3.2.2 LZW Coding

The techniques covered in the previous sections are focused on the removal of coding redundancy. In this section, we consider an error-free compression approach that also addresses spatial redundancies in an image. The technique, *called Lempel-Ziv-Welch (LZW) coding*, assigns fixed-length code words to variable length sequences of source symbols. A key feature of LZW coding is that it requires no a priori knowledge of the probability of occurrence of the symbols to be encoded. Despite the fact that until recently it was protected under a United States patent, LZW compression has been integrated into a variety of mainstream imaging file formats, including GIF, TIFF, and PDF. The PNG format was created to get around LZW licensing requirements.

LZW coding is conceptually very simple. At the onset of the coding process, a codebook or dictionary containing the source symbols to be coded is constructed. For 8-bit monochrome images, the first 256 words of the dictionary are assigned to intensities $0,1,2,\ldots,255$. As the encoder sequentially examines image pixels, intensity sequences that are not in the dictionary are placed in algorithmically determined (e.g., the next unused) locations. If the first two pixels of the image are white, for instance, sequence "255-255" might be assigned to location 256, the address following the locations reserved for intensity levels 0 through 255 . The next time that two consecutive white pixels are encountered, code word 256 , the address of the location containing sequence $255-255$, is used to represent them. If a 9 -bit, 512 -word dictionary is employed in the coding process, the original $(8+8)$ bits that were used to represent the two pixels are replaced by a single 9 -bit code word. Clearly, the size of the dictionary is an important system parameter. If it is too small, the detection of matching intensity-level sequences will be less likely; if it is too large, the size of the code words will adversely affect compression performance.

EXAMPLE: LZW coding

Consider the following 4×4, 8-bit image of a vertical edge:

$$\begin{array}{cccc}
39 & 39 & 126 & 126 \\
39 & 39 & 126 & 126 \\
39 & 39 & 126 & 126 \\
39 & 39 & 126 & 126
\end{array}$$

Table 3.3 details the steps involved in coding its 16 pixels. A 512-word dictionary with the following starting content is assumed:

Dictionary Location	Entry
0	0
1	1
⋮	⋮
255	255
256	-
⋮	⋮
511	-

Locations 256 through 511 initially are unused.
The image is encoded by processing its pixels in a left-to-right, top-to-bottom manner. Each successive intensity value is concatenated with a variablecolumn 1 of Table 3.3 -called the "currently recognized sequence." As can be seen, this variable is initially null or empty. The dictionary is searched for each concatenated sequence and if found, as was the case in the first row of the table, is replaced by the newly concatenated and recognized (i.e., located in the dictionary) sequence. This was done in column 1 of row 2 . No output codes are generated, nor is the dictionary altered. If the concatenated sequence is not found, however, the address of the currently recognized sequence is output as the next encoded value, the concatenated but unrecognized sequence is added to the dictionary, and the currently recognized sequence is initialized to the current pixel value. This occurred in row 2 of the table. The last two columns detail the intensity sequences that are added to the dictionary when scanning the entire 4×4 image. Nine additional code words are defined. At the conclusion of coding, the dictionary contains 265 code words and the LZW algorithm has successfully identified several repeating intensity sequences-leveraging them to reduce the original 128-bit image to 90 bits (i.e., 109 -bit codes). The encoded output is obtained by reading the third column from top to bottom. The resulting compression ratio is 1.42:1.

A unique feature of the LZW coding just demonstrated is that the coding dictionary or code book is created while the data are being encoded. Remarkably, an LZW decoder builds an identical decompression dictionary as it decodes simultaneously the encoded data stream. It is left as an exercise to the reader (see Problem 8.20) to decode the output of the preceding example and reconstruct the code book. Although not needed in this example, most practical applications require a strategy for handling dictionary overflow. A simple solution is to flush or reinitialize the dictionary when it becomes full and continue coding with a new initialized dictionary. A more complex option is to monitor compression performance and flush the dictionary when it

Table 3.3: LZW coding example.

Currently Recognized Sequence	Pixel Being Processed	Encoded Output	Dictionary Location (Code Word)	Dictionary Entry
39	39			
39	39	39	256	39 – 39
126	126	39	257	39 – 126
126	126	126	258	126 – 126
39	39	126	259	126 – 39
39 – 39	39			39 – 39 – 126
126	126	256	260	126 – 126 – 39
126 – 126	126			
39	39	258	261	39 – 39 – 126 – 126
39 – 39	39			126 – 39 – 39
39 – 39 – 126	126	260	262	39 – 126 – 126
126	126			
126 – 39	39	259	263	
39	126			
39 – 126	126	257	264	
126		126		

becomes poor or unacceptable. Alternatively, the least used dictionary entries can be tracked and replaced when necessary.

3.2.3 Run-Length Coding

Images with repeating intensities along their rows (or columns) can often be compressed by representing runs of identical intensities as run-length pairs, where each *run-length pair* specifies the start of a new intensity and the number of consecutive pixels that have that intensity. The technique, referred to as *run-length encoding* (RLE), was developed in the 1950s and became, along with its 2-D extensions, the standard compression approach in facsimile (FAX) coding. Compression is achieved by eliminating a simple form of spatial redundancy - groups of identical intensities. When there are few (or no) runs of identical pixels, *run-length encoding* results in data expansion.

Table 3.4: BMP absolute coding mode options. In this mode, the first byte of the BMP pair is 0.

Second Byte Value	Condition
0	End of line
1	End of image
2	Move to a new position
3 – 255	Specify pixels individually

Example: RLE in the BMP file format:

The BMP file format uses a form of run-length encoding in which image data is represented in two different modes: encoded and absolute-and either mode can occur anywhere in the image. In *encoded* mode, a two byte RLE representation is used. The first byte specifies the number of consecutive pixels that have the color index contained in the second byte. The 8-bit color index selects the run's intensity (color or gray value) from a table of 256 possible intensities.

In *absolute* mode, the first byte is 0 and the second byte signals one of four possible conditions, as shown in Table 3.4. When the second byte is 0 or 1 , the end of a line or the end of the image has been reached. If it is 2 , the next two bytes contain unsigned horizontal and vertical offsets to a new spatial position (and pixel) in the image. If the second byte is between 3 and 255, it specifies the number of uncompressed pixels that follow-with each subsequent byte containing the color index of one pixel. The total number of bytes must be aligned on a 16-bit word boundary.

Run-length encoding is particularly effective when compressing binary images. Because there are only two possible intensities (black and white), adjacent pixels are more likely to be identical. In addition, each image row can be represented by a sequence of lengths only - rather than length-intensity pairs. The basic idea is to code each contiguous group (i.e., run) of 0 s or 1 s encountered in a left to right scan of a row by its length and to establish a convention for determining the value of the run. The most common conventions are (1) to specify the value of the first run of each row, or (2) to assume that each row begins with a white run, whose run length may in fact be zero.

Although run-length encoding is in itself an effective method of compressing binary images, additional compression can be achieved by variable-length coding the run lengths themselves. The black and white run lengths can be coded separately using variable-length codes that are specifically tailored to their own statistics.

3.3 summary

The principal objectives of this chapter were to present the theoretic foundation of digital image compression and to describe the most commonly used compression methods. compression plays a key role in document image storage and transmission, the Internet, and commercial video distribution (e.g., DVDs). It is one of the few areas of image processing that has received a sufficiently broad commercial appeal to warrant the adoption of widely accepted standards. The main advantages of compression are reductions in storage hardware, data transmission time, and communication bandwidth. This can result in significant cost savings. Compressed files require significantly less storage capacity than uncompressed files, meaning a significant decrease in expenses for storage.

CHAPTER 4
IMAGE SEGMENTATION

The material in this chapter begins a transition from image processing methods whose inputs and outputs are images, to methods in which the inputs are images but the outputs are attributes extracted from those images.

Segmentation subdivides an image into its constituent regions or objects. The level of detail to which the subdivision is carried depends on the problem being solved. That is, segmentation should stop when the objects or regions of interest in an application have been detected. For example, in the automated inspection of electronic assemblies, interest lies in analyzing images of products with the objective of determining the presence or absence of specific anomalies, such as missing components or broken connection paths. There is no point in carrying segmentation past the level of detail required to identify those elements.

Segmentation of nontrivial images is one of the most difficult tasks in image processing. Segmentation accuracy determines the eventual success or failure of computerized analysis procedures. For this reason, considerable care should be taken to improve the probability of accurate segmentation. In some situations, such as in industrial inspection applications, at least some measure of control over the environment typically is possible. The experienced image processing system designer invariably pays considerable attention to such opportunities. In other applications, such as autonomous target acquisition, the system designer has no control over the operating environment, and the usual approach is to focus on selecting the types of sensors most likely to enhance the objects of interest while diminishing the contribution of irrelevant image detail. A good example is the use of infrared imaging by the military to detect objects with strong heat signatures, such as equipment and troops in motion.

Most of the segmentation algorithms in this chapter are based on one of two basic properties of intensity values: discontinuity and similarity. In the first category, the approach is to partition an image based on abrupt changes in intensity, such as edges. The principal approaches in the second category are based on partitioning an image into regions that are similar according to a set of predefined criteria.

4.1 Fundamentals

Let R represent the entire spatial region occupied by an image. We may view image segmentation as a process that partitions R into n subregions, $R_1, R_2, \ldots, R_n$, such that
(a) $\bigcup_{i=1}^{n} R_i = R$.
(b) R_i is a connected set $i = 1, 2, \ldots, n$.
(c) $R_i \cap R_j = \varnothing$ for all i and $j, i \neq j$.
(d) $Q(R_i) = \text{TRUE}$ for $i = 1, 2, \ldots, n$.
(e) $Q\left(R_i \cup R_j\right) = \text{FALSE}$ for any adjacent regions R_i and R_j.

Here, $Q(R_k)$ is a logical predicate defined over the points in set R_k, and $\varnothing$ is the null set. The symbols $\cup$ and $\cap$ represent set union and intersection, respectively. Two regions R_i and R_j are said to be adjacent if their union forms a connected set.

Condition (a) indicates that the segmentation must be complete; that is, every pixel must be in a region. Condition (b) requires that points in a region be connected in some predefined sense (e.g., the points must be 4 - or 8 -connected). Condition (c) indicates that the regions must be disjoint.

Condition (d) deals with the properties that must be satisfied by the pixels in a segmented region-for example, $Q(R_i) = \text{TRUE}$ if all pixels in R_i have the same intensity level. Finally, condition (e) indicates that two adjacent regions R_i and R_j must be different in the sense of predicate Q.[†]

Thus, we see that the fundamental problem in segmentation is to partition an image into regions that satisfy the preceding conditions. Segmentation algorithms for monochrome images generally are based on one of two basic categories dealing with properties of intensity values: discontinuity and similarity. In the first category, the assumption is that boundaries of regions are sufficiently different from each other and from the background to allow boundary detection based on local discontinuities in intensity. Edge-based segmentation is the principal approach used in this category. Region-based segmentation approaches in the second category are based on partitioning an image into regions that are similar according to a set of predefined criteria.

Figure 4.1(a) shows an image of a region of constant intensity superimposed on a darker background, also of constant intensity. These two regions comprise the overall image region. Figure 4.1(b) shows the result of computing the boundary of the inner region based on intensity discontinuities. Points on the inside and outside of the boundary are black (zero) because there are no discontinuities in intensity in those regions. To segment the image, we assign one level (say, white) to the pixels on, or interior to, the boundary and another level (say, black) to all points exterior to the boundary. Figure 4.1(c) shows the result of such a procedure. We see that conditions (a) through (c) stated at the beginning of this section are satisfied by this result. The predicate of condition (d) is: If a pixel is on, or inside the boundary, label it white; otherwise label it black. We see that this predicate is TRUE for the points labeled black and white in Figure 4.1(c). Similarly, the two segmented regions (object and background) satisfy condition (e).

[0†] In general, Q can be a compound expression such as, for example, $Q(R_i) = \text{TRUE}$ if the average intensity of the pixels in R_i is less than m_i, AND if the standard deviation of their intensity is greater than σ_i, where m_i and σ_i are specified constants.

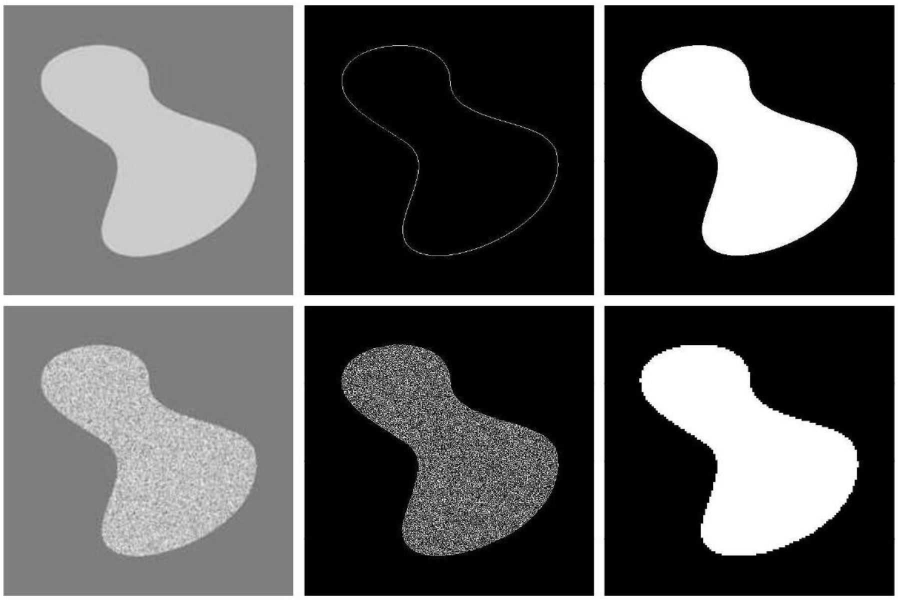


Figure 4.1: (a) Image containing a region of constant intensity. (b) Image showing the boundary of the inner region, obtained from intensity discontinuities. (c) Result of segmenting the image into two regions. (d) Image containing a textured region. (e) Result of edge computations. Note the large number of small edges that are connected to the original boundary, making it difficult to find a unique boundary using only edge information. (f) Result of segmentation based on region properties.

The next three images illustrate region-based segmentation. Figure 10.1(d) is similar to Fig. 10.1(a), but the intensities of the inner region form a textured pattern. Figure 10.1(e) shows the result of computing the edges of this image. Clearly, the numerous spurious changes in intensity make it difficult to identify a unique boundary for the original image because many of the nonzero intensity changes are connected to the boundary, so edge-based segmentation is not a suitable approach. We note however, that the outer region is constant, so all we need to solve this simple segmentation problem is a predicate that differentiates between textured and constant regions. The standard deviation of pixel values is a measure that accomplishes this, because it is nonzero in areas of the texture region and zero otherwise. Figure 10.1(f) shows the result of dividing the original image into subregions of size 4×4. Each subregion was then labeled white if the standard deviation of its pixels was positive (i.e., if the predicate was TRUE) and zero otherwise. The result has a "blocky" appearance around the edge of the region because groups of 4×4 squares were labeled with the same intensity. Finally, note that these results also satisfy the five conditions stated at the beginning of this section.

4.2 Point, Line, and Edge Detection

The focus of this section is on segmentation methods that are based on detecting sharp, local changes in intensity. The three types of image features in which we are interested are isolated points, lines, and edges. Edge pixels are pixels at which the intensity of an image function changes abruptly, and edges (or edge segments) are sets of connected edge pixels. Edge detectors are local image processing methods designed to detect edge pixels. A line may be viewed as an edge segment in which the intensity of the background on either side of the line is either much higher or much lower than the intensity of the line pixels. In fact, as we discuss in the following section, lines give rise to so-called "roof edges." Similarly, an isolated point may be viewed as a line whose length and width are equal to one pixel.

4.2.1 Background

As we discussed earlier, local averaging smooths an image. Given that averaging is analogous to integration, it should come as no surprise that abrupt, local changes in intensity can be detected using derivatives. For reasons that will become evident shortly, first- and second-order derivatives are particularly well suited for this purpose.

Derivatives of a digital function are defined in terms of differences. There are various ways to approximate these differences but, we require that any approximation used for a first derivative (1) must be zero in areas of constant intensity; (2) must be nonzero at the onset of an intensity step or ramp; and (3) must be nonzero at points along an intensity ramp. Similarly, we require that an approximation used for a second derivative (1) must be zero in areas of constant intensity; (2) must be nonzero at the onset and end of an intensity step or ramp; and (3) must be zero along intensity ramps. Because we are dealing with digital quantities whose values are finite, the maximum possible intensity change is also finite, and the shortest distance over which a change can occur is between adjacent pixels.

We obtain an approximation to the first-order derivative at point x of a one-dimensional function $f(x)$ by expanding the function $f(x+\Delta x)$ into a Taylor series about x, letting $\Delta x = 1$, and keeping only the linear terms. The result is the digital difference

$$\frac{\partial f}{\partial x} = f'(x) = f(x+1) - f(x) \tag{4.1}$$

We used a partial derivative here for consistency in notation when we consider an image function of two variables, $f(x,y)$, at which time we will be dealing with partial derivatives along the two spatial axes. Clearly, $\partial f/\partial x = df/dx$ when f is a function of only one variable.

We obtain an expression for the second derivative by differentiating Equation 4.1 with respect to x :

$$\begin{aligned}\frac{\partial^2 f}{\partial x^2} &= \frac{\partial f'(x)}{\partial x} = f'(x+1) - f'(x)\\ &= f(x+2) - f(x+1) - f(x+1) + f(x)\\ &= f(x+2) - 2f(x+1) + f(x)\end{aligned}$$

where the second line follows from Eq. 4.1. This expansion is about point $x+1$. Our interest is on the second derivative about point x, so we subtract 1 from the arguments in the preceding expression and obtain the result

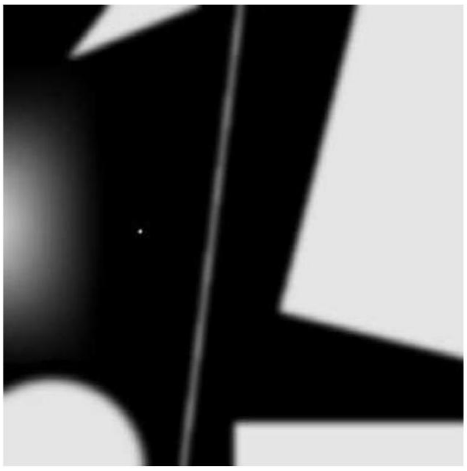
$$\frac{\partial^2 f}{\partial x^2} = f''(x) = f(x+1) + f(x-1) - 2f(x) \tag{4.2}$$

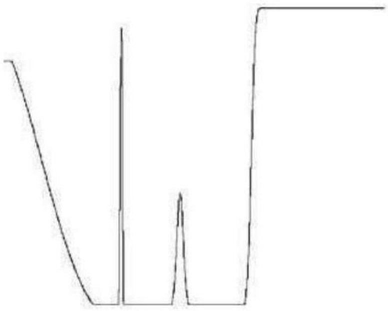
It easily is verified that Eqs. 4.1 and 4.2 satisfy the conditions stated at the beginning of this section regarding derivatives of the first and second order. To illustrate this, and also to highlight the fundamental similarities and differences between first- and second-order derivatives in the context of image processing, consider Fig. 4.2.

Figure 4.2(a) shows an image that contains various solid objects, a line, and a single noise point. Figure 4.2(b) shows a horizontal intensity profile (scan line) of the image approximately through its center, including the isolated point. Transitions in intensity between the solid objects and the background along the scan line show two types of edges: ramp edges (on the left) and step edges (on the right). As we discuss later, intensity transitions involving thin objects such as lines often are referred to as roof edges. Figure 4.2(c) shows a simplification of the profile, with just enough points to make it possible for us to analyze numerically how the first- and second-order derivatives behave as they encounter a noise point, a line, and the edges of objects. In this simplified diagram the transition in the ramp spans four pixels, the noise point is a single pixel, the line is three pixels thick, and the transition of the intensity step takes place between adjacent pixels. The number of intensity levels was limited to eight for simplicity.

Consider the properties of the first and second derivatives as we traverse the profile from left to right. Initially, we note that the first-order derivative is nonzero at the onset and along the entire intensity ramp, while the second-order derivative is nonzero only at the onset and end of the ramp. Because edges of digital images resemble this type of transition, we conclude that first-order derivatives produce "thick" edges and second-order derivatives much finer ones. Next we encounter the isolated noise point. Here, the magnitude of the response at the point is much stronger for the second- than for the first-order derivative. This is not unexpected, because a second-order derivative is much more aggressive than a first-order derivative in enhancing sharp changes. Thus, we can expect second-order derivatives to enhance fine detail (including noise) much more than first-order derivatives. The line in this example is rather thin, so it too is fine detail, and we see again that the second derivative has a larger magnitude. Finally, note in both the ramp and step edges that the second derivative has opposite signs (negative to positive or positive to negative) as it transitions into and out of an edge. This "double-edge" effect is an important characteristic that, can be used to locate edges. The sign of the second derivative is used also to determine whether an edge is a transition from light to dark (negative second derivative) or from dark to light (positive second derivative), where the sign is observed as we move into the edge.

In summary, we arrive at the following conclusions: (1) First-order derivatives generally produce thicker edges in an image. (2) Second-order derivatives have a stronger response to fine detail, such as thin lines, isolated points, and noise. (3) Second-order derivatives produce a double-edge response at ramp and step transitions in intensity. (4) The sign of the second derivative





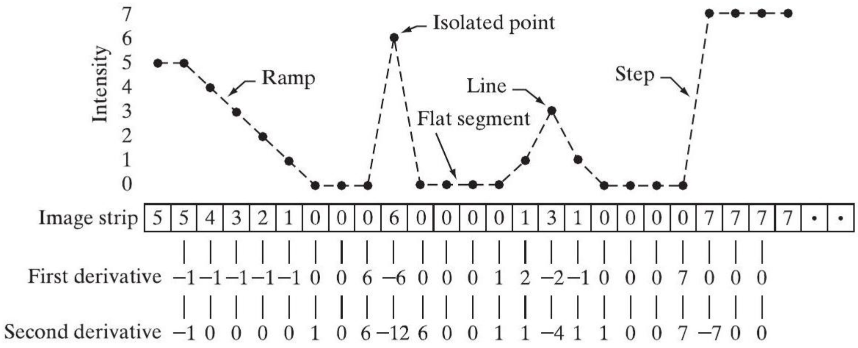

Intensity
7
6
5
4
3
2
1
0
Isolated point
Ramp
Line
Step
Flat segment
Image strip 5 5 4 3 2 1 0 0 0 6 0 0 0 0 1 3 1 0 0 0 0 7 7 7 7 • •
First derivative −1 −1 −1 −1 −1 0 0 6 −6 0 0 0 1 2 −2 −1 0 0 0 7 0 0 0
Second derivative −1 0 0 0 0 1 0 6 −12 6 0 0 1 1 −4 1 1 0 0 7 −7 0 0

Figure 4.2: (a) An image. (b) Horizontal intensity profile through the center of the image, including the isolated noise point. (c) Simplified profile (the points are joined by dashes for clarity). The image strip corresponds to the intensity profile, and the numbers in the boxes are the intensity values of the dots shown in the profile.

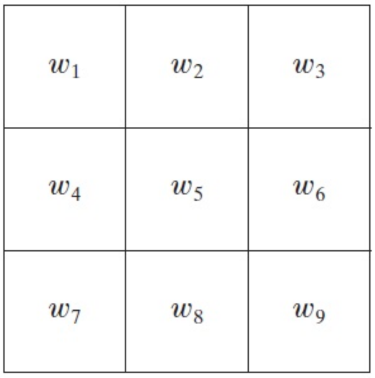


Figure 4.3: A general 3×3 spatial filter mask.

can be used to determine whether a transition into an edge is from light to dark or dark to light.

The approach of choice for computing first and second derivatives at every pixel location in an image is to use spatial filters. For the 3×3 filter mask in Fig. 4.3, the procedure is to compute the sum of products of the mask coefficients with the intensity values in the region encompassed by the mask. That is, the *response* of the mask at the center point of the region is

$$R = w_1 z_1 + w_2 z_2 + \cdots + w_9 z_9 = \sum_{k=1}^{9} w_k z_k \qquad (4.3)$$

where z_k is the intensity of the pixel whose spatial location corresponds to the location of the k th coefficient in the mask. In other words, computation of derivatives based on spatial masks is spatial filtering of an image with those masks, as explained in those sections.

4.2.2 Detection of Isolated Points

Based on the conclusions reached in the preceding section, we know that point detection should be based on the second derivative, which implies using the Laplacian:

$$\nabla^2 f(x,y) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \tag{4.4}$$

where the partials are obtained using Eq. (4.2):

$$\frac{\partial^2 f(x,y)}{\partial x^2} = f(x+1,y) + f(x-1,y) - 2f(x,y) \tag{4.5}$$

and

$$\frac{\partial^2 f(x,y)}{\partial y^2} = f(x,y+1) + f(x,y-1) - 2f(x,y) \tag{4.6}$$

The Laplacian is then

$$\nabla^2 f(x,y) = f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1) - 4f(x,y) \tag{4.7}$$

As explained, this expression can be implemented using a specific mask. Also, as explained in that section, we can extend Eq. 4.7 to include the diagonal terms. Using the Laplacian mask in Fig. 4.4 (a), we say that a point has been detected at the location (x,y) on which the mask is centered if the absolute value of the response of the mask at that point exceeds a specified threshold. Such points are labeled 1 in the output image and all others are labeled 0 , thus producing a binary image. In other words, the output is obtained using the following expression:

$$g(x,y) = \begin{cases} 1 & \text{if } |R(x,y)| \geq T \\ 0 & \text{otherwise} \end{cases} \tag{4.8}$$

where g is the output image, T is a non-negative threshold, and R is given by Eq. (4.3). This formulation simply measures the weighted differences between a pixel and its 8 -neighbors. Intuitively, the idea is that the intensity of an isolated point will be quite different from its surroundings and thus will be easily detectable by this type of mask. The only differences in intensity that are considered of interest are those large enough (as determined by T) to be considered isolated points. Note that, as usual for a derivative mask, the coefficients sum to zero, indicating that the mask response will be zero in areas of constant intensity.

We illustrate segmentation of isolated points in an image with the aid of Fig. 4.4(b), which is an X-ray image of a turbine blade from a jet engine. The blade has a porosity in the upper-right quadrant of the image, and there is a single black pixel embedded within the porosity. Figure 4.4(c) is the result of applying the point detector mask to the X-ray image, and Fig. 4.4(d) shows the result of using Eq. (4.8) with T equal to 90% of the highest absolute pixel value of the image in Fig. 4.4(c). The single pixel is clearly visible in this image (the pixel was enlarged manually to enhance its visibility). This type of detection process is rather specialized, because it is based on abrupt intensity changes at single-pixel locations that are surrounded by a homogeneous background in the area of the detector mask. When this condition is not satisfied, other methods discussed in this chapter are more suitable for detecting intensity changes.

1	1	1
1	−8	1
1	1	1

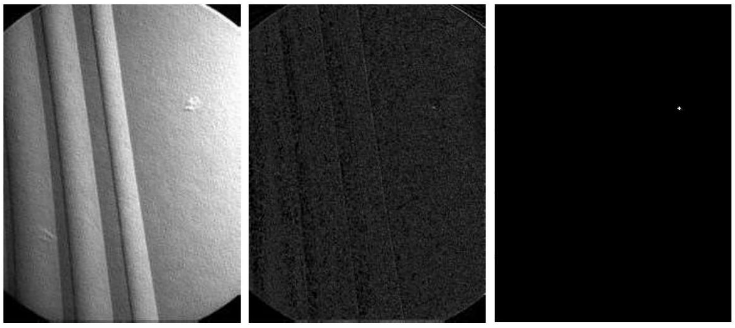


Figure 4.4: (a) Point detection (Laplacian) mask. (b) X-ray image of turbine blade with a porosity. The porosity contains a single black pixel. (c) Result of convolving the mask with the image. (d) Result of using Eq. (4.8) showing a single point (the point was enlarged to make it easier to see).

4.2.3 Line Detection

The next level of complexity is line detection. Based on the discussion in Section 4.2.1, we know that for line detection we can expect second derivatives to result in a stronger response and to produce thinner lines than first derivatives. Thus, we can use the Laplacian mask in Fig. 4.4(a) for line detection also, keeping in mind that the double-line effect of the second derivative must be handled properly. The following example illustrates the procedure.

Using the Laplacian for line detection:

Figure 4.5(a) shows a 486×486 (binary) portion of a wire-bond mask for an electronic circuit, and Fig. 4.5(b) shows its Laplacian image. Because the

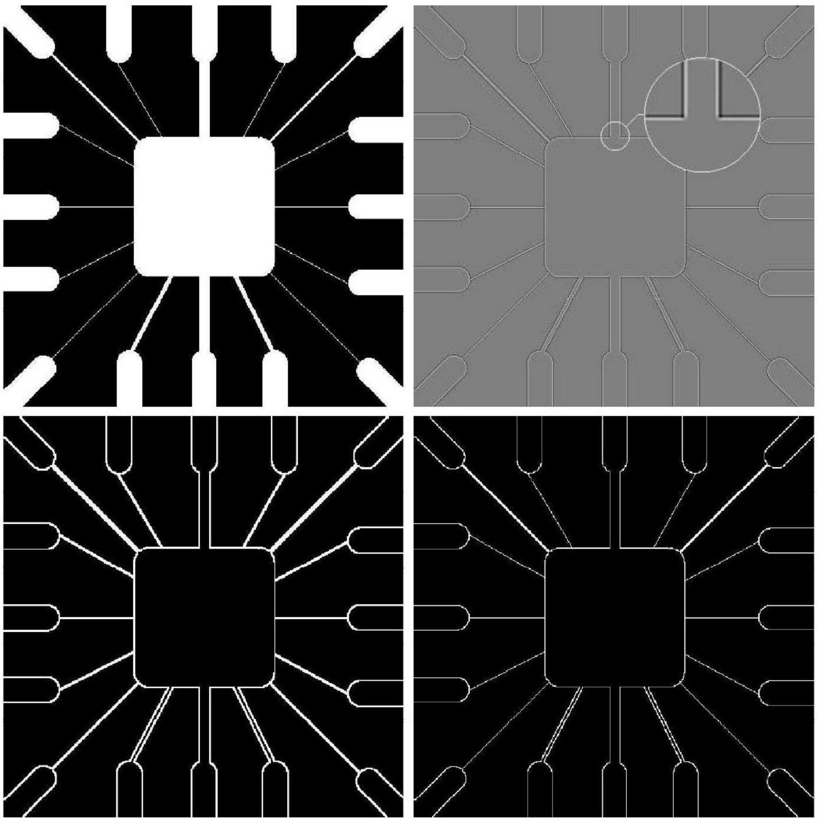


Figure 4.5: (a) Original image. (b) Laplacian image; the magnified section shows the positive/negative double-line effect characteristic of the Laplacian. (c) Absolute value of the Laplacian. (d) Positive values of the Laplacian.

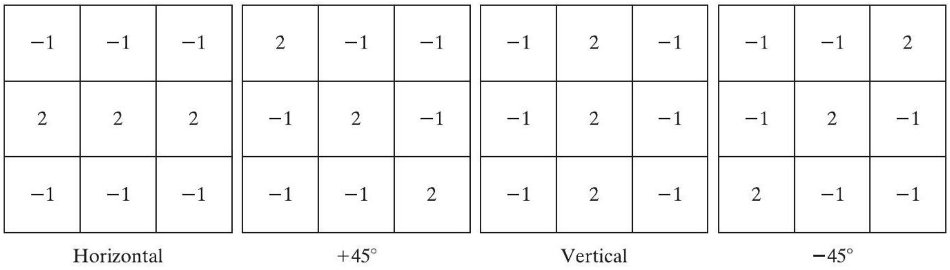
Laplacian image contains negative values, scaling is necessary for display. As the magnified section shows, mid gray represents zero, darker shades of gray represent negative values, and lighter shades are positive. The double-line effect is clearly visible in the magnified region.

At first, it might appear that the negative values can be handled simply by taking the absolute value of the Laplacian image. However, as Fig. 4.5(c) shows, this approach doubles the thickness of the lines. A more suitable approach is to use only the positive values of the Laplacian (in noisy situations we use the values that exceed a positive threshold to eliminate random variations about zero caused by the noise). As the image in Fig. 4.5(d) shows, this approach results in thinner lines, which are considerably more useful. Note in Figs. 4.5(b) through (d) that when the lines are wide with respect to the size of the Laplacian mask, the lines are separated by a zero "valley."

This is not unexpected. For example, when the 3×3 filter is centered on a line of constant intensity 5 pixels wide, the response will be zero, thus producing the effect just mentioned. When we talk about line detection, the assumption is that lines are thin with respect to the size of the detector. Lines that do not satisfy this assumption are best treated as regions and handled by the edge detection methods discussed later in this section.

The Laplacian detector in Fig. 4.4(a) is isotropic, so its response is independent of direction (with respect to the four directions of the 3×3 Laplacian mask: vertical, horizontal, and two diagonals). Often, interest lies in detecting lines in specified directions. Consider the masks in Fig. 4.6. Suppose that an image with a constant background and containing various lines (oriented at $0°$, $\pm 45°$, and $90°$) is filtered with the first mask. The maximum responses would occur at image locations in which a horizontal line passed through the middle row of the mask. This is easily verified by sketching a simple array of 1 s with a line of a different intensity (say, 5 s) running horizontally through the array. A similar experiment would reveal that the second mask in Fig. 4.6 responds best to lines oriented at $+45°$; the third mask to vertical lines; and the fourth mask to lines in the $-45°$ direction. The preferred direction of each mask is weighted with a larger coefficient (i.e., 2) than other possible directions. The coefficients in each mask sum to zero, indicating a zero response in areas of constant intensity.

Let R_1, R_2, R_3, and R_4 denote the responses of the masks in Fig. 10.6, from left to right, where the Rs are given by Eq. (4.3). Suppose that an image is filtered (individually) with the four masks. If, at a given point in



Horizontal

−1	−1	−1
2	2	2
−1	−1	−1

+45°

2	−1	−1
−1	2	−1
−1	−1	2

Vertical

−1	2	−1
−1	2	−1
−1	2	−1

−45°

−1	−1	2
−1	2	−1
2	−1	−1

Figure 4.6: Line detection masks.

the image, $|R_k| > |R_j|$, for all $j \neq k$, that point is said to be more likely associated with a line in the direction of mask k. For example, if at a point in the image, $|R_1| > |R_j|$ for $j = 2, 3, 4$, that particular point is said to be more likely associated with a horizontal line. Alternatively, we may be interested in detecting lines in a specified direction. In this case, we would use the mask associated with that direction and threshold its output, as in Eq. (4.8). In other words, if we are interested in detecting all the lines in an image in the direction defined by a given mask, we simply run the mask through the image and threshold the absolute value of the result. The points that are left are the strongest responses which, for lines 1 pixel thick, correspond closest to the direction defined by the mask. The following example illustrates this procedure.

4.2.4 Edge Models

Edge detection is the approach used most frequently for segmenting images based on abrupt (local) changes in intensity. We begin by introducing several ways to model edges and then discuss a number of approaches for edge detection.

Edge models are classified according to their intensity profiles. A *step edge* involves a transition between two intensity levels occurring ideally over the distance of 1 pixel. Figure 4.7(a) shows a section of a vertical step edge and a horizontal intensity profile through the edge. Step edges occur, for example, in images generated by a computer for use in areas such as solid modeling and animation. These clean, *ideal* edges can occur over the distance of 1 pixel, provided that no additional processing (such as smoothing) is used to make them look "real." Digital step edges are used frequently as edge models in algorithm development.

In practice, digital images have edges that are blurred and noisy, with the degree of blurring determined principally by limitations in the focusing mechanism (e.g., lenses in the case of optical images), and the noise level determined principally by the electronic components of the imaging system. In such situations, edges are more closely modeled as having an intensity *ramp* profile, such as the edge in Fig. 4.7(b). The slope of the ramp is inversely proportional to the degree of blurring in the edge. In this model, we no longer have a thin (1 pixel thick) path. Instead, an edge point now is any point contained in the ramp, and an edge segment would then be a set of such points that are connected.

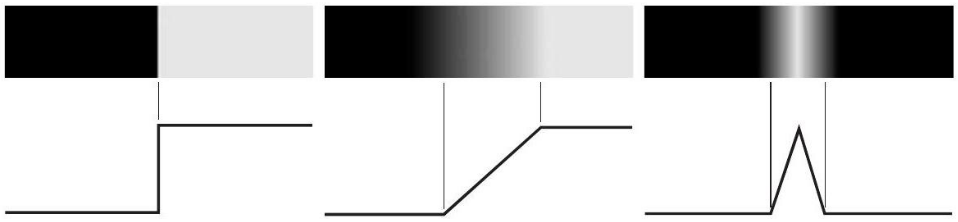


Figure 4.7: From left to right, models (ideal representations) of a step, a ramp, and a roof edge, and their corresponding intensity profiles.

A third model of an edge is the so-called roof edge, having the characteristics illustrated in Fig. 4.7(c). Roof edges are models of lines through a region, with the base (width) of a roof edge being determined by the thickness and sharpness of the line. In the limit, when its base is 1 pixel wide, a roof edge is really nothing more than a 1 -pixel-thick line running through a region in an image. Roof edges arise, for example, in range imaging, when thin objects (such as pipes) are closer to the sensor than their equidistant background (such as walls). The pipes appear brighter and thus create an image similar to the model in Fig. 4.7(c). As mentioned earlier, other areas in which roof edges appear routinely are in the digitization of line drawings and also in satellite images, where thin features, such as roads, can be modeled by this type of edge.

Figure 4.8(a) shows the image from which the segment in Fig. 4.7(b) was extracted. Figure 4.8(b) shows a horizontal intensity profile. This figure shows also the first and second derivatives of the intensity profile. As in the discussion in Section 4.2.1, moving from left to right along the intensity profile, we note that the first derivative is positive at the onset of the ramp and at points on the ramp, and it is zero in areas of constant intensity. The second derivative is positive at the beginning of the ramp, negative at the end of the ramp, zero at points on the ramp, and zero at points of constant intensity. The signs of the derivatives just discussed would be reversed for an edge that transitions from light to dark. The intersection between the zero intensity axis and a line extending between the extrema of the second derivative marks a point called the zero crossing of the second derivative.

We conclude from these observations that the *magnitude* of the first derivative can be used to detect the presence of an edge at a point in an image. Similarly, the *sign* of the second derivative can be used to determine whether an edge pixel lies on the dark or light side of an edge. We note two additional properties of the second derivative around an edge: (1) it produces two values for every edge in an image (an undesirable feature); and (2) its zero crossings can be used for locating the centers of thick edges, as we show later in this section. Some edge models make use of a smooth transition into and out of the ramp. However, the conclusions reached using those models are the same as with an ideal ramp, and working with the latter simplifies theoretical formulations. Finally, although attention thus far has been limited to a 1-D horizontal profile, a similar argument applies to an edge of any orientation

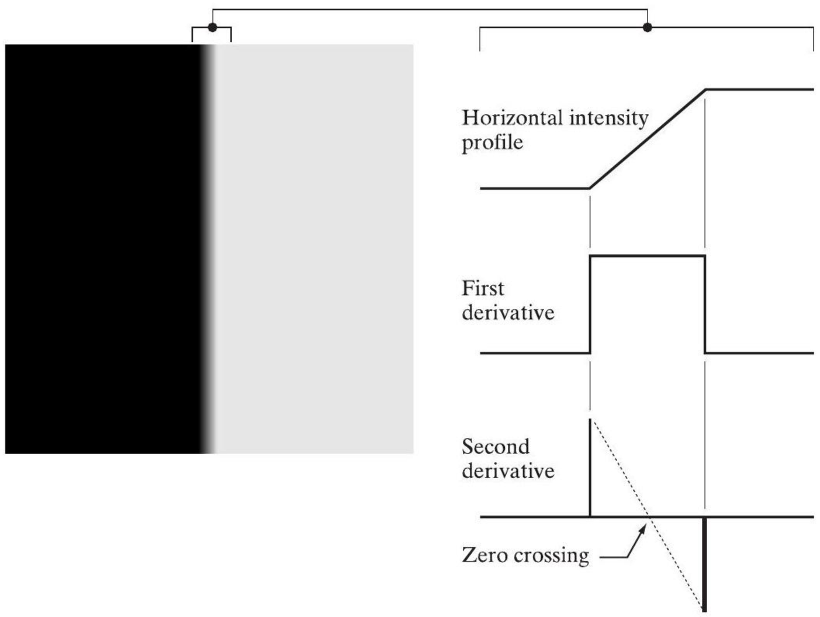


Figure 4.8: (a) Two regions of constant intensity separated by an ideal vertical ramp edge. (b) Detail near the edge, showing a horizontal intensity profile, together with its first and second derivatives.

in an image. We simply define a profile perpendicular to the edge direction at any desired point and interpret the results in the same manner as for the vertical edge just discussed.

We conclude this section by noting that there are three fundamental steps performed in edge detection:

1. *Image smoothing for noise reduction.* The need for this step is amply illustrated by the results in the second and third columns of Fig. 4.9.
2. *Detection of edge points.* As mentioned earlier, this is a local operation that extracts from an image all points that are potential candidates to become edge points.
3. *Edge localization.* The objective of this step is to select from the candidate edge points only the points that are true members of the set

of points comprising an edge. The remainder of this section deals with techniques for achieving these objectives.

4.2.5 Basic Edge Detection

As illustrated in the previous section, detecting changes in intensity for the purpose of finding edges can be accomplished using first- or second-order derivatives. We discuss first-order derivatives in this section and work with second-order derivatives in Section 4.2.6.

The image gradient and its properties

The tool of choice for finding edge strength and direction at location (x,y) of an image, f, is the gradient, denoted by ∇f, and defined as the vector

$$\nabla f \equiv \text{grad}(f) \equiv \begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} \tag{4.9}$$

This vector has the important geometrical property that it points in the direction of the greatest rate of change of f at location (x,y).

The *magnitude (length)* of vector ∇f, denoted as $M(x,y)$, where

$$M(x,y) = \text{mag}(\nabla f) = \sqrt{g_x^2 + g_y^2} \tag{4.10}$$

is the *value* of the rate of change in the direction of the gradient vector. Note that g_x, g_y, and $M(x,y)$ are images of the same size as the original, created when x and y are allowed to vary over all pixel locations in f. It is common practice to refer to the latter image as the *gradient image*, or simply as the *gradient* when the meaning is clear. The summation, square, and square root operations are *array operations*.

The *direction* of the gradient vector is given by the angle

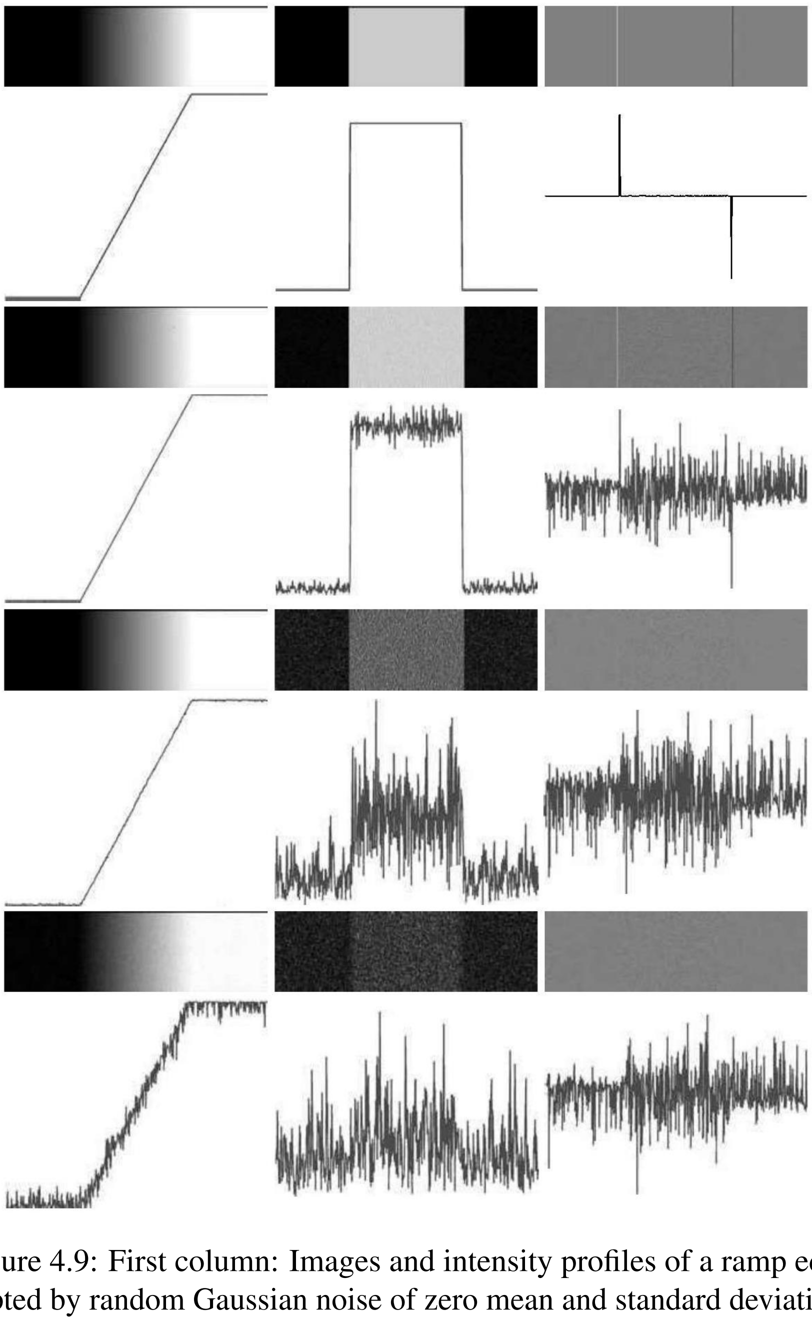


Figure 4.9: First column: Images and intensity profiles of a ramp edge corrupted by random Gaussian noise of zero mean and standard deviations of $0.0, 0.1, 1.0$, and 10.0 intensity levels, respectively. Second column: First-derivative images and intensity profiles. Third column: Second-derivative images and intensity profiles.

$$\alpha(x,y) = \tan^{-1}\left[\frac{g_y}{g_x}\right] \tag{4.11}$$

measured with respect to the x-axis. As in the case of the gradient image, $\alpha(x,y)$ also is an image of the same size as the original created by the array division of image g_y by image g_x. The direction of an edge at an arbitrary point (x,y) is *orthogonal* to the direction, $\alpha(x,y)$, of the gradient vector at the point.

Figure 4.11(a) shows a zoomed section of an image containing a straight edge segment. Each square shown corresponds to a pixel, and we are interested in obtaining the strength and direction of the edge at the point highlighted with a box. The pixels in gray have value 0 and the pixels in white have value 1 . We show following this example that an approach for computing the derivatives in the x - and y-directions using a 3×3 neighborhood centered about a point consists simply of subtracting the pixels in the top row of the neighborhood from the pixels in the bottom row to obtain the partial derivative in the x-direction. Similarly, we subtract the pixels in the left column from the pixels in the right column to obtain the partial derivative in the y-direction. It then follows, using these differences as our estimates of the partials, that $\partial f/\partial x = -2$ and $\partial f/\partial y = 2$ at the point in question. Then,

$$\nabla f = \begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} = \begin{bmatrix} -2 \\ 2 \end{bmatrix}$$

from which we obtain $M(x,y) = 2\sqrt{2}$ at that point. Similarly, the direction of the gradient vector at the same point follows from Eq. (4.11): $\alpha(x,y) = \tan^{-1}(g_y/g_x) = -45^\circ$, which is the same as 135° measured in the positive direction with respect to the x-axis. Figure 4.11(b) shows the gradient vector and its direction angle.

Figure 4.11(c) illustrates the important fact mentioned earlier that the edge at a point is orthogonal to the gradient vector at that point. So the direction angle of the edge in this example is $\alpha - 90^\circ = 45^\circ$. All edge points in Fig.

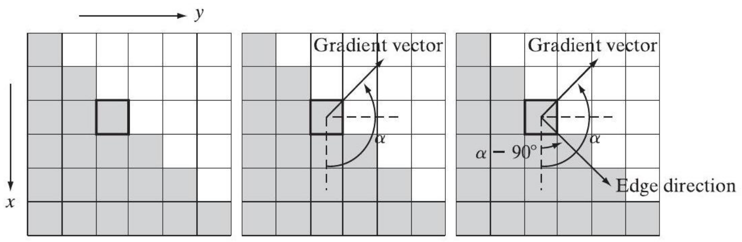


Figure 4.10: Using the gradient to determine edge strength and direction at a point. Note that the edge is perpendicular to the direction of the gradient vector at the point where the gradient is computed. Each square in the figure represents one pixel.

4.11(a) have the same gradient, so the entire edge segment is in the same direction. The gradient vector sometimes is called the edge normal. When the vector is normalized to unit length by dividing it by its magnitude, the resulting vector is commonly referred to as the edge unit normal.

Gradient operators

Obtaining the gradient of an image requires computing the partial derivatives $\partial f/\partial x$ and $\partial f/\partial y$ at every pixel location in the image. We are dealing with digital quantities, so a digital approximation of the partial derivatives over a neighborhood about a point is required. From Section 4.2.1 we know that

$$g_x = \frac{\partial f(x,y)}{\partial x} = f(x+1,y) - f(x,y) \tag{4.12}$$

and

$$g_y = \frac{\partial f(x,y)}{\partial y} = f(x,y+1) - f(x,y) \tag{4.13}$$

These two equations can be implemented for all pertinent values of x and y by filtering $f(x,y)$ with the 1-D masks in Fig. 4.11.

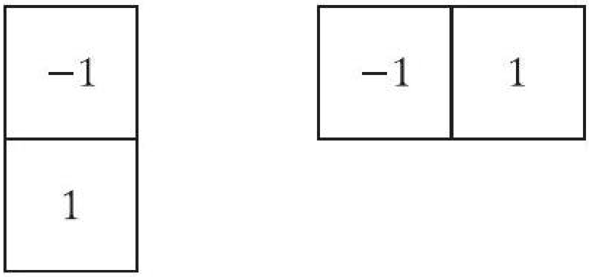


Figure 4.11: One-dimensional masks used to implement Eqs. (4.12) & (4.13).

When diagonal edge direction is of interest, we need a 2-D mask. The *Roberts cross-gradient operators* are one of the earliest attempts to use 2-D masks with a diagonal preference. Consider the 3×3 region in Fig.4.12(a). The Roberts operators are based on implementing the diagonal differences

$$g_x = \frac{\partial f}{\partial x} = (z_9 - z_5) \tag{4.14}$$

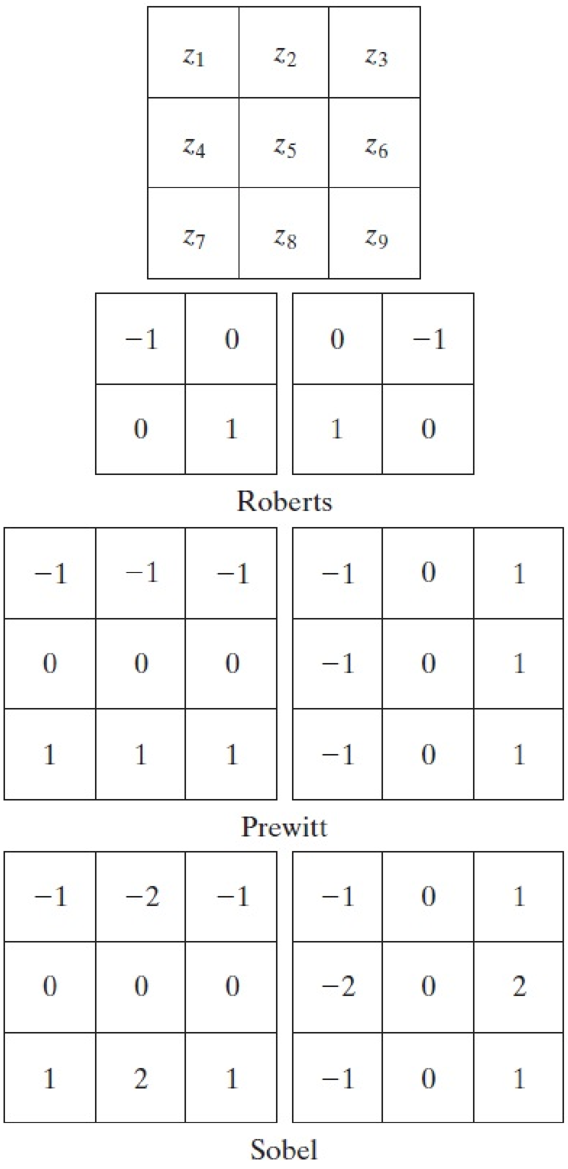
and

$$g_y = \frac{\partial f}{\partial y} = (z_8 - z_6) \tag{4.15}$$

These derivatives can be implemented by filtering an image with the masks in Figs. 4.12(b) and (c).

Masks of size 2×2 are simple conceptually, but they are not as useful for computing edge direction as masks that are symmetric about the center point, the smallest of which are of size 3×3. These masks take into account the nature of the data on opposite sides of the center point and thus carry more information regarding the direction of an edge. The simplest digital approximations to the partial derivatives using masks of size 3×3 are given by

$$g_x = \frac{\partial f}{\partial x} = (z_7 + z_8 + z_9) - (z_1 + z_2 + z_3) \tag{4.16}$$



z_1	z_2	z_3
z_4	z_5	z_6
z_7	z_8	z_9

−1	0		0	−1
0	1		1	0

Roberts

−1	−1	−1		−1	0	1
0	0	0		−1	0	1
1	1	1		−1	0	1

Prewitt

−1	−2	−1		−1	0	1
0	0	0		−2	0	2
1	2	1		−1	0	1

Sobel

Figure 4.12: A 3×3 region of an image (the z 's are intensity values) and various masks used to compute the gradient at the point labeled z_5.

and

$$g_y = \frac{\partial f}{\partial y} = (z_3 + z_6 + z_9) - (z_1 + z_4 + z_7) \tag{4.17}$$

In these formulations, the difference between the third and first rows of the 3×3 region approximates the derivative in the x-direction, and the difference between the third and first columns approximate the derivate in the y-direction. Intuitively, we would expect these approximations to be more accurate than the approximations obtained using the Roberts operators. Equations (4.16) and (4.17) can be implemented over an entire image by filtering f with the two masks in Figs. 4.12(d) and (e). These masks are called the *Prewitt operators*.

A slight variation of the preceding two equations uses a weight of 2 in the center coefficient:

$$g_x = \frac{\partial f}{\partial x} = (z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3) \tag{4.18}$$

and

$$g_y = \frac{\partial f}{\partial y} = (z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7) \tag{4.19}$$

It can be shown that using a 2 in the center location provides image smoothing. Figures 4.12(f) and (g) show the masks used to implement Eqs. (4.18) and (4.19). These masks are called the *Sobel operators*.

The Prewitt masks are simpler to implement than the Sobel masks, but, the slight computational difference between them typically is not an issue. The fact that the Sobel masks have better noise-suppression (smoothing) characteristics makes them preferable because, as mentioned in the previous section, noise suppression is an important issue when dealing with derivatives. Note that the coefficients of all the masks in Fig. 4.12 sum to zero, thus giving a response of zero in areas of constant intensity, as expected of a derivative operator.

0	1	1	−1	−1	0
−1	0	1	−1	0	1
−1	−1	0	0	1	1

Prewitt

0	1	2	−2	−1	0
−1	0	1	−1	0	1
−2	−1	0	0	1	2

Sobel

Figure 4.13: Prewitt and Sobel masks for detecting diagonal edges.

The masks just discussed are used to obtain the gradient components g_x and g_y at every pixel location in an image. These two partial derivatives are then used to estimate edge strength and direction. Computing the magnitude of the gradient requires that g_x and g_y be combined in the manner shown in Eq. (4.10). However, this implementation is not always desirable because of the computational burden required by squares and square roots. An approach used frequently is to approximate the magnitude of the gradient by absolute values:

$$M(x,y) \approx |g_x| + |g_y| \tag{4.20}$$

This equation is more attractive computationally, and it still preserves relative changes in intensity levels. The price paid for this advantage is that the resulting filters will not be isotropic (invariant to rotation) in general. However, this is not an issue when masks such as the Prewitt and Sobel masks are used to compute g_x and g_y, because these masks give isotropic results only for vertical and horizontal edges. Results would be isotropic only for edges in those two directions, regardless of which of the two equations is used. In addition, Eqs. (4.10) and (4.20) give identical results for vertical and horizontal edges when the Sobel or Prewitt masks are used.

It is possible to modify the 3×3 masks in Fig. 4.12 so that they have their strongest responses along the diagonal directions. Figure 4.13 shows the two additional Prewitt and Sobel masks needed for detecting edges in the diagonal directions.

Illustration of the 2-D gradient magnitude and angle.

Figure 4.14 illustrates the absolute value response of the two components of the gradient, $|g_x|$ and $|g_y|$, as well as the gradient image formed from the sum of these two components. The directionality of the horizontal and vertical components of the gradient is evident in Figs. 4.14(b) and (c). Note, for example, how strong the roof tile, horizontal brick joints, and horizontal segments of the windows are in Fig. 4.14(b) compared to other edges. By contrast, Fig. 4.14(c) favors features such as the vertical components of the façade and windows. It is common terminology to use the term edge map when referring to an image whose principal features are edges, such as gradient magnitude images. The intensities of the image in Fig. 4.14(a) were scaled to the range $[0, 1]$. We use values in this range to simplify parameter selection in the various methods for edge detection discussed in this section.

Figure 4.15 shows the gradient angle image computed using Eq. (4.11). In general, angle images are not as useful as gradient magnitude images for edge detection, but they do complement the information extracted from an image using the magnitude of the gradient. For instance, the constant intensity areas in Fig. 4.14(a), such as the front edge of the sloping roof and top horizontal bands of the front wall, are constant in Fig. 4.15, indicating that the gradient vector direction at all the pixel locations in those regions is the same.

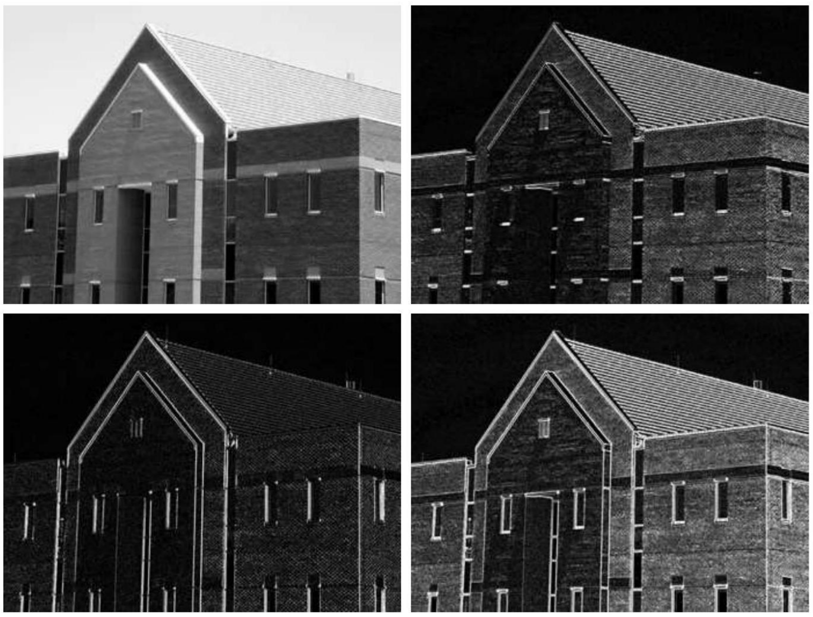


Figure 4.14: (a) Original image of size 834×1114 pixels, with intensity values scaled to the range $[0,1]$. (b) $|g_x|$, the component of the gradient in the x-direction, obtained using the Sobel mask in Fig. 4.12(f) to filter the image. (c) $|g_y|$, obtained using the mask in Fig. 4.12(g). (d) The gradient image, $|g_x| + |g_y|$.

The original image in Fig. 4.14(a) is of reasonably high resolution (834×1114 pixels), and at the distance the image was acquired, the contribution made to image detail by the wall bricks is significant. This level of fine detail often is undesirable in edge detection because it tends to act as noise, which is enhanced by derivative computations and thus complicates detection of the principal edges in an image. One way to reduce fine detail is to smooth the image. Figure 4.16 shows the same sequence of images as in Fig. 4.14, but with the original image smoothed first using a 5×5 averaging filter (see Section 3.5 regarding smoothing filters). The response of each mask now shows almost no contribution due to the bricks, with the results being dominated mostly by the principal edges.

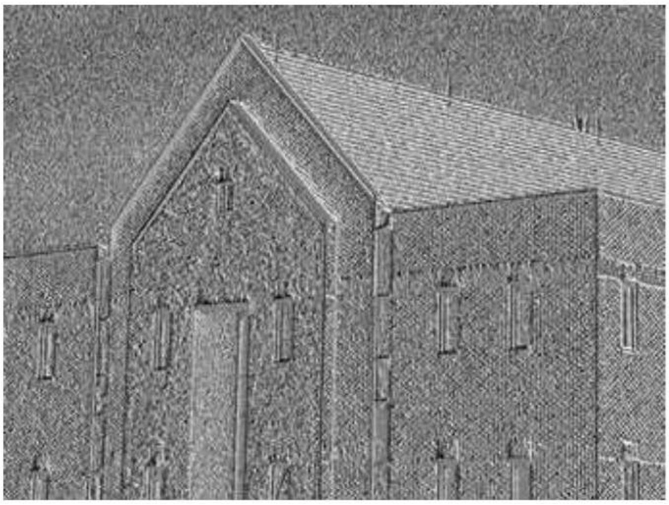


Figure 4.15: Gradient angle image computed using Eq. (4.11). Areas of constant intensity in this image indicate that the direction of the gradient vector is the same at all the pixel locations in those regions.

It is evident in Figs. 4.14 and 4.16 that the horizontal and vertical Sobel masks do not differentiate between edges oriented in the $\pm 45^\circ$ directions. If it is important to emphasize edges along the diagonal directions, then one of the masks in Fig. 4.13 should be used. Figures 4.17(a) and (b) show the absolute responses of the 45° and -45° Sobel masks, respectively. The stronger diagonal response of these masks is evident in these figures. Both diagonal masks have similar response to horizontal and vertical edges but, as expected, their response in these directions is weaker than the response of the horizontal and vertical masks, as discussed earlier.

Combining the gradient with thresholding

The results in Fig. 4.16 show that edge detection can be made more selective by smoothing the image prior to computing the gradient. Another approach aimed at achieving the same basic objective is to threshold the gradient image. For example, Fig. 4.18(a) shows the gradient image from Fig. 4.14(d)

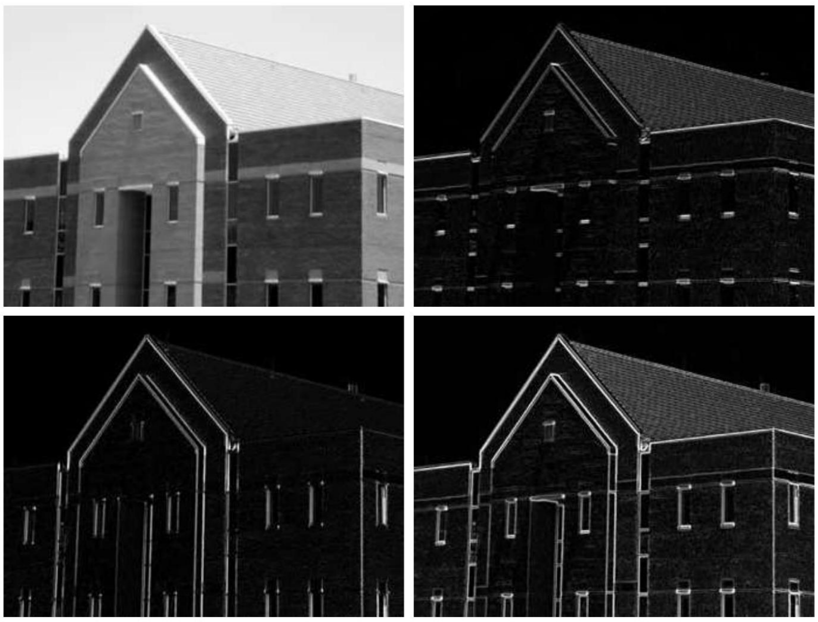


Figure 4.16: Same sequence as in Fig. 4.14, but with the original image smoothed using a 5×5 averaging filter prior to edge detection.

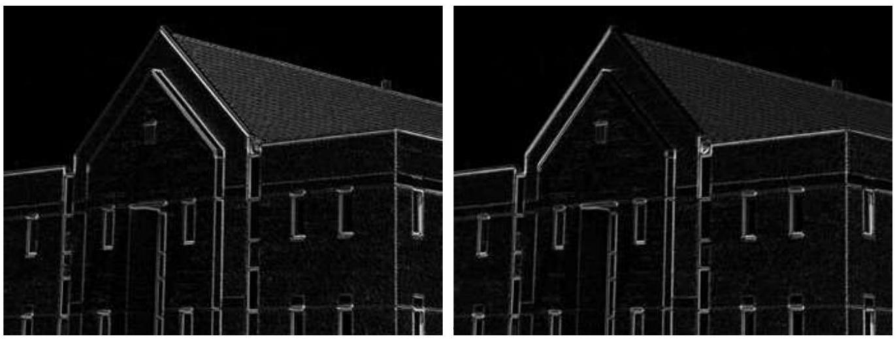


Figure 4.17: Diagonal edge detection. (a) Result of using the mask in Fig. 4.12 (b) Result of using the mask in Fig. 4.12(d).The input image in both cases was Fig. 4.16(a).

thresholded, in the sense that pixels with values greater than or equal to 33% of the maximum value of the gradient image are shown in white, while pixels

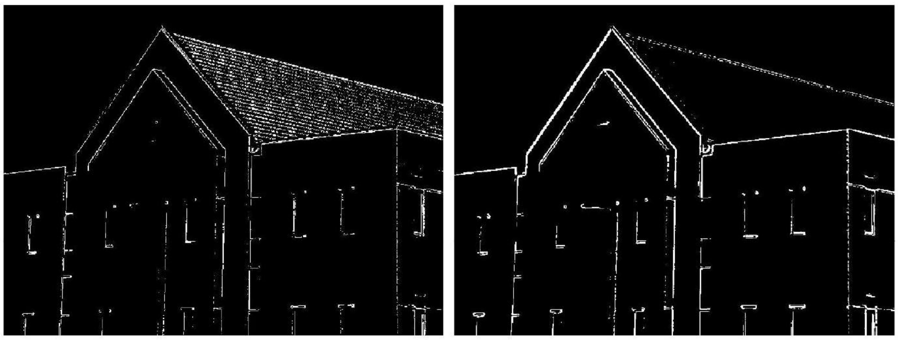


Figure 4.18: (a) Thresholded version of the image in Fig. 4.14(d), with the threshold selected as 33% of the highest value in the image; this threshold was just high enough to eliminate most of the brick edges in the gradient image. (b) Thresholded version of the image in Fig. 4.16(d), obtained using a threshold equal to 33% of the highest value in that image.

below the threshold value are shown in black. Comparing this image with Fig. 4.16(d), we see that there are fewer edges in the thresholded image, and that the edges in this image are much sharper (see, for example, the edges in the roof tile). On the other hand, numerous edges, such as the 45° line defining the far edge of the roof, are broken in the thresholded image.

When interest lies both in highlighting the principal edges and on maintaining as much connectivity as possible, it is common practice to use both smoothing and thresholding. Figure 4.18(b) shows the result of thresholding Fig. 4.16(d), which is the gradient of the smoothed image. This result shows a reduced number of broken edges; for instance, compare the 45° edges in Figs. 4.18(a) and (b). Of course, edges whose intensity values were severely attenuated due to blurring (e.g., the edges in the tile roof) are likely to be totally eliminated by thresholding.

4.2.6 More Advanced Techniques for Edge Detection

The edge-detection methods discussed in the previous section are based simply on filtering an image with one or more masks, with no provisions being made for edge characteristics and noise content. In this section, we discuss more advanced techniques that make an attempt to improve on simple edge detection methods by taking into account factors such as image noise and the nature of edges themselves.

The Marr-Hildreth edge detector

One of the earliest successful attempts at incorporating more sophisticated analysis into the edge-finding process is attributed to Marr and Hildreth. Edge-detection methods in use at the time were based on using small operators (such as the Sobel masks), as discussed in the previous section. Marr and Hildreth argued (1) that intensity changes are not independent of image scale and so their detection requires the use of operators of different sizes; and (2) that a sudden intensity change will give rise to a peak or trough in the first derivative or, equivalently, to a zero crossing in the second derivative.

These ideas suggest that an operator used for edge detection should have two salient features. First and foremost, it should be a differential operator capable of computing a digital approximation of the first or second derivative at every point in the image. Second, it should be capable of being "tuned" to act at any desired scale, so that large operators can be used to detect blurry edges and small operators to detect sharply focused fine detail.

Marr and Hildreth argued that the most satisfactory operator fulfilling these conditions is the filter $\nabla^2 G$ where, ∇^2 is the Laplacian operator, $\left(\partial^2/\partial x^2 + \partial^2/\partial y^2\right)$, and G is the 2-D Gaussian function

$$G(x,y) = e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{4.21}$$

with standard deviation σ (sometimes σ is called the space constant). To find an expression for $\nabla^2 G$ we perform the following differentiations:

$$\nabla^2 G(x,y) = \frac{\partial^2 G(x,y)}{\partial x^2} + \frac{\partial^2 G(x,y)}{\partial y^2} \tag{4.22}$$

$$= \frac{\partial}{\partial x}\left[\frac{-x}{\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}\right] + \frac{\partial}{\partial y}\left[\frac{-y}{\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}\right] \tag{4.23}$$

$$= \left[\frac{x^2}{\sigma^4} - \frac{1}{\sigma^2}\right] e^{-\frac{x^2+y^2}{2\sigma^2}} + \left[\frac{y^2}{\sigma^4} - \frac{1}{\sigma^2}\right] e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{4.24}$$

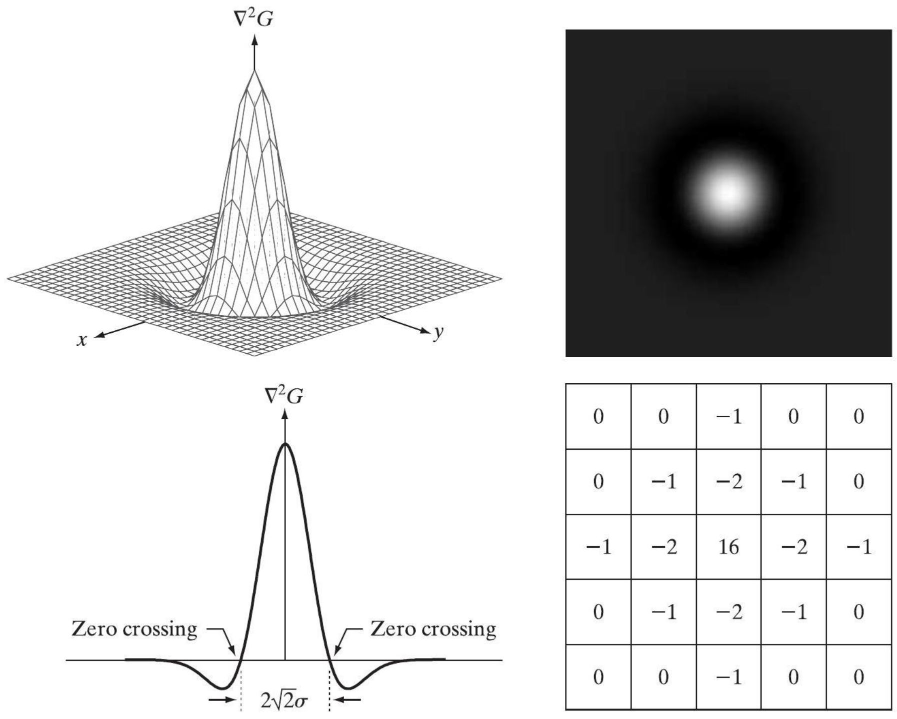
Collecting terms gives the final expression:

$$\nabla^2 G(x,y) = \left[\frac{x^2 + y^2 - 2\sigma^2}{\sigma^4}\right] e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{4.25}$$

This expression is called the Laplacian of a Gaussian (LoG).

Figures 4.19(a) through (c) show a 3-D plot, image, and cross section of the negative of the LoG function (note that the zero crossings of the LoG occur at $x^2 + y^2 = 2\sigma^2$, which defines a circle of radius $\sqrt{2}\sigma$ centered on the origin). Because of the shape illustrated in Fig. 4.19(a), the LoG function sometimes is called the Mexican hat operator. Figure 4.19(d) shows a 5×5 mask that approximates the shape in Fig. 4.19(a) (in practice we would use the negative of this mask). This approximation is not unique. Its purpose is to capture the essential shape of the LoG function; in terms of Fig. 4.19(a), this means a positive, central term surrounded by an adjacent, negative region whose values increase as a function of distance from the origin, and a zero outer region. The coefficients must sum to zero so that the response of the mask is zero in areas of constant intensity.

Masks of arbitrary size can be generated by sampling Eq. (4.25) and scaling the coefficients so that they sum to zero. A more effective approach for generating a LoG filter is to sample Eq. (4.21) to the desired $n \times n$ size and then convolve the resulting array with a Laplacian mask, such as the mask in Fig. 4.4(a). Because convolving an image array with a mask whose



0	0	−1	0	0
0	−1	−2	−1	0
−1	−2	16	−2	−1
0	−1	−2	−1	0
0	0	−1	0	0

Figure 4.19: (a) Three-dimensional plot of the negative of the LoG. (b) Negative of the LoG displayed as an image. (c) Cross section of (a) showing zero crossings. (d) 5×5 mask approximation to the shape in (a). The negative of this mask would be used in practice.

coefficients sum to zero yields a result whose elements also sum to zero, this approach automatically satisfies the requirement that the sum of the LoG filter coefficients be zero. We discuss the issue of selecting the size of LoG filter later in this section.

There are two fundamental ideas behind the selection of the operator $\nabla^2 G$. First, the Gaussian part of the operator blurs the image, thus reducing the intensity of structures (including noise) at scales much smaller than σ. The Gaussian function is smooth in both the spatial and frequency domains, and is thus less likely to introduce artifacts (e.g., ringing) not present in the original image. The other idea concerns ∇^2, the second derivative part of the filter. Although first derivatives can be used for detecting abrupt changes in intensity, they are directional operators. The Laplacian, on the other hand, has the important advantage of being isotropic (invariant to rotation), which not only corresponds to characteristics of the human visual system but also responds equally to changes in intensity in any mask direction, thus avoiding having to use multiple masks to calculate the strongest response at any point in the image.

The Marr-Hildreth algorithm consists of convolving the LoG filter with an input image, $f(x,y)$,

$$g(x,y) = \left[\nabla^2 G(x,y)\right] \star f(x,y) \tag{4.26}$$

and then finding the zero crossings of $g(x,y)$ to determine the locations of edges in $f(x,y)$. Because these are linear processes, Eq. (4.26) can be written also as

$$g(x,y) = \nabla^2 [G(x,y) \star f(x,y)] \tag{4.27}$$

indicating that we can smooth the image first with a Gaussian filter and then compute the Laplacian of the result. These two equations give identical results.

The Marr-Hildreth edge-detection algorithm may be summarized as follows:

1. Filter the input image with an $n \times n$ Gaussian lowpass filter obtained by sampling Eq. (4.21).
2. Compute the Laplacian of the image resulting from Step 1 using, for example, the 3×3 mask in Fig. 4.4(a). [Steps 1 and 2 implement Eq. (4.27).]
3. Find the zero crossings of the image from Step 2.

To specify the size of the Gaussian filter, recall that about 99.7% of the volume under a 2-D Gaussian surface lies between $\pm 3\sigma$ about the mean.

Thus, as a rule of thumb, the size of an $n \times n$ LoG discrete filter should be such that n is the smallest odd integer greater than or equal to 6σ. Choosing a filter mask smaller than this will tend to "truncate" the LoG function, with the degree of truncation being inversely proportional to the size of the mask; using a larger mask would make little difference in the result.

One approach for finding the zero crossings at any pixel, p, of the filtered image, $g(x,y)$, is based on using a 3×3 neighborhood centered at p. A zero crossing at p implies that the signs of at least two of its opposing neighboring pixels must differ. There are four cases to test: left/right, up/down, and the two diagonals. If the values of $g(x,y)$ are being compared against a threshold (a common approach), then not only must the signs of opposing neighbors be different, but the absolute value of their numerical difference must also exceed the threshold before we can call p a zero-crossing pixel.

Zero crossings are the key feature of the Marr-Hildreth edge-detection method. The approach discussed in the previous paragraph is attractive because of its simplicity of implementation and because it generally gives good results. If the accuracy of the zero-crossing locations found using this method is inadequate in a particular application, then a technique proposed by Huertas and Medioni for finding zero crossings with subpixel accuracy can be employed.

A procedure used sometimes to take into account the fact mentioned earlier that intensity changes are scale dependent is to filter an image with various values of σ. The resulting zero-crossings edge maps are then combined by keeping only the edges that are common to all maps. This approach can yield useful information, but, due to its complexity, it is used in practice mostly as a design tool for selecting an appropriate value of σ to use with a single filter.

Marr and Hildreth noted that it is possible to approximate the LoG filter in Eq. (4.25) by a difference of Gaussians (DoG):

$$\mathrm{DoG}(x,y) = \frac{1}{2\pi\sigma_1^2} e^{-\frac{x^2+y^2}{2\sigma_1^2}} - \frac{1}{2\pi\sigma_2^2} e^{-\frac{x^2+y^2}{2\sigma_2^2}} \tag{4.28}$$

with $\sigma_1 > \sigma_2$. Experimental results suggest that certain "channels" in the human vision system are selective with respect to orientation and frequency, and can be modeled using Eq. (4.28) with a ratio of standard deviations of 1.75:1. Marr and Hildreth suggested that using the ratio 1.6:1 preserves the basic characteristics of these observations and also provides a closer "engineering" approximation to the LoG function. To make meaningful comparisons between the LoG and DoG, the value of σ for the LoG must be selected as in the following equation so that the LoG and DoG have the same zero crossings:

$$\sigma^2 = \frac{\sigma_1^2\sigma_2^2}{\sigma_1^2 - \sigma_2^2} \ln\left[\frac{\sigma_1^2}{\sigma_2^2}\right] \tag{4.29}$$

Although the zero crossings of the LoG and DoG will be the same when this value of σ is used, their amplitude scales will be different. We can make them compatible by scaling both functions so that they have the same value at the origin.

Both the LoG and the DoG filtering operations can be implemented with 1-D convolutions instead of using 2-D convolutions directly. For an image of size $M \times N$ and a filter of size $n \times n$ doing so reduces the number of multiplications and additions for each convolution from being proportional to n^2MN for 2-D convolutions to being proportional to nMN for 1-D convolutions.This implementation difference is significant. For example, if $n = 25$, a 1-D implementation will require on the order of 12 times fewer multiplication and addition operations than using 2-D convolution.

4.3 Summary

Image segmentation is an essential preliminary step in most automatic pictorial pattern recognition and scene analysis applications. As indicated in the previous sections, the choice of one segmentation technique over another is dictated mostly by the peculiar characteristics of the problem being considered. The methods discussed in this chapter, although far from exhaustive, are representative of techniques commonly used in practice.

Image segmentation is typically used to locate objects and boundaries (lines, curves, etc.) in images. More precisely, image segmentation is the process of assigning a label to every pixel in an image such that pixels with the same label share certain characteristics.