

8.1 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; 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

$$R = 1 - \frac{1}{C} \quad (8-1)$$

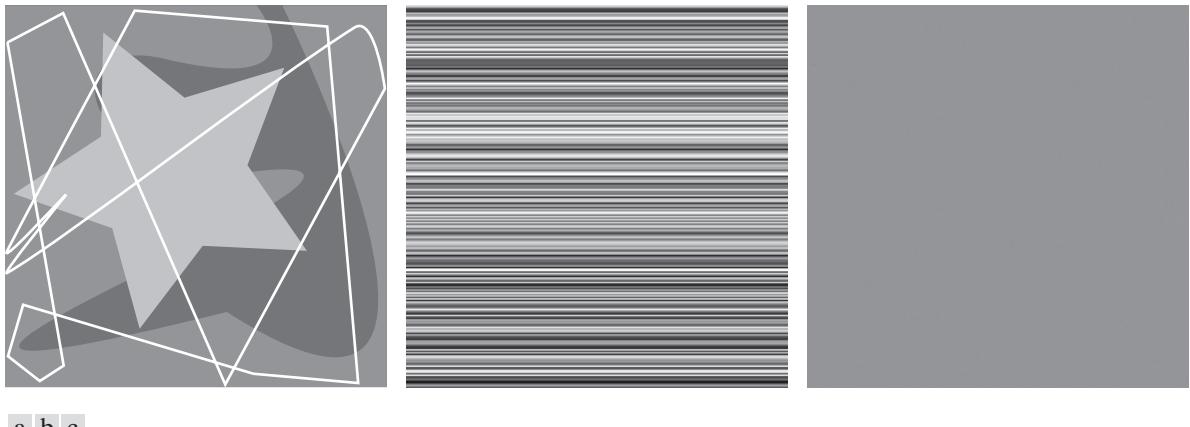
where C , commonly called the *compression ratio*, is defined as

$$C = \frac{b}{b'} \quad (8-2)$$

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. (8-2) usually is the number of bits needed to represent an image as a 2-D array of intensity values. The 2-D intensity arrays introduced in Section 2.4 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 upon 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 upon 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.



a b c

FIGURE 8.1 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.)

The computer-generated images in Figs. 8.1(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.

CODING REDUNDANCY

In Chapter 3, we developed techniques for image enhancement by histogram processing, assuming that the intensity values of an image are random quantities. In this section, we will 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 $p_r(r_k)$. As in Section 3.3,

$$p_r(r_k) = \frac{n_k}{MN} \quad k = 0, 1, 2, \dots, L - 1 \quad (8-3)$$

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

$$L_{\text{avg}} = \sum_{k=0}^{L-1} l(r_k)p_r(r_k) \quad (8-4)$$

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. (8-4) 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 8.1: A simple illustration of variable-length coding.

The computer-generated image in Fig. 8.1(a) has the intensity distribution shown in the second column of Table 8.1. If a natural 8-bit binary code (denoted as code 1 in Table 8.1) is used to represent its four 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 . On the other hand, if the scheme designated as code 2 in Table 8.1 is used, the average length of the encoded pixels is, in accordance with Eq. (8-4),

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

The total number of bits needed to represent the entire image is $MNL_{\text{avg}} = 256 \times 56 \times 1.81$, or 118,621. From Eqs. (8-2) and (8-1), 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 $l_2(128) = 1$], while r_{255} (its least probable occurring intensity) is assigned the 3-bit code word 001 [of length $l_2(255) = 3$]. Note that the best *fixed-length code* that can be assigned to the intensities of the image in Fig. 8.1(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 the objects being depicted are much larger than the picture elements. The natural consequence is that,

[†]A *natural* binary code is one in which each event or piece of information to be encoded (such as intensity value) is assigned one of 2^m codes from an m -bit binary counting sequence.

TABLE 8.1

Example of variable-length coding.

r_k	$p_r(r_k)$	Code 1	$l_1(r_k)$	Code 2	$l_2(r_k)$
$r_{87} = 87$	0.25	01010111	8	01	2
$r_{128} = 128$	0.47	01010111	8	1	1
$r_{186} = 186$	0.25	01010111	8	000	3
$r_{255} = 255$	0.03	01010111	8	001	3
r_k for $k = 87, 128, 186, 255$	0	—	8	—	0

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. (8-4), and resulting in coding redundancy.

SPATIAL AND TEMPORAL REDUNDANCY

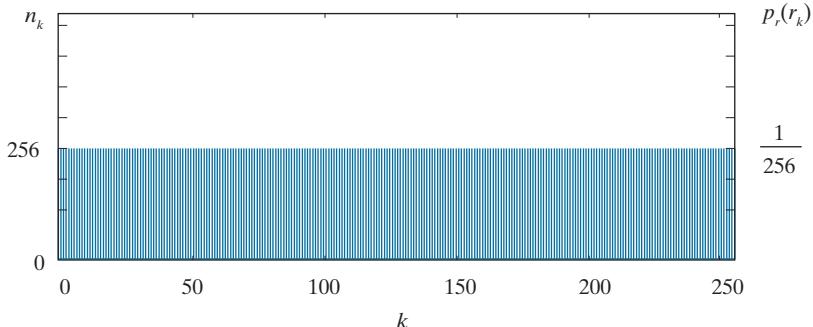
Consider the computer-generated collection of constant intensity lines in Fig. 8.1(b). In the corresponding 2-D intensity array:

1. All 256 intensities are equally probable. As Fig. 8.2 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.

The first observation tells us that the image in Fig. 8.1(b) (when represented as a conventional 8-bit intensity array) cannot be compressed by variable-length coding alone. Unlike the image of Fig. 8.1(a) and Example 8.1, whose histogram was *not* uniform, a fixed-length 8-bit code in this case minimizes Eq. (8-4). Observations 2 and 3 reveal a significant spatial redundancy that can be eliminated by representing the image in Fig. 8.1(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

FIGURE 8.2

The intensity histogram of the image in Fig. 8.1(b).



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-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 *reversible* 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*.

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. 8.1(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.

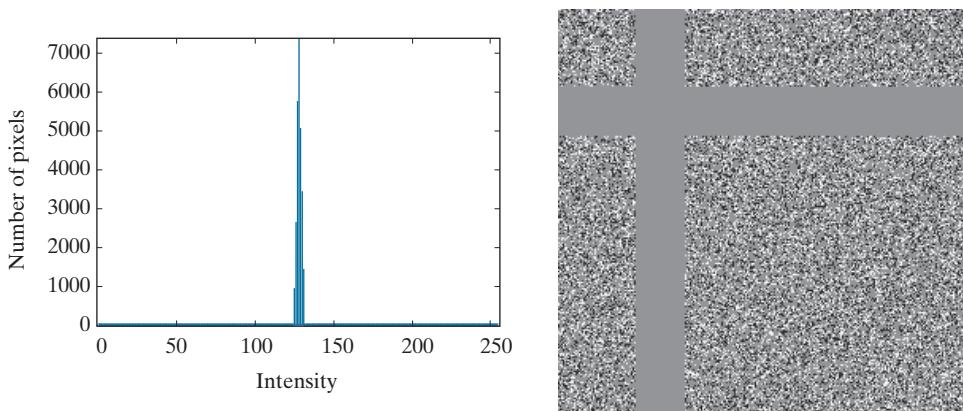
Figure 8.3(a) shows the histogram of the image in Fig. 8.1(c). Note that there are several intensity values (125 through 131) actually present. The human visual system averages these intensities, perceives only the average value, then ignores the small changes in intensity that are present in this case. Figure 8.3(b), a histogram-equalized version of the image in Fig. 8.1(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. 8.1(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 this section by noting that the redundancy examined here is fundamentally different from the redundancies discussed in the previous two sections. 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

a b

FIGURE 8.3

(a) Histogram of the image in Fig. 8.1(c) and (b) a histogram equalized version of the image.



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 (see Section 2.4). Because information is lost, quantization is an irreversible operation.

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: 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 which 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

$$I(E) = \log \frac{1}{P(E)} = -\log P(E) \quad (8-5)$$

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. (8-5) 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.

Consult the book website for a brief review of information and probability theory.

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

$$H = -\sum_{j=1}^J P(a_j) \log P(a_j) \quad (8-6)$$

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

Equation (8-6) is for zero-memory sources with J source symbols. Equation (8-7) uses probability estimates for the $L - 1$ intensity values in an image.

$$\tilde{H} = -\sum_{k=0}^{L-1} p_r(r_k) \log_2 p_r(r_k) \quad (8-7)$$

where variables L , r_k , and $p_r(r_k)$ are as defined earlier and in Section 3.3. Because the base 2 logarithm is used, Eq. (8-7) 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 \tilde{H} bits/pixel.

EXAMPLE 8.2: Image entropy estimates.

The entropy of the image in Fig. 8.1(a) can be estimated by substituting the intensity probabilities from Table 8.1 into Eq. (8-7):

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

In a similar manner, the entropies of the images in Fig. 8.1(b) and (c) can be shown to be 8 bits/pixel and 1.566 bits/pixel, respectively. Note that the image in Fig. 8.1(a) appears to have the most visual information, but has almost the lowest computed entropy—1.66 bits/pixel. The image in Fig. 8.1(b) has almost five times the entropy of the image in (a), but appears to have about the same (or less) visual information. The image in Fig. 8.1(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.

Shannon's First Theorem

Recall that the variable-length code in Example 8.1 was able to represent the intensities of the image in Fig. 8.1(a) using only 1.81 bits/pixel. Although this is higher than the 1.6614 bits/pixel entropy estimate from Example 8.2, Shannon’s first theorem, also called the *noiseless coding theorem* (Shannon [1948]), assures us that the

image in Fig. 8.1(a) can be represented with as few as 1.6614 bits/pixel. To prove it in a general way, Shannon looked at representing groups of consecutive source symbols with a single code word (rather than one code word per source symbol), and showed that

$$\lim_{n \rightarrow \infty} \left[\frac{L_{\text{avg},n}}{n} \right] = H \quad (8-8)$$

where $L_{\text{avg},n}$ is the average number of code symbols required to represent all n -symbol groups. In the proof, he defined the *n th extension* of a zero-memory source to be the hypothetical source that produces n -symbol blocks[†] using the symbols of the original source, and computed $L_{\text{avg},n}$ by applying Eq. (8-4) to the code words used to represent the n -symbol blocks. Equation (8-8) tells us that $L_{\text{avg},n}/n$ can be made arbitrarily close to H by encoding infinitely long extensions of the single-symbol source. That is, it is possible to represent the output of a zero-memory source with an average of H information units per source symbol.

If we now return to the idea that an image is a “sample” of the intensity source that produced it, a block of n source symbols corresponds to a group of n adjacent pixels. To construct a variable-length code for n -pixel blocks, the relative frequencies of the blocks must be computed. But the n th extension of a hypothetical intensity source with 256 intensity values has 256^n possible n -pixel blocks. Even in the simple case of $n = 2$, a 65,536 element histogram and up to 65,536 variable-length code words must be generated. For $n = 3$, as many as 16,777,216 code words are needed. So even for small values of n , computational complexity limits the usefulness of the extension coding approach in practice.

Finally, we note that although Eq. (8-7) provides a lower bound on the compression that can be achieved when directly coding statistically independent pixels, it breaks down when the pixels of an image are correlated. Blocks of correlated pixels can be coded with fewer average bits per pixel than the equation predicts. Rather than using source extensions, less correlated descriptors (such as intensity run-lengths) are normally selected and coded without extension. This was the approach used to compress Fig. 8.1(b) in the section on spatial and temporal redundancy. When the output of a source of information depends on a finite number of preceding outputs, the source is called a *Markov source* or *finite memory source*.

FIDELITY CRITERIA

It was noted earlier 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.

[†]The output of the n th extension is an n -tuple of symbols from the underlying *single-symbol source*. It was considered a *block random variable* in which the probability of each n -tuple is the product of the probabilities of its individual symbols. The entropy of the n th extension is then n times the entropy of the single-symbol source from which it is derived.

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-squared (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

$$e(x, y) = \hat{f}(x, y) - f(x, y) \quad (8-9)$$

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-squared 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

$$e_{\text{rms}} = \left[\frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} [\hat{f}(x, y) - f(x, y)]^2 \right]^{1/2} \quad (8-10)$$

If $\hat{f}(x, y)$ is considered [by a simple rearrangement of the terms in Eq. (8-9)] to be the sum of the original image $f(x, y)$ and an error or “noise” signal $e(x, y)$, the *mean-squared signal-to-noise ratio* of the output image, denoted SNR_{ms} , can be defined as in Section 5.8:

$$\text{SNR}_{\text{ms}} = \frac{\sum_{x=0}^{M-1} \sum_{y=0}^{N-1} \hat{f}(x, y)^2}{\sum_{x=0}^{M-1} \sum_{y=0}^{N-1} [\hat{f}(x, y) - f(x, y)]^2} \quad (8-11)$$

The rms value of the signal-to-noise ratio, denoted SNR_{rms} , is obtained by taking the square root of Eq. (8-11).

While objective fidelity criteria offer a simple and convenient way to evaluate information loss, decompressed images are often 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 8.2 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 $\{\text{much worse}, \text{worse}, \text{slightly worse}, \text{the same}, \text{slightly better}, \text{better}, \text{much better}\}$, respectively. In either case, the evaluations are based on *subjective fidelity criteria*.

TABLE 8.2

Rating scale of the Television Allocations Study Organization. (Frendendall and Behrend.)

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.
5	Inferior	A very poor image, but you could watch it. Objectionable interference is definitely present.
6	Unusable	An image so bad that you could not watch it.

EXAMPLE 8.3: Image quality comparisons.

Figure 8.4 shows three different approximations of the image in Fig. 8.1(a). Using Eq. (8-10) with Fig. 8.1(a) as $f(x, y)$ and Figs. 8.4(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 criterion), the images are ranked in order of decreasing quality as {(a), (c), (b)}. A subjective evaluation of the images using Table 8.2, however, might yield an *excellent* rating for (a), a *marginal* rating for (b), and an *inferior* or *unusable* rating for (c). Thus, using a subjective fidelity criteria, (b) is ranked ahead of (c).

IMAGE COMPRESSION MODELS

As Fig. 8.5 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 applications, 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.

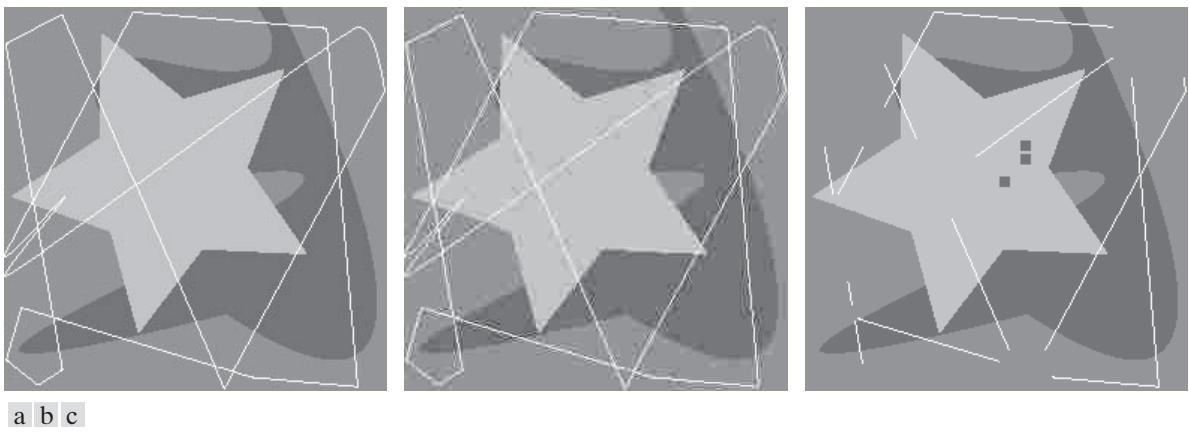
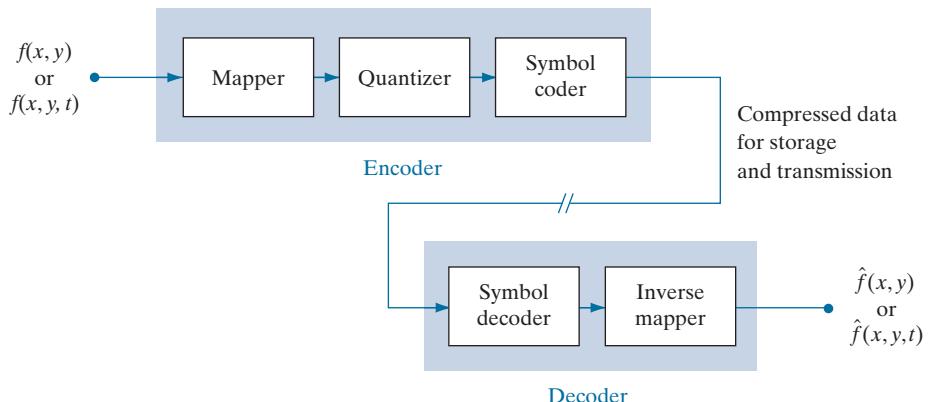


FIGURE 8.4 Three approximations of the image in Fig. 8.1(a).

FIGURE 8.5

Functional block diagram of a general image compression system.



Here, the notation $f(x, \dots)$ is used to denote both $f(x, y)$ and $f(x, y, t)$.

Input image $f(x, \dots)$ 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 $\hat{f}(x, \dots)$ 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 the discrete parameter t specifies time. In general, $\hat{f}(x, \dots)$ may or may not be an exact replica of $f(x, \dots)$. 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. 8.5 is designed to remove the redundancies described in the previous sections 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 directly reduce 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 (see Section 8.9) 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. 8.5 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. As noted earlier, 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 is used to adjust the operation of the quantizer so 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. 8.5 generates a fixed-length 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 in the previous sections.

The Decoding or Decompression Process

The decoder of Fig. 8.5 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.

IMAGE FORMATS, CONTAINERS, AND COMPRESSION STANDARDS

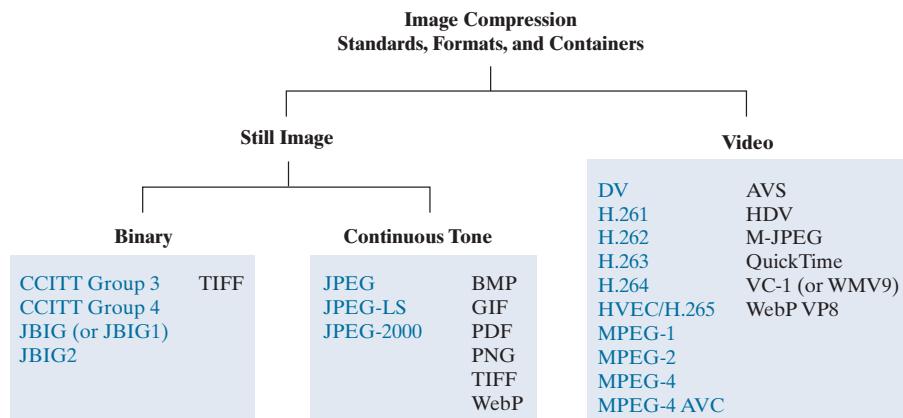
In the context of digital imaging, an *image file format* is a standard way to organize and store image data. It defines how the data is arranged and the type of compression (if any) that is used. An *image container* is similar to a file format, but handles multiple types of image data. Image *compression 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 8.6 lists the most important image compression standards, file formats, and containers in use today, grouped by the type of image handled. The entries in blue 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 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 black, which is used in Fig. 8.6 to denote entries that are not sanctioned by an international standards organization.

Tables 8.3 through 8.5 summarize the standards, formats, and containers listed in Fig. 8.6. Responsible organizations, targeted applications, and key compression methods are identified. The compression methods themselves are the subject of Sections 8.2 through 8.11, where we will describe the principal lossy and error-free compression methods in use today. The focus of these sections is on methods that have proven useful in mainstream binary, continuous-tone still-image, and video compression standards. The standards themselves are used to demonstrate the methods presented. In Tables 8.3 through 8.5, forward references to the relevant sections in which the compression methods are described are enclosed in square brackets.

FIGURE 8.6

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

**TABLE 8.3**

Internationally sanctioned image compression standards. The numbers in brackets refer to sections in this chapter.

Name	Organization	Description
<i>Bi-Level Still Images</i>		
CCITT Group 3	ITU-T	Designed as a facsimile (FAX) method for transmitting binary documents over telephone lines. Supports 1-D and 2-D run-length [8.6] and Huffman [8.2] coding.
CCITT Group 4	ITU-T	A simplified and streamlined version of the CCITT Group 3 standard supporting 2-D run-length coding only.
JBIG or JBIG1	ISO/IEC/ITU-T	A <i>Joint Bi-level Image Experts Group</i> standard for progressive, lossless compression of bi-level images. Continuous-tone images of up to 6 bits/pixel can be coded on a bit-plane basis [8.8]. Context-sensitive arithmetic coding [8.4] is used and an initial low-resolution version of the image can be gradually enhanced with additional compressed data.
JBIG2	ISO/IEC/ITU-T	A follow-on to JBIG1 for bi-level images in desktop, Internet, and FAX applications. The compression method used is content based, with dictionary-based methods [8.7] for text and halftone regions, and Huffman [8.2] or arithmetic coding [8.4] for other image content. It can be lossy or lossless.
<i>Continuous-Tone Still Images</i>		
JPEG	ISO/IEC/ITU-T	A <i>Joint Photographic Experts Group</i> standard for images of photographic quality. Its lossy baseline coding system (most commonly implemented) uses quantized discrete cosine transforms (DCT) on image blocks [8.9], Huffman [8.2], and run-length [8.6] coding. It is one of the most popular methods for compressing images on the Internet.
JPEG-LS	ISO/IEC/ITU-T	A lossless to near-lossless standard for continuous-tone images based on adaptive prediction [8.10], context modeling [8.4], and Golomb coding [8.3].
JPEG-2000	ISO/IEC/ITU-T	A follow-on to JPEG for increased compression of photographic quality images. Arithmetic coding [8.4] and quantized discrete wavelet transforms (DWT) [8.11] are used. The compression can be lossy or lossless.

TABLE 8.4

Internationally sanctioned video compression standards. The numbers in brackets refer to sections in this chapter.

Name	Organization	Description
DV	IEC	<i>Digital Video.</i> A video standard tailored to home and semiprofessional video production applications and equipment, such as electronic news gathering and camcorders. Frames are compressed independently for uncomplicated editing using a DCT-based approach [8.9] similar to JPEG.
H.261	ITU-T	A two-way videoconferencing standard for ISDN (<i>integrated services digital network</i>) lines. It supports non-interlaced 352×288 and 176×144 resolution images, called CIF (<i>Common Intermediate Format</i>) and QCIF (<i>Quarter CIF</i>), respectively. A DCT-based compression approach [8.9] similar to JPEG is used, with frame-to-frame prediction differencing [8.10] to reduce temporal redundancy. A block-based technique is used to compensate for motion between frames.
H.262	ITU-T	See MPEG-2 below.
H.263	ITU-T	An enhanced version of H.261 designed for ordinary telephone modems (i.e., 28.8 Kb/s) with additional resolutions: SQCIF (<i>Sub-Quarter CIF</i> 128×96), 4CIF (704×576) and 16CIF (1408×512).
H.264	ITU-T	An extension of H.261–H.263 for videoconferencing, streaming, and television. It supports prediction differences within frames [8.10], variable block size integer transforms (rather than the DCT), and context adaptive arithmetic coding [8.4].
H.265 MPEG-H HEVC	ISO/IEC	<i>High Efficiency Video Coding</i> (HVEC). An extension of H.264 that includes support for macroblock sizes up to 64×64 and additional intraframe prediction modes, both useful in 4K video applications.
MPEG-1	ISO/IEC	A <i>Motion Pictures Expert Group</i> standard for CD-ROM applications with non-interlaced video at up to 1.5 Mb/s. It is similar to H.261 but frame predictions can be based on the previous frame, next frame, or an interpolation of both. It is supported by almost all computers and DVD players.
MPEG-2	ISO/IEC	An extension of MPEG-1 designed for DVDs with transfer rates at up to 15 Mb/s. Supports interlaced video and HDTV. It is the most successful video standard to date.
MPEG-4	ISO/IEC	An extension of MPEG-2 that supports variable block sizes and prediction differencing [8.10] within frames.
MPEG-4 AVC	ISO/IEC	MPEG-4 Part 10 <i>Advanced Video Coding</i> (AVC). Identical to H.264.

8.2 HUFFMAN CODING

With reference to Tables 8.3–8.5, Huffman codes are used in

- CCITT
- JBIG2
- JPEG
- MPEG-1, 2, 4
- H.261, H.262,
- H.263, H.264

and other compression standards.

One of the most popular techniques for removing coding redundancy is due to Huffman (Huffman [1952]). When coding the symbols of an information source individually, *Huffman coding* yields the smallest possible number of code symbols per source symbol. In terms of Shannon's first theorem (see Section 8.1), the resulting code is optimal for a fixed value of n , subject to the constraint that the source symbols be coded *one at a time*. 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).

TABLE 8.5

Popular image and video compression standards, file formats, and containers not included in Tables 8.3 and 8.4. The numbers in brackets refer to sections in this chapter.

Name	Organization	Description
<i>Continuous-Tone Still Images</i>		
BMP	Microsoft	<i>Windows Bitmap</i> . A file format used mainly for simple uncompressed images.
GIF	CompuServe	<i>Graphic Interchange Format</i> . A file format that uses lossless LZW coding [8.5] for 1- through 8-bit images. It is frequently used to make small animations and short low-resolution films for the Internet.
PDF	Adobe Systems	<i>Portable Document Format</i> . A format for representing 2-D documents in a device and resolution independent way. It can function as a container for JPEG, JPEG-2000, CCITT, and other compressed images. Some PDF versions have become ISO standards.
PNG	World Wide Web Consortium (W3C)	<i>Portable Network Graphics</i> . A file format that losslessly compresses full color images with transparency (up to 48 bits/pixel) by coding the difference between each pixel's value and a predicted value based on past pixels [8.10].
TIFF	Aldus	<i>Tagged Image File Format</i> . A flexible file format supporting a variety of image compression standards, including JPEG, JPEG-LS, JPEG-2000, JBIG2, and others.
WebP	Google	<i>WebP</i> supports lossy compression via WebP VP8 intraframe video compression (see below) and lossless compression using spatial prediction [8.10] and a variant of LZW backward referencing [8.5] and Huffman entropy coding [8.2]. Transparency is also supported.
<i>Video</i>		
AVS	MII	<i>Audio-Video Standard</i> . Similar to H.264 but uses exponential Golomb coding [8.3]. Developed in China.
HDV	Company consortium	<i>High Definition Video</i> . An extension of DV for HD television that uses compression similar to MPEG-2, including temporal redundancy removal by prediction differencing [8.10].
M-JPEG	Various companies	<i>Motion JPEG</i> . A compression format in which each frame is compressed independently using JPEG.
Quick-Time	Apple Computer	A media container supporting DV, H.261, H.262, H.264, MPEG-1, MPEG-2, MPEG-4, and other video compression formats.
VC-1	SMPTE	The most used video format on the Internet. Adopted for HD and <i>Blu-ray</i> high-definition DVDs. It is similar to H.264/AVC, using an integer DCT with varying block sizes [8.9 and 8.10] and context-dependent variable-length code tables [8.2], but no predictions within frames.
WMV9	Microsoft	
WebP VP8	Google	A file format based on block transform coding [8.9] prediction differences within frames and between frames [8.10]. The differences are entropy encoded using an adaptive arithmetic coder [8.4].

The first step in Huffman's approach is to create a series of source reductions by ordering the probabilities of the symbols under consideration, then combining the lowest probability symbols into a single symbol that replaces them in the next source reduction. Figure 8.7 illustrates this process for binary coding (K -ary Huffman codes also can 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. 8.8 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 reached. The final code appears at the far left in Fig. 8.8. The average length of this code is

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

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

FIGURE 8.7

Huffman source reductions.

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

FIGURE 8.8

Huffman code assignment procedure.

Symbol	Probability	Code	Original source				Source reduction				4
			1	2	3	4	1	2	3	4	
a_2	0.4	1	0.4	1	0.4	1	0.6	0			
a_6	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	0.4	1			
a_3	0.06	01010	0.1	0101	0.1	011	0.4	1			
a_5	0.04	01011					0.4	1			

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. 8.8, 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 $a_3a_1a_2a_2a_6$.

EXAMPLE 8.4: Huffman Coding.

The 512×512 8-bit monochrome image in Fig. 8.9(a) has the intensity histogram shown in Fig. 8.9(b). Because the intensities are not equally probable, a MATLAB implementation of Huffman's procedure was used to encode them with 7.428 bits/pixel, including the Huffman code table that is required to reconstruct the original 8-bit image intensities. The compressed representation exceeds the estimated entropy of the image [7.3838 bits/pixel from Eq. (8-7)] by $512^2 \times (7.428 - 7.3838)$ or 11,587 bits—about 0.6%. The resulting compression ratio and corresponding relative redundancy are $C = 8/7.428 = 1.077$, and $R = 1 - (1/1.077) = 0.0715$, respectively. Thus 7.15% of the original 8-bit fixed-length intensity representation was removed as coding redundancy.

When a large number of symbols is to be coded, the construction of an optimal Huffman code is a nontrivial task. For the general case of J source symbols, J symbol probabilities, $J - 2$ source reductions, and $J - 2$ code assignments are required. When source symbol probabilities can be estimated in advance, “near optimal” coding can be achieved with pre-computed Huffman codes. Several popular image compression standards, including the JPEG and MPEG standards discussed in Sections 8.9 and 8.10, specify default Huffman coding tables that have been pre-computed based on experimental data.

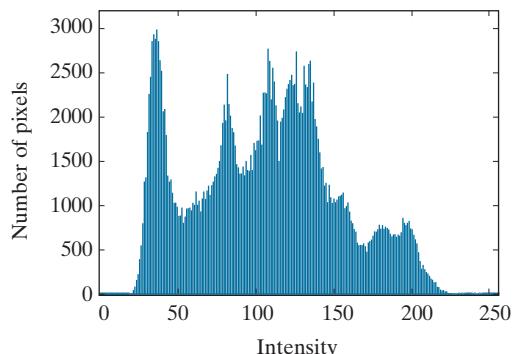
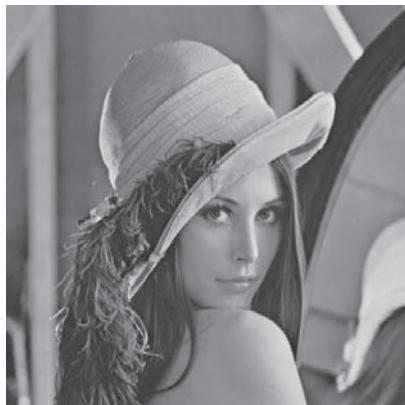
8.3 GOLOMB CODING

With reference to Tables 8.3–8.5, Golomb codes are used in

- JPEG-LS
 - AVS
- compression.

In this section, we consider the coding of nonnegative integer inputs with exponentially decaying probability distributions. Inputs of this type can be optimally encoded (in the sense of Shannon's first theorem) using a family of codes that are computationally simpler than Huffman codes. The codes themselves were first proposed for the representation of nonnegative run lengths (Golomb [1966]). In the discussion

a b

FIGURE 8.9(a) A 512×512 8-bit image and
(b) its histogram.

that follows, the notation $\lfloor x \rfloor$ denotes the largest integer less than or equal to x , $\lceil x \rceil$ means the smallest integer greater than or equal to x , and $x \bmod y$ is the remainder of x divided by y .

Given a nonnegative integer n and a positive integer divisor $m > 0$, the *Golomb code* of n with respect to m , denoted $G_m(n)$, is a combination of the unary code of quotient $\lfloor n/m \rfloor$ and the binary representation of remainder $n \bmod m$. $G_m(n)$ is constructed as follows:

1. Form the unary code of quotient $\lfloor n/m \rfloor$. (The *unary code* of an integer q is defined as q 1's followed by a 0.)
2. Let $k = \lceil \log_2 m \rceil$, $c = 2^k - m$, $r = n \bmod m$, and compute truncated remainder r' such that

$$r' = \begin{cases} r \text{ truncated to } k-1 \text{ bits} & 0 \leq r < c \\ r + c \text{ truncated to } k \text{ bits} & \text{otherwise} \end{cases} \quad (8-12)$$

3. Concatenate the results of Steps 1 and 2.

To compute $G_4(9)$, for example, begin by determining the unary code of the quotient $\lfloor 9/4 \rfloor = \lfloor 2.25 \rfloor = 2$, which is 110 (the result of Step 1). Then let $k = \lceil \log_2 4 \rceil = 2$, $c = 2^2 - 4 = 0$, and $r = 9 \bmod 4$, which in binary is 1001 mod 0100 or 0001. In accordance with Eq. (8-12), r' is then r (i.e., 0001) truncated to 2 bits, which is 01 (the result of Step 2). Finally, concatenate 110 from Step 1 and 01 from Step 2 to get 11001, which is $G_4(9)$.

For the special case of $m = 2^k$, $c = 0$ and $r' = r = n \bmod m$ truncated to k bits in Eq. (8-12) for all n . The divisions required to generate the resulting Golomb codes become binary shift operations, and the computationally simpler codes are called *Golomb-Rice* or *Rice codes* (Rice [1975]). Columns 2, 3, and 4 of Table 8.6 list the G_1 , G_2 , and G_4 codes of the first ten nonnegative integers. Because m is a power of 2 in each case (i.e., $1 = 2^0$, $2 = 2^1$, and $4 = 2^2$), they are the first three Golomb-Rice codes as well. Moreover, G_1 is the unary code of the nonnegative integers because $\lfloor n/1 \rfloor = n$ and $n \bmod 1 = 0$ for all n .

TABLE 8.6

Several Golomb codes for the integers 0–9.

n	$G_1(n)$	$G_2(n)$	$G_4(n)$	$G_{\text{exp}}^0(n)$
0	0	00	000	0
1	10	01	001	100
2	110	100	010	101
3	1110	101	011	11000
4	11110	1100	1000	11001
5	111110	1101	1001	11010
6	1111110	11100	1010	11011
7	11111110	11101	1011	1110000
8	111111110	111100	11000	1110001
9	1111111110	111101	11001	1110010

Keeping in mind that Golomb codes only can be used to represent nonnegative integers, and that there are many Golomb codes to choose from, a key step in their effective application is the selection of divisor m . When the integers to be represented are *geometrically* distributed with a *probability mass function* (PMF)[†]

$$P(n) = (1 - \rho)\rho^n \quad (8-13)$$

for some $0 < \rho < 1$, Golomb codes can be shown to be optimal in the sense that $G_m(n)$ provides the shortest average code length of all uniquely decipherable codes (Gallager and Voorhis [1975]) when

$$m = \left\lceil \frac{\log_2(1 + \rho)}{\log_2(1/\rho)} \right\rceil \quad (8-14)$$

Figure 8.10(a) plots Eq. (8-13) for three values of ρ and graphically illustrates the symbol probabilities that Golomb codes handle well (that is, code efficiently). As is shown in the figure, small integers are much more probable than large ones.

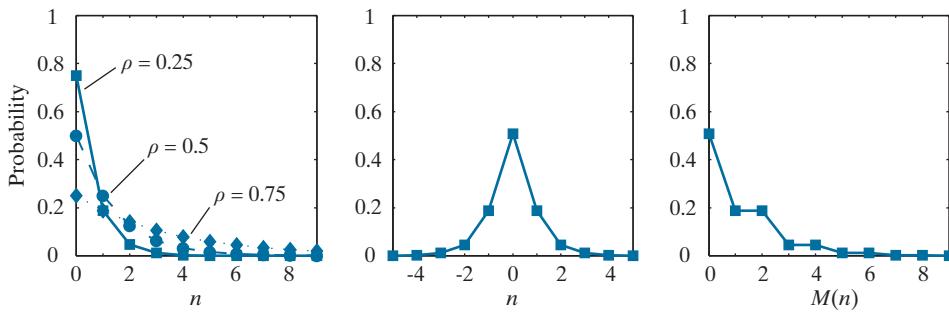
Because the probabilities of the intensities in an image [see, for example, the histogram of Fig. 8.9(b)] are unlikely to match the probabilities specified in Eq. (8-13) and shown in Fig. 8.10(a), Golomb codes are seldom used for the coding of intensities. When intensity differences are to be coded, however, the probabilities of the resulting “difference values” (see Section 8.10) (with the notable exception of the negative differences) often resemble those of Eq. (8-13) and Fig. 8.10(a). To handle negative differences in Golomb coding, which can only represent nonnegative integers, a mapping like

[†]A *probability mass function* (PMF) is a function that defines the probability that a discrete random variable is exactly equal to some value. A PMF differs from a PDF in that a PDF’s values are not probabilities; rather, the integral of a PDF over a specified interval is a probability.

a b c

FIGURE 8.10

(a) Three one-sided geometric distributions from Eq. (8-13); (b) a two-sided exponentially decaying distribution; and (c) a reordered version of (b) using Eq. (8-15).



$$M(n) = \begin{cases} 2n & n \geq 0 \\ 2|n|-1 & n < 0 \end{cases} \quad (8-15)$$

is typically used. Using this mapping, for example, the two-sided PMF shown in Fig. 8.10(b) can be transformed into the one-sided PMF in Fig. 8.10(c). Its integers are reordered, alternating the negative and positive integers so the negative integers are mapped into the odd positive integer positions. If $P(n)$ is two-sided and centered at zero, $P(M(n))$ will be one-sided. The mapped integers, $M(n)$, can then be efficiently encoded using an appropriate Golomb-Rice code (Weinberger et al. [1996]).

EXAMPLE 8.5: Golomb-Rice coding.

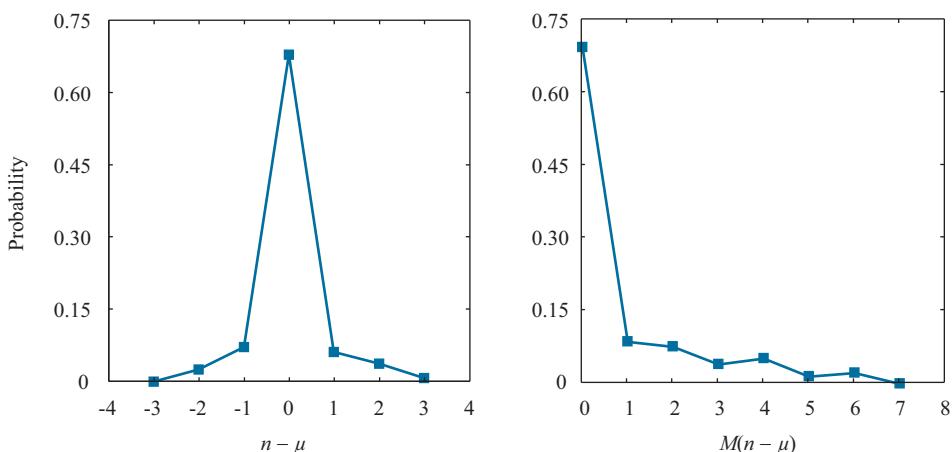
Consider again the image from Fig. 8.1(c) and note that its histogram [see Fig. 8.3(a)] is similar to the two-sided distribution in Fig. 8.10(b) above. If we let n be some nonnegative integer intensity in the image, where $0 \leq n \leq 225$, and μ be the mean intensity, $P(n-\mu)$ is the two-sided distribution shown in Fig. 8.11(a). This plot was generated by normalizing the histogram in Fig. 8.3(a) by the total number of pixels in the image and shifting the normalized values to the left by 128 (which in effect subtracts the mean intensity from the image). In accordance with Eq. (8-15), $P(M(n-\mu))$ is then the one-sided distribution shown in Fig. 8.11(b). If the reordered intensity values are Golomb coded using a MATLAB implementation of code G_1 in column 2 of Table 8.6, the encoded representation is 4.5 times smaller than the original image (i.e., $C = 4.5$). The G_1 code realizes 4.5/5.1, or 88% of the theoretical compression possible with variable-length coding. [Based on the entropy calculated in Example (8-2), the maximum possible compression ratio through variable-length coding is $C = 8/1.566 \approx 5.1$.] Moreover, Golomb coding achieves 96% of the compression provided by a MATLAB implementation of Huffman's approach, and doesn't require the computation of a custom Huffman coding table.

Now consider the image in Fig. 8.9(a). If its intensities are Golomb coded using the same G_1 code as above, $C = 0.0922$. That is, there is *data expansion*. This is due to the fact that the probabilities of the intensities of the image in Fig. 8.9(a) are much different than the probabilities defined in Eq. (8-13). In a similar manner, Huffman codes can produce data expansion when used to encode symbols whose probabilities are different from those for which the code was computed. In practice, the further you depart from the input probability assumptions for which a code is designed, the greater the risk of poor compression performance and data expansion.

a | b

FIGURE 8.11

- (a) The probability distribution of the image in Fig. 8.1(c) after subtracting the mean intensity from each pixel.
 (b) A mapped version of (a) using Eq. (8-15).



To conclude our coverage of Golomb codes, we note that Column 5 of Table 8.6 contains the first 10 codes of the zeroth-order *exponential Golomb code*, denoted $G_{\text{exp}}^0(n)$. Exponential-Golomb codes are useful for the encoding of run lengths, because both short and long runs are encoded efficiently. An order- k exponential-Golomb code $G_{\text{exp}}^k(n)$ is computed as follows:

- Find an integer $i \geq 0$ such that

$$\sum_{j=0}^{i-1} 2^{j+k} \leq n < \sum_{j=0}^i 2^{j+k} \quad (8-16)$$

and form the unary code of i . If $k = 0$, $i = \lfloor \log_2(n+1) \rfloor$ and the code is also known as the *Elias gamma code*.

- Truncate the binary representation of

$$n - \sum_{j=0}^{i-1} 2^{j+k} \quad (8-17)$$

to $k+i$ least significant bits.

- Concatenate the results of Steps 1 and 2.

To find $G_{\text{exp}}^0(8)$, for example, we let $i = \lfloor \log_2 9 \rfloor$ or 3 in Step 1 because $k = 0$. Equation (8-16) is then satisfied because

$$\sum_{j=0}^{3-1} 2^{j+0} \leq 8 < \sum_{j=0}^3 2^{j+0}$$

$$\sum_{j=0}^2 2^j \leq 8 < \sum_{j=0}^3 2^j$$

$$\begin{aligned} 2^0 + 2^1 + 2^2 &\leq 8 < 2^0 + 2^1 + 2^2 + 2^3 \\ 7 &\leq 8 < 15 \end{aligned}$$

The unary code of 3 is 1110 and Eq. (8-17) of Step 2 yields

$$8 - \sum_{j=0}^{3-1} 2^{j+0} = 8 - \sum_{j=0}^2 2^j = 8 - (2^0 + 2^1 + 2^2) = 8 - 7 = 1 = 0001$$

which when truncated to its 3 + 0 least significant bits becomes 001. The concatenation of the results from Steps 1 and 2 then yields 1110001. Note that this is the entry in column 4 of Table 8.6 for $n = 8$. Finally, we note that like the Huffman codes of the last section, the Golomb codes of Table 8.6 are variable-length, instantaneous, and uniquely decodable block codes.

8.4 ARITHMETIC CODING

With reference to Tables 8.3–8.5, arithmetic coding is used in

- JBIG1
- JBIG2
- JPEG-2000
- H.264
- MPEG-4 AVC

and other compression standards.

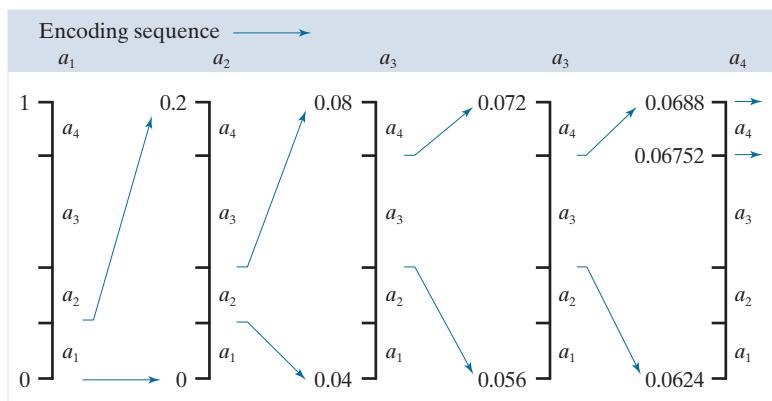
Unlike the variable-length codes of the previous two sections, *arithmetic coding* generates nonblock codes. In arithmetic coding, which can be traced to the work of Elias (Abramson [1963]), a one-to-one correspondence between source symbols and code words does not exist. Instead, an entire sequence of source symbols (or message) is assigned a single arithmetic code word. The code word itself defines an interval of real numbers between 0 and 1. As the number of symbols in the message increases, the interval used to represent it becomes smaller, and the number of information units (say, bits) required to represent the interval becomes larger. Each symbol of the message reduces the size of the interval in accordance with its probability of occurrence. Because the technique does not require, as does Huffman's approach, that each source symbol translate into an integral number of code symbols (that is, that the symbols be coded one at a time), it achieves (but only in theory) the bound established by Shannon's first theorem of Section 8.1.

Figure 8.12 illustrates the basic arithmetic coding process. Here, a five-symbol sequence or message, $a_1a_2a_3a_3a_4$, from a four-symbol source is coded. At the start of the coding process, the message is assumed to occupy the entire half-open interval $[0, 1)$. As Table 8.7 shows, this interval is subdivided initially into four regions based on the probabilities of each source symbol. Symbol a_1 , for example, is associated with subinterval $[0, 0.2)$. Because it is the first symbol of the message being coded, the message interval is initially narrowed to $[0, 0.2)$. Thus, in Fig. 8.12, $[0, 0.2)$ is expanded to the full height of the figure, and its end points labeled by the values of the narrowed range. The narrowed range is then subdivided in accordance with the original

TABLE 8.7
Arithmetic coding example.

Source Symbol	Probability	Initial Subinterval
a_1	0.2	$[0.0, 0.2)$
a_2	0.2	$[0.2, 0.4)$
a_3	0.4	$[0.4, 0.8)$
a_4	0.2	$[0.8, 1.0)$

FIGURE 8.12
Arithmetic coding procedure.



source symbol probabilities, and the process continues with the next message symbol. In this manner, symbol a_2 narrows the subinterval to $[0.04, 0.08)$, a_3 further narrows it to $[0.056, 0.072)$, and so on. The final message symbol, which must be reserved as a special end-of-message indicator, narrows the range to $[0.06752, 0.0688)$. Of course, any number within this subinterval, for example, 0.068, can be used to represent the message. In the arithmetically coded message of Fig. 8.12, three decimal digits are used to represent the five-symbol message. This translates into 0.6 decimal digits per source symbol and compares favorably with the entropy of the source, which, from Eq. 8.6, is 0.58 decimal digits per source symbol. As the length of the sequence being coded increases, the resulting arithmetic code approaches the bound established by Shannon's first theorem. In practice, two factors cause coding performance to fall short of the bound: (1) the addition of the end-of-message indicator that is needed to separate one message from another, and (2) the use of finite precision arithmetic. Practical implementations of arithmetic coding address the latter problem by introducing a scaling strategy and a rounding strategy (Langdon and Rissanen [1981]). The scaling strategy renormalizes each subinterval to the $[0, 1]$ range before subdividing it in accordance with the symbol probabilities. The rounding strategy guarantees that the truncations associated with finite precision arithmetic do not prevent the coding subintervals from being accurately represented.

ADAPTIVE CONTEXT DEPENDENT PROBABILITY ESTIMATES

With accurate input symbol *probability models*, that is, models that provide the true probabilities of the symbols being coded, arithmetic coders are near optimal in the sense of minimizing the average number of code symbols required to represent the symbols being coded. As in both Huffman and Golomb coding, however, inaccurate probability models can lead to non-optimal results. A simple way to improve the accuracy of the probabilities employed is to use an adaptive, context dependent probability model. *Adaptive* probability models update symbol probabilities as symbols are coded or become known. Thus, the probabilities adapt to the local statistics of the symbols being coded. *Context-dependent* models provide probabilities

that are based on a predefined neighborhood of pixels, called the *context*, around the symbols being coded. Normally, a *causal context* (one limited to symbols that have already been coded) is used. Both the Q-coder (Pennebaker et al. [1988]) and MQ-coder (ISO/IEC [2000]), two well-known arithmetic coding techniques that have been incorporated into the JBIG, JPEG-2000, and other important image compression standards, use probability models that are both adaptive and context dependent. The Q-coder dynamically updates symbol probabilities during the interval renormalizations that are part of the arithmetic coding process. Adaptive context dependent models also have been used in Golomb coding, for example, in the JPEG-LS compression standard.

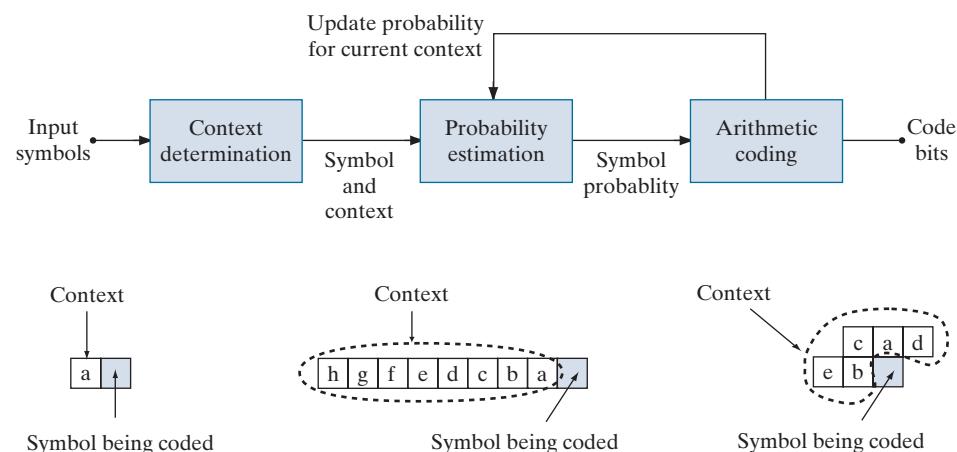
Figure 8.13(a) diagrams the steps involved in adaptive, context-dependent arithmetic coding of *binary* source symbols. Arithmetic coding often is used when binary symbols are to be coded. As each symbol (or bit) begins the coding process, its context is formed in the *Context determination* block of Fig. 8.13(a). Figures 8.13(b) through (d) show three possible contexts that can be used: (1) the immediately preceding symbol, (2) a group of preceding symbols, and (3) some number of preceding symbols plus symbols on the previous scan line. For the three cases shown, the *Probability estimation* block must manage 2^1 (or 2), 2^8 (or 256), and 2^5 (or 32) contexts and their associated probabilities. For instance, if the context in Fig. 8.13(b) is used, conditional probabilities $P(0|a = 0)$ (the probability that the symbol being coded is a 0 given that the preceding symbol is a 0), $P(1|a = 0)$, $P(0|a = 1)$, and $P(1|a = 1)$ must be tracked. The appropriate probabilities are then passed to the *Arithmetic coding* block as a function of the current context, and drive the generation of the arithmetically coded output sequence in accordance with the process illustrated in Fig. 8.12. The probabilities associated with the context involved in the current coding step are then updated to reflect the fact that another symbol within that context has been processed.

Finally, we note that a variety of arithmetic coding techniques are protected by United States patents (and may be protected in other jurisdictions as well). Because



FIGURE 8.13

- (a) An adaptive, context-based arithmetic coding approach (often used for binary source symbols).
 (b)–(d) Three possible context models.



of these patents, and the possibility of unfavorable monetary judgments for their infringement, most implementations of the JPEG compression standard, which contains options for both Huffman and arithmetic coding, typically support Huffman coding alone.

8.5 LZW CODING

With reference to
Tables 8.3–8.5, LZW cod-
ing is used in the

- GIF
- TIFF
- PDF

formats, but not in any
of the internationally
sanctioned compression
standards.

The techniques covered in Sections 8.2 through 8.4 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. Recall from the earlier section on measuring image information that Shannon used the idea of coding sequences of source symbols, rather than individual source symbols, in the proof of his first theorem. 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.

EXAMPLE 8.6: LZW coding Fig. 8.9(a).

Consider again the 512×512 , 8-bit image from Fig. 8.9(a). Using Adobe Photoshop, an uncompressed TIFF version of this image requires 286,740 bytes of disk space—262,144 bytes for the 512×512 8-bit pixels plus 24,596 bytes of overhead. Using TIFF's LZW compression option, however, the resulting file is 224,420 bytes. The compression ratio is $C = 1.28$. Recall that for the Huffman encoded representation of Fig. 8.9(a) in Example 8.4, $C = 1.077$. The additional compression realized by the LZW approach is due the removal of some of the image's spatial redundancy.

LZW coding is conceptually very simple (Welch [1984]). 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, ..., 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 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 8.7: LZW coding.

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

39	39	126	126
39	39	126	126
39	39	126	126
39	39	126	126

Table 8.8 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 variable, column 1 of Table 8.8, 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 128-bit 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., 10 9-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 simultaneously decodes 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

TABLE 8.8
LZW Coding example.

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

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

8.6 RUN-LENGTH CODING

With reference to Tables 8.3–8.5, the coding of run-lengths is used in

- CCITT
- JBIG2
- JPEG
- M-JPEG
- MPEG-1,2,4
- BMP

and other compression standards and file formats.

As was noted earlier, 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.

EXAMPLE 8.8: 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. 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 8.9. 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.

An uncompressed BMP file (saved using Photoshop) of the $512 \times 512 \times 8$ bit image shown in Fig. 8.9(a) requires 263,244 bytes of memory. Compressed using BMP's RLE option, the file expands to 267,706 bytes, and the compression ratio is $C = 0.98$. There are not enough equal intensity runs to make run-length compression effective; a small amount of expansion occurs. For the image in Fig. 8.1(c), however, the BMP RLE option results in a compression ratio $C = 1.35$. (Note that due to differences in overhead, the uncompressed BMP file is smaller than the uncompressed TIFF file in Example 8.6.)

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 as was used in Example 8.8. 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. For example, letting symbol a_j represent a black run of length j , we can estimate the probability that symbol a_j was emitted by an imaginary black run-length source by dividing the number of black run lengths of length j in the entire image by the total number of black runs. An estimate of the entropy of this black run-length source, denoted as H_0 , follows by substituting these probabilities into Eq. (8-6). A similar argument holds for the entropy of the white runs, denoted as H_1 . The approximate run-length entropy of the image is then

$$H_{RL} = \frac{H_0 + H_1}{L_0 + L_1} \quad (8-18)$$

where the variables L_0 and L_1 denote the average values of black and white run lengths, respectively. Equation (8-18) provides an estimate of the average number of bits per pixel required to code the run lengths in a binary image using a variable-length code.

TABLE 8.9

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

Two of the oldest and most widely used image compression standards are the CCITT Group 3 and 4 standards for binary image compression. Although they have been used in a variety of computer applications, they were originally designed as facsimile (FAX) coding methods for transmitting documents over telephone networks. The Group 3 standard uses a 1-D run-length coding technique in which the last $K - 1$ lines of each group of K lines (for $K = 2$ or 4) can be optionally coded in a 2-D manner. The Group 4 standard is a simplified or streamlined version of the Group 3 standard in which only 2-D coding is allowed. Both standards use the same 2-D coding approach, which is two-dimensional in the sense that information from the previous line is used to encode the current line. Both 1-D and 2-D coding will be discussed next.

ONE-DIMENSIONAL CCITT COMPRESSION

In the 1-D CCITT Group 3 compression standard, each line of an image[†] is encoded as a series of variable-length Huffman code words that represent the run lengths of alternating white and black runs in a left-to-right scan of the line. The compression method employed is commonly referred to as *Modified Huffman* (MH) coding. The code words themselves are of two types, which the standard refers to as *terminating codes* and *makeup codes*. If run length r is less than or equal to 63, a terminating code is used to represent it. The standard specifies different terminating codes for black and white runs. If $r > 63$, two codes are used; a makeup code for quotient $\lfloor r/64 \rfloor \times 64$, and terminating code for remainder $r \bmod 64$. Makeup codes may or may not depend on the intensity (black or white) of the run being coded. If $\lfloor r/64 \rfloor \times 64 \leq 1728$, separate black and white run makeup codes are specified; otherwise, makeup codes are independent of run intensity. The standard requires that each line begin with a white run-length code word, which may in fact be 00110101, the code for a white run of length zero. Finally, a unique end-of-line (EOL) code word 000000000001 is used to terminate each line, as well as to signal the first line of each new image. The end of a sequence of images is indicated by six consecutive EOLs.

Consult the book website for tables of the MH terminating and makeup codes.

Recall that the notation $\lfloor x \rfloor$ denotes the largest integer less than or equal to x .

TWO-DIMENSIONAL CCITT COMPRESSION

The 2-D compression approach adopted for both the CCITT Group 3 and 4 standards is a line-by-line method in which the position of each black-to-white or white-to-black run transition is coded with respect to the position of a *reference element* a_0 that is situated on the *current coding line*. The previously coded line is called the *reference line*; the reference line for the first line of each new image is an

[†]In the standard, images are referred to as *pages* and sequences of images are called *documents*.

imaginary white line. The 2-D coding technique that is used is called *Relative Element Address Designate* (READ) coding. In the Group 3 standard, one or three READ coded lines are allowed between successive MH coded lines; this technique is called *Modified READ* (MR) coding. In the Group 4 standard, a greater number of READ coded lines are allowed, and the method is called *Modified Modified READ* (MMR) coding. As was previously noted, the coding is two-dimensional in the sense that information from the previous line is used to encode the current line. Two-dimensional transforms are not involved.

Figure 8.14 shows the basic 2-D coding process for a single scan line. Note that the initial steps of the procedure are directed at locating several key *changing elements*: a_0 , a_1 , a_2 , b_1 , and b_2 . A changing element is defined by the standard as a pixel whose value is different from that of the previous pixel on the same line. The most important changing element is a_0 (the reference element), which is either set to the location of an imaginary white changing element to the left of the first pixel of each new coding line, or determined from the previous coding mode. Coding modes will be discussed in the following paragraph. After a_0 is located, a_1 is identified as the location of the next changing element to the right of a_0 on the current coding line, a_2 as the next changing element to the right of a_1 on the coding line, b_1 as the changing element of the opposite value (of a_0) and to the right of a_0 on the reference (or previous) line, and b_2 as the next changing element to the right of b_1 on the reference line. If any of these changing elements are not detected, they are set to the location of an imaginary pixel to the right of the last pixel on the appropriate line. Figure 8.15 provides two illustrations of the general relationships between the various changing elements.

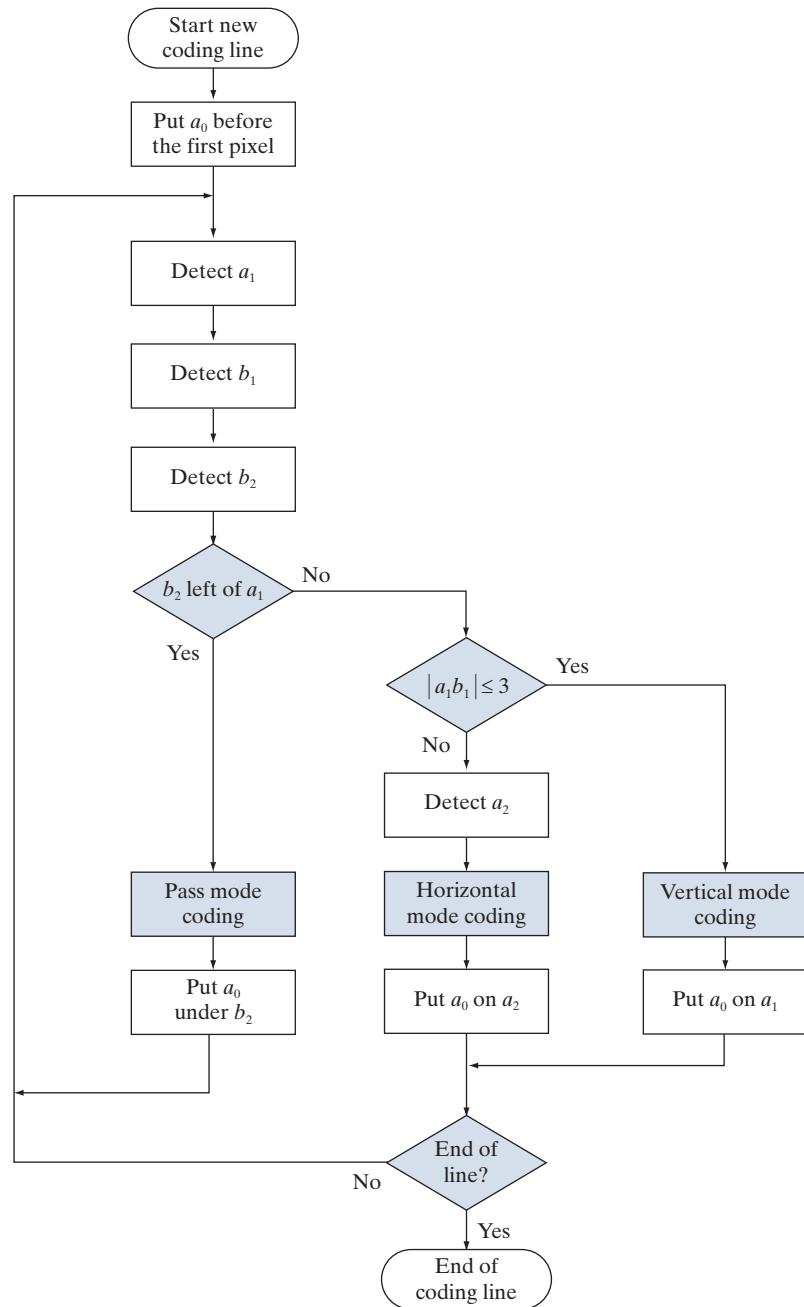
After identification of the current reference element and associated changing elements, two simple tests are performed to select one of three possible coding modes: *pass mode*, *vertical mode*, or *horizontal mode*. The initial test, which corresponds to the first branch point in the flowchart in Fig. 8.14, compares the location of b_2 to that of a_1 . The second test, which corresponds to the second branch point in Fig. 8.14, computes the distance (in pixels) between the locations of a_1 and b_1 and compares it against 3. Depending on the outcome of these tests, one of the three outlined coding blocks of Fig. 8.14 is entered and the appropriate coding procedure is executed. A new reference element is then established, as per the flowchart, in preparation for the next coding iteration.

Table 8.10 defines the specific codes utilized for each of the three possible coding modes. In pass mode, which specifically excludes the case in which b_2 is directly above a_1 , only the pass mode code word 0001 is needed. As Fig. 8.15(a) shows, this mode identifies white or black reference line runs that do not overlap the current white or black coding line runs. In horizontal coding mode, the distances from a_0 to a_1 and a_1 to a_2 must be coded in accordance with the termination and makeup codes of 1-D CCITT Group 3 compression, then appended to the horizontal mode code word 001. This is indicated in Table 8.10 by the notation $001 + M(a_0a_1) + M(a_1a_2)$, where a_0a_1 and a_1a_2 denote the distances from a_0 to a_1 and a_1 to a_2 , respectively. Finally, in vertical coding mode, one of six special variable-length codes is assigned to the distance between a_1 and b_1 . Figure 8.15(b) illustrates the parameters involved

Consult the book website for the coding tables of the CCITT standard.

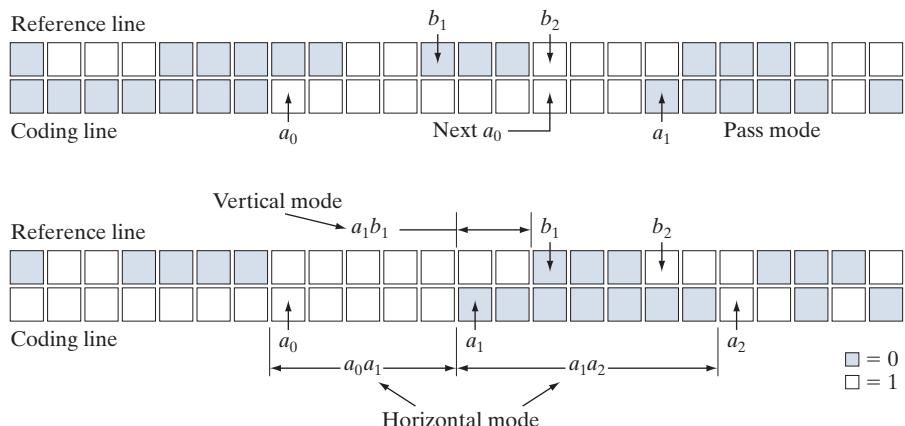
FIGURE 8.14

CCITT 2-D READ coding procedure. The notation $|a_1b_1|$ denotes the absolute value of the distance between changing elements a_1 and b_1 .



a
b

FIGURE 8.15
CCITT (a) pass mode and (b) horizontal and vertical mode coding parameters.



in both horizontal and vertical mode coding. The extension mode code word at the bottom of Table 8.10 is used to enter an optional facsimile coding mode. For example, the 0000001111 code is used to initiate an uncompressed mode of transmission.

EXAMPLE 8.9: CCITT vertical mode coding example.

Although Fig. 8.15(b) is annotated with the parameters for both horizontal and vertical mode coding (to facilitate the discussion above), the depicted pattern of black and white pixels is a case for vertical mode coding. That is, because b_2 is to the right of a_1 , the first (or pass mode) test in Fig. 8.14 fails. The second test, which determines whether the vertical or horizontal coding mode is entered, indicates that vertical mode coding should be used, because the distance from a_1 to b_1 is less than 3. In accordance with Table 8.10, the appropriate code word is 000010, implying that a_1 is two pixels left of b_1 . In preparation for the next coding iteration, a_0 is moved to the location of a_1 .

TABLE 8.10
CCITT
two-dimensional
code table.

Mode	Code Word
Pass	0001
Horizontal	$001 + M(a_0a_1) + M(a_1a_2)$
Vertical	
a_1 below b_1	1
a_1 one to the right of b_1	011
a_1 two to the right of b_1	000011
a_1 three to the right of b_1	0000011
a_1 one to the left of b_1	010
a_1 two to the left of b_1	000010
a_1 three to the left of b_1	0000010
Extension	0000001xxx

EXAMPLE 8.10: CCITT compression example.

Figure 8.16(a) is a 300 dpi scan of a 7×9.25 inch book page displayed at about 1/3 scale. Note that about half of the page contains text, around 9% is occupied by a halftone image, and the rest is white space. A section of the page is enlarged in Fig. 8.16(b). Keep in mind that we are dealing with a binary image; the illusion of gray tones is created, as was described in Section 4.5, by the halftoning process used in printing. If the binary pixels of the image in Fig. 8.16(a) are stored in groups of 8 pixels per byte, the 1952×2697 bit scanned image, commonly called a document, requires 658,068 bytes. An uncompressed PDF file of the document (created in Photoshop) requires 663,445 bytes. CCITT Group 3 compression reduces the file to 123,497 bytes, resulting in a compression ratio $C = 5.37$. CCITT Group 4 compression reduces the file to 110,456 bytes, increasing the compression ratio to about 6.

8.7 SYMBOL-BASED CODING

With reference to Tables 8.3–8.5, symbol-based coding is used in

- JBIG2 compression.

In *symbol-* or *token-based* coding, an image is represented as a collection of frequently occurring subimages, called *symbols*. Each such symbol is stored in a *symbol dictionary* and the image is coded as a set of triplets $\{(x_1, y_1, t_1), (x_2, y_2, t_2), \dots\}$, where each (x_i, y_i) pair specifies the location of a symbol in the image and *t_i* is the address of the symbol or subimage in the dictionary. That is, each triplet represents an instance of a dictionary symbol in the image. Storing repeated symbols only once can compress images significantly, particularly in document storage and retrieval applications where the symbols are often character bitmaps that are repeated many times.

a | b

FIGURE 8.16

A binary scan of a book page: (a) scaled to show the general page content; (b) scaled to show the binary pixels used in dithering.

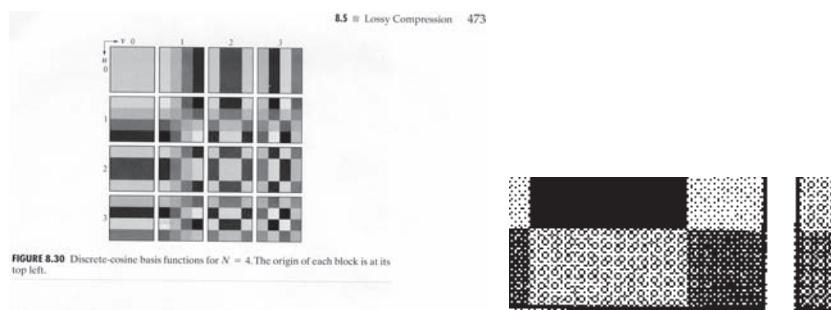


FIGURE 8.30 Discrete-cosine basis functions for $N = 4$. The origin of each block is at its top left.

where

$$\alpha(u) = \begin{cases} \sqrt{\frac{1}{N}} & \text{for } u = 0 \\ \sqrt{\frac{2}{N}} & \text{for } u = 1, 2, \dots, N - 1 \end{cases} \quad (8.5-33)$$

and similarly for $\alpha(v)$. Figure 8.30 shows $g(x, y, u, v)$ for the case $N = 4$. The computation follows the same format as explained for Fig. 8.29, with the difference that the values of g are not integers. In Fig. 8.30, the lighter gray levels correspond to larger values of g .

■ Figures 8.31(a), (c), and (e) show three approximations of the 512×512 monochrome image in Fig. 8.23. These pictures were obtained by dividing the original image into subimages of size 8×8 , representing each subimage using one of the transforms just described (i.e., the DFT, WHT, or DCT transform), truncating 50% of the resulting coefficients, and taking the inverse transform of the truncated coefficient arrays.

In each case, the 32 retained coefficients were selected on the basis of maximum magnitude. When we disregard any quantization or coding issues, this process amounts to compressing the original image by a factor of 2. Note that in all cases, the 32 discarded coefficients had little visual impact on reconstructed image quality. Their elimination, however, was accompanied by some mean-square error, which can be seen in the scaled error images of Figs. 8.31(b), (d), and (f). The actual rms errors were 1.28, 0.86, and 0.68 gray levels, respectively. ■

EXAMPLE 8.19:
Transform coding
with the DFT,
WHT, and DCT.

r N = 4. Tl

Consider the simple bilevel image in Fig. 8.17(a). It contains the single word, *banana*, which is composed of three unique symbols: a *b*, three *a*'s, and two *n*'s. Assuming that the *b* is the first symbol identified in the coding process, its 9×7 bitmap is stored in location 0 of the symbol dictionary. As Fig. 8.17(b) shows, the token identifying the *b* bitmap is 0. Thus, the first triplet in the encoded image's representation [see Fig. 8.17(c)] is (0, 2, 0), indicating that the upper-left corner (an arbitrary convention) of the rectangular bitmap representing the *b* symbol is to be placed at location (0, 2) in the decoded image. After the bitmaps for the *a* and *n* symbols have been identified and added to the dictionary, the remainder of the image can be encoded with five additional triplets. As long as the six triplets required to locate the symbols in the image, together with the three bitmaps required to define them, are smaller than the original image, compression occurs. In this case, the starting image has $9 \times 51 \times 1$ or 459 bits and, assuming that each triplet is composed of three bytes, the compressed representation has $(6 \times 3 \times 8) + [(9 \times 7) + (6 \times 7) + (6 \times 6)]$ or 285 bits; the resulting compression ratio $C = 1.61$. To decode the symbol-based representation in Fig. 8.17(c), you simply read the bitmaps of the symbols specified in the triplets from the symbol dictionary and place them at the spatial coordinates specified in each triplet.

Symbol-based compression was proposed in the early 1970s (Ascher and Nagy [1974]), but has become practical only recently. Advances in symbol-matching algorithms (see Chapter 12) and increased CPU computer processing speeds have made it possible to both select dictionary symbols and to find where they occur in an image in a timely manner. And like many other compression methods, symbol-based decoding is significantly faster than encoding. Finally, we note that both the symbol bitmaps that are stored in the dictionary and the triplets used to reference them themselves can be encoded to further improve compression performance. If, as in Fig. 8.17, only exact symbol matches are allowed, the resulting compression is lossless; if small differences are permitted, some level of reconstruction error will be present.

JBIG2 COMPRESSION

JBIG2 is an international standard for bilevel image compression. By segmenting an image into overlapping and/or non-overlapping regions of text, halftone, and generic content, compression techniques that are specifically optimized for each type of content are employed:

a b c

FIGURE 8.17

(a) A bi-level document, (b) symbol dictionary, and (c) the triplets used to locate the symbols in the document.



Token	Symbol
0	
1	
2	

Triplet
(0, 2, 0)
(3, 10, 1)
(3, 18, 2)
(3, 26, 1)
(3, 34, 2)
(3, 42, 1)

- Text regions* are composed of characters that are ideally suited for a symbol-based coding approach. Typically, each symbol will correspond to a character bitmap—a subimage representing a character of text. There is normally only one character bitmap (or subimage) in the symbol dictionary for each uppercase and lowercase character of the font being used. For example, there would be one “a” bitmap in the dictionary, one “A” bitmap, one “b” bitmap, and so on.

In lossy JBIG2 compression, often called *perceptually lossless* or *visually lossless*, we neglect differences between dictionary bitmaps (i.e., the reference character bitmaps or character templates) and specific instances of the corresponding characters in the image. In lossless compression, the differences are stored and used in conjunction with the triplets encoding each character (by the decoder) to produce the actual image bitmaps. All bitmaps are encoded either arithmetically or using MMR (see Section 8.6); the triplets used to access dictionary entries are either arithmetically or Huffman encoded.

- Halftone regions* are similar to text regions in that they are composed of patterns arranged in a regular grid. The symbols that are stored in the dictionary, however, are not character bitmaps but periodic patterns that represent intensities (e.g., of a photograph) that have been dithered to produce bilevel images for printing.
- Generic regions* contain non-text, non-halftone information, like line art and noise, and are compressed using either arithmetic or MMR coding.

As is true of many image compression standards, JBIG2 defines decoder behavior. It does not explicitly define a standard encoder, but is flexible enough to allow various encoder designs. Although the design of the encoder is left unspecified, it is important because it determines the level of compression that is achieved. After all, the encoder must segment the image into regions, choose the text and halftone symbols that are stored in the dictionaries, and decide when those symbols are essentially the same as, or different from, potential instances of the symbols in the image. The decoder simply uses that information to recreate the original image.

EXAMPLE 8.11: JBIG2 compression example.

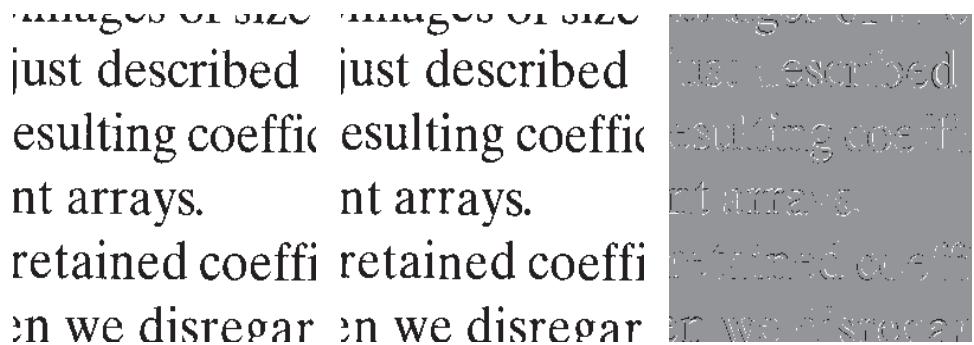
Consider again the bilevel image in Fig. 8.16(a). Figure 8.18(a) shows a reconstructed section of the image after lossless JBIG2 encoding (by a commercially available document compression application). It is an exact replica of the original image. Note that the *ds* in the reconstructed text vary slightly, despite the fact that they were generated from the same *d* entry in the dictionary. The differences between that *d* and the *ds* in the image were used to refine the output of the dictionary. The standard defines an algorithm for accomplishing this during the decoding of the encoded dictionary bitmaps. For the purposes of our discussion, you can think of it as adding the difference between a dictionary bitmap and a specific instance of the corresponding character in the image to the bitmap read from the dictionary.

Figure 8.18(b) is another reconstruction of the area in Fig. 8.18(a) after perceptually lossless JBIG2 compression. Note that the *ds* in this figure are identical. They have been copied directly from the symbol dictionary. The reconstruction is called perceptually lossless because the text is readable and the font is even the same. The small differences shown in Fig. 8.18(c) between the *ds* in the original image and the *d* in the dictionary are considered unimportant because they do not affect readability. Remember that

a b c

FIGURE 8.18

JBIG2 compression comparison: (a) lossless compression and reconstruction; (b) perceptually lossless; and (c) the scaled difference between the two.



we are dealing with bilevel images, so there are only three intensities in Fig. 8.18(c). Intensity 128 indicates areas where there is no difference between the corresponding pixels of the images in Figs. 8.18(a) and (b); intensities 0 (black) and 255 (white) indicate pixels of opposite intensities in the two images—for example, a black pixel in one image that is white in the other, and vice versa.

The lossless JBIG2 compression that was used to generate Fig. 8.18(a) reduces the original 663,445 byte uncompressed PDF image to 32,705 bytes; the compression ratio is $C = 20.3$. Perceptually lossless JBIG2 compression reduces the image to 23,913 bytes, increasing the compression ratio to about 27.7. These compressions are 4 to 5 times greater than the CCITT Group 3 and 4 results from Example 8.10.

8.8 BIT-PLANE CODING

With reference to Tables 8.3–8.5, bit-plane coding is used in

- JBIG2
 - JPEG-2000
- compression standards.

The run-length and symbol-based techniques of the previous sections can be applied to images with more than two intensities by individually processing their bit planes. The technique, called *bit-plane coding*, is based on the concept of decomposing a multilevel (monochrome or color) image into a series of binary images (see Section 3.2) and compressing each binary image via one of several well-known binary compression methods. In this section, we describe the two most popular decomposition approaches.

The intensities of an m -bit monochrome image can be represented in the form of the base-2 polynomial

$$a_{m-1} 2^{m-1} + a_{m-2} 2^{m-2} + \dots + a_1 2^1 + a_0 2^0 \quad (8-19)$$

Based on this property, a simple method of decomposing the image into a collection of binary images is to separate the m coefficients of the polynomial into m 1-bit bit planes. As noted in Section 3.2, the lowest-order bit plane (the plane corresponding to the least significant bit) is generated by collecting the a_0 bits of each pixel, while the highest-order bit plane contains the a_{m-1} bits or coefficients. In general, each bit plane is constructed by setting its pixels equal to the values of the appropriate bits or polynomial coefficients from each pixel in the original image. The inherent disadvantage of this decomposition approach is that small changes in intensity can have a significant impact on the complexity of the bit planes. If a pixel of intensity 127 (01111111) is adjacent to a pixel of intensity 128 (10000000), for instance, every bit

plane will contain a corresponding 0 to 1 (or 1 to 0) transition. For example, because the most significant bits of the binary codes for 127 and 128 are different, the highest bit plane will contain a zero-valued pixel next to a pixel of value 1, creating a 0 to 1 (or 1 to 0) transition at that point.

An alternative decomposition approach (which reduces the effect of small intensity variations) is to first represent the image by an m -bit *Gray code*. The m -bit Gray code $g_{m-1} \dots g_2 g_1 g_0$ that corresponds to the polynomial in Eq. (8-19) can be computed from

$$\begin{aligned} g_i &= a_i \oplus a_{i+1} \quad 0 \leq i \leq m-2 \\ g_{m-1} &= a_{m-1} \end{aligned} \quad (8-20)$$

Here, \oplus denotes the exclusive OR operation. This code has the unique property that successive code words differ in only one bit position. Thus, small changes in intensity are less likely to affect all m bit planes. For instance, when intensity levels 127 and 128 are adjacent, only the highest-order bit plane will contain a 0 to 1 transition, because the Gray codes that correspond to 127 and 128 are 01000000 and 11000000, respectively.

EXAMPLE 8.12: Bit-plane coding.

Figures 8.19 and 8.20 show the eight binary and Gray-coded bit planes of the 8-bit monochrome image of the child in Fig. 8.19(a). Note that the high-order bit planes are far less complex than their low-order counterparts. That is, they contain large uniform areas of significantly less detail, busyness, or randomness. In addition, the Gray-coded bit planes are less complex than the corresponding binary bit planes. Both observations are reflected in the JBIG2 coding results of Table 8.11. Note, for instance, that the a_5 and g_5 results are significantly larger than the a_6 and g_6 compressions, and that both g_5 and g_6 are smaller than their a_5 and a_6 counterparts. This trend continues throughout the table, with the single exception of a_0 . Gray-coding provides a compression advantage of about 1.06:1 on average. Combined together, the Gray-coded files compress the original monochrome image by 678,676/475,964 or 1.43:1; the non-Gray-coded files compress the image by 678,676/503,916 or 1.35:1.

Finally, we note that the two least significant bits in Fig. 8.20 have little apparent structure. Because this is typical of most 8-bit monochrome images, bit-plane coding is usually restricted to images of 6 bits/pixel or less. JBIG1, the predecessor to JBIG2, imposes such a limit.

8.9 BLOCK TRANSFORM CODING

With reference to Tables 8.3–8.5, block transform coding is used in

- JPEG
- M-JPEG
- MPEG-1,2,4
- H.261, H.262, H.263, and H.264
- DV and HDV
- VC-1

In this section, we consider a compression technique that divides an image into small non-overlapping blocks of equal size (e.g., 8×8) and processes the blocks independently using a 2-D transform. In *block transform coding*, a reversible, linear transform (such as the Fourier transform) is used to map each block or subimage into a set of transform coefficients, which are then quantized and coded. For most images, a significant number of the coefficients have small magnitudes and can be coarsely quantized (or discarded entirely) with little image distortion. A variety of

TABLE 8.11

JBIG2 lossless coding results for the binary and Gray-coded bit planes of Fig. 8.19(a). These results include the overhead of each bit plane's PDF representation.

Coefficient <i>m</i>	Binary Code (PDF bits)	Gray Code (PDF bits)	Compression Ratio
7	6,999	6,999	1.00
6	12,791	11,024	1.16
5	40,104	36,914	1.09
4	55,911	47,415	1.18
3	78,915	67,787	1.16
2	101,535	92,630	1.10
1	107,909	105,286	1.03
0	99,753	107,909	0.92

transformations, including the discrete Fourier transform (DFT) of Chapter 4, can be used to transform the image data.

Figure 8.21 shows a typical block transform coding system. The decoder implements the inverse sequence of steps (with the exception of the quantization function) of the encoder, which performs four relatively straightforward operations: subimage decomposition, transformation, quantization, and coding. An $M \times N$ input image is subdivided first into subimages of size $n \times n$, which are then transformed to generate MN/n^2 subimage transform arrays, each of size $n \times n$. The goal of the transformation process is to decorrelate the pixels of each subimage, or to pack as much information as possible into the smallest number of transform coefficients. The quantization stage then selectively eliminates or more coarsely quantizes the coefficients that carry the least amount of information in a predefined sense (several methods will be discussed later in the section). These coefficients have the smallest impact on reconstructed subimage quality. The encoding process terminates by coding (normally using a variable-length code) the quantized coefficients. Any or all of the transform encoding steps can be adapted to local image content, called *adaptive transform coding*, or fixed for all subimages, called *nonadaptive transform coding*.

In this section, we restrict our attention to square subimages (the most commonly used). It is assumed that the input image is padded, if necessary, so that both M and N are multiples of n .

TRANSFORM SELECTION

Block transform coding systems based on a variety of discrete 2-D transforms have been constructed and/or studied extensively. The choice of a particular transform in a given application depends on the amount of reconstruction error that can be tolerated and the computational resources available. Compression is achieved during the quantization of the transformed coefficients (not during the transformation step).

EXAMPLE 8.13: Block transform coding with the DFT, WHT, and DCT.

Figures 8.22(a) through (c) show three approximations of the 512×512 monochrome image in Fig. 8.9(a). These pictures were obtained by dividing the original image into subimages of size 8×8 , representing each subimage using three of the transforms described in Chapter 7 (the DFT, WHT, and DCT transforms), truncating 50% of the resulting coefficients, and taking the inverse transform of the truncated coefficient arrays.

a	b
c	d
e	f
g	h

FIGURE 8.19

(a) A 256-bit monochrome image.

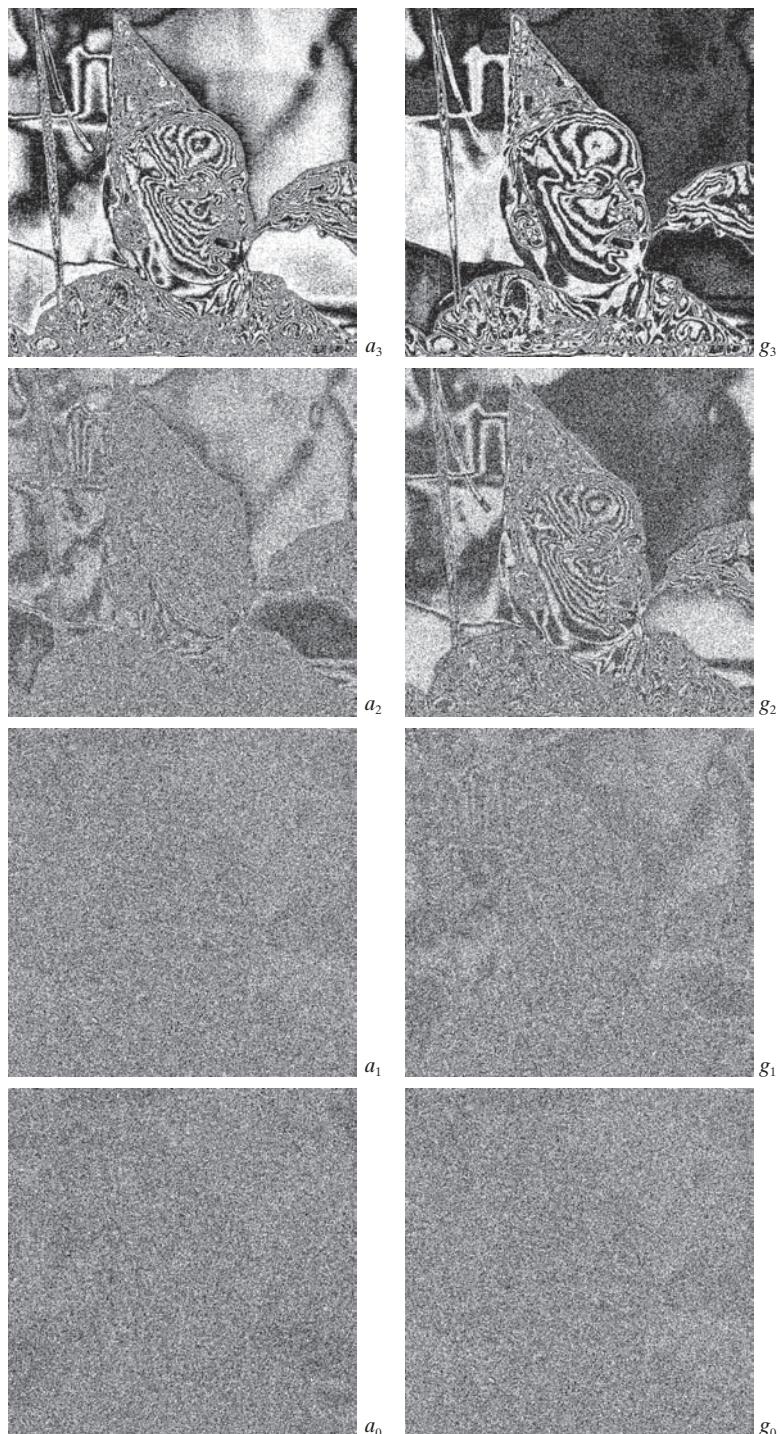
(b)–(h) The four most significant binary and Gray-coded bit planes of the image in (a).

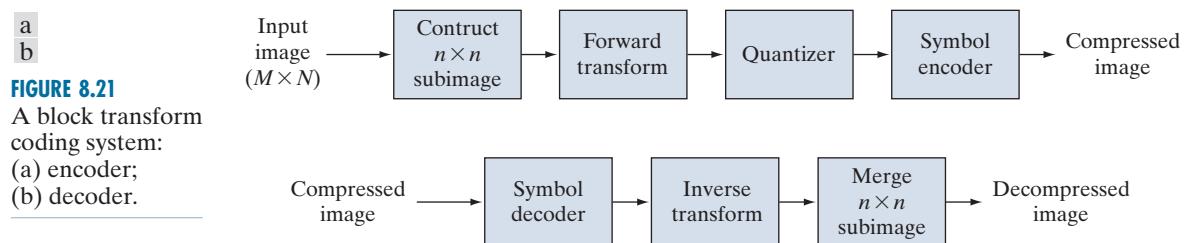


a b
c d
e f
g h

FIGURE 8.20

(a)–(h) The four least significant binary (left column) and Gray-coded (right column) bit planes of the image in Fig. 8.19(a).





In each case, the 32 retained coefficients were selected on the basis of maximum magnitude. Note that in all cases, the 32 discarded coefficients had little visual impact on the quality of the reconstructed image. Their elimination, however, was accompanied by some mean-squared error, which can be seen in the scaled error images of Figs. 8.22(d) through (f). The actual rms errors were 2.32, 1.78, and 1.13 intensities, respectively.



FIGURE 8.22 Approximations of Fig. 8.9(a) using the (a) Fourier, (b) Walsh-Hadamard, and (c) cosine transforms, together with the corresponding scaled error images in (d)–(f).

The small differences in mean-squared reconstruction error noted in the preceding example are related directly to the energy or information packing properties of the transforms employed. In accordance with Eqs. (7-75) and (7-76) of Section 7.5, an $n \times n$ subimage $g(x, y)$ can be expressed as a function of its 2-D transform $T(u, v)$:

$$\mathbf{G} = \sum_{u=0}^{n-1} \sum_{v=0}^{n-1} T(u, v) \mathbf{S}_{uv} \quad (8-21)$$

for $x, y = 0, 1, 2, \dots, n - 1$. \mathbf{G} , the matrix containing the pixels of the input subimage, is explicitly defined as a linear combination of n^2 basis images of size $n \times n$. Recall that the basis images of the DFT, DCT, and WHT transforms for $n = 8$ are shown in Figs. 7.7, 7.10, and 7.16. If we now define a transform coefficient *masking function*

$$\chi(u, v) = \begin{cases} 0 & \text{if } T(u, v) \text{ satisfies a specified truncation criterion} \\ 1 & \text{otherwise} \end{cases} \quad (8-22)$$

for $u, v = 0, 1, 2, \dots, n - 1$, an approximation of \mathbf{G} can be obtained from the truncated expansion

$$\hat{\mathbf{G}} = \sum_{u=0}^{n-1} \sum_{v=0}^{n-1} \chi(u, v) T(u, v) \mathbf{S}_{uv} \quad (8-23)$$

where $\chi(u, v)$ is constructed to eliminate the basis images that make the smallest contribution to the total sum in Eq. (8-21). The mean-squared error between subimage \mathbf{G} and approximation $\hat{\mathbf{G}}$ then is

$$\begin{aligned} e_{ms} &= E \left\{ \|\mathbf{G} - \hat{\mathbf{G}}\|^2 \right\} \\ &= E \left\{ \left\| \sum_{u=0}^{n-1} \sum_{v=0}^{n-1} T(u, v) \mathbf{S}_{uv} - \sum_{u=0}^{n-1} \sum_{v=0}^{n-1} \chi(u, v) T(u, v) \mathbf{S}_{uv} \right\|^2 \right\} \\ &= E \left\{ \left\| \sum_{u=0}^{n-1} \sum_{v=0}^{n-1} T(u, v) \mathbf{S}_{uv} [1 - \chi(u, v)] \right\|^2 \right\} \\ &= \sum_{u=0}^{n-1} \sum_{v=0}^{n-1} \sigma_{T(u, v)}^2 [1 - \chi(u, v)] \end{aligned} \quad (8-24)$$

where $\|\mathbf{G} - \hat{\mathbf{G}}\|$ is the norm of matrix $(\mathbf{G} - \hat{\mathbf{G}})$ and $\sigma_{T(u, v)}^2$ is the variance of the coefficient at transform location (u, v) . The final simplification is based on the orthonormal nature of the basis images and the assumption that the pixels of \mathbf{G} are generated by a random process with zero mean and known covariance. The total mean-squared error of approximation thus is the sum of the variances of the discarded transform coefficients; that is, the coefficients for which $\chi(u, v) = 0$, so that $[1 - \chi(u, v)]$ in Eq. (8-24) is 1. Transformations that redistribute or pack the most information into the fewest coefficients provide the best subimage approximations and, consequently, the smallest reconstruction errors. Finally, under the assumptions

that led to Eq. (8-24), the mean-squared error of the MN/n^2 subimages of an $M \times N$ image are identical. Thus the mean-squared error (being a measure of *average* error) of the $M \times N$ image equals the mean-squared error of a single subimage.

The earlier example showed that the information packing ability of the DCT is superior to that of the DFT and WHT. Although this condition usually holds for most images, the Karhunen-Loëve transform (see Chapter 11), not the DCT, is the optimal transform in an information packing sense. This is due to the fact that the KLT minimizes the mean-squared error in Eq. (8-24) for any input image and any number of retained coefficients (Kramer and Mathews [1956]). However, because the KLT is data dependent, obtaining the KLT basis images for each subimage, in general, is a nontrivial computational task. For this reason, the KLT is used infrequently for image compression. Instead, a transform, such as the DFT, WHT, or DCT, whose basis images are fixed (input independent) is normally used. Of the possible input independent transforms, the nonsinusoidal transforms (such as the WHT transform) are the simplest to implement. The sinusoidal transforms (such as the DFT or DCT) more closely approximate the information packing ability of the optimal KLT.

Hence, most transform coding systems are based on the DCT, which provides a good compromise between information packing ability and computational complexity. In fact, the properties of the DCT have proved to be of such practical value that the DCT is an international standard for transform coding systems. Compared to the other input independent transforms, it has the advantages of having been implemented in a single integrated circuit, packing the most information into the fewest coefficients[†] (for most images), and minimizing the block-like appearance, called *blocking artifact*, that results when the boundaries between subimages become visible. This last property is particularly important in comparisons with the other sinusoidal transforms. As Fig. 7.11(a) of Section 7.6 shows, the implicit n -point periodicity of the DFT gives rise to boundary discontinuities that result in substantial high-frequency transform content. When the DFT transform coefficients are truncated or quantized, the Gibbs phenomenon[‡] causes the boundary points to take on erroneous values, which appear in an image as blocking artifact. That is, the boundaries between adjacent subimages become visible because the boundary pixels of the subimages assume the mean values of discontinuities formed at the boundary points [see Fig. 7.11(a)]. The DCT of Fig. 7.11(b) reduces this effect, because its implicit $2n$ -point periodicity does not inherently produce boundary discontinuities.

SUBIMAGE SIZE SELECTION

Another significant factor affecting transform coding error and computational complexity is subimage size. In most applications, images are subdivided so the correlation (redundancy) between adjacent subimages is reduced to some acceptable level

[†]Ahmed et al. [1974] first noticed that the KLT basis images of a first-order Markov image source closely resemble the DCT's basis images. As the correlation between adjacent pixels approaches one, the input-dependent KLT basis images become identical to the input-independent DCT basis images (Clarke [1985]).

[‡]This phenomenon, described in most electrical engineering texts on circuit analysis, occurs because the Fourier transform fails to converge uniformly at discontinuities. At discontinuities, Fourier expansions take the mean values of the points of discontinuity.

An additional condition for optimality is that the masking function of Eq. (8-22) selects the KLT coefficients of maximum variance.

and so n is an integer power of 2 where, as before, n is the subimage dimension. The latter condition simplifies the computation of the subimage transforms (see the base-2 successive doubling method discussed in Section 4.11). In general, both the level of compression and computational complexity increase as the subimage size increases. The most popular subimage sizes are 8×8 and 16×16 .

EXAMPLE 8.14: Effects of subimage size on transform coding.

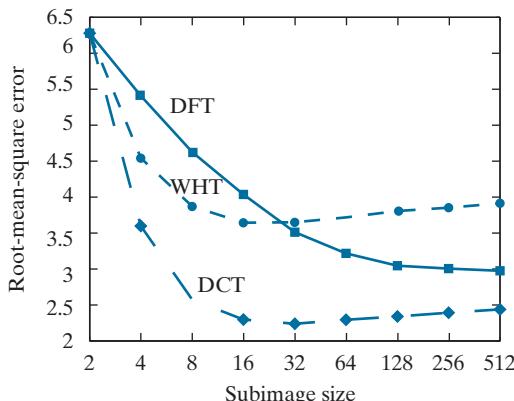
Figure 8.23 illustrates graphically the impact of subimage size on transform coding reconstruction error. The data plotted were obtained by dividing the monochrome image of Fig. 8.9(a) into subimages of size $n \times n$, for $n = 2, 4, 8, 16, \dots, 256, 512$, computing the transform of each subimage, truncating 75% of the resulting coefficients, and taking the inverse transform of the truncated arrays. Note that the Hadamard and cosine curves flatten as the size of the subimage becomes greater than 8×8 , whereas the Fourier reconstruction error continues to decrease in this region. As n further increases, the Fourier reconstruction error crosses the Walsh-Hadamard curve and approaches the cosine result. This result is consistent with the theoretical and experimental findings reported by Netravali and Limb [1980] and by Pratt [2001] for a 2-D Markov image source.

All three curves intersect when 2×2 subimages are used. In this case, only one of the four coefficients (25%) of each transformed array was retained. The coefficient in all cases was the dc component, so the inverse transform simply replaced the four subimage pixels by their average value [see Eq. (4-92)]. This condition is evident in Fig. 8.24(b), which shows a zoomed portion of the 2×2 DCT result. Note that the blocking artifact that is prevalent in this result decreases as the subimage size increases to 4×4 and 8×8 in Figs. 8.24(c) and (d). Figure 8.24(a) shows a zoomed portion of the original image for reference.

BIT ALLOCATION

The reconstruction error associated with the truncated series expansion of Eq. (8-23) is a function of the number and relative importance of the transform coefficients that are discarded, as well as the precision that is used to represent the retained coefficients. In most transform coding systems, the retained coefficients are selected [that is, the masking function of Eq. (8-22) is constructed] on the basis of maximum

FIGURE 8.23
Reconstruction error versus subimage size.



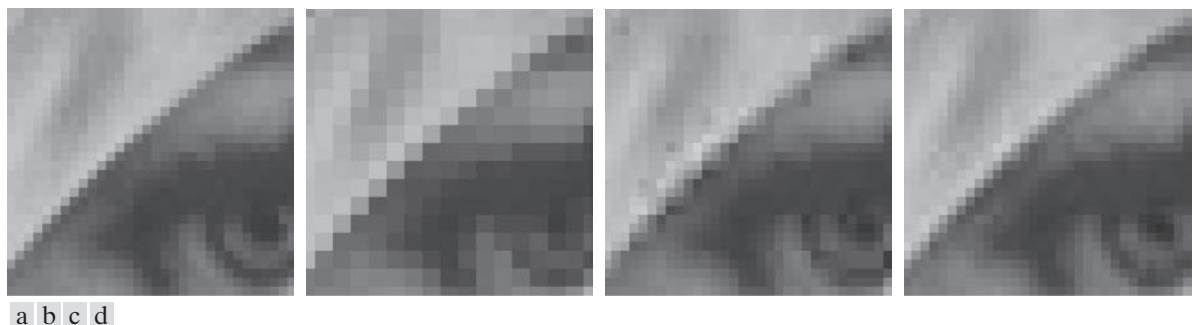


FIGURE 8.24 Approximations of Fig. 8.24(a) using 25% of the DCT coefficients and (b) 2×2 subimages, (c) 4×4 subimages, and (d) 8×8 subimages. The original image in (a) is a zoomed section of Fig. 8.9(a).

variance, called *zonal coding*, or on the basis of maximum magnitude, called *threshold coding*. The overall process of truncating, quantizing, and coding the coefficients of a transformed subimage is commonly called *bit allocation*.

EXAMPLE 8.15: Bit allocation.

Figures 8.25(a) and (c) show two approximations of Fig. 8.9(a) in which 87.5% of the DCT coefficients of each 8×8 subimage were discarded. The first result was obtained via threshold coding by keeping the eight largest transform coefficients, and the second image was generated by using a zonal coding approach. In the latter case, each DCT coefficient was considered a random variable whose distribution could be computed over the ensemble of all transformed subimages. The eight distributions of largest variance (12.5% of the 64 coefficients in the transformed 8×8 subimage) were located and used to determine the coordinates [u and v of the coefficients, $T(u, v)$] that were retained for all subimages. Note that the threshold coding difference image of Fig. 8.25(b) contains less error than the zonal coding result in Fig. 8.25(d). Both images have been scaled to make the errors more visible. The corresponding rms errors are 4.5 and 6.5 intensities, respectively.

Zonal Coding Implementation

Zonal coding is based on the information theory concept of viewing information as uncertainty. Therefore, the transform coefficients of maximum variance carry the most image information, and should be retained in the coding process. The variances themselves can be calculated directly from the ensemble of MN/n^2 transformed subimage arrays (as in the preceding example) or based on an assumed image model (say, a Markov autocorrelation function). In either case, the zonal sampling process can be viewed, in accordance with Eq. (8-23), as multiplying each $T(u, v)$ by the corresponding element in a *zonal mask*, which is constructed by placing a 1 in the locations of maximum variance and a 0 in all other locations. Coefficients of maximum variance usually are located around the origin of an image transform, resulting in the typical zonal mask shown in Fig. 8.26(a).

The coefficients retained during the zonal sampling process must be quantized and coded, so zonal masks are sometimes depicted showing the number of bits used

a	b
c	d

FIGURE 8.25

Approximations of Fig. 8.9(a) using 12.5% of the DCT coefficients: (a)–(b) threshold coding results; (c)–(d) zonal coding results. The difference images are scaled by 4.



to code each coefficient [see Fig. 8.26(b)]. In most cases, the coefficients are allocated the same number of bits, or some fixed number of bits is distributed among them unequally. In the first case, the coefficients generally are normalized by their standard deviations and uniformly quantized. In the second case, a quantizer, such as an optimal Lloyd-Max quantizer (see Optimal quantizers in Section 8.10), is designed for each coefficient. To construct the required quantizers, the zeroth or DC coefficient normally is modeled by a Rayleigh density function, whereas the remaining coefficients are modeled by a Laplacian or Gaussian density.[†] The number of quantization levels (and thus the number of bits) allotted to each quantizer is made proportional to $\log_2 \sigma_{T(u,v)}^2$. Thus, the retained coefficients in Eq. (8-23)—which (in the context of the current discussion) are selected on the basis of maximum variance—are assigned bits in proportion to the logarithm of the coefficient variances.

Threshold Coding Implementation

Zonal coding usually is implemented by using a single fixed mask for all subimages. Threshold coding, however, is inherently adaptive in the sense that the location of the transform coefficients retained for each subimage vary from one subimage to

[†]As each coefficient is a linear combination of the pixels in its subimage [see Eq. (7-31)], the central-limit theorem suggests that, as subimage size increases, the coefficients tend to become Gaussian. This result does not apply to the dc coefficient, however, because nonnegative images always have positive dc coefficients.

a	b
c	d

FIGURE 8.26

A typical
 (a) zonal mask,
 (b) zonal bit allo-
 cation,
 (c) threshold
 mask, and
 (d) thresholded
 coefficient order-
 ing sequence.
 Shading highlights
 the coefficients
 that are retained.

1	1	1	1	1	0	0	0
1	1	1	1	0	0	0	0
1	1	1	0	0	0	0	0
1	1	0	0	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
1	1	0	1	1	0	0	0
1	1	1	0	0	0	0	0
1	1	0	0	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	1	5	6	14	15	27	28
2	4	7	13	16	26	29	42
3	8	12	17	25	30	41	43
9	11	18	24	31	40	44	53
10	19	23	32	39	45	52	54
20	22	33	38	46	51	55	60
21	34	37	47	50	56	59	61
35	36	48	49	57	58	62	63

another. In fact, threshold coding is the adaptive transform coding approach most often used in practice because of its computational simplicity. The underlying concept is that, for any subimage, the transform coefficients of largest magnitude make the most significant contribution to reconstructed subimage quality, as demonstrated in the last example. Because the locations of the maximum coefficients vary from one subimage to another, the elements of $\chi(u,v)T(u,v)$ normally are reordered (in a predefined manner) to form a 1-D, run-length coded sequence. Figure 8.26(c) shows a typical *threshold mask* for one subimage of a hypothetical image. This mask provides a convenient way to visualize the threshold coding process for the corresponding subimage, as well as to mathematically describe the process using Eq. (8-23). When the mask is applied [via Eq. (8-23)] to the subimage for which it was derived, and the resulting $n \times n$ array is reordered to form an n^2 -element coefficient sequence in accordance with the zigzag ordering pattern of Fig. 8.26(d), the reordered 1-D sequence contains several long runs of 0's. [The zigzag pattern becomes evident by starting at 0 in Fig. 8.26(d) and following the numbers in sequence.] These runs normally are run-length coded. The nonzero or retained coefficients, corresponding to the mask locations that contain a 1, are represented using a variable-length code.

There are three basic ways to threshold a transformed subimage or, stated differently, to create a subimage threshold masking function of the form given in Eq. (8-22): (1) A single global threshold can be applied to all subimages; (2) a different threshold can be used for each subimage, or; (3) the threshold can be varied as a function of the location of each coefficient within the subimage. In the first approach,

The N in “ N -largest coding” is not an image dimension, but refers to the number of coefficients that are kept.

the level of compression differs from image to image, depending on the number of coefficients that exceed the global threshold. In the second, called *N-largest coding*, the same number of coefficients is discarded for each subimage. As a result, the code rate is constant and known in advance. The third technique, like the first, results in a variable code rate, but offers the advantage that thresholding and quantization can be combined by replacing $\chi(u,v)T(u,v)$ in Eq. (8-23) with

$$\hat{T}(u,v) = \text{round} \left[\frac{T(u,v)}{Z(u,v)} \right] \quad (8-25)$$

where $\hat{T}(u,v)$ is a thresholded and quantized approximation of $T(u,v)$, and $Z(u,v)$ is an element of the following transform normalization array:

$$\mathbf{Z} = \begin{bmatrix} Z(0,0) & Z(0,1) & \dots & Z(0,n-1) \\ Z(1,0) & \vdots & \dots & \vdots \\ \vdots & \vdots & \dots & \vdots \\ \vdots & \vdots & \dots & \vdots \\ \vdots & \vdots & \dots & \vdots \\ Z(n-1,0) & Z(n-1,1) & \dots & Z(n-1,n-1) \end{bmatrix} \quad (8-26)$$

Before a normalized (thresholded and quantized) subimage transform, $\hat{T}(u,v)$, can be inverse transformed to obtain an approximation of subimage $g(x,y)$, it must be multiplied by $Z(u,v)$. The resulting denormalized array, denoted $\dot{T}(u,v)$ is an approximation of $\hat{T}(u,v)$:

$$\dot{T}(u,v) = \hat{T}(u,v)Z(u,v) \quad (8-27)$$

The inverse transform of $\dot{T}(u,v)$ yields the decompressed subimage approximation.

Figure 8.27(a) graphically depicts Eq. (8-25) for the case in which $Z(u,v)$ is assigned a particular value c . Note that $\hat{T}(u,v)$ assumes integer value k if and only if

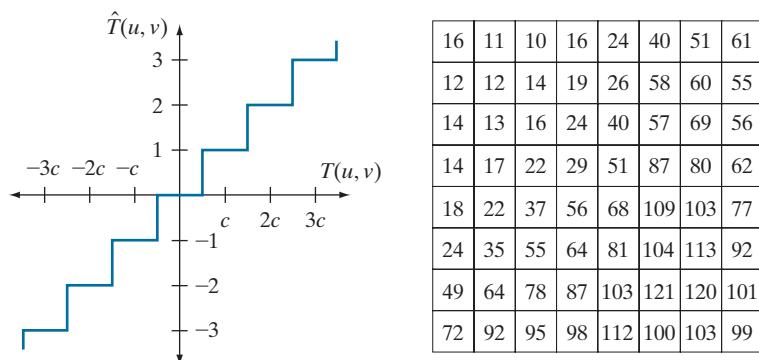
$$kc - \frac{c}{2} \leq T(u,v) < kc + \frac{c}{2} \quad (8-28)$$

If $Z(u,v) > 2T(u,v)$, then $\hat{T}(u,v) = 0$ and the transform coefficient is completely truncated or discarded. When $\hat{T}(u,v)$ is represented with a variable-length code that increases in length as the magnitude of k increases, the number of bits used to represent $T(u,v)$ is controlled by the value of c . Thus, the elements of \mathbf{Z} can be scaled to achieve a variety of compression levels. Figure 8.27(b) shows a typical normalization array. This array, which has been used extensively in the JPEG standardization efforts (see the next section), weighs each coefficient of a transformed subimage according to heuristically determined perceptual or psychovisual importance.

a b

FIGURE 8.27

(a) A threshold coding quantization curve [see Eq. (8-28)]. (b) A typical normalization matrix.

**EXAMPLE 8.16: Illustration of threshold coding.**

Figures 8.28(a) through (f) show six threshold-coded approximations of the monochrome image in Fig. 8.9(a). All images were generated using an 8×8 DCT and the normalization array of Fig. 8.27(b). The first result, which provides a compression ratio of about 12 to 1 (i.e., $C = 12$), was obtained by direct application of that normalization array. The remaining results, which compress the original image by 19, 30, 49, 85, and 182 to 1, were generated after multiplying (scaling) the normalization arrays by 2, 4, 8, 16, and 32, respectively. The corresponding rms errors are 3.83, 4.93, 6.62, 9.35, 13.94, and 22.46 intensity levels.

JPEG

One of the most popular and comprehensive continuous tone, still-frame compression standards is the JPEG standard. It defines three different coding systems: (1) a lossy baseline coding system, which is based on the DCT and is adequate for most compression applications; (2) an extended coding system for greater compression, higher precision, or progressive reconstruction applications; and (3) a lossless independent coding system for reversible compression. To be JPEG compatible, a product or system must include support for the baseline system. No particular file format, spatial resolution, or color space model is specified.

In the baseline system, often called the *sequential baseline system*, the input and output data precision is limited to 8 bits, whereas the quantized DCT values are restricted to 11 bits. The compression itself is performed in three sequential steps: DCT computation, quantization, and variable-length code assignment. The image is first subdivided into pixel blocks of size 8×8 , which are processed left-to-right, top-to-bottom. As each 8×8 block or subimage is encountered, its 64 pixels are level-shifted by subtracting the quantity 2^{k-1} , where 2^k is the maximum number of intensity levels. The 2-D discrete cosine transform of the block is then computed, quantized in accordance with Eq. (8-25), and reordered, using the zigzag pattern of Fig. 8.26(d), to form a 1-D sequence of quantized coefficients.

Because the one-dimensionally reordered array generated under the zigzag pattern of Fig. 8.26(d) is arranged qualitatively according to increasing spatial frequency, the JPEG coding procedure is designed to take advantage of the long runs



FIGURE 8.28 Approximations of Fig. 8.9(a) using the DCT and normalization array of Fig. 8.27(b): (a) \mathbf{Z} , (b) $2\mathbf{Z}$, (c) $4\mathbf{Z}$, (d) $8\mathbf{Z}$, (e) $16\mathbf{Z}$, and (f) $32\mathbf{Z}$.

Consult the book website for the JPEG default Huffman code tables:
 (1) a JPEG coefficient category table, (2) a default DC code table, and (3) a default AC code table.

of zeros that normally result from the reordering. In particular, the nonzero AC^\dagger coefficients are coded using a variable-length code that defines the coefficient values and number of preceding zeros. The DC coefficient is difference coded relative to the DC coefficient of the previous subimage. The default JPEG Huffman codes for the luminance component of a color image, or intensity of a monochrome image, are available on the book website. The JPEG recommended luminance quantization array is given in Fig. 8.27(b) and can be scaled to provide a variety of compression levels. The scaling of this array allows users to select the “quality” of JPEG compressions. Although default coding tables and quantization arrays are provided for both color and monochrome processing, the user is free to construct custom tables and/or arrays, which may be adapted to the characteristics of the image being compressed.

[†]In the standard, the term AC denotes all transform coefficients with the exception of the zeroth or DC coefficient.

EXAMPLE 8.17: JPEG baseline coding and decoding.

Consider compression and reconstruction of the following 8×8 subimage with the JPEG baseline standard:

52	55	61	66	70	61	64	73
63	59	66	90	109	85	69	72
62	59	68	113	144	104	66	73
63	58	71	122	154	106	70	69
67	61	68	104	126	88	68	70
79	65	60	70	77	63	58	75
85	71	64	59	55	61	65	83
87	79	69	68	65	76	78	94

The original image consists of 256 or 2^8 possible intensities, so the coding process begins by level shifting the pixels of the original subimage by -2^7 or -128 intensity levels. The resulting shifted array is

-76	-73	-67	-62	-58	-67	-64	-55
-65	-69	-62	-38	-19	-43	-59	-56
-66	-69	-60	-15	16	-24	-62	-55
-65	-70	-57	-6	26	-22	-58	-59
-61	-67	-60	-24	-2	-40	-60	-58
-49	-63	-68	-58	-51	-65	-70	-53
-43	-57	-64	-69	-73	-67	-63	-45
-41	-49	-59	-60	-63	-52	-50	-34

which, when transformed in accordance with the forward DCT of Eq. (7-31) with $r(x, y, u, v) = s(x, y, u, v)$ of Eq. (7-85) for $n = 8$ becomes

-415	-29	-62	25	55	-20	-1	3
7	-21	-62	9	11	-7	-6	6
-46	8	77	-25	-30	10	7	-5
-50	13	35	-15	-9	6	0	3
11	-8	-13	-2	-1	1	-4	1
-10	1	3	-3	-1	0	2	-1
-4	-1	2	-1	2	-3	1	-2
-1	-1	-1	-2	-1	-1	0	-1

If the JPEG recommended normalization array of Fig. 8.27(b) is used to quantize the transformed array, the scaled and truncated [that is, normalized in accordance with Eq. (8-25)] coefficients are

-26	-3	-6	2	2	0	0	0
1	-2	-4	0	0	0	0	0
-3	1	5	-1	-1	0	0	0
-4	1	2	-1	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

where, for instance, the DC coefficient is computed as

$$\begin{aligned}\hat{T}(0,0) &= \text{round} \left[\frac{T(0,0)}{Z(0,0)} \right] \\ &= \text{round} \left[\frac{-415}{16} \right] = -26\end{aligned}$$

Note that the transformation and normalization process produces a large number of zero-valued coefficients. When the coefficients are reordered in accordance with the zigzag ordering pattern of Fig. 8.26(d), the resulting 1-D coefficient sequence is

[-26 -3 1 -3 -2 -6 2 -4 1 -4 1 1 5 0 2 0 0 -1 2 0 0 0 0 0 -1 -1 EOB]

where the EOB symbol denotes the end-of-block condition. A special EOB Huffman code word (see category 0 and run-length 0 of the JPEG default AC code table on the book website) is provided to indicate that the remainder of the coefficients in a reordered sequence are zeros.

The construction of the default JPEG code for the reordered coefficient sequence begins with the computation of the difference between the current DC coefficient and that of the previously encoded subimage. Assuming the DC coefficient of the transformed and quantized subimage to its immediate left was -17, the resulting DPCM difference is $[-26 - (-17)]$ or -9, which lies in DC difference category 4 of the JPEG coefficient category table (see the book website). In accordance with the default Huffman difference code, the proper base code for a category 4 difference is 101 (a 3-bit code), while the total length of a completely encoded category 4 coefficient is 7 bits. The remaining 4 bits must be generated from the least significant bits (LSBs) of the difference value. For a general DC difference category (say, category K), an additional K bits are needed and computed as either the K LSBs of the positive difference or the K LSBs of the negative difference minus 1. For a difference of -9, the appropriate LSBs are (0111)-1 or 0110, and the complete DPCM coded DC code word is 1010110.

The nonzero AC coefficients of the reordered array are coded similarly. The principal difference is that each default AC Huffman code word depends on the number of zero-valued coefficients preceding the nonzero coefficient to be coded, as well as the magnitude category of the nonzero coefficient. (See the column labeled Run/Category in the JPEG AC code table on the book website.) Thus, the first nonzero AC coefficient of the reordered array (-3) is coded as 0100. The first 2 bits of this code indicate that the coefficient was in magnitude category 2 and preceded by no zero-valued coefficients; the last 2 bits are generated by the same process used to arrive at the LSBs of the DC difference code. Continuing in

this manner, the completely coded (reordered) array is

1010110 0100 001 0100 0101 100001 0110 100011 001 100011 001
001 100101 11100110 110110 0110 11110100 000 1010

where the spaces have been inserted solely for readability. Although it was not needed in this example, the default JPEG code contains a special code word for a run of 15 zeros followed by a zero. The total number of bits in the completely coded reordered array (and thus the number of bits required to represent the entire 8×8 , 8-bit subimage of this example) is 92. The resulting compression ratio is 512/92, or about 5.6:1.

To decompress a JPEG compressed subimage, the decoder must first recreate the normalized transform coefficients that led to the compressed bit stream. Because a Huffman-coded binary sequence is instantaneous and uniquely decodable, this step is easily accomplished in a simple lookup table manner. Here, the regenerated array of quantized coefficients is

-26	-3	-6	2	2	0	0	0
1	-2	-4	0	0	0	0	0
-3	1	5	-1	-1	0	0	0
-4	1	2	-1	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

After denormalization in accordance with Eq. (8-27), the array becomes

-416	-33	-60	32	48	0	0	0
12	-24	-56	0	0	0	0	0
-42	13	80	-24	-40	0	0	0
-56	17	44	-29	0	0	0	0
18	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

where, for example, the DC coefficient is computed as

$$\dot{T}(0,0) = \hat{T}(0,0)Z(0,0) = (-26)(16) = -416$$

The completely reconstructed subimage is obtained by taking the inverse DCT of the denormalized array in accordance with Eqs. (7-32) and (7-85) to obtain

-70	-64	-61	-64	-69	-66	-58	-50
-72	-73	-61	-39	-30	-40	-54	-59
-68	-78	-58	-9	13	-12	-48	-64
-59	-77	-57	0	22	-13	-51	-60
-54	-75	-64	-23	-13	-44	-63	-56
-52	-71	-72	-54	-54	-71	-71	-54
-45	-59	-70	-68	-67	-67	-61	-50
-35	-47	-61	-66	-60	-48	-44	-44

and level shifting each inverse transformed pixel by 2^7 (or +128) to yield

58	64	67	64	59	62	70	78
56	55	67	89	98	88	74	69
60	50	70	119	141	116	80	64
69	51	71	128	149	115	77	68
74	53	64	105	115	84	65	72
76	57	56	74	75	57	57	74
83	69	59	60	61	61	67	78
93	81	67	62	69	80	84	84

Any differences between the original and reconstructed subimage are a result of the lossy nature of the JPEG compression and decompression process. In this example, the errors range from -14 to +11 and are distributed as follows:

-6	-9	-6	2	11	-1	-6	-5
7	4	-1	1	11	-3	-5	3
2	9	-2	-6	-3	-12	-14	9
-6	7	0	-4	-5	-9	-7	1
-7	8	4	-1	6	4	3	-2
3	8	4	-4	2	6	1	1
2	2	5	-1	-6	0	-2	5
-6	-2	2	6	-4	-4	-6	10

The root-mean-squared error of the overall compression and reconstruction process is approximately 5.8 intensity levels.

EXAMPLE 8.18: Illustration of JPEG coding.

Figures 8.29(a) and (d) show two JPEG approximations of the monochrome image in Fig. 8.9(a). The first result provides a compression of 25:1; the second compresses the original image by 52:1. The differences between the original image and the reconstructed images in Figs. 8.29(a) and (d) are shown in Figs. 8.29(b) and (e), respectively. The corresponding rms errors are 5.4 and 10.7 intensities. The errors are clearly visible in the zoomed images in Figs. 8.29(c) and (f). These images show a magnified section of Figs. 8.29(a) and (d), respectively. Note that the JPEG blocking artifact increases with compression.

8.10 PREDICTIVE CODING

With reference to Tables 8.3–8.5, predictive coding is used in

- JBIG2
- JPEG
- JPEG-LS
- MPEG-1,2,4
- H.261, H.262, H.263, and H.264
- HDV
- VC-1

and other compression standards and file formats.

We now turn to a simpler approach that achieves good compression without significant computational overhead *and* can be either error-free or lossy. The approach, commonly referred to as *predictive coding*, is based on eliminating the redundancies of closely spaced pixels—in space and/or time—by extracting and coding only the *new information* in each pixel. The new information of a pixel is defined as the difference between the actual and predicted value of the pixel.

LOSSLESS PREDICTIVE CODING

Figure 8.30 shows the basic components of a *lossless predictive coding* system. The system consists of an encoder and a decoder, each containing an identical *predictor*. As successive samples of discrete time input signal, $f(n)$, are introduced to the encoder, the predictor generates the anticipated value of each sample based on a specified number of past samples. The output of the predictor is then rounded to the nearest integer, denoted $\hat{f}(n)$, and used to form the difference or *prediction error*

$$e(n) = f(n) - \hat{f}(n) \quad (8-29)$$

which is encoded using a variable-length code (by the symbol encoder) to generate the next element of the compressed data stream. The decoder in Fig. 8.30(b) reconstructs $e(n)$ from the received variable-length code words and performs the inverse operation

$$f(n) = e(n) + \hat{f}(n) \quad (8-30)$$

to decompress or recreate the original input sequence.

Various local, global, and adaptive methods (see the later subsection entitled Lossy predictive coding) can be used to generate $\hat{f}(n)$. In many cases, the prediction is formed as a linear combination of m previous samples. That is,

$$\hat{f}(n) = \text{round} \left[\sum_{i=1}^m \alpha_i f(n-i) \right] \quad (8-31)$$

where m is the *order* of the linear predictor, round is a function used to denote the rounding or nearest integer operation, and the α_i for $i = 1, 2, \dots, m$ are prediction

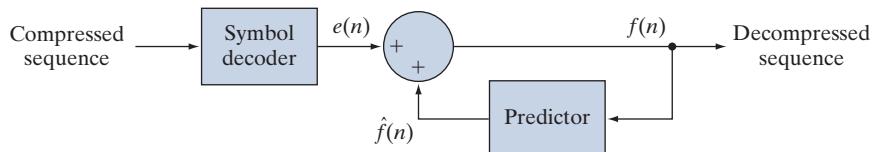
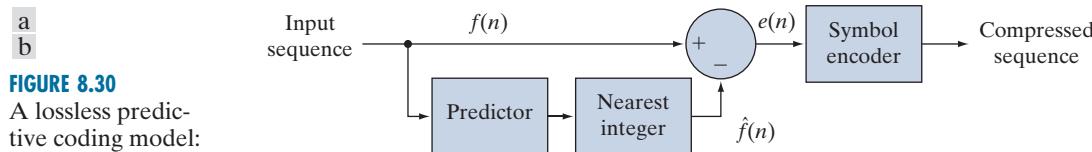


FIGURE 8.29 Two JPEG approximations of Fig. 8.9(a). Each row contains a result after compression and reconstruction, the scaled difference between the result and the original image, and a zoomed portion of the reconstructed image.

coefficients. If the input sequence in Fig. 8.30(a) is considered to be samples of an image, the $f(n)$ in Eqs. (8-29) through (8-31) are pixels—and the m samples used to predict the value of each pixel come from the current scan line (called 1-D linear predictive coding), from the current and previous scan lines (called 2-D linear predictive coding), or from the current image and previous images in a sequence of images (called 3-D linear predictive coding). Thus, for 1-D linear predictive image coding, Eq. (8-31) can be written as

$$\hat{f}(x, y) = \text{round} \left[\sum_{i=1}^m \alpha_i f(x, y - i) \right] \quad (8-32)$$

where each sample is now expressed explicitly as a function of the input image's spatial coordinates, x and y . Note that Eq. (8-32) indicates that the 1-D linear prediction is a function of the previous pixels on the current line alone. In 2-D predictive



coding, the prediction is a function of the previous pixels in a left-to-right, top-to-bottom scan of an image. In the 3-D case, it is based on these pixels and the previous pixels of preceding frames. Equation (8-32) cannot be evaluated for the first m pixels of each line, so those pixels must be coded by using other means (such as a Huffman code) and considered as an overhead of the predictive coding process. Similar comments apply to the higher-dimensional cases.

EXAMPLE 8.19: Predictive coding and spatial redundancy.

Consider encoding the monochrome image of Fig. 8.31(a) using the simple first-order (i.e., $m = 1$) linear predictor from Eq. (8-32)

$$\hat{f}(x, y) = \text{round}[\alpha f(x, y - 1)] \quad (8-33)$$

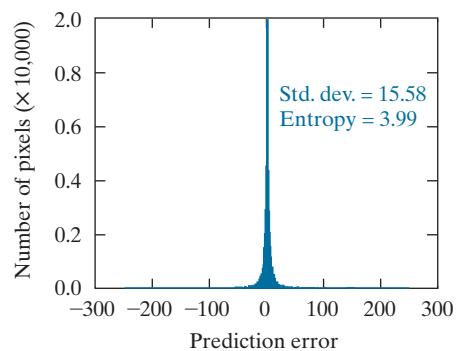
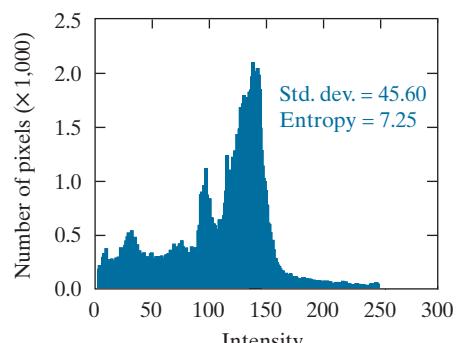
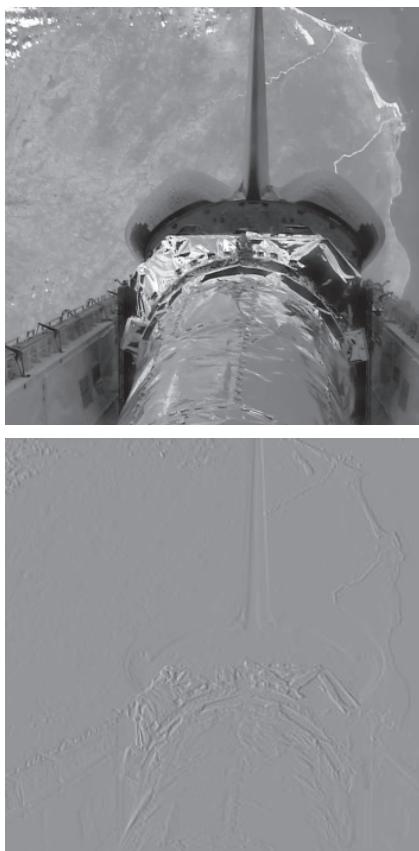
This equation is a simplification of Eq. (8-32), with $m = 1$ and the subscript of lone prediction coefficient α_1 removed as unnecessary. A predictor of this general form is called a *previous pixel* predictor, and the corresponding predictive coding procedure is known as *differential coding* or *previous pixel coding*. Figure 8.31(c) shows the prediction error image, $e(x, y) = f(x, y) - \hat{f}(x, y)$ that results from Eq. (8-33) with $\alpha = 1$. The scaling of this image is such that intensity 128 represents a prediction error of zero, while all nonzero positive and negative prediction errors (under and over estimates) are displayed as lighter and darker shades of gray, respectively. The mean value of the prediction image is 128.26. Because intensity 128 corresponds to a prediction error of 0, the average prediction error is only 0.26 bits.

Figures 8.31(b) and (d) show the intensity histogram of the image in Fig. 8.31(a) and the histogram of prediction error $e(x, y)$, respectively. Note that the standard deviation of the prediction error in Fig. 8.31(d) is much smaller than the standard deviation of the intensities in the original image. Moreover, the entropy of the prediction error, as estimated using Eq. (8-7), is significantly less than the estimated entropy of the original image (3.99 bits/pixel as opposed to 7.25 bits/pixel). This decrease in entropy reflects removal of a great deal of spatial redundancy, despite the fact that for k -bit images, $(k + 1)$ -bit numbers are needed to represent accurately the prediction error sequence $e(x, y)$. (Note that the variable-length encoded prediction error is the compressed image.) In general, the maximum compression of a predictive coding approach can be estimated by dividing the average number of bits used

a
b
c
d

FIGURE 8.31

- (a) A view of the Earth from an orbiting space shuttle. (b) The intensity histogram of (a). (c) The prediction error image resulting from Eq. (8-33). (d) A histogram of the prediction error. (Original image courtesy of NASA.)



to represent each pixel in the original image by an estimate of the entropy of the prediction error. In this example, any variable-length coding procedure can be used to code $e(x, y)$, but the resulting compression will be limited to about 8/3.99, or 2:1.

The preceding example illustrates that the compression achieved in predictive coding is related directly to the entropy reduction that results from mapping an input image into a prediction error sequence, often called a *prediction residual*. Because spatial redundancy is removed by the prediction and differencing process, the probability density function of the prediction residual is, in general, highly peaked at zero, and characterized by a relatively small (in comparison to the input intensity distribution) variance. In fact, it is often modeled by a zero mean uncorrelated Laplacian PDF

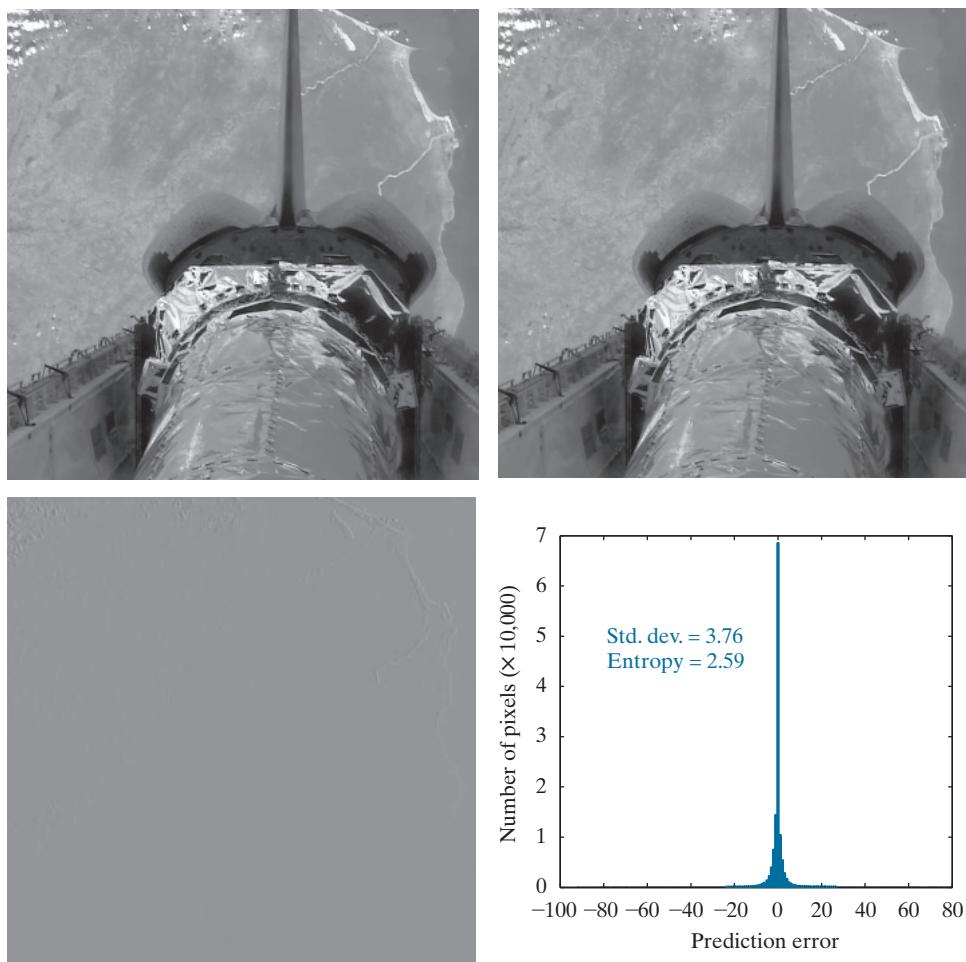
$$p_e(e) = \frac{1}{\sqrt{2}\sigma_e} e^{-\frac{\sqrt{2}|e|}{\sigma_e}} \quad (8-34)$$

where σ_e is the standard deviation of e .

a	b
c	d

FIGURE 8.32

(a) and (b) Two views of Earth from an orbiting space shuttle video. (c) The prediction error image resulting from Eq. (8-35). (d) A histogram of the prediction error.
(Original images courtesy of NASA.)


EXAMPLE 8.20: Predictive coding and temporal redundancy.

The image in Fig. 8.31(a) is a portion of a frame of NASA video in which the Earth is moving from left to right with respect to a stationary camera attached to the space shuttle. It is repeated in Fig. 8.32(b), along with its immediately preceding frame in Fig. 8.32(a). Using the first-order linear predictor

$$\hat{f}(x, y, t) = \text{round}[\alpha f(x, y, t - 1)] \quad (8-35)$$

with $\alpha = 1$, the intensities of the pixels in Fig. 8.32(b) can be predicted from the corresponding pixels in (a). Figure 8.34(c) is the resulting prediction residual image, $e(x, y, t) = f(x, y, t) - \hat{f}(x, y, t)$. Figure 8.31(d) is the histogram of $e(x, y, t)$. Note there is very little prediction error. The standard deviation of the error is much smaller than in the previous example: 3.76 bits/pixel as opposed to 15.58 bits/pixel. In addition, the entropy of the prediction error [computed using Eq. (8-7)] has decreased from 3.99 to 2.59 bits/pixel. (Recall again that the variable-length encoded prediction error is the compressed

image.) By variable-length coding the resulting prediction residual, the original image is compressed by approximately 8/2.59 or 3.1:1, a 50% improvement over the 2:1 compression obtained using the spatially oriented previous pixel predictor in Example 8.19.

MOTION COMPENSATED PREDICTION RESIDUALS

As you saw in Example 8.20, successive frames in a video sequence often are very similar. Coding their differences can reduce temporal redundancy and provide significant compression. However, when a sequence of frames contains rapidly moving objects—or involves camera zoom and pan, sudden scene changes, or fade-ins and fade-outs—the similarity between neighboring frames is reduced, and compression is affected negatively. That is, like most compression techniques (see Example 8.5), temporally based predictive coding works best with certain kinds of inputs, namely, a sequence of images with significant temporal redundancy. When used on images with little temporal redundancy, data expansion can occur. Video compression systems avoid the problem of data expansion in two ways:

1. By tracking object movement and compensating for it during the prediction and differencing process.
2. By switching to an alternate coding method when there is insufficient *inter-frame* correlation (similarity between frames) to make predictive coding advantageous.

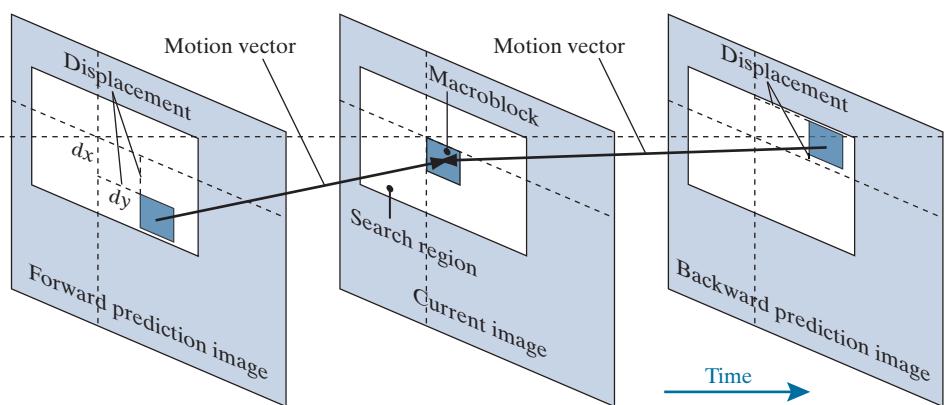
The first of these, called *motion compensation*, is the subject of the remainder of this section. Before proceeding, however, we should note that when there is insufficient interframe correlation to make predictive coding effective, the second problem is typically addressed using a block-oriented 2-D transform, like JPEG's DCT-based coding (see the previous section). Frames compressed in this way (i.e., without a prediction residual) are called *intraframes* or *Independent frames (I-frames)*. They can be decoded without access to other frames in the video to which they belong. I-frames usually resemble JPEG encoded images, and are ideal starting points for the generation of prediction residuals. Moreover, they provide a high degree of random access, ease of editing, and resistance to the propagation of transmission error. As a result, all standards require the periodic insertion of I-frames into the compressed video codestream.

Figure 8.33 illustrates the basics of motion-compensated predictive coding. Each video frame is divided into non-overlapping rectangular regions (typically of size 4×4 to 16×16) called *macroblocks*. (Only one macroblock is shown in Fig. 8.33.) The “movement” of each macroblock with respect to its “most likely” position in the previous (or subsequent) video frame, called the *reference frame*, is encoded in a *motion vector*. The vector describes the motion by defining the horizontal and vertical *displacement* from the “most likely” position. The displacements typically are specified to the nearest pixel, $\frac{1}{2}$ pixel, or $\frac{1}{4}$ pixel precision. If subpixel precision is used, the predictions must be interpolated [e.g., using bilinear interpolation (see Section 2.4)] from a combination of pixels in the reference frame. An encoded frame that is based on the previous frame (a *forward prediction* in Fig. 8.33) is called a *Pre-*

The “most likely” position is the one that minimizes an error measure between the reference macroblock and the macroblock being encoded. The two blocks do not have to be representations of the same object, but they must minimize the error measure.

FIGURE 8.33

Macroblock motion specification.



predictive frame (P-frame); one that is based on the subsequent frame (a *backward prediction* in Fig. 8.33) is called a *Bidirectional frame (B-frame)*. B-frames require the compressed codestream to be reordered so that frames are presented to the decoder in the proper decoding sequence, rather than the natural display order.

As you might expect, *motion estimation* is the key component of motion compensation. During motion estimation, the motion of objects is measured and encoded into motion vectors. The search for the “best” motion vector requires that a criterion of optimality be defined. For example, motion vectors may be selected on the basis of maximum correlation or minimum error between macroblock pixels and the predicted pixels (or interpolated pixels for sub-pixel motion vectors) from the chosen reference frame. One of the most commonly used error measures is *mean absolute distortion (MAD)*

$$MAD(x, y) = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} |f(x+i, y+j) - p(x+i+dx, y+j+dy)| \quad (8-36)$$

where x and y are the coordinates of the upper-left pixel of the $m \times n$ macroblock being coded, dx and dy are displacements from the reference frame as shown in Fig. 8.33, and p is an array of predicted macroblock pixel values. For sub-pixel motion vector estimation, p is interpolated from pixels in the reference frame. Typically, dx and dy must fall within a limited search region (see Fig. 8.33) around each macroblock. Values from ± 8 to ± 64 pixels are common, and the horizontal search area is often slightly larger than the vertical area. A more computationally efficient error measure, called the *sum of absolute distortions (SAD)*, omits the $1/mn$ factor in Eq. (8-36).

Given a selection criterion like that of Eq. (8-36), motion estimation is performed by searching for the dx and dy that minimize $MAD(x, y)$ over the allowed range of motion vector displacements, including subpixel displacements. This process often is called *block matching*. An exhaustive search guarantees the best possible result, but is computationally expensive because every possible motion must be tested over the entire displacement range. For 16×16 macroblocks and a ± 32 pixel displacement

range (not out of the question for action films and sporting events), 4225 16×16 *MAD* calculations must be performed for each macroblock in a frame when integer displacement precision is used. If $\frac{1}{2}$ or $\frac{1}{4}$ pixel precision is desired, the number of calculations is multiplied by a factor of 4 or 16, respectively. Fast search algorithms can reduce the computational burden, but may or may not yield optimal motion vectors. A number of fast block-based motion estimation algorithms have been proposed and studied in the literature (see Furht et al. [1997] or Mitchell et al. [1997]).

EXAMPLE 8.21: Motion compensated prediction.

Figures 8.34(a) and (b) were taken from the same NASA video sequence used in Examples 8.19 and 8.20. Figure 8.34(b) is identical to Figs. 8.31(a) and 8.32(b); Fig. 8.34(a) is the corresponding section of a frame occurring thirteen frames earlier. Figure 8.34(c) is the difference between the two frames, scaled to the full intensity range. Note that the difference is 0 in the area of the stationary (with respect to the camera) space shuttle, but there are significant differences in the remainder of the image due to the relative motion of the Earth. The standard deviation of the prediction residual in Fig. 8.34(c) is 12.73 intensity levels; its entropy [using Eq. (8-7)] is 4.17 bits/pixel. The maximum compression achievable, when variable-length coding the prediction residual, is $C = 8/4.17 = 1.92$.

Figure 8.34(d) shows a motion compensated prediction residual with a much lower standard deviation (5.62 as opposed to 12.73 intensity levels) and slightly lower entropy (3.04 vs. 4.17 bits/pixel). The entropy was computed using Eq. (8-7). If the prediction residual in Fig. 8.34(d) is variable-length coded, the resulting compression ratio is $C = 8/3.04 = 2.63$. To generate this prediction residual, we divided Fig. 8.34(b) into non-overlapping 16×16 macroblocks and compared each macroblock against every 16×16 region in Fig. 8.34(a)—the reference frame—that fell within ± 16 pixels of the macroblock’s position in (b). We then used Eq. (8-36) to determine the best match by selecting displacement (dx, dy) with the lowest *MAD*. The resulting displacements are the x and y components of the motion vectors shown in Fig. 8.34(e). The white dots in the figure are the heads of the motion vectors; they indicate the upper-left-hand corner of the coded macroblocks. As you can see from the pattern of the vectors, the predominant motion in the image is from left to right. In the lower portion of the image, which corresponds to the area of the space shuttle in the original image, there is no motion, and therefore no motion vectors displayed. Macroblocks in this area are predicted from similarly located (i.e., the corresponding) macroblocks in the reference frame. Because the motion vectors in Fig. 8.34(e) are highly correlated, they can be variable-length coded to reduce their storage and transmission requirements

The visual difference between Figs. 8.34(c) and 8.35(a) is due to scaling. The image in Fig. 8.35(a) has been scaled to match Figs. 8.35(b)–(d).

Figure 8.35 illustrates the increased prediction accuracy that is possible with sub-pixel motion compensation. Figure 8.35(a) is repeated from Fig. 8.34(c) and included as a point of reference; it shows the prediction error that results without motion compensation. The images in Figs. 8.35(b), (c), and (d) are motion compensated prediction residuals. They are based on the same two frames that were used in Example 8.21 and computed with macroblock displacements to 1 , $\frac{1}{2}$, and $\frac{1}{4}$ pixel resolution (i.e., precision), respectively. Macroblocks of size 8×8 were used; displacements were limited to ± 8 pixels.

The most significant visual difference between the prediction residuals in Fig. 8.35 is the number and size of intensity peaks and valleys—their darkest and lightest areas of intensity. The $\frac{1}{4}$ pixel residual in Fig. 8.35(d) is the “flattest” of the four

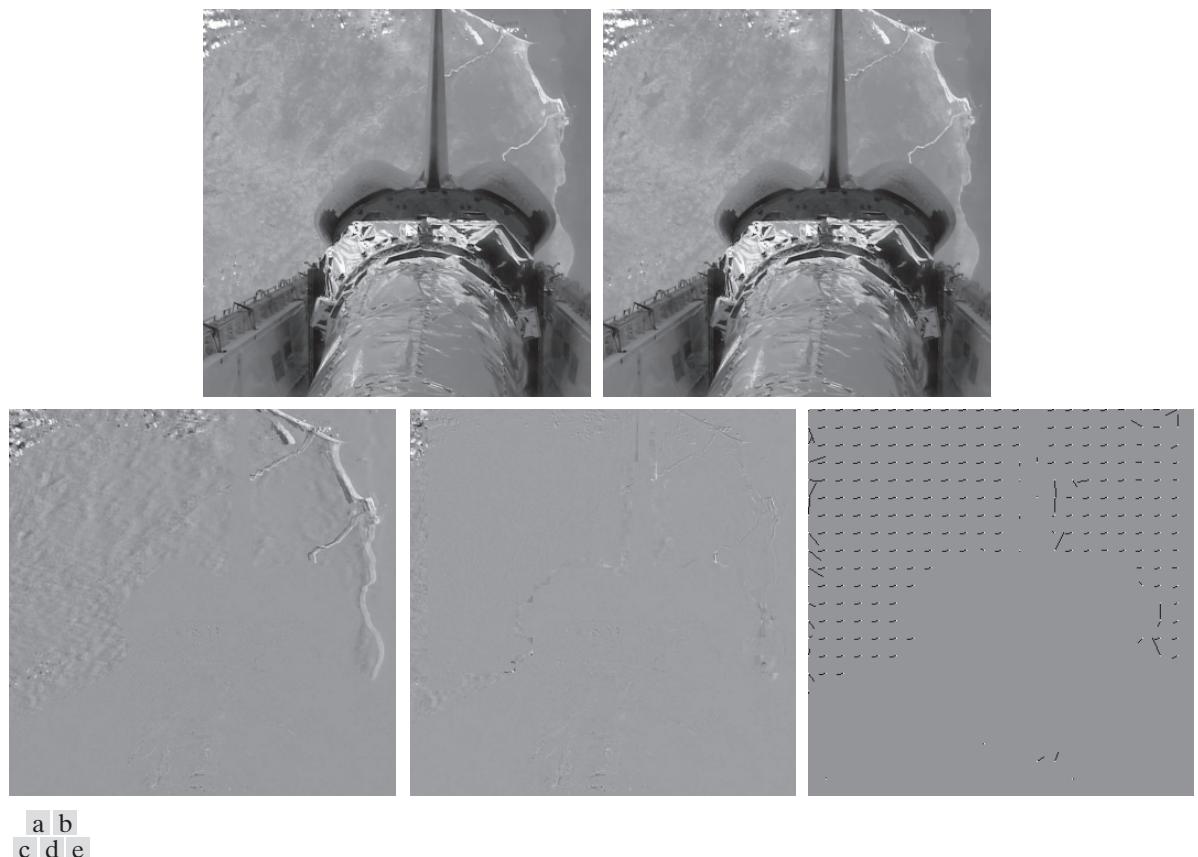


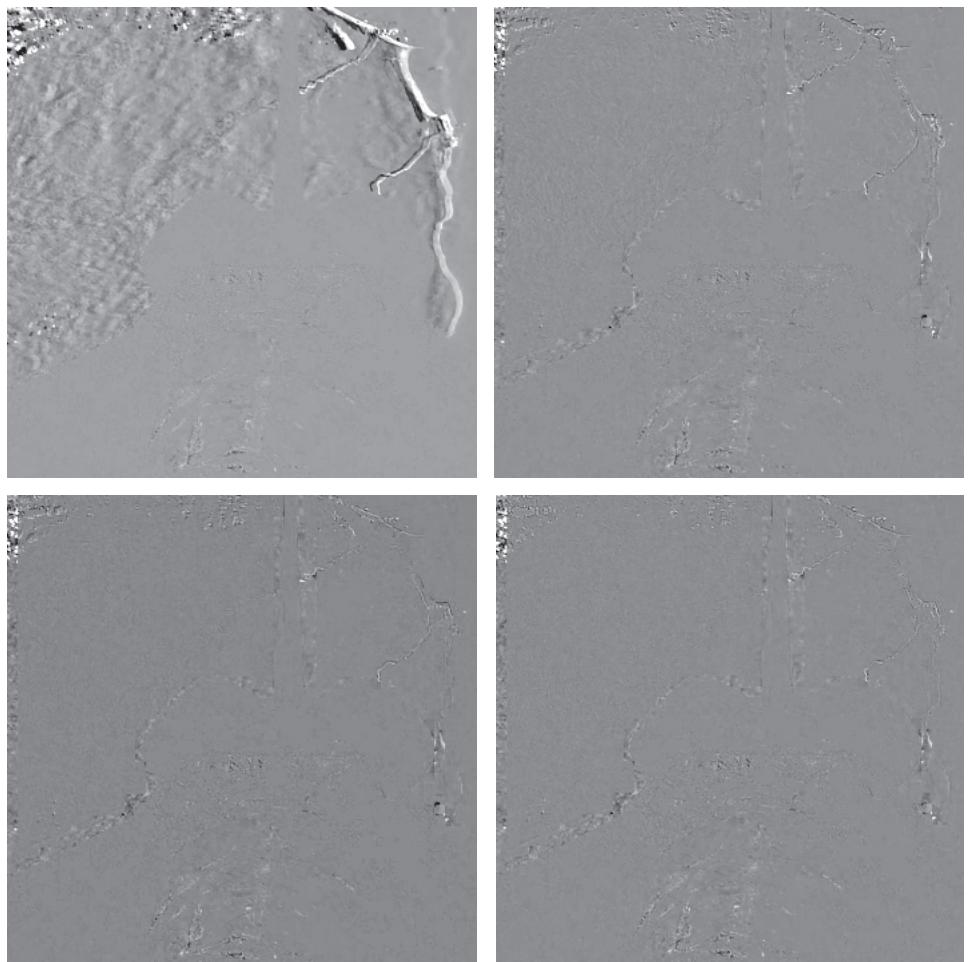
FIGURE 8.34 (a) and (b) Two views of Earth that are thirteen frames apart in an orbiting space shuttle video. (c) A prediction error image without motion compensation. (d) The prediction residual with motion compensation. (e) The motion vectors associated with (d). The white dots in (e) represent the arrow heads of the motion vectors that are depicted. (Original images courtesy of NASA.)

images, with the fewest excursions to black or white. As would be expected, it has the narrowest histogram. The standard deviations of the prediction residuals in Figs. 8.35(a) through (d) decrease as motion vector precision increases from 12.7 to 4.4, 4, and 3.8 pixels, respectively. The entropies of the residuals, as determined using Eq. (8-7), are 4.17, 3.34, 3.35, and 3.34 bits/pixel, respectively. Thus, the motion compensated residuals contain about the same amount of information, despite the fact that the residuals in Figs. 8.35(c) and (d) use additional bits to accommodate $\frac{1}{2}$ and $\frac{1}{4}$ pixel interpolation. Finally, we note that there is an obvious strip of increased prediction error on the left side of each motion compensated residual. This is due to the left-to-right motion of the Earth, which introduces new or previously unseen areas of the Earth's terrain into the left side of each image. Because these areas are absent from the previous frames, they cannot be accurately predicted, regardless of the precision used to compute motion vectors.

a b
c d

FIGURE 8.35

Sub-pixel motion compensated prediction residuals: (a) without motion compensation; (b) single pixel precision; (c) $\frac{1}{2}$ pixel precision; and (d) $\frac{1}{4}$ pixel precision. (All prediction errors have been scaled to the full intensity range and then multiplied by 2 to increase their visibility.)



Motion estimation is a computationally demanding task. Fortunately, only the encoder must estimate macroblock motion. Given the motion vectors of the macroblocks, the decoder simply accesses the areas of the reference frames that were used in the encoder to form the prediction residuals. Because of this, motion estimation is not included in most video compression standards. Compression standards focus on the decoder, placing constraints on macroblock dimensions, motion vector precision, horizontal and vertical displacement ranges, and the like. Table 8.12 gives the key predictive coding parameters of some the most important video compression standards. Note that most of the standards use an 8×8 DCT for I-frame encoding, but specify a larger area (i.e., 16×16 macroblock) for motion compensation. In addition, even the P- and B-frame prediction residuals are transform coded due to the effectiveness of DCT coefficient quantization. Finally, we note that the H.264 and MPEG-4 AVC standards support intraframe predictive coding (in I-frames) to reduce spatial redundancy.

TABLE 8.12

Predictive coding in video compression standards.

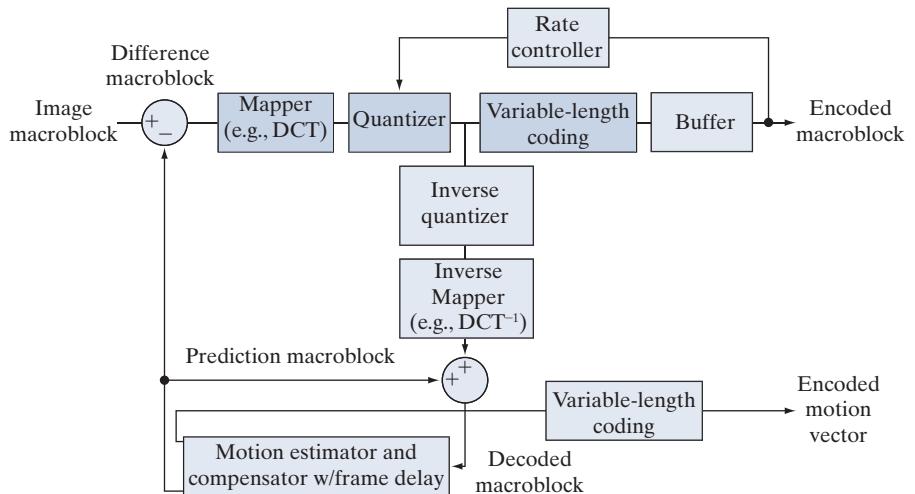
Parameter	H.261	MPEG-1	H.262 MPEG-2	H.263	MPEG-4	VC-1 WMV-9	H.264 MPEG-4 AVC
Motion vector precision	1	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$
Macroblock sizes	16×16	16×16	16×16 16×8	16×16 8×8	16×16 8×8	16×16 8×8 8×4 4×8 4×4	16×16 16×8 8×8 8×4 4×8 4×4
Transform	8×8 DCT	8×8 DCT	8×8 DCT	8×8 DCT	8×8 DCT	8×8 8×4 4×8 4×4 Integer DCT	4×4 8×8 Integer DCT
Interframe predictions	P	P, B	P, B	P, B	P, B	P, B	P, B
I-frame intra-predictions	No	No	No	No	No	No	Yes

Figure 8.36 shows a typical motion compensated video encoder. It exploits redundancies within and between adjacent video frames, motion uniformity between frames, and the psychovisual properties of the human visual system. We can think of the input to the encoder as sequential macroblocks of video. For color video, each macroblock is composed of a luminance block and two chrominance blocks. Because the eye has far less spatial acuity for color than for luminance, the chrominance blocks often are sampled at half the horizontal and vertical resolution of the luminance block. The dark blue elements in the figure parallel the transformation, quantization, and variable-length coding operations of a JPEG encoder. The principal difference is the input, which may be a conventional macroblock of image data (for I-frames), or the difference between a conventional macroblock and a prediction of it based on previous and/or subsequent video frames (for P- and B-frames). The encoder includes an *inverse quantizer* and *inverse mapper* (e.g., inverse DCT) so that its predictions match those of the complementary decoder. Also, it is designed to produce compressed bit streams that match the capacity of the intended video channel. To accomplish this, the quantization parameters are adjusted by a *rate controller* as a function of the occupancy of an output *buffer*. As the buffer becomes fuller, the quantization is made coarser, so fewer bits stream into the buffer.

Quantization as defined earlier in the chapter is irreversible. The “inverse quantizer” in Fig. 8.36 does not prevent information loss.

FIGURE 8.36

A typical motion compensated video encoder.



EXAMPLE 8.22: Video compression example.

We conclude our discussion of motion compensated predictive coding with an example illustrating the kind of compression that is possible with modern video compression methods. Figure 8.37 shows fifteen frames of a 1 minute HD (1280×720) full-color NASA video, parts of which have been used throughout this section. Although the images shown are monochrome, the video is a sequence of 1,829 full-color frames. Note that there are a variety of scenes, a great deal of motion, and multiple fade effects. For example, the video opens with a 150 frame fade-in from black, which includes frames 21 and 44 in Fig. 8.37, and concludes with a fade sequence containing frames 1595, 1609, and 1652 in Fig. 8.37, followed by a final fade to black. There are also several abrupt scene changes, like the change involving frames 1303 and 1304 in Fig. 8.37.

An H.264 compressed version of the NASA video stored as a Quicktime file (see Table 8.5) requires 44.56 MB of storage, plus another 1.39 MB for the associated audio. The video quality is excellent. About 5 GB of data would be needed to store the video frames as uncompressed full-color images. It should be noted that the video contains sequences involving both rotation and scale change (e.g., the sequence including frames 959, 1023, and 1088 in Fig. 8.37). The discussion in this section, however, has been limited to translation alone. (See the book website for the NASA video segment used in this example.)

LOSSY PREDICTIVE CODING

In this section, we add a quantizer to the lossless predictive coding model introduced earlier, and examine the trade-off between reconstruction accuracy and compression performance within the context of spatial predictors. As Fig. 8.38 shows, the quantizer, which replaces the nearest integer function of the error-free encoder, is inserted between the symbol encoder and the point at which the prediction error is formed. It maps the prediction error into a limited range of outputs, denoted $\dot{e}(n)$, which establish the amount of compression and distortion that occurs.

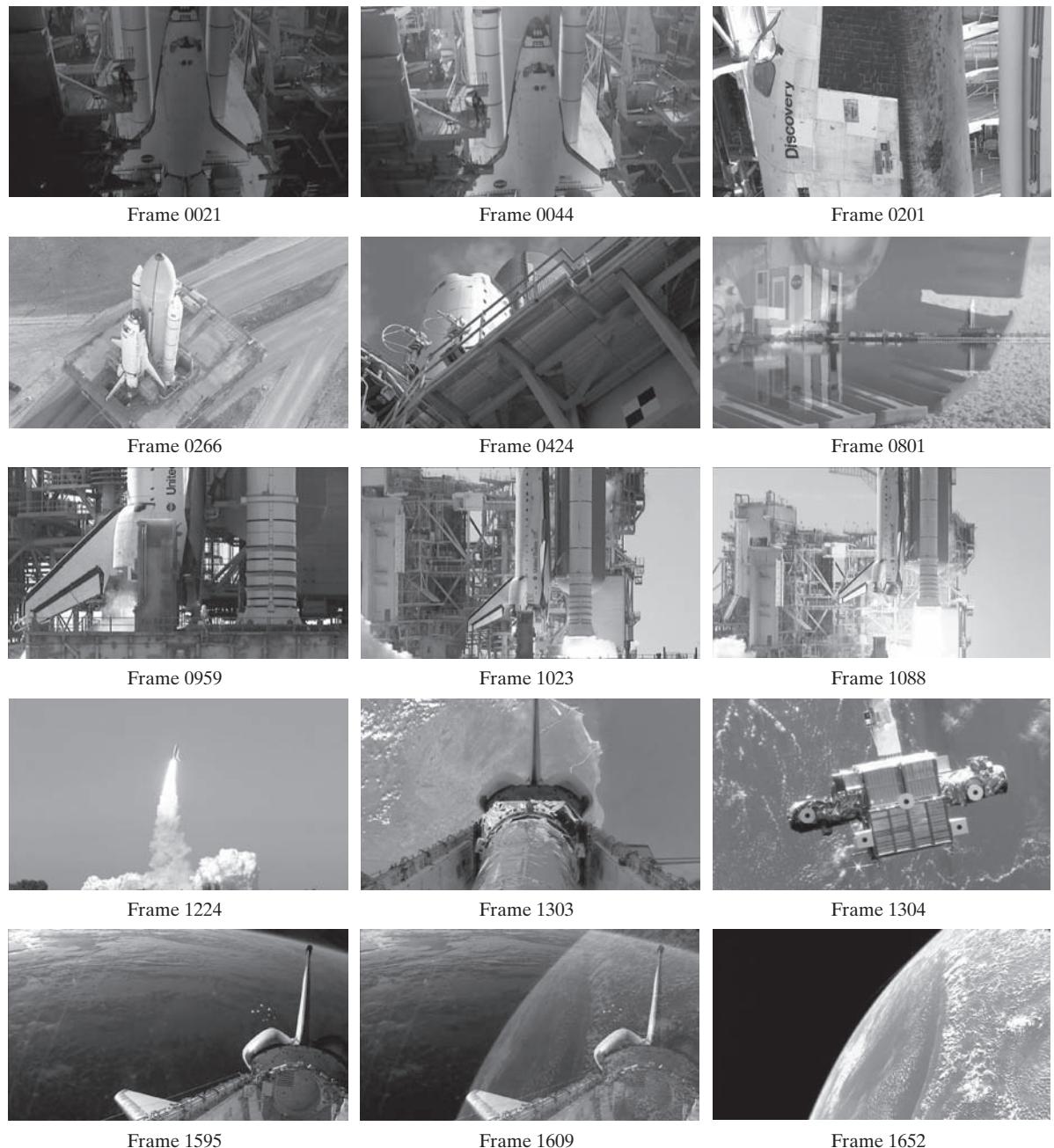


FIGURE 8.37 Fifteen frames from an 1829-frame, 1-minute NASA video. The original video is in HD full color. (Courtesy of NASA.)

In order to accommodate the insertion of the quantization step, the error-free encoder of Fig. 8.30(a) must be altered so the predictions generated by the encoder and decoder are equivalent. As Fig. 8.38(a) shows, this is accomplished by placing the lossy encoder's predictor within a feedback loop, where its input, denoted as $\hat{f}(n)$, is generated as a function of past predictions and the corresponding quantized errors. That is,

$$\hat{f}(n) = \dot{e}(n) + \hat{f}(n) \quad (8-37)$$

where $\hat{f}(n)$ is as defined earlier. This closed loop configuration prevents error buildup at the decoder's output. Note in Fig. 8.38(b) that the output of the decoder is given also by Eq. (8-37).

EXAMPLE 8.23: Delta modulation.

Delta modulation (DM) is a simple but well-known form of lossy predictive coding in which the predictor and quantizer are defined as

$$\hat{f}(n) = \alpha \dot{f}(n - 1) \quad (8-38)$$

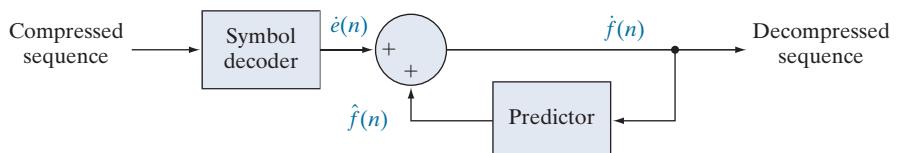
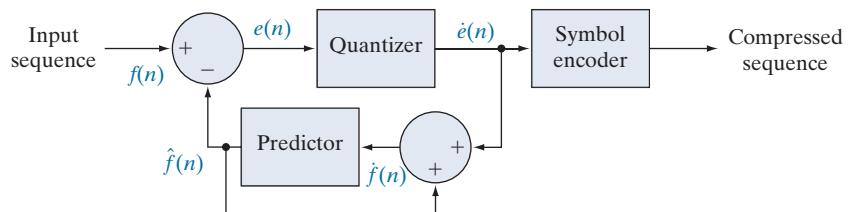
and

$$\dot{e}(n) = \begin{cases} +\zeta & \text{for } e(n) > 0 \\ -\zeta & \text{otherwise} \end{cases} \quad (8-39)$$

where α is a prediction coefficient (normally less than 1), and ζ is a positive constant. The output of the quantizer, $\dot{e}(n)$, can be represented by a single bit [see Fig. 8.39(a)], so the symbol encoder of Fig. 8.38(a) can utilize a 1-bit fixed-length code. The resulting DM code rate is 1 bit/pixel.

Figure 8.39(c) illustrates the mechanics of the delta modulation process, where the calculations needed to compress and reconstruct input sequence {14, 15, 14, 15, 13, 15, 15, 14, 20, 26, 27, 28, 27, 27, 29, 37, 47, 62, 75, 77, 78, 79, 80, 81, 81, 82, 82} with $\alpha = 1$ and $\zeta = 6.5$ are tabulated. The process begins with the

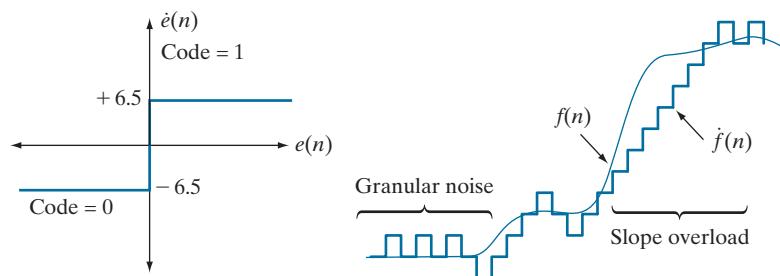
a
b
FIGURE 8.38
A lossy predictive coding model:
(a) encoder;
(b) decoder.



a
b
c

FIGURE 8.39

An example of delta modulation.



Input		Encoder			Decoder		Error	
n	$f(n)$	$\hat{f}(n)$	$e(n)$	$\dot{e}(n)$	$\hat{f}(n)$	$\dot{\hat{f}}(n)$	$f(n) - \hat{f}(n)$	
0	14	—	—	—	14.0	—	14.0	0.0
1	15	14.0	1.0	6.5	20.5	14.0	20.5	-5.5
2	14	20.5	-6.5	-6.5	14.0	20.5	14.0	0.0
3	15	14.0	1.0	6.5	20.5	14.0	20.5	-5.5
.
14	29	20.5	8.5	6.5	27.0	20.5	27.0	2.0
15	37	27.0	10.0	6.5	33.5	27.0	33.5	3.5
16	47	33.5	13.5	6.5	40.0	33.5	40.0	7.0
17	62	40.0	22.0	6.5	46.5	40.0	46.5	15.5
18	75	46.5	28.5	6.5	53.0	46.5	53.0	22.0
19	77	53.0	24.0	6.5	59.6	53.0	59.5	17.5
.

error-free transfer of the first input sample to the decoder. With the initial condition $\hat{f}(0) = f(0) = 14$ established at both the encoder and decoder, the remaining outputs can be computed by repeatedly evaluating Eqs. (8-38), (8-29), (8-39), and (8-37). Thus, when $n = 1$, for example, $\hat{f}(1) = (1)(14) = 14$, $e(1) = 15 - 14 = 1$, $\dot{e}(1) = +6.5$ (because $e(1) > 0$), $\hat{f}(1) = 6.4 + 14 = 20.5$, and the resulting reconstruction error is $(15 - 20.5)$, or -5.5 .

Figure 8.39(b) graphically shows the tabulated data in Fig. 8.39(c). Both the input and completely decoded output [$f(n)$ and $\hat{f}(n)$] are shown. Note that in the rapidly changing area from $n = 14$ to 19 , where ζ was too small to represent the input's largest changes, a distortion known as *slope overload* occurs. Moreover, when ζ was too large to represent the input's smallest changes, as in the relatively smooth region from $n = 0$ to $n = 7$, *granular noise* appears. In images, these two phenomena lead to blurred object edges and grainy or noisy surfaces (that is, distorted smooth areas).

The distortions noted in the preceding example are common to all forms of lossy predictive coding. The severity of these distortions depends on a complex set of interactions between the quantization and prediction methods employed. Despite these interactions, the predictor normally is designed with the assumption of no quantization error, and the quantizer is designed to minimize its own error. That is, the predictor and quantizer are designed independently of each other.

OPTIMAL PREDICTORS

The notation $E\{\cdot\}$ denotes the statistical expectation operator.

In many predictive coding applications, the predictor is chosen to minimize the encoder's mean-squared prediction error

$$E\{e^2(n)\} = E\left\{[f(n) - \hat{f}(n)]^2\right\} \quad (8-40)$$

subject to the constraint that

$$\dot{f}(n) = \dot{e}(n) + \hat{f}(n) \approx e(n) + \hat{f}(n) = f(n) \quad (8-41)$$

and

$$\hat{f}(n) = \sum_{i=1}^m \alpha_i f(n-i) \quad (8-42)$$

That is, the optimization criterion is minimal mean-squared prediction error, the quantization error is assumed to be negligible [$\dot{e}(n) \approx e(n)$], and the prediction is constrained to a linear combination of m previous samples. These restrictions are not essential, but they considerably simplify the analysis and, at the same time, decrease the computational complexity of the predictor. The resulting predictive coding approach is referred to as *differential pulse code modulation* (DPCM).

Under these conditions, the optimal predictor design problem is reduced to the relatively straightforward exercise of selecting the m prediction coefficients that minimize the expression

$$E\{e^2(n)\} = E\left\{\left[f(n) - \sum_{i=1}^m \alpha_i f(n-i)\right]^2\right\} \quad (8-43)$$

Differentiating Eq. (8-43) with respect to each coefficient, equating the derivatives to zero, and solving the resulting set of simultaneous equations under the assumption that $f(n)$ has mean zero and variance σ^2 yields

$$\boldsymbol{\alpha} = \mathbf{R}^{-1} \mathbf{r} \quad (8-44)$$

where \mathbf{R}^{-1} is the inverse of the $m \times m$ autocorrelation matrix

$$\mathbf{R} = \begin{bmatrix} E\{f(n-1)f(n-1)\} & E\{f(n-1)f(n-2)\} & \dots & E\{f(n-1)f(n-m)\} \\ E\{f(n-2)f(n-1)\} & \vdots & \dots & \vdots \\ \vdots & \vdots & \dots & \vdots \\ \vdots & \vdots & \dots & \vdots \\ E\{f(n-m)f(n-1)\} & E\{f(n-m)f(n-2)\} & \dots & E\{f(n-m)f(n-m)\} \end{bmatrix} \quad (8-45)$$

and \mathbf{r} and $\boldsymbol{\alpha}$ are the m -element vectors

$$\mathbf{r} = \begin{bmatrix} E\{f(n)f(n-1)\} \\ E\{f(n)f(n-2)\} \\ \vdots \\ E\{f(n)f(n-m)\} \end{bmatrix} \quad \boldsymbol{\alpha} = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_m \end{bmatrix} \quad (8-46)$$

Thus for any input sequence, the coefficients that minimize Eq. (8-43) can be determined via a series of elementary matrix operations. Moreover, the coefficients depend only on the autocorrelations of the samples in the original sequence. The variance of the prediction error that results from the use of these optimal coefficients is

$$\sigma_e^2 = \sigma^2 - \boldsymbol{\alpha}^T \mathbf{r} = \sigma^2 - \sum_{i=1}^m E\{f(n)f(n-i)\}\alpha_i \quad (8-47)$$

Although the mechanics of evaluating Eq. (8-44) are quite simple, computation of the autocorrelations needed to form \mathbf{R} and \mathbf{r} is so difficult in practice that *local* predictions (those in which the prediction coefficients are computed for each input sequence) are almost never used. In most cases, a set of *global* coefficients is computed by assuming a simple input model and substituting the corresponding autocorrelations into Eqs. (8-45) and (8-46). For instance, when a 2-D Markov image source (see Section 8.1) with separable autocorrelation function

$$E\{f(x,y)f(x-i,y-j)\} = \sigma^2 \rho_v^i \rho_h^j \quad (8-48)$$

and generalized fourth-order linear predictor

$$\begin{aligned} \hat{f}(x,y) = & \alpha_1 f(x,y-1) + \alpha_2 f(x-1,y-1) \\ & + \alpha_3 f(x-1,y) + \alpha_4 f(x-1,y+1) \end{aligned} \quad (8-49)$$

are assumed, the resulting optimal coefficients (Jain [1989]) are

$$\alpha_1 = \rho_h \quad \alpha_2 = -\rho_v \rho_h \quad \alpha_3 = \rho_v \quad \alpha_4 = 0 \quad (8-50)$$

where ρ_h and ρ_v are the horizontal and vertical correlation coefficients, respectively, of the image under consideration.

Finally, the sum of the prediction coefficients in Eq. (8-42) is normally required to be less than or equal to one. That is,

$$\sum_{i=1}^m \alpha_i \leq 1 \quad (8-51)$$

This restriction is made to ensure that the output of the predictor falls within the allowed range of the input, and to reduce the impact of transmission noise [which generally is seen as horizontal streaks in reconstructed images when the input to Fig. 8.38(a) is an image]. Reducing the DPCM decoder's susceptibility to input noise is important, because a single error (under the right circumstances) can propagate to all future outputs. That is, the decoder's output may become unstable. Further restricting Eq. (8-51) to be strictly less than 1 confines the impact of an input error to a small number of outputs.

EXAMPLE 8.24: Comparison of prediction techniques.

Consider the prediction error that results from DPCM coding the monochrome image of Fig. 8.9(a) under the assumption of zero quantization error and with each of four predictors:

$$\hat{f}(x, y) = 0.97f(x, y - 1) \quad (8-52)$$

$$\hat{f}(x, y) = 0.5f(x, y - 1) + 0.5f(x - 1, y) \quad (8-53)$$

$$\hat{f}(x, y) = 0.75f(x, y - 1) + 0.75f(x - 1, y) - 0.5f(x - 1, y - 1) \quad (8-54)$$

$$\hat{f}(x, y) = \begin{cases} 0.97f(x, y - 1) & \text{if } \Delta h \leq \Delta v \\ 0.97f(x - 1, y) & \text{otherwise} \end{cases} \quad (8-55)$$

where $\Delta h = |f(x - 1, y) - f(x - 1, y - 1)|$ and $\Delta v = |f(x, y - 1) - f(x - 1, y - 1)|$ denote the horizontal and vertical gradients at point (x, y) . Equations (8-52) through (8-55) define a relatively robust set of α_i that provide satisfactory performance over a wide range of images. The adaptive predictor of Eq. (8-55) is designed to improve edge rendition by computing a local measure of the directional properties of an image (Δh and Δv), and selecting a predictor specifically tailored to the measured behavior.

Figures 8.40(a) through (d) show the prediction error images that result from using the predictors of Eqs. (8-52) through (8-55). Note that the visually perceptible error decreases as the order of the predictor increases.[†] The standard deviations of the prediction errors follow a similar pattern. They are 11.1, 9.8, 9.1, and 9.7 intensity levels, respectively.

OPTIMAL QUANTIZATION

The staircase quantization function $t = q(s)$ in Fig. 8.41 is an odd function of s [that is, $q(-s) = -q(s)$] that can be described completely by the $L/2$ values of s_i and t_i shown in the first quadrant of the graph. These break points define function discontinuities, and are called the *decision* and *reconstruction levels* of the quantizer. As a matter of convention, s is considered to be mapped to t_i if it lies in the half-open interval $(s_i, s_{i+1}]$.

The quantizer design problem is to select the best s_i and t_i for a particular optimization criterion and input probability density function $p(s)$. If the optimization

[†]Predictors that use more than three or four previous pixels provide little compression gain for the added predictor complexity (Habibi [1971]).

a	b
c	d

FIGURE 8.40

A comparison of four linear prediction techniques.



criterion, which could be either a statistical or psychovisual measure,[†] is the minimization of the mean-squared quantization error (that is $E\{(s_i - t_i)^2\}$) and $p(s)$ is an even function, the conditions for minimal error (Max [1960]) are

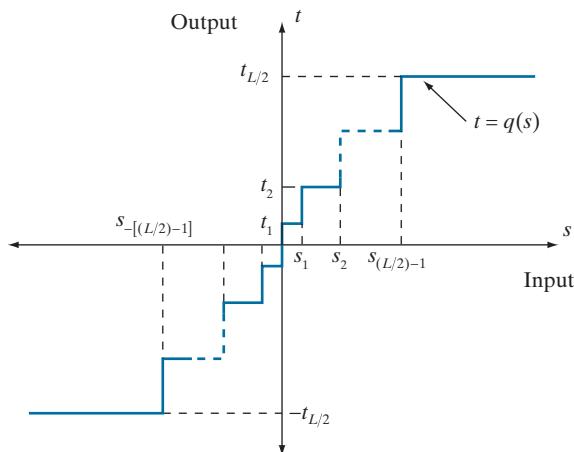
$$\int_{s_{i-1}}^{s_i} (s - t_i)p(s)ds = 0 \quad i = 1, 2, \dots, \frac{L}{2} \quad (8-56)$$

$$s_i = \begin{cases} 0 & i = 0 \\ \frac{t_i + t_{i+1}}{2} & i = 1, 2, \dots, \frac{L}{2} - 1 \\ \infty & i = \frac{L}{2} \end{cases} \quad (8-57)$$

[†]See Netravali [1977] and Limb for more on psychovisual measures.

FIGURE 8.41

A typical quantization function.



and

$$s_{-i} = -s_i \quad t_{-i} = -t_i \quad (8-58)$$

Equation (8-56) indicates that the reconstruction levels are the centroids of the areas under $p(s)$ over the specified decision intervals, whereas Eq. (8-57) indicates that the decision levels are halfway between the reconstruction levels. Equation (8-58) is a consequence of the fact that q is an odd function. For any L , the s_i and t_i that satisfy Eqs. (8-56) through (8-58) are optimal in the mean-squared error sense; the corresponding quantizer is called an L -level *Lloyd-Max* quantizer.

Table 8.13 lists the 2-, 4-, and 8-level Lloyd-Max decision and reconstruction levels for a unit variance Laplacian probability density function [see Eq. (8-34)]. Because obtaining an explicit or closed-form solution to Eqs. (8-56) through (8-58) for most nontrivial $p(s)$ is difficult, these values were generated numerically (Paez and Glisson [1972]). The three quantizers shown provide fixed output rates of 1, 2, and 3 bits/pixel, respectively. As Table 8.13 was constructed for a unit variance distribution, the reconstruction and decision levels for the case of $\sigma \neq 1$ are obtained by multiplying the tabulated values by the standard deviation of the probability density

TABLE 8.13
Lloyd-Max
quantizers for a
Laplacian
probability
density function
of unit variance.

Levels	2		4		8	
	s_i	t_i	s_i	t_i	s_i	t_i
1	∞	0.707	1.102	0.395	0.504	0.222
2			∞	1.810	1.181	0.785
3					2.285	1.576
4					∞	2.994
θ	1.414		1.087		0.731	

function under consideration. The final row of the table lists the step size, θ , that simultaneously satisfies Eqs. (8-56) through (8-58) and the additional constraint that

$$t_i - t_{i-1} = s_i - s_{i-1} = \theta \quad (8-59)$$

If a symbol encoder that utilizes a variable-length code is used in the general lossy predictive encoder of Fig. 8.38(a), an *optimum uniform quantizer* of step size θ will provide a lower code rate (for a Laplacian PDF) than a fixed-length coded Lloyd-Max quantizer with the same output fidelity (O’Neil [1971]).

Although the Lloyd-Max and optimum uniform quantizers are not adaptive, much can be gained from adjusting the quantization levels based on the local behavior of an image. In theory, slowly changing regions can be finely quantized, while the rapidly changing areas are quantized more coarsely. This approach simultaneously reduces both granular noise and slope overload, while requiring only a minimal increase in code rate. The trade-off is increased quantizer complexity.

8.11 WAVELET CODING

With reference to
Tables 8.3–8.5, wavelet
coding is used in the

- JPEG-2000

 compression standard.

As with the block transform coding techniques presented earlier, wavelet coding is based on the idea that the coefficients of a transform that decorrelates the pixels of an image can be coded more efficiently than the original pixels themselves. If the basis functions of the transform (in this case wavelets) pack most of the important visual information into a small number of coefficients, the remaining coefficients can be quantized coarsely or truncated to zero with little image distortion.

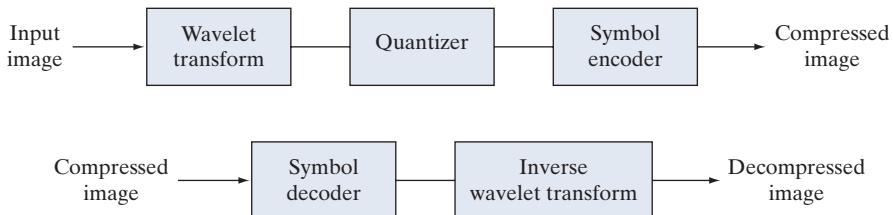
Figure 8.42 shows a typical wavelet coding system. To encode a $2^J \times 2^J$ image, an analyzing wavelet, ψ , and minimum decomposition level, $J - P$, are selected and used to compute the discrete wavelet transform of the image. If the wavelet has a complementary scaling function φ , the fast wavelet transform (see Section 7.10) can be used. In either case, the computed transform converts a large portion of the original image to horizontal, vertical, and diagonal decomposition coefficients with zero mean and Laplacian-like probabilities. Because many of the computed coefficients carry little visual information, they can be quantized and coded to minimize intercoefficient and coding redundancy. Moreover, the quantization can be adapted to exploit any positional correlation across the P decomposition levels. One or more lossless coding methods, such as run-length, Huffman, arithmetic, and bit-plane coding, can be incorporated into the final symbol coding step. Decoding is accomplished by inverting the encoding operations, with the exception of quantization, which cannot be reversed exactly.

The principal difference between the wavelet-based system of Fig. 8.42 and the transform coding system of Fig. 8.21 is the omission of the subimage processing stages of the transform coder. Because wavelet transforms are both computationally efficient and inherently local (i.e., their basis functions are limited in duration), subdivision of the original image is unnecessary. As you will see later in this section, the removal of the subdivision step eliminates the blocking artifact that characterizes DCT-based approximations at high compression ratios.

a
b**FIGURE 8.42**

A wavelet coding system:

- (a) encoder;
- (b) decoder.



WAVELET SELECTION

The wavelets chosen as the basis of the forward and inverse transforms in Fig. 8.42 affect all aspects of wavelet coding system design and performance. They impact directly the computational complexity of the transforms and, less directly, the system's ability to compress and reconstruct images of acceptable error. When the transforming wavelet has a companion scaling function, the transformation can be implemented as a sequence of digital filtering operations, with the number of filter taps equal to the number of nonzero wavelet and scaling vector coefficients. The ability of the wavelet to pack information into a small number of transform coefficients determines its compression and reconstruction performance.

The most widely used expansion functions for wavelet-based compression are the Daubechies wavelets and biorthogonal wavelets. The latter allow useful analysis properties, like the number of vanishing moments (see Section 7.10), to be incorporated into the decomposition filters, while important synthesis properties, like smoothness of reconstruction, are built into the reconstruction filters.

EXAMPLE 8.25: Wavelet bases in wavelet coding.

Figure 8.43 contains four discrete wavelet transforms of Fig. 8.9(a). Haar wavelets, the simplest and only discontinuous wavelets considered in this example, were used as the expansion or basis functions in Fig. 8.43(a). Daubechies wavelets, among the most popular imaging wavelets, were used in Fig. 8.43(b), and symlets, which are an extension of the Daubechies wavelets with increased symmetry, were used in Fig. 8.43(c). The Cohen-Daubechies-Feauveau wavelets employed in Fig. 8.43(d) are included to illustrate the capabilities of biorthogonal wavelets. As in previous results of this type, all detail coefficients were scaled to make the underlying structure more visible, with intensity 128 corresponding to coefficient value 0.

As you can see in Table 8.14, the number of operations involved in the computation of the transforms in Fig. 8.43 increases from 4 to 28 multiplications and additions per coefficient (for each decomposition

TABLE 8.14

Wavelet transform filter taps and zeroed coefficients when truncating the transforms in Fig. 8.43 below 1.5.

Wavelet	Filter Taps (Scaling + Wavelet)	Zeroed Coefficients
Haar	2 + 2	33.3%
Daubechies	8 + 8	40.9%
Symlet	8 + 8	41.2%
Biorthogonal	17 + 11	42.1%

a	b
c	d

FIGURE 8.43

Three-scale wavelet transforms of Fig. 8.9(a) with respect to
 (a) Haar wavelets,
 (b) Daubechies
 wavelets,
 (c) symlets,
 and (d) Cohen-
 Daubechies-Feau-
 veau biorthogonal
 wavelets.



level) as you move from Fig. 8.43(a) to (d). All four transforms were computed using a fast wavelet transform (i.e., filter bank) formulation. Note that as the computational complexity (i.e., the number of filter taps) increases, the information packing performance does as well. When Haar wavelets are employed and the detail coefficients below 1.5 are truncated to zero, 33.8% of the total transform is zeroed. With the more complex biorthogonal wavelets, the number of zeroed coefficients rises to 42.1%, increasing the potential compression by almost 10%.

DECOMPOSITION LEVEL SELECTION

Another factor affecting wavelet coding computational complexity and reconstruction error is the number of transform decomposition levels. Because a P -scale fast wavelet transform involves P filter bank iterations, the number of operations in the computation of the forward and inverse transforms increases with the number of decomposition levels. Moreover, quantizing the increasingly lower-scale

coefficients that result with more decomposition levels affects increasingly larger areas of the reconstructed image. In many applications, like searching image databases or transmitting images for progressive reconstruction, the resolution of the stored or transmitted images, and the scale of the lowest useful approximations, normally determine the number of transform levels.

EXAMPLE 8.26: Decomposition levels in wavelet coding.

Table 8.15 illustrates the effect of decomposition level selection on the coding of Fig. 8.9(a) using biorthogonal wavelets and a fixed global threshold of 25. As in the previous wavelet coding example, only detail coefficients are truncated. The table lists both the percentage of zeroed coefficients and the resulting rms reconstruction errors from Eq. (8-10). Note that the initial decompositions are responsible for the majority of the data compression. There is little change in the number of truncated coefficients above three decomposition levels.

QUANTIZER DESIGN

The most important factor affecting wavelet coding compression and reconstruction error is coefficient quantization. Although the most widely used quantizers are uniform, the effectiveness of the quantization can be improved significantly by (1) introducing a larger quantization interval around zero, called a *dead zone*, or (2) adapting the size of the quantization interval from scale to scale. In either case, the selected quantization intervals must be transmitted to the decoder with the encoded image bit stream. The intervals themselves may be determined heuristically, or computed automatically based on the image being compressed. For example, a global coefficient threshold could be computed as the median of the absolute values of the first-level detail coefficients or as a function of the number of zeroes that are truncated and the amount of energy that is retained in the reconstructed image.

One measure of the energy of a digital signal is the sum of the squared samples.

EXAMPLE 8.27: Dead zone interval selection in wavelet coding.

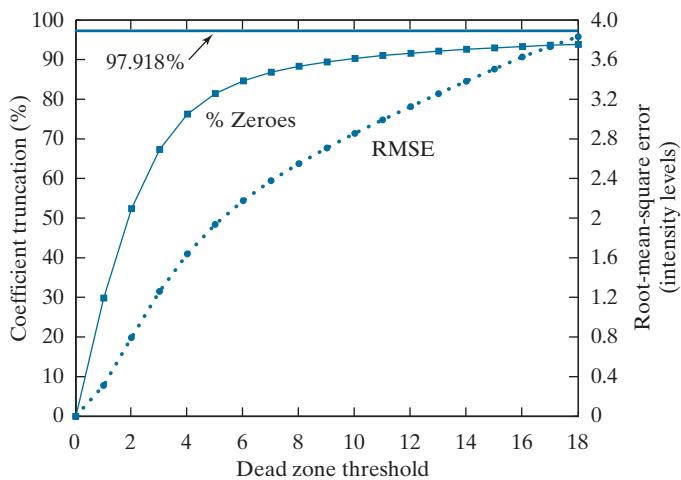
Figure 8.44 illustrates the impact of dead zone interval size on the percentage of truncated detail coefficients for a three-scale biorthogonal wavelet-based encoding of Fig. 8.9(a). As the size of the dead zone increases, the number of truncated coefficients does as well. Above the knee of the curve (i.e., beyond 5),

TABLE 8.15
Decomposition level impact on wavelet coding
the 512×512 image of Fig. 8.9(a).

Decomposition Level (Scales or Filter Bank Iterations)	Approximation Coefficient Image	Truncated Coefficients (%)	Reconstruction Error (rms)
1	256×256	74.7%	3.27
2	128×128	91.7%	4.23
3	64×64	95.1%	4.54
4	32×32	95.6%	4.61
5	16×16	95.5%	4.63

FIGURE 8.44

The impact of dead zone interval selection on wavelet coding.



there is little gain. This is due to the fact that the histogram of the detail coefficients is highly peaked around zero.

The rms reconstruction errors corresponding to the dead zone thresholds in Fig. 8.44 increase from 0 to 1.94 intensity levels at a threshold of 5, and to 3.83 intensity levels for a threshold of 18, where the number of zeroes reaches 93.85%. If every detail coefficient were eliminated, that percentage would increase to about 97.92% (by about 4%), but the reconstruction error would grow to 12.3 intensity levels.

JPEG-2000

JPEG-2000 extends the popular JPEG standard to provide increased flexibility in both the compression of continuous-tone still images and access to the compressed data. For example, portions of a JPEG-2000 compressed image can be extracted for retransmission, storage, display, and/or editing. The standard is based on the wavelet coding techniques just described. Coefficient quantization is adapted to individual scales and subbands, and the quantized coefficients are arithmetically coded on a bit-plane basis (see Sections 8.4 and 8.8). Using the notation of the standard, an image is encoded as follows [ISO/IEC [2000]].

Ssiz is used in the standard to denote intensity resolution.

The irreversible component transform is the component transform used for lossy compression. The component transform itself is not irreversible. A different component transform is used for reversible compression.

The first step of the encoding process is to DC level shift the samples of the *Ssiz*-bit unsigned image to be coded by subtracting 2^{Ssiz-1} . If the image has more than one component, such as the red, green, and blue planes of a color image, each component is shifted individually. If there are exactly three components, they may be optionally decorrelated using a reversible or nonreversible linear combination of the components. The *irreversible component transform* of the standard, for example, is

$$\begin{aligned} Y_0(x, y) &= 0.299I_0(x, y) + 0.587I_1(x, y) + 0.114I_2(x, y) \\ Y_1(x, y) &= -0.16875I_0(x, y) - 0.33126I_1(x, y) + 0.5I_2(x, y) \\ Y_2(x, y) &= 0.5I_0(x, y) - 0.41869I_1(x, y) - 0.08131I_2(x, y) \end{aligned} \quad (8-60)$$

where I_0 , I_1 , and I_2 are the level-shifted input components, and Y_0 , Y_1 , and Y_2 are the corresponding decorrelated components. If the input components are the red, green, and blue planes of a color image, Eq. (8-60) approximates the $R'G'B'$ to $Y'C_bC_r$ color video transform (Poynton [1996]).[†] The goal of the transformation is to improve compression efficiency; transformed components Y_1 and Y_2 are difference images whose histograms are highly peaked around zero.

After the image has been level-shifted and optionally decorrelated, its components can be divided into *tiles*. Tiles are rectangular arrays of pixels that are processed independently. Because an image can have more than one component (e.g., it could be made up of three color components), the tiling process creates *tile components*. Each tile component can be reconstructed independently, providing a simple mechanism for accessing and/or manipulating a limited region of a coded image. For example, an image having a 16:9 aspect ratio could be subdivided into tiles so one of its tiles is a subimage with a 4:3 aspect ratio. That tile then could be reconstructed without accessing the other tiles in the compressed image. If the image is not subdivided into tiles, it is a single tile.

The 1-D discrete wavelet transform of the rows and columns of each tile component is then computed. For error-free compression, the transform is based on a bior-thogonal, 5/3 coefficient scaling and wavelet vector (Le Gall and Tabatabai [1988]). A rounding procedure is defined for non-integer-valued transform coefficients. In lossy applications, a 9/7 coefficient scaling-wavelet vector is employed (Antonini, Barlaud, Mathieu, and Daubechies [1992]). In either case, the transform is computed using the fast wavelet transform of Section 7.10 or via a complementary *lifting-based* approach (Mallat [1999]). For example, in lossy applications, the coefficients used to construct the 9/7 FWT analysis filter bank are given in Table 7.1. The complementary lifting-based implementation involves six sequential “lifting” and “scaling” operations:

Lifting-based implementations are another way to compute wavelet transforms. The coefficients used in the approach are directly related to the FWT filter bank coefficients.

$$\begin{aligned}
 Y(2n+1) &= X(2n+1) + \alpha[X(2n) + X(2n+2)] & i_0 - 3 \leq 2n+1 < i_1 + 3 \\
 Y(2n) &= X(2n) + \beta[Y(2n-1) + Y(2n+1)] & i_0 - 2 \leq 2n < i_1 + 2 \\
 Y(2n+1) &= Y(2n+1) + \gamma[Y(2n) + Y(2n+2)] & i_0 - 1 \leq 2n+1 < i_1 + 1 \\
 Y(2n) &= Y(2n) + \delta[Y(2n-1) + Y(2n+1)] & i_0 \leq 2n < i_1 \\
 Y(2n+1) &= -KY(2n+1) & i_0 \leq 2n+1 < i_1 \\
 Y(2n) &= Y(2n)/K & i_0 \leq 2n < i_1
 \end{aligned} \tag{8-61}$$

Here, X is the tile component being transformed, Y is the resulting transform, and i_0 and i_1 define the position of the tile component within a component. That is, they are the indices of the first sample of the tile-component row or column being transformed and the one immediately following the last sample. Variable

[†] $R'G'B'$ is a gamma-corrected, nonlinear version of a linear CIE (International Commission on Illumination) RGB colorimetry value. Y' is luminance and C_b and C_r are color differences (i.e., scaled $B' - Y'$ and $R' - Y'$ values).

n assumes values based on i_0 , i_1 , and determines which of the six operations is being performed. If $n < i_0$ or $n > i_1$, $X(n)$ is obtained by symmetrically extending X . For example, $X(i_0 - 1) = X(i_0 + 1)$, $X(i_0 - 2) = X(i_0 + 2)$, $X(i_1) = X(i_1 - 2)$, and $X(i_1 + 1) = X(i_1 - 3)$. At the conclusion of the lifting and scaling operations, the even-indexed values of Y are equivalent to the FWT lowpass filtered output; the odd-indexed values of Y correspond to the highpass FWT filtered result. Lifting parameters α , β , γ , and δ are -1.586134342 , -0.052980118 , 0.882911075 , and 0.433506852 , respectively, and scaling factor K is 1.230174105 .

These lifting-based coefficients are specified in the standard.

Recall from Chapter 7 that the DWT decomposes an image into a set of band-limited components called subbands.

The transformation just described produces four subbands; a low-resolution approximation of the tile component and the component's horizontal, vertical, and diagonal frequency characteristics. Repeating the transformation N_L times, with subsequent iterations restricted to the previous decomposition's approximation coefficients, produces an N_L -scale wavelet transform. Adjacent scales are related spatially by powers of 2, and the lowest scale contains the only explicitly defined approximation of the original tile component. As can be surmised from Fig. 8.45, where the notation of the JPEG-2000 standard is summarized for the case of $N_L = 2$, a general N_L -scale transform contains $3N_L + 1$ subbands whose coefficients are denoted a_b for $b = N_L HL, \dots, 1HL, 1LH, 1HH$. The standard does not specify the number of scales to be computed.

When each of the tile components has been processed, the total number of transform coefficients is equal to the number of samples in the original image, but the important visual information is concentrated in a few coefficients. To reduce the number of bits needed to represent the transform, coefficient $a_b(u, v)$ of subband b is quantized to value $q_b(u, v)$ using

$$q_b(u, v) = \text{sign}[a_b(u, v)] \cdot \text{floor}\left[\frac{|a_b(u, v)|}{\Delta_b}\right] \quad (8-62)$$

where the *quantization step size* Δ_b is

$$\Delta_b = 2^{R_b - \varepsilon_b} \left(1 + \frac{\mu_b}{2^{11}}\right) \quad (8-63)$$

Do not confuse the standard's definition of nominal dynamic range with the closely related definition in Chapter 2.

R_b is the *nominal dynamic range* of subband b , while ε_b and μ_b are the number of bits allotted to the *exponent* and *mantissa* of the subband's coefficients. The nominal dynamic range of subband b is the sum of the number of bits used to represent the original image and the *analysis gain* bits for subband b . Subband analysis gain bits follow the simple pattern shown in Fig. 8.45. For example, there are two analysis gain bits for subband $b = 1HH$.

For error-free compression, $\mu_b = 0$, $R_b = \varepsilon_b$, and $\Delta_b = 1$. For irreversible compression, no particular quantization step size is specified in the standard. Instead, the number of exponent and mantissa bits must be provided to the decoder on a subband basis, called *expounded quantization*, or for the $N_L LL$ subband only, called *derived quantization*. In the latter case, the remaining subbands are quantized using

extrapolated $N_L LL$ subband parameters. Letting ε_0 and μ_0 be the number of bits allocated to the $N_L LL$ subband, the extrapolated parameters for subband b are

$$\begin{aligned}\mu_b &= \mu_0 \\ \varepsilon_b &= \varepsilon_0 + n_b - N_L\end{aligned}\quad (8-64)$$

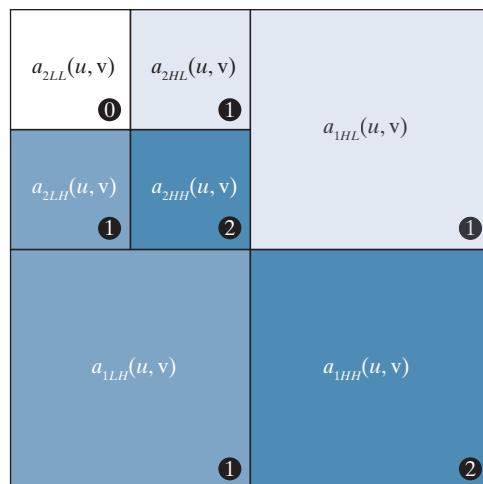
where n_b denotes the number of subband decomposition levels from the original image tile component to subband b .

In the final steps of the encoding process, the coefficients of each transformed tile-component's subbands are arranged into rectangular blocks called *code blocks*, which are coded individually, one bit plane at a time. Starting from the most significant bit plane with a nonzero element, each bit plane is processed in three passes. Each bit (in a bit plane) is coded in only one of the three passes, which are called *significance propagation*, *magnitude refinement*, and *cleanup*. The outputs are then arithmetically coded and grouped with similar passes from other code blocks to form *layers*. A layer is an arbitrary number of groupings of coding passes from each code block. The resulting layers finally are partitioned into *packets*, providing an additional method of extracting a spatial region of interest from the total code stream. Packets are the fundamental unit of the encoded code stream.

JPEG-2000 decoders simply invert the operations previously described. After reconstructing the subbands of the tile-components from the arithmetically coded JPEG-2000 packets, a user-selected number of the subbands is decoded. Although the encoder may have encoded M_b bit planes for a particular subband, the user, due to the embedded nature of the code stream, may choose to decode only N_b bit planes. This amounts to quantizing the coefficients of the code block using a step

FIGURE 8.45

JPEG 2000 two-scale wavelet transform tile-component coefficient notation and analysis gain.



Quantization as defined earlier in the chapter is irreversible. The term “inverse quantized” does not mean that there is no information loss. This process is lossy except for the case of reversible JPEG-2000 compression, where $\mu_b = 0$, $R_b = \varepsilon_b$, and $\Delta_b = 1$.

size of $2^{M_b - N_b} \cdot \Delta_b$. Any nondecoded bits are set to zero and the resulting coefficients, denoted $\bar{q}_b(u, v)$, are inverse quantized using

$$R_{q_b}(u, v) = \begin{cases} (\bar{q}_b(u, v) + r \cdot 2^{M_b - N_b(u, v)}) \cdot \Delta_b & \bar{q}_b(u, v) > 0 \\ (\bar{q}_b(u, v) - r \cdot 2^{M_b - N_b(u, v)}) \cdot \Delta_b & \bar{q}_b(u, v) < 0 \\ 0 & \bar{q}_b(u, v) = 0 \end{cases} \quad (8-65)$$

where $R_{q_b}(u, v)$ denotes an inverse-quantized transform coefficient, and $N_b(u, v)$ is the number of decoded bit planes for $\bar{q}_b(u, v)$. Reconstruction parameter r is chosen by the decoder to produce the best visual or objective quality of reconstruction. Generally, $0 \leq r < 1$, with a common value being $r = 1/2$. The inverse-quantized coefficients then are inverse-transformed by column and by row using an FWT⁻¹ filter bank whose coefficients are obtained from Table 7.1, or via the following lifting-based operations:

$$\begin{aligned} X(2n) &= K \cdot Y(2n) & i_0 - 3 \leq 2n < i_1 + 3 \\ X(2n+1) &= (-1/K)Y(2n+1) & i_0 - 2 \leq 2n - 1 < i_1 + 2 \\ X(2n) &= X(2n) - \delta [X(2n-1) + X(2n+1)] & i_0 - 3 \leq 2n < i_1 + 3 \\ X(2n+1) &= X(2n+1) - \gamma [X(2n) + X(2n+2)] & i_0 - 2 \leq 2n + 1 < i_1 + 2 \\ X(2n) &= X(2n) - \beta [X(2n-1) + X(2n+1)] & i_0 - 1 \leq 2n < i_1 + 1 \\ X(2n+1) &= X(2n+1) - \alpha [X(2n) + X(2n+2)] & i_0 \leq 2n + 1 < i_1 \end{aligned} \quad (8-66)$$

where parameters α , β , γ , δ , and K are as defined for Eq. (8-61). Inverse-quantized coefficient row or column element $Y(n)$ is symmetrically extended when necessary. The final decoding steps are the assembly of the component tiles, inverse component transformation (if required), and DC level shifting. For irreversible coding, the inverse component transformation is

$$\begin{aligned} I_0(x, y) &= Y_0(x, y) + 1.402Y_2(x, y) \\ I_1(x, y) &= Y_0(x, y) - 0.34413Y_1(x, y) - 0.71414Y_2(x, y) \\ I_2(x, y) &= Y_0(x, y) + 1.772Y_1(x, y) \end{aligned} \quad (8-67)$$

and the transformed pixels are shifted by $+2^{Ssiz-1}$.

EXAMPLE 8.28: A comparison of JPEG-2000 wavelet-based coding and JPEG DCT-based compression.

Figure 8.46 shows four JPEG-2000 approximations of the monochrome image in Figure 8.9(a). Successive rows of the figure illustrate increasing levels of compression, including $C = 25, 52, 75$, and 105 . The images in column 1 are decompressed JPEG-2000 encodings. The differences between these images and the original image [see Fig. 8.9(a)] are shown in the second column, and the third column contains a zoomed portion of the reconstructions in column 1. Because the compression ratios for the first two rows are virtually identical to the compression ratios in Example 8.18, these results can be compared (both qualitatively and quantitatively) to the JPEG transform-based results in Figs. 8.29(a) through (f).



FIGURE 8.46 Four JPEG-2000 approximations of Fig. 8.9(a). Each row contains a result after compression and reconstruction, the scaled difference between the result and the original image, and a zoomed portion of the reconstructed image. (Compare the results in rows 1 and 2 with the JPEG results in Fig. 8.29.).

A visual comparison of the error images in rows 1 and 2 of Fig. 8.46 with the corresponding images in Figs. 8.29(b) and (e) reveals a noticeable decrease of error in the JPEG-2000 results—3.86 and 5.77 intensity levels, as opposed to 5.4 and 10.7 intensity levels for the JPEG results. The computed errors favor the wavelet-based results at both compression levels. Besides decreasing reconstruction error, wavelet coding dramatically increases (in a subjective sense) image quality. Note that the blocking artifact that dominated the JPEG results [see Figs. 8.29(c) and (f)] is not present in Fig. 8.46. Finally, we note that the compression achieved in rows 3 and 4 of Fig. 8.46 is not practical with JPEG. JPEG-2000 provides useable images that are compressed by more than 100:1, with the most objectionable degradation being increased image blur.

8.12 DIGITAL IMAGE WATERMARKING

The methods and standards of Sections 8.2 through 8.11 make the distribution of images (in photographs or videos) on digital media and over the Internet practical. Unfortunately, the images so distributed can be copied repeatedly and without error, putting the rights of their owners at risk. Even when encrypted for distribution, images are unprotected after decryption. One way to discourage illegal duplication is to insert one or more items of information, collectively called a *watermark*, into potentially vulnerable images in such a way that the watermarks are inseparable from the images themselves. As integral parts of the watermarked images, they protect the rights of their owners in a variety of ways, including:

1. *Copyright identification.* Watermarks can provide information that serves as proof of ownership when the rights of the owner have been infringed.
2. *User identification or fingerprinting.* The identity of legal users can be encoded in watermarks and used to identify sources of illegal copies.
3. *Authenticity determination.* The presence of a watermark can guarantee that an image has not been altered, assuming the watermark is designed to be destroyed by any modification of the image.
4. *Automated monitoring.* Watermarks can be monitored by systems that track when and where images are used (e.g., programs that search the Web for images placed on Web pages). Monitoring is useful for royalty collection and/or the location of illegal users.
5. *Copy protection.* Watermarks can specify rules of image usage and copying (e.g., to DVD players).

In this section, we provide a brief overview of *digital image watermarking*, which is the process of inserting data into an image in such a way that it can be used to make an assertion about the image. The methods described have little in common with the compression techniques presented in the previous sections (although they do involve the coding of information). In fact, watermarking and compression are in some ways opposites. While the objective in compression is to reduce the amount of data used to represent images, the goal in watermarking is to add information and data (i.e., watermarks) to them. As will be seen in the remainder of the section, the watermarks themselves can be either visible or invisible.

A *visible watermark* is an opaque or semi-transparent subimage or image that is placed on top of another image (i.e., the image being watermarked) so that it is obvious to the viewer. Television networks often place visible watermarks (fashioned after their logos) in the upper or lower right-hand corner of the television screen. As the following example illustrates, visible watermarking typically is performed in the spatial domain.

EXAMPLE 8.29: A simple visible watermark.

The image in Fig. 8.47(b) is the lower right-hand quadrant of the image in Fig. 8.9(a) with a scaled version of the watermark in Fig. 8.47(a) overlaid on top of it. Letting f_w denote the watermarked image, we can express it as a linear combination of the unmarked image f and watermark w using

$$f_w = (1 - \alpha)f + \alpha w \quad (8-68)$$

where constant α controls the relative visibility of the watermark and the underlying image. If α is 1, the watermark is opaque and the underlying image is completely obscured. As α approaches 0, more of the underlying image and less of the watermark is seen. In general, $0 < \alpha \leq 1$; in Fig. 8.47(b), $\alpha = 0.3$. Figure 8.47(c) is the computed difference (scaled in intensity) between the watermarked image in (b) and the unmarked image in Fig. 8.9(a). Intensity 128 represents a difference of 0. Note that the underlying image is clearly visible through the “semi-transparent” watermark. This is evident in both Fig. 8.47(b) and the difference image in Fig. 8.47(c).

Unlike the visible watermark of the previous example, *invisible watermarks* cannot be seen with the naked eye. They are imperceptible but can be recovered with an appropriate decoding algorithm. Invisibility is assured by inserting them as visually redundant information [information that the human visual system ignores or cannot

a
b c

FIGURE 8.47

A simple visible watermark:
 (a) watermark;
 (b) the watermarked image;
 and
 (c) the difference between the watermarked image and the original (non-watermarked) image.



perceive (see Section 8.1)]. Figure 8.48(a) provides a simple example. Because the least significant bits of an 8-bit image have virtually no effect on our perception of the image, the watermark from Fig. 8.47(a) was inserted or “hidden” in its two least significant bits. Using the notation introduced above, we let

$$f_w = 4\left(\frac{f}{4}\right) + \frac{w}{64} \quad (8-69)$$

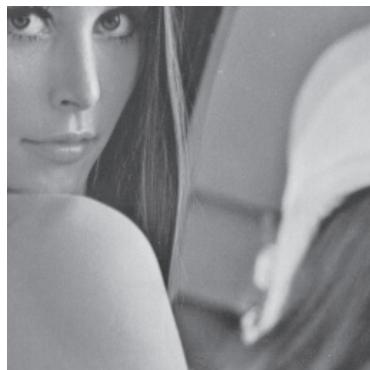
and use unsigned integer arithmetic to perform the calculations. Dividing and multiplying by 4 sets the two least significant bits of f to 0, dividing w by 64 shifts its two most significant bits into the two least significant bit positions, and adding the two results generates the *LSB watermarked image*. Note that the embedded watermark is not visible in Fig. 8.48(a). By zeroing the most significant 6 bits of this image and scaling the remaining values to the full intensity range, however, the watermark can be extracted as in Fig. 8.48(b).

An important property of invisible watermarks is their resistance to both accidental and intentional attempts to remove them. *Fragile invisible watermarks* are destroyed by any modification of the images in which they are embedded. In some applications, like image authentication, this is a desirable characteristic. As Figs. 8.48(c) and (d) show, the LSB watermarked image in Fig. 8.48(a) contains a fragile invisible watermark. If the image in (a) is compressed and decompressed using lossy JPEG, the watermark is destroyed. Figure 8.48(c) is the result after com-

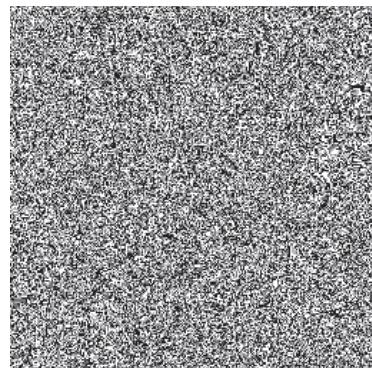
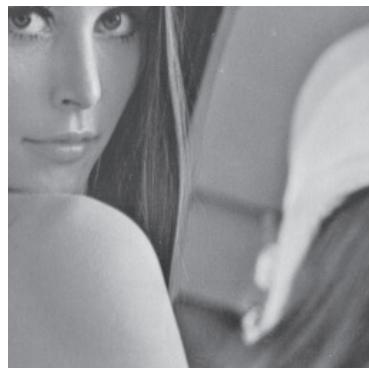
a	b
c	d

FIGURE 8.48

A simple invisible watermark:
 (a) watermarked image;
 (b) the extracted watermark;
 (c) the watermarked image after high quality JPEG compression and decompression; and
 (d) the extracted watermark from (c).



Digital Image Processing



pressing and decompressing Fig. 8.48(a); the rms error is 2.1 bits. If we try to extract the watermark from this image using the same method as in (b), the result is unintelligible [see Fig. 8.48(d)]. Although lossy compression and decompression preserved the important visual information in the image, the fragile watermark was destroyed.

Robust invisible watermarks are designed to survive image modification, whether the so-called *attacks* are inadvertent or intentional. Common inadvertent attacks include lossy compression, linear and nonlinear filtering, cropping, rotation, resampling, and the like. Intentional attacks range from printing and rescanning to adding additional watermarks and/or noise. Of course, it is unnecessary to withstand attacks that leave the image itself unusable.

Figure 8.49 shows the basic components of a typical image watermarking system. The encoder in Fig. 8.49(a) inserts watermark w_i into image f_i producing watermarked image f_{w_i} ; the complementary decoder in (b) extracts and validates the presence of w_i in watermarked input f_{w_i} or unmarked input f_j . If w_i is visible, the decoder is not needed. If it is invisible, the decoder may or may not require a copy of f_i and w_i [shown in blue in Fig. 8.49(b)] to do its job. If f_i and/or w_i are used, the watermarking system is known as a *private* or *restricted-key* system; if not, it is a *public* or *unrestricted-key* system. Because the decoder must process both marked and unmarked images, w_\emptyset is used in Fig. 8.49(b) to denote the absence of a mark. Finally, we note that to determine the presence of w_i in an image, the decoder must correlate extracted watermark w_j with w_i and compare the result to a predefined threshold. The threshold sets the degree of similarity that is acceptable for a “match.”

EXAMPLE 8.30: A DCT-based invisible robust watermark.

Mark insertion and *extraction* can be performed in the spatial domain, as in the previous examples, or in the transform domain. Figures 8.50(a) and (c) show two watermarked versions of the image in Fig. 8.9(a) using the DCT-based watermarking approach outlined here (Cox et al. [1997]):

1. Compute the 2-D DCT of the image to be watermarked.
2. Locate its K largest coefficients, c_1, c_2, \dots, c_K , by magnitude.
3. Create a watermark by generating a K -element pseudo-random sequence of numbers, $\omega_1, \omega_2, \dots, \omega_K$, taken from a Gaussian distribution with mean $\mu = 0$ and variance $\sigma^2 = 1$. (Note: A pseudo-random number sequence approximates the properties of random numbers. It is not truly random because it depends on a predetermined initial value.)
4. Embed the watermark from Step 3 into the K largest DCT coefficients from Step 2 using the following equation

$$c'_i = c_i \cdot (1 + \alpha\omega_i) \quad 1 \leq i \leq K \quad (8-70)$$

for a specified constant $\alpha > 0$ (that controls the extent to which ω_i alters c_i). Replace the original c_i with the computed c'_i from Eq. (8-70). (For the images in Fig. 8.50, $\alpha = 0.1$ and $K = 1000$.)

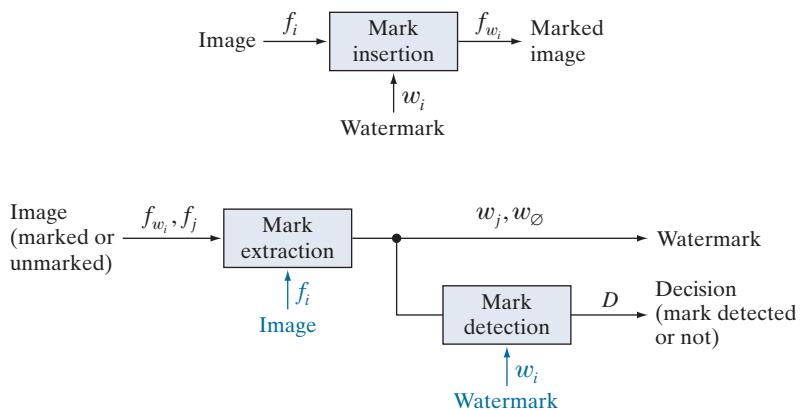
5. Compute the inverse DCT of the result from Step 4.

By employing watermarks made from pseudo-random numbers and spreading them across an image’s perceptually significant frequency components, α can be made small, reducing watermark visibility. At

a
b

FIGURE 8.49

A typical image watermarking system:
(a) encoder;
(b) decoder.



a b
c d

FIGURE 8.50

(a) and (c) Two watermarked versions of Fig. 8.9(a);
(b) and (d) the differences (scaled in intensity) between the watermarked versions and the unmarked image. These two images show the intensity contribution (although scaled dramatically) of the pseudo-random watermarks on the original image.



the same time, watermark security is kept high because (1) the watermarks are composed of pseudo-random numbers with no obvious structure, (2) the watermarks are embedded in multiple frequency components with spatial impact over the entire 2-D image (so their location is not obvious) and (3) attacks against them tend to degrade the image as well (i.e., the image's most important frequency components must be altered to affect the watermarks).

Figures 8.50(b) and (d) make the changes in image intensity that result from the pseudo-random numbers that are embedded in the DCT coefficients of the watermarked images in Figs. 8.50(a) and (c)

visible. Obviously, the pseudo-random numbers must have an effect (even if too small to see) on the watermarked images. To display the effect, the images in Figs. 8.50(a) and (c) were subtracted from the unmarked image in Fig. 8.9(a) and scaled in intensity to the range [0, 255]. Figures 8.50(b) and (d) are the resulting images; they show the 2-D spatial contributions of the pseudo-random numbers. Because they have been scaled, however, you cannot simply add these images to the image in Fig. 8.9(a) and get the watermarked images in Figs. 8.50(a) and (c). As can be seen in Figs. 8.50(a) and (c), their actual intensity perturbations are small to negligible.

To determine whether a particular image is a copy of a previously watermarked image with watermark $\omega_1, \omega_2, \dots, \omega_K$ and DCT coefficients c_1, c_2, \dots, c_K , we use the following procedure:

1. Compute the 2-D DCT of the image in question.
2. Extract the K DCT coefficients (in the positions corresponding to c_1, c_2, \dots, c_K of Step 2 in the watermarking procedure) and denote the coefficients as $\hat{c}_1, \hat{c}_2, \dots, \hat{c}_K$. If the image in question is the previously watermarked image (without modification), $\hat{c}_i = c'_i$ for $1 \leq i \leq K$. If it is a modified copy of the watermarked image (i.e., it has undergone some sort of attack), $\hat{c}_i \approx c'_i$ for $1 \leq i \leq K$ (the \hat{c}_i will be approximations of the c'_i). Otherwise, the image in question will be an unmarked image or an image with a completely different watermark, and the \hat{c}_i will bear no resemblance to the original \hat{c}_i .
3. Compute watermark $\hat{\omega}_1, \hat{\omega}_2, \dots, \hat{\omega}_K$ using

$$\hat{\omega}_i = \frac{\hat{c}_i - c_i}{\alpha c_i} \quad \text{for } 1 \leq i \leq k \quad (8-71)$$

Recall that watermarks are sequences of pseudo-random numbers.

4. Measure the similarity of $\hat{\omega}_1, \hat{\omega}_2, \dots, \hat{\omega}_K$ (from Step 3) and $\omega_1, \omega_2, \dots, \omega_K$ (from Step 3 of the watermarking procedure) using a metric such as the correlation coefficient

$$\gamma = \frac{\sum_{i=1}^K (\hat{\omega}_i - \bar{\hat{\omega}})(\omega_i - \bar{\omega})}{\sqrt{\sum_{i=1}^K (\hat{\omega}_i - \bar{\hat{\omega}})^2 \cdot \sum_{i=1}^K (\omega_i - \bar{\omega})^2}} \quad 1 \leq i \leq K \quad (8-72)$$

where $\bar{\omega}$ and $\bar{\hat{\omega}}$ are the means of the two K -element watermarks. (Note: Correlation coefficients are discussed in detail in Section 12.3.)

5. Compare the measured similarity, γ , to a predefined threshold, T , and make a binary detection decision:

$$D = \begin{cases} 1 & \text{if } \gamma \geq T \\ 0 & \text{otherwise} \end{cases} \quad (8-73)$$

In other words, $D = 1$ indicates that watermark $\omega_1, \omega_2, \dots, \omega_K$ is present (with respect to the specified threshold, T); $D = 0$ indicates that it was not.

Using this procedure, the original watermarked image in Fig. 8.50(a), measured against itself, yields a correlation coefficient of 0.9999, i.e., $\gamma = 0.9999$. It is an unmistakable match. In a similar manner, the image in Fig. 8.50(b), when measured against the image in Fig. 8.50(a), results in a γ of 0.0417. It could not be mistaken for the watermarked image in Fig. 8.50(a) because the correlation coefficient is so low.

To conclude the section, we note that the DCT-based watermarking approach of the previous example is fairly resistant to watermark attacks, partly because it is a private or restricted-key method. Restricted-key methods are always more resilient than their unrestricted-key counterparts. Using the watermarked image in Fig. 8.50(a), Fig. 8.51 illustrates the ability of the method to withstand a variety of common attacks. As can be seen in the figure, watermark detection is quite good over the range of attacks that were implemented; the resulting correlation coefficients (shown under each image in the figure) vary from 0.3113 to 0.9945. When subjected to a high quality but lossy (resulting in an rms error of 7 intensities) JPEG compression and decompression, $\gamma = 0.9945$. Even when the compression and reconstruction yields an rms error of 10 intensity levels, $\gamma = 0.7395$; and the usability of this image has been significantly degraded. Significant smoothing by spatial filtering and the addition of Gaussian noise do not reduce the correlation coefficient below 0.8230. However, histogram equalization reduces γ to 0.5210; and rotation has the largest effect; reducing γ to 0.3313. All attacks, except for the lossy JPEG compression and reconstruction in Fig. 8.51(a), have significantly reduced the usability of the original watermarked image.

Summary, References, and Further Reading

The principal objectives of this chapter were to present the theoretic foundation of digital image compression, to describe the most commonly used compression methods, and to introduce the related area of digital image watermarking. Although the level of the presentation is introductory in nature, the references provide an entry into the extensive body of literature dealing with the topics discussed. As evidenced by the international standards listed in Tables 8.3 through 8.5, 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. Image watermarking is becoming increasingly important as more and more images are distributed in compressed digital form.

The introductory material of the chapter, which is generally confined to Section 8.1, is basic to image compression, and may be found in one form or another in most of the general image processing books cited at the end of Chapter 1. For additional information on the human visual system, see Netravali and Limb [1980], as well as Huang [1966], Schreiber and Knapp [1958], and the references cited at the end of Chapter 2. For more on information theory, see the book website or Abramson [1963], Blahut [1987], and Berger [1971]. Shannon's classic paper, "A Mathematical Theory of Communication" [1948], lays the foundation for the area and is another excellent reference. Subjective fidelity criteria are discussed in Frendendall and Behrend [1960]. Throughout the chapter, a variety of compression standards are used in examples. Most of them were implemented using Adobe Photoshop (with freely available compression plug-ins) and/or MATLAB, which is described in Gonzalez et al. [2004]. Compression standards, as a rule, are lengthy and complex; we have not attempted to cover any of them in their entirety. For more information on a particular standard, see the published documents of the appropriate standards organization—the International Standards Organization, International Electrotechnical Commission, and/or the International Telecommunications Union.

The lossy and error-free compression techniques described in Sections 8.2 through 8.11 and watermarking techniques in Section 8.12 are, for the most part, based on the original papers cited in the text. The algorithms covered are representative of the work in this area, but are by no means exhaustive. The material on LZW coding has its origins in the work of Ziv and Lempel [1977, 1978]. The material on arithmetic coding follows the development in Witten, Neal, and Cleary [1987]. One of the more important implementations of arithmetic coding is summarized in Pennebaker et al. [1988]. For a good discussion of lossless predictive coding, see the tutorial by Rabbani and Jones [1991]. The adaptive predictor of Eq. (8-55) is from Graham [1958]. For more on motion compensation, see S. Solari [1997], which also contains an introduction to general video compression and compression standards, and Mitchell et al. [1997]. The DCT-based watermarking technique in Section 8.12 is based on the paper by Cox et al. [1997]. For more on watermarking, see the books by Cox et al. [2001] and Parhi and Nishitani [1999]. See also the paper by S. Mohanty [1999].



a	b	c
d	e	f

FIGURE 8.51 Attacks on the watermarked image in Fig. 8.50(a): (a) lossy JPEG compression and decompression with an rms error of seven intensity levels; (b) lossy JPEG compression and decompression with an rms error of 10 intensity levels (note the blocking artifact); (c) smoothing by spatial filtering; (d) the addition of Gaussian noise; (e) histogram equalization; and (f) rotation. Each image is a modified version of the watermarked image in Fig. 8.50(a). After modification, they retain their watermarks to varying degrees, as indicated by the correlation coefficients below each image.

Many survey articles have been devoted to the field of image compression. Noteworthy are Netravali and Limb [1980], A. K. Jain [1981], a special issue on picture communication systems in the *IEEE Transactions on Communications* [1981], a special issue on the encoding of graphics in the *Proceedings of IEEE* [1980], a special issue on visual communication systems in the *Proceedings of the IEEE* [1985], a special issue on image sequence compression in the *IEEE Transactions on Image Processing* [1994], and a special issue on vector quantization in the *IEEE Transactions on Image Processing* [1996]. In addition, most issues of the *IEEE Transactions on Image Processing*, *IEEE Transactions on Circuits and Systems for Video Technology*, and *IEEE Transactions on Multimedia* include articles on video and still image compression, motion compensation, and watermarking.

Problems

Solutions to the problems marked with an asterisk (*) are in the DIP4E Student Support Package (consult the book website: www.ImageProcessingPlace.com).

8.1 Answer the following.

- (a) Can variable-length coding procedures be used to compress a histogram equalized image with 2^n intensity levels? Explain.
- (b) Can such an image contain spatial or temporal redundancies that could be exploited for data compression?

8.2 One variation of run-length coding involves (1) coding only the runs of 0's or 1's (not both) and (2) assigning a special code to the start of each line to reduce the effect of transmission errors. One possible code pair is (x_k, r_k) , where x_k and r_k represent the k th run's starting coordinate and run length, respectively. The code $(0, 0)$ is used to signal each new line.

- (a) Derive a general expression for the maximum average runs per scan line required to guarantee data compression when run-length coding a $2^n \times 2^n$ binary image.
- (b) Compute the maximum allowable value for $n = 10$.

8.3 Consider an 8-pixel line of intensity data, $\{108, 139, 135, 244, 172, 173, 56, 99\}$. If it is uniformly quantized with 4-bit accuracy, compute the rms error and rms signal-to-noise ratios for the quantized data.

8.4* Although quantization results in information loss, it is sometimes invisible to the eye. For example, when 8-bit pixels are uniformly quantized to fewer bits/pixel, false contouring often occurs. It can be reduced or eliminated using *improved gray-scale (IGS) quantization*. A sum (initially set to zero) is formed from the current 8-bit intensity value and the four least significant bits of the previously generated sum. If the four most significant bits of the intensity value are 1111_2 , however, 0000_2 is added instead. The four most significant bits of the resulting sum are used as the coded pixel value.

- (a) Construct the IGS code for the intensity data in Problem 8.3.

(b) Compute the rms error and rms signal-to-noise ratios for the IGS data.

8.5 A 1024×1024 8-bit image with 5.3 bits/pixel entropy [computed from its histogram using Eq. (8-7)] is to be Huffman coded.

- (a) What is the maximum compression that can be expected?
- (b) Will it be obtained?
- (c) If a greater level of lossless compression is required, what else can be done?

8.6* The base e unit of information is commonly called a *nat*, and the base-10 information unit is called a *Hartley*. Compute the conversion factors needed to relate these units to the base-2 unit of information (the bit).

8.7* Prove that, for a zero-memory source with q symbols, the maximum value of the entropy is $\log q$, which is achieved if and only if all source symbols are equiprobable. [Hint: Consider the quantity $\log q - H(z)$ and note the inequality $\ln x \leq x - 1$.]

8.8 Answer the following.

- (a) How many unique Huffman codes are there for a three-symbol source?
- (b) Construct them.

8.9 Consider the simple 4×8 , 8-bit image:

21	21	21	95	169	243	243	243
21	21	21	95	169	243	243	243
21	21	21	95	169	243	243	243
21	21	21	95	169	243	243	243

- (a) Compute the entropy of the image.
- (b) Compress the image using Huffman coding.
- (c) Compute the compression achieved and the effectiveness of the Huffman coding.
- (d)* Consider Huffman encoding pairs of pixels rather than individual pixels. That is, consider the image to be produced by the second extension of the zero-memory source

that produced the original image. What is the entropy of the image when looked at as pairs of pixels?

- (e) Consider coding the differences between adjacent pixels. What is the entropy of the new difference image? What does this tell us about compressing the image?
- (f) Explain the entropy differences in (a), (d) and (e).

8.10 Using the Huffman code in Fig. 8.8, decode the encoded string 010100000101011110100.

8.11 Compute Golomb code $G_3(n)$ for $0 \leq n \leq 15$.

8.12 Write a general procedure for decoding Golomb code $G_m(n)$.

8.13 Why is it not possible to compute the Huffman code of the nonnegative integers, $n \geq 0$, with the probability mass function of Eq. (8-13)?

8.14 Compute exponential Golomb code $G_{\text{exp}}^2(n)$ for $0 \leq n \leq 15$.

8.15* Write a general procedure for decoding exponential Golomb code $G_{\text{exp}}^k(n)$.

8.16 Plot the optimal Golomb coding parameter m as a function of ρ for $0 < \rho < 1$ in Eq. (8-14).

8.17 Given a four-symbol source $\{a, b, c, d\}$ with source probabilities $\{0.1, 0.4, 0.3, 0.2\}$, arithmetically encode the sequence *bbadc*.

8.18* The arithmetic decoding process is the reverse of the encoding procedure. Decode the message 0.23355 given the coding model

Symbol	Probability
a	0.2
e	0.3
i	0.1
o	0.2
u	0.1
!	0.1

8.19 Use the LZW coding algorithm to encode the 7-bit ASCII string “aaaaaaaaaa”.

8.20* Devise an algorithm for decoding the LZW encoded output of Example 8.7. Since the dictionary that was used during the encoding is not available, the code book must be reproduced as the output is decoded.

8.21 Decode the BMP encoded sequence $\{3, 4, 5, 6, 0, 3, 103, 125, 67, 0, 2, 47\}$.

8.22 Do the following:

(a) Construct the entire 4-bit Gray code.

(b) Create a general procedure for converting a Gray-coded number to its binary equivalent and use it to decode 0111010100111.

8.23 Use the CCITT Group 4 compression algorithm to code the second line of the following two-line segment:

0110011100111111100001

11111110001110000111111

Assume that the initial reference element a_0 is located on the first pixel of the second line segment. (*Note:* Employ the CCITT 2-D code table from the book website.)

8.24* Do the following.

(a) List all the members of JPEG DC coefficient difference category 3.

(b) Compute their default Huffman codes using the appropriate Huffman code table from the book website.

8.25 How many computations are required to find the optimal motion vector of a macroblock of size 8×8 using the MAD optimality criterion, single pixel precision, and a maximum allowable displacement of 8 pixels? What would it become for $\frac{1}{4}$ pixel precision?

8.26 What are the advantages of using B-frames for motion compensation?

8.27* Draw the block diagram of the companion motion compensated video decoder for the encoder in Fig. 8.36.

8.28 An image whose autocorrelation function is of the form of Eq. (8-48) with $\rho_h = 0$ is to be DPCM coded using a second-order predictor.

(a) Form the autocorrelation matrix \mathbf{R} and vector \mathbf{r} .

(b) Find the optimal prediction coefficients.

(c) Compute the variance of the prediction error that would result from using the optimal coefficients.

- 8.29*** Derive the Lloyd-Max decision and reconstruction levels for $L = 4$ and the uniform probability density function

$$p(s) = \begin{cases} \frac{1}{2A} & -A \leq s \leq A \\ 0 & \text{otherwise} \end{cases}$$

- 8.30** A radiologist from a well-known research hospital recently attended a medical conference at which a system that could transmit 4096×4096 12-bit digitized X-ray images over standard T1 (1.544 Mb/s) phone lines was exhibited. The system transmitted the images in a compressed format using a progressive technique in which a reasonably good approximation of the X-ray was first reconstructed at the viewing station, then refined gradually to produce an error-free display. The transmission of the data needed to generate the first approximation took approximately 5 or 6 s. Refinements were made every 5 or 6 s (on the average) for the next 1 min, with the first and last refinements having the most and least significant impact on the reconstructed X-ray, respectively. The physician was favorably impressed with the system, because she could begin her diagnosis by using the first approximation of the X-ray and complete it as the error-free reconstruction of the X-ray was being generated. Upon returning to her office, she submitted a purchase request to the hospital administrator. Unfortunately, the hospital was on a relatively tight budget, which recently had been stretched by the hiring of an aspiring young electrical engineering graduate. To

appease the radiologist, the administrator gave the young engineer the task of designing such a system. (He thought it might be cheaper to design and build a similar system in-house. The hospital currently owned some of the elements of such a system, but the transmission of the raw X-ray data took more than 2 min.) The administrator asked the engineer to have an initial block diagram by the afternoon staff meeting. With little time and only a copy of *Digital Image Processing* from his recent school days in hand, the engineer was able to devise a system conceptually to satisfy the transmission and associated compression requirements. Construct a conceptual block diagram of such a system, specifying the compression techniques you would recommend.

- 8.31** Show that the lifting-based wavelet transform defined by Eq. (8-61) is equivalent to the traditional FWT filter bank implementation using the coefficients in Table 7.1. Define the filter coefficients in terms of α , β , γ , δ , and K .
- 8.32** Compute the quantization step sizes of the subbands for a JPEG-2000 encoded image in which derived quantization is used and 8 bits are allotted to the mantissa and exponent of the $2LL$ subband.
- 8.33** How would you add a visible watermark to an image in the frequency domain?
- 8.34*** Design an invisible watermarking system based on the discrete Fourier transform.
- 8.35** Design an invisible watermarking system based on the discrete wavelet transform.

9

Morphological Image Processing



In form and feature, face and limb,
I grew so like my brother
That folks got taking me for him
And each for one another.

Henry Sambrook Leigh, Carols of Cockayne, The Twins

Preview

The word *morphology* commonly denotes a branch of biology that deals with the form and structure of animals and plants. We use the same word here in the context of *mathematical morphology* as a tool for extracting image components that are useful in the representation and description of region shape, such as boundaries, skeletons, and the convex hull. We are interested also in morphological techniques for pre- or postprocessing, such as morphological filtering, thinning, and pruning.

In the following sections, we will develop a number of fundamental concepts in mathematical morphology, and illustrate how they are applied in image processing. The material in this chapter begins a transition from methods whose inputs and outputs are images, to methods whose outputs are image attributes, for tasks such as object extraction and description. Morphology is one of several tools developed in the remainder of the book—such as segmentation, feature extraction, and object recognition—that form the foundation of techniques for extracting “meaning” from an image. The material in the following sections of this chapter deals with methods for processing both binary and grayscale images.

Upon completion of this chapter, readers should:

- Understand basic concepts of mathematical morphology, and how to apply them to digital image processing.
- Be familiar with the tools used for binary image morphology, including erosion, dilation, opening, closing, and how to combine them to generate more complex tools.
- Be able to develop algorithms based on binary image morphology for performing tasks such as morphological smoothing, edge detection, extracting connected components, and skeletonizing.
- Be familiar with how binary image morphology can be extended to grayscale images.
- Be able to develop algorithms for grayscale image processing for tasks such as textural segmentation, granulometry, computing grayscale image gradients, and others.

9.1 PRELIMINARIES

Before proceeding, you will find it helpful to review the discussion in Section 2.4 dealing with representing images, the discussion on connectivity in Section 2.5, and the discussion on sets in Section 2.6.

The language of mathematical morphology is set theory. As such, morphology offers a unified and powerful approach to numerous image processing problems. When working with images, sets in mathematical morphology represent objects in those images. In binary images, the sets in question are members of the 2-D integer space Z^2 , where each element of a set is a tuple (2-D vector) whose coordinates are the coordinates of an object (typically foreground) pixel in the image. Grayscale digital images can be represented as sets whose components are in Z^3 . In this case, two components of each element of the set refer to the coordinates of a pixel, and the third corresponds to its discrete intensity value. Sets in higher dimensional spaces can contain other image attributes, such as color and time-varying components.

Morphological operations are defined in terms of sets. In image processing, we use morphology with two types of sets of pixels: *objects* and *structuring elements* (SE's). Typically, objects are defined as sets of foreground pixels. Structuring elements can be specified in terms of both foreground and background pixels. In addition, structuring elements sometimes contain so-called “don't care” elements, denoted by \times , signifying that the value of that particular element in the SE does not matter. In this sense, the value can be ignored, or it can be made to fit a desired value in the evaluation of an expression; for example, it might take on the value of a pixel in an image in applications in which value matching is the objective.

Because the images with which we work are rectangular arrays, and sets in general are of arbitrary shape, applications of morphology in image processing require that sets be embedded in rectangular arrays. In forming such arrays, we assign a background value to all pixels that are not members of object sets. The top row in Fig. 9.1 shows an example. On the left are sets in the graphical format you are accustomed to seeing in book figures. In the center, the sets have been embedded in a rectangular background (white) to form a graphical image.[†] On the right, we show a digital image (notice the grid) which is the format we use for digital image processing.

Structuring elements are defined in the same manner, and the second row in Fig. 9.1 shows an example. There is an important difference between the way we represent digital images and digital structuring elements. Observe on the top right that there is a border of background pixels surrounding the objects, while there is none in the SE. As you will learn shortly, structuring elements are used in a form similar to spatial convolution kernels (see Fig. 3.28), and the image border just described is similar to the padding we discussed in Section 3.4 and 3.5. The operations are different in morphology, but the padding and sliding operations are the same as in convolution.

In addition to the set definitions given in Section 2.6, the concept of set reflection and translation are used extensively in morphology in connection with structuring elements. The *reflection* of a set (structuring element) B about its origin, denoted by \tilde{B} , is defined as

[†]Sets are shown as drawings of objects (e.g. squares and triangles) of arbitrary shape. A graphical image contains sets that have been embedded into a background to form a rectangular array. When we intend for a drawing to be interpreted as a *digital image* (or structuring element), we include a grid in illustrations that might otherwise be ambiguous. Objects in all drawings are shaded, and the background is shown in white. When working with actual binary images, we say that objects are *foreground* pixels. All other pixels are *background*.

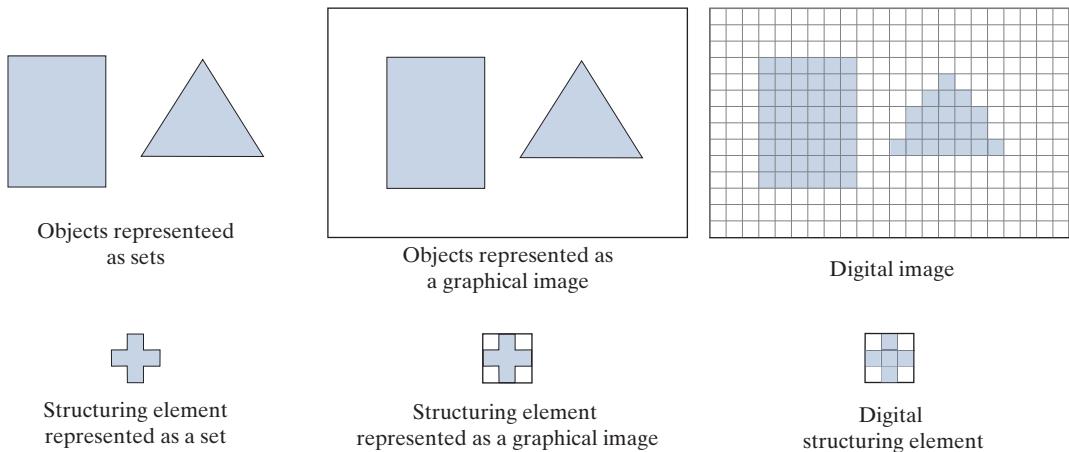


FIGURE 9.1 Top row. *Left:* Objects represented as graphical sets. *Center:* Objects embedded in a background to form a graphical image. *Right:* Object and background are digitized to form a digital image (note the grid). Second row: Example of a structuring element represented as a set, a graphical image, and finally as a digital SE.

$$\hat{B} = \{w \mid w = -b, \text{ for } b \in B\} \quad (9-1)$$

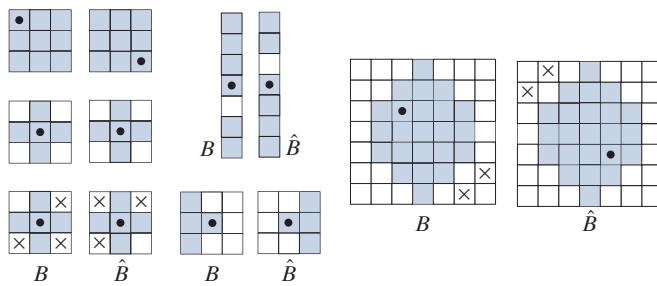
That is, if B is a set of points in 2-D, then \hat{B} is the set of points in B whose (x, y) coordinates have been replaced by $(-x, -y)$. Figure 9.2 shows several examples of digital sets (structuring elements) and their reflection. The dot denotes the origin of the SE. Note that reflection consists simply of rotating an SE by 180° about its origin, and that all elements, including the background and don't care elements, are rotated.

The *translation* of a set B by point $z = (z_1, z_2)$, denoted $(B)_z$, is defined as

$$(B)_z = \{c \mid c = b + z, \text{ for } b \in B\} \quad (9-2)$$

That is, if B is a set of pixels in 2-D, then $(B)_z$ is the set of pixels in B whose (x, y) coordinates have been replaced by $(x + z_1, y + z_2)$. This construct is used to translate (slide) a structuring element over an image, and each location perform a set

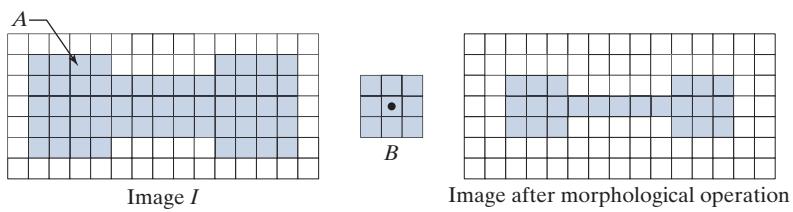
FIGURE 9.2
Structuring elements and their reflections about the origin (the x 's are don't care elements, and the dots denote the origin). Reflection is rotation by 180° of an SE about its origin.



a b c

FIGURE 9.3

- (a) A binary image containing one object (set), A . (b) A structuring element, B .
 (c) Image resulting from a morphological operation (see text).



operation between the structuring element and the area of the image directly under it, as we explained in Fig. 3.28 for correlation and convolution. Both reflection and translation are defined with respect to the *origin* of B .

As an introduction to how morphological operations between images and structuring elements are performed, consider Fig 9.3, which shows a simple binary image, I , consisting of an object (set) A , shown shaded, and a 3×3 SE whose elements are all 1's (foreground pixels). The background pixels (0's) are shown in white. We are interested in performing the following morphological operations: (1) form a new image, of the same size as I , consisting only of background values initially, (2) translate (slide) B over image I , and (3) at each increment of translation, if B is *completely* contained in A , mark the *location* of the origin of B as a *foreground* pixel in the new image; otherwise, leave it as a *background* point. Figure 9.3(c) is the result after the origin of B has visited every element of I . We see that, when the origin of B is on a border element of A , part of B ceases to be contained in A , thus eliminating that location of the origin of B as a possible foreground point of the new image. The net result is that the boundary of set A is *eroded*, as Fig. 9.3(e) shows. Because of the way in which we defined the operation, the maximum excursion needed for B in I is when the origin of B (which is at its center) is contained in A . With B being of size 3×3 , the narrowest background padding we needed was one pixel wide, as shown in Fig. 9.3(a). By using the smallest border needed for an operation, we keep the drawings smaller. In practice, we specify the width of padding based on the maximum dimensions of the structuring elements used, regardless of the operations being performed.

The reason we generally specify the padding border to be of the same dimensions as B , is that some morphological operations are defined for an entire structuring element, and cannot be interpreted with respect to the location of its origin.

When we use terminology such as “the structuring element B is contained in set A ,” we mean *specifically* that the *foreground* elements of B overlap *only* elements of A . This becomes an important issue when B also contains background and, possibly, don't care elements. Also, we use set A to denote *all* foreground pixels of I . Those foreground elements can be a *single* object, as in Fig. 9.3, or they can represent *disjoint* subsets of foreground elements, as in the first row of Fig. 9.1. We will discuss binary images and structuring elements from Sections 9.2 through 9.7. Then, in Section 9.8, we will extend the binary ideas to grayscale images and structuring elements.

9.2 EROSION AND DILATION

We begin the discussion of morphology by studying two operations: *erosion* and *dilation*. These operations are fundamental to morphological processing. In fact, many of the morphological algorithms discussed in this chapter are based on these two primitive operations.

EROSION

Remember, set A can represent (be the union of) multiple disjoint sets of foreground pixels (i.e., objects).

Morphological expressions are written in terms of structuring elements and a set, A , of foreground pixels, or in terms of structuring elements and an image, I , that contains A . We consider the former approach first. With A and B as sets in Z^2 , the *erosion* of A by B , denoted $A \ominus B$, is defined as

$$A \ominus B = \{z \mid (B)_z \subseteq A\} \quad (9-3)$$

where A is a set of foreground pixels, B is a structuring element, and the z 's are foreground values (1's). In words, this equation indicates that the erosion of A by B is the set of all points z such that B , translated by z , is contained in A . (Remember, displacement is defined with respect to the *origin* of B .) Equation (9-3) is the formulation that resulted in the *foreground* pixels of the image in Fig. 9.3(c).

As noted, we work with sets of foreground pixels embedded in a set of background pixels to form a complete image, I . Thus, inputs and outputs of our morphological procedures are images, not individual sets. We *could* make this fact explicit by writing Eq. (9-3) as

$$I \ominus B = \{z \mid (B)_z \subseteq A \text{ and } A \subseteq I\} \cup \{A^c \mid A^c \subseteq I\} \quad (9-4)$$

where I is a rectangular array of foreground and background pixels. The contents of the first braces say the same thing as Eq. (9-3), with the added clarification that A is a subset of (i.e., is contained in) I . The union with the operation inside the second set of braces “adds” the pixels that are not in subset A (i.e., A^c , which is the set of background pixels) to the result from the first braces, requiring also that the background pixels be part of the rectangle defined by I . In words, all this equation says is that erosion of I by B is the set of all points, z , such that B , translated by z , is contained in A . The equation also makes explicit that A is contained in I , that the result is embedded in a set of background pixels, and that the entire process is of the same size as I .

Of course, we do not use the cumbersome notation of Eq. (9-4), which we show only to emphasize an important point. Instead, we use the notation $A \ominus B$ when a morphological operation uses *only* foreground elements, and $I \ominus B$ when the operation uses foreground *and* background elements. This distinction may seem trivial, but suppose that we want to perform erosion with Eq. (9-3), using the foreground elements of the structuring element in the last column in Fig. 9.2. This structuring element also has background elements, but Eq. (9-3) assumes that B only has foreground elements. In fact, erosion is *defined* only for operations between foreground elements, so writing $I \ominus B$ would be meaningless without the “explanation” embedded in Eq. (9-4). To avoid confusion, we use A in morphological expressions when the operation involves only foreground elements, and I when the operation also involves background and/or “don’t-care” elements. We also avoid using standard morphological symbols like \ominus when working with “mixed” SEs. For example, later in Eq. (9-17) we use the symbol \circledast in the expression $I \circledast B = \{z \mid (B)_z \subseteq I\}$, which has the same *form* as Eq. (9-3), but instead involves an entire image and the mixed-value SE in the last column of Fig. 9.2. As you will see, using SE's with mixed values adds considerable power to morphological operations.

Returning to our discussion of Eq. (9-3), because the statement that B has to be contained in A is equivalent to B not sharing any common elements with the background (i.e., the set complement of A), we can express erosion equivalently as

$$A \ominus B = \left\{ z \mid (B)_z \cap A^c = \emptyset \right\} \quad (9-5)$$

where, as defined in Section 2.6, \emptyset is the empty set.

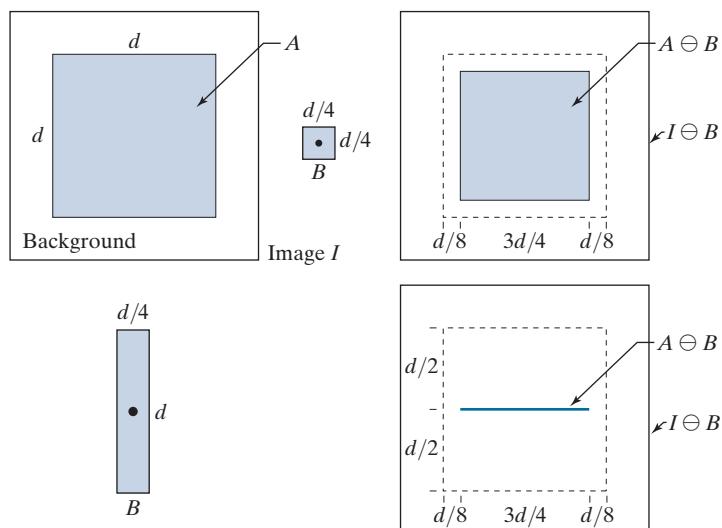
Figure 9.4 shows an example of erosion. The elements of set A (shaded) are the foreground pixels of image I , and, as before, the background is shown in white. The solid boundary inside the dashed boundary in Fig. 9.4(c) is the limit beyond which further displacements of the origin of B would cause some elements of the structuring element to cease being completely contained in A . Thus, the locus of points (locations of the origin of B) within (and including) this boundary constitutes the foreground elements of the erosion of A by B . We show the resulting erosion shaded in Fig. 9.4(c), and the background as white. Erosion is the set of values of z that satisfy Eqs. (9-3) or (9-5). The boundary of A is shown dashed in Figs. 9.4(c) and (e) as a reference; it is not part of the erosion. Figure 9.4(d) shows an elongated structuring element, and Fig. 9.4(e) shows the erosion of A by this element. Note that the original object was eroded to a line. As you can see, the result of erosion is controlled by the shape of the structuring element. In both cases, the assumption is that the image was padded to accommodate all excursions of B , and that the result was cropped to the same size as the original image, just as we did with images processed by spatial convolution in Chapter 3.

Equations (9-3) and (9-5) are not the only definitions of erosion (see Problems 9.12 and 9.13 for two additional, equivalent definitions). However, the former equations have the advantage of being more intuitive when the structuring element B is viewed as if it were a spatial kernel that slides over a set, as in convolution.

a	b	c
d	e	

FIGURE 9.4

- (a) Image I , consisting of a set (object) A , and background.
- (b) Square SE, B (the dot is the origin).
- (c) Erosion of A by B (shown shaded in the resulting image).
- (d) Elongated SE.
- (e) Erosion of A by B . (The erosion is a line.) The dotted border in (c) and (e) is the boundary of A , shown for reference.



EXAMPLE 9.1: Using erosion to remove image components.

Figure 9.5(a) is a binary image depicting a simple wire-bond mask. As mentioned previously, we generally show the foreground pixels in binary images in white and the background in black. Suppose that we want to remove the lines connecting the center region to the border pads in Fig. 9.5(a). Eroding the image (i.e., eroding the *foreground* pixels of the image) with a square structuring element of size 11×11 whose components are all 1's removed most of the lines, as Fig. 9.5(b) shows. The reason that the two vertical lines in the center were thinned but not removed completely is that their width is greater than 11 pixels. Changing the SE size to 15×15 elements and eroding the original image again did remove all the connecting lines, as Fig. 9.5(c) shows. An alternate approach would have been to erode the image in Fig. 9.5(b) again, using the same 11×11 , or smaller, SE. Increasing the size of the structuring element even more would eliminate larger components. For example, the connecting lines and the border pads can be removed with a structuring element of size 45×45 elements applied to the original image, as Fig. 9.5(d) shows.

We see from this example that erosion shrinks or thins objects in a binary image. In fact, we can view erosion as a *morphological filtering* operation in which image details smaller than the structuring element are filtered (removed) from the image. In Fig. 9.5, erosion performed the function of a “line filter.” We will return to the concept of morphological filters in Sections 9.4 and 9.8.

DILATION

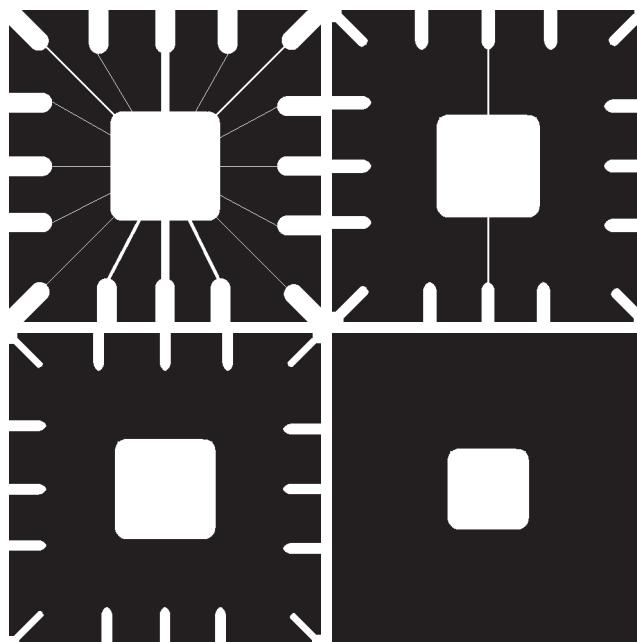
With A and B as sets in Z^2 , the dilation of A by B , denoted as $A \oplus B$, is defined as

$$A \oplus B = \{z \mid (\hat{B})_z \cap A \neq \emptyset\} \quad (9-6)$$

a	b
c	d

FIGURE 9.5

Using erosion to remove image components.
 (a) A 486×486 binary image of a wire-bond mask in which foreground pixels are shown in white.
 (b)–(d) Image eroded using square structuring elements of sizes 11×11 , 15×15 , and 45×45 elements, respectively, all valued 1.



This equation is based on reflecting B about its origin and translating the reflection by z , as in erosion. The dilation of A by B then is the set of all displacements, z , such that the foreground elements of \hat{B} overlap at least one element of A . (Remember, z is the displacement of the origin of \hat{B} .) Based on this interpretation, Eq. (9-6) can be written equivalently as

$$A \oplus B = \left\{ z \mid [(\hat{B})_z \cap A] \subseteq A \right\} \quad (9-7)$$

Equations (9-6) and (9-7) are not the only definitions of dilation currently in use (see Problems 9.14 and 9.15 for two different, yet equivalent, definitions). As with erosion, the preceding definitions have the advantage of being more intuitive when structuring element B is viewed as a convolution kernel. As noted earlier, the basic process of flipping (rotating) B about its origin and then successively displacing it so that it slides over set A is analogous to spatial convolution. However, keep in mind that dilation is based on set operations and therefore is a nonlinear operation, whereas convolution is a sum of products, which is a linear operation.

Unlike erosion, which is a shrinking or thinning operation, dilation “grows” or “thickens” objects in a binary image. The manner and extent of this thickening is controlled by the shape and size of the structuring element used. Figure 9.6(a) shows the same object used in Fig. 9.4 (the background area is larger to accommodate the dilation), and Fig. 9.6(b) shows a structuring element (in this case $\hat{B} = B$ because the SE is symmetric about its origin). The dashed line in Fig. 9.6(c) shows the boundary of the original object for reference, and the solid line shows the limit beyond which any further displacements of the origin of \hat{B} by z would cause the intersection of \hat{B} and A to be empty. Therefore, all points on and inside this boundary constitute the dilation of A by B . Figure 9.6(d) shows a structuring element designed to achieve more dilation vertically than horizontally, and Fig. 9.6(e) shows the dilation achieved with this element.

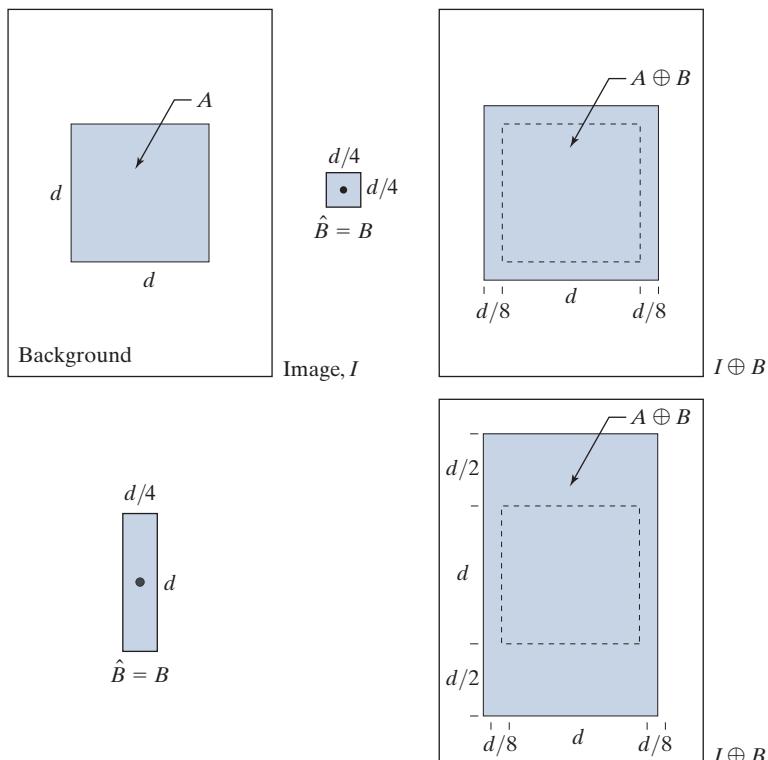
EXAMPLE 9.2: Using dilation to repair broken characters in an image.

One of the simplest applications of dilation is for bridging gaps. Figure 9.7(a) shows the same image with broken characters that we studied in Fig. 4.48 in connection with lowpass filtering. The maximum length of the breaks is known to be two pixels. Figure 9.7(b) shows a structuring element that can be used for repairing the gaps. As noted earlier, we use white (1) to denote the foreground and black (0) for the background when working with images. Figure 9.7(c) shows the result of dilating the original image with the structuring element. The gaps were bridged. One important advantage of the morphological approach over the lowpass filtering method we used to bridge the gaps in Fig. 4.48 is that the morphological method resulted directly in a binary image. Lowpass filtering, on the other hand, started with a binary image and produced a grayscale image that would require thresholding to convert it back to binary form (we will discuss thresholding in Chapter 10). Observe that set A in this application consists of numerous disjointed objects of foreground pixels.

a	b	c
d	e	

FIGURE 9.6

- (a) Image I , composed of set (object) A and background.
- (b) Square SE (the dot is the origin).
- (c) Dilation of A by B (shown shaded).
- (d) Elongated SE.
- (e) Dilation of A by this element. The dotted line in (c) and (e) is the boundary of A , shown for reference.



a	c
b	

FIGURE 9.7

- (a) Low-resolution text showing broken characters (see magnified view).
- (b) Structuring element.
- (c) Dilation of (a) by (b). Broken segments were joined.

Historically, certain computer programs were written using only two digits rather than four to define the applicable year. Accordingly, the company's software may recognize a date using "00" as 1900 rather than the year 2000.



Historically, certain computer programs were written using only two digits rather than four to define the applicable year. Accordingly, the company's software may recognize a date using "00" as 1900 rather than the year 2000.



1	1	1
1	1	1
1	1	1

DUALITY

Erosion and dilation are *duals* of each other with respect to set complementation and reflection. That is,

$$(A \ominus B)^c = A^c \oplus \hat{B} \quad (9-8)$$

and

$$(A \oplus B)^c = A^c \ominus \hat{B} \quad (9-9)$$

Equation (9-8) indicates that erosion of A by B is the complement of the dilation of A^c by \hat{B} , and vice versa. The duality property is useful when the structuring element values are symmetric with respect to its origin (as often is the case), so that $\hat{B} = B$. Then, we can obtain the erosion of A simply by dilating its background (i.e., dilating A^c) with the same structuring element and complementing the result. Similar comments apply to Eq. (9-9).

We proceed to prove formally the validity of Eq. (9-8) in order to illustrate a typical approach for establishing the validity of morphological expressions. Starting with the definition of erosion, it follows that

$$(A \ominus B)^c = \left\{ z \mid (B)_z \subseteq A \right\}^c$$

If set $(B)_z$ is contained in A , then it follows that $(B)_z \cap A^c = \emptyset$, in which case the preceding expression becomes

$$(A \ominus B)^c = \left\{ z \mid (B)_z \cap A^c = \emptyset \right\}^c$$

But the *complement* of the set of z 's that satisfy $(B)_z \cap A^c = \emptyset$ is the set of z 's such that $(B)_z \cap A^c \neq \emptyset$. Therefore,

$$\begin{aligned} (A \ominus B)^c &= \left\{ z \mid (B)_z \cap A^c \neq \emptyset \right\}^c \\ &= A^c \oplus \hat{B} \end{aligned}$$

where the last step follows from the definition of dilation in Eq. (9-6) and its equivalent form in Eq. (9-7). This concludes the proof. A similar line of reasoning can be used to prove Eq. (9-9) (see Problem 9.16).

9.3 OPENING AND CLOSING

As you saw in the previous section, dilation expands the components of a set and erosion shrinks it. In this section, we discuss two other important morphological operations: opening and closing. Opening generally smoothes the contour of an object, breaks narrow isthmuses, and eliminates thin protrusions. Closing also tends

to smooth sections of contours, but, as opposed to opening, it generally fuses narrow breaks and long thin gulfs, eliminates small holes, and fills gaps in the contour.

The *opening* of set A by structuring element B , denoted by $A \circ B$, is defined as

$$A \circ B = (A \ominus B) \oplus B \quad (9-10)$$

Thus, the opening A by B is the erosion of A by B , followed by a dilation of the result by B .

Similarly, the *closing* of set A by structuring element B , denoted $A \bullet B$, is defined as

$$A \bullet B = (A \oplus B) \ominus B \quad (9-11)$$

which says that the closing of A by B is simply the dilation of A by B , followed by erosion of the result by B .

Equation (9-10) has a simple geometrical interpretation: The opening of A by B is the union of all the translations of B so that B fits entirely in A . Figure 9.8(a) shows an image containing a set (object) A and Fig. 9.8(b) is a solid, circular structuring element, B . Figure 9.8(c) shows some of the translations of B such that it is contained within A , and the set shown shaded in Fig. 9.8(d) is the union of all such possible translations. Observe that, in this case, the opening is a set composed of two disjoint subsets, resulting from the fact that B could not fit in the narrow segment in the center of A . As you will see shortly, the ability to eliminate regions narrower than the structuring element is one of the key features of morphological opening.

The interpretation that the opening of A by B is the union of all the translations of B such that B fits entirely within A can be written in equation form as

$$A \circ B = \bigcup \{(B)_z \mid (B)_z \subseteq A\} \quad (9-12)$$

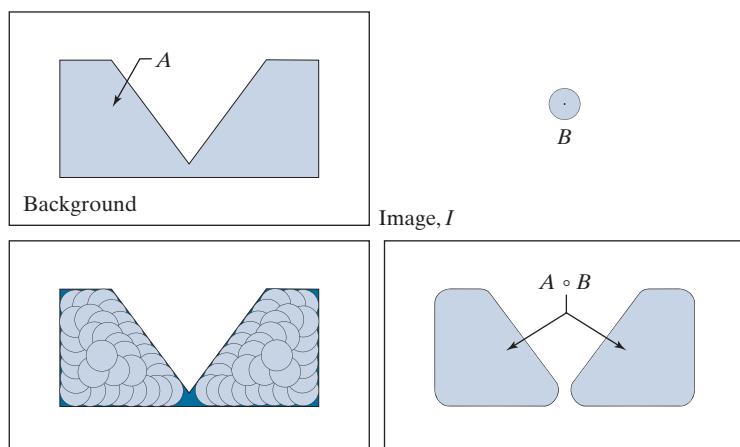
where \bigcup denotes the union of the sets inside the braces.

When a circular structuring element is used for opening, the analogy is often made of the shape of the opening being determined by a “rolling ball” reaching as far as it can on the inner boundary of a set. For morphological closing the ball rolls outside, and the shape of the closing is determined by how far the ball can reach into the boundary.

a
b
c
d

FIGURE 9.8

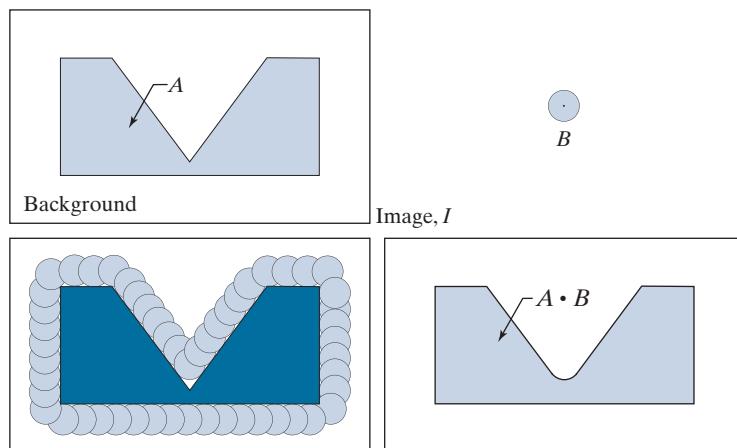
- (a) Image I , composed of set (object) A and background.
- (b) Structuring element, B .
- (c) Translations of B while being contained in A . (A is shown dark for clarity.)
- (d) Opening of A by B .



a	b
c	d

FIGURE 9.9

- (a) Image I , composed of set (object) A , and background.
 (b) Structuring element B .
 (c) Translations of B such that B does not overlap any part of A . (A is shown dark for clarity.)
 (d) Closing of A by B .



Closing has a similar geometric interpretation, except that now we translate B outside A . The closing is then the *complement* of the union of all translations of B that *do not* overlap A . Figure 9.9 illustrates this concept. Note that the boundary of the closing is determined by the furthest points B could reach without going inside any part of A . Based on this interpretation, we can write the closing of A by B as

$$A \bullet B = \left[\bigcup \left\{ (B)_z \mid (B)_z \cap A = \emptyset \right\} \right]^c \quad (9-13)$$

EXAMPLE 9.3: Morphological opening and closing.

Figure 9.10 shows in more detail the process and properties of opening and closing. Unlike Figs. 9.8 and 9.9, whose main objectives are overall geometrical interpretations, this figure shows the individual processes and also pays more attention to the relationship between the scale of the final results and the size of the structuring elements.

Figure 9.10(a) shows an image containing a single object (set) A , and a disk structuring element. Figure 9.10(b) shows various positions of the structuring element during erosion. This process resulted in the disjoint set in Fig. 9.10(c). Note how the bridge between the two main sections was eliminated. Its width was thin in relation to the diameter of the structuring element, which could not be completely contained in this part of the set, thus violating the definition of erosion. The same was true of the two rightmost members of the object. Protruding elements where the disk did not fit were eliminated. Figure 9.10(d) shows the process of dilating the eroded set, and Fig. 9.10(e) shows the final result of opening. Morphological opening removes regions that cannot contain the structuring element, smoothes object contours, breaks thin connections, and removes thin protrusions.

Figures 9.10(f) through (i) show the results of closing A with the same structuring element. As with opening, closing also smoothes the contours of objects. However, unlike opening, closing tends to join narrow breaks, fills long thin gulfs, and fills objects smaller than the structuring element. In this example, the principal result of closing was that it filled the small gulf on the left of set A .

As with erosion and dilation, opening and closing are duals of each other with respect to set complementation and reflection:

$$(A \circ B)^c = (A^c \bullet \hat{B}) \quad (9-14)$$

and

$$(A \bullet B)^c = (A^c \circ \hat{B}) \quad (9-15)$$

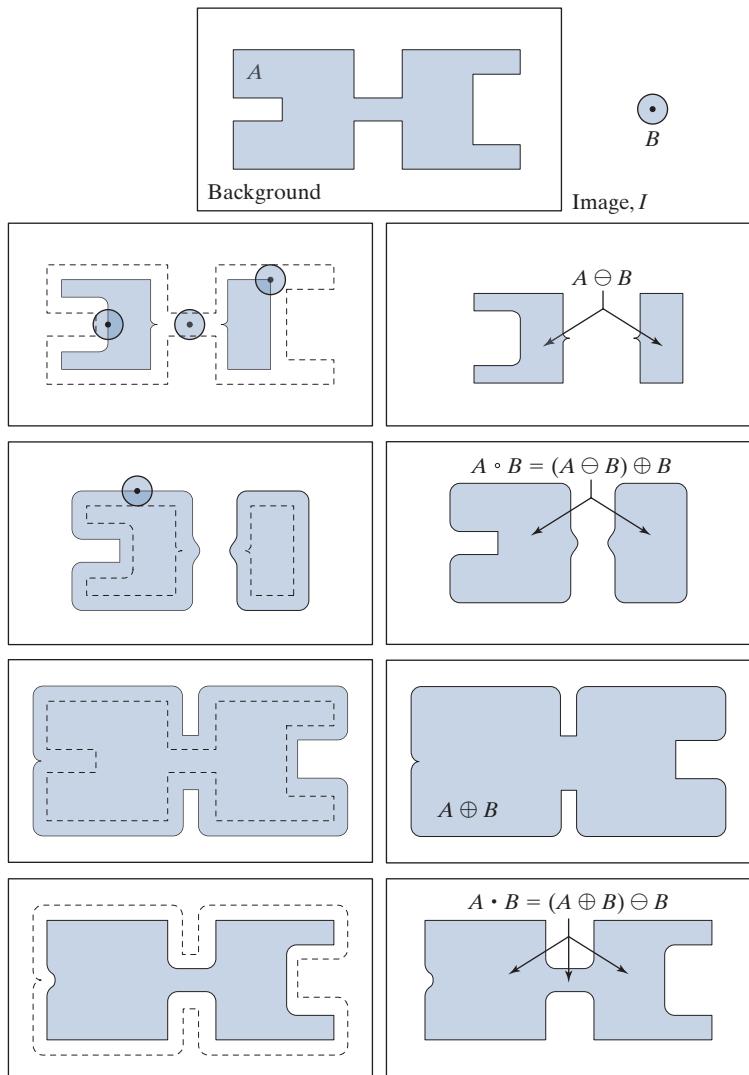
We leave the proof of these equations as an exercise (see Problem 9.20).

a
b
c
d
e
f
g
h
i

FIGURE 9.10

Morphological opening and closing.

- (a) Image I , composed of a set (object) A and background; a solid, circular structuring element is shown also. (The dot is the origin.)
- (b) Structuring element in various positions.
- (c)-(i) The morphological operations used to obtain the opening and closing.



Morphological opening has the following properties:

- (a) $A \circ B$ is a subset of A .
- (b) If C is a subset of D , then $C \circ B$ is a subset of $D \circ B$.
- (c) $(A \circ B) \circ B = A \circ B$.

Similarly, closing satisfies the following properties:

- (a) A is a subset of $A \bullet B$.
- (b) If C is a subset of D , then $C \bullet B$ is a subset of $D \bullet B$.
- (c) $(A \bullet B) \bullet B = A \bullet B$.

Note from condition (c) in both cases that multiple openings or closings of a set have no effect after the operation has been applied once.

EXAMPLE 9.4: Using opening and closing for morphological filtering.

Morphological operations can be used to construct filters similar in concept to the spatial filters discussed in Chapter 3. The binary image in Fig. 9.11(a) shows a section of a fingerprint corrupted by noise. In terms of our previous notation, A is the set of all foreground (white) pixels, which includes objects of interest (the fingerprint ridges) as well as white specks of random noise. The background is black, as before. The noise manifests itself as white specks on a dark background and dark specks on the white components of the fingerprint. The objective is to eliminate the noise and its effects on the print, while distorting it as little as possible. A morphological filter consisting of an opening followed by a closing can be used to accomplish this objective.

Figure 9.11(b) shows the structuring element we used. The rest of Fig. 9.11 shows the sequence of steps in the filtering operation. Figure 9.11(c) is the result of eroding A by B . The white speckled noise in the background was eliminated almost completely in the erosion stage of opening because in this case most noise components are smaller than the structuring element. The size of the noise elements (dark spots) contained within the fingerprint actually increased in size. The reason is that these elements are inner boundaries that increase in size as objects are eroded. This enlargement is countered by performing dilation on Fig. 9.11(c). Figure 9.11(d) shows the result.

The two operations just described constitute the opening of A by B . We note in Fig. 9.11(d) that the net effect of opening was to reduce all noise components in both the background and the fingerprint itself. However, new gaps between the fingerprint ridges were created. To counter this undesirable effect, we perform a dilation on the opening, as shown in Fig. 9.11(e). Most of the breaks were restored, but the ridges were thickened, a condition that can be remedied by erosion. The result, shown in Fig. 9.11(f), is the closing of the opening of Fig. 9.11(d). This final result is remarkably clean of noise specks, but it still shows some specks of noise that appear as single pixels. These could be eliminated by methods we will discuss later in this chapter.

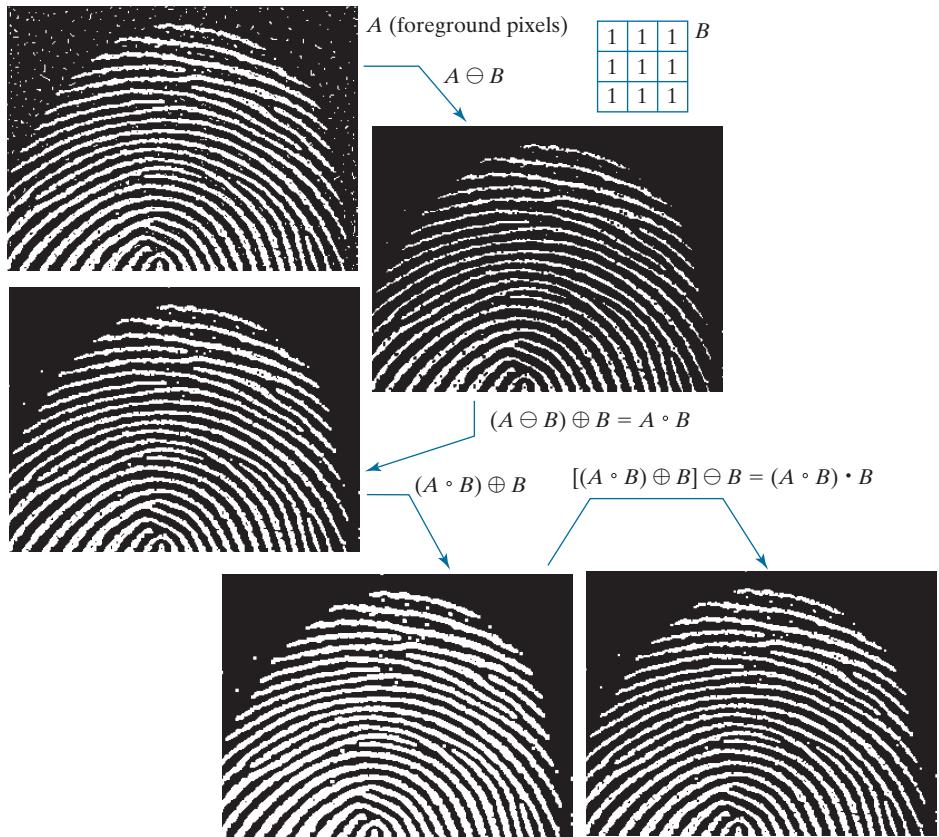
9.4 THE HIT-OR-MISS TRANSFORM

The morphological *hit-or-miss transform* (HMT) is a basic tool for shape detection. Let I be a binary image composed of foreground (A) and background pixels, respectively. Unlike the morphological methods discussed thus far, the HMT utilizes two

a	b
d	c
e	f

FIGURE 9.11

- (a) Noisy image.
 (b) Structuring element.
 (c) Eroded image.
 (d) Dilation of the erosion (opening of A).
 (e) Dilation of the opening.
 (f) Closing of the opening.
 (Original image courtesy of the National Institute of Standards and Technology.)



With reference to the explanation of Eq. (9-4), we show the morphological HMT operation working directly on image I , to make it explicit that the structuring elements work on sets of foreground and background pixels simultaneously.

structuring elements: B_1 , for detecting shapes in the foreground, and B_2 , for detecting shapes in the background. The HMT of image I is defined as

$$\begin{aligned} I \circledast B_{1,2} &= \left\{ z \mid (B_1)_z \subseteq A \text{ and } (B_2)_z \subseteq A^c \right\} \\ &= (A \ominus B_1) \cap (A^c \ominus B_2) \end{aligned} \quad (9-16)$$

where the second line follows from the definition of erosion in Eq. (9-3). In words, this equation says that the morphological HMT is the set of translations, z , of structuring elements B_1 and B_2 such that, *simultaneously*, B_1 found a match in the foreground (i.e., B_1 is contained in A) *and* B_2 found a match in the background (i.e., B_2 is contained in A^c). The word “simultaneous” implies that z is the *same* translation of both structuring elements. The word “miss” in the HMT arises from the fact that B_2 finding a match in A^c is the same as B_2 not finding (missing) a match in A .

Figure 9.12 illustrates the concepts just introduced. Suppose that we want to find the location of the origin of object (set) D in image I . Here, A is the union of all object sets, so D is a subset of A . The need for two structuring elements capable

a	b
c	d
e	f

FIGURE 9.12

(a) Image consisting of a foreground (1's) equal to the union, A , of set of objects, and a background of 0's.

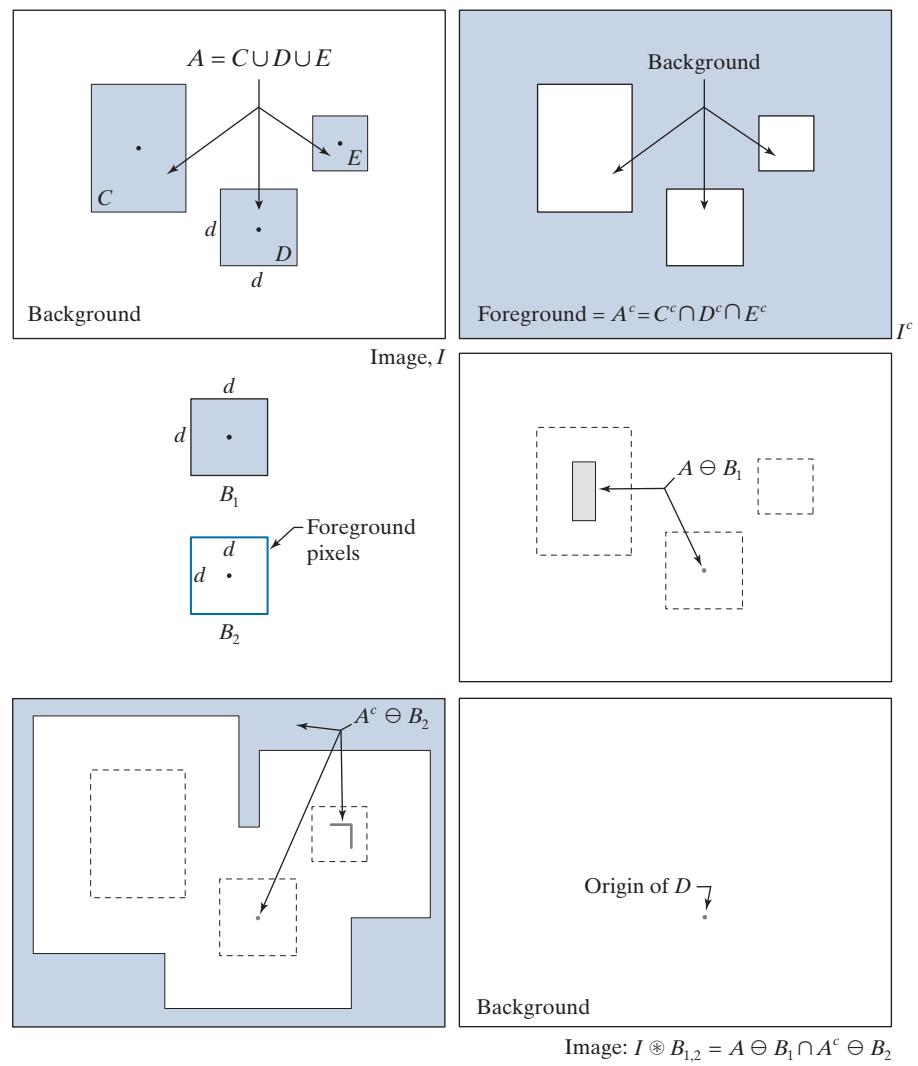
(b) Image with its foreground defined as A^c .

(c) Structuring elements designed to detect object D .

(d) Erosion of A by B_1 .

(e) Erosion of A^c by B_2 .

(f) Intersection of (d) and (e), showing the location of the origin of D , as desired. The dots indicate the origin of their respective components. Each dot is a single pixel.



of detecting properties of both the foreground and background becomes immediately obvious. All three objects are composed of foreground pixels, and one way of explaining why they appear as different shapes is because each occupies a different area of the background. In other words, the nature of a shape is determined by the geometrical arrangement of both foreground and background pixels.

Figure 9.12(a) shows that I is composed of foreground (A) and background pixels. Figure 9.12(b) is I^c , the complement of I . The foreground of I^c is defined as the set of pixels in A^c , and the background is the union of the complement of the three objects. Figure 9.12(c) shows the two structuring elements needed to detect D . Element B_1 is equal to D itself. As Fig. 9.12(d) shows, the erosion of A by B_1 contains a single point: the origin of D , as desired, but it also contains parts of object C .

Structuring element B_2 is designed to detect D in I^c . Because D is composed of background elements in I^c , and erosion works with foreground elements, B_2 has to be designed to detect the *border* of D , which is composed of foreground pixels in I^c . The SE in Fig. 9.12(c) does precisely this. It consists of a rectangle of foreground elements one pixel thick. The size of the rectangle is such that it encloses the size of D . Figure 9.12(e) shows (shaded) the erosion of the foreground of I^c by B_2 . It contains the origin of D , but it also contains parts of sets A^c and C . (The outer shaded area in Fig. 9.12(e) is larger than shown (see Problem 9.25); the result was cropped to the same size as image I for consistency.) The only elements that are common in Figs. 9.12(d) and (e) is the origin of D , so the intersection of these two sets of elements gives the location of that point, as desired. Figure 9.12(f) shows the final result.

The preceding explanation is the classic way of presenting the HMT using erosion, which is defined only for foreground pixels. A good question at this point is: Why not try to detect D directly in image I using a single structuring element, instead of going through such a laborious process? The answer is that it is possible to do so, but not in the “traditional” context of erosion the way we defined it in Eqs. (9-3) and (9-5). In order to detect D directly in image I , we would have to be able to process foreground and background pixels *simultaneously*, rather than processing just foreground pixels, as required by the definition of erosion.

To show how this can be done for the example in Fig. 9.12, we define a structuring element, B , identical to D , but having in addition a border of *background* elements with a width of one pixel. We can use a structuring element formed in such a way to restate the HMT as

$$I \circledast B = \left\{ z \mid (B)_z \subseteq I \right\} \quad (9-17)$$

The *form* is the same as Eq. (9-3), but now we test to see if $(B)_z$ is a subset of *image* I , which is composed of *both* foreground and background pixels. This formulation is general, in the sense that B can be structured to detect any arrangement of pixels in image I , as Figs. 9.13 and 9.14 will illustrate.

Figure 9.13 shows graphically the same solution as Fig. 9.12(f), but using the single structuring element discussed in the previous paragraph. Figure 9.14 shows several examples based on using Eq. (9-17). The first row shows the result of using a small SE composed of both foreground (shaded) and background elements. This SE is designed to detect one-pixel holes (i.e., one background pixel surrounded by a connected border of foreground pixels) contained in image I . The SE in the second row is capable of detecting the foreground corner pixel of the top, right corner of the object in I . Using this SE in Eq. (9-17) yielded the image on the right. As you can see, the correct pixel was identified. The last row of Fig. 9.14 is more interesting, as it shows a structuring element composed of foreground, background, and “don’t care” elements which, as mentioned earlier, we denote by ‘x’s. You can think of the value of a don’t care element as always matching its corresponding pixel in an image. In this example, when the SE is centered on the top, right corner pixel, the don’t care elements in the top of the SE can be considered to be background, and the don’t care elements on the bottom row as foreground, producing a correct

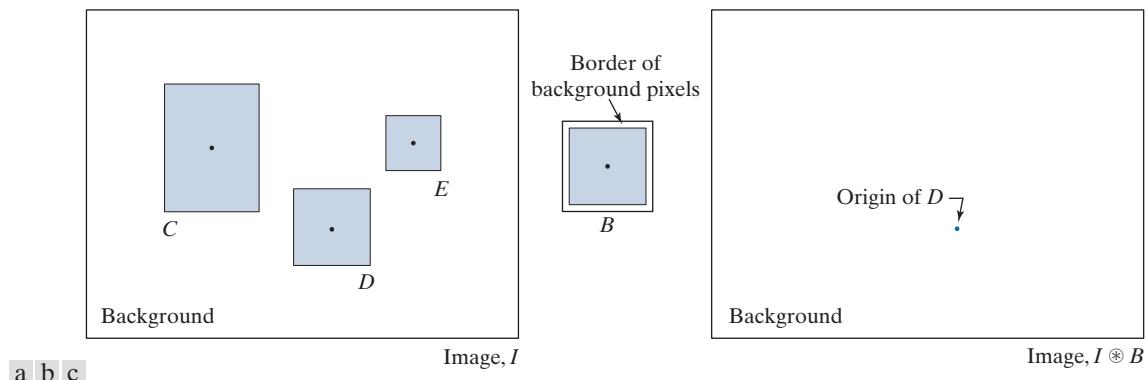


FIGURE 9.13 Same solution as in Fig. 9.12, but using Eq. (9-17) with a single structuring element.

match. When the SE is centered on the bottom, right corner pixel, the role of the don't care elements is reversed, again resulting in a correct match. The other border pixels between the two corners were similarly detected by considering all don't care elements as foreground. Thus, using don't care elements increases the flexibility of structuring elements to perform multiple roles.

9.5 SOME BASIC MORPHOLOGICAL ALGORITHMS

With the preceding discussion as a foundation, we are now ready to consider some practical uses of morphology. When dealing with binary images, one of the principal applications of morphology is in extracting image components that are useful in the

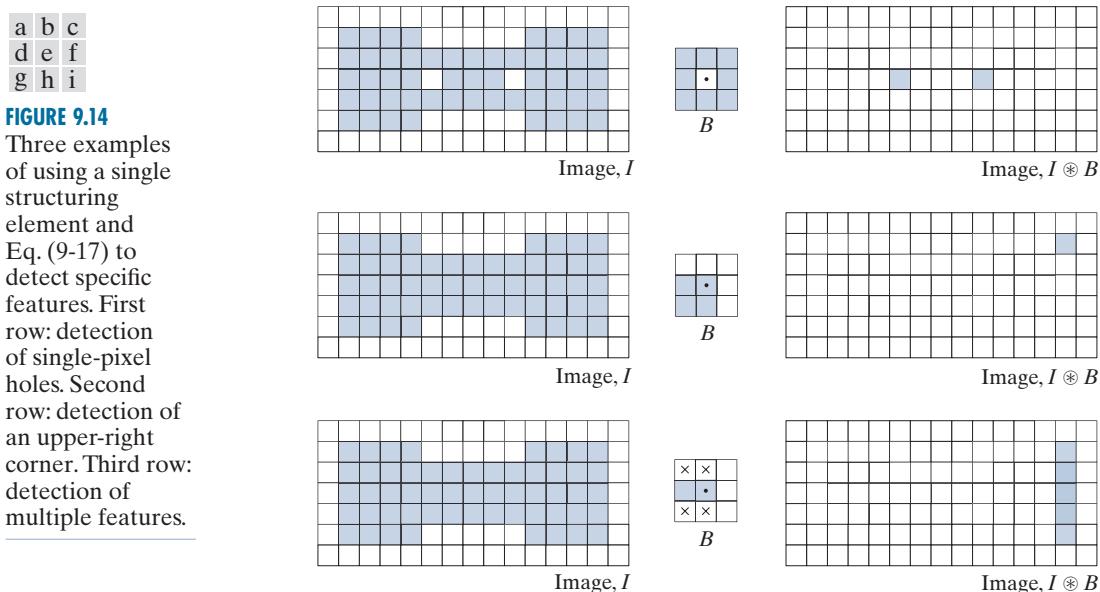


FIGURE 9.14

Three examples of using a single structuring element and Eq. (9-17) to detect specific features. First row: detection of single-pixel holes. Second row: detection of an upper-right corner. Third row: detection of multiple features.

representation and description of shape. In particular, we consider morphological algorithms for extracting boundaries, connected components, the convex hull, and the skeleton of a region. We also develop several methods (for region filling, thinning, thickening, and pruning) that are used frequently for pre- or post-processing. We make extensive use in this section of “mini-images,” designed to clarify the mechanics of each morphological method as we introduce it. These binary images are shown graphically with foreground (1’s) shaded and background (0’s) in white, as before.

BOUNDARY EXTRACTION

The boundary of a set A of foreground pixels, denoted by $\beta(A)$, can be obtained by first eroding A by a suitable structuring element B , and then performing the set difference between A and its erosion. That is,

$$\beta(A) = A - (A \ominus B) \quad (9-18)$$

Figure 9.15 illustrates the mechanics of boundary extraction. It shows a simple binary object, a structuring element B , and the result of using Eq. (9-18). The structuring element in Fig. 9.15(b) is among the most frequently used, but it is not unique. For example, using a 5×5 structuring element of 1’s would result in a boundary between 2 and 3 pixels thick. It is understood that the image in Fig. 9.15(a) was padded with a border of background elements, and that the results were cropped back to the original size after the morphological operations were completed.

EXAMPLE 9.5: Boundary extraction.

Figure 9.16 further illustrates the use of Eq. (9-18) using a 3×3 structuring element of 1’s. As before when working with images, we show foreground pixels (1’s) in white and background pixels (0’s) in black. The elements of the SE, which are 1’s, also are treated as white. Because of the size of the structuring element used, the boundary in Fig. 9.16(b) is one pixel thick.

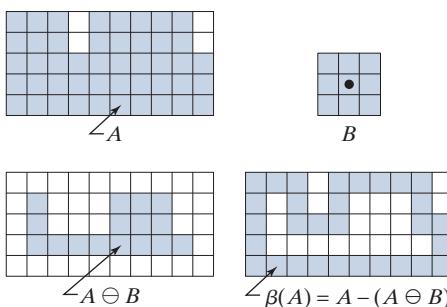
HOLE FILLING

As mentioned in the discussion of Fig. 9.14, a *hole* may be defined as a background region surrounded by a connected border of foreground pixels. In this section, we develop an algorithm based on set dilation, complementation, and intersection for

a	b
c	d

FIGURE 9.15

- (a) Set, A , of foreground pixels.
- (b) Structuring element.
- (c) A eroded by B .
- (d) Boundary of A .



a b

FIGURE 9.16

- (a) A binary image.
 (b) Result of using Eq. (9-18) with the structuring element in Fig. 9.15(b).



Remember, the dilation of image X by B is the dilation of the foreground elements of X by B .

filling holes in an image. Let A denote a set whose elements are 8-connected boundaries, with each boundary enclosing a background region (i.e., a hole). Given a point in each hole, the objective is to fill all the holes with foreground elements (1's).

We begin by forming an array, X_0 , of 0's (the same size as I , the image containing A), except at locations in X_0 that correspond to pixels that are known to be holes, which we set to 1. Then, the following procedure fills all the holes with 1's:

$$X_k = (X_{k-1} \oplus B) \cap I^c \quad k = 1, 2, 3, \dots \quad (9-19)$$

where B is the symmetric structuring element in Fig. 9.17(c). The algorithm terminates at iteration step k if $X_k = X_{k-1}$. Then, X_k contains all the filled holes. The set union of X_k and I contains all the filled holes and their boundaries.

The dilation in Eq. (9-19) would fill the entire area if left unchecked, but the intersection at each step with I^c limits the result to inside the region of interest. This is our first example of how a morphological process can be *conditioned* to meet a desired property. In the current application, the process is appropriately called *conditional dilation*. The rest of Fig. 9.17 illustrates further the mechanics of Eq. (9-19). This example only has one hole, but the concept applies to any finite number of holes, assuming that a point inside each hole is given (we remove this requirement in Section 9.6).

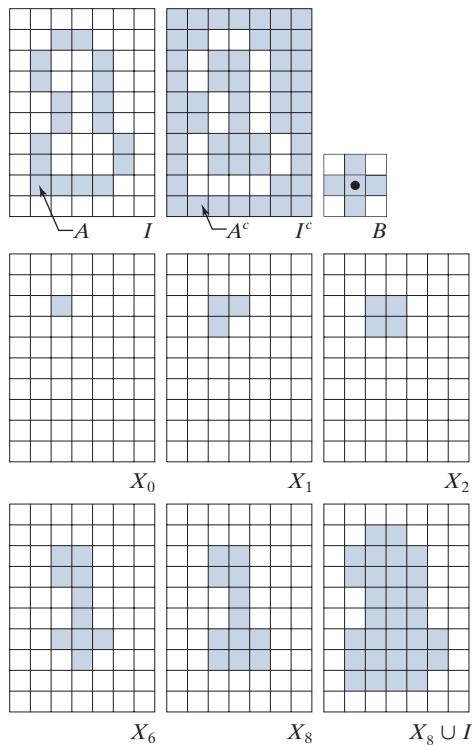
EXAMPLE 9.6: Morphological hole filling.

Figure 9.18(a) shows an image of white circles with black holes. An image such as this might result from thresholding into two levels a scene containing polished spheres (e.g., ball bearings). The dark circular areas inside the spheres would result from reflections. The objective is to eliminate the reflections by filling the holes in the image. Figure 9.18(b) shows the result of filling all the spheres. Because it must be known whether black points are background points or sphere inner points (i.e., holes), fully automating this procedure requires that additional “intelligence” be built into the algorithm. We will give a fully automatic approach in Section 9.6 based on morphological reconstruction (see Problem 9.36 also).

a	b	c
d	e	f
g	h	i

FIGURE 9.17

- Hole filling.
- Set A (shown shaded) contained in image I .
 - Complement of I .
 - Structuring element B . Only the foreground elements are used in computations
 - Initial point inside hole, set to 1.
 - Various steps of Eq. (9-19).
 - Final result [union of (a) and (h)].



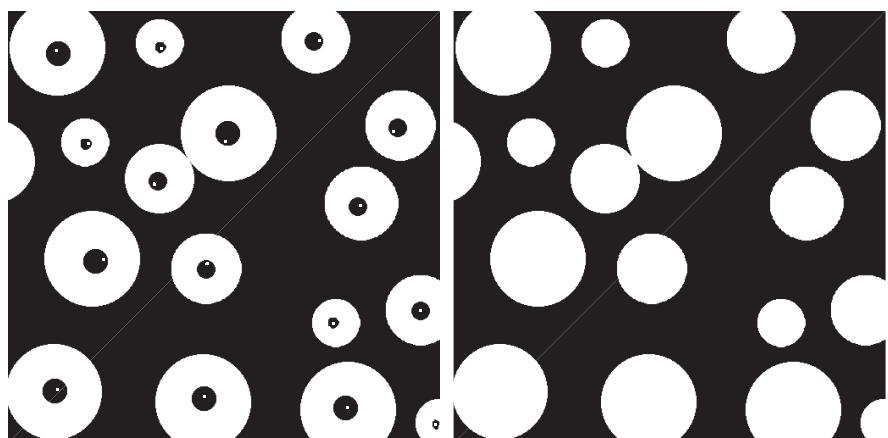
EXTRACTION OF CONNECTED COMPONENTS

Connectivity and connected components are discussed in Section 2.5.

a	b
---	---

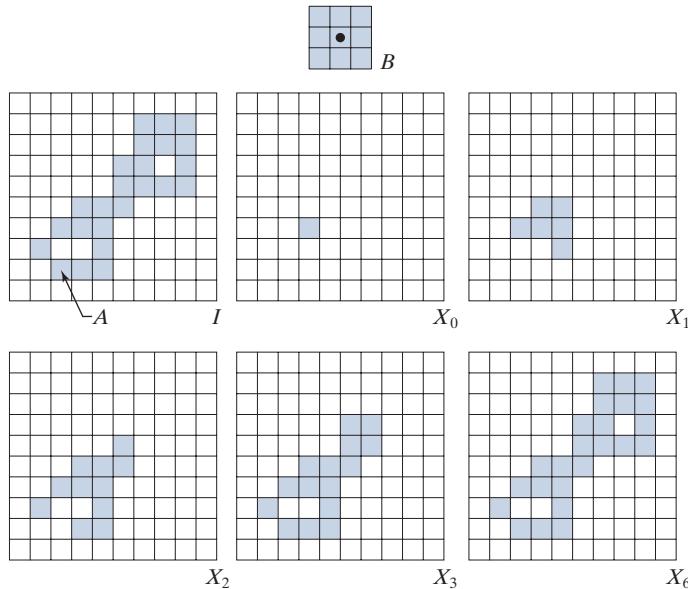
FIGURE 9.18

- (a) Binary image. The white dots inside the regions (shown enlarged for clarity) are the starting points for the hole-filling algorithm.
- (b) Result of filling all holes.



**FIGURE 9.19**

- (a) Structuring element.
- (b) Image containing a set with one connected component.
- (c) Initial array containing a 1 in the region of the connected component.
- (d)–(g) Various steps in the iteration of Eq. (9-20)



which we set to 1 (foreground value). The objective is to start with X_0 and find all the connected components in I . The following iterative procedure accomplishes this:

$$X_k = (X_{k-1} \oplus B) \cap I \quad k = 1, 2, 3, \dots \quad (9-20)$$

where B is the SE in Fig. 9.19(a). The procedure terminates when $X_k = X_{k-1}$, with X_k containing all the connected components of foreground pixels in the image. Both Eqs. (9-19) and (9-20) use conditional dilation to limit the growth of set dilation, but Eq. (9-20) uses I instead of I^c . This is because here we are looking for foreground points, while the objective of (9-19) is to find background points. Figure 9.19 illustrates the mechanics of Eq. (9-20), with convergence being achieved for $k = 6$. Note that the shape of the structuring element used is based on 8-connectivity between pixels. As in the hole-filling algorithm, Eq. (9-20) is applicable to any finite number of connected components contained in I , assuming that a point is known in each. See Problem 9.37 for a completely automated procedure that removes this requirement.

EXAMPLE 9.7: Using connected components to detect foreign objects in packaged food.

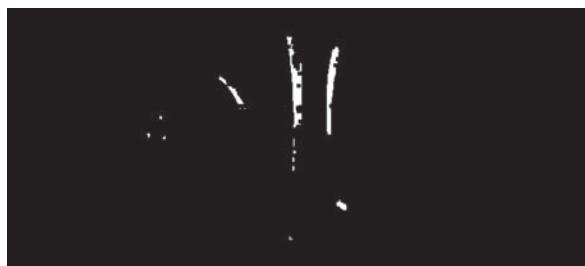
Connected components are used frequently for automated inspection. Figure 9.20(a) shows an X-ray image of a chicken breast that contains bone fragments. It is important to be able to detect such foreign objects in processed foods before shipping. In this application, the density of the bones is such that their nominal intensity values are significantly different from the background. This makes extraction of the bones from the background a simple matter by using a single threshold (thresholding was introduced in Section 3.1 and we will discuss in more detail in Section 10.3). The result is the binary image in Fig. 9.20(b).

The most significant feature in this figure is the fact that the points that remain after thresholding are clustered into objects (bones), rather than being scattered. We can make sure that only objects of

a
b
c d

FIGURE 9.20

- (a) X-ray image of a chicken fillet with bone fragments.
- (b) Thresholded image (shown as the negative for clarity).
- (c) Image eroded with a 5×5 SE of 1's.
- (d) Number of pixels in the connected components of (c). (Image (a) courtesy of NTB Elektronische Geraete GmbH, Diepholz, Germany, [www.ntbxray.com.](http://www.ntbxray.com/))



Connected component	No. of pixels in connected comp
01	11
02	9
03	9
04	39
05	133
06	1
07	1
08	743
09	7
10	11
11	11
12	9
13	9
14	674
15	85

“significant” size are contained in the binary image by eroding its foreground. In this example, we define as significant any object that remains after erosion with a 5×5 SE of 1's. Figure 9.20(c) shows the result of erosion. The next step is to analyze the size of the objects that remain. We label (identify) these objects by extracting the connected components in the image. The table in Fig. 9.20(d) lists the results of the extraction. There are 15 connected components, with four of them being dominant in size. This is enough evidence to conclude that significant, undesirable objects are contained in the original image. If needed, further characterization (such as shape) is possible using the techniques discussed in Chapter 11.

CONVEX HULL

A set, S , of points in the Euclidean plane is said to be *convex* if and only if a straight line segment joining any two points in S lies entirely within S . The *convex hull*, H , of S is the smallest convex set containing S . The *convex deficiency* of S is defined as the set difference $H - S$. Unlike the Euclidean plane, the digital image plane (see Fig. 2.19) only allows points at discrete coordinates. Thus, the sets with which we work are *digital sets*. The same concepts of convexity are applicable to digital sets, but the definition of a convex digital set is slightly different. A *digital set*, A , is said to be *convex* if and only if its Euclidean convex hull only contains digital points

belonging to A . A simple way to visualize if a digital set of foreground points is convex is to join its boundary points by straight (continuous) Euclidean line segments. If only foreground points are contained within the set formed by the line segments, then the set is convex; otherwise it is not. The definitions of convex hull and convex deficiency given above for S , extend directly to digital sets. The following morphological algorithm can be used to obtain an approximation of the convex hull of a set A of foreground pixels, embedded in a binary image, I .

Let B^i , $i = 1, 2, 3, 4$, denote the four structuring elements in Fig. 9.21(a). The procedure consists of implementing the morphological equation

$$X_k^i = (X_{k-1}^i \circledast B^i) \cup X_{k-1}^i \quad i = 1, 2, 3, 4 \quad \text{and} \quad k = 1, 2, 3, \dots \quad (9-21)$$

with $X_0^i = I$. When the procedure converges using the i th structuring element (i.e., when $X_k^i = X_{k-1}^i$), we let $D^i = X_k^i$. Then, the convex hull of A is the union of the four results:

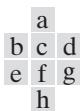
$$C(A) = \bigcup_{i=1}^4 D^i \quad (9-22)$$

Thus, the method consists of iteratively applying the hit-or-miss transform to I with B^1 until convergence, then letting $D^1 = X_k^1$, where k is the step at which convergence occurred. The procedure is repeated with B^2 (applied to I) until no further changes occur, and so on. The union of the four resulting D^i constitutes the convex hull of A . The algorithm is initialized with $k = 0$ and $X_0^i = I$ every time that i (i.e., the structuring element) changes.

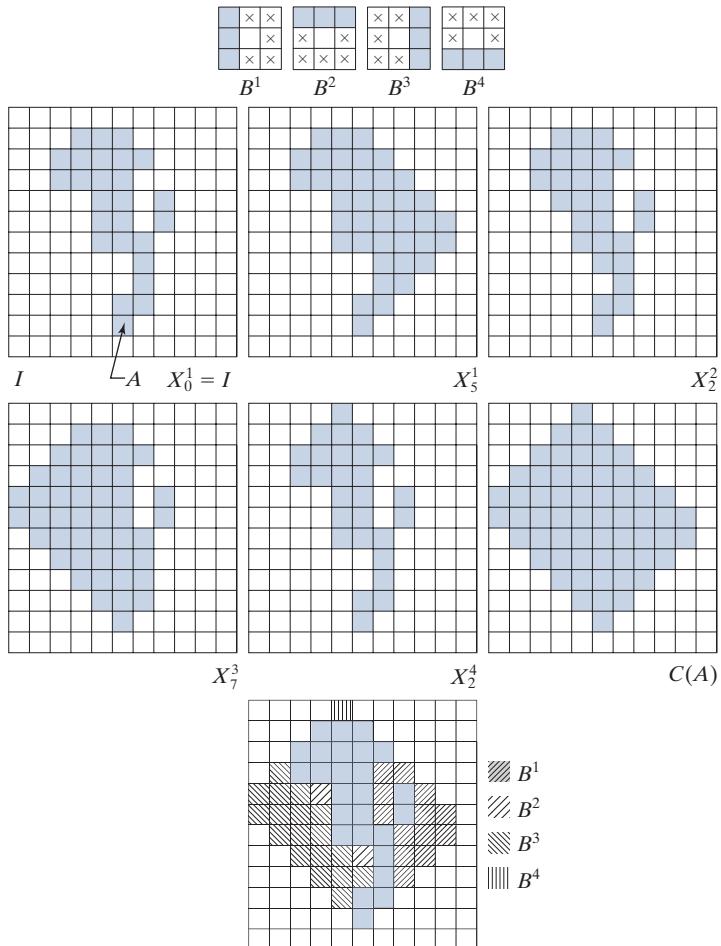
Figure 9.21 illustrates the use of Eqs. (9-21) and (9-22). Figure 9.21(a) shows the structuring elements used to extract the convex hull. The origin of each element is at its center. As before, the \times entries indicate “don’t care” elements. Recall that the HMT is said to have found a match of structuring element B^i in a 3×3 region of I , if all the elements of B^i find corresponding matches in that region. As noted earlier, when computing a match, a “don’t care” element can be interpreted as always matching the value of its corresponding element in the image. Note in Fig. 9.21(a) that B^i is a clockwise rotation of B^{i-1} by 90° .

Figure 9.21(b) shows a set A for which the convex hull is sought. As before, the set is embedded in an array of background elements to form an image, I . Starting with $X_0^1 = I$ resulted in the set in Fig. 9.21(c) after five iterations of Eq. (9-21). Then, letting $X_0^2 = I$ and again using Eq. (9-21) resulted in the set in Fig. 9.21(d) (convergence was achieved in only two steps in this case). The next two results were obtained in the same manner. Finally, forming the union of the sets in Figs. 9.21(c), (d), (e), and (f) resulted in the convex hull in Fig. 9.21(g). The contribution of each structuring element is highlighted in the composite set shown in Fig. 9.21(h).

One obvious shortcoming of the procedure just discussed is that the convex hull can grow beyond the minimum dimensions required to guarantee convexity, thus violating the definition of the convex hull. This, in fact, is what happened in this case. One simple approach to reduce this growth is to place limits so that it does not extend beyond the vertical and horizontal dimensions of set A . Imposing this

**FIGURE 9.21**

- (a) Structuring elements.
- (b) Set A .
- (c)–(f) Results of convergence with the structuring elements shown in (a).
- (g) Convex hull.
- (h) Convex hull showing the contribution of each structuring element.



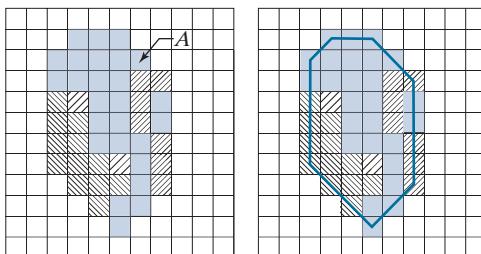
limitation on the example in Fig. 9.21 resulted in Fig. 9.22(a). Joining the boundary pixels of the reduced set (remember, the pixels are the center points of the squares) show that no set points lie outside these lines, indicating that the set is convex. By inspection, you can see that no points can be deleted from this set without losing convexity, so the reduced set is the convex hull of A .

Of course, the limits we used to produce Fig. 9.22 do not constitute a general approach for obtaining the minimum convex set enclosing a set in question; it is simply an easy-to-implement heuristic. The reason why the convex hull algorithm did not yield a closer approximation of the actual convex hull is because of the structuring elements used. The SEs in Fig. 9.21(a) “look” only in four orthogonal directions. We could achieve greater accuracy by looking in additional directions, such as the diagonals, for example. The price paid is increased algorithm complexity and a higher computational load.

a b

FIGURE 9.22

- (a) Result of limiting growth of the convex hull algorithm.
 (b) Straight lines connecting the boundary points show that the new set is convex also.



THINNING

Thinning of a set A of foreground pixels by a structuring element B , denoted $A \otimes B$, can be defined in terms of the hit-or-miss transform:

$$\begin{aligned} A \otimes B &= A - (A \circledast B) \\ &= A \cap (A \circledast B)^c \end{aligned} \quad (9-23)$$

where the second line follows from the definition of set difference given in Eq. (2-40). A more useful expression for thinning A symmetrically is based on a *sequence* of structuring elements:

$$\{B\} = \{B^1, B^2, B^3, \dots, B^n\} \quad (9-24)$$

Using this concept, we now define thinning by a sequence of structuring elements as

$$A \otimes \{B\} = \left(\left(\dots \left((A \otimes B^1) \otimes B^2 \right) \dots \right) \otimes B^n \right) \quad (9-25)$$

The process is to thin A by one pass with B^1 , then thin the result with one pass of B^2 , and so on, until A is thinned with one pass of B^n . The entire process is repeated until no further changes occur after one complete pass through all structuring elements. Each individual thinning pass is performed using Eq. (9-23).

As before, we assume that the image containing A was padded to accommodate all excursions of B , and that the result was cropped. We show only A for simplicity.

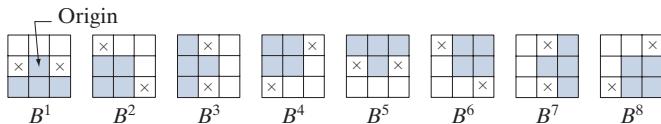
Figure 9.23(a) shows a set of structuring elements used routinely for thinning (note that B^i is equal to B^{i-1} rotated clockwise by 45°), and Fig. 9.23(b) shows a set A to be thinned, using the procedure just discussed. Figure 9.23(c) shows the result of thinning A with one pass of B^1 to obtain A_1 . Figure 9.23(c) is the result of thinning A_1 with B^2 , and Figs. 9.21(e) through (k) show the results of passes with the remaining structuring elements (there were no changes from A_7 to A_8 or from A_9 to A_{11} .) Convergence was achieved after the second pass of B^6 . Figure 9.23(l) shows the thinned result. Finally, Fig. 9.23(m) shows the thinned set converted to m -connectivity (see Section 2.5 and Problem 9.29) to eliminate multiple paths.

THICKENING

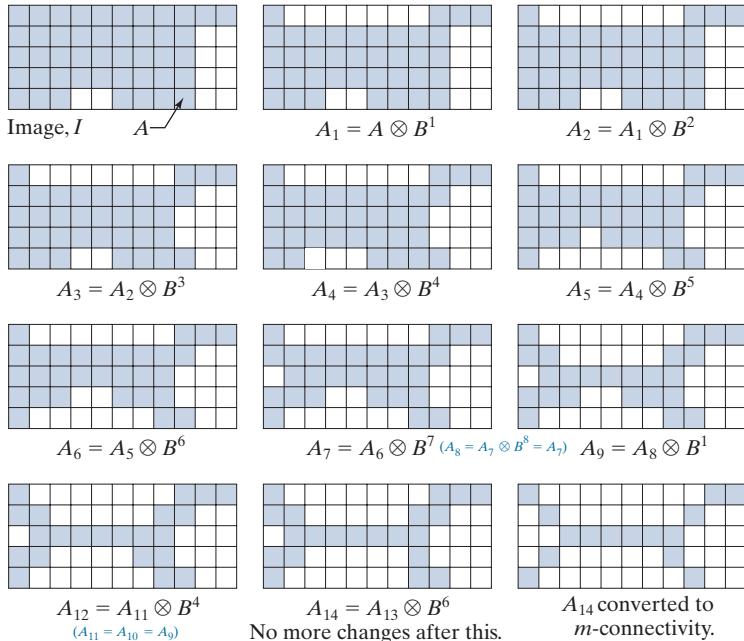
Thickening is the morphological dual of thinning and is defined by the expression

$$A \odot B = A \cup (A \circledast B) \quad (9-26)$$

a
b
c
d
e
f
g
h
i
j
k
l
m

**FIGURE 9.23**

- (a) Structuring elements.
- (b) Set A .
- (c) Result of thinning A with B^1 (shaded).
- (d) Result of thinning A_1 with B_2 .
- (e)–(i) Results of thinning with the next six SEs. (There was no change between A_7 and A_8 .)
- (j)–(k) Result of using the first four elements again.
- (l) Result after convergence.
- (m) Result converted to m -connectivity.



where B is a structuring element suitable for thickening. As in thinning, thickening can be defined as a sequential operation:

$$A \odot \{B\} = \left(\left(\dots \left((A \odot B^1) \odot B^2 \right) \dots \right) \odot B^n \right) \quad (9-27)$$

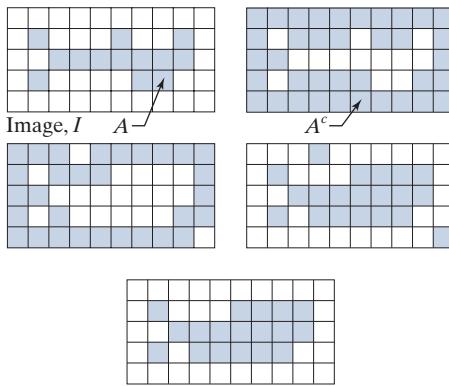
The structuring elements used for thickening have the same form as those shown in Fig. 9.23(a), but with all 1's and 0's interchanged. However, a separate algorithm for thickening is seldom used in practice. Instead, the usual procedure is to thin the background of the set in question, then complement the result. In other words, to thicken a set A we form A^c , thin A^c , and then complement the thinned set to obtain the thickening of A . Figure 9.24 illustrates this procedure. As before, we show only set A and image I , and not the padded version of I .

Depending on the structure of A , this procedure can result in disconnected points, as Fig. 9.24(d) shows. Hence thickening by this method usually is followed by post-processing to remove disconnected points. Note from Fig. 9.24(c) that the thinned background forms a boundary for the thickening process. This useful feature is not present in the direct implementation of thickening using Eq. (9-27), and it is one of the principal reasons for using background thinning to accomplish thickening.

a	b
c	d
e	

FIGURE 9.24

- (a) Set A .
- (b) Complement of A .
- (c) Result of thinning the complement.
- (d) Thickened set obtained by complementing (c).
- (e) Final result, with no disconnected points.



We will discuss skeletons in more detail in Section 11.2.

SKELETONS

As Fig. 9.25 shows, the notion of a *skeleton* $S(A)$ of a set A is intuitively simple. We deduce from this figure that

- (a) If z is a point of $S(A)$, and $(D)_z$ is the largest disk centered at z and contained in A , one cannot find a larger disk (not necessarily centered at z) containing $(D)_z$ and simultaneously included in A . A disk $(D)_z$ satisfying these conditions is called a *maximum disk*.
- (b) If $(D)_z$ is a maximum disk, it touches the boundary of A at two or more different places.

The skeleton of A can be expressed in terms of erosions and openings. That is, it can be shown (Serra [1982]) that

$$S(A) = \bigcup_{k=0}^K S_k(A) \quad (9-28)$$

with

$$S_k(A) = (A \ominus kB) - (A \ominus kB) \circ B \quad (9-29)$$

where B is a structuring element, and $(A \ominus kB)$ indicates k successive erosions starting with A ; that is, A is first eroded by B , the result is eroded by B , and so on:

$$(A \ominus kB) = ((\dots((A \ominus B) \ominus B) \ominus \dots) \ominus B) \quad (9-30)$$

k times. K in Eq. (9-28) is the last iterative step before A erodes to an empty set. In other words,

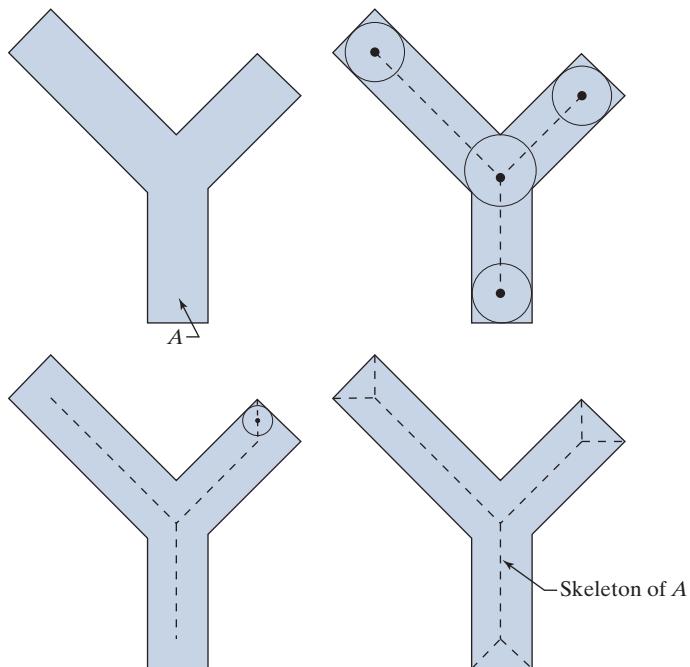
$$K = \max \{ k \mid (A \ominus kB) \neq \emptyset \} \quad (9-31)$$

The formulation in Eqs. (9-28) and (9-29) indicate that $S(A)$ can be obtained as the union of the skeleton subsets $S_k(A)$, $k = 0, 1, 2, \dots, K$.

a	b
c	d

FIGURE 9.25

- (a) Set A .
- (b) Various positions of maximum disks whose centers partially define the skeleton of A .
- (c) Another maximum disk, whose center defines a different segment of the skeleton of A .
- (d) Complete skeleton (dashed).



It can be shown (Serra [1982]) that A can be reconstructed from these subsets:

$$A = \bigcup_{k=0}^K (S_k(A) \oplus kB) \quad (9-32)$$

where $(S_k(A) \oplus kB)$ denotes k successive dilations, starting with $S_k(A)$; that is,

$$(S_k(A) \oplus kB) = ((\dots((S_k(A) \oplus B) \oplus B) \oplus \dots) \oplus B) \quad (9-33)$$

EXAMPLE 9.8: Computing the skeleton of a simple set.

Figure 9.26 illustrates the concepts just discussed. The first column shows the original set (at the top) and two erosions by the structuring element B shown in the figure. Note that one more erosion would yield the empty set, so $K = 2$ in this case. The second column shows the opening by B of the sets in the first column. These results are easily explained by the fitting characterization of the opening operation discussed in connection with Fig. 9.8. The third column contains the set differences between the first and second columns. Thus, the three entries in the third column are $S_0(A)$, $S_1(A)$, and $S_2(A)$, respectively.

The fourth column contains two partial skeletons, and the final result at the bottom of the column. The final skeleton not only is thicker than it needs to be but, more important, it is not connected. This result is not unexpected, as nothing in the preceding formulation of the morphological skeleton guarantees connectivity. Morphology produces an elegant formulation in terms of erosions and openings of the given set. However, heuristic formulations (see Section 11.2) are needed if, as is usually the case, the skeleton must be maximally thin, connected, and minimally eroded.

FIGURE 9.26

Implementation of Eqs. (9-28) through (9-33).

The original set is at the top left, and its morphological skeleton is at the bottom of the fourth column. The reconstructed set is at the bottom of the sixth column.

$k \backslash$	$A \ominus kB$	$(A \ominus kB) \circ B$	$S_k(A)$	$\bigcup_{k=0}^K S_k(A)$	$S_k(A) \oplus kB$	$\bigcup_{k=0}^K S_k(A) \oplus kB$
0						
1						
2				 $S(A)$		 A

B

The entries in the fifth and sixth columns deal with reconstructing the original set from its skeleton subsets. The fifth column are the dilations of $S_k(A)$; that is, $S_0(A)$, $S_1(k) \oplus B$, and $S_2(A) \oplus 2B = (S_2(A) \oplus B) \oplus B$. Finally, the last column shows reconstruction of set A which, according to Eq. (9-32), is the union of the dilated skeleton subsets shown in the fifth column.

PRUNING

Pruning methods are an essential complement to thinning and skeletonizing algorithms, because these procedures tend to leave *spurs* (“parasitic” components) that need to be “cleaned up” by postprocessing. We begin the discussion with a pruning problem, then develop a solution based on the material introduced in the preceding sections. Thus, we take this opportunity to illustrate how to solve a problem by combining several of the morphological techniques discussed up to this point.

A common approach in the automated recognition of hand-printed characters is to analyze the shape of the skeleton of a character. These skeletons often contain spurs, caused during erosion by noise and non-uniformities in the character strokes.

In this section we develop a morphological technique for handling this problem, starting with the assumption that the length of a parasitic component does not exceed a specified number of pixels.

We may define an *end point* as the center point of a 3×3 region that satisfies any of the arrangements in Fig. 9.27(b).

Figure 9.27(a) shows the skeleton of a hand-printed letter “a.” The spur on the leftmost part of the character exemplifies what we are interested in removing. The solution is based on suppressing a spur branch by successively eliminating its end point. Of course, this also shortens (or eliminates) other branches in the character but, in the absence of other structural information, the assumption in this example is that any branch with three or less pixels is to be eliminated. Thinning of a set A , with a sequence of structuring elements designed to detect only end points, achieves the desired result. That is, let

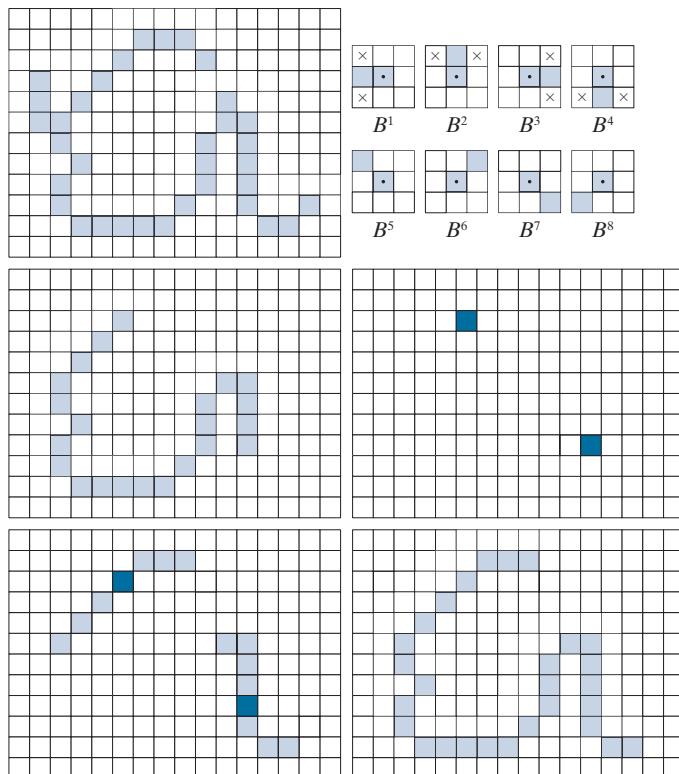
$$X_1 = A \otimes \{B\} \quad (9-34)$$

where $\{B\}$ denotes the structuring element sequence in Fig. 9.27(b) [see Eq. (9-24) regarding structuring-element sequences]. The sequence of structuring elements consists of two different structures, each of which is rotated 90° for a total of eight elements. The \times in Fig. 9.27(b) signifies a “don’t care” condition, as defined earlier. (Note that each SE is a detector for an end point in a particular orientation.)

a b
c d
e f

FIGURE 9.27

- (a) Set A of foreground pixels (shaded).
- (b) SEs used for deleting end points.
- (c) Result of three cycles of thinning.
- (d) End points of (c).
- (e) Dilation of end points conditioned on (a).
- (f) Pruned image.



Applying Eq. (9-34) to A three times yielded the set X_1 in Fig. 9.27(c). The next step is to “restore” the character to its original form, but with the parasitic branches removed. This requires that we first form a set X_2 containing all end points in X_1 [Fig. 9.27(e)]:

$$X_2 = \bigcup_{k=1}^8 (X_1 \circledast B^k) \quad (9-35)$$

where the B^k are the end-point detectors in Fig. 9.27(b). The next step is dilation of the end points. Typically, the number of dilations is less than the number of end-point removals to reduce the probability of “growing” back some of the spurs. In this case, we know by inspection that no new spurs are created, so we dilate the end points three times using A as a delimiter. This is the same number of thinning passes:

$$X_3 = (X_2 \oplus H) \cap A \quad (9-36)$$

where H is a 3×3 structuring element of 1’s, and the intersection with A is applied after each step. As in the case of region filling, this type of conditional dilation prevents the creation of 1-valued elements outside the region of interest, as illustrated by the result in Fig. 9.27(e). Finally, the union of X_1 and X_3 ,

$$X_4 = X_1 \cup X_3 \quad (9-37)$$

yields the desired result in Fig. 9.27(f).

In more complex scenarios, using Eq. (9-36) sometimes picks up the “tips” of some branches. This can occur when the end points of these branches are near the skeleton. Although Eq. (9-36) may eliminate them, they can be picked up again during dilation because they are valid points in A . However, unless entire parasitic elements are picked up again (a rare case if these elements are short with respect to valid strokes), detecting and eliminating the reconstructed elements is easy because they are disconnected regions.

A natural thought at this juncture is that there must be easier ways to solve this problem. For example, we could just keep track of all deleted points and simply reconnect the appropriate points to all end points left after application of Eq. (9-34). This argument is valid, but the advantage of the formulation just presented is that we used existing morphological constructs to solve the problem. When a set of such tools is available, the advantage is that no new algorithms have to be written. We simply combine the necessary morphological functions into a sequence of operations.

Sometimes you will encounter end point detectors based on a single structuring element, similar to the first SE in Fig. 9.27(b), but having “don’t care” conditions along the entire first column instead having a foreground element separating the corner \times ’s. This is incorrect. For example, the former element would identify the point located in the eighth row, fourth column of Fig. 9.27(a) as an end point, thus eliminating it and breaking the connectivity of that part of the stroke.

9.6 MORPHOLOGICAL RECONSTRUCTION

The morphological concepts discussed thus far involve a single image and one or more structuring elements. In this section, we discuss a powerful morphological transformation called *morphological reconstruction* that involves two images and a structuring element. One image, the *marker*, which we denote by F , contains the starting points for reconstruction. The other image, the *mask*, denoted by G , constrains (conditions) the reconstruction. The structuring element is used to define connectivity.[†] For 2-D applications, connectivity typically is defined as 8-connectivity, which is implied by a structuring element of size 3×3 whose elements are all 1's.

See Section 2.5 regarding connectivity.

GEODESIC DILATION AND EROSION

Central to morphological reconstruction are the concepts of geodesic dilation and geodesic erosion. Let F denote the marker image and G the mask image. We assume in this discussion that both are binary images and that $F \subseteq G$. The *geodesic dilation of size 1* of the marker image with respect to the mask, denoted by $D_G^{(1)}(F)$, is defined as

$$D_G^{(1)}(F) = (F \oplus B) \cap G \quad (9-38)$$

where, as usual, \cap denotes the set intersection (here \cap may be interpreted as a logical AND because we are dealing with binary quantities). The *geodesic dilation of size n* of F with respect to G is defined as

$$D_G^{(n)}(F) = D_G^{(1)}\left(D_G^{(n-1)}(F)\right) \quad (9-39)$$

where $n \geq 1$ is an integer, and $D_G^{(0)}(F) = F$. In this recursive expression, the set intersection indicated in Eq. (9-38) is performed at each step.[‡] Note that the intersection operation guarantees that mask G will limit the growth (dilation) of marker F . Figure 9.28 shows a simple example of a geodesic dilation of size 1. The steps in the figure are a direct implementation of Eq. (9-38). Note that the marker F consists of just one point from the object in G . The idea is to grow (dilate) this point successively, masking of the result at each step by G . Continuing with this process would yield a result whose shape is influenced by the structure of G . In this simple case, the reconstruction would eventually result in an image identical to G (see Fig. 9.30).

The *geodesic erosion of size 1* of marker F with respect to mask G is defined as

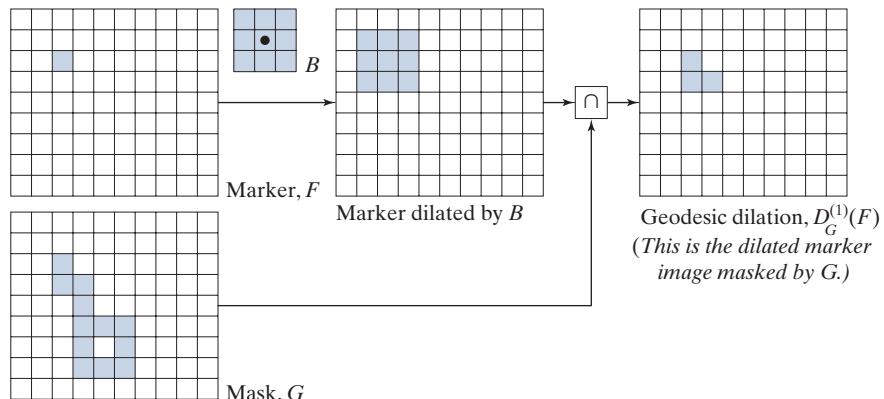
$$E_G^{(1)}(F) = (F \ominus B) \cup G \quad (9-40)$$

[†]In much of the literature on morphological reconstruction, the structuring element is tacitly assumed to be isotropic and typically is called an *elementary isotropic structuring element*. In the context of this chapter, an example of such an SE is a 3×3 array of 1's with the origin at the center.

[‡]Although it is more intuitive to develop morphological reconstruction methods using recursive formulations (as we do here), their practical implementation typically is based on more computationally efficient algorithms (see, for example, Vincent [1993] and Soille [2003]).

FIGURE 9.28

Illustration of a geodesic dilation of size 1. Note that the marker image contains a point from the object in G . If continued, subsequent dilations and maskings would eventually result in the object contained in G .



where \cup denotes set union (or logical OR operation). The geodesic erosion of size n of F with respect to G is defined as

$$E_G^{(n)}(F) = E_G^{(1)}\left(E_G^{(n-1)}(F)\right) \quad (9-41)$$

where $n \geq 1$ is an integer and $E_G^{(0)}(F) = F$. The set union in Eq. (9-40) is performed at each step, and guarantees that geodesic erosion of an image remains greater than or equal to its mask image. As you might have expected from the forms in Eqs. (9-38) and (9-40), geodesic dilation and erosion are duals with respect to set complementation (see Problem 9.41). Figure 9.29 shows an example of a geodesic erosion of size 1. The steps in the figure are a direct implementation of Eq. (9-40).

Geodesic dilation and erosion converge after a finite number of iterative steps, because propagation or shrinking of the marker image is constrained by the mask.

MORPHOLOGICAL RECONSTRUCTION BY DILATION AND BY EROSION

Based on the preceding concepts, *morphological reconstruction by dilation* of a marker image F with respect to a mask image G , denoted $R_G^D(F)$, is defined as the geodesic dilation of F with respect to G , iterated until stability is achieved; that is,

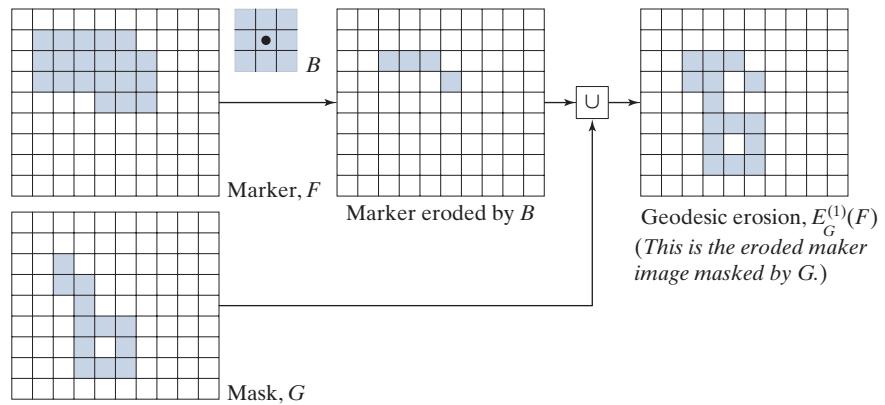
$$R_G^D(F) = D_G^{(k)}(F) \quad (9-42)$$

with k such that $D_G^{(k)}(F) = D_G^{(k+1)}(F)$.

Figure 9.30 illustrates reconstruction by dilation. Figure 9.30(a) continues the process begun in Fig. 9.28. The next step in reconstruction after obtaining $D_G^{(1)}(F)$ is to dilate this result, then AND it with mask G to yield $D_G^{(2)}(F)$, as Fig. 9.30(b) shows. Dilation of $D_G^{(2)}(F)$ and masking with G then yields $D_G^{(3)}(F)$, and so on. This procedure is repeated until stability is reached. Carrying out this example one more step would give $D_G^{(5)}(F) = D_G^{(6)}(F)$, so the image, morphologically reconstructed by dilation, is given by $R_G^D(F) = D_G^{(5)}(F)$, as indicated in Eq. (9-42). The reconstructed image is identical to the mask, as expected.

FIGURE 9.29

Illustration of a geodesic erosion of size 1.



In a similar manner, the *morphological reconstruction by erosion* of a marker image F with respect to a mask image G , denoted $R_G^E(F)$, is defined as the geodesic erosion of F with respect to G , iterated until stability; that is,

$$R_G^E(F) = E_G^{(k)}(F) \quad (9-43)$$

with k such that $E_G^{(k)}(F) = E_G^{(k+1)}(F)$. As an exercise, generate a figure similar to Fig. 9.30 for morphological reconstruction by erosion. Reconstruction by dilation and erosion are duals with respect to set complementation (see Problem 9.42).

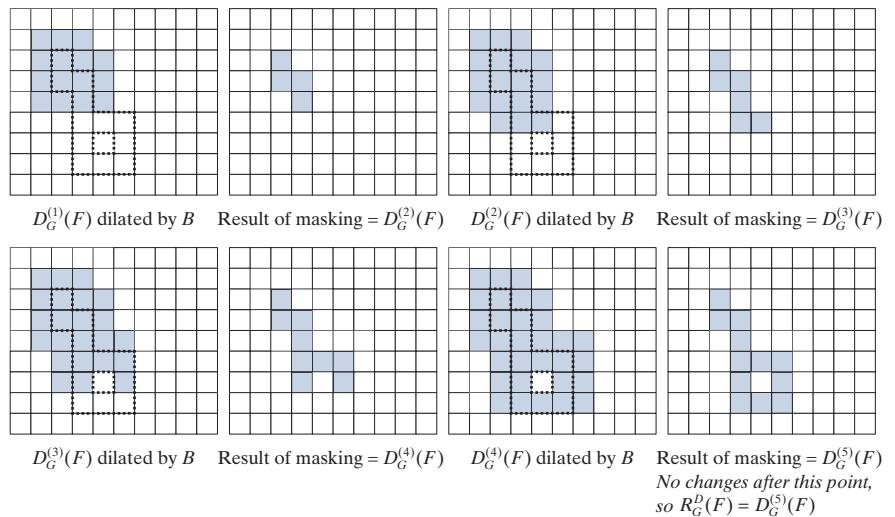
SAMPLE APPLICATIONS

Morphological reconstruction has a broad spectrum of practical applications, each determined by the selection of the marker and mask images, by the structuring

a	b	c	d
e	f	g	h

FIGURE 9.30

Illustration of morphological reconstruction by dilation. Sets $D_G^{(1)}(F)$, G , B and F are from Fig. 9.28. The mask (G) is shown dotted for reference.



elements, and by combinations of the morphological operations defined in the preceding discussion. The following examples illustrate the usefulness of these concepts.

Opening by Reconstruction

In morphological opening, erosion removes small objects and then dilation attempts to restore the shape of the objects that remain. The accuracy of this restoration depends on the similarity of the shapes and the structuring element(s) used. Opening by reconstruction restores exactly the shapes of the objects that remain after erosion. The *opening by reconstruction* of size n of an image F is defined as the reconstruction by dilation of the erosion of size n of F with respect to F ; that is,

$$O_R^{(n)}(F) = R_F^D(F \ominus nB) \quad (9-44)$$

A expression similar to this equation can be written for closing by reconstruction (see Table 9.1 and Problem 9.44).

where $F \ominus nB$ indicates n erosions by B starting with F , as defined in Eq. (9-30). Note that F itself is used as the mask. By comparing this equation with Eq. (9-42), we see that Eq. (9-44) indicates that the opening by reconstruction uses an eroded version of F as the marker in reconstruction by dilation.

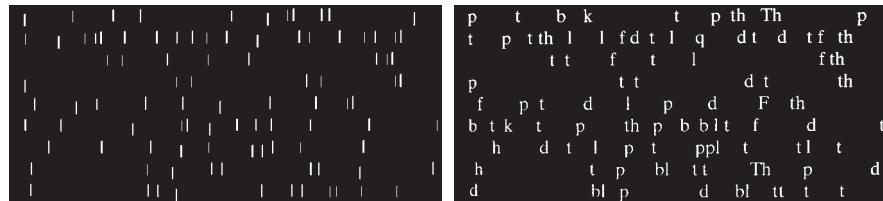
As you will see in Fig. 9.31, Eq. (9-44) can lead to some interesting results. Typically, the structuring element, B , used in Eq. (9-44) is designed to extract some feature of interest, based on erosion. However, as mentioned at the beginning of this section, the structuring element used in reconstruction (i.e., in the dilation that is performed to obtain R_F^D) is designed to define connectivity and, for 2-D, that structuring element typically is a 3×3 array of 1's. It is important that you do not confuse this SE with the structuring element, B , used for erosion in Eq. (9-44). Finally, we point out that this equation is most commonly used with $n = 1$.

Figure 9.31 shows an example of opening by reconstruction. We are interested in extracting from Fig. 9.31(a) the characters that contain long, vertical strokes. This objective determines the nature of B in Eq. (9-44). The average height of the tall characters in the figure is 51 pixels. By eroding the image with a thin structuring element of size 51×1 , we should be able to isolate these characters. Figure 9.31(b) shows one erosion [$n = 1$ in Eq. (9-44)] of Fig. 9.31(a) with the structuring element just mentioned. As you can see, the locations of the tall characters were extracted successively. For the purpose of comparison, we computed the opening (remember this is erosion followed by the dilation) of the image using the same structuring element. Figure 9.31(c) shows the result. As noted earlier, simply dilating an eroded image does not always restore the original. Finally, Fig. 9.31(d) is the reconstruction by dilation of the original image using that image as the mask and the eroded image as the marker. The dilation in the reconstruction was done using a 3×3 SE of 1's, for the reason mentioned earlier. Because we only performed one erosion, the steps just followed constitute the opening by reconstruction (of size 1) of F [i.e., $O_R^{(1)}(F)$] given in Eq. (9-44). As the figure shows, characters containing long vertical strokes were restored accurately from the eroded image (i.e., the marker); all other characters were removed.

A expression similar to Eq. (9-44) can be written for *closing by reconstruction* (see Table 9.1 and Problem 9.44). The difference is that the marker used for closing by reconstruction is the dilation of F and, instead of R_F^D , we use R_F^E . As you saw,

ponents or broken connection paths. There is no point past the level of detail required to identify those.

Segmentation of nontrivial images is one of the most processing. Segmentation accuracy determines the effectiveness of computerized analysis procedures. For this reason, care be taken to improve the probability of rugged segments such as industrial inspection applications, at least some of the environment is possible at times. The experienced designer invariably pays considerable attention to such



a
b
c
d

FIGURE 9.31 (a) Text image of size 918×2018 pixels. The approximate average height of the tall characters is 51 pixels. (b) Erosion of (a) with a structuring element of size 51×1 elements (all 1's). (c) Opening of (a) with the same structuring element, shown for comparison. (d) Result of opening by reconstruction.

opening by reconstruction works with images in which the background is black (0) and the foreground is white (1). Closing by reconstruction works with the opposite scenario. For example, if we were working with the complement of Fig. 9.31(a), the background would be white and the foreground black. To solve the same problem of extracting the tall characters, we would use opening by reconstruction. All the other images in Fig. 9.31 would be identical, except that they would be black on white. The structuring element used would be the *same* in both cases, so the operations of closing by reconstruction would be performed on background pixels.

Automatic Algorithm for Filling Holes

In Section 9.5, we developed an algorithm for filling holes based on knowing a starting point in each hole. Here, we develop a fully automated procedure based on morphological reconstruction. Let $I(x, y)$ denote a binary image, and suppose that we form a marker image F that is 0 everywhere, except at the image border, where it is set to $1 - I$, that is,

$$F(x, y) = \begin{cases} 1 - I(x, y) & \text{if } (x, y) \text{ is on the border of } I \\ 0 & \text{otherwise} \end{cases} \quad (9-45)$$

Then,

$$H = [R_{I^c}^D(F)]^c \quad (9-46)$$

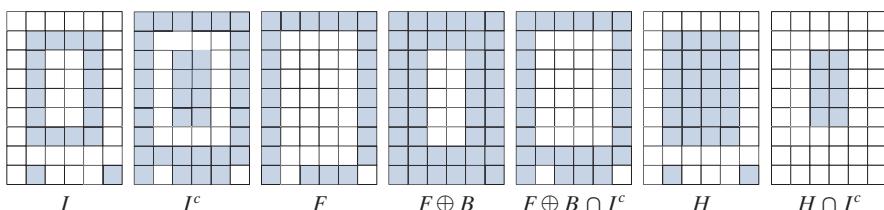
is a binary image equal to I with all holes filled.

To see how Eqs. (9-45) and (9-46) cause holes in an image to be filled, consider Figs. 9.32(a) and (b), which show an image, I , containing one hole, and the image

a b c d e f g

FIGURE 9.32

Hole filling using morphological reconstruction.



complement, respectively. The complement of I sets all foreground (1-valued) pixels to background (0-valued) pixels, and vice versa. By definition, a hole is surrounded by foreground pixels. Therefore, this operation builds a “wall” of 0’s around the hole. Because I^c is used as an AND mask, what we are doing is protecting all foreground pixels from changing during iteration. Figure 9.32(c) is array F , formed according to Eq. (9-45), and Fig. 9.32(d), using a 3×3 SE of 1’s. The marker F has a border of 1’s (except at locations where I is 1), so the dilation of the marker points starts at the border and proceeds inward. Figure 9.32(e) shows the geodesic dilation of F using I^c as the mask. We see that all locations in this result that correspond to foreground pixels of I are now 0, and that this is true for the hole pixels as well. Another iteration will yield the same result which, when complemented as required by Eq. (9-46), gives the result in Fig. 9.32(f). The hole is now filled and the rest of image I was unchanged. The operation $H \cap I^c$ yields an image containing 1-valued pixels in the locations corresponding to the holes in I and 0’s elsewhere, as Fig. 9.32(g) shows.

Figure 9.33 shows a more practical example. Figure 9.33(b) shows the complement of the text image in Fig. 9.33(a), and Fig. 9.33(c) is the marker image, F , generated using Eq. (9-45). This image is all black with a white (1’s) border, except at locations corresponding to 1’s in the border of the original image (the border values are not easily discernible by eye at the magnification shown, and also because the page is nearly white). Finally, Fig. 9.33(d) shows the image with all the holes filled.

Border Clearing

Extracting objects from an image for subsequent shape analysis is a fundamental task in automated image processing. An algorithm for detecting objects that touch (i.e., are connected to) the border is a useful tool because (1) it can be used to screen images so that only complete objects remain for further processing, or (2) it can be used as a signal that partial objects are present in the field of view. As a final illustration of the concepts introduced in this section, we develop a border-clearing procedure based on morphological reconstruction. In this application, we use the original image as the mask and the following marker image:

$$F(x, y) = \begin{cases} I(x, y) & \text{if } (x, y) \text{ is on the border of } I \\ 0 & \text{otherwise} \end{cases} \quad (9-47)$$

The border-clearing algorithm first computes the morphological reconstruction $R_I^D(F)$ (which extracts the objects touching the border), and then computes the following difference:

a b
c d

FIGURE 9.33

- (a) Text image of size 918×2018 pixels.
- (b) Complement of (a) for use as a mask image.
- (c) Marker image.
- (d) Result of hole-filling using Eqs. (9-45) and (9-46).

ponents or broken connection paths. There is no point past the level of detail required to identify those.

Segmentation of nontrivial images is one of the most processing. Segmentation accuracy determines the ev of computerized analysis procedures. For this reason, be taken to improve the probability of rugged segment such as industrial inspection applications, at least some the environment is possible at times. The experienced designer invariably pays considerable attention to suc



ponents or broken connection paths. There is no point past the level of detail required to identify those.

Segmentation of nontrivial images is one of the most processing. Segmentation accuracy determines the ev of computerized analysis procedures. For this reason, be taken to improve the probability of rugged segment such as industrial inspection applications, at least some the environment is possible at times. The experienced designer invariably pays considerable attention to suc

ponents or broken connection paths. There is no point past the level of detail required to identify those.

Segmentation of nontrivial images is one of the most processing. Segmentation accuracy determines the ev of computerized analysis procedures. For this reason, be taken to improve the probability of rugged segment such as industrial inspection applications, at least some the environment is possible at times. The experienced designer invariably pays considerable attention to suc

$$X = I - R_I^D(F) \quad (9-48)$$

to obtain an image, X , with no objects touching the border.

As an example, consider the original text image from Fig. 9.31(a) again. Figure 9.34(a) shows the reconstruction $R_I^D(F)$ obtained using a 3×3 structuring element of 1's. The objects touching the border of the original image are visible in the right side of Fig. 9.34(a). Figure 9.34(b) shows image X , computed using Eq. (9-48). If the task at hand were automated character recognition, having an image in which no characters touch the border is most useful because the problem of having to recognize partial characters (a difficult task at best) is avoided.

9.7 SUMMARY OF MORPHOLOGICAL OPERATIONS ON BINARY IMAGES

Figure 9.35 summarizes the types of structuring elements used in the various binary morphological methods discussed thus far. The shaded elements are foreground values (typically denoted by 1's in numerical arrays), the elements in white are background values (typically denoted by 0's), and the x's are "don't care" elements. Table 9.1 summarizes the binary morphological results developed in the preceding sections. The Roman numerals in the third column of Table 9.1 refer to the structuring elements in Fig. 9.35.

a b

FIGURE 9.34

- (a) Reconstruction by dilation of marker image. (b) Image with no objects touching the border. The original image is Fig. 9.31(a).

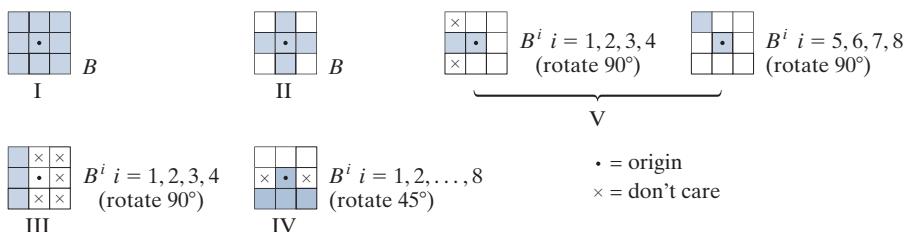


ponents or broken connection paths. There is no point past the level of detail required to identify those.

Segmentation of nontrivial images is one of the most processing. Segmentation accuracy determines the ev of computerized analysis procedures. For this reason, be taken to improve the probability of rugged segment such as industrial inspection applications, at least some the environment is possible at times. The experienced designer invariably pays considerable attention to suc

FIGURE 9.35

Five basic types of structuring elements used for binary morphology.



9.8 GRayscale MORPHOLOGY

In this section, we extend to grayscale images the basic operations of dilation, erosion, opening, and closing. We then use these operations to develop several basic grayscale morphological algorithms. Throughout the discussion that follows, we deal with digital functions of the form $f(x, y)$ and $b(x, y)$, where $f(x, y)$ is a grayscale image and $b(x, y)$ is a structuring element. The assumption is that these functions are discrete in the sense defined in Section 2.4. That is, if Z denotes the set of real integers, then the coordinates (x, y) are integers from the Cartesian product Z^2 , and $f(x, y)$ and $b(x, y)$ are functions that assign an intensity value (a real number from the set of real numbers, R) to each distinct pair of coordinates (x, y) . If the intensity levels are integers also, then Z replaces R .

Structuring elements in grayscale morphology perform the same basic functions as their binary counterparts: They are used as “probes” to examine a given image for specific properties. Structuring elements in grayscale morphology belong to one of two categories: *nonflat* and *flat*. Figure 9.36 shows an example of each. Figure 9.36(a) is a hemispherical grayscale SE shown as an image, and Fig. 9.36(c) is a horizontal intensity profile through its center. Figure 9.34(b) shows a flat structuring element in the shape of a disk, and Fig. 9.36(d) is its corresponding intensity profile. (The shape of this profile explains the origin of the word “flat.”) The elements in Fig. 9.36 are shown as continuous quantities for clarity; their computer implementation is based on digital approximations. Because of a number of difficulties discussed later in this section, grayscale nonflat SEs are not used frequently in practice. Finally, we mention that, as in the binary case, the origin of grayscale structuring elements must be clearly identified. Unless mentioned otherwise, all the examples in this section are based on symmetrical, flat structuring elements of unit height whose origins are at the center. The reflection of an SE in grayscale morphology is as defined in Section 9.1; we denote it in the following discussion by $\hat{b}(x, y) = b(-x, -y)$.

GRAYSCALE EROSION AND DILATION

The *grayscale erosion* of f by a flat structuring element b at location (x, y) is defined as the *minimum* value of the image in the region coincident with $b(x, y)$ when the origin of b is at (x, y) . In equation form, the erosion at (x, y) of an image f by a structuring element b is given as

$$[f \ominus b](x, y) = \min_{(s, t) \in b} \{f(x + s, y + t)\} \quad (9-49)$$

TABLE 9.1

Summary of binary morphological operations and their properties. A is a set of foreground pixels contained in binary image I , and B is a structuring element. I is a binary image (containing A), with 1's corresponding to the elements of A and 0's elsewhere. The Roman numerals refer to the structuring elements in Fig. 9.35.

Operation	Equation	Comments
Translation	$(B)_z = \{c \mid c = b + z, \text{ for } b \in B\}$	Translates the origin of B to point z .
Reflection	$\hat{B} = \{w \mid w = -b, \text{ for } b \in B\}$	Reflects B about its origin.
Complement	$A^c = \{w \mid w \notin A\}$	Set of points not in A .
Difference	$A - B = \{w \mid w \in A, w \notin B\}$ $= A \cap B^c$	Set of points in A , but not in B .
Erosion	$A \ominus B = \{z \mid (B)_z \subseteq A\}$	Erodes the boundary of A . (I)
Dilation	$A \oplus B = \{z \mid (\hat{B})_z \cap A \neq \emptyset\}$	Dilates the boundary of A . (I)
Opening	$A \circ B = (A \ominus B) \oplus B$	Smoothes contours, breaks narrow isthmuses, and eliminates small islands and sharp peaks. (I)
Closing	$A \bullet B = (A \oplus B) \ominus B$	Smoothes contours, fuses narrow breaks and long thin gulfis, and eliminates small holes. (I)
Hit-or-miss transform	$I \circledast B = \{z \mid (B)_z \subseteq I\}$	Finds instances of B in image I . B contains both foreground and background elements.
Boundary extraction	$\beta(A) = A - (A \ominus B)$	Set of points on the boundary of set A . (I)
Hole filling	$X_k = (X_{k-1} \oplus B) \cap I^c$ $k = 1, 2, 3, \dots$	Fills holes in A . X_0 is of same size as I , with a 1 in each hole and 0's elsewhere. (II)
Connected components	$X_k = (X_{k-1} \oplus B) \cap I$ $k = 1, 2, 3, \dots$	Finds connected components in I . X_0 is a set, the same size as I , with a 1 in each connected component and 0's elsewhere. (I)
Convex hull	$X_k^i = (X_{k-1}^i \circledast B^i) \cup X_{k-1}^i;$ $i = 1, 2, 3, 4 \quad k = 1, 2, 3, \dots$ $X_0^i = I; D^i = X_{conv}^i; C(A) = \bigcup_{i=1}^4 D^i$	Finds the convex hull, $C(A)$, of a set, A , of foreground pixels contained in image I . X_{conv}^i means that $X_k^i = X_{k-1}^i$. (III)

TABLE 9.1
(Continued)

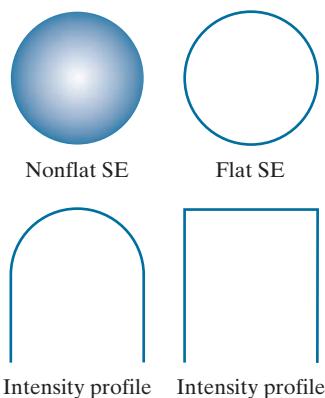
Operation	Equation	Comments
Thinning	$A \otimes B = A - (A \circledast B)$ $= A \cap (A \circledast B)^c$ $A \otimes \{B\} =$ $\left(\dots \left((A \otimes B^1) \otimes B^2 \right) \dots \right) \otimes B^n$ $\{B\} = \{B^1, B^2, B^3, \dots, B^n\}$	Thins set A . The first two equations give the basic definition of thinning. The last two equations denote thinning by a sequence of structuring elements. This method is normally used in practice. (IV)
Thickening	$A \odot B = A \cup (A \circledast B)$ $A \odot \{B\} =$ $\left(\dots \left((A \odot B^1) \odot B^2 \right) \dots \right) \odot B^n$	Thickens set A using a sequence of structuring elements, as above. Uses (IV) with 0's and 1's reversed.
Skeletons	$S(A) = \bigcup_{k=0}^K S_k(A)$ $S_k(A) = (A \ominus kB) - (A \ominus kB) \circ B$	Finds the skeleton $S(A)$ of set A . The last equation indicates that A can be reconstructed from its skeleton subsets $S_k(A)$. K is the value of the iterative step after which the set A erodes to the empty set. The notation $(A \ominus kB)$ denotes the k th iteration of successive erosions of A by B . (I)
Reconstruction of A :	$A = \bigcup_{k=0}^K (S_k(A) \oplus kB)$	
Pruning	$X_1 = A \otimes \{B\}$ $X_2 = \bigcup_{k=1}^8 (X_1 \circledast B^k)$ $X_3 = (X_2 \oplus H) \cap A$ $X_4 = X_1 \cup X_3$	X_4 is the result of pruning set A . The number of times that the first equation is applied to obtain X_1 must be specified. Structuring elements (V) are used for the first two equations. In the third equation H denotes structuring element. (I)
Geodesic dilation-size 1	$D_G^{(1)}(F) = (F \oplus B) \cap G$	F and G are called the <i>marker</i> and the <i>mask</i> images, respectively. (I)
Geodesic dilation-size n	$D_G^{(n)}(F) = D_G^{(1)}(D_G^{(n-1)}(F))$	Same comment as above.
Geodesic erosion-size 1	$E_G^{(1)}(F) = (F \ominus B) \cup G$	Same comment as above.
Geodesic erosion-size n	$E_G^{(n)}(F) = E_G^{(1)}(E_G^{(n-1)}(F))$	Same comment as above.
Morphological reconstruction by dilation	$R_G^D(F) = D_G^{(k)}(F)$	With k is such that $D_G^{(k)}(F) = D_G^{(k+1)}(F)$.

TABLE 9.1
(Continued)

Operation	Equation	Comments
Morphological reconstruction by erosion	$R_G^E(F) = E_G^{(k)}(F)$	With k such that $E_G^{(k)}(F) = E_G^{(k+1)}(F)$.
Opening by reconstruction	$O_R^{(n)}(F) = R_F^D(F \ominus nB)$	$F \ominus nB$ indicates n successive erosions by B , starting with F . The form of B is application-dependent.
Closing by reconstruction	$C_R^{(n)}(F) = R_F^E(F \oplus nB)$	$F \oplus nB$ indicates n successive dilations by B , starting with F . The form of B is application-dependent.
Hole filling	$H = [R_{I^c}^D(F)]^c$	H is equal to the input image I , but with all holes filled. See Eq. (9-45) for the definition of marker image F .
Border clearing	$X = I - R_I^D(F)$	X is equal to the input image I , but with all objects that touch (are connected to) the boundary removed. See Eq. (9-47) for the definition of marker image F .

a	b
c	d

FIGURE 9.36
Nonflat and flat structuring elements, and corresponding horizontal intensity profiles through their centers. All examples in this section are based on flat SEs.



where, in a manner similar to spatial correlation (see Section 3.4), x and y are incremented through all values required so that the origin of b visits every pixel in f . That is, to find the erosion of f by b , we place the origin of the structuring element at every pixel location in the image. The erosion at any location is determined by selecting the minimum value of f in the region coincident with b . For example, if b is a square structuring element of size 3×3 , obtaining the erosion at a point requires finding the minimum of the nine values of f contained in the 3×3 region spanned by b when its origin is at that point.

Similarly, the *grayscale dilation* of f by a flat structuring element b at any location (x, y) is defined as the *maximum* value of the image in the window spanned by \hat{b} when the origin of \hat{b} is at (x, y) . That is,

$$[f \oplus b](x, y) = \max_{(s, t) \in \hat{b}} \{f(x - s, y - t)\} \quad (9-50)$$

where we used the fact stated earlier that $\hat{b}(c, d) = b(-c, -d)$. The explanation of this equation is identical to the explanation in the previous paragraph, but using the maximum, rather than the minimum operation, and keeping in mind that the structuring element is reflected about its origin, which we take into account by using $(-s, -t)$ in the argument of the function. This is analogous to spatial convolution, as explained in Section 3.4.

EXAMPLE 9.9: Grayscale erosion and dilation.

Because grayscale erosion with a flat SE computes the minimum intensity value of f in every neighborhood of (x, y) coincident with b , we expect in general that an eroded grayscale image will be darker than the original, that the sizes (with respect to the size of the SE) of bright features will be reduced, and that the sizes of dark features will be increased. Figure 9.37(b) shows the erosion of Fig. 9.37(a) using a disk SE of unit height and a radius of 2 pixels. The effects just mentioned are clearly visible in the eroded image. For instance, note how the intensities of the small bright dots were reduced, making them barely visible in Fig. 9.37(b), while the dark features grew in thickness. The general background of the eroded image is slightly darker than the background of the original image.

Similarly, Fig. 9.37(c) is the result of dilation with the same SE. The effects are the opposite of using erosion. The bright features were thickened and the intensities of the darker features were reduced. In particular, the thin black connecting wires in the left, middle, and right bottom of Fig. 9.37(a) are barely visible in Fig. 9.37(c). The sizes of the dark dots were reduced as a result of dilation, but, unlike the eroded small white dots in Fig. 9.37(b), they still are easily visible in the dilated image. The reason is that the black dots were originally larger than the white dots with respect to the size of the SE. Finally, observe that the background of the dilated image is slightly lighter than that of Fig. 9.37(a).

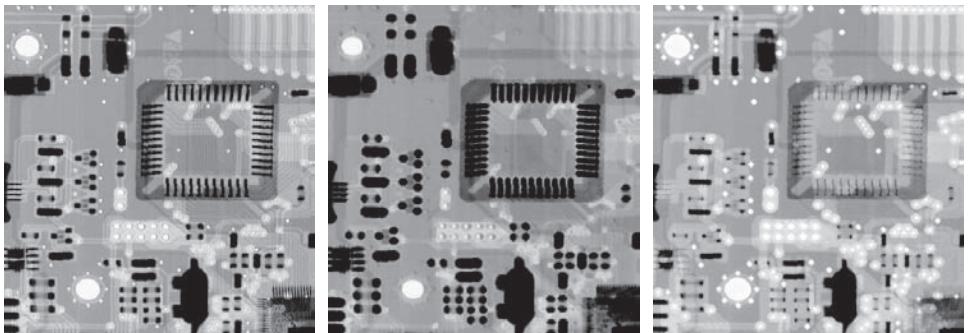
Nonflat SEs have grayscale values that vary over their domain of definition. The erosion of image f by nonflat structuring element, b_N , is defined as

$$[f \ominus b_N](x, y) = \min_{(s, t) \in b_N} \{f(x + s, y + t) - b_N(s, t)\} \quad (9-51)$$

a b c

FIGURE 9.37

(a) Gray-scale X-ray image of size 448×425 pixels. (b) Erosion using a flat disk SE with a radius of 2 pixels. (c) Dilation using the same SE. (Original image courtesy of Lixi, Inc.)



Here, we subtract values from f to determine the erosion at any point. Unlike Eq. (9-49), erosion using a nonflat SE is not bounded in general by the values of f , which can be problematic in interpreting results. Grayscale SEs are seldom used in practice because of this, the potential difficulties in selecting meaningful elements for b_N , and the added computational burden when compared with Eq. (9-49).

In a similar manner, dilation using a nonflat SE is defined as

$$[f \oplus b_N](x, y) = \max_{(s, t) \in \hat{b}_N} \{ f(x-s, y-t) + \hat{b}_N(s, t) \} \quad (9-52)$$

The same comments made in the previous paragraph are applicable to dilation with nonflat SEs. When all the elements of b_N are constant (i.e., the SE is flat), Eqs. (9-51) and (9-52) reduce to Eqs. (9-49) and (9-50), respectively, within a scalar constant equal to the amplitude of the SE.

As in the binary case, grayscale erosion and dilation are duals with respect complementation and reflection; that is,

$$[f \ominus b]^c(x, y) = [f^c \oplus \hat{b}](x, y) \quad (9-53)$$

where $f^c(x, y) = -f(x, y)$ and $\hat{b}(x, y) = b(-x, -y)$. The same expression holds for nonflat structuring elements. Except as needed for clarity, we simplify the notation in the following discussion by suppressing the arguments of all functions, in which case the preceding equation is written as

$$(f \ominus b)^c = f^c \oplus \hat{b} \quad (9-54)$$

Similarly,

$$(f \oplus b)^c = f^c \ominus \hat{b} \quad (9-55)$$

Erosion and dilation by themselves are not particularly useful in grayscale image processing. As with their binary counterparts, these operations become powerful when used in combination to derive higher-level algorithms.

Although we deal with flat SEs in the following discussion, the concepts discussed are applicable also to nonflat structuring elements.

GRAYSCALE OPENING AND CLOSING

The expressions for opening and closing grayscale images have the same form as their binary counterparts. The *grayscale opening* of image f by structuring element b , denoted $f \circ b$, is

$$f \circ b = (f \ominus b) \oplus b \quad (9-56)$$

As before, opening is simply the erosion of f by b , followed by a dilation of the result by b . Similarly, the *grayscale closing* of f by b , denoted $f \bullet b$, is

$$f \bullet b = (f \oplus b) \ominus b \quad (9-57)$$

The opening and closing for grayscale images are duals with respect to complementation and SE reflection:

$$(f \bullet b)^c = f^c \circ \hat{b} \quad (9-58)$$

and

$$(f \circ b)^c = f^c \bullet \hat{b} \quad (9-59)$$

Because $f^c = -f$, we can write Eq. (9-58) as $-(f \bullet b) = (-f \circ b)$, and similarly for Eq. (9-59).

Opening and closing of grayscale images have a simple geometric interpretation. Suppose that an image function $f(x, y)$ is viewed as a 3-D surface; that is, its intensity values are interpreted as height values over the xy -plane, as in Fig. 2.18(a). Then the opening of f by b can be interpreted geometrically as pushing the structuring element up from below against the undersurface of f . At each location of the origin of b , the opening is the highest value reached by any part of b as it pushes up against the undersurface of f . The complete opening is then the set of all such values obtained by the origin of b visiting every (x, y) coordinate of f .

Figure 9.38 illustrates the concept in one dimension. Suppose the curve in Fig. 9.38(a) is the intensity profile along a single row of an image. Figure 9.38(b) shows a flat structuring element in several positions, pushed up against the bottom of the curve. The heavy curve in Fig. 9.38(c) is the complete opening. Because the structuring element is too large to fit completely inside the upward peaks of the curve, the tops of the peaks are clipped by the opening, with the amount removed being proportional to how far the structuring element was able to reach into the peak. In general, openings are used to remove small, bright details, while leaving the overall intensity levels and larger bright features relatively undisturbed.

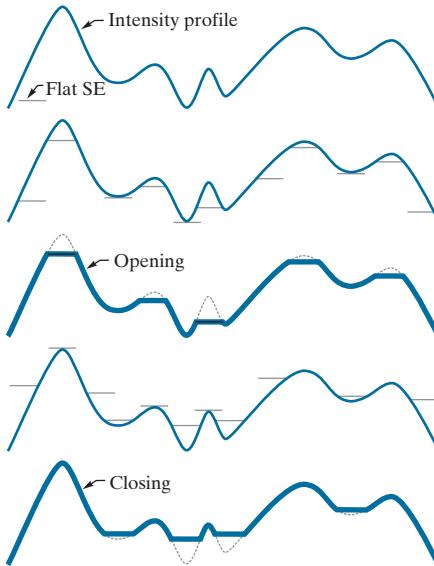
Figure 9.38(d) is a graphical illustration of closing. Observe that the structuring element is pushed down on top of the curve while being translated to all locations. The closing, shown in Fig. 9.38(e), is constructed by finding the lowest points reached by any part of the structuring element as it slides against the upper side of the curve. The grayscale opening satisfies the following properties:

a
b
c
d
e

FIGURE 9.38

Grayscale opening and closing in one dimension.

- (a) Original 1-D signal.
- (b) Flat structuring element pushed up underneath the signal.
- (c) Opening.
- (d) Flat structuring element pushed down along the top of the signal.
- (e) Closing.



(a) $f \circ b \lhd f$

(b) If $f_1 \lhd f_2$, then $(f_1 \circ b) \lhd (f_2 \circ b)$

(c) $(f \circ b) \circ b = f \circ b$

The notation $q \lhd r$ is used to indicate that the domain of q is a subset of the domain of r , and also that $q(x, y) \leq r(x, y)$ for any (x, y) in the domain of q .

Similarly, the closing operation satisfies the following properties:

(a) $f \lhd f \bullet b$

(b) If $f_1 \lhd f_2$, then $(f_1 \bullet b) \lhd (f_2 \bullet b)$

(c) $(f \bullet b) \bullet b = f \bullet b$

The usefulness of these properties is similar to that of their binary counterparts.

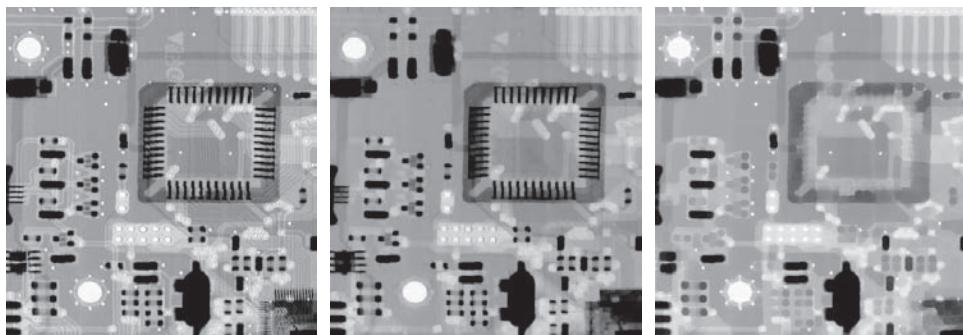
EXAMPLE 9.10: Grayscale opening and closing.

Figure 9.39 extends to 2-D the 1-D concepts illustrated in Fig. 9.38. Figure 9.39(a) is the same image we used in Example 9.9, and Fig. 9.39(b) is the opening obtained using a disk structuring element of unit height and radius of 3 pixels. As expected, the intensity of all bright features decreased, depending on the sizes of the features relative to the size of the SE. Comparing this figure with Fig. 9.37(b), we see that, unlike the result of erosion, opening had negligible effect on the dark features of the image, and the effect on the background was negligible. Similarly, Fig. 9.39(c) shows the closing of the image with a disk of radius 5 (the small round black dots are larger than the small white dots, so a larger disk was needed to achieve results comparable to the opening). In this image, the bright details and background were relatively unaffected, but the dark features were attenuated, with the degree of attenuation being dependent on the relative sizes of the features with respect to the SE.

a b c

FIGURE 9.39

- (a) A grayscale X-ray image of size 448×425 pixels.
 (b) Opening using a disk SE with a radius of 3 pixels.
 (c) Closing using an SE of radius 5.



SOME BASIC GRayscale MORPHOLOGICAL ALGORITHMS

Numerous grayscale morphological techniques are based on the grayscale morphological concepts introduced thus far. We illustrate some of these algorithms in the following discussion.

Morphological Smoothing

Because opening suppresses bright details smaller than the specified SE while leaving dark details relatively unaffected, and closing generally has the opposite effect, these two operations are used often in combination as *morphological filters* for image smoothing and noise removal. Consider Fig. 9.40(a), which shows an image of the Cygnus Loop supernova taken in the X-ray band (see Fig. 1.7 for details about this image). For purposes of the present discussion, suppose that the central light region is the object of interest, and that the smaller components are noise. Our objective is to remove the noise. Figure 9.40(b) shows the result of opening the original image with a flat disk of radius 1, then closing the opening with an SE of the same size. Figures 9.40(c) and (d) show the results of the same operation using SEs of radii 3 and 5, respectively. As expected, this sequence shows progressive removal of small components as a function of SE size. In the last result, we see that the noise has been almost eliminated. The noise components on the lower right side of the image could not be removed completely because their sizes are larger than the other image elements that were successfully removed.

The results in Fig. 9.40 are based on opening the original image, then closing the opening. A procedure used sometimes is to perform *alternating sequential filtering*, in which the opening–closing sequence starts with the original image, but subsequent steps perform the opening and closing on the results of the previous step. This type of filtering is useful in automated image analysis, in which results at each step are compared against a specified metric. This approach generally results in more blurring for the same size SE than the method illustrated in Fig. 9.40.

See Section 3.6 for a definition of the image gradient.

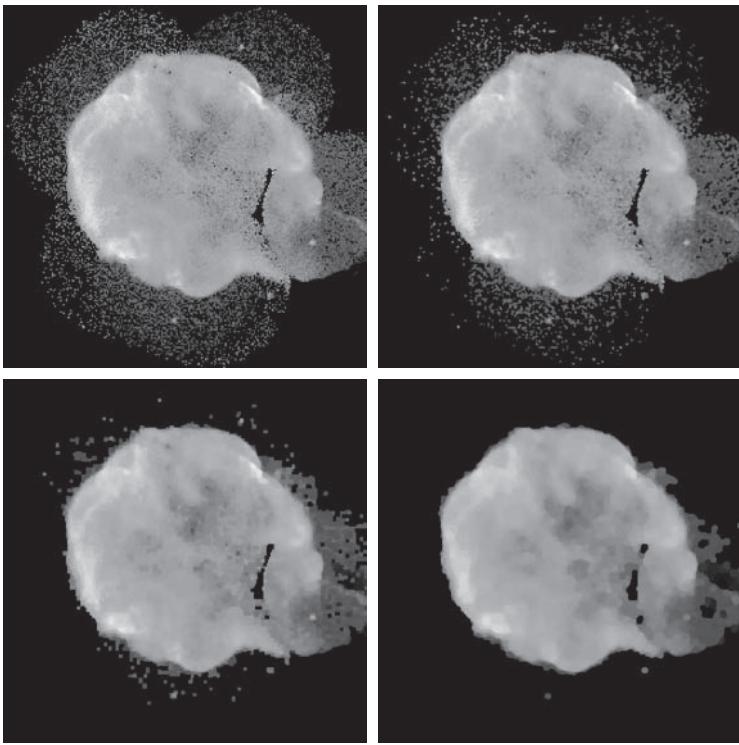
Morphological Gradient

Dilation and erosion can be used in combination with image subtraction to obtain the morphological gradient, g , of a grayscale image f , as follows:

a	b
c	d

FIGURE 9.40

(a) 566×566 image of the Cygnus Loop supernova, taken in the X-ray band by NASA's Hubble Telescope.
 (b)–(d) Results of performing opening and closing sequences on the original image with disk structuring elements of radii, 1, 3, and 5, respectively. (Original image courtesy of NASA.)



$$g = (f \oplus b) - (f \ominus b) \quad (9-60)$$

where b is a suitable structuring element. The overall effect achieved by using this equation is that dilation thickens regions in an image, and erosion shrinks them. Their difference emphasizes the boundaries between regions. Homogenous areas are not affected (provided that the SE is not too large relative to the resolution of the image) so the subtraction operation tends to eliminate them. The net result is an image in which the edges are enhanced and the contribution of the homogeneous areas is suppressed, thus producing a “derivative-like” (gradient) effect.

Figure 9.41 shows an example. Figure 9.41(a) is a head CT scan, and the next two figures are the opening and closing with a 3×3 flat SE of 1's. Note the thickening and shrinking just mentioned. Figure 9.41(d) is the morphological gradient obtained using Eq. (9-60). As you can see, the boundaries between regions were clearly delineated, as expected of a 2-D derivative image.

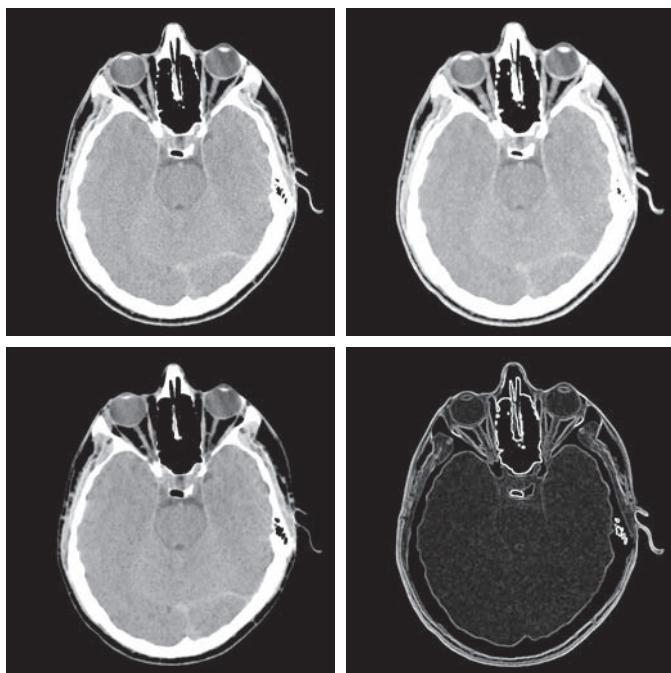
Top-Hat and Bottom-Hat Transformations

Combining image subtraction with openings and closings results in so-called top-hat and bottom-hat transformations. The *top-hat transformation* of a grayscale image f is defined as f minus its opening:

a	b
c	d

FIGURE 9.41

(a) 512×512 image of a head CT scan.
 (b) Dilation.
 (c) Erosion.
 (d) Morphological gradient, computed as the difference between (b) and (c). (Original image courtesy of Dr. David R. Pickens, Vanderbilt University.)



$$T_{\text{hat}}(f) = f - (f \circ b) \quad (9-61)$$

Similarly, the *bottom-hat transformation* of f is defined as the closing of f minus f :

$$B_{\text{hat}}(f) = (f \bullet b) - f \quad (9-62)$$

One of the principal applications of these transformations is in removing objects from an image by using a structuring element in the opening or closing operation that does not fit the objects to be removed. The difference operation then yields an image in which only the removed components remain. The top-hat transformation is used for light objects on a dark background, and the bottom-hat transformation is used for the opposite situation. For this reason, the names *white top-hat* and *black top-hat*, respectively, are used frequently when referring to these two transformations.

An important use of top-hat transformations is for correcting the effects of non-uniform illumination. As you will learn in Chapter 10, proper (uniform) illumination plays a central role in being able to extract objects from the background in an image. This process is fundamental in automated image analysis, and is often used in conjunction with thresholding, as you will learn in Chapter 10.

To illustrate, consider Fig. 9.42(a), which shows an image of grains of rice. This image was obtained under nonuniform lighting, as evidenced by the darker area in the bottom rightmost part of the image. Figure 9.42(b) shows the result of thresholding using Otsu's method, an optimal thresholding method to be discussed in Section 10.3. The net result of nonuniform illumination was to cause segmentation

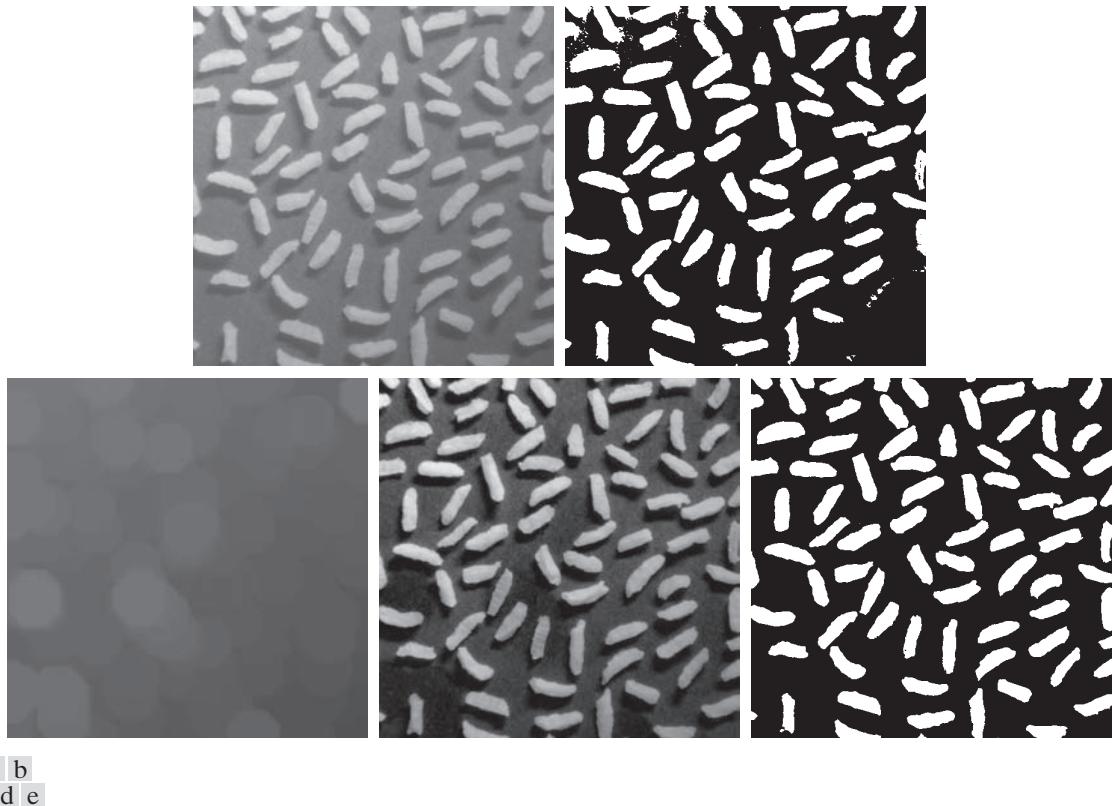


FIGURE 9.42 Using the top-hat transformation for *shading correction*. (a) Original image of size 600×600 pixels. (b) Thresholded image. (c) Image opened using a disk SE of radius 40. (d) Top-hat transformation (the image minus its opening). (e) Thresholded top-hat image.

errors in the dark area (several grains of rice were not extracted from the background), as well as in the top left part of the image, where parts of the background were interpreted as rice. Figure 9.42(c) shows the opening of the image with a disk of radius 40. This SE was large enough so that it would not fit in any of the objects. As a result, the objects were eliminated, leaving only an approximation of the background. The shading pattern is clear in this image. By subtracting this image from the original (i.e., by applying a top-hat transformation), the background should become more uniform. This is indeed the case, as Fig. 9.42(d) shows. The background is not perfectly uniform, but the differences between light and dark extremes are less, and this was enough to yield a correct thresholding result, in which all the rice grains were properly extracted using Otsu's method, as Fig. 9.42(e) shows.

Granulometry

In the context of this discussion, *granulometry* is a field that deals with determining the size distribution of particles in an image. Particles seldom are neatly separated,

which makes counting based on identifying individual particles a difficult task. Morphology can be used to estimate particle size distribution indirectly, without having to identify and measure individual particles.

The approach is simple. With particles having regular shapes that are lighter than the background, the method consists of applying openings with SEs of increasing sizes. The basic idea is that opening operations of a particular size should have the most effect on regions of the input image that contain particles of similar size. For each image resulting from an opening, we compute the sum of the pixel values. This sum, called the *surface area*, decreases as a function of increasing SE size because, as we discussed earlier, openings decrease the intensity of light features in an image. This procedure yields a 1-D array each element of which is the sum of the pixels in the opening for the size SE corresponding to that location in the array. To emphasize changes between successive openings, we compute the difference between adjacent elements of the 1-D array. If the differences are plotted, the peaks in the plot are an indication of the predominant size distributions of the particles in the image.

As an example, consider the image of wood dowel plugs of two dominant sizes in Fig. 9.43(a). The wood grain in the dowels is likely to introduce variations in the openings, so smoothing is a sensible preprocessing step. Figure Fig. 9.43(b) shows the image smoothed using the morphological smoothing filter discussed earlier, with a disk of radius 5. Figures 9.43(c) through (f) show image openings with disks of radii 10, 20, 25, and 30, respectively. Note in Fig. 9.43(d) that the intensity contribution due to the small dowels has been almost eliminated. In Fig. 9.43(e) the contribution of the large dowels has been reduced significantly, and in Fig. 9.43(f) even more so. Observe in Fig. 9.43(e) that the large dowel near the top right of the image is much darker than the others because its size is smaller than other larger dowels. This would be useful information if we had been attempting to detect defective dowels.

Figure 9.44 shows a plot of the difference array. As mentioned previously, we expect significant differences (peaks in the plot) around radii at which the SE is

a	b	c
d	e	f

FIGURE 9.43

- (a) 531×675 image of wood dowels.
- (b) Smoothed image.
- (c)–(f) Openings of (b) with disks of radii equal to 10, 20, 25, and 30 pixels, respectively.
(Original image courtesy of Dr. Steve Eddins, MathWorks, Inc.)

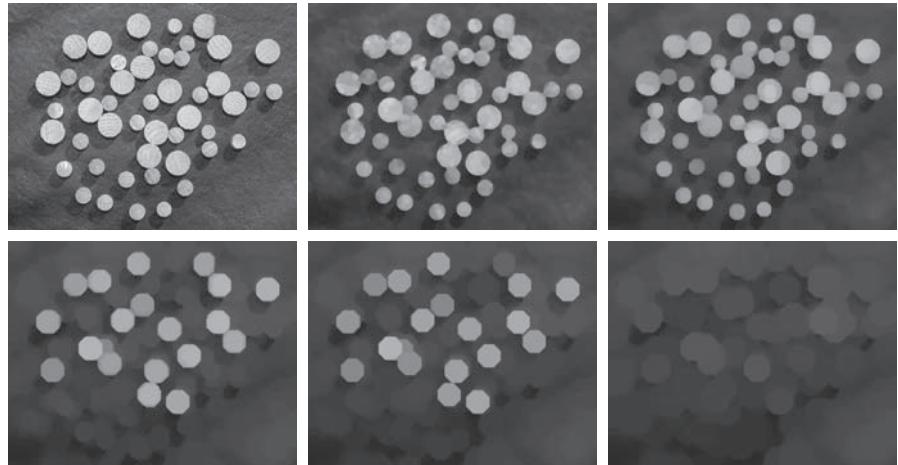
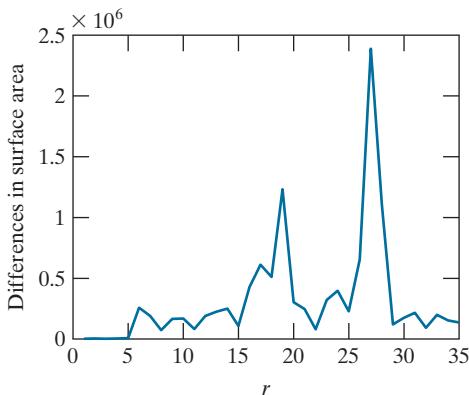


FIGURE 9.44

Differences in surface area as a function of SE disk radius, r . The two peaks indicate that there are two dominant particle sizes in the image.



large enough to encompass a set of particles of approximately the same diameter. The result in Fig. 9.44 has two distinct peaks, clearly indicating the presence of two dominant object sizes in the image.

Textural Segmentation

Figure 9.45(a) shows a noisy image of dark blobs superimposed on a light background. The image has two textural regions: a region composed of large blobs on the right and a region on the left composed of smaller blobs. The objective is to find a boundary between the two regions based on their textural content, which in this case is determined by the sizes and spatial distribution of the blobs (we discuss texture in Chapter 11). The process of partitioning an image into regions is called *segmentation*, which is the topic of Chapter 10.

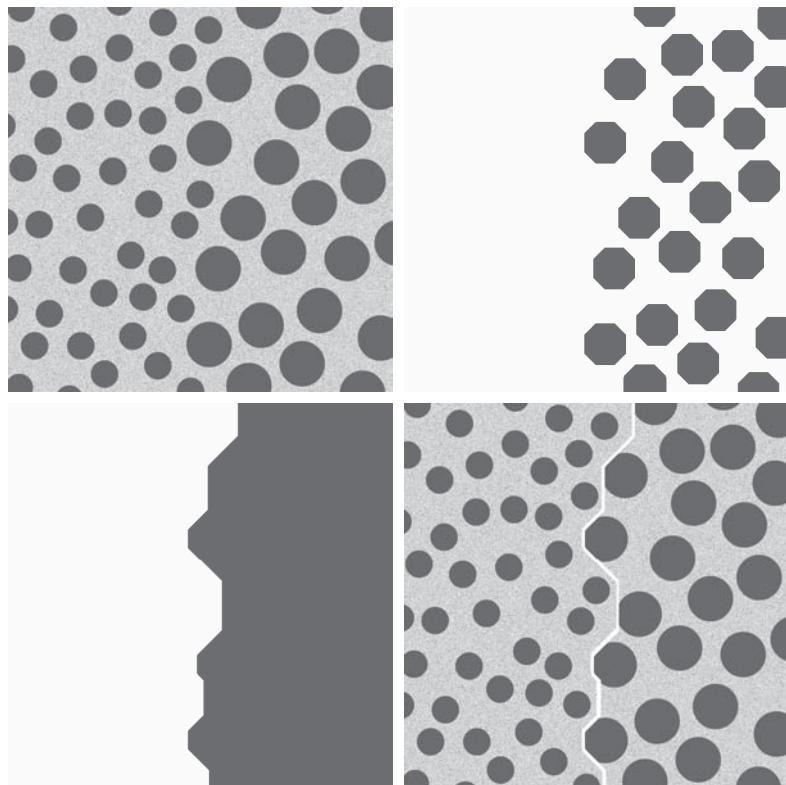
The objects of interest are darker than the background, and we know that if we close the image with a structuring element larger than the small blobs, these blobs will be removed. The result in Fig. 9.45(b), obtained by closing the input image using a disk with a radius of 30 pixels, shows that indeed this is the case. (The radius of the smaller blobs is approximately 25 pixels.) So, at this point, we have an image with large, dark blobs on a light background. If we open this image with a structuring element that is large relative to the separation between these blobs, the net result should be an image in which the light patches between the blobs are removed, leaving the dark blobs, and also the now dark patches between these blobs. Figure 9.45(c) shows the result, obtained using a disk of radius 60.

Performing a morphological gradient on this image with, say, a 3×3 SE of 1's, will give us the boundary between the two regions. Figure 9.45(d) shows the boundary obtained from the morphological gradient operation, superimposed on the original image. All pixels to the right of this boundary are said to belong to the texture region characterized by large blobs, and conversely for the pixels on the left of the boundary. You will find it instructive to work through this example in more detail using the graphical analogy for opening and closing illustrated in Fig. 9.38.

a	b
c	d

FIGURE 9.45

Textural segmentation.
 (a) A 600×600 image consisting of two types of blobs.
 (b) Image with small blobs removed by closing (a).
 (c) Image with light patches between large blobs removed by opening (b).
 (d) Original image with boundary between the two regions in (c) superimposed. The boundary was obtained using a morphological gradient.



GRAYSCALE MORPHOLOGICAL RECONSTRUCTION

As mentioned earlier, it is understood that f and g are functions of x and y . We omit the coordinates to simplify the notation.

Grayscale morphological reconstruction is defined in the same manner introduced in Section 9.6 for binary images. Let f and g denote the *marker* and *mask* images, respectively. We assume that both are grayscale images of the same size and that $f \leq g$, meaning that the intensity of f at any point in the image is less than the intensity of g at that point. The *geodesic dilation* of size 1 of f with respect to g is defined as

$$D_g^{(1)}(f) = (f \oplus b) \wedge g \quad (9-63)$$

where \wedge denotes the *point-wise minimum operator*, and b is a suitable structuring element. We see that the geodesic dilation of size 1 is obtained by first computing the dilation of f by b , then selecting the minimum between the result and g at every point (x, y) . The dilation is given by Eq. (9-50) if b is a flat SE, or by Eq. (9-52) if it is not.

The *geodesic dilation* of size n of f with respect to g is defined as

$$D_g^{(n)}(f) = D_g^{(1)}\left(D_g^{(n-1)}(f)\right) \quad (9-64)$$

with $D_g^{(0)}(f) = f$.

See Problem 9.33 for a list of dual relationships between expressions in this section.

Similarly, the *geodesic erosion of size 1* of f with respect to g is defined as

$$E_g^{(1)}(f) = (f \ominus b) \vee g \quad (9-65)$$

where \vee denotes the *point-wise maximum operator*. The *geodesic erosion of size n* is defined as

$$E_g^{(n)}(f) = E_g^{(1)}(E_g^{(n-1)}(f)) \quad (9-66)$$

with $E_g^{(0)}(f) = f$.

The *morphological reconstruction by dilation* of a grayscale mask image, g , by a grayscale marker image, f , denoted by $R_g^D(f)$, is defined as the geodesic dilation of f with respect to g , iterated until stability is reached; that is,

$$R_g^D(f) = D_g^{(k)}(f) \quad (9-67)$$

with k such that $D_g^{(k)}(f) = D_g^{(k+1)}(f)$. The *morphological reconstruction by erosion* of g by f , denoted by $R_g^E(f)$, is similarly defined as

$$R_g^E(f) = E_g^{(k)}(f) \quad (9-68)$$

with k such that $E_g^{(k)}(f) = E_g^{(k+1)}(f)$.

As in the binary case, opening by reconstruction of grayscale images first erodes the input image and uses it as a marker, and uses the image itself as the mask. The *opening by reconstruction of size n* of an image f is defined as the reconstruction by dilation of the erosion of size n of f with respect to f ; that is,

$$O_R^{(n)}(f) = R_f^D(f \ominus nb) \quad (9-69)$$

where $f \ominus nb$ denotes n successive erosions by b , starting with f , as explained in connection with Eq. (9-30) (note that f itself is used as the mask). Recall also from the discussion of Eq. (9-44) for binary images that the objective of opening by reconstruction is to preserve the shape of the image components that remain after erosion.

Similarly, the *closing by reconstruction of size n* of an image f is defined as the reconstruction by erosion of the dilation of size n of f with respect to f ; that is,

$$C_R^{(n)}(f) = R_f^E(f \oplus nb) \quad (9-70)$$

where $f \oplus nb$ denotes n successive dilations by b , starting with f . Because of duality, the closing by reconstruction of an image can be obtained by complementing the image, obtaining the opening by reconstruction, and complementing the result. Finally, as the following example shows, a useful technique called *top-hat by reconstruction* consists of subtracting from an image its opening by reconstruction.

EXAMPLE 9.11: Using grayscale morphological reconstruction to flatten a complex background.

In this example, we illustrate the use of grayscale reconstruction in several steps. The objective is to normalize the irregular background of the image in Fig. 9.46(a), leaving only text on a background of

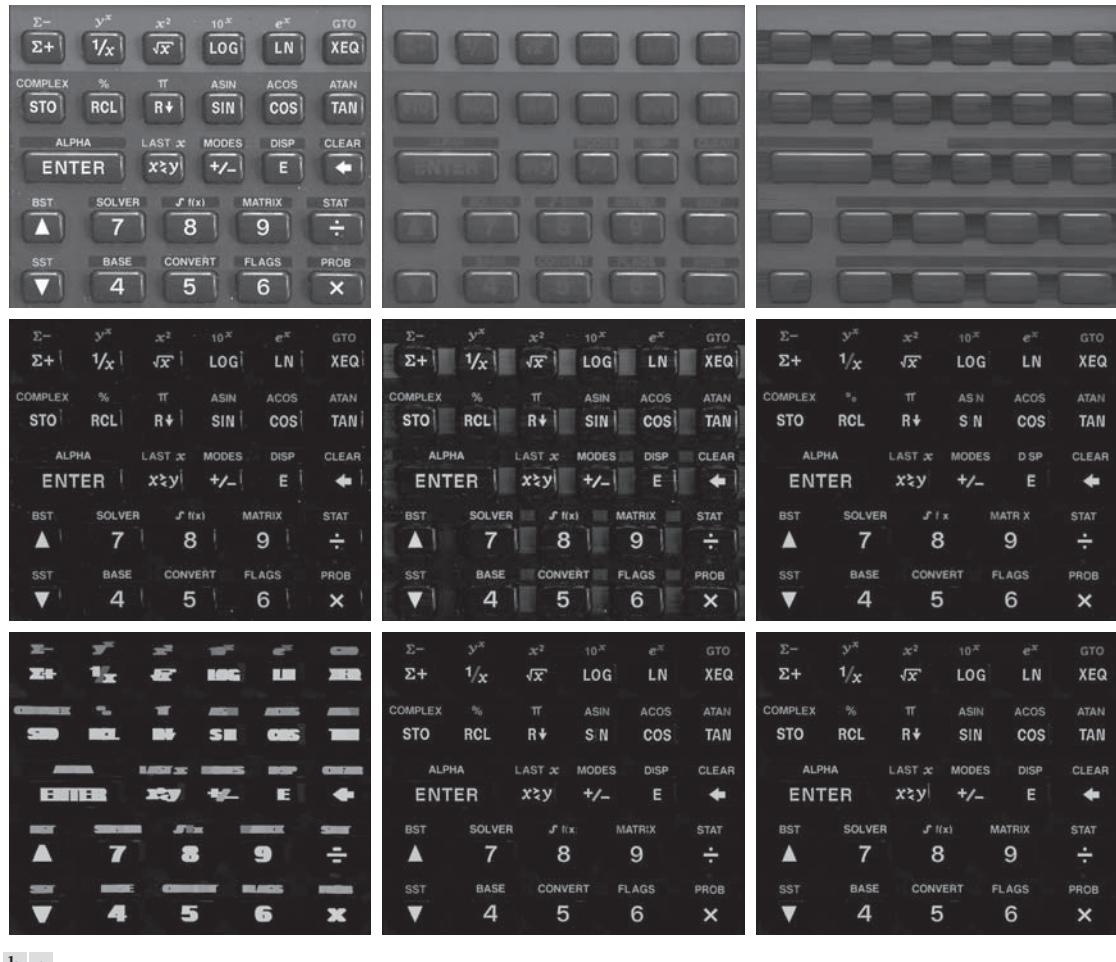


FIGURE 9.46 (a) Original image of size 1134×1360 pixels. (b) Opening by reconstruction of (a), using a structuring element consisting of a horizontal line 71 pixels long in the erosion. (c) Opening of (a) using the same SE. (d) Top-hat by reconstruction. (e) Result of applying just a top-hat transformation. (f) Opening by reconstruction of (d), using a horizontal line 11 pixels long. (g) Dilation of (f) using a horizontal line 21 pixels long. (h) Minimum of (d) and (g). (i) Final reconstruction result. (Images courtesy of Dr. Steve Eddins, MathWorks, Inc.)

constant intensity. The solution of this problem is a good illustration of the power of grayscale morphology. We begin by suppressing the horizontal reflection on the top of the keys. The reflections are wider than any single character in the image, so we should be able to suppress them by performing an opening by reconstruction using a long horizontal line in the erosion operation. This operation will yield the background containing the keys and their reflections. Subtracting this from the original image (i.e., performing a top-hat by reconstruction) will eliminate the horizontal reflections and variations in background from the original image.

Figure 9.46(b) shows the result of opening by reconstruction of the original image using a horizontal line of size 1×71 pixels for the SE in the erosion operation. We could have used an opening to remove the characters, but the resulting background would not have been as uniform, as Fig. 9.46(c) shows (compare the regions between the keys in the two images). Figure 9.46(d) shows the result of subtracting Fig. 9.46(b) from Fig. 9.46(a). As expected, the horizontal reflections and variations in background were suppressed. For comparison, Fig. 9.46(e) shows the result of performing just a top-hat transformation (i.e., subtracting the “standard” opening from the image). As expected from the characteristics of the background in Fig. 9.46(c), the background in Fig. 9.46(e) is not nearly as uniform as in Fig. 9.46(d).

The next step is to remove the vertical reflections from the edges of the keys, visible in Fig. 9.46(d). We can do this by performing an opening by reconstruction with a line SE whose width is approximately equal to the reflections (about 11 pixels in this case). Figure 9.46(f) shows the result of performing this operation on Fig. 9.46(d). The vertical reflections were suppressed, but so were thin, vertical strokes that are valid characters (for example, the I in SIN), so we have to find a way to restore the latter. The suppressed characters are very close to the other characters so, if we dilate the remaining characters horizontally, the dilated characters will overlap the area previously occupied by the suppressed characters. Figure 9.46(g), obtained by dilating Fig. 9.46(f) with a line SE of size 1×21 elements, shows that indeed this is the case.

All that remains at this point is to restore the suppressed characters. Consider an image formed as the point-wise minimum between the dilated image in Fig. 9.46(g) and the top-hat by reconstruction in Fig. 9.46(d). Figure 9.46(h) shows the minimum image (although this result appears to be close to our objective, note that the I in SIN is still missing). By using this image as a marker and the dilated image as the mask in grayscale reconstruction [Eq. (9-67)], we obtained the final result in Fig. 9.46(i). This image shows that all characters were properly extracted from the original, irregular background, including the background of the keys. The background in Fig. 9.46(i) is uniform throughout.

Summary, References, and Further Reading

The morphological concepts and techniques introduced in this chapter constitute a powerful set of tools for extracting features of interest in an image. One of the most appealing aspects of morphological image processing is the extensive set-theoretic foundation from which morphological techniques have evolved. A significant advantage in terms of implementation is that dilation and erosion are primitive operations, which are the basis for a broad class of morphological algorithms. As will be shown in the following chapter, morphology can be used as the basis for developing image segmentation procedures with numerous applications. As we will discuss in Chapter 11, morphological techniques also play a major role in procedures for image feature extraction.

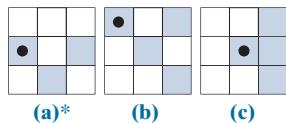
The book by Serra [1982] is a fundamental reference on morphological image processing. See also Serra [1988], Giardina and Dougherty [1988], and Haralick and Shapiro [1992]. For an overview of both binary and gray-scale morphology, see Basart and Gonzalez [1992] and Basart et al. [1992]. This set of references provides ample basic background for the material covered in Sections 9.1 through 9.4. For a good overview of the material in Sections 9.5 and 9.6, see the book by Soille [2003].

Important issues of implementing morphological algorithms such as the ones given in Section 9.5 and 9.6 are exemplified in the papers by Jones and Svalbe [1994], Sussner and Ritter [1997], and Shaked and Bruckstein [1998]. A paper by Vincent [1993] is especially important in terms of practical details for implementing gray-scale morphological algorithms. For additional reading on the theory and applications of morphological image processing, see the books by Goutsias and Bloomberg [2000], and by Beyerer et al. [2016]. To get an idea of the state of the art in fast computer implementation of morphological algorithms, see Thurley and Danell [2012]. For details on the software aspects of many of the examples in this chapter, see Gonzalez, Woods, and Eddins [2009].

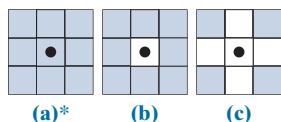
Problems

Solutions to the problems marked with an asterisk (*) are in the DIP4E Student Support Package (consult the book website: www.ImageProcessingPlace.com).

- 9.1** Find the reflection, \hat{B} , of each of the following structuring elements. The dot indicates the origin of the SE.



- 9.2** Sketch the result of eroding Fig. 9.3(a) with each of the following structuring elements.



- 9.3*** Erosion of a set A by structuring element B is a subset of A , provided that the origin of B lies within B . Give an example in which the erosion $A \ominus B$ lies outside, or partially outside, A .

- 9.4** Let B be a structuring element containing a single point, valued 1, and let A be a set of foreground pixels.

(a)* What do you think would happen if we erode A by B ?

(b) What do you think would happen if we dilate A by B ?

- 9.5** You are given a “black-box” function that computes erosion. You are told that this function automatically pads the input image with a border whose width is the thinnest border possible, as determined by the dimensions of the structuring element (e.g., for a 3×3 structuring element the border would be one pixel wide). However, you are not told whether the padding is composed of background (0) or foreground (1) values. Propose an experiment for answering this question.

- 9.6** Do the following:

(a)* Dilate Fig. 9.3(a) using the structuring element in figure (a) of Problem 9.2.

(b) Repeat (a) using the structuring element in figure (b).

- (c) Repeat (a) using the structuring element in figure (c).

- 9.7** Dilation of a set A by structuring element B is the set of locations of the origin of B such that A contains at least one (foreground) element of B . Give an example in which the dilation of A by B lies completely outside of A . (Hint: Let A and B be disks of different radii.)

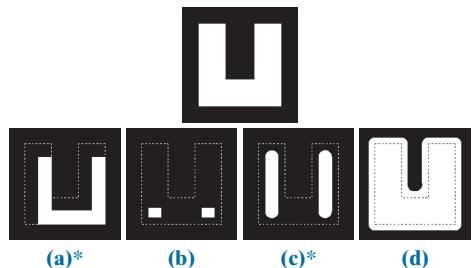
- 9.8** With reference to the image at the top of the figure shown below, answer the following:

(a)* Give the structuring element and morphological operation(s) that produced image (a). Show the origin of the structuring element. The dashed lines denote the boundary of the original object and are shown for reference; they are not part of the result. (The white elements are foreground pixels.)

(b) Repeat part (a) for the output shown in image (b).

(c)* Repeat part (a) for the output shown in image (c).

(d) Repeat part (a) for the solution shown in figure (d). Note that in image (d) all corners are rounded.

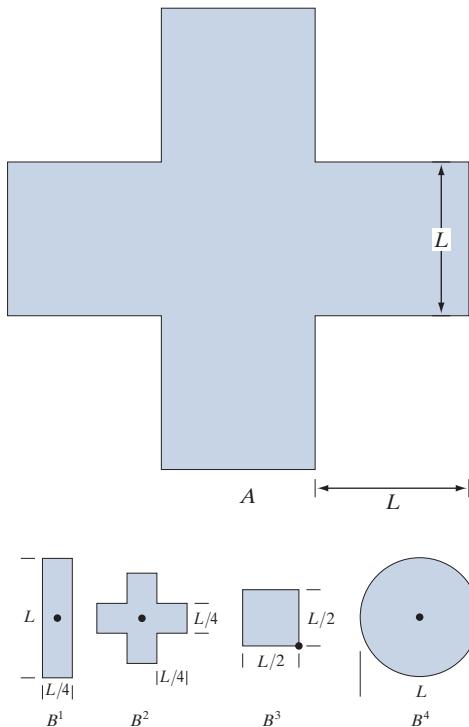


- 9.9** Let A denote the set shown shaded in the following figure, and refer to the structuring elements shown (the black dots denote the origin). Sketch the result of the following operations:

(a)* $(A \ominus B^4) \oplus B^2$.

(b) $(A \ominus B^1) \oplus B^3$.

(c) $(A \oplus B^1) \ominus B^3$.



9.10 Be specific in answering the following:

- (a)* What is the limiting effect of repeatedly dilating a set of foreground pixels in an image? Assume that a trivial (one point) structuring element is not used.
- (b) What is the smallest set from which you can start in order for your answer in (a) to hold?

9.11 Be specific in answering the following:

- (a) What is the limiting effect of repeatedly eroding a set of foreground pixels in an image? Assume that a trivial (one point) structuring element is not used.
- (b) What is the smallest set of foreground pixels from which you can start in order for your answer in (a) to hold?

9.12* An alternative definition of erosion is

$$A \ominus B = \{w \in Z^2 \mid w + b \in A \text{ for every } b \in B\}$$

Show that this definition is equivalent to the definition in Eq. (9-3).

9.13 Do the following:

- (a) Show that the definition of erosion given in Problem 9.12 is equivalent to yet another definition of erosion:

$$A \ominus B = \bigcap_{b \in B} (A)_{-b}$$

(If $-b$ is replaced with b , this expression is called the *Minkowsky subtraction* of two sets.)

- (b)* Show that the expression in (a) is equivalent to the definition in Eq. (9-3).

9.14* An alternative definition of dilation is

$$A \oplus B = \{w \in Z^2 \mid w = a + b, \text{ for some } a \in A \text{ and } b \in B\}$$

Show that this definition and the definition in Eq. (9-6) are equivalent.

9.15 Do the following:

- (a) Show that the definition of dilation given in Problem 9.14 is equivalent to yet another definition of dilation:

$$A \oplus B = \bigcup_{b \in B} (A)_b$$

(This expression is called the *Minkowsky addition* of two sets.)

- (b)* Show that the expression in (a) is equivalent also to the definition in Eq. (9-6).

9.16 Prove the validity of the duality expression given in Eq. (9-9).

9.17 Answer the following:

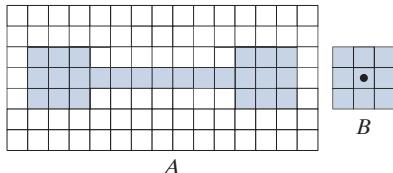
- (a)* The curved portions the black border of Fig. 9.8(d) delineate the opening of set A in Fig. 9.8(a), but those curved segments are not part of the boundary of A . Are the black straight-line portions in (d) part of the boundary of A ? Explain.

- (b) The curved portions the black border of Fig. 9.9(d) delineate the closing of set A in Fig. 9.9(a), but those curved segments are not part of the boundary of A . Are the black straight line portions of the boundary in (d) part of the boundary of A ? Explain.

- 9.18** Show all intermediate steps of your computations for the following:

(a)* Obtain the opening of the figure below using a 3×3 SE of 1's. Do all operations manually.

(b) Repeat (a) for the closing operation.



- 9.19** A is a solid rectangle of 1's of size $M \times N$ with a 1-pixel border of 0's, and m and n below are odd integers. Discuss what the result will be in each case.

(a)* A is opened with a structuring element of 1's of size $m \times n$.

(b) A is closed with a structuring element of 1's of size $m \times n$.

- 9.20** Show the validity of the following duality expressions [these are Eqs. (9-14) and (9-15)]:

(a)* $(A \circ B)^c = A^c \cdot \hat{B}$.

(b) $(A \cdot B)^c = A^c \circ \hat{B}$.

- 9.21** Show the validity of the following expressions:

(a)* $A \circ B$ is a subset of A . You may assume that Eq. (9-12) is valid. (Hint: Start with this equation and Fig. 9.8.)

(b)* If C is a subset of D , then $C \circ B$ is a subset of $D \circ B$. [Hint: Start with Eq. (9-12).]

(c) $(A \circ B) \circ B = A \circ B$. [Hint: Start with the definition of opening.]

- 9.22** Show the validity of the following expressions. (Hint: Study the solution to Problem 9.21.)

(a) A is a subset of $A \cdot B$.

(b) If C is a subset of D , then $C \cdot B$ is a subset of $D \cdot B$.

(c) $(A \cdot B) \cdot B = A \cdot B$.

- 9.23** Refer to the image and the disk structuring element shown in the lower right of the image. Sketch what the sets C , D , E , and F would look like for the following sequence of operations: $C = A \ominus B$; $D = C \oplus B$; $E = D \oplus B$; and $F = E \ominus B$. Set A consists of all the foreground pixels (white),

except the structuring element, B , which you may assume is just large enough to encompass any of the random elements in the image. Note that the sequence of operations above is simply the opening of A by B followed by a closing of the result by B .



- 9.24*** Assume that SE B_2 in Fig. 9.12 has a border of foreground pixels that is more than one pixel wide. Assuming that all four sides of the border are the same, what is the maximum width of a border we can use around B_2 before the solution shown in Fig. 9.12(f) fails?

- 9.25** We mentioned when discussing Fig. 9.12(e) that the image had been cropped for consistency. Assume that Fig. 9.12(b) was padded with the minimum border required to encompass the maximum excursions of B_2 after which no further changes would occur in the erosion. What did Fig. 9.12(e) look like before it was cropped?

- 9.26** Sketch the result of applying the hit-or-miss transform to the image below, using the SE shown. Indicate clearly the origin and border you selected for the structuring element.



Image



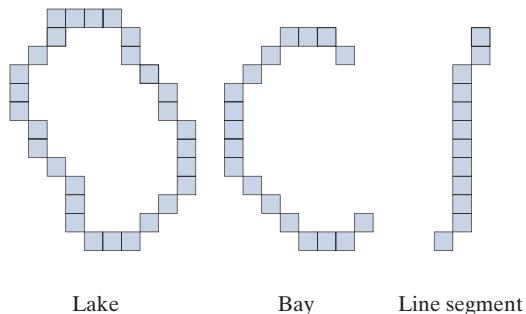
Structuring element

9.27* Give the foundation of an algorithm for converting an 8-connected, closed curve to a 4-connected curve (see Section 2.5 regarding connectivity). The input is a binary image, I , in which the curve consists of 1-valued pixels embedded in a background of 0's. The output should be a binary image also, containing the new curve. You may assume that the curve is fully connected, is one pixel thick, and has no branches. You do not need to (but you may) state the algorithm in a step-by-step manner. An overall plan containing all the information needed to implement a working algorithm is sufficient.

9.28 Give the foundation of an algorithm for converting a 4-connected closed curve to a curve containing *only* 8-connected pixels (see Section 2.5 regarding connectivity). The input is a binary image, I , in which the curve consists of 1-valued pixels embedded in a background of 0's. The output should be a binary image also, containing the new curve. You may assume that the curve is fully connected, it is one-pixel-wide, and has no branches. You do not need to (but you may) state the algorithm in a step-by-step manner. An overall plan containing all the information needed to implement a working algorithm is sufficient.

9.29 Give the foundation of an algorithm for converting an 8-connected closed curve to an m -connected curve (see Section 2.5 regarding connectivity). The input is a binary image, I , in which the curve consists of 1-valued pixels embedded in a background of 0's. The output should be a binary image also, containing the new curve. You may assume that the curve is fully connected, it is one-pixel-wide, and has no branches. You do not need to (but you may) state the algorithm in a step-by-step manner. An overall plan containing all the information needed to implement a working algorithm is sufficient.

9.30* Three curve types (lake, bay, and line segment) useful for differentiating thinned objects in an image are shown in the following figure. Develop a morphological/logical algorithm for differentiating between these shapes. The input to your algorithm would be one of these three curves. The output must be the type of the input. You may assume that the curves are 1 pixel thick and are fully connected. They can appear in any orientation.



9.31 Write Eq. (9-18) in terms of a dilation, instead of an erosion, of A . (*Hint:* Take a look at the definition of set difference in Eq. (2-40) and then consider the duality relationship between erosion and dilation.)

9.32 Answer the following:

- (a)* Discuss the effect of using the structuring element in Fig. 9.17(c) for boundary extraction, instead of the element in Fig. 9.15(b).
- (b) What would be the effect of using a 3×3 structuring element composed of all 1's in the hole filling algorithm of Eq. (9-19), instead of the structuring element in Fig. 9.17(c)?

9.33 Discuss what you would expect the result to be in each of the following cases:

- (a)* The starting point of the hole filling algorithm of Eq. (9-19) is a point *on* the outer boundary of the object containing the hole.
- (b) The starting point in the hole filling algorithm is *outside* of the boundary (i.e., the starting point is a background pixel).

9.34 Sketch the convex hull of the large figure in Problem 9.9. Assume that $L = 3$ pixels.

9.35 Obtain the convex deficiency of set A shown in Fig. 9.21(b). Use the convex hull in Fig. 9.22(a).

9.36 Do the following:

- (a)* Propose a method using any of the methods developed in this chapter for automating the example in Fig. 9.18. You may assume that the spheres do not touch each other and that none touch the border of the image.
- (b) Repeat (a), but allowing the spheres to touch in arbitrary ways, including the border.

9.37* The algorithm for extracting connected components discussed in Section 9.5 requires that a point be known in each connected component in order to extract them all. Suppose that you are given a binary image containing an arbitrary (unknown) number of connected components. Propose a completely automated procedure for extracting all connected components. Assume that points belonging to connected components are labeled 1 and background points are labeled 0.

9.38 Give an expression based on reconstruction by dilation capable of extracting all the holes in a binary image.

9.39 With reference to the hole-filling algorithm in Eqs. (9-45) and (9-46):

- (a)* Explain what would happen if all border points of I are 1 (foreground).
- (b) If the result in (a) gives the result that you would expect, explain why. If it does not, explain how you would modify the algorithm so that it works as expected.

9.40* As explained in Eqs. (9-44) and (9-69), opening by reconstruction preserves the shape of the image components that remain after erosion. What does closing by reconstruction do?

9.41 Show that geodesic erosion and dilation (Section 9.6) are duals with respect to set complementation. That is, assuming that the structuring element is symmetric about its origin, show that:

- (a)* $E_G^{(n)}(F) = \left[D_{G^c}^{(1)} \left[D_{G^c}^{(n-1)}(F^c) \right] \right]^c$ and, conversely, that
- (b) $D_G^{(n)}(F) = \left[E_{G^c}^{(1)} \left[E_{G^c}^{(n-1)}(F^c) \right] \right]^c$.

(Hint: Use proof by induction.)

9.42 Show that reconstruction by dilation and reconstruction by erosion (Section 9.6) are duals with respect to set complementation. That is, assuming that the structuring element is symmetric about its origin, show that $R_G^D(F) = \left[R_{G^c}^E(F^c) \right]^c$ and, conversely, that $R_G^E(F) = \left[R_{G^c}^D(F^c) \right]^c$. (Hint: Consider using the results from Problem 9.41.)

9.43 Show that:

- (a)* $(F \ominus nB)^c = F^c \oplus n\hat{B}$, where $F \ominus nB$ indicates n successive erosions, starting with B ; and similarly, that
- (b) $(F \oplus nB)^c = F^c \ominus n\hat{B}$.

9.44 Show the validity of the following binary morpho-

logical expressions. You may assume that the structuring element is symmetric about its origin.

(a)* $O_R^{(n)}(F) = \left[C_R^{(n)}(F^c) \right]^c$.

(b) $C_R^{(n)}(F) = \left[O_R^{(n)}(F^c) \right]^c$.

9.45 Prove the validity of the following grayscale morphological expressions. Recall from the discussion in Section 9.8 that $f^c(x, y) = -f(x, y)$ and that $\hat{b}(x, y) = b(-x, -y)$.

(a)* $(f \ominus b) = f^c \oplus b$.

(b) $(f \oplus b)^c = f^c \ominus \hat{b}$.

(c) $(f \bullet b)^c = f^c \circ \hat{b}$.

(d)* $(f \circ b)^c = f^c \bullet \hat{b}$.

9.46 Prove the validity of the following grayscale morphological expressions. Recall that $f^c(x, y) = -f(x, y)$ and that $\hat{b}(x, y) = b(-x, -y)$. (Hint: Use proof by induction.)

(a)* $D_g^{(n)}(f) = \left[E_{g^c}^{(1)} \left[E_{g^c}^{(n-1)}(f^c) \right] \right]^c$. Assume a symmetric structuring element.

(b) $E_g^{(n)}(f) = \left[D_{g^c}^{(1)} \left[D_{g^c}^{(n-1)}(f^c) \right] \right]^c$. Assume a symmetric structuring element.

9.47 Prove the validity of the following grayscale morphological expressions.

(a)* $R_g^D(f) = \left[R_{g^c}^E(f^c) \right]^c$.

(b) $R_g^E(f) = \left[R_{g^c}^D(f^c) \right]^c$.

9.48 Prove the validity of the following grayscale morphological expressions.

(a)* $(f \ominus nb)^c = (f^c \oplus n\hat{b})$, where $(f \ominus nb)$ indicates n successive erosions, starting with b .

(b) $(f \oplus nb)^c = (f^c \ominus n\hat{b})$.

9.49 Prove the validity of the following grayscale morphological expressions. Recall that $f^c(x, y) = -f(x, y)$ and that $\hat{b}(x, y) = b(-x, -y)$. Assume a symmetric structuring element.

(a)* $O_R^{(n)}(f) = \left[C_R^{(n)}(f^c) \right]^c$.

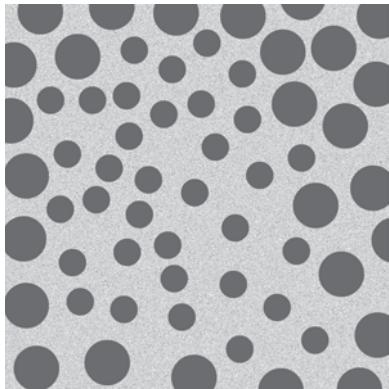
(b) $C_R^{(n)}(f) = \left[O_R^{(n)}(f^c) \right]^c$.

9.50 Consider the image below, which shows a region of small circles enclosed by a region of larger circles.

(a) Would you expect the method used to generate Fig. 9.45(d) to work with this image also? Explain your reasoning, including any

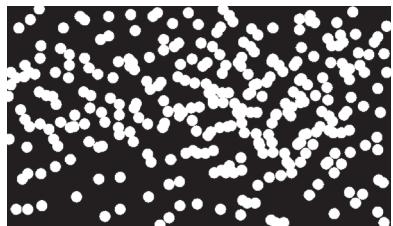
assumptions that you need to make for the method to work.

- (b)*** If your answer to (a) is yes, sketch what the boundary will look like.

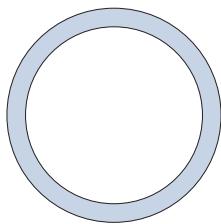


- 9.51** A preprocessing step in an application of microscopy is concerned with the issue of isolating individual round particles from similar particles that overlap in groups of two or more particles (see the following image). Assuming that all particles are of the same size, propose a morphological algorithm that produces three images consisting respectively of:

- (a)*** Only particles that have merged with the boundary of the image.
- (b)** Only overlapping particles.
- (c)** Only nonoverlapping particles.



- 9.52** A high-technology manufacturing plant is awarded a government contract to manufacture high-precision washers of the form shown. The terms of the contract require that the shape of all washers be inspected by an imaging system. In this context, shape inspection refers to deviations from round on the inner and outer edges of the washers. You may assume the following: (1) A “golden” (perfect with respect to the problem) image of an acceptable washer is available; and (2) the imaging and positioning components ultimately used in the system will have an accuracy high enough to allow you to ignore errors due to digitalization and positioning. You are hired as a consultant to help specify the visual inspection part of the system. Propose a solution based on morphological/logical operations.



This page intentionally left blank

10

Image Segmentation



The whole is equal to the sum of its parts.

Euclid

The whole is greater than the sum of its parts.

Max Wertheimer

Preview

The material in the previous chapter began 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. Most of the segmentation algorithms in this chapter are based on one of two basic properties of image intensity values: *discontinuity* and *similarity*. In the first category, the approach is to partition an image into regions based on abrupt changes in intensity, such as edges. Approaches in the second category are based on partitioning an image into regions that are similar according to a set of predefined criteria. Thresholding, region growing, and region splitting and merging are examples of methods in this category. We show that improvements in segmentation performance can be achieved by combining methods from distinct categories, such as techniques in which edge detection is combined with thresholding. We discuss also image segmentation using clustering and superpixels, and give an introduction to graph cuts, an approach ideally suited for extracting the principal regions of an image. This is followed by a discussion of image segmentation based on morphology, an approach that combines several of the attributes of segmentation based on the techniques presented in the first part of the chapter. We conclude the chapter with a brief discussion on the use of motion cues for segmentation.

Upon completion of this chapter, readers should:

- Understand the characteristics of various types of edges found in practice.
- Understand how to use spatial filtering for edge detection.
- Be familiar with other types of edge detection methods that go beyond spatial filtering.
- Understand image thresholding using several different approaches.
- Know how to combine thresholding and spatial filtering to improve segmentation.
- Be familiar with region-based segmentation, including clustering and superpixels.
- Understand how graph cuts and morphological watersheds are used for segmentation.
- Be familiar with basic techniques for utilizing motion in image segmentation.

10.1 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, \dots, R_n , such that

- (a) $\bigcup_{i=1}^n R_i = R$.
- (b) R_i is a connected set, for $i = 0, 1, 2, \dots, n$.
- (c) $R_i \cap R_j = \emptyset$ for all i and j , $i \neq j$.
- (d) $Q(R_i) = \text{TRUE}$ for $i = 0, 1, 2, \dots, n$.
- (e) $Q(R_i \cup R_j) = \text{FALSE}$ for any adjacent regions R_i and R_j .

where $Q(R_k)$ is a logical predicate defined over the points in set R_k , and \emptyset is the null set. The symbols \cup and \cap represent set union and intersection, respectively, as defined in Section 2.6. Two regions R_i and R_j are said to be *adjacent* if their union forms a connected set, as defined in Section 2.5. If the set formed by the union of two regions is not connected, the regions are said to be *disjoint*.

Condition (a) indicates that the segmentation must be *complete*, in the sense that 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 8-connected). Condition (c) says 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. 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, we assume 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.

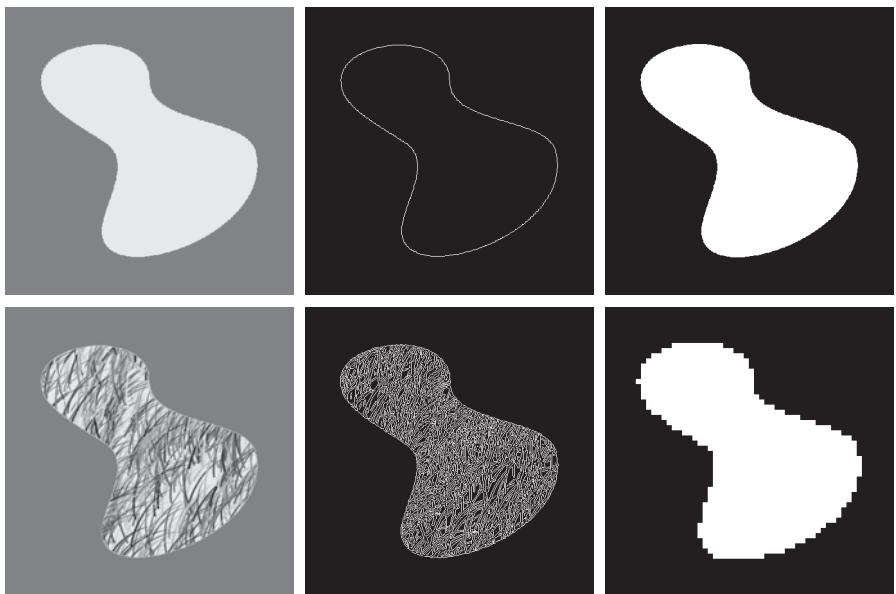
Figure 10.1 illustrates the preceding concepts. Figure 10.1(a) shows an image of a region of constant intensity superimposed on a darker background, also of constant intensity. These two regions comprise the overall image. Figure 10.1(b) shows the result of computing the boundary of the inner region 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 inside the boundary, and another level (e.g., black) to all points exterior to the boundary. Figure 10.1(c) shows the result of such a procedure. We see that conditions (a) through (c) stated at the beginning of this

[†] In general, Q can be a compound expression such as, “ $Q(R_i) = \text{TRUE}$ if the average intensity of the pixels in region 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.

a	b	c
d	e	f

FIGURE 10.1

- (a) Image of a constant intensity region.
- (b) Boundary based on intensity discontinuities.
- (c) Result of segmentation.
- (d) Image of a texture region.
- (e) Result of intensity discontinuity computations (note the large number of small edges).
- (f) Result of segmentation based on region properties.



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 or white in Fig. 10.1(c). Similarly, the two segmented regions (object and background) satisfy condition (e).

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 intensity discontinuities in this image. 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. However, we note that the outer region is constant, so all we need to solve this 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 8×8 . 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 8×8 squares were labeled with the same intensity (smaller squares would have given a smoother region boundary). Finally, note that these results also satisfy the five segmentation conditions stated at the beginning of this section.

10.2 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 characteristics in which

When we refer to lines, we are referring to thin structures, typically just a few pixels thick. Such lines may correspond, for example, to elements of a digitized architectural drawing, or roads in a satellite image.

we are interested are isolated points, lines, and edges. *Edge pixels* are pixels at which the intensity of an image changes abruptly, and *edges* (or *edge segments*) are sets of connected edge pixels (see Section 2.5 regarding connectivity). *Edge detectors* are local image processing tools designed to detect edge pixels. A *line* may be viewed as a (typically) thin 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 will discuss later, lines give rise to so-called “roof edges.” Finally, an *isolated point* may be viewed as a foreground (background) pixel surrounded by background (foreground) pixels.

BACKGROUND

As we saw in Section 3.5, local averaging smoothes an image. Given that averaging is analogous to integration, it is intuitive 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 *finite differences*. There are various ways to compute these differences but, as explained in Section 3.6, we require that any approximation used for first derivatives (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 second derivatives (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 an arbitrary point x of a one-dimensional function $f(x)$ by expanding the function $f(x + \Delta x)$ into a Taylor series about x

$$\begin{aligned} f(x + \Delta x) &= f(x) + \Delta x \frac{\partial f(x)}{\partial x} + \frac{(\Delta x)^2}{2!} \frac{\partial^2 f(x)}{\partial x^2} + \frac{(\Delta x)^3}{3!} \frac{\partial^3 f(x)}{\partial x^3} + \dots \\ &= \sum_{n=0}^{\infty} \frac{(\Delta x)^n}{n!} \frac{\partial^n f(x)}{\partial x^n} \end{aligned} \quad (10-1)$$

Remember, the notation $n!$ means “ n factorial”: $n! = 1 \times 2 \times \dots \times n$.

where Δx is the separation between samples of f . For our purposes, this separation is measured in pixel units. Thus, following the convention in the book, $\Delta x = 1$ for the sample preceding x and $\Delta x = -1$ for the sample following x . When $\Delta x = 1$, Eq. (10-1) becomes

$$\begin{aligned} f(x+1) &= f(x) + \frac{\partial f(x)}{\partial x} + \frac{1}{2!} \frac{\partial^2 f(x)}{\partial x^2} + \frac{1}{3!} \frac{\partial^3 f(x)}{\partial x^3} + \dots \\ &= \sum_{n=0}^{\infty} \frac{1}{n!} \frac{\partial^n f(x)}{\partial x^n} \end{aligned} \quad (10-2)$$

Although this is an expression of only one variable, we used partial derivatives notation for consistency when we discuss functions of two variables later in this section.

Similarly, when $\Delta x = -1$,

$$\begin{aligned} f(x-1) &= f(x) - \frac{\partial f(x)}{\partial x} + \frac{1}{2!} \frac{\partial^2 f(x)}{\partial x^2} - \frac{1}{3!} \frac{\partial^3 f(x)}{\partial x^3} + \dots \\ &= \sum_{n=0}^{\infty} \frac{(-1)^n}{n!} \frac{\partial^n f(x)}{\partial x^n} \end{aligned} \quad (10-3)$$

In what follows, we compute *intensity differences* using just a few terms of the Taylor series. For first-order derivatives we use only the linear terms, and we can form differences in one of three ways.

The *forward difference* is obtained from Eq. (10-2):

$$\frac{\partial f(x)}{\partial x} = f'(x) = f(x+1) - f(x) \quad (10-4)$$

where, as you can see, we kept only the linear terms. The *backward difference* is similarly obtained by keeping only the linear terms in Eq. (10-3):

$$\frac{\partial f(x)}{\partial x} = f'(x) = f(x) - f(x-1) \quad (10-5)$$

and the *central difference* is obtained by subtracting Eq. (10-3) from Eq. (10-2):

$$\frac{\partial f(x)}{\partial x} = f'(x) = \frac{f(x+1) - f(x-1)}{2} \quad (10-6)$$

The higher terms of the series that we did not use represent the error between an exact and an approximate derivative expansion. In general, the more terms we use from the Taylor series to represent a derivative, the more accurate the approximation will be. To include more terms implies that more points are used in the approximation, yielding a lower error. However, it turns out that central differences have a lower error for the same number of points (see Problem 10.1). For this reason, derivatives are usually expressed as central differences.

The *second order* derivative based on a central difference, $\partial^2 f(x)/\partial x^2$, is obtained by adding Eqs. (10-2) and (10-3):

$$\frac{\partial^2 f(x)}{\partial x^2} = f''(x) = f(x+1) - 2f(x) + f(x-1) \quad (10-7)$$

To obtain the *third order, central derivative* we need one more point on either side of x . That is, we need the Taylor expansions for $f(x+2)$ and $f(x-2)$, which we obtain from Eqs. (10-2) and (10-3) with $\Delta x = 2$ and $\Delta x = -2$, respectively. The strategy is to combine the two Taylor expansions to eliminate all derivatives lower than the third. The result after ignoring all higher-order terms [see Problem 10.2(a)] is

$$\frac{\partial^3 f(x)}{\partial x^3} = f'''(x) = \frac{f(x+2) - 2f(x+1) + 0f(x) + 2f(x-1) - f(x-2)}{2} \quad (10-8)$$

Similarly [see Problem 10.2(b)], the *fourth* finite difference (the highest we use in the book) after ignoring all higher order terms is given by

$$\frac{\partial^4 f(x)}{\partial x^4} = f''''(x) = f(x+2) - 4f(x+1) + 6f(x) - 4f(x-1) + f(x-2) \quad (10-9)$$

Table 10.1 summarizes the first four central derivatives just discussed. Note the symmetry of the coefficients about the center point. This symmetry is at the root of why central differences have a lower approximation error for the same number of points than the other two differences. For two variables, we apply the results in Table 10.1 to each variable independently. For example,

$$\frac{\partial^2 f(x, y)}{\partial x^2} = f(x+1, y) - 2f(x, y) + f(x-1, y) \quad (10-10)$$

and

$$\frac{\partial^2 f(x, y)}{\partial y^2} = f(x, y+1) - 2f(x, y) + f(x, y-1) \quad (10-11)$$

It is easily verified that the first and second-order derivatives in Eqs. (10-4) through (10-7) satisfy the conditions stated at the beginning of this section regarding derivatives of the first and second order. To illustrate this, consider Fig. 10.2. Part (a) shows an image of various objects, a line, and an isolated point. Figure 10.2(b) shows a horizontal intensity profile (scan line) through the center of the image, 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 will discuss later, intensity transitions involving thin objects such as lines often are referred to as *roof edges*.

Figure 10.2(c) shows a simplified profile, with just enough points to make it possible for us to analyze manually how the first- and second-order derivatives behave as they encounter a point, a line, and the edges of objects. In this diagram the transition

TABLE 10.1

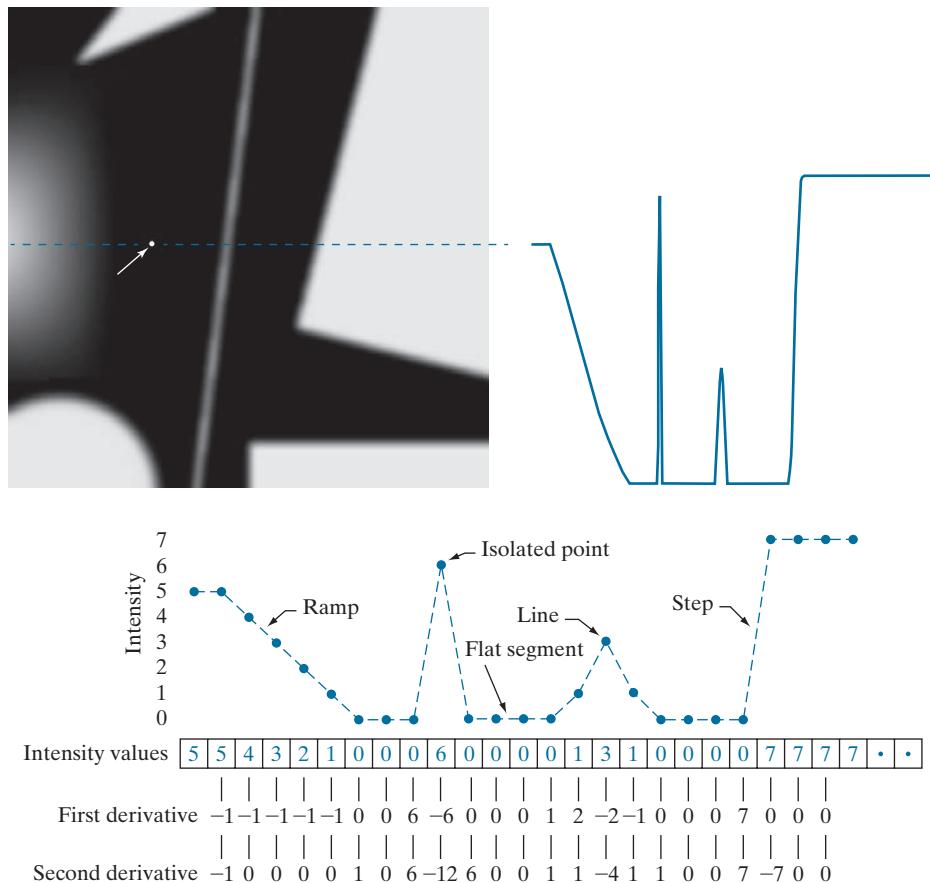
First four central digital derivatives (finite differences) for samples taken uniformly, $\Delta x = 1$ units apart.

	$f(x+2)$	$f(x+1)$	$f(x)$	$f(x-1)$	$f(x-2)$
$2f'(x)$		1	0	-1	
$f''(x)$		1	-2	1	
$2f'''(x)$	1	-2	0	2	-1
$f''''(x)$	1	-4	6	-4	1

a b
c

FIGURE 10.2

- (a) Image.
 - (b) Horizontal intensity profile that includes the isolated point indicated by the arrow.
 - (c) Subsampled profile; the dashes were added for clarity. The numbers in the boxes are the intensity values of the dots shown in the profile. The derivatives were obtained using Eqs. (10-4) for the first derivative and Eq. (10-7) for the second.



in the ramp spans four pixels, the noise point is a single pixel, the line is three pixels thick, and the transition of the step edge 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, 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 the edges of digital images resemble this type of transition, we conclude that first-order derivatives produce “thick” edges, and second-order derivatives much thinner 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

FIGURE 10.3

A general 3×3 spatial filter kernel. The w 's are the kernel coefficients (weights).

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

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, as we will show later in this section. As we move into the edge, 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)

In summary, we arrive at the following conclusions: (1) First-order derivatives generally produce thicker edges. (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 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 convolution. For the 3×3 filter kernel in Fig. 10.3, the procedure is to compute the sum of products of the kernel coefficients with the intensity values in the region encompassed by the kernel, as we explained in Section 3.4. That is, the response of the filter at the center point of the kernel is

This equation is an expansion of Eq. (3-35) for a 3×3 kernel, valid at one point, and using simplified subscript notation for the kernel coefficients.

$$\begin{aligned} Z &= w_1 z_1 + w_2 z_2 + \dots + w_9 z_9 \\ &= \sum_{k=1}^9 w_k z_k \end{aligned} \quad (10-12)$$

where z_k is the intensity of the pixel whose spatial location corresponds to the location of the k th kernel coefficient.

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, from the discussion in Section 3.6, means using the Laplacian:

$$\nabla^2 f(x, y) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \quad (10-13)$$

where the partial derivatives are computed using the second-order finite differences in Eqs. (10-10) and (10-11). The Laplacian is then

$$\nabla^2 f(x, y) = f(x+1, y) + f(x-1, y) + f(x, y+1) + f(x, y-1) - 4f(x, y) \quad (10-14)$$

As explained in Section 3.6, this expression can be implemented using the Laplacian kernel in Fig. 10.4(a) in Example 10.1. We then say that a point has been detected at a location (x, y) on which the kernel is centered if the absolute value of the response of the filter at that point exceeds a specified threshold. Such points are labeled 1 and all others are labeled 0 in the output image, thus producing a binary image. In other words, we use the expression:

$$g(x, y) = \begin{cases} 1 & \text{if } |Z(x, y)| > T \\ 0 & \text{otherwise} \end{cases} \quad (10-15)$$

where $g(x, y)$ is the output image, T is a nonnegative threshold, and Z is given by Eq. (10-12). 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 kernel. 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 kernel, the coefficients sum to zero, indicating that the filter response will be zero in areas of constant intensity.

EXAMPLE 10.1: Detection of isolated points in an image.

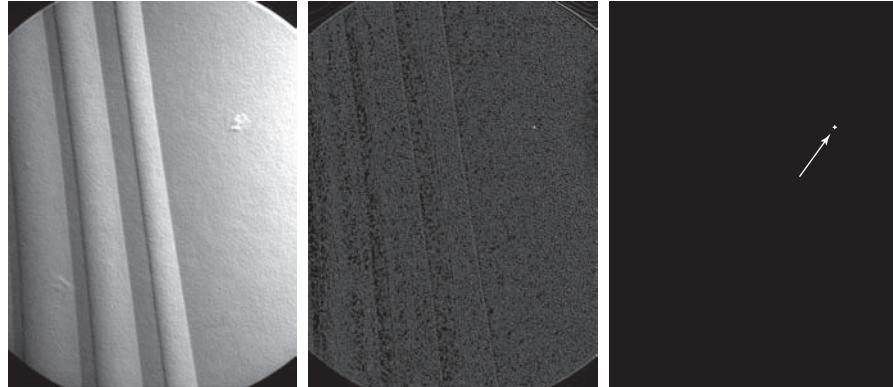
Figure 10.4(b) is an X-ray image of a turbine blade from a jet engine. The blade has a porosity manifested by a single black pixel in the upper-right quadrant of the image. Figure 10.4(c) is the result of filtering the image with the Laplacian kernel, and Fig. 10.4(d) shows the result of Eq. (10-15) with T equal to 90% of the highest absolute pixel value of the image in Fig. 10.4(c). The single pixel is clearly visible in this image at the tip of the arrow (the pixel was enlarged to enhance its visibility). This type of detection process is 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 kernel. When this condition is not satisfied, other methods discussed in this chapter are more suitable for detecting intensity changes.

LINE DETECTION

The next level of complexity is line detection. Based on the discussion earlier in this section, we know that for line detection we can expect second derivatives to result in a stronger filter response, and to produce thinner lines than first derivatives. Thus, we can use the Laplacian kernel in Fig. 10.4(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.

**FIGURE 10.4**

- (a) Laplacian kernel used for point detection.
- (b) X-ray image of a turbine blade with a porosity manifested by a single black pixel.
- (c) Result of convolving the kernel with the image.
- (d) Result of using Eq. (10-15) was a single point (shown enlarged at the tip of the arrow). (Original image courtesy of X-TEK Systems, Ltd.)



EXAMPLE 10.2: Using the Laplacian for line detection.

Figure 10.5(a) shows a 486×486 (binary) portion of a wire-bond mask for an electronic circuit, and Fig. 10.5(b) shows its Laplacian image. Because the Laplacian image contains negative values (see the discussion after Example 3.18), 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.

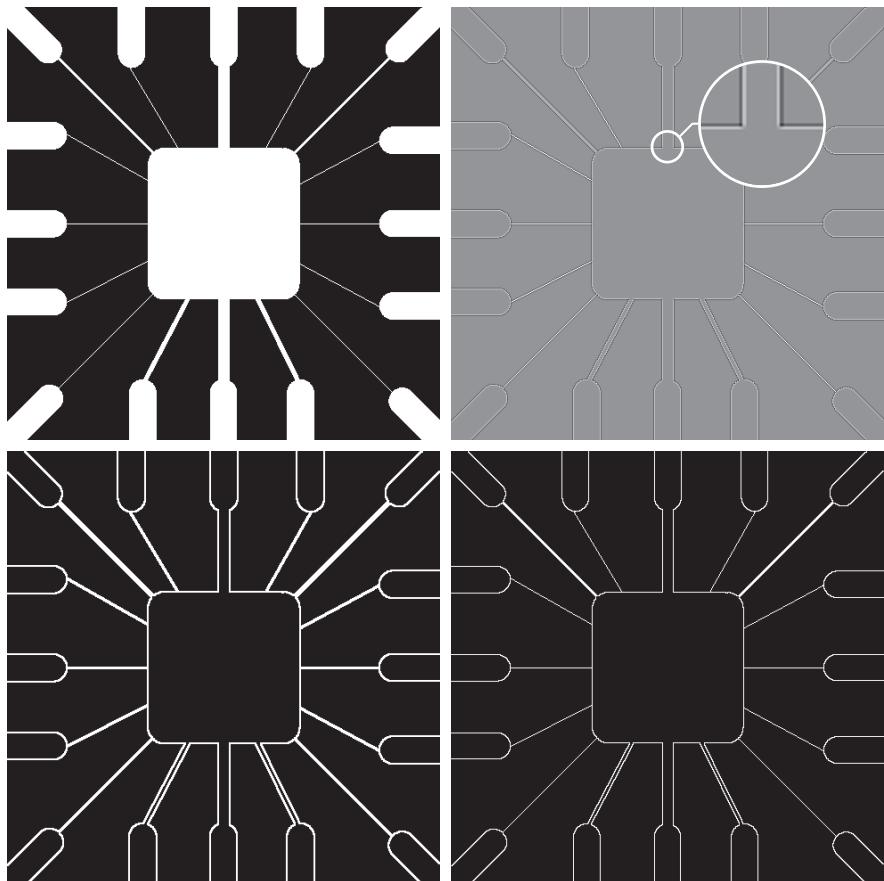
At first, it might appear that the negative values can be handled simply by taking the absolute value of the Laplacian image. However, as Fig. 10.5(c) shows, this approach doubles the thickness of the lines. A more suitable approach is to use only the positive values of the Laplacian (in noisy situations we use the values that exceed a positive threshold to eliminate random variations about zero caused by the noise). As Fig. 10.5(d) shows, this approach results in thinner lines that generally are more useful. Note in Figs. 10.5(b) through (d) that when the lines are wide with respect to the size of the Laplacian kernel, the lines are separated by a zero “valley.” This is not unexpected. For example, when the 3×3 kernel 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 in the following section.

The Laplacian detector kernel in Fig. 10.4(a) is isotropic, so its response is independent of direction (with respect to the four directions of the 3×3 kernel: vertical, horizontal, and two diagonals). Often, interest lies in detecting lines in *specified*

a
b
c
d

FIGURE 10.5

- (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.



directions. Consider the kernels in Fig. 10.6. Suppose that an image with a constant background and containing various lines (oriented at 0° , $\pm 45^\circ$, and 90°) is filtered with the first kernel. The maximum responses would occur at image locations in which a horizontal line passes through the middle row of the kernel. This is easily verified by sketching a simple array of 1's with a line of a different intensity (say, 5s) running horizontally through the array. A similar experiment would reveal that the second kernel in Fig. 10.6 responds best to lines oriented at $+45^\circ$; the third kernel to vertical lines; and the fourth kernel to lines in the -45° direction. The preferred direction of each kernel is weighted with a larger coefficient (i.e., 2) than other possible directions. The coefficients in each kernel sum to zero, indicating a zero response in areas of constant intensity.

Let Z_1, Z_2, Z_3 , and Z_4 denote the responses of the kernels in Fig. 10.6, from left to right, where the Z s are given by Eq. (10-12). Suppose that an image is filtered with these four kernels, one at a time. If, at a given point in the image, $|Z_k| > |Z_j|$, for all $j \neq k$, that point is said to be more likely associated with a line in the direction of kernel k . For example, if at a point in the image, $|Z_1| > |Z_j|$ for $j = 2, 3, 4$, that

-1	-1	-1	2	-1	-1	-1	2	-1	-1	-1	2
2	2	2	-1	2	-1	-1	2	-1	-1	2	-1
-1	-1	-1	-1	-1	2	-1	2	-1	2	-1	-1

Horizontal

+45°

Vertical

-45°

a b c d

FIGURE 10.6 Line detection kernels. Detection angles are with respect to the axis system in Fig. 2.19, with positive angles measured counterclockwise with respect to the (vertical) x -axis.

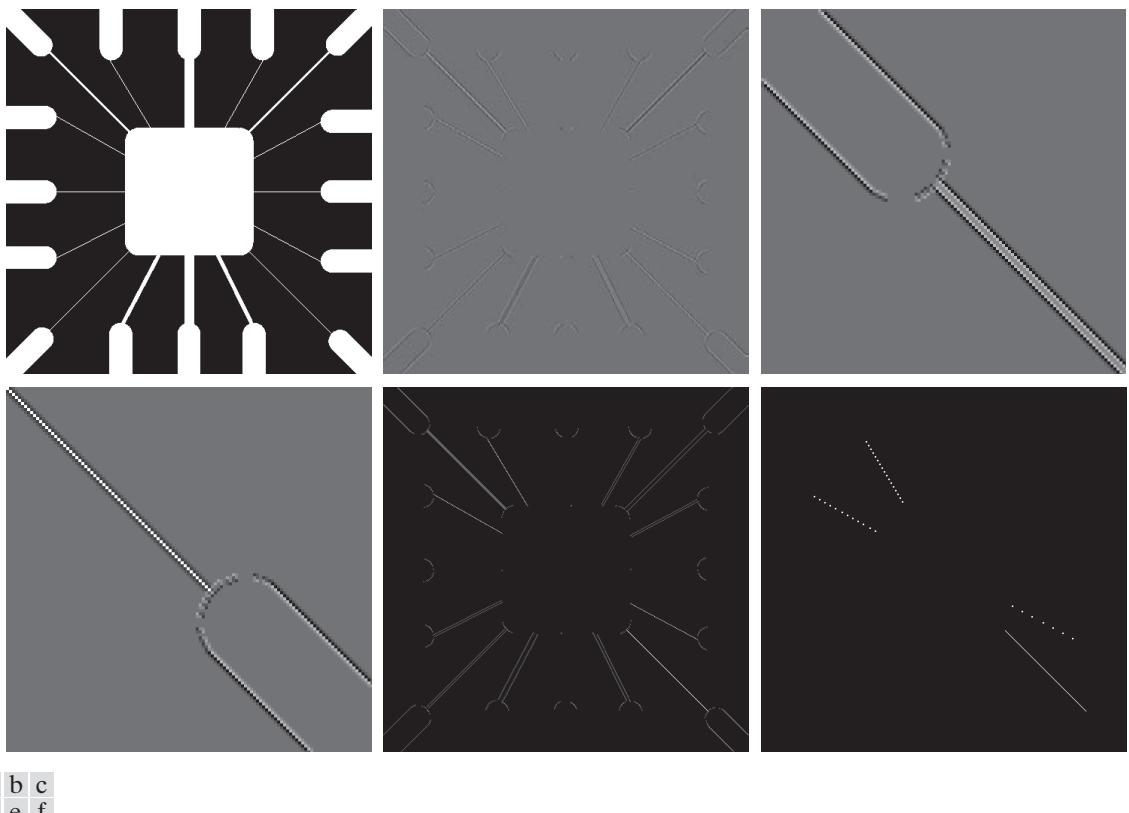
point is said to be more likely associated with a horizontal line. If we are interested in detecting all the lines in an image in the direction defined by a given kernel, we simply run the kernel through the image and threshold the absolute value of the result, as in Eq. (10-15). The nonzero points remaining after thresholding are the strongest responses which, for lines one pixel thick, correspond closest to the direction defined by the kernel. The following example illustrates this procedure.

EXAMPLE 10.3: Detecting lines in specified directions.

Figure 10.7(a) shows the image used in the previous example. Suppose that we are interested in finding all the lines that are one pixel thick and oriented at +45°. For this purpose, we use the kernel in Fig. 10.6(b). Figure 10.7(b) is the result of filtering the image with that kernel. As before, the shades darker than the gray background in Fig. 10.7(b) correspond to negative values. There are two principal segments in the image oriented in the +45° direction, one in the top left and one at the bottom right. Figures 10.7(c) and (d) show zoomed sections of Fig. 10.7(b) corresponding to these two areas. The straight line segment in Fig. 10.7(d) is brighter than the segment in Fig. 10.7(c) because the line segment in the bottom right of Fig. 10.7(a) is one pixel thick, while the one at the top left is not. The kernel is “tuned” to detect one-pixel-thick lines in the +45° direction, so we expect its response to be stronger when such lines are detected. Figure 10.7(e) shows the positive values of Fig. 10.7(b). Because we are interested in the strongest response, we let T equal 254 (the maximum value in Fig. 10.7(e) minus one). Figure 10.7(f) shows in white the points whose values satisfied the condition $g > T$, where g is the image in Fig. 10.7(e). The isolated points in the figure are points that also had similarly strong responses to the kernel. In the original image, these points and their immediate neighbors are oriented in such a way that the kernel produced a maximum response at those locations. These isolated points can be detected using the kernel in Fig. 10.4(a) and then deleted, or they can be deleted using morphological operators, as discussed in the last chapter.

EDGE MODELS

Edge detection is an approach used 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.

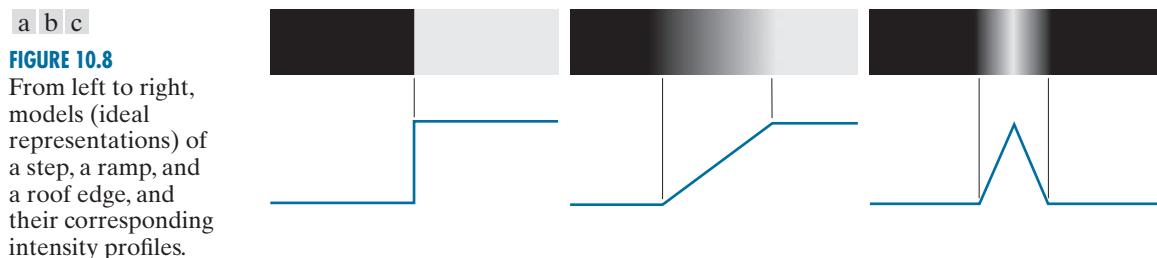


a b c
d e f

FIGURE 10.7 (a) Image of a wire-bond template. (b) Result of processing with the $+45^\circ$ line detector kernel in Fig. 10.6. (c) Zoomed view of the top left region of (b). (d) Zoomed view of the bottom right region of (b). (e) The image in (b) with all negative values set to zero. (f) All points (in white) whose values satisfied the condition $g > T$, where g is the image in (e) and $T = 254$ (the maximum pixel value in the image minus 1). (The points in (f) were enlarged to make them easier to see.)

Edge models are classified according to their intensity profiles. A *step edge* is characterized by a transition between two intensity levels occurring ideally over the distance of one pixel. Figure 10.8(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 one 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. For example, the Canny edge detection algorithm discussed later in this section was derived originally using a step-edge model.

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. 10.8(b). The slope of the ramp is inversely proportional to the degree to which the edge is blurred. In this model, we no longer have a single “edge point” along the profile. 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.

A third type of edge is the so-called *roof edge*, having the characteristics illustrated in Fig. 10.8(c). Roof edges are models of lines through a region, with the base (width) of the edge being determined by the thickness and sharpness of the line. In the limit, when its base is one pixel wide, a roof edge is nothing more than a one-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 the background (such as walls). The pipes appear brighter and thus create an image similar to the model in Fig. 10.8(c). 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.

It is not unusual to find images that contain all three types of edges. Although blurring and noise result in deviations from the ideal shapes, edges in images that are reasonably sharp and have a moderate amount of noise do resemble the characteristics of the edge models in Fig. 10.8, as the profiles in Fig. 10.9 illustrate. What the models in Fig. 10.8 allow us to do is write mathematical expressions for edges in the development of image processing algorithms. The performance of these algorithms will depend on the differences between actual edges and the models used in developing the algorithms.

Figure 10.10(a) shows the image from which the segment in Fig. 10.8(b) was extracted. Figure 10.10(b) shows a horizontal intensity profile. This figure shows also the first and second derivatives of the intensity profile. 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

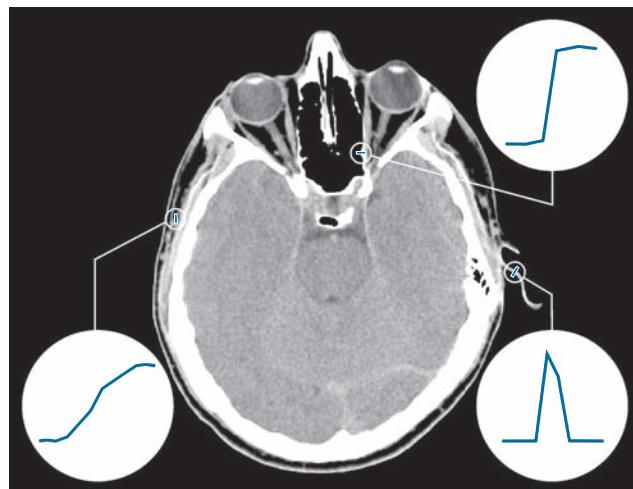


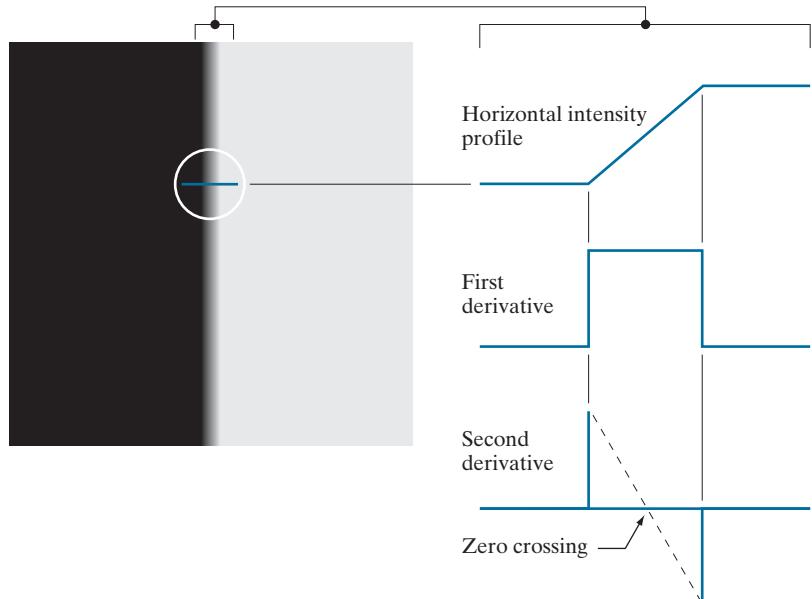
FIGURE 10.9 A 1508×1970 image showing (zoomed) actual ramp (bottom, left), step (top, right), and roof edge profiles. The profiles are from dark to light, in the areas enclosed by the small circles. The ramp and step profiles span 9 pixels and 2 pixels, respectively. The base of the roof edge is 3 pixels. (Original image courtesy of Dr. David R. Pickens, Vanderbilt University.)

the dark or light side of an edge. Two additional properties of the second derivative around an edge are: (1) it produces two values for every edge in an image; and (2) its zero crossings can be used for locating the centers of thick edges, as we will show later in this section. Some edge models utilize a smooth transition into and out of

a b

FIGURE 10.10

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



the ramp (see Problem 10.9). 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 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.

EXAMPLE 10.4: Behavior of the first and second derivatives in the region of a noisy edge.

The edge models in Fig. 10.8 are free of noise. The image segments in the first column in Fig. 10.11 show close-ups of four ramp edges that transition from a black region on the left to a white region on the right (keep in mind that the entire transition from black to white is a single edge). The image segment at the top left is free of noise. The other three images in the first column are corrupted by additive Gaussian noise with zero mean and standard deviation of 0.1, 1.0, and 10.0 intensity levels, respectively. The graph below each image is a horizontal intensity profile passing through the center of the image. All images have 8 bits of intensity resolution, with 0 and 255 representing black and white, respectively.

Consider the image at the top of the center column. As discussed in connection with Fig. 10.10(b), the derivative of the scan line on the left is zero in the constant areas. These are the two black bands shown in the derivative image. The derivatives at points on the ramp are constant and equal to the slope of the ramp. These constant values in the derivative image are shown in gray. As we move down the center column, the derivatives become increasingly different from the noiseless case. In fact, it would be difficult to associate the last profile in the center column with the first derivative of a ramp edge. What makes these results interesting is that the noise is almost visually undetectable in the images on the left column. These examples are good illustrations of the sensitivity of derivatives to noise.

As expected, the second derivative is even more sensitive to noise. The second derivative of the noiseless image is shown at the top of the right column. The thin white and black vertical lines are the positive and negative components of the second derivative, as explained in Fig. 10.10. The gray in these images represents zero (as discussed earlier, scaling causes zero to show as gray). The only noisy second derivative image that barely resembles the noiseless case corresponds to noise with a standard deviation of 0.1. The remaining second-derivative images and profiles clearly illustrate that it would be difficult indeed to detect their positive and negative components, which are the truly useful features of the second derivative in terms of edge detection.

The fact that such little visual noise can have such a significant impact on the two key derivatives used for detecting edges is an important issue to keep in mind. In particular, image smoothing should be a serious consideration prior to the use of derivatives in applications where noise with levels similar to those we have just discussed is likely to be present.

In summary, the three steps performed typically for edge detection are:

1. *Image smoothing for noise reduction.* The need for this step is illustrated by the results in the second and third columns of Fig. 10.11.
2. *Detection of edge points.* As mentioned earlier, this is a local operation that extracts from an image all points that are potential edge-point candidates.
3. *Edge localization.* The objective of this step is to select from the candidate points only the points that are members of the set of points comprising an edge.

The remainder of this section deals with techniques for achieving these objectives.

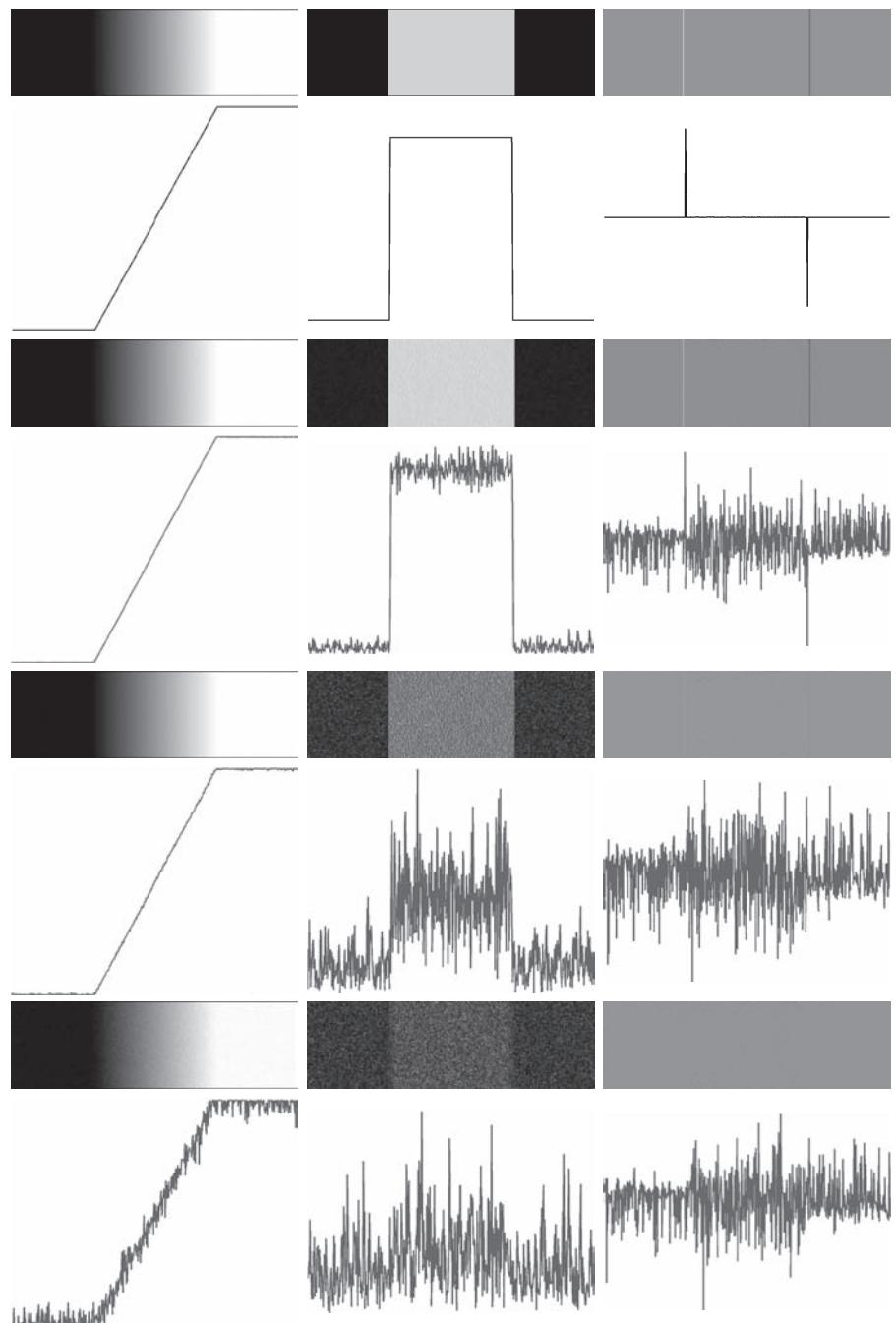


FIGURE 10.11 First column: 8-bit images with values in the range [0,255], and intensity profiles of a ramp edge corrupted by Gaussian noise of zero mean and standard deviations of 0.0, 0.1, 1.0, and 10.0 intensity levels, respectively. Second column: First-derivative images and intensity profiles. Third column: Second-derivative images and intensity profiles.

BASIC EDGE DETECTION

As illustrated in the preceding discussion, detecting changes in intensity for the purpose of finding edges can be accomplished using first- or second-order derivatives. We begin with first-order derivatives, and work with second-order derivatives in the following subsection.

The Image Gradient and Its Properties

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

For convenience, we repeat here some of the gradient concepts and equations introduced in Chapter 3.

$$\nabla f(x, y) \equiv \text{grad}[f(x, y)] \equiv \begin{bmatrix} g_x(x, y) \\ g_y(x, y) \end{bmatrix} = \begin{bmatrix} \frac{\partial f(x, y)}{\partial x} \\ \frac{\partial f(x, y)}{\partial y} \end{bmatrix} \quad (10-16)$$

This vector has the well-known property that it points in the direction of maximum rate of change of f at (x, y) (see Problem 10.10). Equation (10-16) is valid at an arbitrary (but *single*) point (x, y) . When evaluated for all applicable values of x and y , $\nabla f(x, y)$ becomes a *vector image*, each element of which is a vector given by Eq. (10-16). The *magnitude*, $M(x, y)$, of this gradient vector at a point (x, y) is given by its Euclidean vector norm:

$$M(x, y) = \|\nabla f(x, y)\| = \sqrt{g_x^2(x, y) + g_y^2(x, y)} \quad (10-17)$$

This is the *value* of the rate of change in the direction of the gradient vector at point (x, y) . Note that $M(x, y)$, $\|\nabla f(x, y)\|$, $g_x(x, y)$, and $g_y(x, y)$ are arrays of the same size as f , created when x and y are allowed to vary over all pixel locations in f . It is common practice to refer to $M(x, y)$ and $\|\nabla f(x, y)\|$ as the *gradient image*, or simply as the *gradient* when the meaning is clear. The summation, square, and square root operations are elementwise operations, as defined in Section 2.6.

The *direction* of the gradient vector at a point (x, y) is given by

$$\alpha(x, y) = \tan^{-1} \left[\frac{g_y(x, y)}{g_x(x, y)} \right] \quad (10-18)$$

Angles are measured in the counterclockwise direction with respect to the x -axis (see Fig. 2.19). This is also an image of the same size as f , created by the elementwise division of g_x and g_y over all applicable values of x and y . The following example illustrates, the direction of an edge at a point (x, y) is orthogonal to the direction, $\alpha(x, y)$, of the gradient vector at the point.

EXAMPLE 10.5: Computing the gradient.

Figure 10.12(a) shows a zoomed section of an image containing a straight edge segment. Each square 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 shaded pixels in this figure are assumed to have value 0, and the white

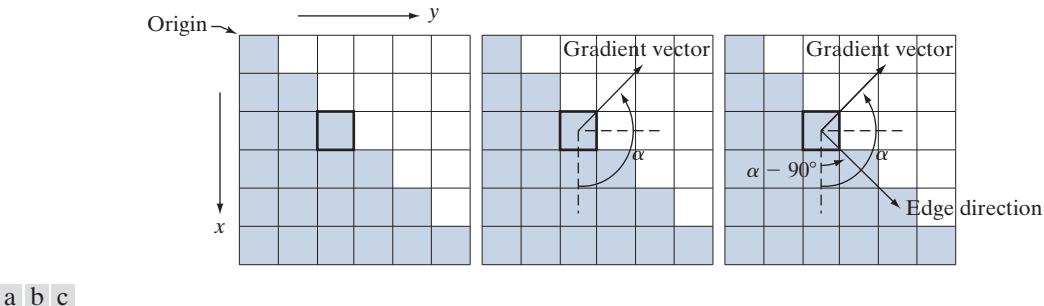


FIGURE 10.12 Using the gradient to determine edge strength and direction at a point. Note that the edge direction is perpendicular to the direction of the gradient vector at the point where the gradient is computed. Each square represents one pixel. (Recall from Fig. 2.19 that the origin of our coordinate system is at the top, left.)

pixels have value 1. We discuss after this example an approach for computing the derivatives in the x - and y -directions using a 3×3 neighborhood centered at a point. The method consists 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 of the neighborhood 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 $\|\nabla f\| = 2\sqrt{2}$ at that point. Similarly, the direction of the gradient vector at the same point follows from Eq. (10-18): $\alpha = \tan^{-1}(g_y/g_x) = -45^\circ$, which is the same as 135° measured in the positive (counterclockwise) direction with respect to the x -axis in our image coordinate system (see Fig. 2.19). Figure 10.12(b) shows the gradient vector and its direction angle.

As mentioned earlier, the direction of an edge at a point is orthogonal to the gradient vector at that point. So the direction angle of the edge in this example is $\alpha - 90^\circ = 135^\circ - 90^\circ = 45^\circ$, as Fig. 10.12(c) shows. All edge points in Fig. 10.12(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 referred to as the *edge unit normal*.

Gradient Operators

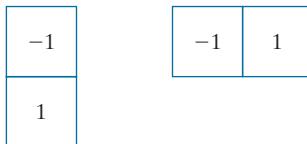
Obtaining the gradient of an image requires computing the partial derivatives $\partial f / \partial x$ and $\partial f / \partial y$ at every pixel location in the image. For the gradient, we typically use a forward or centered finite difference (see Table 10.1). Using forward differences we obtain

$$g_x(x, y) = \frac{\partial f(x, y)}{\partial x} = f(x+1, y) - f(x, y) \quad (10-19)$$

a b

FIGURE 10.13

1-D kernels used to implement Eqs. (10-19) and (10-20).



and

$$g_y(x, y) = \frac{\partial f(x, y)}{\partial y} = f(x, y + 1) - f(x, y) \quad (10-20)$$

These two equations can be implemented for all values of x and y by filtering $f(x, y)$ with the 1-D kernels in Fig. 10.13.

When diagonal edge direction is of interest, we need 2-D kernels. The *Roberts cross-gradient operators* (Roberts [1965]) are one of the earliest attempts to use 2-D kernels with a diagonal preference. Consider the 3×3 region in Fig. 10.14(a). The Roberts operators are based on implementing the diagonal differences

$$g_x = \frac{\partial f}{\partial x} = (z_9 - z_5) \quad (10-21)$$

and

$$g_y = \frac{\partial f}{\partial y} = (z_8 - z_6) \quad (10-22)$$

These derivatives can be implemented by filtering an image with the kernels shown in Figs. 10.14(b) and (c).

Kernels of size 2×2 are simple conceptually, but they are not as useful for computing edge direction as kernels that are symmetric about their centers, the smallest of which are of size 3×3 . These kernels 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 kernels of size 3×3 are given by

$$g_x = \frac{\partial f}{\partial x} = (z_7 + z_8 + z_9) - (z_1 + z_2 + z_3) \quad (10-23)$$

and

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

In this formulation, 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 derivative in the y -direction. Intuitively, we would expect these approximations to be more accurate than the approximations obtained using the Roberts operators. Equations (10-22) and (10-23) can be implemented over an entire image by filtering it with the two kernels in Figs. 10.14(d) and (e). These kernels are called the *Prewitt operators* (Prewitt [1970]).

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

Filter kernels used to compute the derivatives needed for the gradient are often called *gradient operators*, *difference operators*, *edge operators*, or *edge detectors*.

Observe that these two equations are first-order central differences as given in Eq. (10-6), but multiplied by 2.

a
b c
d e
f g

FIGURE 10.14

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

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

$$g_x = \frac{\partial f}{\partial x} = (z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3) \quad (10-24)$$

and

$$g_y = \frac{\partial f}{\partial y} = (z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7) \quad (10-25)$$

It can be demonstrated (see Problem 10.12) that using a 2 in the center location provides image smoothing. Figures 10.14(f) and (g) show the kernels used to implement Eqs. (10-24) and (10-25). These kernels are called the *Sobel operators* (Sobel [1970]).

The Prewitt kernels are simpler to implement than the Sobel kernels, but the slight computational difference between them typically is not an issue. The fact that the Sobel kernels have better noise-suppression (smoothing) characteristics makes them preferable because, as mentioned earlier in the discussion of Fig. 10.11, noise suppression is an important issue when dealing with derivatives. Note that the

Recall the important result in Problem 3.32 that using a kernel whose coefficients sum to zero produces a filtered image whose pixels also sum to zero. This implies in general that some pixels will be negative. Similarly, if the kernel coefficients sum to 1, the sum of pixels in the original and filtered images will be the same (see Problem 3.31).

coefficients of all the kernels in Fig. 10.14 sum to zero, thus giving a response of zero in areas of constant intensity, as expected of derivative operators.

Any of the pairs of kernels from Fig. 10.14 are convolved with an image to obtain the gradient components g_x and g_y at every pixel location. These two partial derivative arrays are then used to estimate edge strength and direction. Obtaining the magnitude of the gradient requires the computations in Eq. (10-17). This implementation is not always desirable because of the computational burden required by squares and square roots, and an approach used frequently is to approximate the magnitude of the gradient by absolute values:

$$M(x, y) \approx |g_x| + |g_y| \quad (10-26)$$

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 kernels such as the Prewitt and Sobel kernels are used to compute g_x and g_y because these kernels give isotropic results only for vertical and horizontal edges. This means that results would be isotropic only for edges in those two directions anyway, regardless of which of the two equations is used. That is, Eqs. (10-17) and (10-26) give identical results for vertical and horizontal edges when either the Sobel or Prewitt kernels are used (see Problem 10.11).

The 3×3 kernels in Fig. 10.14 exhibit their strongest response predominantly for vertical and horizontal edges. The *Kirsch compass kernels* (Kirsch [1971]) in Fig. 10.15, are designed to detect edge magnitude *and* direction (angle) in all eight compass directions. Instead of computing the magnitude using Eq. (10-17) and angle using Eq. (10-18), Kirsch's approach was to determine the edge magnitude by convolving an image with all eight kernels and assign the edge magnitude at a point as the response of the kernel that gave strongest convolution value at that point. The edge angle at that point is then the direction associated with that kernel. For example, if the strongest value at a point in the image resulted from using the north (N) kernel, the edge magnitude at that point would be assigned the response of that kernel, and the direction would be 0° (because compass kernel pairs differ by a rotation of 180° ; choosing the maximum response will always result in a positive number). Although when working with, say, the Sobel kernels, we think of a north or south edge as being vertical, the N and S compass kernels differentiate between the two, the difference being the direction of the intensity transitions defining the edge. For example, assuming that intensity values are in the range $[0, 1]$, the binary edge in Fig. 10.8(a) is defined by black (0) on the left and white (1) on the right. When all Kirsch kernels are applied to this edge, the N kernel will yield the highest value, thus indicating an edge oriented in the north direction (at the point of the computation).

EXAMPLE 10.6: Illustration of the 2-D gradient magnitude and angle.

Figure 10.16 illustrates the Sobel 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. 10.16(b) and (c). Note, for

a	b	c	d
e	f	g	h

FIGURE 10.15

Kirsch compass kernels. The edge direction of strongest response of each kernel is labeled below it.

-3	-3	5	-3	5	5	5	5	-3
-3	0	5	-3	0	5	-3	0	-3
-3	-3	5	-3	-3	-3	-3	-3	-3
N			NW			W		
5	-3	-3	-3	-3	-3	-3	-3	-3
5	0	-3	5	0	-3	-3	0	5
5	-3	-3	5	5	-3	5	5	5
S			SE			E		
NE								

example, how strong the roof tile, horizontal brick joints, and horizontal segments of the windows are in Fig. 10.16(b) compared to other edges. In contrast, Fig. 10.16(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. 10.16(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.

a	b
c	d

FIGURE 10.16

- (a) 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 kernel in Fig. 10.14(f) to filter the image.
- (c) $|g_y|$, obtained using the kernel in Fig. 10.14(g).
- (d) The gradient image, $|g_x| + |g_y|$.



FIGURE 10.17

Gradient angle image computed using Eq. (10-18). 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.

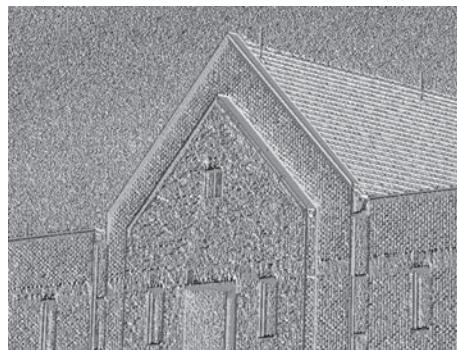


Figure 10.17 shows the gradient angle image computed using Eq. (10-18). In general, angle images are not as useful as gradient magnitude images for edge detection, but they do complement the information extracted from an image using the magnitude of the gradient. For instance, the constant intensity areas in Fig. 10.16(a), such as the front edge of the sloping roof and top horizontal bands of the front wall, are constant in Fig. 10.17, indicating that the gradient vector direction at all the pixel locations in those regions is the same. As we will show later in this section, angle information plays a key supporting role in the implementation of the Canny edge detection algorithm, a widely used edge detection scheme.

The original image in Fig. 10.16(a) is of reasonably high resolution, 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. One way to reduce fine detail is to smooth the image prior to computing the edges. Figure 10.18 shows the same sequence of images as in Fig. 10.16, but with the original image smoothed first using a 5×5 averaging filter (see Section 3.5 regarding smoothing filters). The response of each kernel now shows almost no contribution due to the bricks, with the results being dominated mostly by the principal edges in the image.

Figures 10.16 and 10.18 show that the horizontal and vertical Sobel kernels do not differentiate between edges in the $\pm 45^\circ$ directions. If it is important to emphasize edges oriented in particular diagonal directions, then one of the Kirsch kernels in Fig. 10.15 should be used. Figures 10.19(a) and (b) show the responses of the 45° (NW) and -45° (SW) Kirsch kernels, respectively. The stronger diagonal selectivity of these kernels is evident in these figures. Both kernels have similar responses to horizontal and vertical edges, but the response in these directions is weaker.

The threshold used to generate Fig. 10.20(a) was selected so that most of the small edges caused by the bricks were eliminated. This was the same objective as when the image in Fig. 10.16(a) was smoothed prior to computing the gradient.

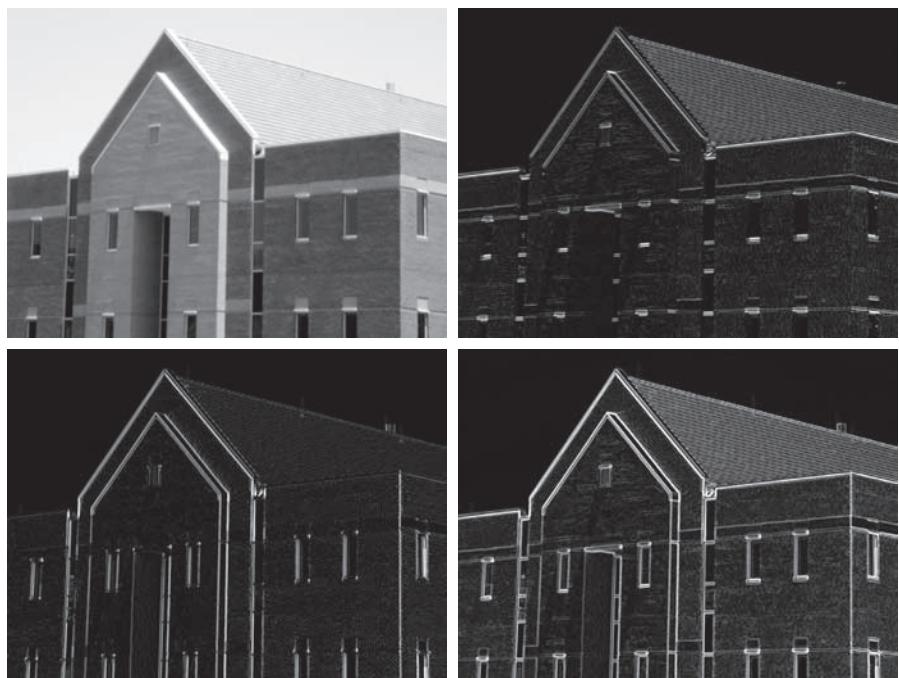
Combining the Gradient with Thresholding

The results in Fig. 10.18 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 objective is to threshold the gradient image. For example, Fig. 10.20(a) shows the gradient image from Fig. 10.16(d), thresholded so that pixels with values greater than or equal to 33% of the maximum value of the gradient image are shown in white, while pixels below the threshold value are shown in

a
b
c
d

FIGURE 10.18

Same sequence as in Fig. 10.16, but with the original image smoothed using a 5×5 averaging kernel prior to edge detection.



black. Comparing this image with Fig. 10.16(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 sloping line defining the far edge of the roof (see arrow), 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 10.20(b) shows the result of thresholding Fig. 10.18(d), which is the gradient of the smoothed image. This result shows a reduced number of broken edges; for instance, compare the corresponding edges identified by the arrows in Figs. 10.20(a) and (b).

a
b

FIGURE 10.19

Diagonal edge detection.

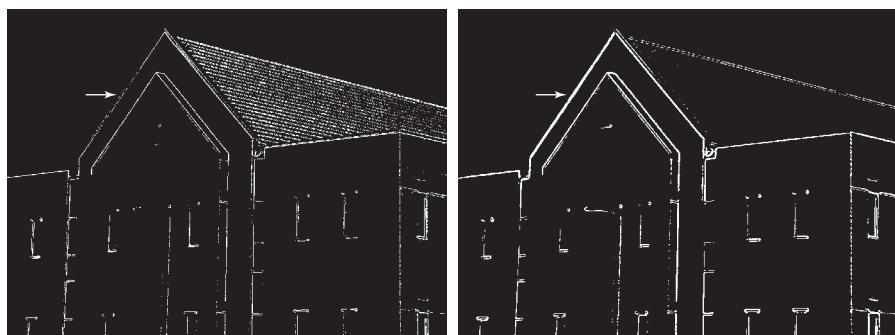
- (a) Result of using the Kirsch kernel in Fig. 10.15(c).
- (b) Result of using the kernel in Fig. 10.15(d). The input image in both cases was Fig. 10.18(a).



a b

FIGURE 10.20

- (a) Result of thresholding Fig. 10.16(d), the gradient of the original image.
 (b) Result of thresholding Fig. 10.18(d), the gradient of the smoothed image.



MORE ADVANCED TECHNIQUES FOR EDGE DETECTION

The edge-detection methods discussed in the previous subsections are based on filtering an image with one or more kernels, with no provisions made for edge characteristics and noise content. In this section, we discuss more advanced techniques that 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 [1980]. Edge-detection methods in use at the time were based on small operators, such as the Sobel kernels discussed earlier. Marr and Hildreth argued (1) that intensity changes are not independent of image scale, implying that their detection requires using 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 (as we saw in Fig. 10.10).

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 suggested that the most satisfactory operator fulfilling these conditions is the filter $\nabla^2 G$ where, as defined in Section 3.6, ∇^2 is the Laplacian, and G is the 2-D Gaussian function

$$G(x, y) = e^{-\frac{x^2 + y^2}{2\sigma^2}} \quad (10-27)$$

Equation (10-27) differs from the definition of a Gaussian function by a multiplicative constant [see Eq. (3-45)]. Here, we are interested only in the general shape of the Gaussian function.

with standard deviation σ (sometimes σ is called the *space constant* in this context). We find an expression for $\nabla^2 G$ by applying the Laplacian to Eq. (10-27):

$$\begin{aligned}
 \nabla^2 G(x, y) &= \frac{\partial^2 G(x, y)}{\partial x^2} + \frac{\partial^2 G(x, y)}{\partial y^2} \\
 &= \frac{\partial}{\partial x} \left(\frac{-x}{\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \right) + \frac{\partial}{\partial y} \left(\frac{-y}{\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \right) \\
 &= \left(\frac{x^2}{\sigma^4} - \frac{1}{\sigma^2} \right) e^{-\frac{x^2+y^2}{2\sigma^2}} + \left(\frac{y^2}{\sigma^4} - \frac{1}{\sigma^2} \right) e^{-\frac{x^2+y^2}{2\sigma^2}}
 \end{aligned} \tag{10-28}$$

Collecting terms, we obtain

$$\nabla^2 G(x, y) = \left(\frac{x^2 + y^2 - 2\sigma^2}{\sigma^4} \right) e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{10-29}$$

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

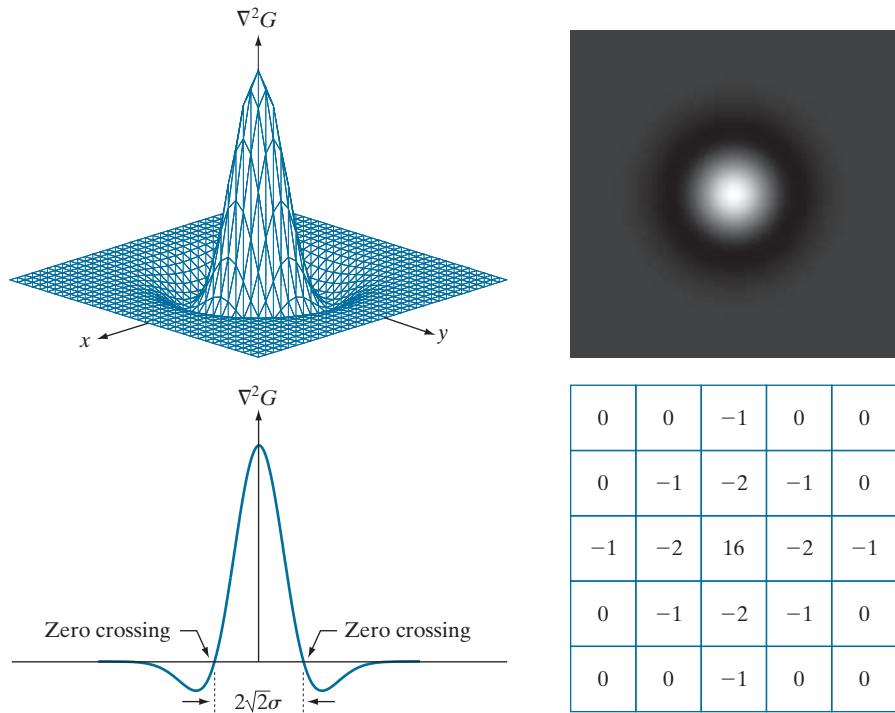
Figures 10.21(a) through (c) show a 3-D plot, image, and cross-section of the negative of the LoG function (note that the zero crossings of the LoG occur at $x^2 + y^2 = 2\sigma^2$, which defines a circle of radius $\sqrt{2}\sigma$ centered on the peak of the Gaussian function). Because of the shape illustrated in Fig. 10.21(a), the LoG function sometimes is called the *Mexican hat operator*. Figure 10.21(d) shows a 5×5 kernel that approximates the shape in Fig. 10.21(a) (normally, we would use the negative of this kernel). This approximation is not unique. Its purpose is to capture the essential shape of the LoG function; in terms of Fig. 10.21(a), this means a positive, central term surrounded by an adjacent, negative region whose values decrease 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 kernel is zero in areas of constant intensity.

Filter kernels of arbitrary size (but fixed σ) can be generated by sampling Eq. (10-29), and scaling the coefficients so that they sum to zero. A more effective approach for generating a LoG kernel is sampling Eq. (10-27) to the desired size, then convolving the resulting array with a Laplacian kernel, such as the kernel in Fig. 10.4(a). Because convolving an image with a kernel whose coefficients sum to zero yields an image whose elements also sum to zero (see Problems 3.32 and 10.16), this approach automatically satisfies the requirement that the sum of the LoG kernel coefficients be zero. We will discuss size selection for 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 . Unlike the averaging filter used in Fig. 10.18, the Gaussian function is smooth in both the spatial and frequency domains (see Section 4.8), and is thus less likely to introduce artifacts (e.g., ringing) not present in the original image. The other idea concerns the second-derivative properties of the Laplacian operator, ∇^2 . 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 (Marr [1982]) but also responds equally to changes in intensity in any kernel

a	b
c	d

FIGURE 10.21
 (a) 3-D 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 kernel approximation to the shape in (a). The negative of this kernel would be used in practice.



direction, thus avoiding having to use multiple kernels to calculate the strongest response at any point in the image.

The Marr-Hildreth algorithm consists of convolving the LoG kernel with an input image,

$$g(x, y) = [\nabla^2 G(x, y)] \star f(x, y) \quad (10-30)$$

and then finding the zero crossings of $g(x, y)$ to determine the locations of edges in $f(x, y)$. Because the Laplacian and convolution are linear processes, we can write Eq. (10-30) as

$$g(x, y) = \nabla^2 [G(x, y) \star f(x, y)] \quad (10-31)$$

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 kernel obtained by sampling Eq. (10-27).
2. Compute the Laplacian of the image resulting from Step 1 using, for example, the 3×3 kernel in Fig. 10.4(a). [Steps 1 and 2 implement Eq. (10-31).]
3. Find the zero crossings of the image from Step 2.

This expression is implemented in the spatial domain using Eq. (3-35). It can be implemented also in the frequency domain using Eq. (4-104).

As explained in Section 3.5, $\lceil \cdot \rceil$ and $\lfloor \cdot \rfloor$ denote the ceiling and floor functions. That is, the ceiling and floor functions map a real number to the smallest following, or the largest previous, integer, respectively.

Attempts to find zero crossings by finding the coordinates (x, y) where $g(x, y) = 0$ are impractical because of noise and other computational inaccuracies.

To specify the size of the Gaussian kernel, recall from our discussion of Fig. 3.35 that the values of a Gaussian function at a distance larger than 3σ from the mean are small enough so that they can be ignored. As discussed in Section 3.5, this implies using a Gaussian kernel of size $[6\sigma] \times [6\sigma]$, where $[6\sigma]$ denotes the ceiling of 6σ ; that is, smallest integer not less than 6σ . Because we work with kernels of odd dimensions, we would use the smallest *odd* integer satisfying this condition. Using a kernel smaller than this will “truncate” the LoG function, with the degree of truncation being inversely proportional to the size of the kernel. Using a larger kernel 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 to use 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. We illustrate this approach in Example 10.7.

Computing zero crossings is 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 [1986] for finding zero crossings with *subpixel accuracy* can be employed.

EXAMPLE 10.7: Illustration of the Marr-Hildreth edge-detection method.

Figure 10.22(a) shows the building image used earlier and Fig. 10.22(b) is the result of Steps 1 and 2 of the Marr-Hildreth algorithm, using $\sigma = 4$ (approximately 0.5% of the short dimension of the image) and $n = 25$ to satisfy the size condition stated above. As in Fig. 10.5, the gray tones in this image are due to scaling. Figure 10.22(c) shows the zero crossings obtained using the 3×3 neighborhood approach just discussed, with a threshold of zero. Note that all the edges form closed loops. This so-called “spaghetti effect” is a serious drawback of this method when a threshold value of zero is used (see Problem 10.17). We avoid closed-loop edges by using a positive threshold.

Figure 10.22(d) shows the result of using a threshold approximately equal to 4% of the maximum value of the LoG image. The majority of the principal edges were readily detected, and “irrelevant” features, such as the edges due to the bricks and the tile roof, were filtered out. This type of performance is virtually impossible to obtain using the gradient-based edge-detection techniques discussed earlier. Another important consequence of using zero crossings for edge detection is that the resulting edges are 1 pixel thick. This property simplifies subsequent stages of processing, such as edge linking.

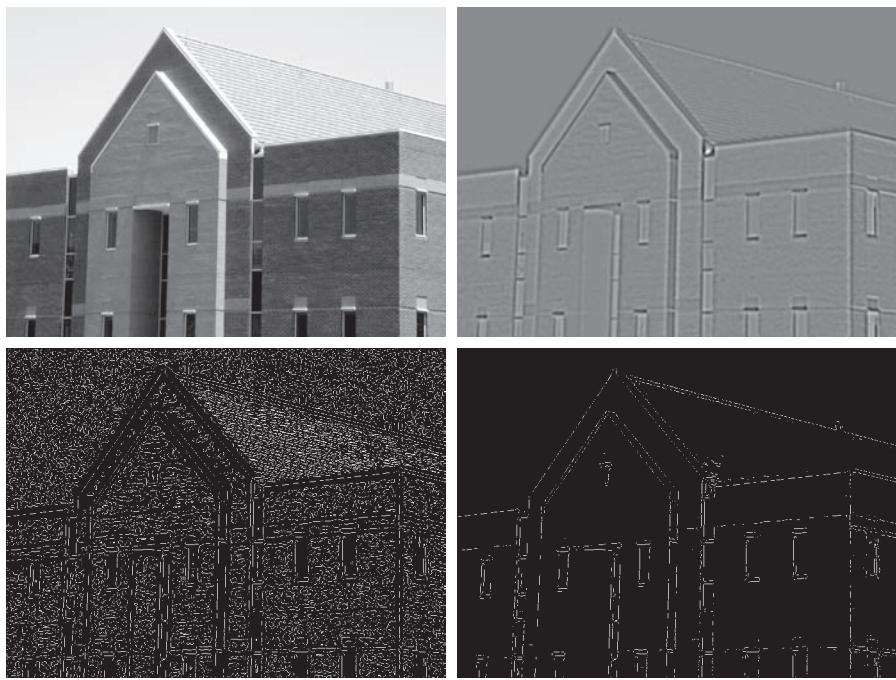
It is possible to approximate the LoG function in Eq. (10-29) by a *difference of Gaussians* (DoG):

$$D_G(x, y) = \frac{1}{2\pi\sigma_1^2} e^{-\frac{x^2+y^2}{2\sigma_1^2}} - \frac{1}{2\pi\sigma_2^2} e^{-\frac{x^2+y^2}{2\sigma_2^2}} \quad (10-32)$$

a	b
c	d

FIGURE 10.22

- (a) Image of size 834×1114 pixels, with intensity values scaled to the range $[0, 1]$.
 (b) Result of Steps 1 and 2 of the Marr-Hildreth algorithm using $\sigma = 4$ and $n = 25$.
 (c) Zero crossings of (b) using a threshold of 0 (note the closed-loop edges).
 (d) Zero crossings found using a threshold equal to 4% of the maximum value of the image in (b). Note the thin edges.



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. (10-32) with a ratio of standard deviations of 1.75:1. Using the ratio 1.6:1 preserves the basic characteristics of these observations and also provides a closer “engineering” approximation to the LoG function (Marr and Hildreth [1980]). In order for the LoG and DoG to have the same zero crossings, the value of σ for the LoG must be selected based on the following equation (see Problem 10.19):

$$\sigma^2 = \frac{\sigma_1^2 \sigma_2^2}{\sigma_1^2 - \sigma_2^2} \ln \left[\frac{\sigma_1^2}{\sigma_2^2} \right] \quad (10-33)$$

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.

The profiles in Figs. 10.23(a) and (b) were generated with standard deviation ratios of 1:1.75 and 1:1.6, respectively (by convention, the curves shown are inverted, as in Fig. 10.21). The LoG profiles are the solid lines, and the DoG profiles are dotted. The curves shown are intensity profiles through the center of the LoG and DoG arrays, generated by sampling Eqs. (10-29) and (10-32), respectively. The amplitude of all curves at the origin were normalized to 1. As Fig. 10.23(b) shows, the ratio 1:1.6 yielded a slightly closer approximation of the LoG and DoG functions (for example, compare the bottom lobes of the two figures).

a b

FIGURE 10.23

- (a) Negatives of the LoG (solid) and DoG (dotted) profiles using a σ ratio of 1.75:1. (b) Profiles obtained using a ratio of 1.6:1.



Gaussian kernels are separable (see Section 3.4). Therefore, both the LoG and the DoG filtering operations can be implemented with 1-D convolutions instead of using 2-D convolutions directly (see Problem 10.19). For an image of size $M \times N$ and a kernel of size $n \times n$, doing so reduces the number of multiplications and additions for each convolution from being proportional to $n^2 MN$ 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.

The Canny Edge Detector

Although the algorithm is more complex, the performance of the Canny edge detector (Canny [1986]) discussed in this section is superior in general to the edge detectors discussed thus far. Canny's approach is based on three basic objectives:

- Low error rate.* All edges should be found, and there should be no spurious responses.
- Edge points should be well localized.* The edges located must be as close as possible to the true edges. That is, the distance between a point marked as an edge by the detector and the center of the true edge should be minimum.
- Single edge point response.* The detector should return only one point for each true edge point. That is, the number of local maxima around the true edge should be minimum. This means that the detector should not identify multiple edge pixels where only a single edge point exists.

The essence of Canny's work was in expressing the preceding three criteria mathematically, and then attempting to find optimal solutions to these formulations. In general, it is difficult (or impossible) to find a closed-form solution that satisfies all the preceding objectives. However, using numerical optimization with 1-D step edges corrupted by additive white Gaussian noise[†] led to the conclusion that a good approximation to the optimal step edge detector is the *first derivative of a Gaussian*,

$$\frac{d}{dx} e^{-\frac{x^2}{2\sigma^2}} = \frac{-x}{\sigma^2} e^{-\frac{x^2}{2\sigma^2}} \quad (10-34)$$

[†]Recall that *white noise* is noise having a frequency spectrum that is continuous and uniform over a specified frequency band. *White Gaussian noise* is white noise in which the distribution of amplitude values is Gaussian. Gaussian white noise is a good approximation of many real-world situations and generates mathematically tractable models. It has the useful property that its values are statistically independent.

where the approximation was only about 20% worse than using the optimized numerical solution (a difference of this magnitude generally is visually imperceptible in most applications).

Generalizing the preceding result to 2-D involves recognizing that the 1-D approach still applies in the direction of the edge normal (see Fig. 10.12). Because the direction of the normal is unknown beforehand, this would require applying the 1-D edge detector in all possible directions. This task can be approximated by first smoothing the image with a circular 2-D Gaussian function, computing the gradient of the result, and then using the gradient magnitude and direction to estimate edge strength and direction at every point.

Let $f(x, y)$ denote the input image and $G(x, y)$ denote the Gaussian function:

$$G(x, y) = e^{-\frac{x^2 + y^2}{2\sigma^2}} \quad (10-35)$$

We form a smoothed image, $f_s(x, y)$, by convolving f and G :

$$f_s(x, y) = G(x, y) \star f(x, y) \quad (10-36)$$

This operation is followed by computing the gradient magnitude and direction (angle), as discussed earlier:

$$M_s(x, y) = \|\nabla f_s(x, y)\| = \sqrt{g_x^2(x, y) + g_y^2(x, y)} \quad (10-37)$$

and

$$\alpha(x, y) = \tan^{-1} \left[\frac{g_y(x, y)}{g_x(x, y)} \right] \quad (10-38)$$

with $g_x(x, y) = \partial f_s(x, y) / \partial x$ and $g_y(x, y) = \partial f_s(x, y) / \partial y$. Any of the derivative filter kernel pairs in Fig. 10.14 can be used to obtain $g_x(x, y)$ and $g_y(x, y)$. Equation (10-36) is implemented using an $n \times n$ Gaussian kernel whose size is discussed below. Keep in mind that $\|\nabla f_s(x, y)\|$ and $\alpha(x, y)$ are arrays of the same size as the image from which they are computed.

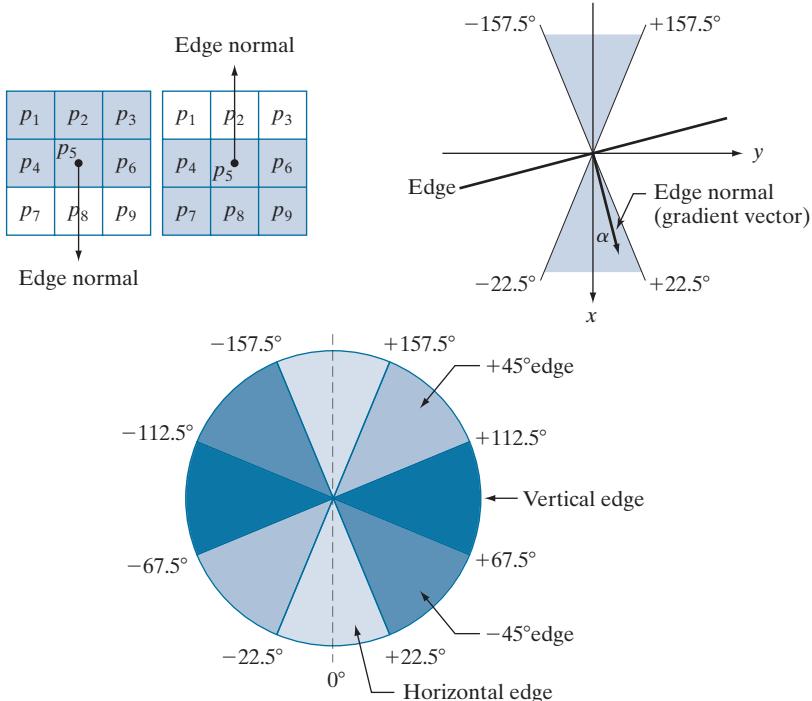
Gradient image $\|\nabla f_s(x, y)\|$ typically contains wide ridges around local maxima. The next step is to thin those ridges. One approach is to use *nonmaxima suppression*. The essence of this approach is to specify a number of discrete orientations of the edge normal (gradient vector). For example, in a 3×3 region we can define four orientations[†] for an edge passing through the center point of the region: horizontal, vertical, $+45^\circ$, and -45° . Figure 10.24(a) shows the situation for the two possible orientations of a horizontal edge. Because we have to quantize all possible edge directions into four ranges, we have to define a range of directions over which we consider an edge to be horizontal. We determine edge direction from the direction of the edge normal, which we obtain directly from the image data using Eq. (10-38). As Fig. 10.24(b) shows, if the edge normal is in the range of directions from -22.5° to

[†]Every edge has two possible orientations. For example, an edge whose normal is oriented at 0° and an edge whose normal is oriented at 180° are the same *horizontal* edge.

a
b
c

FIGURE 10.24

- (a) Two possible orientations of a horizontal edge (shaded) in a 3×3 neighborhood.
 (b) Range of values (shaded) of α , the direction angle of the edge normal for a horizontal edge. (c) The angle ranges of the edge normals for the four types of edge directions in a 3×3 neighborhood. Each edge direction has two ranges, shown in corresponding shades.



22.5° or from -157.5° to 157.5° , we call the edge a horizontal edge. Figure 10.24(c) shows the angle ranges corresponding to the four directions under consideration.

Let d_1, d_2, d_3 , and d_4 denote the four basic edge directions just discussed for a 3×3 region: horizontal, -45° , vertical, and $+45^\circ$, respectively. We can formulate the following nonmaxima suppression scheme for a 3×3 region centered at an arbitrary point (x, y) in α :

1. Find the direction d_k that is closest to $\alpha(x, y)$.
2. Let K denote the value of $\|\nabla f_s\|$ at (x, y) . If K is less than the value of $\|\nabla f_s\|$ at one or both of the neighbors of point (x, y) along d_k , let $g_N(x, y) = 0$ (suppression); otherwise, let $g_N(x, y) = K$.

When repeated for all values of x and y , this procedure yields a nonmaxima suppressed image $g_N(x, y)$ that is of the same size as $f_s(x, y)$. For example, with reference to Fig. 10.24(a), letting (x, y) be at p_5 , and assuming a horizontal edge through p_5 , the pixels of interest in Step 2 would be p_2 and p_8 . Image $g_N(x, y)$ contains only the thinned edges; it is equal to image $\|\nabla f_s(x, y)\|$ with the nonmaxima edge points suppressed.

The final operation is to threshold $g_N(x, y)$ to reduce false edge points. In the Marr-Hildreth algorithm we did this using a single threshold, in which all values below the threshold were set to 0. If we set the threshold too low, there will still be some false edges (called *false positives*). If the threshold is set too high, then valid edge points will be eliminated (*false negatives*). Canny's algorithm attempts to

improve on this situation by using *hysteresis thresholding* which, as we will discuss in Section 10.3, uses two thresholds: a low threshold, T_L and a high threshold, T_H . Experimental evidence (Canny [1986]) suggests that the ratio of the high to low threshold should be in the range of 2:1 to 3:1.

We can visualize the thresholding operation as creating two additional images:

$$g_{NH}(x, y) = g_N(x, y) \geq T_H \quad (10-39)$$

and

$$g_{NL}(x, y) = g_N(x, y) \geq T_L \quad (10-40)$$

Initially, $g_{NH}(x, y)$ and $g_{NL}(x, y)$ are set to 0. After thresholding, $g_{NH}(x, y)$ will usually have fewer nonzero pixels than $g_{NL}(x, y)$, but all the nonzero pixels in $g_{NH}(x, y)$ will be contained in $g_{NL}(x, y)$ because the latter image is formed with a lower threshold. We eliminate from $g_{NL}(x, y)$ all the nonzero pixels from $g_{NH}(x, y)$ by letting

$$g_{NL}(x, y) = g_{NL}(x, y) - g_{NH}(x, y) \quad (10-41)$$

The nonzero pixels in $g_{NH}(x, y)$ and $g_{NL}(x, y)$ may be viewed as being “strong” and “weak” edge pixels, respectively. After the thresholding operations, all strong pixels in $g_{NH}(x, y)$ are assumed to be valid edge pixels, and are so marked immediately. Depending on the value of T_H , the edges in $g_{NH}(x, y)$ typically have gaps. Longer edges are formed using the following procedure:

- (a) Locate the next unvisited edge pixel, p , in $g_{NH}(x, y)$.
- (b) Mark as valid edge pixels all the weak pixels in $g_{NL}(x, y)$ that are connected to p using, say, 8-connectivity.
- (c) If all nonzero pixels in $g_{NH}(x, y)$ have been visited go to Step (d). Else, return to Step (a).
- (d) Set to zero all pixels in $g_{NL}(x, y)$ that were not marked as valid edge pixels.

At the end of this procedure, the final image output by the Canny algorithm is formed by appending to $g_{NH}(x, y)$ all the nonzero pixels from $g_{NL}(x, y)$.

We used two additional images, $g_{NH}(x, y)$ and $g_{NL}(x, y)$ to simplify the discussion. In practice, hysteresis thresholding can be implemented directly during nonmaxima suppression, and thresholding can be implemented directly on $g_N(x, y)$ by forming a list of strong pixels and the weak pixels connected to them.

Summarizing, the Canny edge detection algorithm consists of the following steps:

1. Smooth the input image with a Gaussian filter.
2. Compute the gradient magnitude and angle images.
3. Apply nonmaxima suppression to the gradient magnitude image.
4. Use double thresholding and connectivity analysis to detect and link edges.

Although the edges after nonmaxima suppression are thinner than raw gradient edges, the former can still be thicker than one pixel. To obtain edges one pixel thick, it is typical to follow Step 4 with one pass of an edge-thinning algorithm (see Section 9.5).

Usually, selecting a suitable value of σ for the first time in an application requires experimentation.

As mentioned earlier, smoothing is accomplished by convolving the input image with a Gaussian kernel whose size, $n \times n$, must be chosen. Once a value of σ has been specified, we can use the approach discussed in connection with the Marr-Hildreth algorithm to determine an odd value of n that provides the “full” smoothing capability of the Gaussian filter for the specified value of σ .

Some final comments on implementation: As noted earlier in the discussion of the Marr-Hildreth edge detector, the 2-D Gaussian function in Eq. (10-35) is separable into a product of two 1-D Gaussians. Thus, Step 1 of the Canny algorithm can be formulated as 1-D convolutions that operate on the rows (columns) of the image one at a time, and then work on the columns (rows) of the result. Furthermore, if we use the approximations in Eqs. (10-19) and (10-20), we can also implement the gradient computations required for Step 2 as 1-D convolutions (see Problem 10.22).

EXAMPLE 10.8: Illustration and comparison of the Canny edge-detection method.

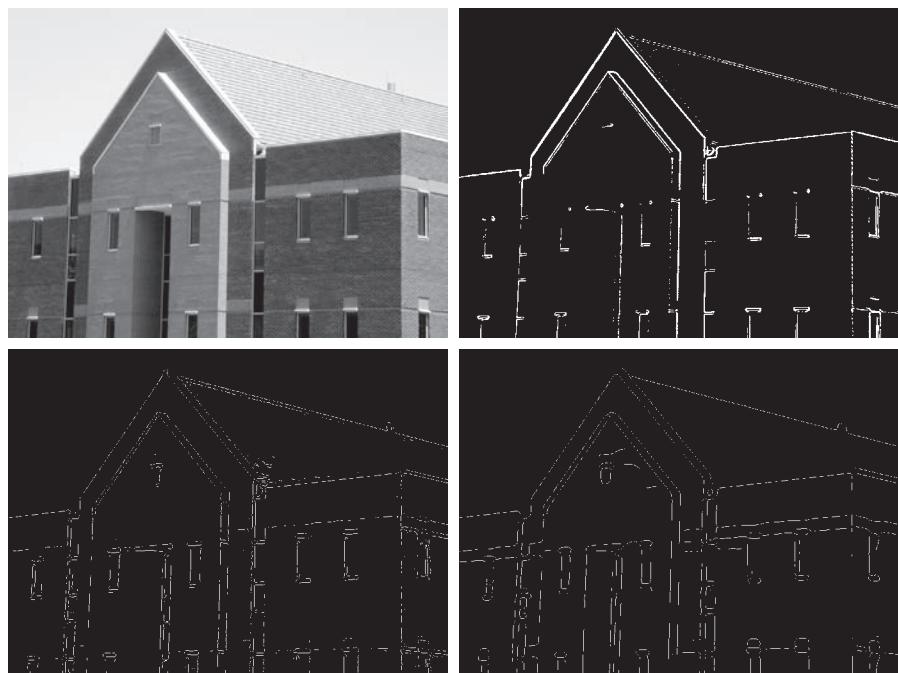
Figure 10.25(a) shows the familiar building image. For comparison, Figs. 10.25(b) and (c) show, respectively, the result in Fig. 10.20(b) obtained using the thresholded gradient, and Fig. 10.22(d) using the Marr-Hildreth detector. Recall that the parameters used in generating those two images were selected to detect the principal edges, while attempting to reduce “irrelevant” features, such as the edges of the bricks and the roof tiles.

Figure 10.25(d) shows the result obtained with the Canny algorithm using the parameters $T_L = 0.04$, $T_H = 0.10$ (2.5 times the value of the low threshold), $\sigma = 4$, and a kernel of size 25×25 , which corresponds to the smallest odd integer not less than 6σ . These parameters were chosen experimentally

a b
c d

FIGURE 10.25

- (a) Original image of size 834×1114 pixels, with intensity values scaled to the range $[0, 1]$.
- (b) Thresholded gradient of the smoothed image.
- (c) Image obtained using the Marr-Hildreth algorithm.
- (d) Image obtained using the Canny algorithm. Note the significant improvement of the Canny image compared to the other two.



to achieve the objectives stated in the previous paragraph for the gradient and Marr-Hildreth images. Comparing the Canny image with the other two images, we see in the Canny result significant improvements in detail of the principal edges and, at the same time, more rejection of irrelevant features. For example, note that both edges of the concrete band lining the bricks in the upper section of the image were detected by the Canny algorithm, whereas the thresholded gradient lost both of these edges, and the Marr-Hildreth method detected only the upper one. In terms of filtering out irrelevant detail, the Canny image does not contain a single edge due to the roof tiles; this is not true in the other two images. The quality of the lines with regard to continuity, thinness, and straightness is also superior in the Canny image. Results such as these have made the Canny algorithm a tool of choice for edge detection.

EXAMPLE 10.9: Another illustration of the three principal edge-detection methods discussed in this section.

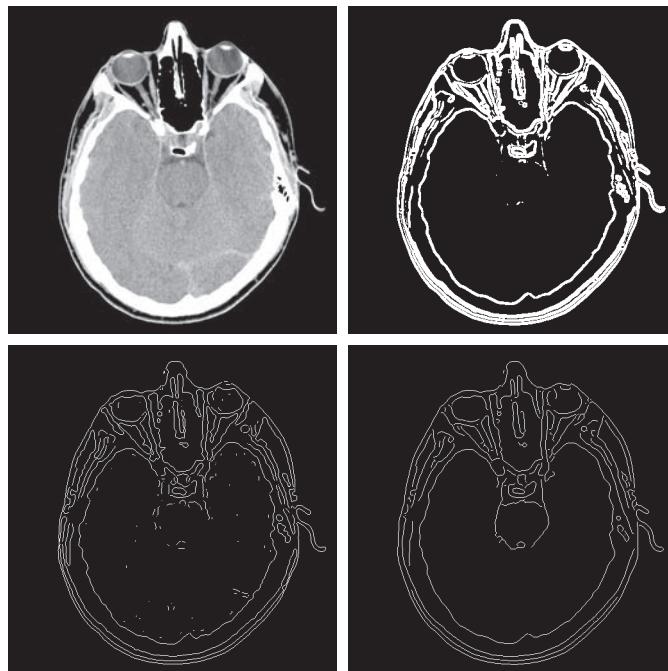
As another comparison of the three principal edge-detection methods discussed in this section, consider Fig. 10.26(a), which shows a 512×512 head CT image. Our objective is to extract the edges of the outer contour of the brain (the gray region in the image), the contour of the spinal region (shown directly behind the nose, toward the front of the brain), and the outer contour of the head. We wish to generate the thinnest, continuous contours possible, while eliminating edge details related to the gray content in the eyes and brain areas.

Figure 10.26(b) shows a thresholded gradient image that was first smoothed using a 5×5 averaging kernel. The threshold required to achieve the result shown was 15% of the maximum value of the gradient image. Figure 10.26(c) shows the result obtained with the Marr-Hildreth edge-detection algorithm with a threshold of 0.002, $\sigma = 3$, and a kernel of size 19×19 . Figure 10.26(d) was obtained using the Canny algorithm with $T_L = 0.05$, $T_H = 0.15$ (3 times the value of the low threshold), $\sigma = 2$, and a kernel of size 13×13 .

a	b
c	d

FIGURE 10.26

- (a) Head CT image of size 512×512 pixels, with intensity values scaled to the range $[0, 1]$.
 - (b) Thresholded gradient of the smoothed image.
 - (c) Image obtained using the Marr-Hildreth algorithm.
 - (d) Image obtained using the Canny algorithm.
- (Original image courtesy of Dr. David R. Pickens, Vanderbilt University.)



In terms of edge quality and the ability to eliminate irrelevant detail, the results in Fig. 10.26 correspond closely to the results and conclusions in the previous example. Note also that the Canny algorithm was the only procedure capable of yielding a totally unbroken edge for the posterior boundary of the brain, and the closest boundary of the spinal cord. It was also the only procedure capable of finding the cleanest contours, while eliminating all the edges associated with the gray brain matter in the original image.

The price paid for the improved performance of the Canny algorithm is a significantly more complex implementation than the two approaches discussed earlier. In some applications, such as real-time industrial image processing, cost and speed requirements usually dictate the use of simpler techniques, principally the thresholded gradient approach. When edge quality is the driving force, the Marr-Hildreth and Canny algorithms, especially the latter, offer superior alternatives.

LINKING EDGE POINTS

Ideally, edge detection should yield sets of pixels lying only on edges. In practice, these pixels seldom characterize edges completely because of noise, breaks in the edges caused by nonuniform illumination, and other effects that introduce discontinuities in intensity values. Therefore, edge detection typically is followed by linking algorithms designed to assemble edge pixels into meaningful edges and/or region boundaries. In this section, we discuss two fundamental approaches to edge linking that are representative of techniques used in practice. The first requires knowledge about edge points in a local region (e.g., a 3×3 neighborhood), and the second is a global approach that works with an entire edge map. As it turns out, linking points along the boundary of a region is also an important aspect of some of the segmentation methods discussed in the next chapter, and in extracting features from a segmented image, as we will do in Chapter 11. Thus, you will encounter additional edge-point linking methods in the next two chapters.

Local Processing

A simple approach for linking edge points is to analyze the characteristics of pixels in a small neighborhood about every point (x, y) that has been declared an edge point by one of the techniques discussed in the preceding sections. All points that are similar according to predefined criteria are linked, forming an edge of pixels that share common properties according to the specified criteria.

The two principal properties used for establishing similarity of edge pixels in this kind of local analysis are (1) the strength (magnitude) and (2) the direction of the gradient vector. The first property is based on Eq. (10-17). Let S_{xy} denote the set of coordinates of a neighborhood centered at point (x, y) in an image. An edge pixel with coordinates (s, t) in S_{xy} is similar in *magnitude* to the pixel at (x, y) if

$$|M(s, t) - M(x, y)| \leq E \quad (10-42)$$

where E is a positive threshold.

The direction angle of the gradient vector is given by Eq. (10-18). An edge pixel with coordinates (s, t) in S_{xy} has an *angle* similar to the pixel at (x, y) if

$$|\alpha(s, t) - \alpha(x, y)| \leq A \quad (10-43)$$

where A is a positive angle threshold. As noted earlier, the direction of the edge at (x, y) is perpendicular to the direction of the gradient vector at that point.

A pixel with coordinates (s, t) in S_{xy} is considered to be linked to the pixel at (x, y) if both magnitude and direction criteria are satisfied. This process is repeated for every edge pixel. As the center of the neighborhood is moved from pixel to pixel, a record of linked points is kept. A simple bookkeeping procedure is to assign a different intensity value to each set of linked edge pixels.

The preceding formulation is computationally expensive because all neighbors of every point have to be examined. A simplification particularly well suited for real time applications consists of the following steps:

1. Compute the gradient magnitude and angle arrays, $M(x, y)$ and $\alpha(x, y)$, of the input image, $f(x, y)$.
2. Form a binary image, $g(x, y)$, whose value at any point (x, y) is given by:

$$g(x, y) = \begin{cases} 1 & \text{if } M(x, y) > T_M \text{ AND } \alpha(x, y) = A \pm T_A \\ 0 & \text{otherwise} \end{cases}$$

where T_M is a threshold, A is a specified angle direction, and $\pm T_A$ defines a “band” of acceptable directions about A .

3. Scan the rows of g and fill (set to 1) all gaps (sets of 0's) in each row that do not exceed a specified length, L . Note that, by definition, a gap is bounded at both ends by one or more 1's. The rows are processed individually, with no “memory” kept between them.
4. To detect gaps in any other direction, θ , rotate g by this angle and apply the horizontal scanning procedure in Step 3. Rotate the result back by $-\theta$.

When interest lies in horizontal and vertical edge linking, Step 4 becomes a simple procedure in which g is rotated ninety degrees, the rows are scanned, and the result is rotated back. This is the application found most frequently in practice and, as the following example shows, this approach can yield good results. In general, image rotation is an expensive computational process so, when linking in numerous angle directions is required, it is more practical to combine Steps 3 and 4 into a single, radial scanning procedure.

EXAMPLE 10.10: Edge linking using local processing.

Figure 10.27(a) shows a 534×566 image of the rear of a vehicle. The objective of this example is to illustrate the use of the preceding algorithm for finding rectangles whose sizes makes them suitable candidates for license plates. The formation of these rectangles can be accomplished by detecting

a	b	c
d	e	f

FIGURE 10.27

- (a) Image of the rear of a vehicle.
- (b) Gradient magnitude image.
- (c) Horizontally connected edge pixels.
- (d) Vertically connected edge pixels.
- (e) The logical OR of (c) and (d).
- (f) Final result, using morphological thinning. (Original image courtesy of Perceptics Corporation.)



strong horizontal and vertical edges. Figure 10.27(b) shows the gradient magnitude image, $M(x, y)$, and Figs. 10.27(c) and (d) show the result of Steps 3 and 4 of the algorithm, obtained by letting T_M equal to 30% of the maximum gradient value, $A = 90^\circ$, $T_A = 45^\circ$, and filling all gaps of 25 or fewer pixels (approximately 5% of the image width). A large range of allowable angle directions was required to detect the rounded corners of the license plate enclosure, as well as the rear windows of the vehicle. Figure 10.27(e) is the result of forming the logical OR of the two preceding images, and Fig. 10.27(f) was obtained by thinning 10.27(e) with the thinning procedure discussed in Section 9.5. As Fig. 10.27(f) shows, the rectangle corresponding to the license plate was clearly detected in the image. It would be a simple matter to isolate the license plate from all the rectangles in the image, using the fact that the width-to-height ratio of license plates have distinctive proportions (e.g., a 2:1 ratio in U.S. plates).

Global Processing Using the Hough Transform

The method discussed in the previous section is applicable in situations in which knowledge about pixels belonging to individual objects is available. Often, we have to work in unstructured environments in which all we have is an edge map and no knowledge about where objects of interest might be. In such situations, all pixels are candidates for linking, and thus have to be accepted or eliminated based on pre-defined *global* properties. In this section, we develop an approach based on whether sets of pixels lie on curves of a specified shape. Once detected, these curves form the edges or region boundaries of interest.

Given n points in an image, suppose that we want to find subsets of these points that lie on straight lines. One possible solution is to find all lines determined by every pair of points, then find all subsets of points that are close to particular lines. This approach involves finding $n(n - 1)/2 \sim n^2$ lines, then performing $(n)(n(n - 1))/2 \sim n^3$

The original formulation of the Hough transform presented here works with straight lines. For a generalization to arbitrary shapes, see Ballard [1981].

comparisons of every point to all lines. This is a computationally prohibitive task in most applications.

Hough [1962] proposed an alternative approach, commonly referred to as the *Hough transform*. Let (x_i, y_i) denote a point in the xy -plane and consider the general equation of a straight line in slope-intercept form: $y_i = ax_i + b$. Infinitely many lines pass through (x_i, y_i) , but they all satisfy the equation $y_i = ax_i + b$ for varying values of a and b . However, writing this equation as $b = -x_i a + y_i$ and considering the ab -plane (also called *parameter space*) yields the equation of a *single* line for a fixed point (x_i, y_i) . Furthermore, a second point (x_j, y_j) also has a single line in parameter space associated with it, which intersects the line associated with (x_i, y_i) at some point (a', b') in parameter space, where a' is the slope and b' the intercept of the line containing both (x_i, y_i) and (x_j, y_j) in the xy -plane (we are assuming, of course, that the lines are not parallel). In fact, *all* points on this line have lines in parameter space that intersect at (a', b') . Figure 10.28 illustrates these concepts.

In principle, the parameter space lines corresponding to all points (x_k, y_k) in the xy -plane could be plotted, and the principal lines in that plane could be found by identifying points in parameter space where large numbers of parameter-space lines intersect. However, a difficulty with this approach is that a , (the slope of a line) approaches infinity as the line approaches the vertical direction. One way around this difficulty is to use the *normal representation* of a line:

$$x \cos \theta + y \sin \theta = \rho \quad (10-44)$$

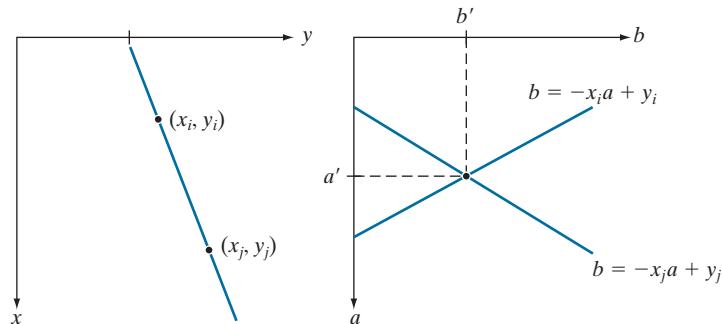
Figure 10.29(a) illustrates the geometrical interpretation of the parameters ρ and θ . A horizontal line has $\theta = 0^\circ$, with ρ being equal to the positive x -intercept. Similarly, a vertical line has $\theta = 90^\circ$, with ρ being equal to the positive y -intercept, or $\theta = -90^\circ$, with ρ being equal to the negative y -intercept (we limit the angle to the range $-90^\circ \leq \theta \leq 90^\circ$). Each sinusoidal curve in Figure 10.29(b) represents the family of lines that pass through a particular point (x_k, y_k) in the xy -plane. The intersection point (ρ', θ') in Fig. 10.29(b) corresponds to the line that passes through both (x_i, y_i) and (x_j, y_j) in Fig. 10.29(a).

The computational attractiveness of the Hough transform arises from subdividing the $\rho\theta$ parameter space into so-called *accumulator cells*, as Fig. 10.29(c) illustrates, where $(\rho_{\min}, \rho_{\max})$ and $(\theta_{\min}, \theta_{\max})$ are the expected ranges of the parameter values:

a b

FIGURE 10.28

(a) xy -plane.
(b) Parameter space.



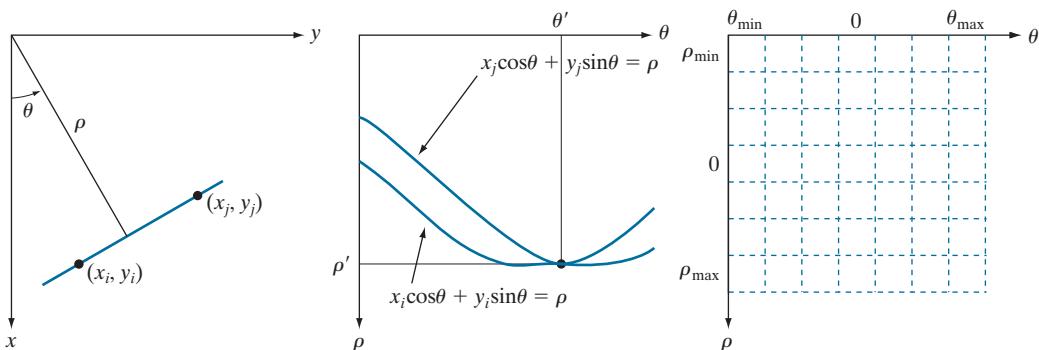


FIGURE 10.29 (a) (ρ, θ) parameterization of a line in the xy -plane. (b) Sinusoidal curves in the $\rho\theta$ -plane; the point of intersection (ρ', θ') corresponds to the line passing through points (x_i, y_i) and (x_j, y_j) in the xy -plane. (c) Division of the $\rho\theta$ -plane into accumulator cells.

$-90^\circ \leq \theta \leq 90^\circ$ and $-D \leq \rho \leq D$, where D is the maximum distance between opposite corners in an image. The cell at coordinates (i, j) with accumulator value $A(i, j)$ corresponds to the square associated with parameter-space coordinates (ρ_i, θ_j) . Initially, these cells are set to zero. Then, for every non-background point (x_k, y_k) in the xy -plane, we let θ equal each of the allowed subdivision values on the θ -axis and solve for the corresponding ρ using the equation $\rho = x_k \cos \theta + y_k \sin \theta$. The resulting ρ values are then rounded off to the nearest allowed cell value along the ρ axis. If a choice of θ_q results in the solution ρ_p , then we let $A(p, q) = A(p, q) + 1$. At the end of the procedure, a value of K in a cell $A(i, j)$ means that K points in the xy -plane lie on the line $x \cos \theta_j + y \sin \theta_j = \rho_i$. The number of subdivisions in the $\rho\theta$ -plane determines the accuracy of the colinearity of these points. It can be shown (see Problem 10.27) that the number of computations in the method just discussed is linear with respect to n , the number of non-background points in the xy -plane.

EXAMPLE 10.11: Some basic properties of the Hough transform.

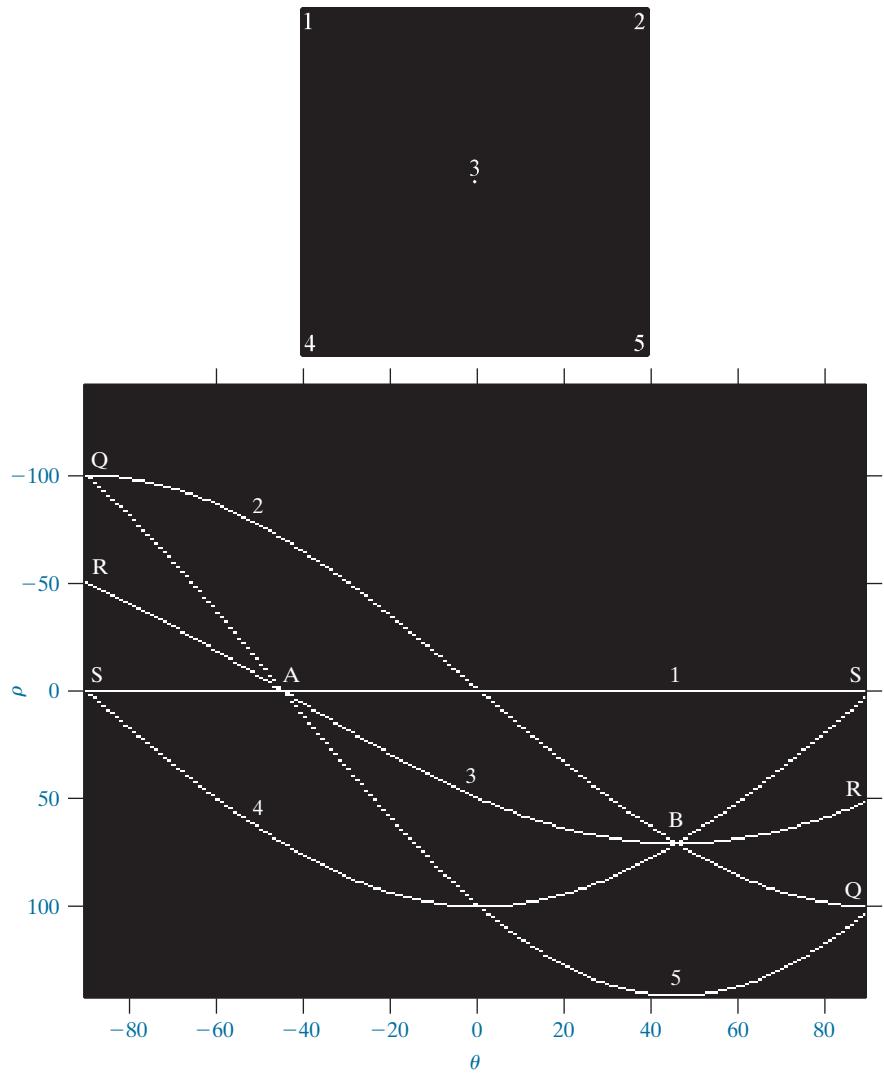
Figure 10.30 illustrates the Hough transform based on Eq. (10-44). Figure 10.30(a) shows an image of size $M \times M$ ($M = 101$) with five labeled white points, and Fig. 10.30(b) shows each of these points mapped onto the $\rho\theta$ -plane using subdivisions of one unit for the ρ and θ axes. The range of θ values is $\pm 90^\circ$, and the range of ρ values is $\pm \sqrt{2}M$. As Fig. 10.30(b) shows, each curve has a different sinusoidal shape. The horizontal line resulting from the mapping of point 1 is a sinusoid of zero amplitude.

The points labeled A (not to be confused with accumulator values) and B in Fig. 10.30(b) illustrate the colinearity detection property of the Hough transform. For example, point B , marks the intersection of the curves corresponding to points 2, 3, and 4 in the xy image plane. The location of point A indicates that these three points lie on a straight line passing through the origin ($\rho = 0$) and oriented at -45° [see Fig. 10.29(a)]. Similarly, the curves intersecting at point B in parameter space indicate that points 2, 3, and 4 lie on a straight line oriented at 45° , and whose distance from the origin is $\rho = 71$ (one-half the diagonal distance from the origin of the image to the opposite corner, rounded to the nearest integer).

a
b

FIGURE 10.30

(a) Image of size 101×101 pixels, containing five white points (four in the corners and one in the center). (b) Corresponding parameter space.



value). Finally, the points labeled Q , R , and S in Fig. 10.30(b) illustrate the fact that the Hough transform exhibits a reflective adjacency relationship at the right and left edges of the parameter space. This property is the result of the manner in which ρ and θ change sign at the $\pm 90^\circ$ boundaries.

Although the focus thus far has been on straight lines, the Hough transform is applicable to any function of the form $g(\mathbf{v}, \mathbf{c}) = 0$, where \mathbf{v} is a vector of coordinates and \mathbf{c} is a vector of coefficients. For example, points lying on the circle

$$(x - c_1)^2 + (y - c_2)^2 = c_3^2 \quad (10-45)$$

can be detected by using the basic approach just discussed. The difference is the presence of three parameters c_1 , c_2 , and c_3 that result in a 3-D parameter space with

cube-like cells, and accumulators of the form $A(i, j, k)$. The procedure is to increment c_1 and c_2 , solve for the value of c_3 that satisfies Eq. (10-45), and update the accumulator cell associated with the triplet (c_1, c_2, c_3) . Clearly, the complexity of the Hough transform depends on the number of coordinates and coefficients in a given functional representation. As noted earlier, generalizations of the Hough transform to detect curves with no simple analytic representations are possible, as is the application of the transform to grayscale images.

Returning to the edge-linking problem, an approach based on the Hough transform is as follows:

1. Obtain a binary edge map using any of the methods discussed earlier in this section.
2. Specify subdivisions in the $\rho\theta$ -plane.
3. Examine the counts of the accumulator cells for high pixel concentrations.
4. Examine the relationship (principally for continuity) between pixels in a chosen cell.

Continuity in this case usually is based on computing the distance between disconnected pixels corresponding to a given accumulator cell. A gap in a line associated with a given cell is bridged if the length of the gap is less than a specified threshold. Being able to group lines based on direction is a global concept applicable over the entire image, requiring only that we examine pixels associated with specific accumulator cells. The following example illustrates these concepts.

EXAMPLE 10.12: Using the Hough transform for edge linking.

Figure 10.31(a) shows an aerial image of an airport. The objective of this example is to use the Hough transform to extract the two edges defining the principal runway. A solution to such a problem might be of interest, for instance, in applications involving autonomous air navigation.

The first step is to obtain an edge map. Figure 10.31(b) shows the edge map obtained using Canny's algorithm with the same parameters and procedure used in Example 10.9. For the purpose of computing the Hough transform, similar results can be obtained using any of the other edge-detection techniques discussed earlier. Figure 10.31(c) shows the Hough parameter space obtained using 1° increments for θ , and one-pixel increments for ρ .

The runway of interest is oriented approximately 1° off the north direction, so we select the cells corresponding to $\pm 90^\circ$ and containing the highest count because the runways are the longest lines oriented in these directions. The small boxes on the edges of Fig. 10.31(c) highlight these cells. As mentioned earlier in connection with Fig. 10.30(b), the Hough transform exhibits adjacency at the edges. Another way of interpreting this property is that a line oriented at $+90^\circ$ and a line oriented at -90° are equivalent (i.e., they are both vertical). Figure 10.31(d) shows the lines corresponding to the two accumulator cells just discussed, and Fig. 10.31(e) shows the lines superimposed on the original image. The lines were obtained by joining all gaps not exceeding 20% (approximately 100 pixels) of the image height. These lines clearly correspond to the edges of the runway of interest.

Note that the only information needed to solve this problem was the orientation of the runway and the observer's position relative to it. In other words, a vehicle navigating autonomously would know that if the runway of interest faces north, and the vehicle's direction of travel also is north, the runway should appear vertically in the image. Other relative orientations are handled in a similar manner. The

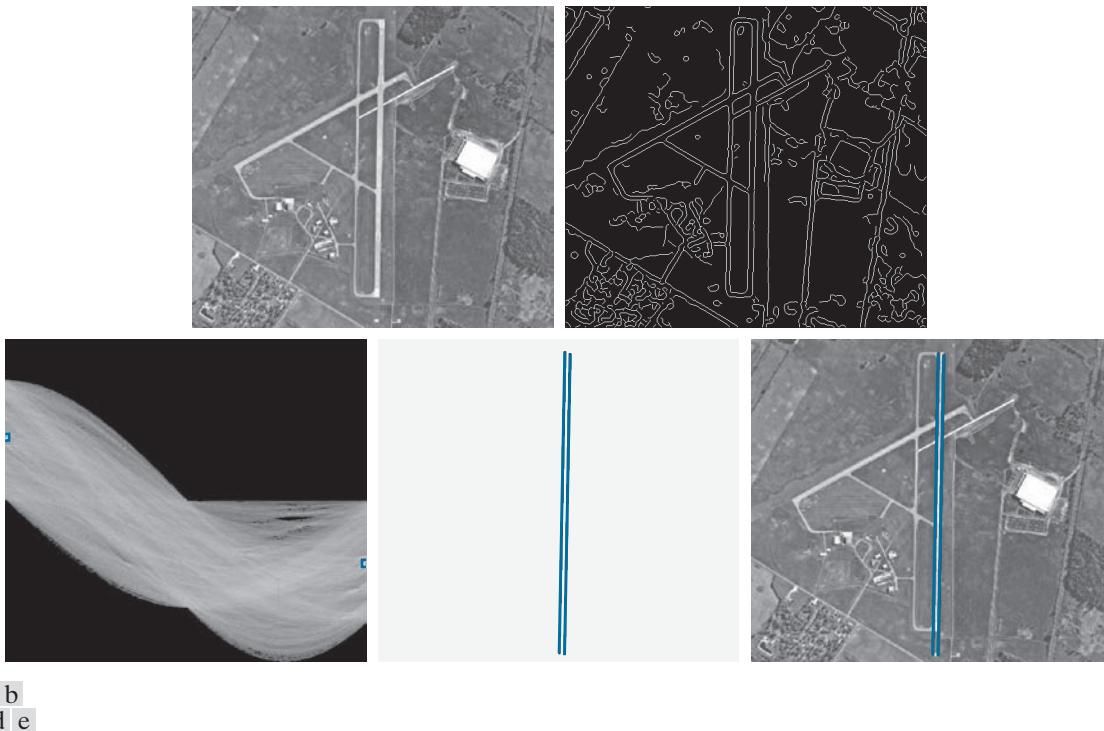


FIGURE 10.31 (a) A 502×564 aerial image of an airport. (b) Edge map obtained using Canny’s algorithm. (c) Hough parameter space (the boxes highlight the points associated with long vertical lines). (d) Lines in the image plane corresponding to the points highlighted by the boxes. (e) Lines superimposed on the original image.

orientations of runways throughout the world are available in flight charts, and the direction of travel is easily obtainable using GPS (Global Positioning System) information. This information also could be used to compute the distance between the vehicle and the runway, thus allowing estimates of parameters such as expected length of lines relative to image size, as we did in this example.

10.3 THRESHOLDING

Because of its intuitive properties, simplicity of implementation, and computational speed, image thresholding enjoys a central position in applications of image segmentation. Thresholding was introduced in Section 3.1, and we have used it in various discussions since then. In this section, we discuss thresholding in a more formal way, and develop techniques that are considerably more general than what has been presented thus far.

FOUNDATION

In the previous section, regions were identified by first finding edge segments, then attempting to link the segments into boundaries. In this section, we discuss

techniques for partitioning images directly into regions based on intensity values and/or properties of these values.

The Basics of Intensity Thresholding

Suppose that the intensity histogram in Fig. 10.32(a) corresponds to an image, $f(x, y)$, composed of light objects on a dark background, in such a way that object and background pixels have intensity values grouped into two dominant modes. One obvious way to extract the objects from the background is to select a threshold, T , that separates these modes. Then, any point (x, y) in the image at which $f(x, y) > T$ is called an *object point*. Otherwise, the point is called a *background point*. In other words, the segmented image, denoted by $g(x, y)$, is given by

$$g(x, y) = \begin{cases} 1 & \text{if } f(x, y) > T \\ 0 & \text{if } f(x, y) \leq T \end{cases} \quad (10-46)$$

When T is a constant applicable over an entire image, the process given in this equation is referred to as *global thresholding*. When the value of T changes over an image, we use the term *variable thresholding*. The terms *local* or *regional* thresholding are used sometimes to denote variable thresholding in which the value of T at any point (x, y) in an image depends on properties of a neighborhood of (x, y) (for example, the average intensity of the pixels in the neighborhood). If T depends on the spatial coordinates (x, y) themselves, then variable thresholding is often referred to as *dynamic* or *adaptive* thresholding. Use of these terms is not universal.

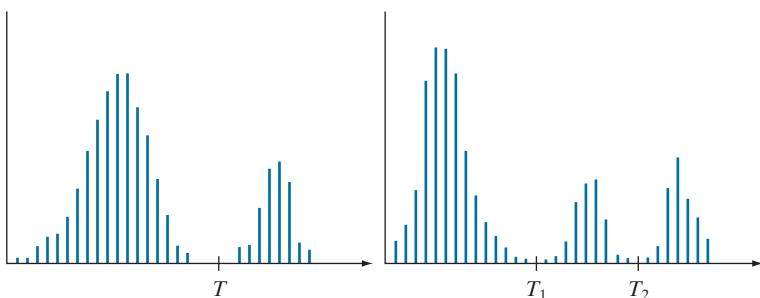
Figure 10.32(b) shows a more difficult thresholding problem involving a histogram with three dominant modes corresponding, for example, to two types of light objects on a dark background. Here, *multiple thresholding* classifies a point (x, y) as belonging to the background if $f(x, y) \leq T_1$, to one object class if $T_1 < f(x, y) \leq T_2$, and to the other object class if $f(x, y) > T_2$. That is, the segmented image is given by

$$g(x, y) = \begin{cases} a & \text{if } f(x, y) > T_2 \\ b & \text{if } T_1 < f(x, y) \leq T_2 \\ c & \text{if } f(x, y) \leq T_1 \end{cases} \quad (10-47)$$

a | b

FIGURE 10.32

Intensity histograms that can be partitioned
(a) by a single threshold, and
(b) by dual thresholds.



where a , b , and c are any three distinct intensity values. We will discuss dual thresholding later in this section. Segmentation problems requiring more than two thresholds are difficult (or often impossible) to solve, and better results usually are obtained using other methods, such as variable thresholding, as will be discussed later in this section, or region growing, as we will discuss in Section 10.4.

Based on the preceding discussion, we may infer intuitively that the success of intensity thresholding is related directly to the width and depth of the valley(s) separating the histogram modes. In turn, the key factors affecting the properties of the valley(s) are: (1) the separation between peaks (the further apart the peaks are, the better the chances of separating the modes); (2) the noise content in the image (the modes broaden as noise increases); (3) the relative sizes of objects and background; (4) the uniformity of the illumination source; and (5) the uniformity of the reflectance properties of the image.

The Role of Noise in Image Thresholding

The simple synthetic image in Fig. 10.33(a) is free of noise, so its histogram consists of two “spike” modes, as Fig. 10.33(d) shows. Segmenting this image into two regions is a trivial task: we just select a threshold anywhere between the two modes. Figure 10.33(b) shows the original image corrupted by Gaussian noise of zero mean and a standard deviation of 10 intensity levels. The modes are broader now

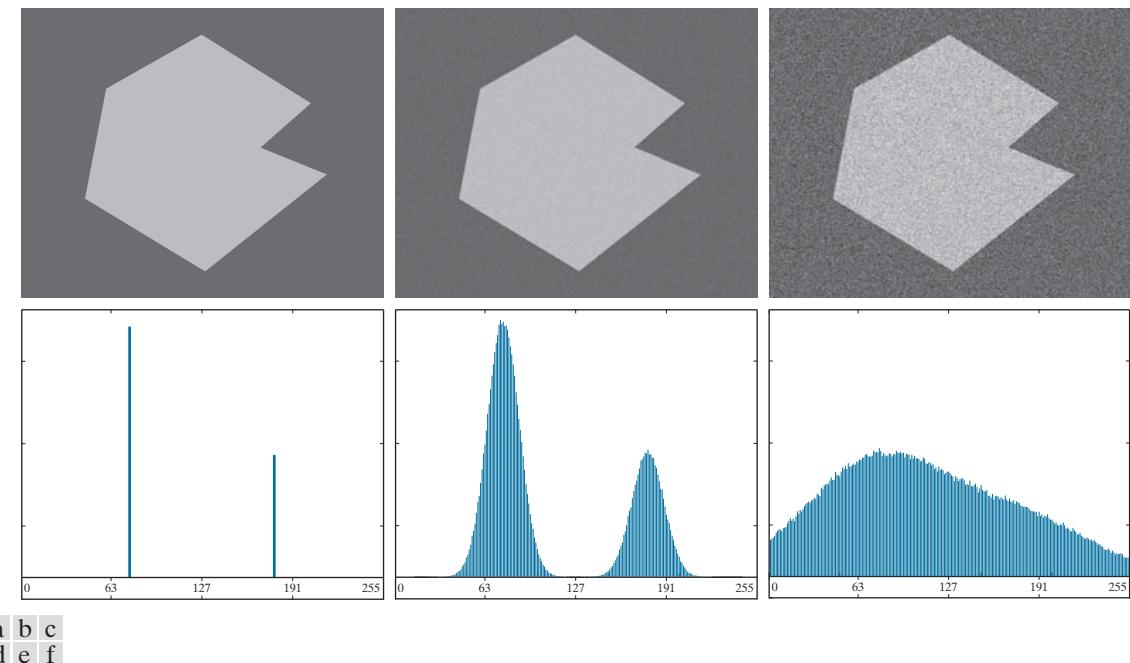


FIGURE 10.33 (a) Noiseless 8-bit image. (b) Image with additive Gaussian noise of mean 0 and standard deviation of 10 intensity levels. (c) Image with additive Gaussian noise of mean 0 and standard deviation of 50 intensity levels. (d) through (f) Corresponding histograms.

[see Fig. 10.33(e)], but their separation is enough so that the depth of the valley between them is sufficient to make the modes easy to separate. A threshold placed midway between the two peaks would do the job. Figure 10.33(c) shows the result of corrupting the image with Gaussian noise of zero mean and a standard deviation of 50 intensity levels. As the histogram in Fig. 10.33(f) shows, the situation is much more serious now, as there is no way to differentiate between the two modes. Without additional processing (such as the methods discussed later in this section) we have little hope of finding a suitable threshold for segmenting this image.

The Role of Illumination and Reflectance in Image Thresholding

Figure 10.34 illustrates the effect that illumination can have on the histogram of an image. Figure 10.34(a) is the noisy image from Fig. 10.33(b), and Fig. 10.34(d) shows its histogram. As before, this image is easily segmentable with a single threshold. With reference to the image formation model discussed in Section 2.3, suppose that we multiply the image in Fig. 10.34(a) by a nonuniform intensity function, such as the intensity ramp in Fig. 10.37(b), whose histogram is shown in Fig. 10.34(e). Figure 10.34(c) shows the product of these two images, and Fig. 10.34(f) is the resulting histogram. The deep valley between peaks was corrupted to the point where separation of the modes without additional processing (to be discussed later in this section) is no longer possible. Similar results would be obtained if the illumination was

In theory, the histogram of a ramp image is uniform. In practice, the degree of uniformity depends on the size of the image and number of intensity levels.

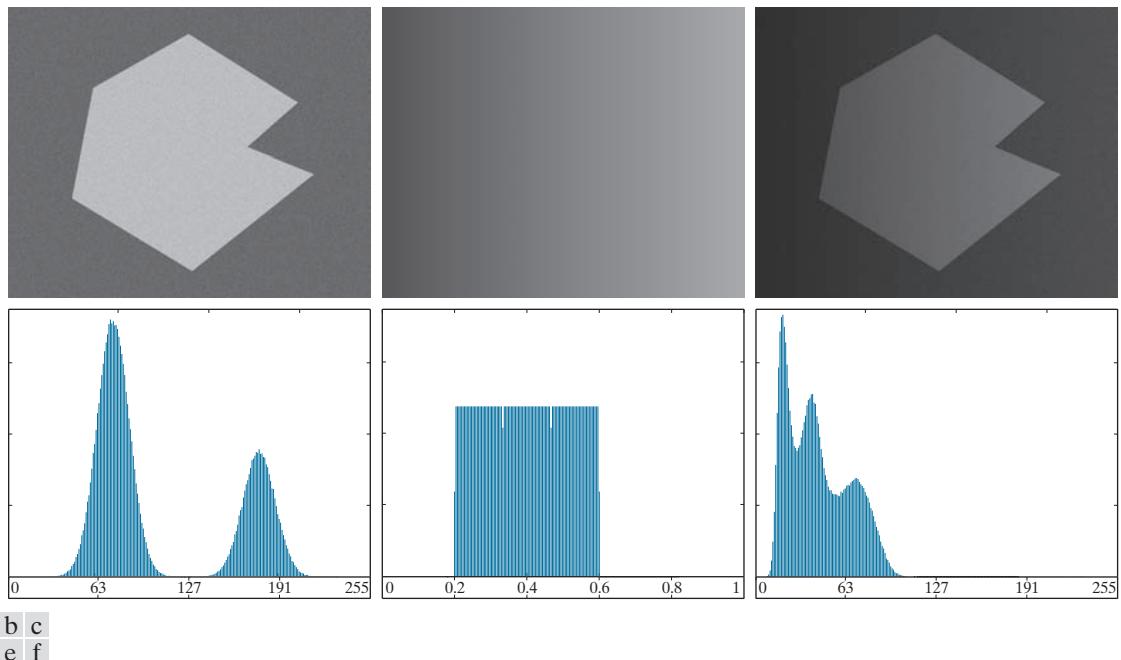


FIGURE 10.34 (a) Noisy image. (b) Intensity ramp in the range [0.2, 0.6]. (c) Product of (a) and (b). (d) through (f) Corresponding histograms.

perfectly uniform, but the reflectance of the image was not, as a result, for example, of natural reflectivity variations in the surface of objects and/or background.

The important point is that illumination and reflectance play a central role in the success of image segmentation using thresholding or other segmentation techniques. Therefore, controlling these factors when possible should be the first step considered in the solution of a segmentation problem. There are three basic approaches to the problem when control over these factors is not possible. The first is to correct the shading pattern directly. For example, nonuniform (but fixed) illumination can be corrected by multiplying the image by the inverse of the pattern, which can be obtained by imaging a flat surface of constant intensity. The second is to attempt to correct the global shading pattern via processing using, for example, the top-hat transformation introduced in Section 9.8. The third approach is to “work around” nonuniformities using variable thresholding, as discussed later in this section.

BASIC GLOBAL THRESHOLDING

When the intensity distributions of objects and background pixels are sufficiently distinct, it is possible to use a single (*global*) threshold applicable over the entire image. In most applications, there is usually enough variability between images that, even if global thresholding is a suitable approach, an algorithm capable of estimating the threshold value for each image is required. The following iterative algorithm can be used for this purpose:

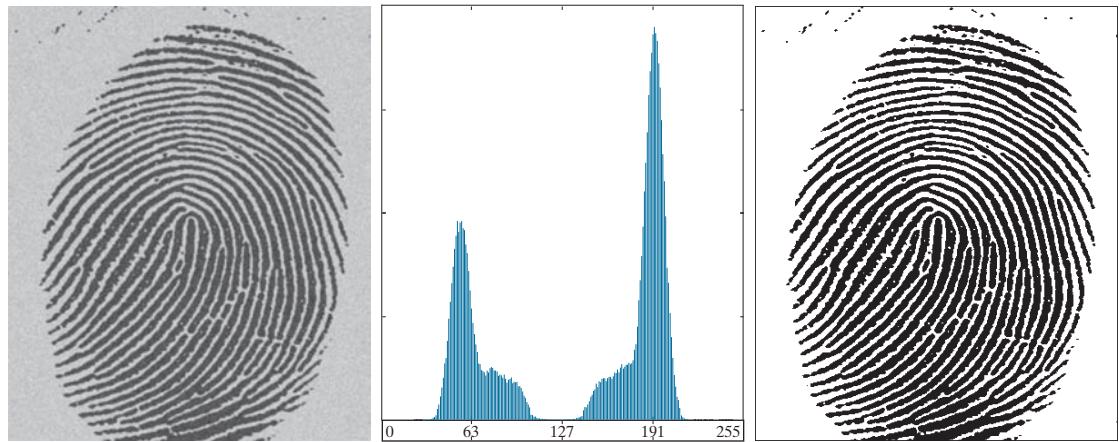
1. Select an initial estimate for the global threshold, T .
2. Segment the image using T in Eq. (10-46). This will produce two groups of pixels: G_1 , consisting of pixels with intensity values $> T$; and G_2 , consisting of pixels with values $\leq T$.
3. Compute the average (mean) intensity values m_1 and m_2 for the pixels in G_1 and G_2 , respectively.
4. Compute a new threshold value midway between m_1 and m_2 :

$$T = \frac{1}{2}(m_1 + m_2)$$

5. Repeat Steps 2 through 4 until the difference between values of T in successive iterations is smaller than a predefined value, ΔT .

The algorithm is stated here in terms of successively thresholding the input image and calculating the means at each step, because it is more intuitive to introduce it in this manner. However, it is possible to develop an equivalent (and more efficient) procedure by expressing all computations in the terms of the image histogram, which has to be computed only once (see Problem 10.29).

The preceding algorithm works well in situations where there is a reasonably clear valley between the modes of the histogram related to objects and background. Parameter ΔT is used to stop iterating when the changes in threshold values is small. The initial threshold must be chosen greater than the minimum and less than the maximum intensity level in the image (the average intensity of the image is a good



a b c

FIGURE 10.35 (a) Noisy fingerprint. (b) Histogram. (c) Segmented result using a global threshold (thin image border added for clarity). (Original image courtesy of the National Institute of Standards and Technology.).

initial choice for T). If this condition is met, the algorithm converges in a finite number of steps, whether or not the modes are separable (see Problem 10.30).

EXAMPLE 10.13: Global thresholding.

Figure 10.35 shows an example of segmentation using the preceding iterative algorithm. Figure 10.35(a) is the original image and Fig. 10.35(b) is the image histogram, showing a distinct valley. Application of the basic global algorithm resulted in the threshold $T = 125.4$ after three iterations, starting with T equal to the average intensity of the image, and using $\Delta T = 0$. Figure 10.35(c) shows the result obtained using $T = 125$ to segment the original image. As expected from the clear separation of modes in the histogram, the segmentation between object and background was perfect.

OPTIMUM GLOBAL THRESHOLDING USING OTSU'S METHOD

Thresholding may be viewed as a statistical-decision theory problem whose objective is to minimize the average error incurred in assigning pixels to two or more groups (also called *classes*). This problem is known to have an elegant closed-form solution known as the *Bayes decision function* (see Section 12.4). The solution is based on only two parameters: the probability density function (PDF) of the intensity levels of each class, and the probability that each class occurs in a given application. Unfortunately, estimating PDFs is not a trivial matter, so the problem usually is simplified by making workable assumptions about the form of the PDFs, such as assuming that they are Gaussian functions. Even with simplifications, the process of implementing solutions using these assumptions can be complex and not always well-suited for real-time applications.

The approach in the following discussion, called *Otsu's method* (Otsu [1979]), is an attractive alternative. The method is optimum in the sense that it maximizes the

between-class variance, a well-known measure used in statistical discriminant analysis. The basic idea is that properly thresholded classes should be distinct with respect to the intensity values of their pixels and, conversely, that a threshold giving the best separation between classes in terms of their intensity values would be the best (optimum) threshold. In addition to its optimality, Otsu's method has the important property that it is based entirely on computations performed on the histogram of an image, an easily obtainable 1-D array (see Section 3.3).

Let $\{0, 1, 2, \dots, L - 1\}$ denote the set of L distinct integer intensity levels in a digital image of size $M \times N$ pixels, and let n_i denote the number of pixels with intensity i . The total number, MN , of pixels in the image is $MN = n_0 + n_1 + n_2 + \dots + n_{L-1}$. The normalized histogram (see Section 3.3) has components $p_i = n_i/MN$, from which it follows that

$$\sum_{i=0}^{L-1} p_i = 1 \quad p_i \geq 0 \quad (10-48)$$

Now, suppose that we select a threshold $T(k) = k$, $0 < k < L - 1$, and use it to threshold the input image into two classes, c_1 and c_2 , where c_1 consists of all the pixels in the image with intensity values in the range $[0, k]$ and c_2 consists of the pixels with values in the range $[k + 1, L - 1]$. Using this threshold, the probability, $P_1(k)$, that a pixel is assigned to (i.e., thresholded into) class c_1 is given by the cumulative sum

$$P_1(k) = \sum_{i=0}^k p_i \quad (10-49)$$

Viewed another way, this is the probability of class c_1 occurring. For example, if we set $k = 0$, the probability of class c_1 having any pixels assigned to it is zero. Similarly, the probability of class c_2 occurring is

$$P_2(k) = \sum_{i=k+1}^{L-1} p_i = 1 - P_1(k) \quad (10-50)$$

From Eq. (3-25), the *mean intensity* value of the pixels in c_1 is

$$\begin{aligned} m_1(k) &= \sum_{i=0}^k i P(i/c_1) = \sum_{i=0}^k i P(c_1/i) P(i)/P(c_1) \\ &= \frac{1}{P_1(k)} \sum_{i=0}^k i p_i \end{aligned} \quad (10-51)$$

where $P_1(k)$ is given by Eq. (10-49). The term $P(i/c_1)$ in Eq. (10-51) is the probability of intensity value i , given that i comes from class c_1 . The rightmost term in the first line of the equation follows from Bayes' formula:

$$P(A/B) = P(B/A)P(A)/P(B)$$

The second line follows from the fact that $P(c_1/i)$, the probability of c_1 given i , is 1 because we are dealing only with values of i from class c_1 . Also, $P(i)$ is the probability of the i th value, which is the i th component of the histogram, p_i . Finally, $P(c_1)$ is the probability of class c_1 which, from Eq. (10-49), is equal to $P_1(k)$.

Similarly, the *mean intensity* value of the pixels assigned to class c_2 is

$$\begin{aligned} m_2(k) &= \sum_{i=k+1}^{L-1} i P(i/c_2) \\ &= \frac{1}{P_2(k)} \sum_{i=k+1}^{L-1} i p_i \end{aligned} \quad (10-52)$$

The *cumulative mean* (average intensity) up to level k is given by

$$m(k) = \sum_{i=0}^k i p_i \quad (10-53)$$

and the average intensity of the entire image (i.e., the *global* mean) is given by

$$m_G = \sum_{i=0}^{L-1} i p_i \quad (10-54)$$

The validity of the following two equations can be verified by direct substitution of the preceding results:

$$P_1 m_1 + P_2 m_2 = m_G \quad (10-55)$$

and

$$P_1 + P_2 = 1 \quad (10-56)$$

where we have omitted the ks temporarily in favor of notational clarity.

In order to evaluate the effectiveness of the threshold at level k , we use the normalized, dimensionless measure

$$\eta = \frac{\sigma_B^2}{\sigma_G^2} \quad (10-57)$$

where σ_G^2 is the *global variance* [i.e., the intensity variance of all the pixels in the image, as given in Eq. (3-26)],

$$\sigma_G^2 = \sum_{i=0}^{L-1} (i - m_G)^2 p_i \quad (10-58)$$

and σ_B^2 is the *between-class variance*, defined as

$$\sigma_B^2 = P_1 (m_1 - m_G)^2 + P_2 (m_2 - m_G)^2 \quad (10-59)$$

This expression can also be written as

$$\begin{aligned} \sigma_B^2 &= P_1 P_2 (m_1 - m_2)^2 \\ &= \frac{(m_G P_1 - m)^2}{P_1 (1 - P_1)} \end{aligned} \quad (10-60)$$

The second step in this equation makes sense only if P_1 is greater than 0 and less than 1, which, in view of Eq. (10-56), implies that P_2 must satisfy the same condition.

The first line of this equation follows from Eqs. (10-55), (10-56), and (10-59). The second line follows from Eqs. (10-50) through (10-54). This form is slightly more efficient computationally because the global mean, m_G , is computed only once, so only two parameters, m_1 and P_1 , need to be computed for any value of k .

The first line in Eq. (10-60) indicates that the farther the two means m_1 and m_2 are from each other, the larger σ_B^2 will be, implying that the between-class variance is a measure of separability between classes. Because σ_G^2 is a constant, it follows that η also is a measure of separability, and maximizing this metric is equivalent to maximizing σ_B^2 . The objective, then, is to determine the threshold value, k , that maximizes the between-class variance, as stated earlier. Note that Eq. (10-57) assumes implicitly that $\sigma_G^2 > 0$. This variance can be zero only when all the intensity levels in the image are the same, which implies the existence of only one class of pixels. This in turn means that $\eta = 0$ for a constant image because the separability of a single class from itself is zero.

Reintroducing k , we have the final results:

$$\eta(k) = \frac{\sigma_B^2(k)}{\sigma_G^2} \quad (10-61)$$

and

$$\sigma_B^2(k) = \frac{[m_G P_1(k) - m(k)]^2}{P_1(k)[1 - P_1(k)]} \quad (10-62)$$

Then, the optimum threshold is the value, k^* , that maximizes $\sigma_B^2(k)$:

$$\sigma_B^2(k^*) = \max_{0 \leq k \leq L-1} \sigma_B^2(k) \quad (10-63)$$

To find k^* we simply evaluate this equation for all *integer* values of k (subject to the condition $0 < P_1(k) < 1$) and select the value of k that yielded the maximum $\sigma_B^2(k)$. If the maximum exists for more than one value of k , it is customary to average the various values of k for which $\sigma_B^2(k)$ is maximum. It can be shown (see Problem 10.36) that a maximum always exists, subject to the condition $0 < P_1(k) < 1$. Evaluating Eqs. (10-62) and (10-63) for all values of k is a relatively inexpensive computational procedure, because the maximum number of integer values that k can have is L , which is only 256 for 8-bit images.

Once k^* has been obtained, input image $f(x, y)$ is segmented as before:

$$g(x, y) = \begin{cases} 1 & \text{if } f(x, y) > k^* \\ 0 & \text{if } f(x, y) \leq k^* \end{cases} \quad (10-64)$$

for $x = 0, 1, 2, \dots, M - 1$ and $y = 0, 1, 2, \dots, N - 1$. Note that all the quantities needed to evaluate Eq. (10-62) are obtained using only the histogram of $f(x, y)$. In addition to the optimum threshold, other information regarding the segmented image can be extracted from the histogram. For example, $P_1(k^*)$ and $P_2(k^*)$, the class probabilities evaluated at the optimum threshold, indicate the portions of the areas occupied by the classes (groups of pixels) in the thresholded image. Similarly, the means $m_1(k^*)$ and $m_2(k^*)$ are estimates of the average intensity of the classes in the original image.

In general, the measure in Eq.(10-61) has values in the range

$$0 \leq \eta(k) \leq 1 \quad (10-65)$$

for values of k in the range $[0, L - 1]$. When evaluated at the optimum threshold k^* , this measure is a quantitative estimate of the separability of classes, which in turn gives us an idea of the accuracy of thresholding a given image with k^* . The lower bound in Eq. (10-65) is attainable only by images with a single, constant intensity level. The upper bound is attainable only by two-valued images with intensities equal to 0 and $L - 1$ (see Problem 10.37).

Otsu's algorithm may be summarized as follows:

1. Compute the normalized histogram of the input image. Denote the components of the histogram by $p_i, i = 0, 1, 2, \dots, L - 1$.
2. Compute the cumulative sums, $P_1(k)$, for $k = 0, 1, 2, \dots, L - 1$, using Eq. (10-49).
3. Compute the cumulative means, $m(k)$, for $k = 0, 1, 2, \dots, L - 1$, using Eq. (10-53).
4. Compute the global mean, m_G , using Eq. (10-54).
5. Compute the between-class variance term, $\sigma_B^2(k)$, for $k = 0, 1, 2, \dots, L - 1$, using Eq. (10-62).
6. Obtain the Otsu threshold, k^* , as the value of k for which $\sigma_B^2(k)$ is maximum. If the maximum is not unique, obtain k^* by averaging the values of k corresponding to the various maxima detected.
7. Compute the global variance, σ_G^2 , using Eq. (10-58), and then obtain the separability measure, η^* , by evaluating Eq. (10-61) with $k = k^*$.

The following example illustrates the use of this algorithm.

EXAMPLE 10.14: Optimum global thresholding using Otsu's method.

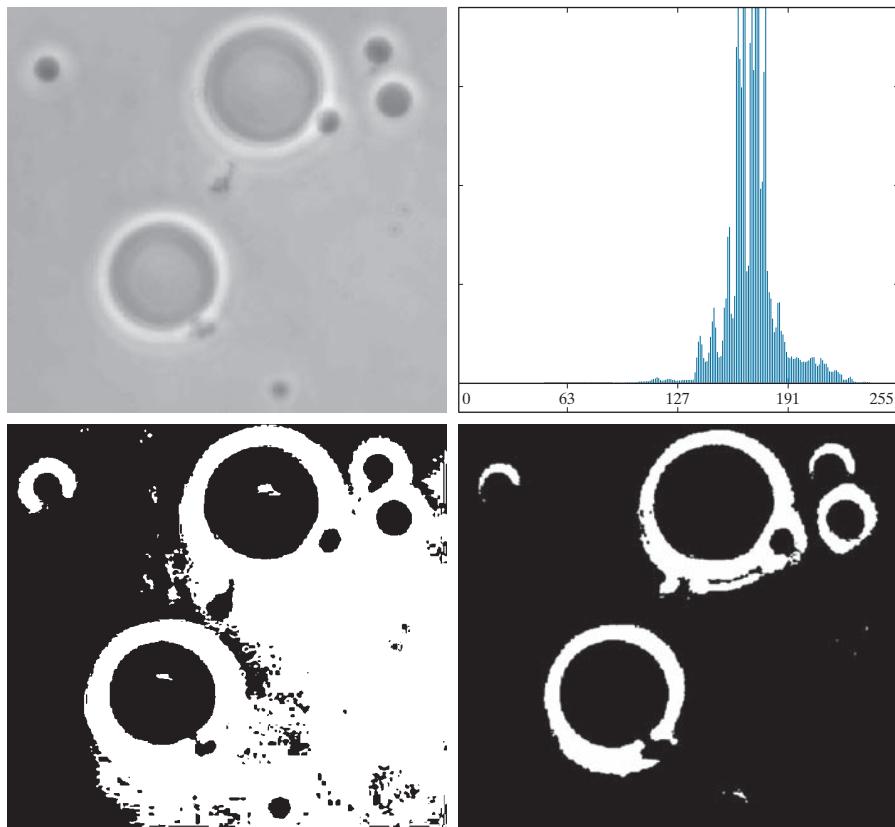
Figure 10.36(a) shows an optical microscope image of polymersome cells. These are cells artificially engineered using polymers. They are invisible to the human immune system and can be used, for example, to deliver medication to targeted regions of the body. Figure 10.36(b) shows the image histogram. The objective of this example is to segment the molecules from the background. Figure 10.36(c) is the result of using the basic global thresholding algorithm discussed earlier. Because the histogram has no distinct valleys and the intensity difference between the background and objects is small, the algorithm failed to achieve the desired segmentation. Figure 10.36(d) shows the result obtained using Otsu's method. This result obviously is superior to Fig. 10.36(c). The threshold value computed by the basic algorithm was 169, while the threshold computed by Otsu's method was 182, which is closer to the lighter areas in the image defining the cells. The separability measure η^* was 0.467.

As a point of interest, applying Otsu's method to the fingerprint image in Example 10.13 yielded a threshold of 125 and a separability measure of 0.944. The threshold is identical to the value (rounded to the nearest integer) obtained with the basic algorithm. This is not unexpected, given the nature of the histogram. In fact, the separability measure is high because of the relatively large separation between modes and the deep valley between them.

a	b
c	d

FIGURE 10.36

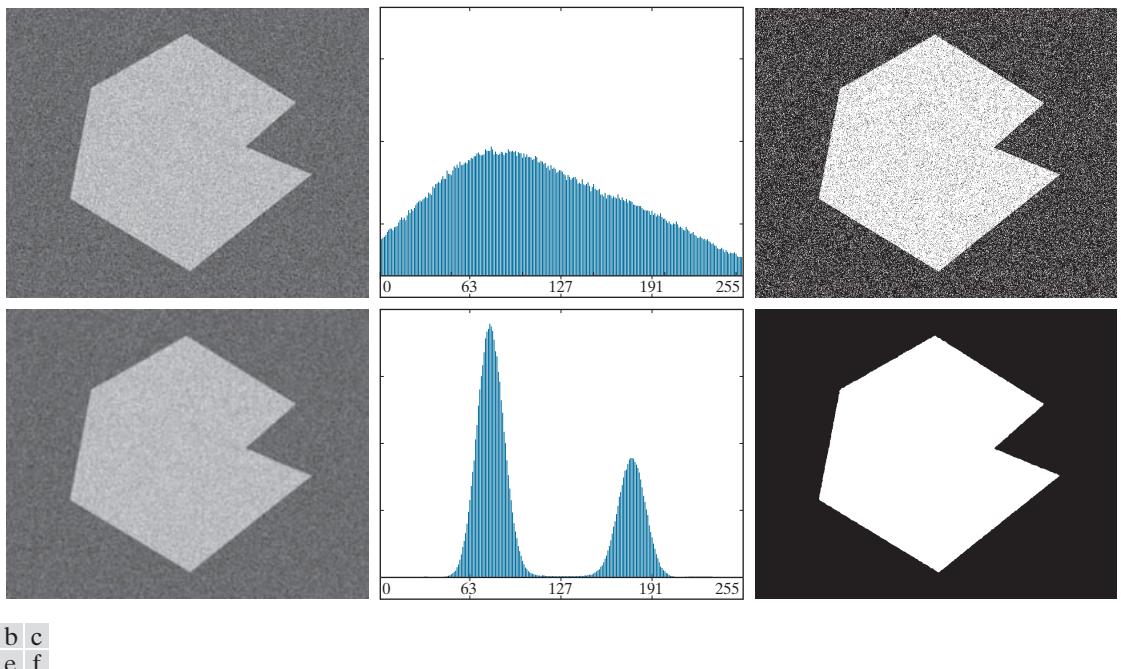
- (a) Original image.
 (b) Histogram (high peaks were clipped to highlight details in the lower values).
 (c) Segmentation result using the basic global algorithm from Section 10.3.
 (d) Result using Otsu's method.
 (Original image courtesy of Professor Daniel A. Hammer, the University of Pennsylvania.)



USING IMAGE SMOOTHING TO IMPROVE GLOBAL THRESHOLDING

As illustrated in Fig. 10.33, noise can turn a simple thresholding problem into an unsolvable one. When noise cannot be reduced at the source, and thresholding is the preferred segmentation method, a technique that often enhances performance is to smooth the image prior to thresholding. We illustrate this approach with an example.

Figure 10.37(a) is the image from Fig. 10.33(c), Fig. 10.37(b) shows its histogram, and Fig. 10.37(c) is the image thresholded using Otsu's method. Every black point in the white region and every white point in the black region is a thresholding error, so the segmentation was highly unsuccessful. Figure 10.37(d) shows the result of smoothing the noisy image with an averaging kernel of size 5×5 (the image is of size 651×814 pixels), and Fig. 10.37(e) is its histogram. The improvement in the shape of the histogram as a result of smoothing is evident, and we would expect thresholding of the smoothed image to be nearly perfect. Figure 10.37(f) shows this to be the case. The slight distortion of the boundary between object and background in the segmented, smoothed image was caused by the blurring of the boundary. In fact, the more aggressively we smooth an image, the more boundary errors we should anticipate in the segmented result.



a b c
d e f

FIGURE 10.37 (a) Noisy image from Fig. 10.33(c) and (b) its histogram. (c) Result obtained using Otsu’s method. (d) Noisy image smoothed using a 5×5 averaging kernel and (e) its histogram. (f) Result of thresholding using Otsu’s method.

Next, we investigate the effect of severely reducing the size of the foreground region with respect to the background. Figure 10.38(a) shows the result. The noise in this image is additive Gaussian noise with zero mean and a standard deviation of 10 intensity levels (as opposed to 50 in the previous example). As Fig. 10.38(b) shows, the histogram has no clear valley, so we would expect segmentation to fail, a fact that is confirmed by the result in Fig. 10.38(c). Figure 10.38(d) shows the image smoothed with an averaging kernel of size 5×5 , and Fig. 10.38(e) is the corresponding histogram. As expected, the net effect was to reduce the spread of the histogram, but the distribution still is unimodal. As Fig. 10.38(f) shows, segmentation failed again. The reason for the failure can be traced to the fact that the region is so small that its contribution to the histogram is insignificant compared to the intensity spread caused by noise. In situations such as this, the approach discussed in the following section is more likely to succeed.

USING EDGES TO IMPROVE GLOBAL THRESHOLDING

Based on the discussion thus far, we conclude that the chances of finding a “good” threshold are enhanced considerably if the histogram peaks are tall, narrow, symmetric, and separated by deep valleys. One approach for improving the shape of histograms is to consider only those pixels that lie on or near the edges between

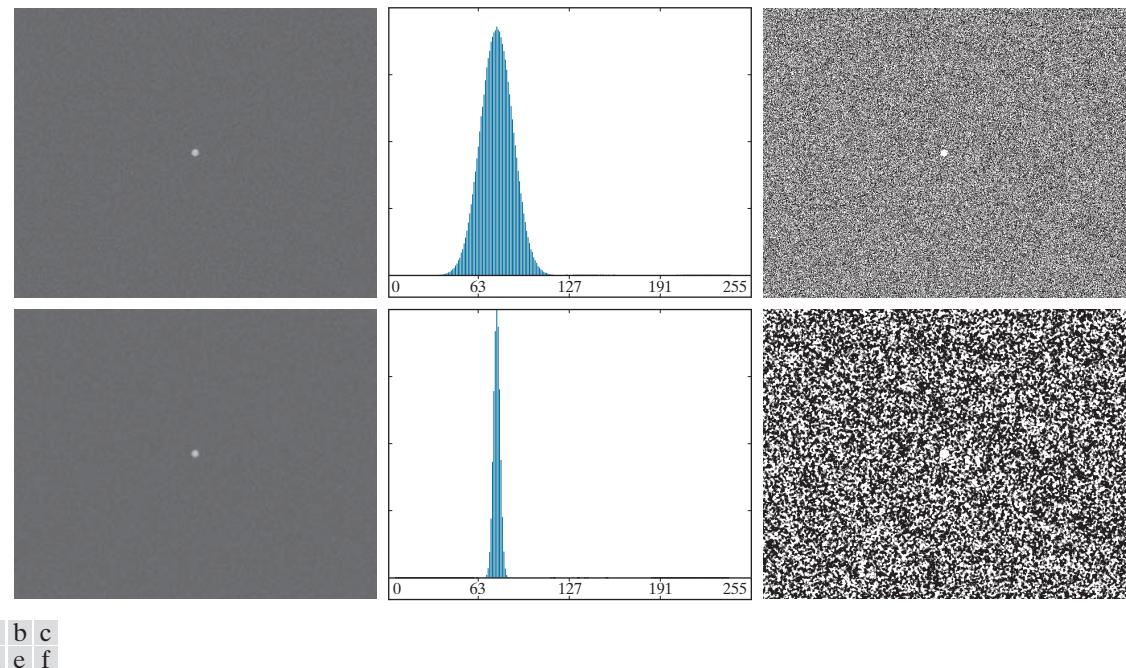


FIGURE 10.38 (a) Noisy image and (b) its histogram. (c) Result obtained using Otsu’s method. (d) Noisy image smoothed using a 5×5 averaging kernel and (e) its histogram. (f) Result of thresholding using Otsu’s method. Thresholding failed in both cases to extract the object of interest. (See Fig. 10.39 for a better solution.)

objects and the background. An immediate and obvious improvement is that histograms should be less dependent on the relative sizes of objects and background. For instance, the histogram of an image composed of a small object on a large background area (or vice versa) would be dominated by a large peak because of the high concentration of one type of pixels. We saw in Fig. 10.38 that this can lead to failure in thresholding.

If only the pixels on or near the edges between objects and background were used, the resulting histogram would have peaks of approximately the same height. In addition, the probability that any of those pixels lies on an object would be approximately equal to the probability that it lies on the background, thus improving the symmetry of the histogram modes. Finally, as indicated in the following paragraph, using pixels that satisfy some simple measures based on gradient and Laplacian operators has a tendency to deepen the valley between histogram peaks.

The approach just discussed assumes that the edges between objects and background are known. This information clearly is not available during segmentation, as finding a division between objects and background is precisely what segmentation aims to do. However, an indication of whether a pixel is on an edge may be obtained by computing its gradient or Laplacian. For example, the average value of the Laplacian is 0 at the transition of an edge (see Fig. 10.10), so the valleys of

histograms formed from the pixels selected by a Laplacian criterion can be expected to be sparsely populated. This property tends to produce the desirable deep valleys discussed above. In practice, comparable results typically are obtained using either the gradient or Laplacian images, with the latter being favored because it is computationally more attractive and is also created using an isotropic edge detector.

The preceding discussion is summarized in the following algorithm, where $f(x, y)$ is the input image:

1. Compute an edge image as either the magnitude of the gradient, or absolute value of the Laplacian, of $f(x, y)$ using any of the methods in Section 10.2.
2. Specify a threshold value, T .
3. Threshold the image from Step 1 using T from Step 2 to produce a binary image, $g_T(x, y)$. This image is used as a mask image in the following step to select pixels from $f(x, y)$ corresponding to “strong” edge pixels in the mask.
4. Compute a histogram using only the pixels in $f(x, y)$ that correspond to the locations of the 1-valued pixels in $g_T(x, y)$.
5. Use the histogram from Step 4 to segment $f(x, y)$ globally using, for example, Otsu’s method.

It is possible to modify this algorithm so that both the magnitude of the gradient and the absolute value of the Laplacian images are used. In this case, we would specify a threshold for each image and form the logical OR of the two results to obtain the marker image. This approach is useful when more control is desired over the points deemed to be valid edge points.

The n th percentile is the smallest number that is greater than $n\%$ of the numbers in a given set. For example, if you received a 95 in a test and this score was greater than 85% of all the students taking the test, then you would be in the 85th percentile with respect to the test scores.

If T is set to any value less than the minimum value of the edge image then, according to Eq. (10-46), $g_T(x, y)$ will consist of all 1’s, implying that all pixels of $f(x, y)$ will be used to compute the image histogram. In this case, the preceding algorithm becomes global thresholding using the histogram of the original image. It is customary to specify the value of T to correspond to a percentile, which typically is set high (e.g., in the high 90’s) so that few pixels in the gradient/Laplacian image will be used in the computation. The following examples illustrate the concepts just discussed. The first example uses the gradient, and the second uses the Laplacian. Similar results can be obtained in both examples using either approach. The important issue is to generate a suitable derivative image.

EXAMPLE 10.15: Using edge information based on the gradient to improve global thresholding.

Figures 10.39(a) and (b) show the image and histogram from Fig. 10.38. You saw that this image could not be segmented by smoothing followed by thresholding. The objective of this example is to solve the problem using edge information. Figure 10.39(c) is the mask image, $g_T(x, y)$, formed as gradient magnitude image thresholded at the 99.7 percentile. Figure 10.39(d) is the image formed by multiplying the mask by the input image. Figure 10.39(e) is the histogram of the nonzero elements in Fig. 10.39(d). Note that this histogram has the important features discussed earlier; that is, it has reasonably symmetrical modes separated by a deep valley. Thus, while the histogram of the original noisy image offered no hope for successful thresholding, the histogram in Fig. 10.39(e) indicates that thresholding of the small object from the background is indeed possible. The result in Fig. 10.39(f) shows that this is the case. This image was generated using Otsu’s method [to obtain a threshold based on the histogram in Fig. 10.42(e)], and then applying the Otsu threshold globally to the noisy image in Fig. 10.39(a). The result is nearly perfect.

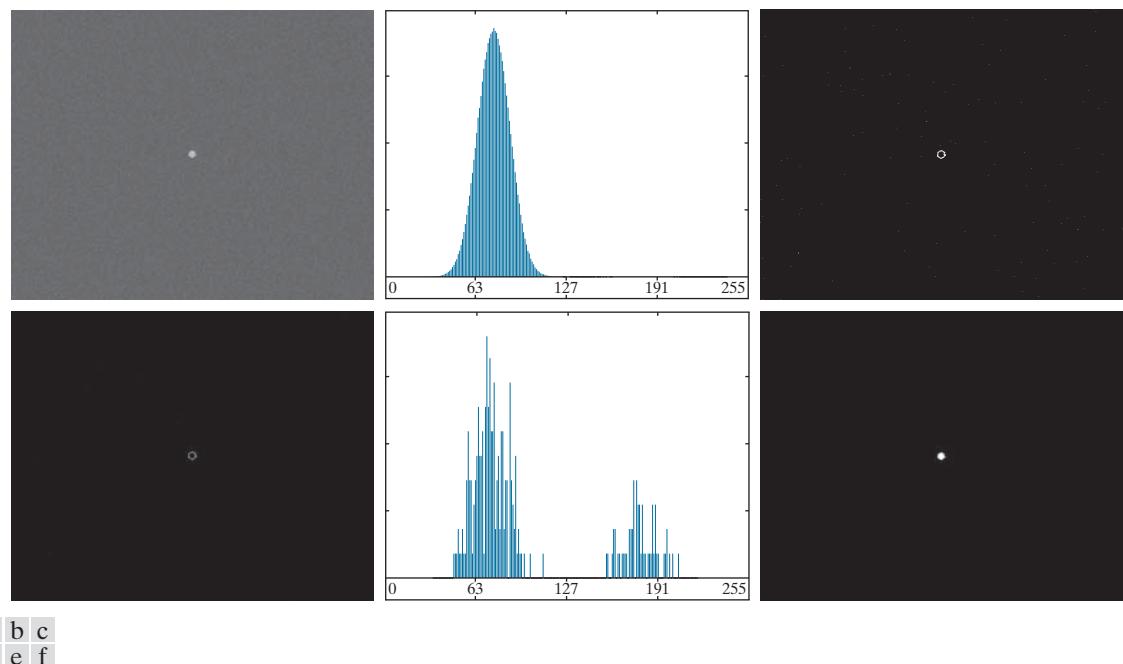


FIGURE 10.39 (a) Noisy image from Fig. 10.38(a) and (b) its histogram. (c) Mask image formed as the gradient magnitude image thresholded at the 99.7 percentile. (d) Image formed as the product of (a) and (c). (e) Histogram of the nonzero pixels in the image in (d). (f) Result of segmenting image (a) with the Otsu threshold based on the histogram in (e). The threshold was 134, which is approximately midway between the peaks in this histogram.

EXAMPLE 10.16: Using edge information based on the Laplacian to improve global thresholding.

In this example, we consider a more complex thresholding problem. Figure 10.40(a) shows an 8-bit image of yeast cells for which we want to use global thresholding to obtain the regions corresponding to the bright spots. As a starting point, Fig. 10.40(b) shows the image histogram, and Fig. 10.40(c) is the result obtained using Otsu's method directly on the image, based on the histogram shown. We see that Otsu's method failed to achieve the original objective of detecting the bright spots. Although the method was able to isolate some of the cell regions themselves, several of the segmented regions on the right were actually joined. The threshold computed by the Otsu method was 42, and the separability measure was 0.636.

Figure 10.40(d) shows the mask image $g_T(x, y)$ obtained by computing the absolute value of the Laplacian image, then thresholding it with T set to 115 on an intensity scale in the range $[0, 255]$. This value of T corresponds approximately to the 99.5 percentile of the values in the absolute Laplacian image, so thresholding at this level results in a sparse set of pixels, as Fig. 10.40(d) shows. Note in this image how the points cluster near the edges of the bright spots, as expected from the preceding discussion. Figure 10.40(e) is the histogram of the nonzero pixels in the product of (a) and (d). Finally, Fig. 10.40(f) shows the result of globally segmenting the original image using Otsu's method based on the histogram in Fig. 10.40(e). This result agrees with the locations of the bright spots in the image. The threshold computed by the Otsu method was 115, and the separability measure was 0.762, both of which are higher than the values obtained by using the original histogram.

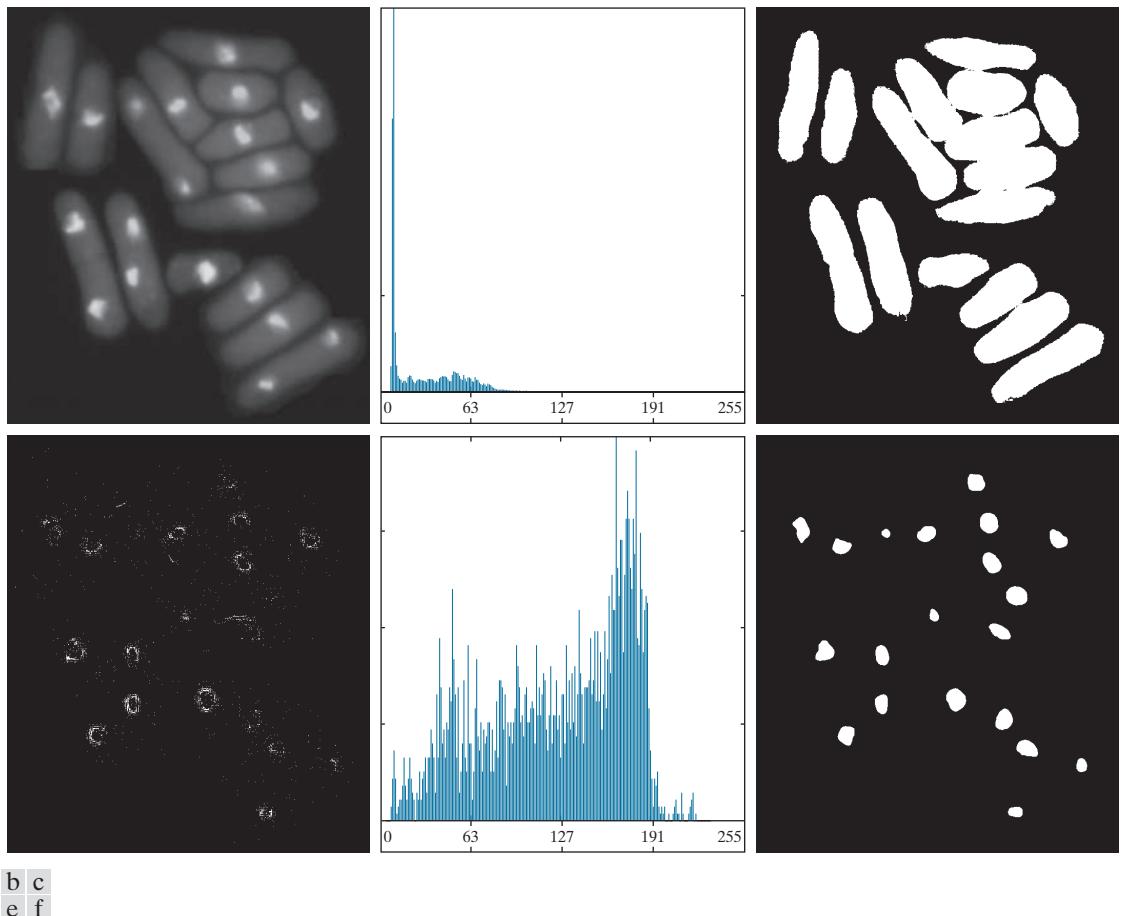


FIGURE 10.40 (a) Image of yeast cells. (b) Histogram of (a). (c) Segmentation of (a) with Otsu’s method using the histogram in (b). (d) Mask image formed by thresholding the absolute Laplacian image. (e) Histogram of the non-zero pixels in the product of (a) and (d). (f) Original image thresholded using Otsu’s method based on the histogram in (e). (Original image courtesy of Professor Susan L. Forsburg, University of Southern California.)

By varying the percentile at which the threshold is set, we can even improve the segmentation of the complete cell regions. For example, Fig. 10.41 shows the result obtained using the same procedure as in the previous paragraph, but with the threshold set at 55, which is approximately 5% of the maximum value of the absolute Laplacian image. This value is at the 53.9 percentile of the values in that image. This result clearly is superior to the result in Fig. 10.40(c) obtained using Otsu’s method with the histogram of the original image.

MULTIPLE THRESHOLDS

Thus far, we have focused attention on image segmentation using a single global threshold. Otsu’s method can be extended to an arbitrary number of thresholds

FIGURE 10.41

Image in Fig. 10.40(a) segmented using the same procedure as explained in Figs. 10.40(d) through (f), but using a lower value to threshold the absolute Laplacian image.



In applications involving more than one variable (for example the RGB components of a color image), thresholding can be implemented using a distance measure, such as the *Euclidean distance*, or *Mahalanobis distance* discussed in Section 6.7 (see Eqs. (6-48), (6-49), and Example 6.15).

because the separability measure on which it is based also extends to an arbitrary number of classes (Fukunaga [1972]). In the case of K classes, c_1, c_2, \dots, c_K , the between-class variance generalizes to the expression

$$\sigma_B^2 = \sum_{k=1}^K P_k (m_k - m_G)^2 \quad (10-66)$$

where

$$P_k = \sum_{i \in c_k} p_i \quad (10-67)$$

and

$$m_k = \frac{1}{P_k} \sum_{i \in c_k} i p_i \quad (10-68)$$

As before, m_G is the global mean given in Eq. (10-54). The K classes are separated by $K - 1$ thresholds whose values, $k_1^*, k_2^*, \dots, k_{K-1}^*$, are the values that maximize Eq. (10-66):

$$\sigma_B^2(k_1^*, k_2^*, \dots, k_{K-1}^*) = \max_{0 < k_1 < k_2 < \dots < k_{K-1}} \sigma_B^2(k_1, k_2, \dots, k_{K-1}) \quad (10-69)$$

Although this result is applicable to an arbitrary number of classes, it begins to lose meaning as the number of classes increases because we are dealing with only one variable (intensity). In fact, the between-class variance usually is cast in terms of multiple variables expressed as vectors (Fukunaga [1972]). In practice, using multiple global thresholding is considered a viable approach when there is reason to believe that the problem can be solved effectively with two thresholds. Applications that require more than two thresholds generally are solved using more than just intensity values. Instead, the approach is to use additional descriptors (e.g., color) and the application is cast as a pattern recognition problem, as you will learn shortly in the discussion on multivariable thresholding.

Recall from the discussion of the Canny edge detector that thresholding with two thresholds is referred to as *hysteresis thresholding*.

For three classes consisting of three intensity intervals (which are separated by two thresholds), the between-class variance is given by:

$$\sigma_B^2 = P_1(m_1 - m_G)^2 + P_2(m_2 - m_G)^2 + P_3(m_3 - m_G)^2 \quad (10-70)$$

where

$$\begin{aligned} P_1 &= \sum_{i=0}^{k_1} p_i \\ P_2 &= \sum_{i=k_1+1}^{k_2} p_i \\ P_3 &= \sum_{i=k_2+1}^{L-1} p_i \end{aligned} \quad (10-71)$$

and

$$\begin{aligned} m_1 &= \frac{1}{P_1} \sum_{i=0}^{k_1} i p_i \\ m_2 &= \frac{1}{P_2} \sum_{i=k_1+1}^{k_2} i p_i \\ m_3 &= \frac{1}{P_3} \sum_{i=k_2+1}^{L-1} i p_i \end{aligned} \quad (10-72)$$

As in Eqs. (10-55) and (10-56), the following relationships hold:

$$P_1 m_1 + P_2 m_2 + P_3 m_3 = m_G \quad (10-73)$$

and

$$P_1 + P_2 + P_3 = 1 \quad (10-74)$$

We see from Eqs. (10-71) and (10-72) that P and m , and therefore σ_B^2 , are functions of k_1 and k_2 . The two optimum threshold values, k_1^* and k_2^* , are the values that maximize $\sigma_B^2(k_1, k_2)$. That is, as indicated in Eq. (10-69), we find the optimum thresholds by finding

$$\sigma_B^2(k_1^*, k_2^*) = \max_{0 < k_1 < k_2 < L-1} \sigma_B^2(k_1, k_2) \quad (10-75)$$

The procedure starts by selecting the first value of k_1 (that value is 1 because looking for a threshold at 0 intensity makes no sense; also, keep in mind that the increment values are integers because we are dealing with integer intensity values). Next, k_2 is incremented through all its values greater than k_1 and less than $L-1$ (i.e., $k_2 = k_1+1, \dots, L-2$). Then, k_1 is incremented to its next value and k_2 is incremented again through all its values greater than k_1 . This procedure is repeated until $k_1 = L-3$. The result of this procedure is a 2-D array, $\sigma_B^2(k_1, k_2)$, and the last step is to look for the maximum value in this array. The values of k_1 and k_2 corresponding to that maximum in the array are the optimum thresholds, k_1^* and k_2^* .

If there are several maxima, the corresponding values of k_1 and k_2 are averaged to obtain the final thresholds. The thresholded image is then given by

$$g(x,y) = \begin{cases} a & \text{if } f(x,y) \leq k_1^* \\ b & \text{if } k_1^* < f(x,y) \leq k_2^* \\ c & \text{if } f(x,y) > k_2^* \end{cases} \quad (10-76)$$

where a , b , and c are any three distinct intensity values.

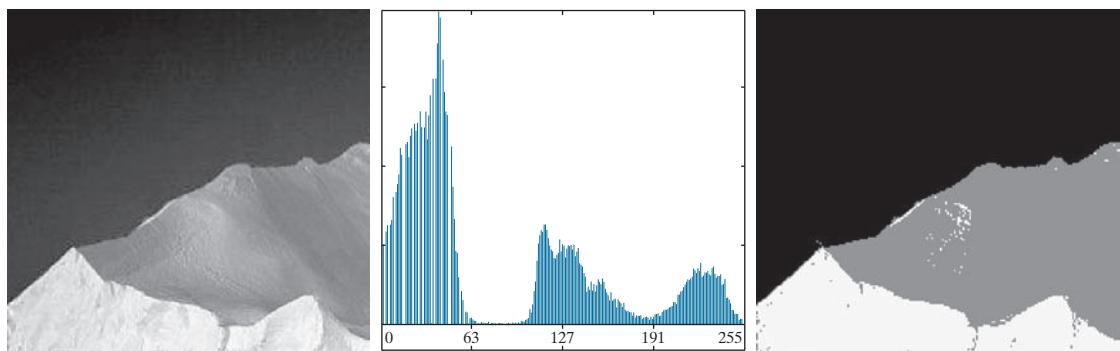
Finally, the separability measure defined earlier for one threshold extends directly to multiple thresholds:

$$\eta(k_1^*, k_2^*) = \frac{\sigma_B^2(k_1^*, k_2^*)}{\sigma_G^2} \quad (10-77)$$

where σ_G^2 is the total image variance from Eq. (10-58).

EXAMPLE 10.17: Multiple global thresholding.

Figure 10.42(a) shows an image of an iceberg. The objective of this example is to segment the image into three regions: the dark background, the illuminated area of the iceberg, and the area in shadows. It is evident from the image histogram in Fig. 10.42(b) that two thresholds are required to solve this problem. The procedure discussed above resulted in the thresholds $k_1^* = 80$ and $k_2^* = 177$, which we note from Fig. 10.45(b) are near the centers of the two histogram valleys. Figure 10.42(c) is the segmentation that resulted using these two thresholds in Eq. (10-76). The separability measure was 0.954. The principal reason this example worked out so well can be traced to the histogram having three distinct modes separated by reasonably wide, deep valleys. But we can do even better using superpixels, as you will see in Section 10.5.



a b c

FIGURE 10.42 (a) Image of an iceberg. (b) Histogram. (c) Image segmented into three regions using dual Otsu thresholds. (Original image courtesy of NOAA.)

VARIABLE THRESHOLDING

As discussed earlier in this section, factors such as noise and nonuniform illumination play a major role in the performance of a thresholding algorithm. We showed that image smoothing and the use of edge information can help significantly. However, sometimes this type of preprocessing is either impractical or ineffective in improving the situation, to the point where the problem cannot be solved by any of the thresholding methods discussed thus far. In such situations, the next level of thresholding complexity involves variable thresholding, as we will illustrate in the following discussion.

Variable Thresholding Based on Local Image Properties

A basic approach to variable thresholding is to compute a threshold at every point, (x, y) , in the image based on one or more specified properties in a neighborhood of (x, y) . Although this may seem like a laborious process, modern algorithms and hardware allow for fast neighborhood processing, especially for common functions such as logical and arithmetic operations.

We illustrate the approach using the mean and standard deviation of the pixel values in a neighborhood of every point in an image. These two quantities are useful for determining local thresholds because, as you know from Chapter 3, they are descriptors of average intensity and contrast. Let m_{xy} and σ_{xy} denote the mean and standard deviation of the set of pixel values in a neighborhood, S_{xy} , centered at coordinates (x, y) in an image (see Section 3.3 regarding computation of the local mean and standard deviation). The following are common forms of variable thresholds based on the local image properties:

$$T_{xy} = a\sigma_{xy} + bm_{xy} \quad (10-78)$$

where a and b are nonnegative constants, and

$$T_{xy} = a\sigma_{xy} + bm_G \quad (10-79)$$

where m_G is the global image mean. The segmented image is computed as

$$g(x, y) = \begin{cases} 1 & \text{if } f(x, y) > T_{xy} \\ 0 & \text{if } f(x, y) \leq T_{xy} \end{cases} \quad (10-80)$$

where $f(x, y)$ is the input image. This equation is evaluated for all pixel locations in the image, and a different threshold is computed at each location (x, y) using the pixels in the neighborhood S_{xy} .

Significant power (with a modest increase in computation) can be added to variable thresholding by using predicates based on the parameters computed in the neighborhood of a point (x, y) :

$$g(x, y) = \begin{cases} 1 & \text{if } Q(\text{local parameters}) \text{ is TRUE} \\ 0 & \text{if } Q(\text{local parameters}) \text{ is FALSE} \end{cases} \quad (10-81)$$

We simplified the notation slightly from the form we used in Eqs. (3-27) and (3-28) by letting xy imply a neighborhood S , centered at coordinates (x, y) .

Note that T_{xy} is a threshold array of the same size as the image from which it was obtained. The threshold at a location (x, y) in the array is used to segment the value of an image at that location.

where Q is a *predicate* based on parameters computed using the pixels in neighborhood S_{xy} . For example, consider the following predicate, $Q(\sigma_{xy}, m_{xy})$, based on the local mean and standard deviation:

$$Q(\sigma_{xy}, m_{xy}) = \begin{cases} \text{TRUE} & \text{if } f(x,y) > a\sigma_{xy} \text{ AND } f(x,y) > bm_{xy} \\ \text{FALSE} & \text{otherwise} \end{cases} \quad (10-82)$$

Note that Eq. (10-80) is a special case of Eq. (10-81), obtained by letting Q be TRUE if $f(x,y) > T_{xy}$ and FALSE otherwise. In this case, the predicate is based simply on the intensity at a point.

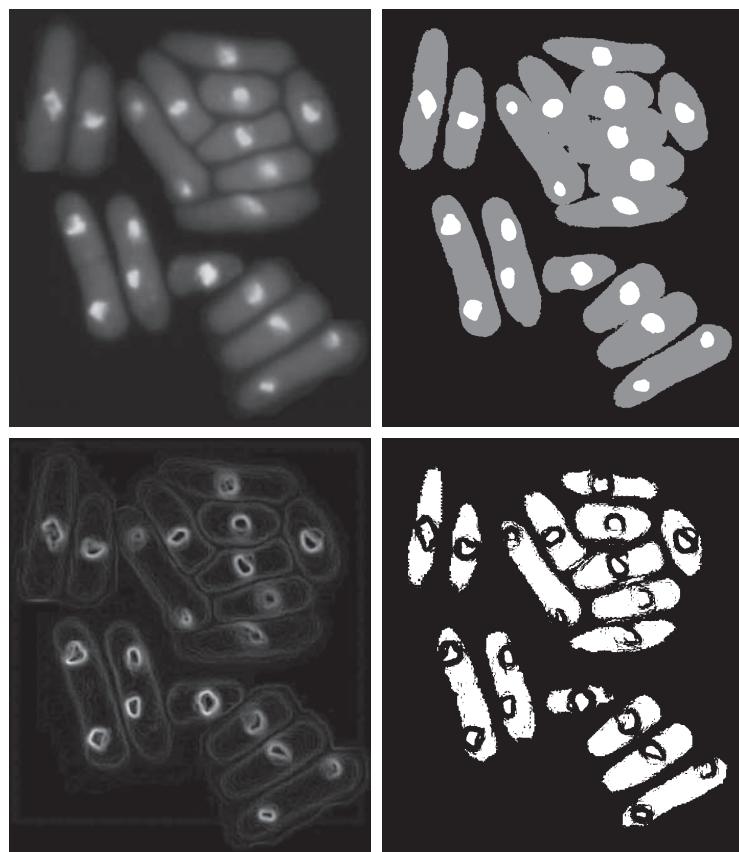
EXAMPLE 10.18: Variable thresholding based on local image properties.

Figure 10.43(a) shows the yeast image from Example 10.16. This image has three predominant intensity levels, so it is reasonable to assume that perhaps dual thresholding could be a good segmentation approach. Figure 10.43(b) is the result of using the dual thresholding method summarized in Eq. (10-76). As the figure shows, it was possible to isolate the bright areas from the background, but the mid-gray regions on the right side of the image were not segmented (i.e., separated) properly. To illustrate the use

a	b
c	d

FIGURE 10.43

- (a) Image from Fig. 10.40.
- (b) Image segmented using the dual thresholding approach given by Eq. (10-76).
- (c) Image of local standard deviations.
- (d) Result obtained using local thresholding.



of local thresholding, we computed the local standard deviation σ_{xy} for all (x, y) in the input image using a neighborhood of size 3×3 . Figure 10.43(c) shows the result. Note how the faint outer lines correctly delineate the boundaries of the cells. Next, we formed a predicate of the form shown in Eq. (10-82), but using the global mean instead of m_{xy} . Choosing the global mean generally gives better results when the background is nearly constant and all the object intensities are above or below the background intensity. The values $a = 30$ and $b = 1.5$ were used to complete the specification of the predicate (these values were determined experimentally, as is usually the case in applications such as this). The image was then segmented using Eq. (10-82). As Fig. 10.43(d) shows, the segmentation was quite successful. Note in particular that all the outer regions were segmented properly, and that most of the inner, brighter regions were isolated correctly.

Variable Thresholding Based on Moving Averages

A special case of the variable thresholding method discussed in the previous section is based on computing a moving average along scan lines of an image. This implementation is useful in applications such as document processing, where speed is a fundamental requirement. The scanning typically is carried out line by line in a zigzag pattern to reduce illumination bias. Let z_{k+1} denote the intensity of the point encountered in the scanning sequence at step $k + 1$. The moving average (mean intensity) at this new point is given by

$$\begin{aligned} m(k+1) &= \frac{1}{n} \sum_{i=k+2-n}^{k+1} z_i && \text{for } k \geq n-1 \\ &= m(k) + \frac{1}{n} (z_{k+1} - z_{k-n}) && \text{for } k \geq n+1 \end{aligned} \quad (10-83)$$

where n is the number of points used in computing the average, and $m(1) = z_1$. The conditions imposed on k are so that all subscripts on z_k are positive. All this means is that n points must be available for computing the average. When k is less than the limits shown (this happens near the image borders) the averages are formed with the available image points. Because a moving average is computed for every point in the image, segmentation is implemented using Eq. (10-80) with $T_{xy} = cm_{xy}$, where c is positive scalar, and m_{xy} is the moving average from Eq. (10-83) at point (x, y) in the input image.

EXAMPLE 10.19: Document thresholding using moving averages.

Figure 10.44(a) shows an image of handwritten text shaded by a spot intensity pattern. This form of intensity shading is typical of images obtained using spot illumination (such as a photographic flash). Figure 10.44(b) is the result of segmentation using the Otsu global thresholding method. It is not unexpected that global thresholding could not overcome the intensity variation because the method generally performs poorly when the areas of interest are embedded in a nonuniform illumination field. Figure 10.44(c) shows successful segmentation with local thresholding using moving averages. For images of written material, a rule of thumb is to let n equal five times the average stroke width. In this case, the average width was 4 pixels, so we let $n = 20$ in Eq. (10-83) and used $c = 0.5$.

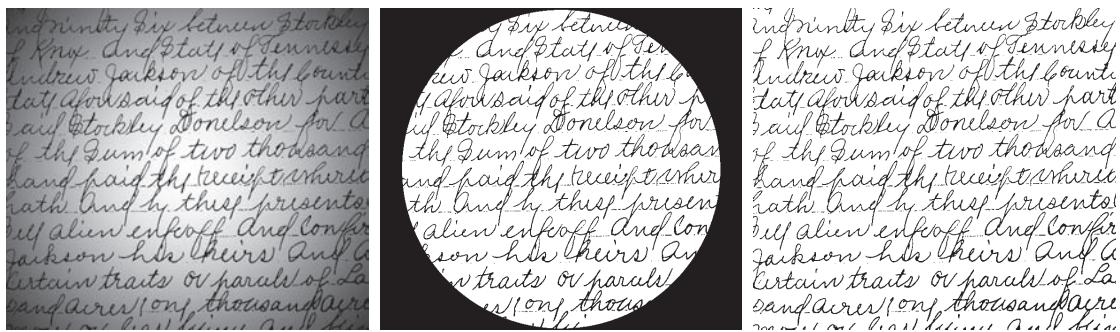


FIGURE 10.44 (a) Text image corrupted by spot shading. (b) Result of global thresholding using Otsu’s method. (c) Result of local thresholding using moving averages.

As another illustration of the effectiveness of this segmentation approach, we used the same parameters as in the previous paragraph to segment the image in Fig. 10.45(a), which is corrupted by a sinusoidal intensity variation typical of the variations that may occur when the power supply in a document scanner is not properly grounded. As Figs. 10.45(b) and (c) show, the segmentation results are comparable to those in Fig. 10.44.

Note that successful segmentation results were obtained in both cases using the same values for n and c , which shows the relative ruggedness of the approach. In general, thresholding based on moving averages works well when the objects of interest are small (or thin) with respect to the image size, a condition satisfied by images of typed or handwritten text.

10.4 SEGMENTATION BY REGION GROWING AND BY REGION SPLITTING AND MERGING |

You should review the terminology introduced in Section 10.1 before proceeding.

As we discussed in Section 10.1, the objective of segmentation is to partition an image into regions. In Section 10.2, we approached this problem by attempting to find boundaries between regions based on discontinuities in intensity levels, whereas in Section 10.3, segmentation was accomplished via thresholds based on the distribution of pixel properties, such as intensity values or color. In this section and in Sections 10.5 and 10.6, we discuss segmentation techniques that find the regions directly. In Section 10.7, we will discuss a method that finds the regions and their boundaries simultaneously.

REGION GROWING

As its name implies, *region growing* is a procedure that groups pixels or subregions into larger regions based on predefined criteria for growth. The basic approach is to start with a set of “seed” points, and from these grow regions by appending to each seed those neighboring pixels that have predefined properties similar to the seed (such as ranges of intensity or color).

Selecting a set of one or more starting points can often be based on the nature of the problem, as we show later in Example 10.20. When a priori information is not



a b c

FIGURE 10.45 (a) Text image corrupted by sinusoidal shading. (b) Result of global thresholding using Otsu’s method. (c) Result of local thresholding using moving averages..

available, the procedure is to compute at every pixel the same set of properties that ultimately will be used to assign pixels to regions during the growing process. If the result of these computations shows clusters of values, the pixels whose properties place them near the centroid of these clusters can be used as seeds.

The selection of similarity criteria depends not only on the problem under consideration, but also on the type of image data available. For example, the analysis of land-use satellite imagery depends heavily on the use of color. This problem would be significantly more difficult, or even impossible, to solve without the inherent information available in color images. When the images are monochrome, region analysis must be carried out with a set of descriptors based on intensity levels and spatial properties (such as moments or texture). We will discuss descriptors useful for region characterization in Chapter 11.

Descriptors alone can yield misleading results if connectivity properties are not used in the region-growing process. For example, visualize a random arrangement of pixels that have three distinct intensity values. Grouping pixels with the same intensity value to form a “region,” without paying attention to connectivity, would yield a segmentation result that is meaningless in the context of this discussion.

Another problem in region growing is the formulation of a stopping rule. Region growth should stop when no more pixels satisfy the criteria for inclusion in that region. Criteria such as intensity values, texture, and color are local in nature and do not take into account the “history” of region growth. Additional criteria that can increase the power of a region-growing algorithm utilize the concept of size, likeness between a candidate pixel and the pixels grown so far (such as a comparison of the intensity of a candidate and the average intensity of the grown region), and the shape of the region being grown. The use of these types of descriptors is based on the assumption that a model of expected results is at least partially available.

Let: $f(x, y)$ denote an input image; $S(x, y)$ denote a *seed* array containing 1’s at the locations of seed points and 0’s elsewhere; and Q denote a *predicate* to be applied at each location (x, y) . Arrays f and S are assumed to be of the same size. A basic region-growing algorithm based on 8-connectivity may be stated as follows.

See Sections 2.5 and 9.5 regarding connected components, and Section 9.2 regarding erosion.

1. Find all connected components in $S(x, y)$ and reduce each connected component to one pixel; label all such pixels found as 1. All other pixels in S are labeled 0.
2. Form an image f_Q such that, at each point (x, y) , $f_Q(x, y) = 1$ if the input image satisfies a given predicate, Q , at those coordinates, and $f_Q(x, y) = 0$ otherwise.
3. Let g be an image formed by appending to each seed point in S all the 1-valued points in f_Q that are 8-connected to that seed point.
4. Label each connected component in g with a different region label (e.g., integers or letters). This is the segmented image obtained by region growing.

The following example illustrates the mechanics of this algorithm.

EXAMPLE 10.20: Segmentation by region growing.

Figure 10.46(a) shows an 8-bit X-ray image of a weld (the horizontal dark region) containing several cracks and porosities (the bright regions running horizontally through the center of the image). We illustrate the use of region growing by segmenting the defective weld regions. These regions could be used in applications such as weld inspection, for inclusion in a database of historical studies, or for controlling an automated welding system.

The first thing we do is determine the seed points. From the physics of the problem, we know that cracks and porosities will attenuate X-rays considerably less than solid welds, so we expect the regions containing these types of defects to be significantly brighter than other parts of the X-ray image. We can extract the seed points by thresholding the original image, using a threshold set at a high percentile. Figure 10.46(b) shows the histogram of the image, and Fig. 10.46(c) shows the thresholded result obtained with a threshold equal to the 99.9 percentile of intensity values in the image, which in this case was 254 (see Section 10.3 regarding percentiles). Figure 10.46(d) shows the result of morphologically eroding each connected component in Fig. 10.46(c) to a single point.

Next, we have to specify a predicate. In this example, we are interested in appending to each seed all the pixels that (a) are 8-connected to that seed, and (b) are “similar” to it. Using absolute intensity differences as a measure of similarity, our predicate applied at each location (x, y) is

$$Q = \begin{cases} \text{TRUE} & \text{if the absolute difference of intensities} \\ & \text{between the seed and the pixel at } (x, y) \text{ is } \leq T \\ \text{FALSE} & \text{otherwise} \end{cases}$$

where T is a specified threshold. Although this predicate is based on intensity differences and uses a single threshold, we could specify more complex schemes in which a different threshold is applied to each pixel, and properties other than differences are used. In this case, the preceding predicate is sufficient to solve the problem, as the rest of this example shows.

From the previous paragraph, we know that all seed values are 255 because the image was thresholded with a threshold of 254. Figure 10.46(e) shows the difference between the seed value (255) and Fig. 10.46(a). The image in Fig. 10.46(e) contains all the differences needed to compute the predicate at each location (x, y) . Figure 10.46(f) shows the corresponding histogram. We need a threshold to use in the predicate to establish similarity. The histogram has three principal modes, so we can start by applying to the difference image the dual thresholding technique discussed in Section 10.3. The resulting two thresholds in this case were $T_1 = 68$ and $T_2 = 126$, which we see correspond closely to the valleys of the histogram. (As a brief digression, we segmented the image using these two thresholds. The result in

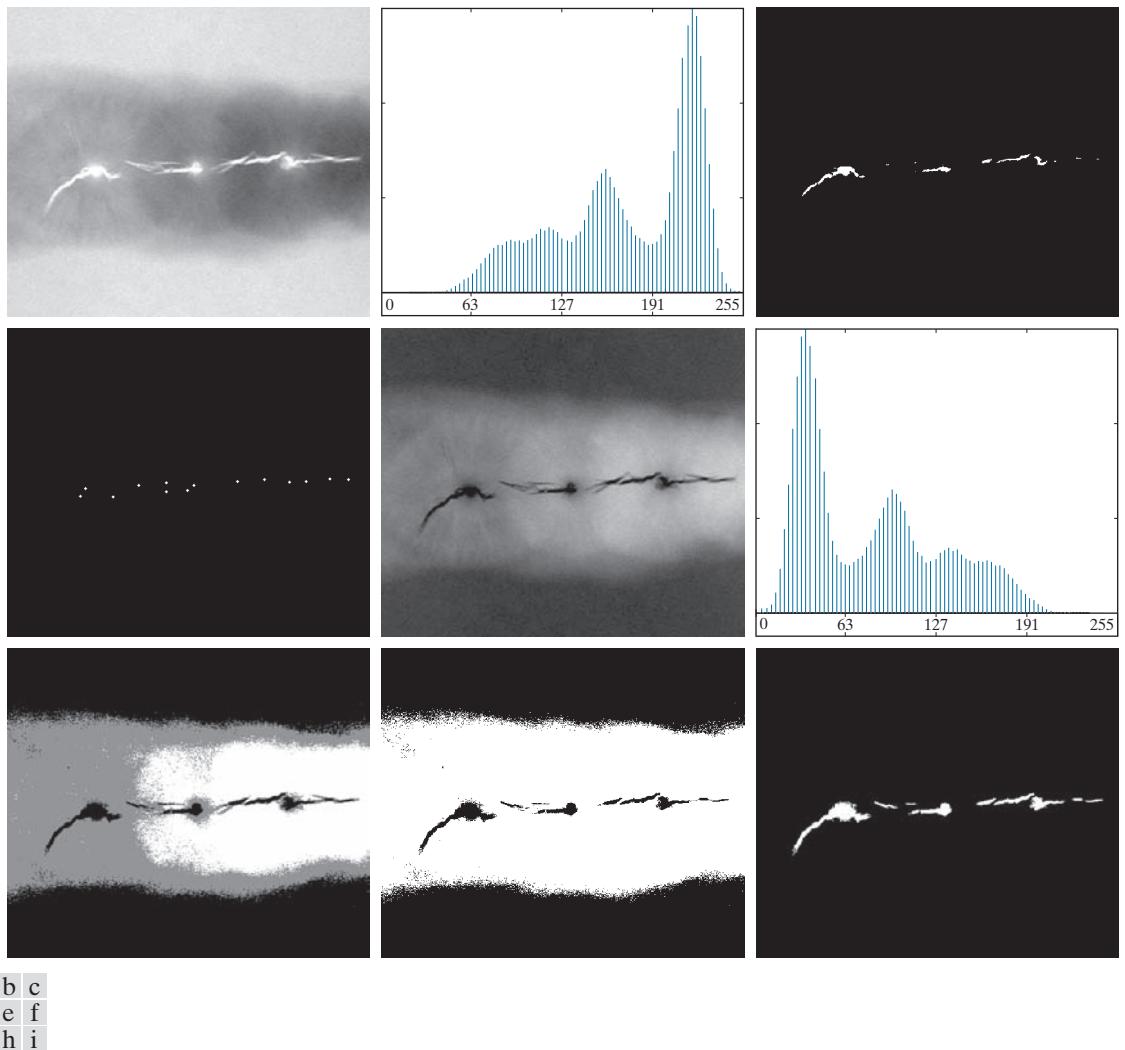


Figure 10.46 (a) X-ray image of a defective weld. (b) Histogram. (c) Initial seed image. (d) Final seed image (the points were enlarged for clarity). (e) Absolute value of the difference between the seed value (255) and (a). (f) Histogram of (e). (g) Difference image thresholded using dual thresholds. (h) Difference image thresholded with the smallest of the dual thresholds. (i) Segmentation result obtained by region growing. (Original image courtesy of X-TEK Systems, Ltd.)

Fig. 10.46(g) shows that segmenting the defects cannot be accomplished using dual thresholds, despite the fact that the thresholds are in the deep valleys of the histogram.)

Figure 10.46(h) shows the result of thresholding the difference image with only T_1 . The black points are the pixels for which the predicate was TRUE; the others failed the predicate. The important result here is that the points in the good regions of the weld failed the predicate, so they will not be included in the final result. The points in the outer region will be considered by the region-growing algorithm as

candidates. However, Step 3 will reject the outer points because they are not 8-connected to the seeds. In fact, as Fig. 10.46(i) shows, this step resulted in the correct segmentation, indicating that the use of connectivity was a fundamental requirement in this case. Finally, note that in Step 4 we used the same value for all the regions found by the algorithm. In this case, it was visually preferable to do so because all those regions have the same physical meaning in this application—they all represent porosities.

REGION SPLITTING AND MERGING

The procedure just discussed grows regions from seed points. An alternative is to subdivide an image initially into a set of disjoint regions and then merge and/or split the regions in an attempt to satisfy the conditions of segmentation stated in Section 10.1. The basics of region splitting and merging are discussed next.

Let R represent the entire image region and select a predicate Q . One approach for segmenting R is to subdivide it successively into smaller and smaller quadrant regions so that, for any region R_i , $Q(R_i) = \text{TRUE}$. We start with the entire region, R . If $Q(R) = \text{FALSE}$, we divide the image into quadrants. If Q is FALSE for any quadrant, we subdivide that quadrant into sub-quadrants, and so on. This splitting technique has a convenient representation in the form of so-called *quadtrees*; that is, trees in which each node has exactly four descendants, as Fig. 10.47 shows (the images corresponding to the nodes of a quadtree sometimes are called *quadregions* or *quadimages*). Note that the root of the tree corresponds to the entire image, and that each node corresponds to the subdivision of a node into four descendant nodes. In this case, only R_4 was subdivided further.

If only splitting is used, the final partition normally contains adjacent regions with identical properties. This drawback can be remedied by allowing *merging* as well as splitting. Satisfying the constraints of segmentation outlined in Section 10.1 requires merging only adjacent regions whose combined pixels satisfy the predicate Q . That is, two adjacent regions R_j and R_k are merged only if $Q(R_j \cup R_k) = \text{TRUE}$.

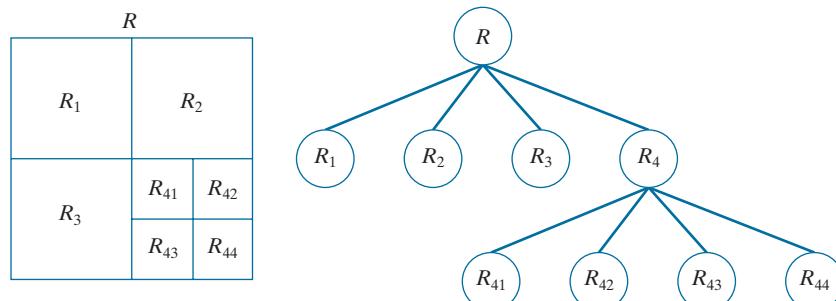
The preceding discussion can be summarized by the following procedure in which, at any step, we

1. Split into four disjoint quadrants any region R_i for which $Q(R_i) = \text{FALSE}$.
2. When no further splitting is possible, merge any adjacent regions R_j and R_k for which $Q(R_j \cup R_k) = \text{TRUE}$.

See Section 2.5
regarding region
adjacency.

a b

FIGURE 10.47
(a) Partitioned
image.
(b) Corresponding
quadtree.
 R represents
the entire image
region.



3. Stop when no further merging is possible.

Numerous variations of this basic theme are possible. For example, a significant simplification results if in Step 2 we allow merging of any two adjacent regions R_j and R_k if each one satisfies the predicate individually. This results in a much simpler (and faster) algorithm, because testing of the predicate is limited to individual quadregions. As the following example shows, this simplification is still capable of yielding good segmentation results.

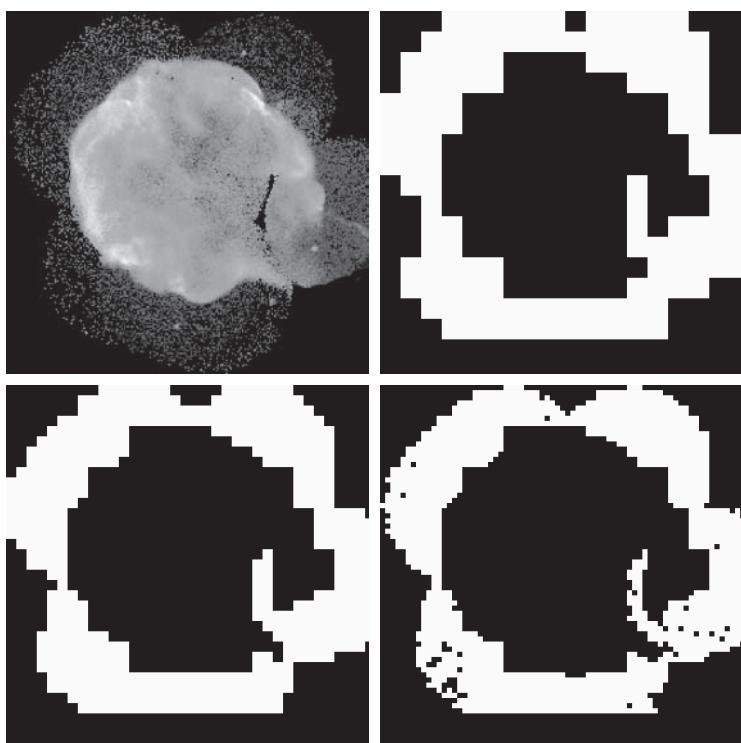
EXAMPLE 10.21: Segmentation by region splitting and merging.

Figure 10.48(a) shows a 566×566 X-ray image of the Cygnus Loop supernova. The objective of this example is to segment (extract from the image) the “ring” of less dense matter surrounding the dense inner region. The region of interest has some obvious characteristics that should help in its segmentation. First, we note that the data in this region has a random nature, indicating that its standard deviation should be greater than the standard deviation of the background (which is near 0) and of the large central region, which is smooth. Similarly, the mean value (average intensity) of a region containing data from the outer ring should be greater than the mean of the darker background and less than the mean of the lighter central region. Thus, we should be able to segment the region of interest using the following predicate:

a b
c d

FIGURE 10.48

(a) Image of the Cygnus Loop supernova, taken in the X-ray band by NASA’s Hubble Telescope.
 (b) through (d)
 Results of limiting the smallest allowed quadregion to be of sizes of 32×32 , 16×16 , and 8×8 pixels,
 respectively.
 (Original image courtesy of NASA.)



$$Q(R) = \begin{cases} \text{TRUE} & \text{if } \sigma_R > a \text{ AND } 0 < m_R < b \\ \text{FALSE} & \text{otherwise} \end{cases}$$

where σ_R and m_R are the standard deviation and mean of the region being processed, and a and b are nonnegative constants.

Analysis of several regions in the outer area of interest revealed that the mean intensity of pixels in those regions did not exceed 125, and the standard deviation was always greater than 10. Figures 10.48(b) through (d) show the results obtained using these values for a and b , and varying the minimum size allowed for the quadregions from 32 to 8. The pixels in a quadregion that satisfied the predicate were set to white; all others in that region were set to black. The best result in terms of capturing the shape of the outer region was obtained using quadregions of size 16×16 . The small black squares in Fig. 10.48(d) are quadregions of size 8×8 whose pixels did not satisfy the predicate. Using smaller quadregions would result in increasing numbers of such black regions. Using regions larger than the one illustrated here would result in a more “block-like” segmentation. Note that in all cases the segmented region (white pixels) was a connected region that completely separates the inner, smoother region from the background. Thus, the segmentation effectively partitioned the image into three distinct areas that correspond to the three principal features in the image: background, a dense region, and a sparse region. Using any of the white regions in Fig. 10.48 as a mask would make it a relatively simple task to extract these regions from the original image (see Problem 10.43). As in Example 10.20, these results could not have been obtained using edge- or threshold-based segmentation.

As used in the preceding example, properties based on the mean and standard deviation of pixel intensities in a region attempt to quantify the texture of the region (see Section 11.3 for a discussion on texture). The concept of texture segmentation is based on using measures of texture in the predicates. In other words, we can perform texture segmentation by any of the methods discussed in this section simply by specifying predicates based on texture content.

10.5 REGION SEGMENTATION USING CLUSTERING AND SUPERPIXELS

In this section, we discuss two related approaches to region segmentation. The first is a classical approach based on seeking clusters in data, related to such variables as intensity and color. The second approach is significantly more modern, and is based on using clustering to extract “superpixels” from an image.

A more general form of clustering is *unsupervised clustering*, in which a clustering algorithm attempts to find a meaningful set of clusters in a given set of samples. We do not address this topic, as our focus in this brief introduction is only to illustrate how *supervised clustering* is used for image segmentation.

REGION SEGMENTATION USING K-MEANS CLUSTERING

The basic idea behind the clustering approach used in this chapter is to partition a set, Q , of observations into a specified number, k , of clusters. In *k-means* clustering, each observation is assigned to the cluster with the nearest mean (hence the name of the method), and each mean is called the *prototype* of its cluster. A *k-means algorithm* is an iterative procedure that successively refines the means until convergence is achieved.

Let $\{\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_Q\}$ be set of vector observations (samples). These vectors have the form

$$\mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{bmatrix} \quad (10-84)$$

In image segmentation, each component of a vector \mathbf{z} represents a numerical pixel attribute. For example, if segmentation is based on just grayscale intensity, then $\mathbf{z} = z$ is a scalar representing the intensity of a pixel. If we are segmenting RGB color images, \mathbf{z} typically is a 3-D vector, each component of which is the intensity of a pixel in one of the three primary color images, as we discussed in Chapter 6. The objective of k -means clustering is to partition the set Q of observations into k ($k \leq Q$) disjoint cluster sets $C = \{C_1, C_2, \dots, C_k\}$, so that the following criterion of optimality is satisfied:[†]

$$\arg \min_C \left(\sum_{i=1}^k \sum_{\mathbf{z} \in C_i} \|\mathbf{z} - \mathbf{m}_i\|^2 \right) \quad (10-85)$$

where \mathbf{m}_i is the *mean vector* (or *centroid*) of the samples in set C_i and $\|\arg\|$ is the vector norm of the argument. Typically, the Euclidean norm is used, so the term $\|\mathbf{z} - \mathbf{m}_i\|^2$ is the familiar *Euclidean distance* from a sample in C_i to mean \mathbf{m}_i . In words, this equation says that we are interested in finding the sets $C = \{C_1, C_2, \dots, C_k\}$ such that the *sum of the distances* from each point in a set to the mean of that set is minimum.

Unfortunately, finding this minimum is an NP-hard problem for which no practical solution is known. As a result, a number of heuristic methods that attempt to find approximations to the minimum have been proposed over the years. In this section, we discuss what is generally considered to be the “standard” k -means algorithm, which is based on the Euclidean distance (see Section 2.6). Given a set $\{\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_Q\}$ of vector observation and a specified value of k , the algorithm is as follows:

These initial means are the initial cluster centers. They are also called *seeds*.

- 1. Initialize the algorithm:** Specify an initial set of means, $\mathbf{m}_i(1)$, $i = 1, 2, \dots, k$.
- 2. Assign samples to clusters:** Assign each sample to the cluster set whose mean is the closest (ties are resolved arbitrarily, but samples are assigned to only *one* cluster):

$$\mathbf{z}_q \rightarrow C_i \text{ if } \|\mathbf{z}_q - \mathbf{m}_i\|^2 < \|\mathbf{z}_q - \mathbf{m}_j\|^2 \quad j = 1, 2, \dots, k \quad (j \neq i); \quad q = 1, 2, \dots, Q$$

- 3. Update the cluster centers (means):**

$$\mathbf{m}_i = \frac{1}{|C_i|} \sum_{\mathbf{z} \in C_i} \mathbf{z} \quad i = 1, 2, \dots, k$$

where $|C_i|$ is the number of samples in cluster set C_i .

- 4. Test for completion:** Compute the Euclidean norms of the differences between the mean vectors in the current and previous steps. Compute the residual error, E , as the sum of the k norms. Stop if $E \leq T$, where T a specified, nonnegative threshold. Else, go back to Step 2.

[†] Remember, $\min_x(h(x))$ is the minimum of h with respect to x , whereas $\arg \min_x(h(x))$ is the value (or values) of x at which h is minimum.

a b

FIGURE 10.49

(a) Image of size 688×688 pixels.
 (b) Image segmented using the k -means algorithm with $k = 3$.



When $T = 0$, this algorithm is known to converge in a finite number of iterations to a local minimum. It is not guaranteed to yield the global minimum required to minimize Eq. (10-85). The result at convergence does depend on the initial values chosen for \mathbf{m}_i . An approach used frequently in data analysis is to specify the initial means as k randomly chosen samples from the given sample set, and to run the algorithm several times, with a new random set of initial samples each time. This is to test the “stability” of the solution. In image segmentation, the important issue is the value selected for k because this determines the number of segmented regions; thus, multiple passes are rarely used.

EXAMPLE 10.22: Using k-means clustering for segmentation.

Figure 10.49(a) shows an image of size 688×688 pixels, and Fig. 10.49(b) is the segmentation obtained using the k -means algorithm with $k = 3$. As you can see, the algorithm was able to extract all the meaningful regions of this image with high accuracy. For example, compare the quality of the characters in both images. It is important to realize that the entire segmentation was done by clustering of a single variable (intensity). Because k -means works with vector observations in general, its power to discriminate between regions increases as the number of components of vector \mathbf{z} in Eq. (10-84) increases.

REGION SEGMENTATION USING SUPERPIXELS

The idea behind *superpixels* is to replace the standard pixel grid by grouping pixels into primitive regions that are more perceptually meaningful than individual pixels. The objectives are to lessen computational load, and to improve the performance of segmentation algorithms by reducing irrelevant detail. A simple example will help explain the basic approach of superpixel representations.

Figure 10.50(a) shows an image of size 600×800 (480,000) pixels containing various levels of detail that could be described verbally as: “This is an image of two large carved figures in the foreground, and at least three, much smaller, carved figures resting on a fence behind the large figures. The figures are on a beach, with

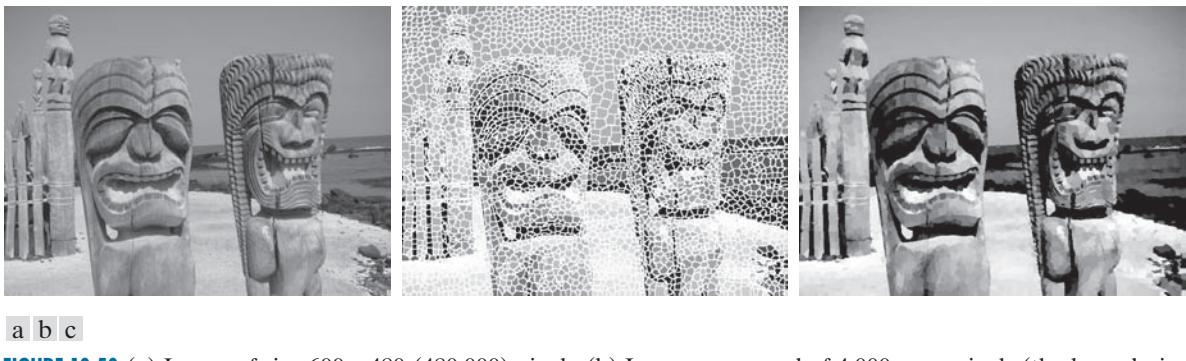


FIGURE 10.50 (a) Image of size 600×480 (480,000) pixels. (b) Image composed of 4,000 superpixels (the boundaries between superpixels (in white) are superimposed on the superpixel image for reference—the boundaries are not part of the data). (c) Superpixel image. (Original image courtesy of the U.S. National Park Services.).

Figures 10.50(b) and (c) were obtained using a method to be discussed later in this section.

the ocean and sky in the background.” Figure 10.50(b) shows the same image represented by 4,000 superpixels and their boundaries (the boundaries are shown for reference—they are not part of the data), and Fig. 10.50(c) shows the superpixel image. One could argue that the level of detail in the superpixel image would lead to the same description as the original, but the former contains only 4,000 primitive units, as opposed to 480,000 in the original. Whether the superpixel representation is “adequate” depends on the application. If the objective is to describe the image at the level of detail mentioned above, then the answer is yes. On the other hand, if the objective is to detect imperfections at pixel-level resolutions, then the answer obviously is no. And there are applications, such as computerized medical diagnosis, in which approximate representations of any kind are not acceptable. Nevertheless, numerous application areas, such as image-database queries, autonomous navigation, and certain branches of robotics, in which economy of implementation and potential improvements in segmentation performance far outweigh any appreciable loss of image detail.

One important requirement of any superpixel representation is *adherence to boundaries*. This means that boundaries between regions of interest must be preserved in a superpixel image. We can see that this indeed is the case with the image in Fig. 10.50(c). Note, for example, how clear the boundaries between the figures and the background are. The same is true of the boundaries between the beach and the ocean, and between the ocean and the sky. Other important characteristics are the preservations of topological properties and, of course, computational efficiency. The superpixel algorithm discussed in this section meets these requirements.

As another illustration, we show the results of severely decreasing the number of superpixels to 1,000, 500, and 250. The results in Fig. 10.51, show a significant loss of detail compared to Fig. 10.50(a), but the first two images contain most of the detail relevant to the image description discussed earlier. A notable difference is that two of the three small carvings on the fence in the back were eliminated. The 250-element superpixel image even lost the third. However, the boundaries between the principal regions, as well as the basic topology of the images, were preserved.

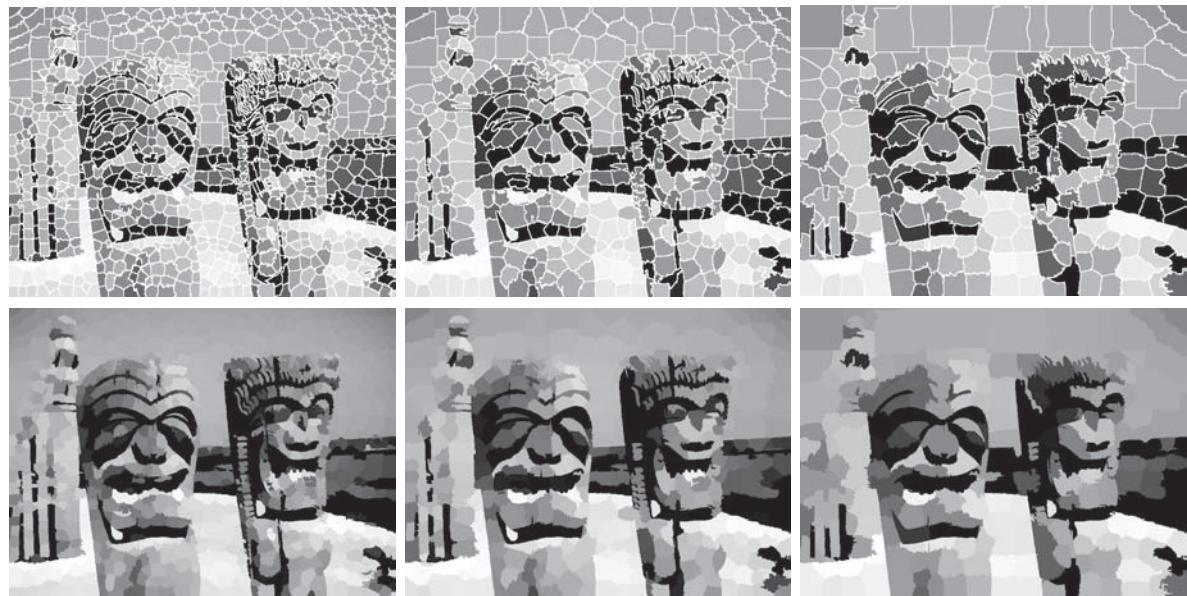


FIGURE 10.51 Top row: Results of using 1,000, 500, and 250 superpixels in the representation of Fig. 10.50(a). As before, the boundaries between superpixels are superimposed on the images for reference. Bottom row: Superpixel images.

SLIC Superpixel Algorithm

In this section we discuss an algorithm for generating superpixels, called *simple linear iterative clustering* (SLIC). This algorithm, developed by Achanta et al. [2012], is conceptually simple, and has computational and other performance advantages over other superpixels techniques. SLIC is a modification of the k -means algorithm discussed in the previous section. SLIC observations typically use (but are not limited to) 5-dimensional vectors containing three color components and two spatial coordinates. For example, if we are using the RGB color system, the 5-dimensional vector associated with an image pixel has the form

As you will learn in
Chapter 11, vectors
containing image
attributes are called
feature vectors.

$$\mathbf{z} = \begin{bmatrix} r \\ g \\ b \\ x \\ y \end{bmatrix} \quad (10-86)$$

where (r, g, b) are the three color components of a pixel, and (x, y) are its two spatial coordinates. Let n_{sp} denote the desired number of superpixels and let n_{tp} denote the total number of pixels in the image. The initial superpixel centers, $\mathbf{m}_i = [r_i \ g_i \ b_i \ x_i \ y_i]^T$, $i = 1, 2, \dots, n_{sp}$, are obtained by sampling the image on a regular grid spaced s units apart. To generate superpixels approximately equal in size (i.e., area), the grid spac-

ing interval is selected as $s = [n_{sp}/n_{sp}]^{1/2}$. To prevent centering a superpixel on the edge of the image, and to reduce the chances of starting at a noisy point, the initial cluster centers are moved to the lowest gradient position in the 3×3 neighborhood about each center.

The SLIC superpixel algorithm consists of the following steps. Keep in mind that superpixels are vectors in general. When we refer to a “pixel” in the algorithm, we are referring to the (x, y) location of the superpixel relative to the image.

- 1. Initialize the algorithm:** Compute the initial superpixel cluster centers,

$$\mathbf{m}_i = [r_i \ g_i \ b_i \ x_i \ y_i]^T, \quad i = 1, 2, \dots, n_{sp}$$

by sampling the image at regular grid steps, s . Move the cluster centers to the lowest gradient position in a 3×3 neighborhood. For each pixel location, p , in the image, set a label $L(p) = -1$ and a distance $d(p) = \infty$.

- 2. Assign samples to cluster centers:** For each cluster center \mathbf{m}_i , $i = 1, 2, \dots, n_{sp}$, compute the distance, $D_i(p)$ between \mathbf{m}_i and each pixel p in a $2s \times 2s$ neighborhood about \mathbf{m}_i . Then, for each p and $i = 1, 2, \dots, n_{sp}$, if $D_i < d(p)$, let $d(p) = D_i$ and $L(p) = i$.
- 3. Update the cluster centers:** Let C_i denote the set of pixels in the image with label $L(p) = i$. Update \mathbf{m}_i :

$$\mathbf{m}_i = \frac{1}{|C_i|} \sum_{\mathbf{z} \in C_i} \mathbf{z} \quad i = 1, 2, \dots, n_{sp}$$

where $|C_i|$ is the number of pixels in set C_i , and the \mathbf{z} 's are given by Eq. (10-86).

- 4. Test for convergence:** Compute the Euclidean norms of the differences between the mean vectors in the current and previous steps. Compute the residual error, E , as the sum of the n_{sp} norms. If $E < T$, where T a specified nonnegative threshold, go to Step 5. Else, go back to Step 2.
- 5. Post-process the superpixel regions:** Replace all the superpixels in each region, C_i , by their average value, \mathbf{m}_i .

Note in Step 5 that superpixels end up as contiguous regions of constant value. The average value is not the only way to compute this constant, but it is the most widely used. For graylevel images, the average is just the average intensity of all the pixels in the region spanned by the superpixel. This algorithm is similar to the k -means algorithm in the previous section, with the exceptions that the distances, D_i , are not specified as Euclidean distances (see below), and that these distances are computed for regions of size $2s \times 2s$, rather than for all the pixels in the image, thus reducing computation time significantly. In practice, SLIC convergence with respect to E can be achieved with fairly large values of T . For example, all results reported by Achanta et al. [2012] were obtained using $T = 10$.

Specifying the Distance Measure

SLIC superpixels correspond to clusters in a space whose coordinates are colors and spatial variables. It would be senseless to use a single Euclidean distance in this case, because the scales in the axes of this coordinate system are different and unrelated. In other words, spatial and color distances must be treated separately. This is accomplished by normalizing the distance of the various components, then combining them into a single measure. Let d_c and d_s denote the color and spatial Euclidean distances between two points in a cluster, respectively:

$$d_c = \left[(r_j - r_i)^2 + (g_j - g_i)^2 + (b_j - b_i)^2 \right]^{1/2} \quad (10-87)$$

and

$$d_s = \left[(x_j - x_i)^2 + (y_j - y_i)^2 \right]^{1/2} \quad (10-88)$$

We then define D as the *composite* distance

$$D = \left[\left(\frac{d_c}{d_{cm}} \right)^2 + \left(\frac{d_s}{d_{sm}} \right)^2 \right]^{1/2} \quad (10-89)$$

where d_{cm} and d_{sm} are the maximum expected values of d_c and d_s . The maximum spatial distance should correspond to the sampling interval; that is, $d_{sm} = s = [n_{lp}/n_{sp}]^{1/2}$. Determining the maximum color distance is not as straightforward, because these distances can vary significantly from cluster to cluster, and from image to image. A solution is to set d_{cm} to a constant c so that Eq. (10-89) becomes

$$D = \left[\left(\frac{d_c}{c} \right)^2 + \left(\frac{d_s}{s} \right)^2 \right]^{1/2} \quad (10-90)$$

We can write this equation as

$$D = \left[d_c^2 + \left(\frac{d_s}{s} \right)^2 c^2 \right]^{1/2} \quad (10-91)$$

This is the distance measure used for each cluster in the algorithm. Constant c can be used to weigh the relative importance between color similarity and spatial proximity. When c is large, spatial proximity is more important, and the resulting superpixels are more compact. When c is small, the resulting superpixels adhere more tightly to image boundaries, but have less regular size and shape.

For grayscale images, as in Example 10.23 below, we use

$$d_c = \left[(l_j - l_i)^2 \right]^{1/2} \quad (10-92)$$

in Eq. (10-91), where the l 's are intensity levels of the points for which the distance is being computed.

In 3-D, superpixels become *supervoxels*, which are handled by defining

$$d_s = \left[(x_j - x_i)^2 + (y_j - y_i)^2 + (z_j - z_i)^2 \right]^{1/2} \quad (10-93)$$

where the z 's are the coordinates of the third spatial dimension. We must also add the third spatial variable, z , to the vector in Eq. (10-86).

Because no provision is made in the algorithm to enforce connectivity, it is possible for isolated pixels to remain after convergence. These are assigned the label of the nearest cluster using a connected components algorithm (see Section 9.6). Although we explained the algorithm in the context of RGB color components, the method is equally applicable to other colors systems. In fact, other components of vector \mathbf{z} in Eq. (10-86) (with the exception of the spatial variables) could be other real-valued feature values, provided that a meaningful distance measure can be defined for them.

EXAMPLE 10.23: Using superpixels for image segmentation.

Figure 10.52(a) shows an image of an iceberg, and Fig. 10.52(b) shows the result of segmenting this image using the k -means algorithm developed in the last section, with $k = 3$. Although the main regions of the image were segmented, there are numerous segmentation errors in both regions of the iceberg, and also on the boundary separating it from the background. Errors are visible as isolated pixels (and also as small groups of pixels) with the wrong shade (e.g., black pixels within a white region). Figure 10.52(c) shows a 100-superpixel representation of the image with the superpixel boundaries superimposed for reference, and Fig. 10.52(d) shows the same image without the boundaries. Figure 10.52(e) is the segmentation of (d) using the k -means algorithm with $k = 3$ as before. Note the significant improvement over the result in (b), indicating that the original image has considerably more (irrelevant) detail than is needed for a proper segmentation. In terms of computational advantage, consider that generating Fig. 10.52(b) required individual processing of over 300K pixels, while (e) required processing of 100 pixels with considerably fewer shades of gray.

10.6 REGION SEGMENTATION USING GRAPH CUTS

In this section, we discuss an approach for partitioning an image into regions by expressing the pixels of the image as nodes of a graph, and then finding an optimum partition (*cut*) of the graph into groups of nodes. Optimality is based on criteria whose values are high for members within a group (i.e., a region) and low across members of different groups. As you will see later in this section, graph-cut segmentation is capable in some cases of results that can be superior to the results achievable by any of the segmentation methods studied thus far. The price of this potential benefit is added complexity in implementation, which generally translates into slower execution.

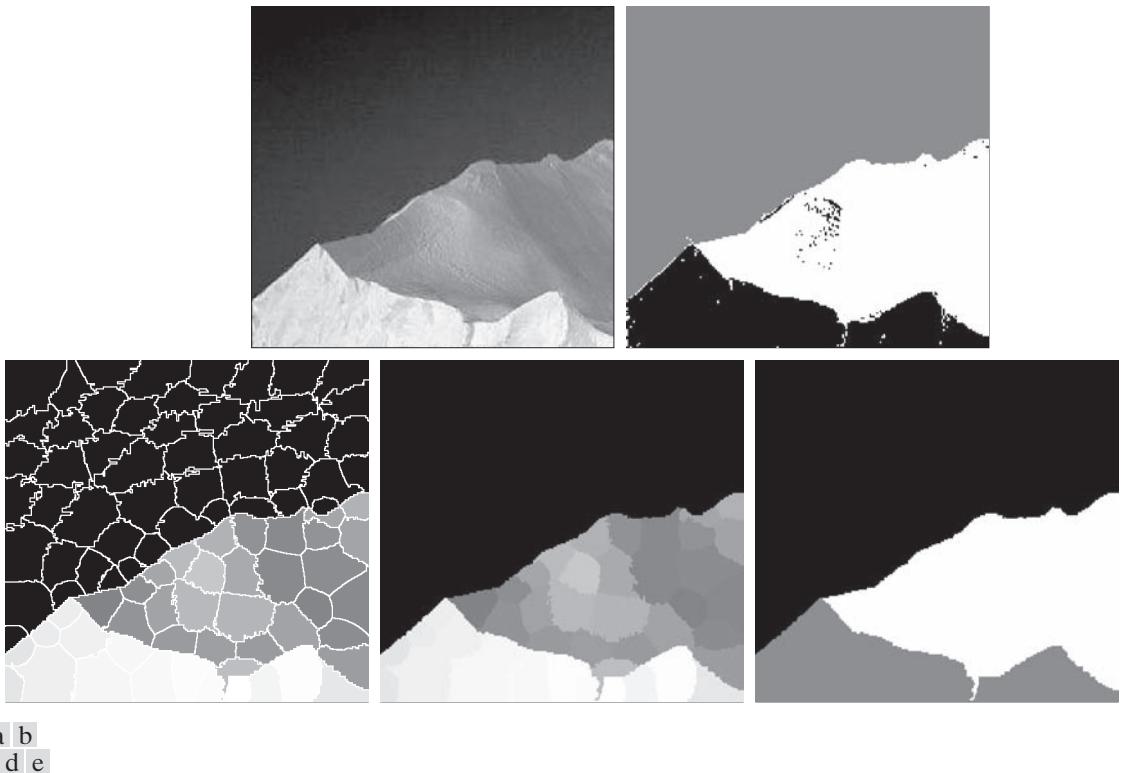


FIGURE 10.52 (a) Image of size 533×566 (301,678) pixels. (b) Image segmented using the k -means algorithm. (c) 100-element superpixel image showing boundaries for reference. (d) Same image without boundaries. (e) Superpixel image (d) segmented using the k -means algorithm. (Original image courtesy of NOAA.)

IMAGES AS GRAPHS

Nodes and edges are also referred to as *vertices* and *links*, respectively.

A graph, G , is a mathematical structure consisting of a set V of *nodes* and a set E of *edges* connecting those vertices:

$$G = (V, E) \quad (10-94)$$

where V is a set and

$$E \subseteq V \times V \quad (10-95)$$

See Section 2.5 for an explanation of the Cartesian product $V \times V$ and for a review of the set symbols used in this section.

is a set of ordered pairs of elements from V . If $(u, v) \in E$ implies that $(v, u) \in E$, and vice versa, the graph is said to be *undirected*; otherwise the graph is *directed*. For example, we may consider a street map as a graph in which the nodes are street intersections, and the edges are the streets connecting those intersections. If all streets are bidirectional, the graph is undirected (meaning that we can travel both ways from any two intersections). Otherwise, if at least one street is a one-way street, the graph is directed.

The types of graphs in which we are interested are undirected graphs whose edges are further characterized by a matrix, \mathbf{W} , whose element $w(i, j)$ is a weight associated with the edge that connects nodes i and j . Because the graph is undirected, $w(i, j) = w(j, i)$, which means that \mathbf{W} is a symmetric matrix. The weights are selected to be proportional to one or more similarity measures between all pairs of nodes. A graph whose edges are associated with weights is called a *weighted graph*.

The essence of the material in this section is to represent an image to be segmented as a weighted, undirected graph, where the nodes of the graph are the pixels in the image, and an edge is formed between every pair of nodes. The weight, $w(i, j)$, of each edge is a function of the similarity between nodes i and j . We then seek to partition the nodes of the graph into disjoint subsets V_1, V_2, \dots, V_K where, by some measure, the similarity among the nodes within a subset is high, and the similarity across the nodes of different subsets is low. The nodes of the partitioned subsets correspond to the regions in the segmented image.

Set V is partitioned into subsets by cutting the graph. A *cut* of a graph is a partition of V into two subsets A and B such that

$$A \cup B = V \text{ and } A \cap B = \emptyset \quad (10-96)$$

where the cut is implemented by removing the edges connecting subgraphs A and B . There are two key aspects of using graph cuts for image segmentation: (1) how to associate a graph with an image; and (2) how to cut the graph in a way that makes sense in terms of partitioning the image into background and foreground (object) pixels. We address these two questions next.

Figure 10.53 shows a simplified approach for generating a graph from an image. The nodes of the graph correspond to the pixels in the image and, to keep the explanation simple, we allow edges only between adjacent pixels using 4-connectivity, which means that there are no diagonal edges linking the pixels. But, keep in mind that, in general, edges are specified between every pair of pixels. The weights for the edges typically are formed from spatial relationships (for example, distance from the vertex pixel) and intensity measures (for example, texture and color), consistent with exhibiting similarity between pixels. In this simple example, we define the degree of similarity between two pixels as the inverse of the difference in their intensities. That is, for two nodes (pixels) n_i and n_j , the weight of the edge between them is $w(i, j) = 1 / (|I(n_i) - I(n_j)| + c)$, where $I(n_i)$ and $I(n_j)$ are the intensities of the two nodes (pixels) and c is a constant included to prevent division by 0. Thus, the closer the values of intensity between adjacent pixels is, the larger the value of w will be.

For illustrative purposes, the thickness of each edge in Fig. 10.53 is shown proportional to the degree of similarity between the pixels that it connects (see Problem 10.44). As you can see in the figure, the edges between the dark pixels are stronger than the edges between dark and light pixels, and vice versa. Conceptually, segmentation is achieved by cutting the graph along its weak edges, as illustrated by the dashed line in Fig. 10.53(d). Figure 10.53(c) shows the segmented image.

Although the basic structure in Fig. 10.53 is the focus of the discussion in this section, we mention for completeness another common approach for constructing

Superpixels are also well suited for use as graph nodes. Thus, when we refer in this section to "pixels" in an image, we are, by implication, also referring to superpixels.

a	b
c	d

FIGURE 10.53

- (a) A 3×3 image.
- (c) A corresponding graph.
- (d) Graph cut.
- (e) Segmented image.

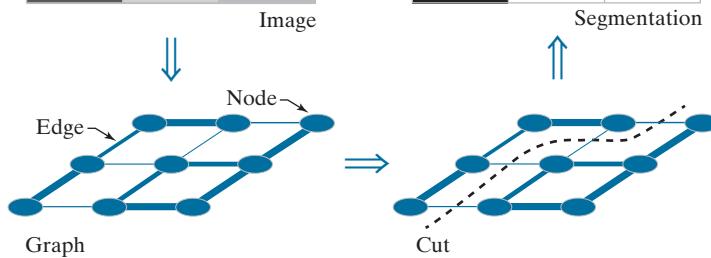
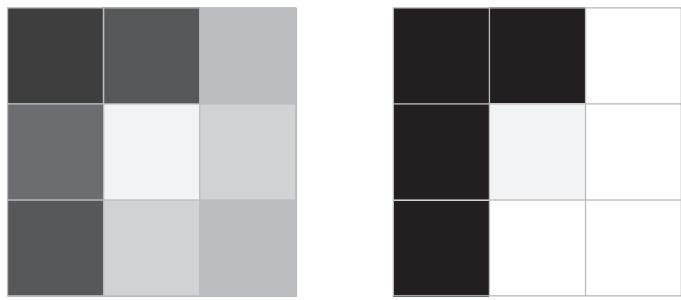


image graphs. Figure 10.54 shows the same graph as the one we just discussed, but here you see two additional nodes called the *source* and *sink terminal nodes*, respectively, each connected to all nodes in the graph via unidirectional links called *t-links*. The terminal nodes are not part of the image; their role, for example, is to associate with each pixel a probability that it is a background or foreground (object) pixel. The probabilities are the weights of the t-links. In Figs. 10.54(c) and (d), the thickness of each t-link is proportional to the value of the probability that the graph node to which it is connected is a foreground or background pixel (the thicknesses shown are so that the segmentation result would be the same as in Fig. 10.53). Which of the two nodes we call background or foreground is arbitrary.

MINIMUM GRAPH CUTS

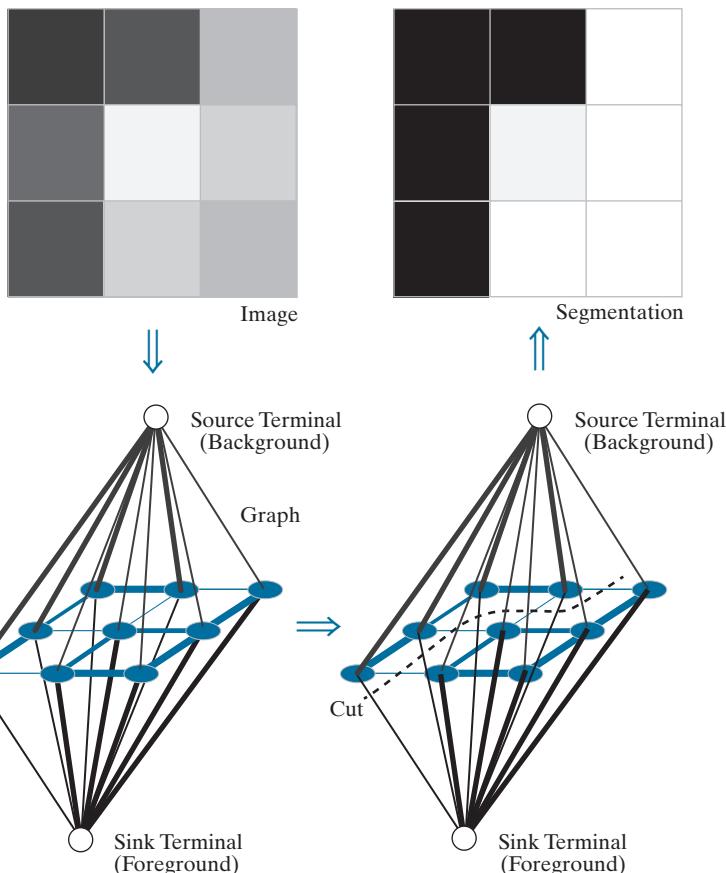
Once an image has been expressed as a graph, the next step is to cut the graph into two or more subgraphs. The nodes (pixels) in each resulting subgraph correspond to a region in the segmented image. Approaches based on Fig. 10.54 rely on interpreting the graph as a flow network (of pipes, for example) and obtaining what is commonly referred to as a *minimum graph cut*. This formulation is based on the so-called *Max-Flow, Min-Cut Theorem*. This theorem states that, in a flow network, the maximum amount of flow passing from the source to the sink is equal to the *minimum cut*. This minimum cut is defined as the smallest *total weight* of the edges that, if removed, would disconnect the sink from the source:

$$\text{cut}(A, B) = \sum_{u \in A, v \in B} w(u, v) \quad (10-97)$$

a	b
c	d

FIGURE 10.54

- (a) Same image as in Fig. 10.53(a).
 (c) Corresponding graph and terminal nodes. (d) Graph cut. (b) Segmented image.

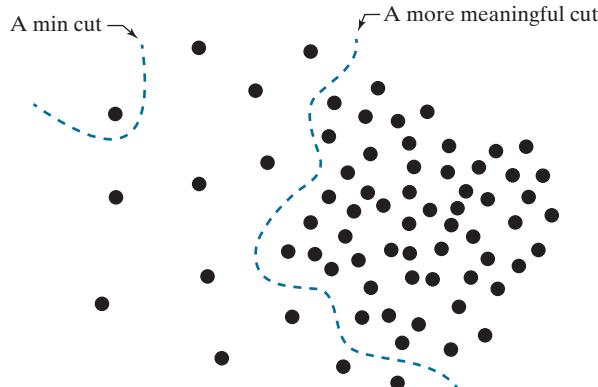


where A and B satisfy Eq. (10-96). The optimum partition of a graph is the one that minimizes this cut value. There is an exponential number of such partitions, which would present us with an intractable computational problem. However, efficient algorithms that run in polynomial time have been developed for solving max-flow problems. Therefore, based on the Max-Flow, Min-Cut Theorem, we can apply these algorithms to image segmentation, provided that we cast segmentation as a flow problem and select the weights for the edges and t-links such that minimum graph cuts will result in meaningful segmentations.

Although the min-cut approach offers an elegant solution, it can result in groupings that favor cutting small sets of isolated nodes in a graph, leading to improper segmentations. Figure 10.55 shows an example, in which the two regions of interest are characterized by the tightness of the pixel groupings. Meaningful edge weights that reflect this property would be inversely proportional to the distance between pairs of points. But this would lead to weights that would be smaller for isolated points, resulting in min cuts such as the example in Fig. 10.55. In fact, any cut that partitions out individual points on the left of the figure will have a smaller cut value in Eq. (10-4) than a cut that properly partitions the points into two groups based on

FIGURE 10.55

An example showing how a min cut can lead to a meaningless segmentation. In this example, the similarity between pixels is defined as their spatial proximity, which results in two distinct regions.



their proximity, such as the partition shown in Fig. 10.55. The approach presented in this section, proposed by Shi and Malik [2000] (see also Hochbaum [2010]), is aimed at avoiding this type of behavior by redefining the concept of a cut.

Instead of looking at the total weight value of the edges that connect two partitions, the idea is to work with a measure of “disassociation” that computes the cost as a fraction of the total edge connections to all nodes in the graph. This measure, called the *normalized cut* ($Ncut$), is defined as

$$Ncut(A, B) = \frac{cut(A, B)}{assoc(A, V)} + \frac{cut(A, B)}{assoc(B, V)} \quad (10-98)$$

where $cut(A, B)$ is given by Eq. (10-97) and

$$assoc(A, V) = \sum_{u \in A, z \in V} w(u, z) \quad (10-99)$$

is the sum of the weights of all the edges from the nodes of subgraph A to the nodes of the entire graph. Similarly,

$$assoc(B, V) = \sum_{v \in B, z \in V} w(v, z) \quad (10-100)$$

is the sum of the weights of the edges from all the edges in B to the entire graph. As you can see, $assoc(A, V)$ is simply the cut of A from the rest of the graph, and similarly for $assoc(B, V)$.

By using $Ncut(A, B)$ instead of $cut(A, B)$, the cut that partitions isolated points will no longer have small values. You can see this, for example, by noting in Fig. 10.55 that if A is the single node shown, $cut(A, B)$ and $assoc(A, V)$ will have the same value. Thus, independently of how small $cut(A, B)$ is, $Ncut(A, B)$ will always be greater than or equal to 1, thus providing normalization for “pathological” cases such as this.

Based on similar concepts, we can define a measure for total *normalized association* within graph partitions as

$$Nassoc(A, B) = \frac{assoc(A, A)}{assoc(A, V)} + \frac{assoc(B, B)}{assoc(B, V)} \quad (10-101)$$

where $assoc(A, A)$ and $assoc(B, B)$ are the total weights connecting the nodes within A and within B , respectively. It is not difficult to show (see Problem 10.46) that

$$Ncut(A, B) = 2 - Nassoc(A, B) \quad (10-102)$$

which implies that minimizing $Ncut(A, B)$ simultaneously maximizes $Nassoc(A, B)$.

Based on the preceding discussion, image segmentation using graph cuts is now based on finding a partition that minimizes $Ncut(A, B)$. Unfortunately, minimizing this quantity exactly is an NP-complete computational task, and we can no longer rely on the solutions available for max flow because the approach being followed now is based on the concepts explained in connection with Fig. 10.53. However, Shi and Malik [2000] (see also Hochbaum [2010]) were able to find an approximate discrete solution to minimizing $Ncut(A, B)$ by formulating minimization as a generalized eigenvalue problem, for which numerous implementations exist.

COMPUTING MINIMAL GRAPH CUTS

As above, let V denote the nodes of a graph G , and let A and B be two subsets of V satisfying Eq. (10-96). Let K denote the number of nodes in V and define a K -dimensional *indicator* vector, \mathbf{x} , whose element x_i has the property $x_i = 1$ if node n_i of V is in A and $x_i = -1$ if it is in B . Let

$$d_i = \sum_j w(i, j) \quad (10-103)$$

be the sum of the weights from node n_i to all other nodes in V . Using these definitions, we can write Eq. (10-98) as

$$\begin{aligned} Ncut(A, B) &= \frac{cut(A, B)}{cut(A, V)} + \frac{cut(A, B)}{cut(B, V)} \\ &= \frac{\sum_{x_i > 0, x_j < 0} -w(i, j)x_i x_j}{\sum_{x_i > 0} d_i} + \frac{\sum_{x_i < 0, x_j > 0} -w(i, j)x_i x_j}{\sum_{x_i < 0} d_i} \end{aligned} \quad (10-104)$$

The objective is to find a vector, \mathbf{x} , that minimizes $Ncut(A, B)$. A closed-form solution that minimizes Eq. (10-104) can be found, but only if the elements of \mathbf{x} are allowed to be real, continuous numbers instead of being constrained to be ± 1 . The solution derived by Shi and Malik [2000] is given by solving the generalized eigen-system expression

$$(\mathbf{D} - \mathbf{W})\mathbf{y} = \lambda \mathbf{D}\mathbf{y} \quad (10-105)$$

where \mathbf{D} is a $K \times K$ diagonal matrix with main-diagonal elements d_i , $i = 1, 2, \dots, K$, and \mathbf{W} is a $K \times K$ weight matrix with elements $w(i, j)$, as defined earlier. Solving

If the nodes of graph G are the pixels in an image, then $K = M \times N$, where M and N are the number of rows and columns in the image.

Eq. (10-105) gives K eigenvalues and K eigenvectors, each corresponding to one eigenvalue. The solution to our problem is the eigenvector corresponding the *second* smallest eigenvalue.

We can convert the preceding generalized eigenvalue formulation into a standard eigenvalue problem by writing Eq. (10-105) as (see Problem 10.45):

$$\mathbf{A}\mathbf{z} = \lambda\mathbf{z} \quad (10-106)$$

where

$$\mathbf{A} = \mathbf{D}^{-\frac{1}{2}}(\mathbf{D} - \mathbf{W})\mathbf{D}^{-\frac{1}{2}} \quad (10-107)$$

and

$$\mathbf{z} = \mathbf{D}^{\frac{1}{2}}\mathbf{y} \quad (10-108)$$

from which it follows that

$$\mathbf{y} = \mathbf{D}^{-\frac{1}{2}}\mathbf{z} \quad (10-109)$$

Thus, we can find the (continuous-valued) eigenvector corresponding to the second smallest eigenvalue using either a generalized or a standard eigenvalue solver. The desired (discrete) vector \mathbf{x} can be generated from the resulting, continuous valued solution vector by finding a splitting point that divides the values of the continuous eigenvector elements into two parts. We do this by finding the splitting point that yields the smallest value of $Ncut(A, B)$, since this is the quantity we are trying to minimize. To simplify the search, we divide the range of values in the continuous vector into Q evenly spaced values, evaluate Eq. (10-104) for each value, and choose the splitting point that yields the smallest value of $Ncut(A, B)$. Then, all values of the eigenvector with values above the split point are assigned the value 1; all others are assigned the value -1. The result is the desired vector \mathbf{x} . Then, partition A is the set nodes in V corresponding to 1's in \mathbf{x} ; the remaining nodes correspond to partition B . This partitioning is carried out only if the stability criterion discussed in the following paragraph is met.

Searching for a splitting point implies computing a total of Q values of $Ncut(A, B)$ and selecting the smallest one. A region that is not clearly segmentable into two subregions using the specified weights will usually result in many splitting points with similar values of $Ncut(A, B)$. Trying to segment such a region is likely to result in a meaningless partition. To avoid this behavior, a region (i.e., subgraph) is split only if it satisfies a *stability criterion*, obtained by first computing the histogram of the eigenvector values, then forming the ratio of the minimum to the maximum bin counts. In an “uncertain” eigenvector, the values in the histogram will stay relatively the same, and the ratio will be relatively high. Shi and Malik [2000] found experimentally that thresholding the ratio at 0.06 was an effective criterion for not splitting the region in question.

GRAPH CUT SEGMENTATION ALGORITHM

In the preceding discussion, we illustrated two ways in which edge weights can be generated from an image. In Figs. 10.53 and 10.54, we looked at weights generated using image intensity values, and in Fig. 10.55 we considered weights based on the distance between pixels. But these are just two examples of the many ways that we can generate a graph and corresponding weights from an image. For example, we could use color, texture, statistical moments about a region, and other types of features to be discussed in Chapter 11. In general, then, graphs can be constructed from image *features*, of which pixel intensities are a special case. With this concept as background, we can summarize the discussion thus far in this section as the following algorithm:

1. Given a set of features, specify a weighted graph, $G = (V, E)$ in which V contains the points in the feature space, and E contains the edges of the graph. Compute the edge weights and use them to construct matrices \mathbf{W} and \mathbf{D} . Let K denote the desired number of partitions of the graph.
2. Solve the eigenvalue system $(\mathbf{D} - \mathbf{W})\mathbf{y} = \lambda\mathbf{D}\mathbf{y}$ to find the eigenvector with the second smallest eigenvalue.
3. Use the eigenvector from Step 2 to bipartition the graph by finding the splitting point such that $Ncut(A, B)$ is minimized.
4. If the number of cuts has not reached K , decide if the current partition should be subdivided by checking the stability of the cut.
5. Recursively repartition the segmented parts if necessary.

Note that the algorithm works by recursively generating two-way cuts. The number of groups (e.g., regions) in the segmented image is controlled by K . Other criteria, such as the maximum size allowed for each cut, can further refine the final segmentation. For example, when using pixels and their intensities as the basis for constructing the graph, we can specify the maximum and/or minimum size allowed for each region.

EXAMPLE 10.24: Specifying weights for graph cut segmentation.

In Fig. 10.53, we illustrated how to generate graph weights using intensity values, and in Fig. 10.55 we discussed briefly how to generate weights based on the distance between pixels. In this example, we give a more practical approach for generating weights that include both intensity and distance from a pixel, thus introducing the concept of a neighborhood in graph segmentation.

Let n_i and n_j denote two nodes (image pixels). As mentioned earlier in this section, weights are supposed to reflect the similarity between nodes in a graph. When considering segmentation, one of the principal ways to establish how likely two pixels in an image are to be a part of the same region or object is to determine the difference in their intensity values, and how close the pixels are to each other. The weight value of the edge between two pixels should be large when the pixels are very close in intensity and proximity (i.e., when the pixels are “similar”), and should decrease as their intensity difference and distance from each other increases. That is, the weight value should be a function of how similar the pixels are in intensity and distance. These two concepts can be embedded into a single weight function using the following expression:

$$w(i,j) = \begin{cases} e^{-\frac{[I(n_i)-I(n_j)]^2}{\sigma_I^2}} e^{-\frac{dist(n_i,n_j)}{\sigma_d^2}} & \text{if } dist(n_i,n_j) < r \\ 0 & \text{otherwise} \end{cases}$$

where $I(n_i)$ is the intensity of node n_i , σ_I^2 and σ_d^2 are constants determining the spread of the two Gaussian-like functions, $dist(n_i, n_j)$ is the distance (e.g., the Euclidean distance) between the two nodes, and r is a radial constant that establishes how far away we are willing to consider similarity. The exponential terms decrease as a function of dissimilarity in intensity and as function of distance between the nodes, as required of our measure of similarity in this case.

EXAMPLE 10.25: Segmentation using graph cuts.

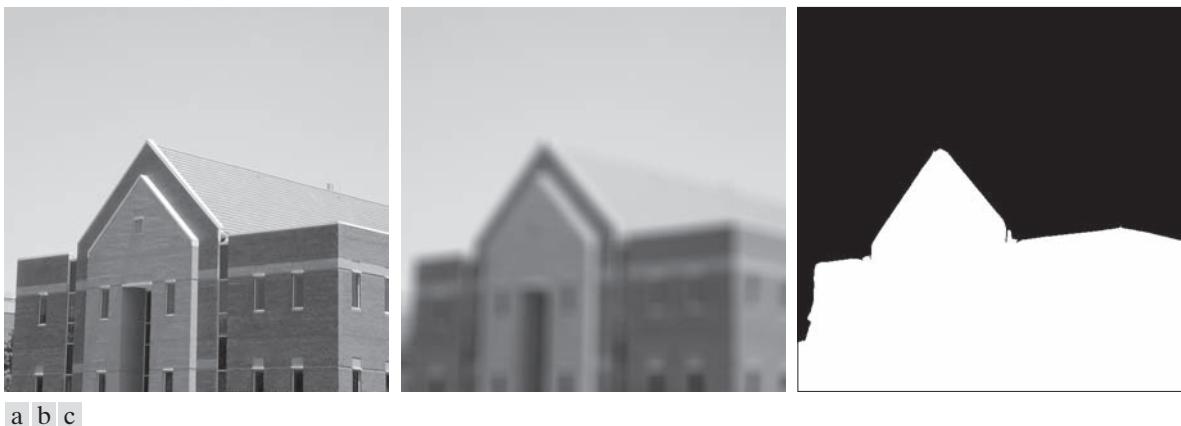
Graph cuts are ideally suited for obtaining a rough segmentation of the principal regions in an image. Figure 10.56 shows a typical result. Figure 10.56(a) is the familiar building image. Consistent with the idea of extracting the principal regions of an image, Fig. 10.56(b) shows the image smoothed with a simple 25×25 box kernel. Observe how the fine detail is smoothed out, leaving only major regional features such as the facade and sky. Figure 10.56(c) is the result of segmentation using the graph cut algorithm just developed, with weights of the form discussed in the previous example, and allowing only two partitions. Note how well the region corresponding to the building was extracted, with none of the details characteristic of the methods discussed earlier in this chapter. In fact, it would have been nearly impossible to obtain comparable results using any of the methods we have discussed thus far without significant additional processing. This type of result is ideal for tasks such as providing broad cues for autonomous navigation, for searching image databases, and for low-level image analysis.

10.7 SEGMENTATION USING MORPHOLOGICAL WATERSHEDS

Thus far, we have discussed segmentation based on three principal concepts: edge detection, thresholding, and region extraction. Each of these approaches was found to have advantages (for example, speed in the case of global thresholding) and disadvantages (for example, the need for post-processing, such as edge linking, in edge-based segmentation). In this section, we discuss an approach based on the concept of so-called *morphological watersheds*. Segmentation by watersheds embodies many of the concepts of the other three approaches and, as such, often produces more stable segmentation results, including connected segmentation boundaries. This approach also provides a simple framework for incorporating knowledge-based constraints (see Fig. 1.23) in the segmentation process, as we discuss at the end of this section.

BACKGROUND

The concept of a watershed is based on visualizing an image in three dimensions, two spatial coordinates versus intensity, as in Fig. 2.18(a). In such a “topographic” interpretation, we consider three types of points: (1) points belonging to a regional minimum; (2) points at which a drop of water, if placed at the location of any of those



a b c

FIGURE 10.56 (a) Image of size 600×600 pixels. (b) Image smoothed with a 25×25 box kernel. (c) Graph cut segmentation obtained by specifying two regions.

points, would fall with certainty to a single minimum; and (3) points at which water would be equally likely to fall to more than one such minimum. For a particular regional minimum, the set of points satisfying condition (2) is called the *catchment basin* or *watershed* of that minimum. The points satisfying condition (3) form crest lines on the topographic surface, and are referred to as *divide lines* or *watershed lines*.

The principal objective of segmentation algorithms based on these concepts is to find the watershed lines. The method for doing this can be explained with the aid of Fig. 10.57. Figure 10.57(a) shows a gray-scale image and Fig. 10.57(b) is a topographic view, in which the height of the “mountains” is proportional to intensity values in the input image. For ease of interpretation, the backsides of structures are shaded. This is not to be confused with intensity values; only the general topography of the three-dimensional representation is of interest. In order to prevent the rising water from spilling out through the edges of the image, we imagine the perimeter of the entire topography (image) being enclosed by dams that are higher than the highest possible mountain, whose value is determined by the highest possible intensity value in the input image.

Suppose that a hole is punched in each regional minimum [shown as dark areas in Fig. 10.57(b)] and that the entire topography is flooded from below by letting water rise through the holes at a uniform rate. Figure 10.57(c) shows the first stage of flooding, where the “water,” shown in light gray, has covered only areas that correspond to the black *background* in the image. In Figs. 10.57(d) and (e) we see that the water now has risen into the first and second catchment basins, respectively. As the water continues to rise, it will eventually overflow from one catchment basin into another. The first indication of this is shown in 10.57(f). Here, water from the lower part of the left basin overflowed into the basin on the right, and a short “dam” (consisting of single pixels) was built to prevent water from merging at that level of flooding (the mathematical details of dam building are discussed in the following section). The

Because of neighboring contrast, the leftmost basin in Fig. 10.57(c) appears black, but it is a few shades lighter than the black background. The mid-gray in the second basin is a natural gray from the image in (a).

a	c
b	d

FIGURE 10.57

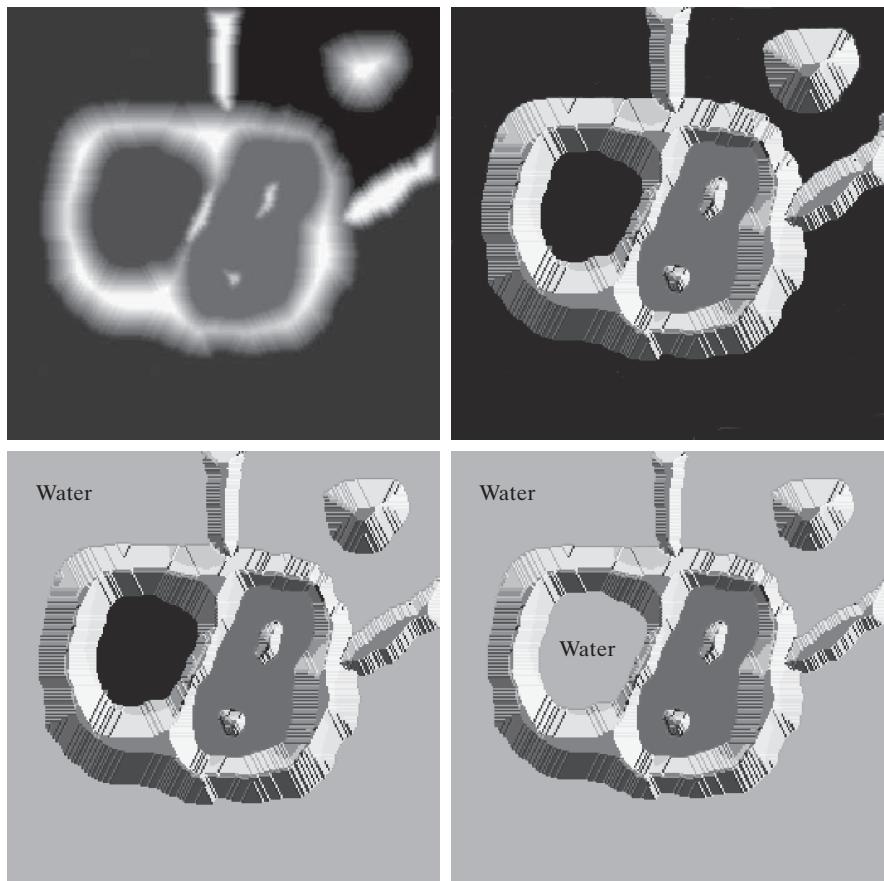
(a) Original image.

(b) Topographic view. Only the background is black. The basin on the left is slightly lighter than black.

(c) and (d) Two stages of flooding. All constant dark values of gray are intensities in the original image. Only constant light gray represents “water.”

(Courtesy of Dr. S. Beucher, CMM/Ecole des Mines de Paris.)

(Continued on next page.)



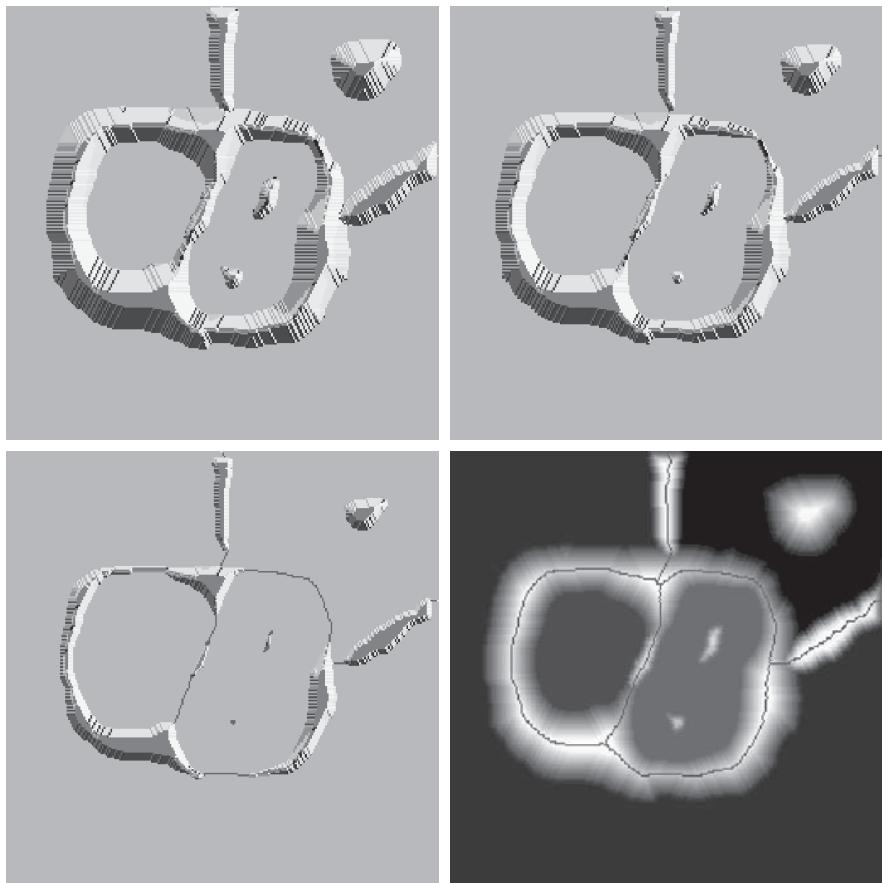
effect is more pronounced as water continues to rise, as shown in Fig. 10.57(g). This figure shows a longer dam between the two catchment basins and another dam in the top part of the right basin. The latter dam was built to prevent merging of water from that basin with water from areas corresponding to the background. This process is continued until the maximum level of flooding (corresponding to the highest intensity value in the image) is reached. The final dams correspond to the *watershed lines*, which are the desired segmentation boundaries. The result for this example is shown in Fig. 10.57(h) as dark, one-pixel-thick paths superimposed on the original image. Note the important property that the watershed lines form connected paths, thus giving continuous boundaries between regions.

One of the principal applications of watershed segmentation is in the extraction of nearly uniform (blob-like) objects from the background. Regions characterized by small variations in intensity have small gradient values. Thus, in practice, we often see watershed segmentation applied to the gradient of an image, rather than to the image itself. In this formulation, the regional minima of catchment basins correlate nicely with the small value of the gradient corresponding to the objects of interest.

e f
g h

FIGURE 10.57

- (Continued)
- (e) Result of further flooding.
 - (f) Beginning of merging of water from two catchment basins (a short dam was built between them).
 - (g) Longer dams.
 - (h) Final watershed (segmentation) lines superimposed on the original image.
- (Courtesy of Dr. S. Beucher, CMM/Ecole des Mines de Paris.)



DAM CONSTRUCTION

Dam construction is based on binary images, which are members of 2-D integer space Z^2 (see Sections 2.4 and 2.6). The simplest way to construct dams separating sets of binary points is to use morphological dilation (see Section 9.2).

Figure 10.58 illustrates the basics of dam construction using dilation. Part (a) shows portions of two catchment basins at flooding step $n - 1$, and Fig. 10.58(b) shows the result at the next flooding step, n . The water has spilled from one basin to the another and, therefore, a dam must be built to keep this from happening. In order to be consistent with notation to be introduced shortly, let M_1 and M_2 denote the sets of coordinates of points in two regional minima. Then let the set of coordinates of points in the catchment basin associated with these two minima at stage $n - 1$ of flooding be denoted by $C_{n-1}(M_1)$ and $C_{n-1}(M_2)$, respectively. These are the two gray regions in Fig. 10.58(a).

Let $C[n - 1]$ denote the union of these two sets. There are two connected components in Fig. 10.58(a), and only one component in Fig. 10.58(b). This connected

See Sections 2.5 and 9.5 regarding connected components.

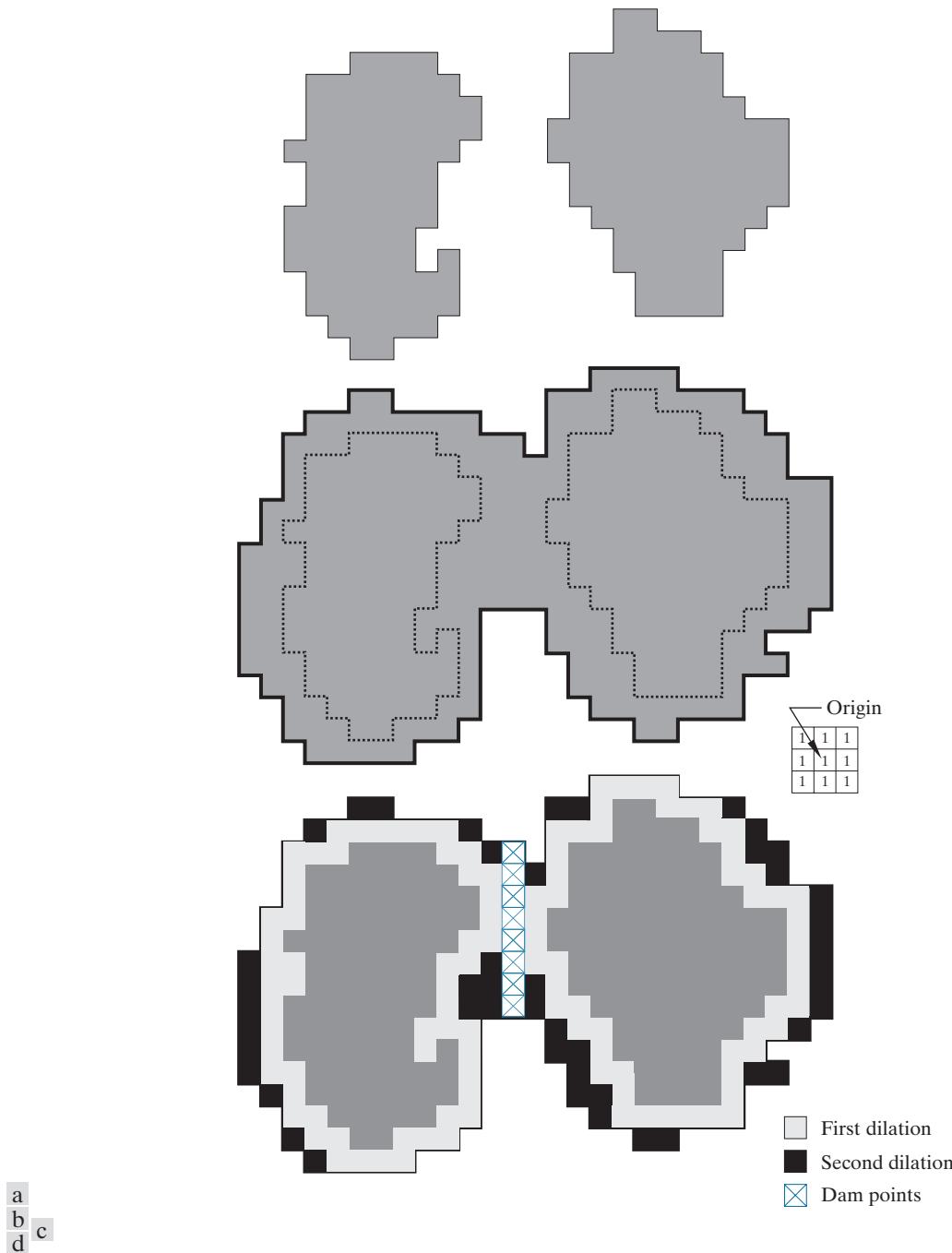


FIGURE 10.58 (a) Two partially flooded catchment basins at stage $n - 1$ of flooding. (b) Flooding at stage n , showing that water has spilled between basins. (c) Structuring element used for dilation. (d) Result of dilation and dam construction.

component encompasses the earlier two components, which are shown dashed. Two connected components having become a *single* component indicates that water between the two catchment basins has merged at flooding step n . Let this connected component be denoted by q . Note that the two components from step $n - 1$ can be extracted from q by performing a logical AND operation, $q \cap C[n - 1]$. Observe also that all points belonging to an individual catchment basin form a single connected component.

Suppose that each of the connected components in Fig. 10.58(a) is dilated by the structuring element in Fig. 10.58(c), subject to two conditions: (1) The dilation has to be constrained to q (this means that the center of the structuring element can be located only at points in q during dilation); and (2) the dilation cannot be performed on points that would cause the sets being dilated to merge (i.e., become a single connected component). Figure 10.58(d) shows that a first dilation pass (in light gray) expanded the boundary of each original connected component. Note that condition (1) was satisfied by every point during dilation, and that condition (2) did not apply to any point during the dilation process; thus, the boundary of each region was expanded uniformly.

In the second dilation, shown in black in 10.58(d), several points failed condition (1) while meeting condition (2), resulting in the broken perimeter shown in the figure. It is evident that the only points in q that satisfy the two conditions under consideration describe the one-pixel-thick connected path shown crossed-hatched in Fig. 10.58(d). This path is the desired separating dam at stage n of flooding. Construction of the dam at this level of flooding is completed by setting all the points in the path just determined to a value greater than the maximum possible intensity value of the image (e.g., greater than 255 for an 8-bit image). This will prevent water from crossing over the part of the completed dam as the level of flooding is increased. As noted earlier, dams built by this procedure, which are the desired segmentation boundaries, are connected components. In other words, this method eliminates the problems of broken segmentation lines.

Although the procedure just described is based on a simple example, the method used for more complex situations is exactly the same, including the use of the 3×3 symmetric structuring element in Fig. 10.58(c).

WATERSHED SEGMENTATION ALGORITHM

Let M_1, M_2, \dots, M_R be sets denoting the *coordinates* of the points in the regional minima of an image, $g(x, y)$. As mentioned earlier, this typically will be a gradient image. Let $C(M_i)$ be a set denoting the coordinates of the points in the catchment basin associated with regional minimum M_i (recall that the points in any catchment basin form a connected component). The notation *min* and *max* will be used to denote the minimum and maximum values of $g(x, y)$. Finally, let $T[n]$ represent the set of coordinates (s, t) for which $g(s, t) < n$. That is,

$$T[n] = \{(s, t) | g(s, t) < n\} \quad (10-110)$$

Geometrically, $T[n]$ is the set of coordinates of points in $g(x,y)$ lying below the plane $g(x,y) = n$.

The topography will be flooded in *integer* flood increments, from $n = \min + 1$ to $n = \max + 1$. At any step n of the flooding process, the algorithm needs to know the number of points below the flood depth. Conceptually, suppose that the coordinates in $T[n]$ that are below the plane $g(x,y) = n$ are “marked” black, and all other coordinates are marked white. Then when we look “down” on the xy -plane at any increment n of flooding, we will see a binary image in which black points correspond to points in the function that are below the plane $g(x,y) = n$. This interpretation is quite useful, and will make it easier to understand the following discussion.

Let $C_n(M_i)$ denote the set of coordinates of points in the catchment basin associated with minimum M_i that are flooded at stage n . With reference to the discussion in the previous paragraph, we may view $C_n(M_i)$ as a binary image given by

$$C_n(M_i) = C(M_i) \cap T[n] \quad (10-111)$$

In other words, $C_n(M_i) = 1$ at location (x,y) if $(x,y) \in C(M_i)$ AND $(x,y) \in T[n]$; otherwise $C_n(M_i) = 0$. The geometrical interpretation of this result is straightforward. We are simply using the AND operator to isolate at stage n of flooding the portion of the binary image in $T[n]$ that is associated with regional minimum M_i .

Next, let B denote the number of flooded catchment basins at stage n , and let $C[n]$ denote the union of these basins at stage n :

$$C[n] = \bigcup_{i=1}^B C_n(M_i) \quad (10-112)$$

Then $C[\max + 1]$ is the union of all catchment basins:

$$C[\max + 1] = \bigcup_{i=1}^B C(M_i) \quad (10-113)$$

It can be shown (see Problem 10.47) that the elements in both $C_n(M_i)$ and $T[n]$ are never replaced during execution of the algorithm, and that the number of elements in these two sets either increases or remains the same as n increases. Thus, it follows that $C[n - 1]$ is a subset of $C[n]$. According to Eqs. (10-112) and (10-113), $C[n]$ is a subset of $T[n]$, so it follows that $C[n - 1]$ is also a subset of $T[n]$. From this we have the important result that each connected component of $C[n - 1]$ is contained in exactly one connected component of $T[n]$.

The algorithm for finding the watershed lines is initialized by letting $C[\min + 1] = T[\min + 1]$. The procedure then proceeds recursively, successively computing $C[n]$ from $C[n - 1]$, using the following approach. Let Q denote the set of connected components in $T[n]$. Then, for each connected component $q \in Q[n]$, there are three possibilities:

- 1.** $q \cap C[n - 1]$ is empty.

2. $q \cap C[n - 1]$ contains one connected component of $C[n - 1]$.
3. $q \cap C[n - 1]$ contains more than one connected component of $C[n - 1]$.

The construction of $C[n]$ from $C[n - 1]$ depends on which of these three conditions holds. Condition 1 occurs when a new minimum is encountered, in which case connected component q is incorporated into $C[n - 1]$ to form $C[n]$. Condition 2 occurs when q lies within the catchment basin of some regional minimum, in which case q is incorporated into $C[n - 1]$ to form $C[n]$. Condition 3 occurs when all (or part) of a ridge separating two or more catchment basins is encountered. Further flooding would cause the water level in these catchment basins to merge. Thus, a dam (or dams if more than two catchment basins are involved) must be built within q to prevent overflow between the catchment basins. As explained earlier, a one-pixel-thick dam can be constructed when needed by dilating $q \cap C[n - 1]$ with a 3×3 structuring element of 1's, and constraining the dilation to q .

Algorithm efficiency is improved by using only values of n that correspond to existing intensity values in $g(x, y)$. We can determine these values, as well as the values of min and max, from the histogram of $g(x, y)$.

EXAMPLE 10.26: Illustration of the watershed segmentation algorithm.

Consider the image and its gradient in Figs. 10.59(a) and (b), respectively. Application of the watershed algorithm just described yielded the watershed lines (white paths) shown superimposed on the gradient image in Fig. 10.59(c). These segmentation boundaries are shown superimposed on the original image in Fig. 10.59(d). As noted at the beginning of this section, the segmentation boundaries have the important property of being connected paths.

THE USE OF MARKERS

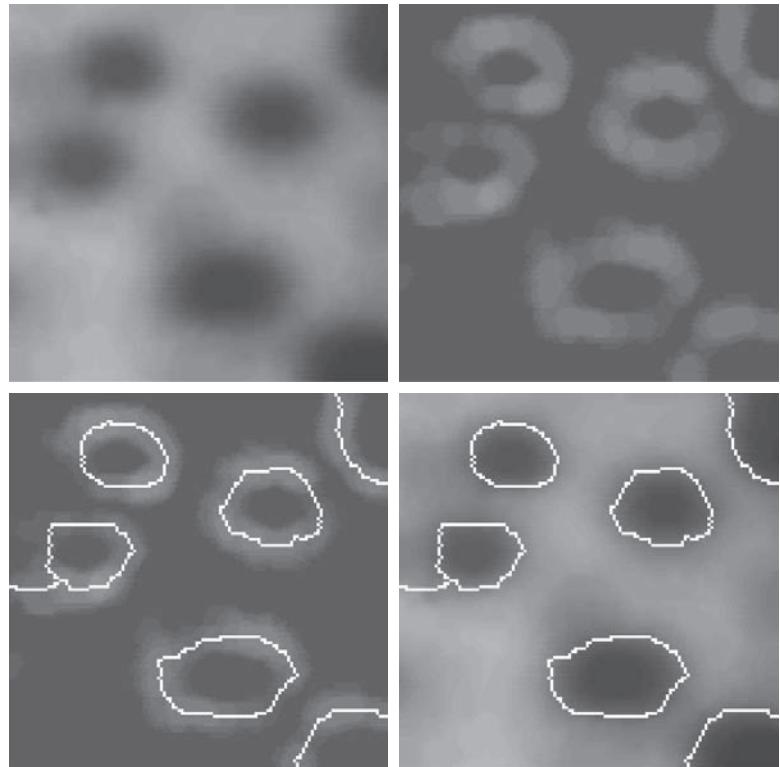
Direct application of the watershed segmentation algorithm in the form discussed in the previous section generally leads to over-segmentation, caused by noise and other local irregularities of the gradient. As Fig. 10.60 illustrates, over-segmentation can be serious enough to render the result of the algorithm virtually useless. In this case, this means a large number of segmented regions. A practical solution to this problem is to limit the number of allowable regions by incorporating a preprocessing stage designed to bring additional knowledge into the segmentation procedure.

An approach used to control over-segmentation is based on the concept of markers. A *marker* is a connected component belonging to an image. We have *internal markers*, associated with objects of interest, and *external markers*, associated with the background. A procedure for marker selection typically will consist of two principal steps: (1) preprocessing; and (2) definition of a set of criteria that markers must satisfy. To illustrate, consider Fig. 10.60(a) again. Part of the problem that led to the over-segmented result in Fig. 10.60(b) is the large number of potential minima. Because of their size, many of these minima are irrelevant detail. As has been pointed out several times in earlier discussions, an effective method for minimizing the effect of small spatial detail is to filter the image with a smoothing filter. This is an appropriate preprocessing scheme in this case also.

a b
c d

FIGURE 10.59

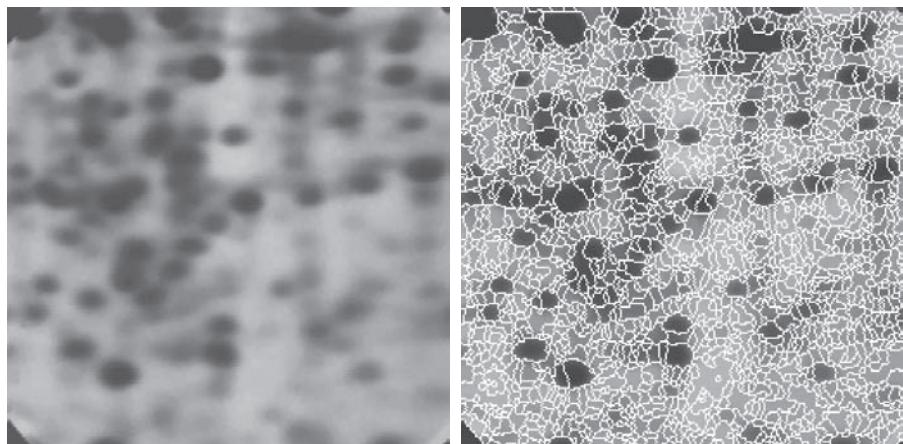
- (a) Image of blobs.
(b) Image gradient.
(c) Watershed lines,
superimposed on
the gradient image.
(d) Watershed lines
superimposed on
the original image.
(Courtesy of Dr.
S. Beucher, CMM/
Ecole des Mines de
Paris.)



a b

FIGURE 10.60

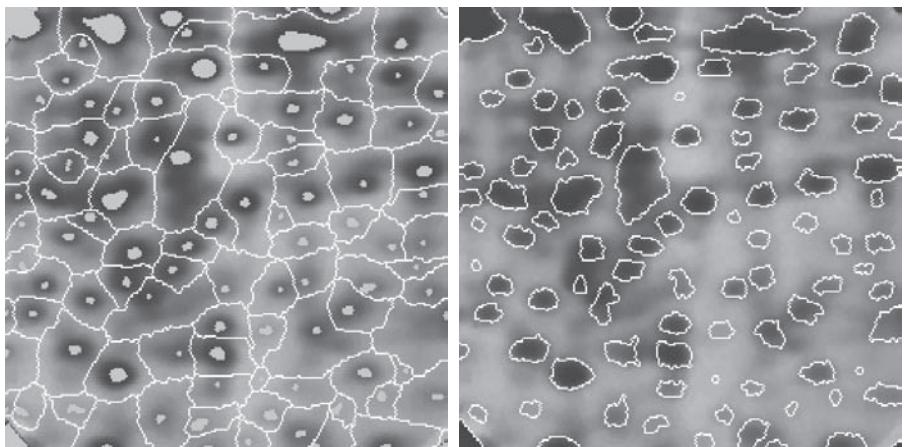
- (a) Electrophoresis
image.
(b) Result of applying
the watershed
algorithm to the
gradient
image.
Over-segmentation
is evident.
(Courtesy of Dr.
S. Beucher, CMM/
Ecole des Mines de
Paris.)



a b

FIGURE 10.61

- (a) Image showing internal markers (light gray regions) and external markers (watershed lines).
 (b) Result of segmentation. Note the improvement over Fig. 10.60(b). (Courtesy of Dr. S. Beucher, CMM/Ecole des Mines de Paris.)



Suppose that we define an internal marker as (1) a region that is surrounded by points of higher “altitude”; (2) such that the points in the region form a connected component; and (3) in which all the points in the connected component have the same intensity value. After the image was smoothed, the internal markers resulting from this definition are shown as light gray, blob-like regions in Fig. 10.61(a). Next, the watershed algorithm was applied to the smoothed image, under the restriction that these internal markers be the only allowed regional minima. Figure 10.61(a) shows the resulting watershed lines. These watershed lines are defined as the external markers. Note that the points along the watershed line pass along the highest points between neighboring markers.

The external markers in Fig. 10.61(a) effectively partition the image into regions, with each region containing a single internal marker and part of the background. The problem is thus reduced to partitioning each of these regions into two: a single object, and its background. We can bring to bear on this simplified problem many of the segmentation techniques discussed earlier in this chapter. Another approach is simply to apply the watershed segmentation algorithm to each individual region. In other words, we simply take the gradient of the smoothed image [as in Fig. 10.59(b)] and restrict the algorithm to operate on a single watershed that contains the marker in that particular region. Figure 10.61(b) shows the result obtained using this approach. The improvement over the image in 10.60(b) is evident.

Marker selection can range from simple procedures based on intensity values and connectivity, as we just illustrated, to more complex descriptions involving size, shape, location, relative distances, texture content, and so on (see Chapter 11 regarding feature descriptors). The point is that using markers brings a priori knowledge to bear on the segmentation problem. Keep in mind that humans often aid segmentation and higher-level tasks in everyday vision by using a priori knowledge, one of the most familiar being the use of context. Thus, the fact that segmentation by watersheds offers a framework that can make effective use of this type of knowledge is a significant advantage of this method.

10.8 THE USE OF MOTION IN SEGMENTATION

Motion is a powerful cue used by humans and many animals to extract objects or regions of interest from a background of irrelevant detail. In imaging applications, motion arises from a relative displacement between the sensing system and the scene being viewed, such as in robotic applications, autonomous navigation, and dynamic scene analysis. In the following discussion we consider the use of motion in segmentation both spatially and in the frequency domain.

Spatial Techniques

In what follows, we will consider two approaches for detecting motion, working directly in the spatial domain. The key objective is to give you an idea how to measure changes in digital images using some straightforward techniques.

A Basic Approach

One of the simplest approaches for detecting changes between two image frames $f(x, y, t_i)$ and $f(x, y, t_j)$ taken at times t_i and t_j , respectively, is to compare the two images pixel by pixel. One procedure for doing this is to form a difference image. Suppose that we have a *reference image* containing only stationary components. Comparing this image against a subsequent image of the same scene, but including one or more moving objects, results in the difference of the two images canceling the stationary elements, leaving only nonzero entries that correspond to the nonstationary image components.

A *difference image* of two images (of the same size) taken at times t_i and t_j may be defined as

$$d_{ij}(x, y) = \begin{cases} 1 & \text{if } |f(x, y, t_i) - f(x, y, t_j)| > T \\ 0 & \text{otherwise} \end{cases} \quad (10-114)$$

where T is a nonnegative threshold. Note that $d_{ij}(x, y)$ has a value of 1 at spatial coordinates (x, y) only if the intensity difference between the two images is appreciably different at those coordinates, as determined by T . Note also that coordinates (x, y) in Eq. (10-114) span the dimensions of the two images, so the difference image is of the same size as the images in the sequence.

In the discussion that follows, all pixels in $d_{ij}(x, y)$ that have value 1 are considered the result of object motion. This approach is applicable only if the two images are registered spatially, and if the illumination is relatively constant within the bounds established by T . In practice, 1-valued entries in $d_{ij}(x, y)$ may arise as a result of noise also. Typically, these entries are isolated points in the difference image, and a simple approach to their removal is to form 4- or 8-connected regions of 1's in image $d_{ij}(x, y)$, then ignore any region that has less than a predetermined number of elements. Although it may result in ignoring small and/or slow-moving objects, this approach improves the chances that the remaining entries in the difference image actually are the result of motion, and not noise.

Although the method just described is simple, it is used frequently as the basis of imaging systems designed to detect changes in controlled environments, such as in surveillance of parking facilities, buildings, and similar fixed locales.

Accumulative Differences

Consider a sequence of image frames denoted by $f(x, y, t_1), f(x, y, t_2), \dots, f(x, y, t_n)$, and let $f(x, y, t_1)$ be the reference image. An *accumulative difference image* (ADI) is formed by comparing this reference image with every subsequent image in the sequence. A counter for each pixel location in the accumulative image is incremented every time a difference occurs at that pixel location between the reference and an image in the sequence. Thus, when the k th frame is being compared with the reference, the entry in a given pixel of the accumulative image gives the number of times the intensity at that position was different [as determined by T in Eq. (10-114)] from the corresponding pixel value in the reference image.

Assuming that the intensity values of the moving objects are greater than the background, we consider three types of ADIs. Let $R(x, y)$ denote the reference image and, to simplify the notation, let k denote t_k so that $f(x, y, k) = f(x, y, t_k)$. We assume that $R(x, y) = f(x, y, 1)$. Then, for any $k > 1$, and keeping in mind that the values of the ADIs are counts, we define the following accumulative differences for all relevant values of (x, y) :

$$A_k(x, y) = \begin{cases} A_{k-1}(x, y) + 1 & \text{if } |R(x, y) - f(x, y, k)| > T \\ A_{k-1}(x, y) & \text{otherwise} \end{cases} \quad (10-115)$$

$$P_k(x, y) = \begin{cases} P_{k-1}(x, y) + 1 & \text{if } |R(x, y) - f(x, y, k)| > T \\ P_{k-1}(x, y) & \text{otherwise} \end{cases} \quad (10-116)$$

and

$$N_k(x, y) = \begin{cases} N_{k-1}(x, y) + 1 & \text{if } |R(x, y) - f(x, y, k)| < -T \\ N_{k-1}(x, y) & \text{otherwise} \end{cases} \quad (10-117)$$

where $A_k(x, y)$, $P_k(x, y)$, and $N_k(x, y)$ are the *absolute*, *positive*, and *negative* ADIs, respectively, computed using the k th image in the sequence. All three ADIs start out with zero counts and are of the same size as the images in the sequence. The order of the inequalities and signs of the thresholds in Eqs. (10-116) and (10-117) are reversed if the intensity values of the background pixels are greater than the values of the moving objects.

EXAMPLE 10.27: Computation of the absolute, positive, and negative accumulative difference images.

Figure 10.62 shows the three ADIs displayed as intensity images for a rectangular object of dimension 75×50 pixels that is moving in a southeasterly direction at a speed of $5\sqrt{2}$ pixels per frame. The images



FIGURE 10.62 ADIs of a rectangular object moving in a southeasterly direction. (a) Absolute ADI. (b) Positive ADI. (c) Negative ADI.

are of size 256×256 pixels. We note the following: (1) The nonzero area of the positive ADI is equal to the size of the moving object; (2) the location of the positive ADI corresponds to the location of the moving object in the reference frame; (3) the number of counts in the positive ADI stops increasing when the moving object is displaced completely with respect to the same object in the reference frame; (4) the absolute ADI contains the regions of the positive and negative ADI; and (5) the direction and speed of the moving object can be determined from the entries in the absolute and negative ADIs.

Establishing a Reference Image

A key to the success of the techniques just discussed is having a reference image against which subsequent comparisons can be made. The difference between two images in a dynamic imaging problem has the tendency to cancel all stationary components, leaving only image elements that correspond to noise and to the moving objects.

Obtaining a reference image with only stationary elements is not always possible, and building a reference from a set of images containing one or more moving objects becomes necessary. This applies particularly to situations describing busy scenes or in cases where frequent updating is required. One procedure for generating a reference image is as follows. Consider the first image in a sequence to be the reference image. When a nonstationary component has moved completely out of its position in the reference frame, the corresponding background in the present frame can be duplicated in the location originally occupied by the object in the reference frame. When all moving objects have moved completely out of their original positions, a reference image containing only stationary components will have been created. Object displacement can be established by monitoring the changes in the positive ADI, as indicated earlier. The following example illustrates how to build a reference frame using the approach just described.

EXAMPLE 10.28: Building a reference image.

Figures 10.63(a) and (b) show two image frames of a traffic intersection. The first image is considered the reference, and the second depicts the same scene some time later. The objective is to remove the principal moving objects in the reference image in order to create a static image. Although there are other smaller moving objects, the principal moving feature is the automobile at the intersection moving from left to right. For illustrative purposes we focus on this object. By monitoring the changes in the positive ADI, it is possible to determine the initial position of a moving object, as explained above. Once the area occupied by this object is identified, the object can be removed from the image by subtraction. By looking at the frame in the sequence at which the positive ADI stopped changing, we can copy from this image the area previously occupied by the moving object in the initial frame. This area then is pasted onto the image from which the object was cut out, thus restoring the background of that area. If this is done for all moving objects, the result is a reference image with only static components against which we can compare subsequent frames for motion detection. The reference image resulting from removing the east-bound moving vehicle and restoring the background is shown in Fig. 10.63(c).

FREQUENCY DOMAIN TECHNIQUES

In this section, we consider the problem of determining motion via a Fourier transform formulation. Consider a sequence $f(x, y, t)$, $t = 0, 1, 2, \dots, K - 1$, of K digital image frames of size $M \times N$ pixels, generated by a stationary camera. We begin the development by assuming that all frames have a homogeneous background of zero intensity. The exception is a single, 1-pixel object of unit intensity that is moving with constant velocity. Suppose that for frame one ($t = 0$), the object is at location (x', y') and the image plane is projected onto the x -axis; that is, the pixel intensities are summed (for each row) across the columns in the image. This operation yields a 1-D array with M entries that are zero, except at x' , which is the x -coordinate of the single-point object. If we now multiply all the components of the 1-D array by the quantity $\exp[j2\pi a_1 x \Delta t]$ for $x = 0, 1, 2, \dots, M - 1$ and add the results, we obtain the single term $\exp[j2\pi a_1 x' \Delta t]$ because there is only one nonzero point in the array. In this notation, a_1 is a positive integer, and Δt is the time interval between frames.



a b c

FIGURE 10.63 Building a static reference image. (a) and (b) Two frames in a sequence. (c) Eastbound automobile subtracted from (a), and the background restored from the corresponding area in (b). (Jain and Jain.)

Suppose that in frame two ($t = 1$), the object has moved to coordinates $(x' + 1, y')$; that is, it has moved 1 pixel parallel to the x -axis. Then, repeating the projection procedure discussed in the previous paragraph yields the sum $\exp[j2\pi a_1(x' + 1)\Delta t]$. If the object continues to move 1 pixel location per frame then, at any integer instant of time, t , the result will be $\exp[j2\pi a_1(x' + t)\Delta t]$, which, using Euler's formula, may be expressed as

$$e^{j2\pi a_1(x' + t)\Delta t} = \cos[2\pi a_1(x' + t)\Delta t] + j\sin[2\pi a_1(x' + t)\Delta t] \quad (10-118)$$

for $t = 0, 1, 2, \dots, K - 1$. In other words, this procedure yields a complex sinusoid with frequency a_1 . If the object were moving V_1 pixels (in the x -direction) between frames, the sinusoid would have frequency $V_1 a_1$. Because t varies between 0 and $K - 1$ in integer increments, restricting a_1 to have integer values causes the discrete Fourier transform of the complex sinusoid to have two peaks—one located at frequency $V_1 a_1$ and the other at $K - V_1 a_1$. This latter peak is the result of symmetry in the discrete Fourier transform, as discussed in Section 4.6, and may be ignored. Thus a peak search in the Fourier spectrum would yield one peak with value $V_1 a_1$. Dividing this quantity by a_1 yields V_1 , which is the velocity component in the x -direction, as the frame rate is assumed to be known. A similar analysis would yield V_2 , the component of velocity in the y -direction.

A sequence of frames in which no motion takes place produces identical exponential terms, whose Fourier transform would consist of a single peak at a frequency of 0 (a single dc term). Therefore, because the operations discussed so far are linear, the general case involving one or more moving objects in an arbitrary static background would have a Fourier transform with a peak at dc corresponding to static image components, and peaks at locations proportional to the velocities of the objects.

These concepts may be summarized as follows. For a sequence of k digital images of size $M \times N$ pixels, the sum of the weighted projections onto the x -axis at any integer instant of time is

$$g_x(t, a_1) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y, t) e^{j2\pi a_1 x \Delta t} \quad t = 0, 1, \dots, K - 1 \quad (10-119)$$

Similarly, the sum of the projections onto the y -axis is

$$g_y(t, a_2) = \sum_{y=0}^{N-1} \sum_{x=0}^{M-1} f(x, y, t) e^{j2\pi a_2 y \Delta t} \quad t = 0, 1, \dots, K - 1 \quad (10-120)$$

where, as noted earlier, a_1 and a_2 are positive integers.

The 1D Fourier transforms of Eqs. (10-119) and (10-120), respectively, are

$$G_x(u_1, a_1) = \sum_{t=0}^{K-1} g_x(t, a_1) e^{-j2\pi u_1 t / K} \quad u_1 = 0, 1, \dots, K - 1 \quad (10-121)$$

and

$$G_y(u_2, a_2) = \sum_{t=0}^{K-1} g_y(t, a_2) e^{-j2\pi u_2 t / K} \quad u_2 = 0, 1, \dots, K-1 \quad (10-122)$$

These transforms are computed using an FFT algorithm, as discussed in Section 4.11.

The frequency-velocity relationship is

$$u_1 = a_1 V_1 \quad (10-123)$$

and

$$u_2 = a_2 V_2 \quad (10-124)$$

In the preceding formulation, the unit of velocity is in pixels per total frame time. For example, $V_1 = 10$ indicates motion of 10 pixels in K frames. For frames that are taken uniformly, the actual physical speed depends on the frame rate and the distance between pixels. Thus, if $V_1 = 10$, and $K = 30$, the frame rate is two images per second, and the distance between pixels is 0.5 m, then the actual physical speed in the x -direction is

$$V_1 = (10 \text{ pixels})(0.5 \text{ m/pixel})(2 \text{ frames/s})(30 \text{ frames})$$

The sign of the x -component of the velocity is obtained by computing

$$S_{1x} = \left. \frac{d^2 \operatorname{Re}[g_x(t, a_1)]}{dt^2} \right|_{t=n} \quad (10-125)$$

and

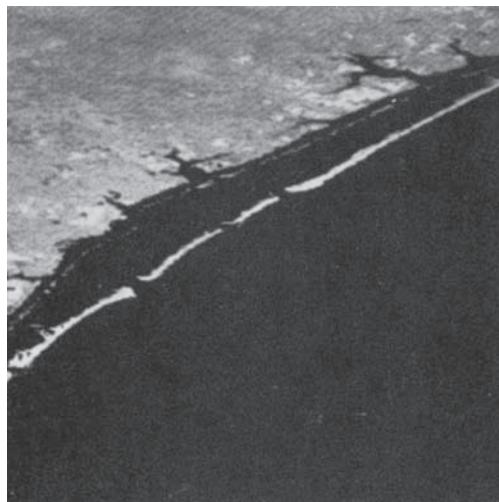
$$S_{2x} = \left. \frac{d^2 \operatorname{Im}[g_x(t, a_1)]}{dt^2} \right|_{t=n} \quad (10-126)$$

Because g_x is sinusoidal, it can be shown (see Problem 10.53) that S_{1x} and S_{2x} will have the same sign at an arbitrary point in time, n , if the velocity component V_1 is positive. Conversely, opposite signs in S_{1x} and S_{2x} indicate a negative velocity component. If either S_{1x} or S_{2x} is zero, we consider the next closest point in time, $t = n \pm \Delta t$. Similar comments apply to computing the sign of V_2 .

EXAMPLE 10.29: Detection of a small moving object via frequency-domain analysis.

Figures 10.64 through 10.66 illustrate the effectiveness of the approach just developed. Figure 10.64 shows one of a 32-frame sequence of LANDSAT images generated by adding white noise to a reference image. The sequence contains a superimposed target moving at 0.5 pixel per frame in the x -direction and 1 pixel per frame in the y -direction. The target, shown circled in Fig. 10.65, has a Gaussian intensity distribution spread over a small (9-pixel) area, and is not easily discernible by eye. Figure 10.66 shows

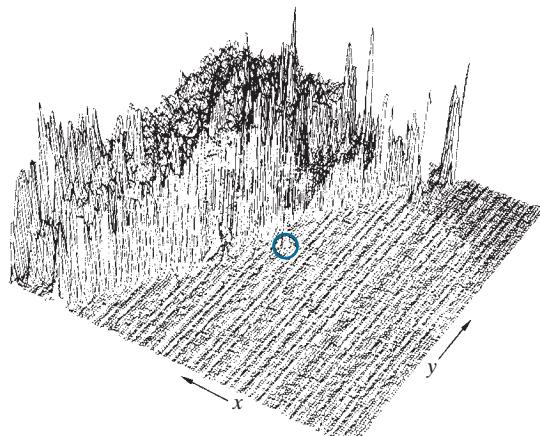
FIGURE 10.64
LANDSAT
frame. (Cowart,
Snyder, and
Ruedger.)



the results of computing Eqs. (10-121) and (10-122) with $a_1 = 6$ and $a_2 = 4$, respectively. The peak at $u_1 = 3$ in Fig. 10.66(a) yields $V_1 = 0.5$ from Eq. (10-123). Similarly, the peak at $u_2 = 4$ in Fig. 10.66(b) yields $V_2 = 1.0$ from Eq. (10-124).

Guidelines for selecting a_1 and a_2 can be explained with the aid of Fig. 10.66. For instance, suppose that we had used $a_2 = 15$ instead of $a_2 = 4$. In that case, the peaks in Fig. 10.66(b) would now be at $u_2 = 15$ and 17 because $V_2 = 1.0$. This would be a seriously aliased result. As discussed in Section 4.5, aliasing is caused by under-sampling (too few frames in the present discussion, as the range of u is determined by K). Because $u = aV$, one possibility is to select a as the integer closest to $a = u_{\max}/V_{\max}$,

FIGURE 10.65
Intensity plot of
the image in
Fig. 10.64, with
the target circled.
(Rajala, Riddle,
and Snyder.)



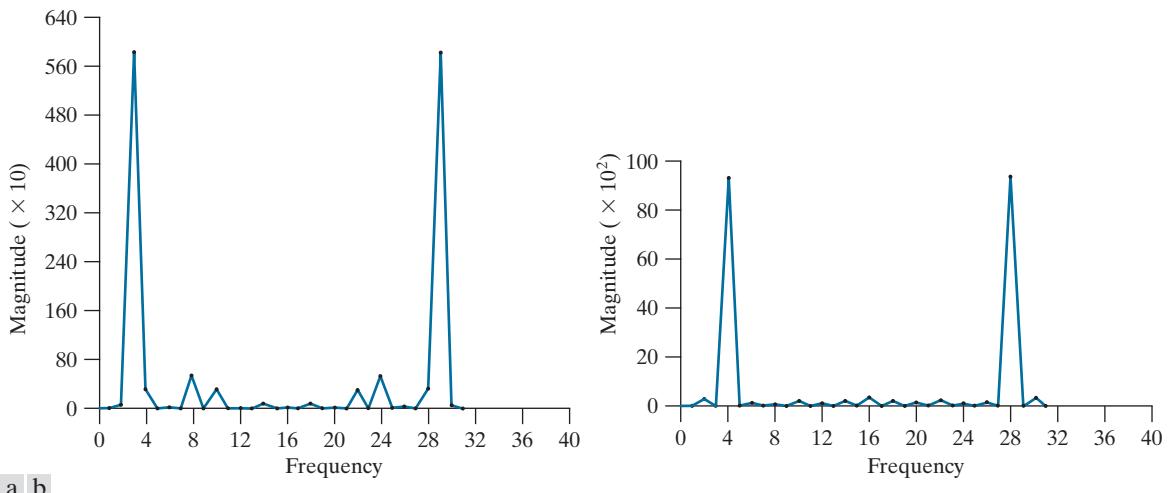


FIGURE 10.66 (a) Spectrum of Eq. (10-121) showing a peak at $u_1 = 3$. (b) Spectrum of Eq. (10-122) showing a peak at $u_2 = 4$. (Rajala, Riddle, and Snyder.)

where u_{\max} is the aliasing frequency limitation established by K , and V_{\max} is the maximum expected object velocity.

Summary, References, and Further Reading

Because of its central role in autonomous image processing, segmentation is a topic covered in most books dealing with image processing, image analysis, and computer vision. The following books provide complementary and/or supplementary reading for our coverage of this topic: Umbaugh [2010]; Prince [2012]; Nixon and Aguado, A [2012]; Pratt [2014]; and Petrou and Petrou [2010].

Work dealing with the use of kernels to detect intensity discontinuities (see Section 10.2) has a long history. Numerous kernels have been proposed over the years: Roberts [1965]; Prewitt [1970]; and Kirsh [1971]. The Sobel operators are from [Sobel]; see also Danielsson and Seger [1990]. Our presentation of the zero-crossing properties of the Laplacian is based on Marr [1982]. The Canny edge detector discussed in Section 10.2 is due to Canny [1986]. The basic reference for the Hough transform is Hough [1962]. See Ballard [1981], for a generalization to arbitrary shapes.

Other approaches used to deal with the effects of illumination and reflectance on thresholding are illustrated by the work of Perez and Gonzalez [1987], Drew et al. [1999], and Toro and Funt [2007]. The optimum thresholding approach due to Otsu [1979] has gained considerable acceptance because it combines excellent performance with simplicity of implementation, requiring only estimation of image histograms. The basic idea of using preprocessing to improve thresholding dates back to an early paper by White and Rohrer [1983]), which combined thresholding, the gradient, and the Laplacian in the solution of a difficult segmentation problem.

See Fu and Mui [1981] for an early survey on the topic of region-oriented segmentation. The work of Haddon and Boyce [1990] and of Pavlidis and Liow [1990] are among the earliest efforts to integrate region and boundary information for the purpose of segmentation. Region growing is still an active area of research in image processing, as exemplified by Liangjia et al. [2013]. The basic reference on the k -means algorithm presented in Section 10.5 goes way back several decades to an obscure 1957 Bell Labs report by Lloyd, who subsequently published in Lloyd [1982]. This algorithm was already being in used in areas such as pattern recognition in the 1960s and '70s (Tou and

Gonzalez [1974]). The superpixel algorithm presented in Section 10.5 is from Achanta et al. [2012]. See their paper for a listing and comparison of other superpixel approaches. The material on graph cuts is based on the paper by Shi and Malik [2000]. See Hochbaum [2010] for an example of faster implementations.

Segmentation by watersheds was shown in Section 10.7 to be a powerful concept. Early references dealing with segmentation by watersheds are Serra [1988], and Beucher and Meyer [1992]. As indicated in our discussion in Section 10.7, one of the key issues with watersheds is the problem of over-segmentation. The papers by Bleau and Leon [2000] and by Gaetano et al. [2015] are illustrative of approaches for dealing with this problem.

The material in Section 10.8 dealing with accumulative differences is from Jain, R. [1981]. See also Jain, Kasturi, and Schunck [1995]. The material dealing with motion via Fourier techniques is from Rajala, Riddle, and Snyder [1983]. The books by Snyder and Qi [2004], and by Chakrabarti et al. [2015], provide additional reading on motion estimation. For details on the software aspects of many of the examples in this chapter, see Gonzalez, Woods, and Eddins [2009].

Problems

Solutions to the problems marked with an asterisk (*) are in the DIP4E Student Support Package (consult the book website: www.ImageProcessingPlace.com).

10.1* In a Taylor series approximation, the *remainder* (also called the *truncation error*) consists of all the terms not used in the approximation. The first term in the remainder of a finite difference approximation is indicative of the error in the approximation. The higher the derivative order of that term is, the lower the error will be in the approximation. All three approximations to the first derivative given in Eqs. (10-4)-(10-6) are computed using the same number of sample points. However, the error of the central difference approximation is less than the other two. Show that this is true.

10.2 Do the following:

- (a)* Show how Eq. (10-8) was obtained.
- (b) Show how Eq. (10-9) was obtained.

10.3 A binary image contains straight lines oriented horizontally, vertically, at 45° , and at -45° . Give a set of 3×3 kernels that can be used to detect one-pixel breaks in these lines. Assume that the intensities of the lines and background are 1 and 0, respectively.

10.4 Propose a technique for detecting gaps of length ranging between 1 and K pixels in line segments of a binary image. Assume that the lines are one pixel thick. Base your technique on 8-neighbor connectivity analysis, rather than attempting to construct kernels for detecting the gaps.

10.5* With reference to Fig. 10.6, what are the angles (measured with respect to the x -axis of the book axis convention in Fig. 2.19) of the horizontal and vertical lines to which the kernels in Figs. 10.6(a) and (c) are most responsive?

10.6 Refer to Fig. 10.7 in answering the following questions.

- (a)* Some of the lines joining the pads and center element in Fig. 10.7(e) are single lines, while others are double lines. Explain why.
- (b) Propose a method for eliminating the components in Fig. 10.7(f) that are not part of the line oriented at -45° .

(c)

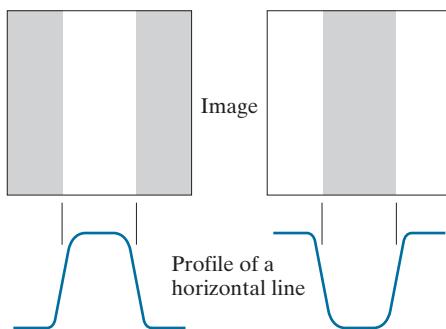
10.7 With reference to the edge models in Fig. 10.8, answer the following without generating the gradient and angle images. Simply provide sketches of the profiles that show what you would expect the profiles of the magnitude and angle images to look like.

- (a)* Suppose that we compute the gradient magnitude of each of these models using the Prewitt kernels in Fig. 10.14. Sketch what a horizontal profile through the center of each gradient image would look like.
- (b) Sketch a horizontal profile for each corresponding angle image.

10.8 Consider a horizontal intensity profile through

the middle of a binary image that contains a vertical step edge through the center of the image. Draw what the profile would look like after the image has been blurred by an averaging kernel of size $n \times n$ with coefficients equal to $1/n^2$. For simplicity, assume that the image was scaled so that its intensity levels are 0 on the left of the edge and 1 on its right. Also, assume that the size of the kernel is much smaller than the image, so that image border effects are not a concern near the center of the image.

- 10.9*** Suppose that we had used the edge models in the following image, instead of the ramp in Fig. 10.10. Sketch the gradient and Laplacian of each profile.



- 10.10** Do the following:

(a)* Show that the *direction* of steepest (maximum) ascent of a function f at point (x, y) is given by the vector $\nabla f(x, y)$ in Eq. (10-16), and that the rate of that descent is $\|\nabla f(x, y)\|$, defined in Eq. (10-17).

(b) Show that the *direction of steepest descent* is given by the vector $-\nabla f(x, y)$, and that the *rate* of the steepest descent is $\|\nabla f(x, y)\|$.

(c) Give the description of an image whose gradient magnitude image would be the same, whether we computed it using Eq. (10-17) or (10-26). A constant image is not acceptable answer.

- 10.11** Do the following.

(a) How would you modify the Sobel and Prewitt kernels in Fig. 10.14 so that they give their strongest gradient response for edges oriented at $\pm 45^\circ$?

(b)* Show that the Sobel and Prewitt kernels

in Fig. 10.14, and in (a) above, and give isotropic results only for horizontal and vertical edges, and for edges oriented at $\pm 45^\circ$, respectively.

- 10.12** The results obtained by a single pass through an image of some 2-D kernels can be achieved also by two passes using 1-D kernels. For example, the same result of using a 3×3 smoothing kernel with coefficients $1/9$ can be obtained by a pass of the kernel $[1 \ 1 \ 1]$ through an image, followed by a pass of the result with the kernel $[1 \ 1 \ 1]^T$. The final result is then scaled by $1/9$. Show that the response of Sobel kernels (Fig. 10.14) can be implemented similarly by one pass of the differencing kernel $[-1 \ 0 \ 1]$ (or its vertical counterpart) followed by the smoothing kernel $[1 \ 2 \ 1]$ (or its vertical counterpart).

- 10.13** A popular variation of the compass kernels shown in Fig. 10.15 is based on using coefficients with values 0, 1, and -1 .

(a)* Give the form of the eight compass kernels using these coefficients. As in Fig. 10.15, let N, NW, ... denote the direction of the edge that gives the strongest response.

(b) Specify the gradient vector direction of the edges detected by each kernel in (a).

- 10.14** The rectangle in the following binary image is of size $m \times n$ pixels.



(a)* What would the magnitude of the gradient of this image look like based on using the approximation in Eq. (10-26)? Assume that g_x and g_y are obtained using the Sobel kernels. Show all relevant different pixel values in the gradient image.

(b) With reference to Eq. (10-18) and Fig. 10.12,

sketch the histogram of edge *directions*. Be precise in labeling the height of each component of the histogram.

- (c) What would the Laplacian of this image look like based on using Eq. (10-14)? Show all relevant different pixel values in the Laplacian image.

10.15 Suppose that an image $f(x,y)$ is convolved with a kernel of size $n \times n$ (with coefficients $1/n^2$) to produce a smoothed image $\bar{f}(x,y)$.

- (a)* Derive an expression for edge strength (edge magnitude) as a function of n . Assume that n is odd and that the partial derivatives are computed using Eqs. (10-19) and (10-20).
 (b) Show that the ratio of the maximum edge strength of the smoothed image to the maximum edge strength of the original image is $1/n$. In other words, edge strength is inversely proportional to the size of the smoothing kernel, as one would expect.

10.16 With reference to Eq. (10-29),

- (a)* Show that the average value of the LoG operator, $\nabla^2 G(x,y)$, is zero.
 (b) Show that the average value of any image convolved with this operator also is zero. (*Hint:* Consider solving this problem in the frequency domain, using the convolution theorem and the fact that the average value of a function is proportional to its Fourier transform evaluated at the origin.)
 (c) Suppose that we: (1) used the kernel in Fig. 10.4(a) to approximate the Laplacian of a Gaussian, and (2) convolved this result with any image. What would be true in general of the values of the resulting image? Explain. (*Hint:* Take a look at Problem 3.32.)

10.17 Refer to Fig. 10.22(c).

- (a) Explain why the edges form closed contours.
 (b)* Does the zero-crossing method for finding edge location always result in closed contours? Explain.

10.18 One often finds in the literature a derivation of the Laplacian of a Gaussian (LoG) that starts with the expression

$$G(r) = e^{-r^2/2\sigma^2}$$

where $r^2 = x^2 + y^2$. The LoG is then derived by taking the second partial derivative with respect to r : $\nabla^2 G(r) = \partial^2 G(r)/\partial r^2$. Finally, $x^2 + y^2$ is substituted for r^2 to get the final (*incorrect*) result:

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

Derive this result and explain the reason for the difference between this expression and Eq. (10-29).

10.19 Do the following:

- (a)* Derive Eq. (10-33).

- (b) Let $k = \sigma_1/\sigma_2$ denote the standard deviation ratio discussed in connection with the DoG function, and express Eq. (10-33) in terms of k and σ_2 .

10.20 In the following, assume that G and f are discrete arrays of size $n \times n$ and $M \times N$, respectively.

- (a) Show that the 2-D convolution of the Gaussian function $G(x,y)$ in Eq. (10-27) with an image $f(x,y)$ can be expressed as a 1-D convolution along the rows (columns) of $f(x,y)$, followed by a 1-D convolution along the columns (rows) of the result. (*Hints:* See Section 3.4 regarding discrete convolution and separability.)

- (b)* Derive an expression for the computational advantage using the 1-D convolution approach in (a) as opposed to implementing the 2-D convolution directly. Assume that $G(x,y)$ is sampled to produce an array of size $n \times n$ and that $f(x,y)$ is of size $M \times N$. The computational advantage is the ratio of the number of multiplications required for 2-D convolution to the number required for 1-D convolution. (*Hint:* Review the subsection on separable kernels in Section 3.4.)

10.21 Do the following.

- (a) Show that Steps 1 and 2 of the Marr-Hildreth algorithm can be implemented using four 1-D convolutions. (*Hints:* Refer to Problem 10.20(a) and express the Laplacian operator as the sum of two partial derivatives, given

by Eqs. (10-10) and (10-11), and implement each derivative using a 1-D kernel, as in Problem 10.12.)

- (b) Derive an expression for the computational advantage of using the 1-D convolution approach in (a) as opposed to implementing the 2-D convolution directly. Assume that $G(x, y)$ is sampled to produce an array of size $n \times n$ and that $f(x, y)$ is of size $M \times N$. The computational advantage is the ratio of the number of multiplications required for 2-D convolution to the number required for 1-D convolution (see Problem 10.20).

10.22 Do the following.

- (a)* Formulate Step 1 and the gradient magnitude image computation in Step 2 of the Canny algorithm using 1-D instead of 2-D convolutions.
- (b) What is the computational advantage of using the 1-D convolution approach as opposed to implementing a 2-D convolution. Assume that the 2-D Gaussian filter in Step 1 is sampled into an array of size $n \times n$ and that the input image is of size $M \times N$. Express the computational advantage as the ratio of the number of multiplications required by each method.

10.23 With reference to the three vertical edge models and corresponding profiles in Fig. 10.8 provide sketches of the profiles that would result from each of the following methods. You may sketch the profiles manually.

- (a)* Suppose that we compute the gradient magnitude of each of the three edge model images using the Sobel kernels. Sketch the horizontal intensity profiles of the three resulting gradient images.
- (b) Sketch the horizontal intensity profiles that would result from using the 3×3 Laplacian kernel in Fig. 10.10.4(a).
- (c)* Repeat (b) using only the first two steps of the Marr-Hildreth edge detector.
- (d) Repeat (b) using the first two steps of the Canny edge detector. You may ignore the angle images.

- (e) Sketch the horizontal profiles of the angle images resulting from using the Canny edge detector.

10.24 In Example 10.9, we used a smoothing kernel of size 19×19 to generate Fig. 10.26(c) and a kernel of size 13×13 to generate Fig. 10.26(d). What was the rationale that led to choosing these values? (*Hint:* Observe that both are Gaussian kernels, and refer to the discussion of lowpass Gaussian kernels in Section 3.5.)

10.25 Refer to the Hough transform in Section 10.2.

- (a) Propose a general procedure for obtaining the normal representation of a line from its slope-intercept form, $y = ax + b$.
- (b)* Find the normal representation of the line $y = -2x + 1$.

10.26 Refer to the Hough transform in Section 10.2.

- (a)* Explain why the Hough mapping of the point labeled 1 in Fig. 10.30(a) is a straight line in Fig. 10.30(b).
- (b)* Is this the only point that would produce that result? Explain.
- (c) Explain the reflective adjacency relationship illustrated by, for example, the curve labeled Q in Fig. 10.30(b).

10.27 Show that the number of operations required to implement the accumulator-cell approach discussed in Section 10.2 is linear in n , the number of non-background points in the image plane (i.e., the xy -plane).

10.28 An important application of image segmentation is in processing images resulting from so-called *bubble chamber* events. These images arise from experiments in high-energy physics in which a beam of particles of known properties is directed onto a target of known nuclei. A typical event consists of incoming tracks, any one of which, upon a collision, branches out into secondary tracks of particles emanating from the point of collision. Propose a segmentation approach for detecting all tracks angled at any of the following six directions off the horizontal: $\pm 25^\circ$, $\pm 50^\circ$, and $\pm 75^\circ$. The estimation error allowed in any of these six directions is $\pm 5^\circ$. For a track to be valid it must be at least 100 pixels long and have no more than three gaps, each not exceeding 10 pixels. You may

assume that the images have been preprocessed so that they are binary and that all tracks are 1 thick, except at the point of collision from which they emanate. Your procedure should be able to differentiate between tracks that have the same direction but different origins. (*Hint:* Base your solution on the Hough transform.)

10.29* Restate the basic global thresholding algorithm in Section 10.3 so that it uses the histogram of an image instead of the image itself.

10.30* Prove that the basic global thresholding algorithm in Section 10.3 converges in a finite number of steps. (*Hint:* Use the histogram formulation from Problem 10.29.)

10.31 Give an explanation why the initial threshold in the basic global thresholding algorithm in Section 10.3 must be between the minimum and maximum values in the image. (*Hint:* Construct an example that shows the algorithm failing for a threshold value selected outside this range.)

10.32* Assume that the initial threshold in the basic global thresholding algorithm in Section 10.3 is selected as a value between the minimum and maximum intensity values in an image. Do you think the final value of the threshold at convergence depends on the specific initial value used? Explain. (You can use a simple image example to support your conclusion.)

10.33 You may assume in both of the following cases that the initial threshold is in the open interval $(0, L - 1)$.

(a)* Show that if the histogram of an image is uniform over all possible intensity levels, the basic global thresholding algorithm converges to the average intensity of the image.

(b) Show that if the histogram of an image is bimodal, with identical modes that are symmetric about their means, then the basic global thresholding algorithm will converge to the point halfway between the means of the modes.

10.34 Refer to the basic global thresholding algorithm in Section 10.3. Assume that in a given problem, the histogram is bimodal with modes that are Gaussian curves of the form $A_1 \exp[-(z - m_1)^2 / 2\sigma_1^2]$ and $A_2 \exp[-(z - m_2)^2 / 2\sigma_2^2]$. Assume that m_1 is

greater than m_2 , and that the initial T is between the max and min image intensities. Give conditions (in terms of the parameters of these curves) for the following to be true when the algorithm converges:

(a)* The threshold is equal to $(m_1 + m_2)/2$.

(b)* The threshold is to the left of m_2 .

(c) The threshold is in the interval given by the equation $(m_1 + m_2)/2 < T < m_1$.

10.35 Do the following:

(a)* Show how the first line in Eq. (10-60) follows from Eqs. (10-55), (10-56), and (10-59).

(b) Show how the second line in Eq. (10-60) follows from the first.

10.36 Show that a maximum value for Eq. (10-63) always exists for k in the range $0 \leq k \leq L - 1$.

10.37* With reference to Eq. (10-65), advance an argument that establishes that $0 \leq \eta(k) \leq 1$ for k in the range $0 \leq k \leq L - 1$, where the minimum is achievable only by images with constant intensity, and the maximum occurs only for 2-valued images with values 0 and $(L - 1)$.

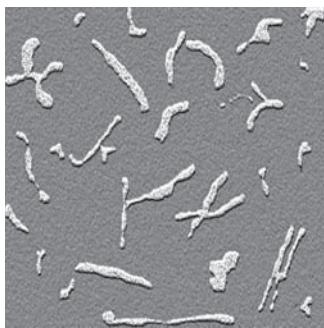
10.38 Do the following:

(a)* Suppose that the intensities of a digital image $f(x, y)$ are in the range $[0, 1]$ and that a threshold, T , successfully segmented the image into objects and background. Show that the threshold $T' = 1 - T$ will successfully segment the negative of $f(x, y)$ into the same regions. The term negative is used here in the sense defined in Section 3.2.

(b) The intensity transformation function in (a) that maps an image into its negative is a linear function with negative slope. State the conditions that an arbitrary intensity transformation function must satisfy for the segmentability of the original image with respect to a threshold, T , to be preserved. What would be the value of the threshold after the intensity transformation?

10.39 The objects and background in the image below have a mean intensity of 170 and 60, respectively, on a $[0, 255]$ scale. The image is corrupted by Gaussian noise with 0 mean and a standard deviation of 10 intensity levels. Propose a thresholding

method capable of a correct segmentation rate of 90% or higher. (Recall that 99.7% of the area of a Gaussian curve lies in a $\pm 3\sigma$ interval about the mean, where σ is the standard deviation.)

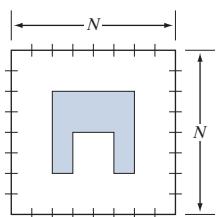


- 10.40** Refer to the intensity ramp image in Fig. 10.34(b) and the moving-average algorithm discussed in Section 10.3. Assume that the image is of size 500×700 pixels and that its minimum and maximum values are 0 and 1, where 0's are contained only in the first column.

- (a)* What would be the result of segmenting this image with the moving-average algorithm using $b = 0$ and an arbitrary value for n . Explain what the segmented image would look like.
- (b) Now reverse the direction of the ramp so that its leftmost value is 1 and the rightmost value is 0 and repeat (a).
- (c) Repeat (a) but with $b = 1$ and $n = 2$.
- (d) Repeat (a) but with $b = 1$ and $n = 100$.

- 10.41** Propose a region-growing algorithm to segment the image in Problem 10.39.

- 10.42*** Segment the image shown by using the split and merge procedure discussed in Section 10.4. Let $Q(R_i) = \text{TRUE}$ if all pixels in R_i have the same intensity. Show the quadtree corresponding to your segmentation.



- 10.43** Consider the region of 1's resulting from the segmentation of the sparse regions in the image of the Cygnus Loop in Example 10.21. Propose a technique for using this region as a mask to isolate the three main components of the image: (1) background; (2) dense inner region; and (3) sparse outer region.

- 10.44** Let the pixels in the first row of a 3×3 image, like the one in Fig. 10.53(a), be labeled as 1, 2, 3, and the pixels in the second and third rows be labeled as 4, 5, 6 and 7, 8, 9, respectively. Let the intensity of these pixels be [90, 80, 30; 70, 5, 20; 80, 20, 30] where, for example, the intensity of pixel 2 is 80 and of pixel 4 it is 70. Compute the weights for the edges for the graph in Fig. 10.53(c), using the formula $w(i,j) = 30[1/(|I(n_i) - I(n_j)| + c)]$ explained in the text in connection with that figure (we scaled the formula by 30 to make the numerical results easier to interpret). Let $c = 0$ in this case.

- 10.45*** Show how Eqs. (10-106) through (10-108) follow from Eq. (10-105).

- 10.46** Demonstrate the validity of Eq. (10-102).

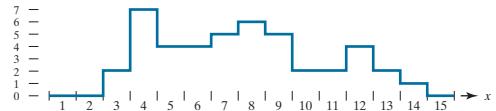
- 10.47** Refer to the discussion in Section 10.7.

- (a)* Show that the elements of $C_n(M_i)$ and $T[n]$ are never replaced during execution of the watershed segmentation algorithm.

- (b) Show that the number of elements in sets $C_n(M_i)$ and $T[n]$ either increases or remains the same as n increases.

- 10.48** You saw in Section 10.7 that the boundaries obtained using the watershed segmentation algorithm form closed loops (for example, see Figs. 10.59 and 10.61). Advance an argument that establishes whether or not closed boundaries *always* result from application of this algorithm.

- 10.49*** Give a step-by-step implementation of the dam-building procedure for the one-dimensional intensity cross section shown below. Show a drawing of the cross section at each step, showing "water" levels and dams constructed.



10.50 What would the negative ADI image shown in Fig. 10.62(c) look like if we tested against T (instead of testing against $-T$) in Eq. (10-117)?

10.51 Are the following statements true or false? Explain the reason for your answer in each.

(a)* The nonzero entries in the absolute ADI continue to grow in dimension, provided that the object is moving.

(b) The nonzero entries in the positive ADI always occupy the same area, regardless of the motion undergone by the object.

(c) The nonzero entries in the negative ADI continue to grow in dimension, provided that the object is moving.

10.52 Suppose that in Example 10.29 motion along the x -axis is set to zero. The object now moves only along the y -axis at 1 pixel per frame for 32 frames and then (instantaneously) reverses direction and moves in exactly the opposite direction for another 32 frames. What would Figs. 10.66(a) and (b) look like under these conditions?

10.53* Advance an argument that demonstrates that when the signs of S_{1x} and S_{2x} in Eqs. (10-125) and (10-126) are the same, velocity component V_1 is positive.

10.54 An automated pharmaceutical plant uses image processing to measure the shapes of medication tablets for the purpose of quality control. The segmentation stage of the system is based on Otsu's method. The speed of the inspection lines is so high that a very high rate flash illumination is required to "stop" motion. When new, the illumination lamps project a uniform pattern of light. However, as the lamps age, the illumination pattern deteriorates as a function of time and spatial coordinates according to the equation

$$i(x, y) = A(t) - t^2 e^{-[(x - M/2)^2 + (y - N/2)^2]}$$

where $(M/2, N/2)$ is the center of the viewing area and t is time measured in increments of months. The lamps are still experimental and the behavior of $A(t)$ is not fully understood by

the manufacturer. All that is known is that, during the life of the lamps, $A(t)$ is always greater than the negative component in the preceding equation because illumination cannot be negative. It has been observed that Otsu's algorithm works well when the lamps are new, and their pattern of illumination is nearly constant over the entire image. However, segmentation performance deteriorates with time. Being experimental, the lamps are exceptionally expensive, so you are employed as a consultant to help solve the problem using digital image processing techniques to compensate for the changes in illumination, and thus extend the useful life of the lamps. You are given flexibility to install any special markers or other visual cues in the viewing area of the imaging cameras. Propose a solution in sufficient detail that the engineering plant manager can understand your approach. (*Hint:* Review the image model discussed in Section 2.3 and consider using one or more targets of known reflectivity.)

10.55 The speed of a bullet in flight is to be estimated by using high-speed imaging techniques. The method of choice involves the use of a CCD camera and flash that exposes the scene for K seconds. The bullet is 2.5 cm long, 1 cm wide, and its range of speed is 750 ± 250 m/s. The camera optics produce an image in which the bullet occupies 10% of the horizontal resolution of a 256×256 digital image.

(a)* Determine the maximum value of K that will guarantee that the blur from motion does not exceed 1 pixel.

(b) Determine the minimum number of frames per second that would have to be acquired in order to guarantee that at least two complete images of the bullet are obtained during its path through the field of view of the camera.

(c)* Propose a segmentation procedure for automatically extracting the bullet from a sequence of frames.

(d) Propose a method for automatically determining the speed of the bullet.



Feature Extraction

Well, but reflect; have we not several times acknowledged that names rightly given are the likenesses and images of the things which they name?

Socrates

Preview

After an image has been segmented into regions or their boundaries using methods such as those in Chapters 10 and 11, the resulting sets of segmented pixels usually have to be converted into a form suitable for further computer processing. Typically, the step after segmentation is *feature extraction*, which consists of feature detection and feature description. *Feature detection* refers to finding the features in an image, region, or boundary. *Feature description* assigns quantitative attributes to the detected features. For example, we might *detect* corners in a region boundary, and *describe* those corners by their orientation and location, both of which are quantitative attributes. Feature processing methods discussed in this chapter are subdivided into three principal categories, depending on whether they are applicable to boundaries, regions, or whole images. Some features are applicable to more than one category. Feature descriptors should be as insensitive as possible to variations in parameters such as scale, translation, rotation, illumination, and viewpoint. The descriptors discussed in this chapter are either insensitive to, or can be normalized to compensate for, variations in one or more of these parameters.

Upon completion of this chapter, readers should:

- Understand the meaning and applicability of a broad class of features suitable for image processing.
- Understand the concepts of feature vectors and feature space, and how to relate them to the various descriptors developed in this chapter.
- Be skilled in the mathematical tools used in feature extraction algorithms.
- Be familiar with the limitations of the various feature extraction methods discussed.
- Understand the principal steps used in the solution of feature extraction problems.
- Be able to formulate feature extraction algorithms.
- Have a “feel” for the types of features that have a good chance of success in a given application.

11.1 BACKGROUND

Although there is no universally accepted, formal definition of what constitutes an *image feature*, there is little argument that, intuitively, we generally think of a feature as a distinctive attribute or description of “something” we want to label or differentiate. For our purposes, the key words here are *label* and *differentiate*. The “something” of interest in this chapter refers either to individual image objects, or even to entire images or sets of images. Thus, we think of features as attributes that are going to help us assign unique labels to objects in an image or, more generally, are going to be of value in differentiating between entire images or families of images.

There are two principal aspects of image *feature extraction*: *feature detection*, and *feature description*. That is, when we refer to feature extraction, we are referring to both detecting the features and then describing them. To be useful, the extraction process must encompass both. The terminology you are likely to encounter in image processing and analysis to describe feature detection and description varies, but a simple example will help clarify our use of these terms. Suppose that we use object corners as features for some image processing task. In this chapter, detection refers to *finding* the corners in a region or image. Description, on the other hand, refers to *assigning quantitative* (or sometimes *qualitative*) *attributes* to the detected features, such as corner orientation, and location with respect to other corners. In other words, knowing that there are corners in an image has limited use without additional information that can help us differentiate between objects in an image, or between images, based on corners and their attributes.

Given that we want to use features for purposes of differentiation, the next question is: What are the important characteristics that these features must possess in the realm of digital image processing? You are already familiar with some of these characteristics. In general, features should be independent of location, rotation, and scale. Other factors, such as independence of illumination levels and changes caused by the viewpoint between the imaging sensor(s) and the scene, also are important. Whenever possible, preprocessing should be used to normalize input images before feature extraction. For example, in situations where changes in illumination are severe enough to cause difficulties in feature detection, it would make sense to preprocess an image to compensate for those changes. Histogram equalization or specification come to mind as automatic techniques that we know are helpful in this regard. The idea is to use as much *a priori* information as possible to preprocess images in order to improve the chances of accurate feature extraction.

When used in the context of a feature, the word “independent” usually has one of two meanings: invariant or covariant. A feature descriptor is *invariant* with respect to a set of transformations if its value remains unchanged after the application (to the entity being described) of any transformation from the family. A feature descriptor is *covariant* with respect to a set of transformations if applying to the entity any transformation from the set produces the same result in the descriptor. For example, consider this set of affine transformations: {*translation*, *reflection*, *rotation*}, and suppose that we have an elliptical region to which we assign the feature descriptor *area*. Clearly, applying any of these transformations to the region does not change its area.

See Table 2.3 regarding affine transformations.

Therefore, *area* is an invariant feature descriptor with respect to the given family of transformations. However, if we add the affine transformation *scaling* to the family, descriptor *area* ceases to be invariant with respect to the extended family. The descriptor is now covariant with respect to the family, because scaling the area of the region by any factor scales the value of the descriptor by the same factor. Similarly, the descriptor *direction* (of the principal axis of the region) is covariant because rotating the region by any angle has the same effect on the value of the descriptor. Most of the feature descriptors we use in this chapter are covariant in general, in the sense that they may be invariant to some transformations of interest, but not to others that may be equally as important. As you will see shortly, it is good practice to normalize as many relevant invariances as possible out of covariances. For instance, we can compensate for changes in direction of a region by computing its actual direction and rotating the region so that its principal axis points in a predefined direction. If we do this for every region detected in an image, *rotation* will cease to be covariant.

Another major classification of features is *local* vs. *global*. You are likely to see many different attempts to classify features as belonging to one of these two categories. What makes this difficult is that a feature may belong to both, depending on the application. For example, consider the descriptor *area* again, and suppose that we are applying it to the task of inspecting the degree to which bottles moving past an imaging sensor on a production line are full of liquid. The sensor and its accompanying software are capable of generating images of ten bottles at once, in which liquid in each bottle appears as a bright region, and the rest of the image appears as dark background. The area of a region in this fixed geometry is directly proportional to the amount of liquid in a bottle and, if detected and measured reliably, *area* is the only feature we need to solve the inspection problem. Each image has ten regions, so we consider area to be a local feature, in the sense that it is applicable to *individual* elements (regions) of an image. If the problem were to detect the *total* amount (area) of liquid in an image, we would now consider area to be a global descriptor. But the story does not end there. Suppose that the liquid inspection task is redefined so that it calculates the entire amount of liquid per day passing by the imaging station. We no longer care about the area of individual regions per se. Our units now are images. If we know the total area in an image, and we know the number of images, calculating the total amount of liquid in a day is trivial. Now the area of an entire image is a local feature, and the area of the total at the end of the day is global. Obviously, we could redefine the task so that the area at the end of a day becomes a local feature descriptor, and the area for all assembly lines becomes a global measure. And so on, endlessly. In this chapter, we call a feature *local* if it applies to a member of a set, and *global* if it applies to the entire set, where “member” and “set” are determined by the application.

Features by themselves are seldom generated for human consumption, except in applications such as interactive image processing, topics that are not in the mainstream of this book. In fact, as you will see later, some feature extraction methods generate tens, hundreds, or even thousands of descriptor values that would appear meaningless if examined visually. Instead, feature description typically is used as a preprocessing step for higher-level tasks, such as image registration, object

recognition for automated inspection, searching for patterns (e.g., individual faces and/or fingerprints) in image databases, and autonomous applications, such as robot and vehicle navigation. For these applications, numerical features usually are “packaged” in the form of a *feature vector*, (i.e., a $1 \times n$ or $n \times 1$ matrix) whose elements are the descriptors. An RGB image is one of the simplest examples. As you know from Chapter 6, each pixel of an RGB image can be expressed as 3-D vector,

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

in which x_1 is the intensity value of the red image at a point, and the other components are the intensity values of the green and blue images at the same point. If color is used as a feature, then a region in an RGB image would be represented as a set of feature vectors (points) in 3-D space. When n descriptors are used, feature vectors become n -dimensional, and the space containing them is referred to as an *n-dimensional feature space*. You may “visualize” a set of n -dimensional feature vectors as a “hypercloud” of points in n -dimensional Euclidean space.

In this chapter, we group features into three principal categories: *boundary*, *region*, and *whole image* features. This subdivision is not based on the applicability of the methods we are about to discuss; rather, it is based on the fact that some categories make more sense than others when considered in the context of what is being described. For example, it is implied that when we refer to the “length of a boundary” we are referring to the “length of the boundary of a region,” but it makes no sense to refer to the “length” of an image. It will become clear that many of the features we will be discussing are applicable to boundaries and regions, and some apply to whole images as well.

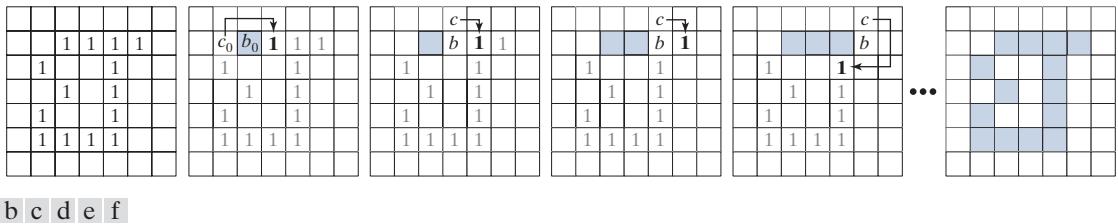
11.2 BOUNDARY PREPROCESSING

The segmentation techniques discussed in the previous two chapters yield raw data in the form of pixels along a boundary or pixels contained in a region. It is standard practice to use schemes that compact the segmented data into representations that facilitate the computation of descriptors. In this section, we discuss various boundary preprocessing approaches suitable for this purpose.

BOUNDARY FOLLOWING (TRACING)

You will find it helpful to review the discussion in Sections 2.5 on neighborhoods, adjacency and connectivity, and the discussion in Section 9.6 dealing with connected components.

Several of the algorithms discussed in this chapter require that the points in the boundary of a region be ordered in a clockwise or counterclockwise direction. Consequently, we begin our discussion by introducing a boundary-following algorithm whose output is an *ordered* sequence of points. We assume (1) that we are working with binary images in which object and background points are labeled 1 and 0, respectively; and (2) that images are padded with a border of 0's to eliminate the possibility of an object merging with the image border. For clarity, we limit the discussion to single regions. The approach is extended to multiple, disjoint regions by processing the regions individually.



a b c d e f

FIGURE 11.1 Illustration of the first few steps in the boundary-following algorithm. The point to be processed next is labeled in bold, black; the points yet to be processed are gray; and the points found by the algorithm are shaded. Squares without labels are considered background (0) values.

See Section 2.5 for the definition of 4-neighbors, 8-neighbors, and m -neighbors of a point,

The following algorithm traces the boundary of a 1-valued region, R , in a binary image.

1. Let the starting point, b_0 , be the *uppermost-leftmost* point[†] in the image that is labeled 1. Denote by c_0 the *west* neighbor of b_0 [see Fig. 11.1(b)]. Clearly, c_0 is always a background point. Examine the 8-neighbors of b_0 , starting at c_0 and proceeding in a clockwise direction. Let b_1 denote the *first* neighbor encountered whose value is 1, and let c_1 be the (background) point immediately preceding b_1 in the sequence. Store the locations of b_0 for use in Step 5.
2. Let $b = b_0$ and $c = c_0$.
3. Let the 8-neighbors of b , starting at c and proceeding in a clockwise direction, be denoted by n_1, n_2, \dots, n_8 . Find the first neighbor labeled 1 and denote it by n_k .
4. Let $b = n_k$ and $c = n_{k-1}$.
5. Repeat Steps 3 and 4 until $b = b_0$. The sequence of b points found when the algorithm stops is the set of ordered boundary points.

Note that c in Step 4 is always a background point because n_k is the first 1-valued point found in the clockwise scan. This algorithm is referred to as the *Moore boundary tracing algorithm* after Edward F. Moore, a pioneer in cellular automata theory.

Figure 11.1 illustrates the first few steps of the algorithm. It is easily verified (see Problem 11.1) that continuing with this procedure will yield the correct boundary, shown in Fig. 11.1(f), whose points are ordered in a clockwise sequence. The algorithm works equally well with more complex boundaries, such as the boundary with an attached branch in Fig. 11.2(a) or the self-intersecting boundary in Fig. 11.2(b). Multiple boundaries [Fig. 11.2(c)] are handled by processing one boundary at a time.

If we start with a binary region instead of a boundary, the algorithm extracts the *outer boundary* of the region. Typically, the resulting boundary will be one pixel thick, but not always [see Problem 11.1(b)]. If the objective is to find the boundaries of holes in a region (these are called the *inner* or *interior boundaries* of the region),

[†]As you will see later in this chapter and in Problem 11.8, the uppermost-leftmost point in a 1-valued boundary has the important property that a polygonal approximation to the boundary has a convex vertex at that location. Also, the left and north neighbors of the point are guaranteed to be background points. These properties make it a good “standard” point at which to start boundary-following algorithms.

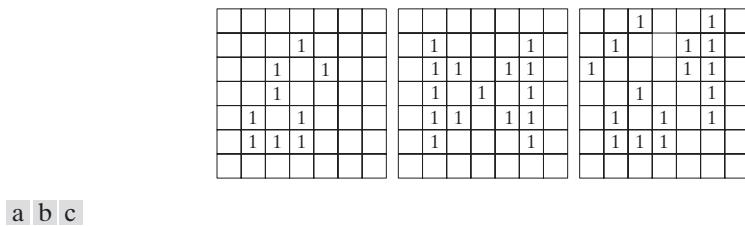


FIGURE 11.2 Examples of boundaries that can be processed by the boundary-following algorithm. (a) Closed boundary with a branch. (b) Self-intersecting boundary. (c) Multiple boundaries (processed one at a time).

a straightforward approach is to extract the holes (see Section 9.6) and treat them as 1-valued regions on a background of 0's. Applying the boundary-following algorithm to these regions will yield the inner boundaries of the original region.

We could have stated the algorithm just as easily based on following a boundary in the counterclockwise direction but you will find it easier to have just one algorithm and then reverse the order of the result to obtain a sequence in the opposite direction. We use both directions interchangeably (but consistently) in the following sections to help you become familiar with both approaches.

CHAIN CODES

Chain codes are used to represent a boundary by a connected sequence of straight-line segments of specified length and direction. We assume in this section that all curves are closed, simple curves (i.e., curves that are closed and not self intersecting).

Freeman Chain Codes

Typically, a chain code representation is based on 4- or 8-connectivity of the segments. The *direction* of each segment is coded by using a numbering scheme, as in Fig. 11.3. A boundary code formed as a sequence of such directional numbers is referred to as a *Freeman chain code*.

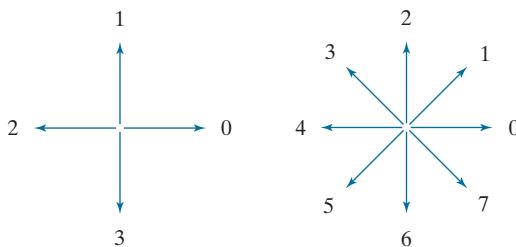
Digital images usually are acquired and processed in a grid format with equal spacing in the *x*- and *y*-directions, so a chain code could be generated by following a boundary in, say, a clockwise direction and assigning a direction to the segments connecting every pair of pixels. This level of detail generally is not used for two principal reasons: (1) The resulting chain would be quite long and (2) any small disturbances along the boundary due to noise or imperfect segmentation would cause changes in the code that may not be related to the principal shape features of the boundary.

An approach used to address these problems is to resample the boundary by selecting a larger grid spacing, as in Fig. 11.4(a). Then, as the boundary is traversed, a boundary point is assigned to a node of the coarser grid, depending on the proximity of the original boundary point to that node, as in Fig. 11.4(b). The resampled boundary obtained in this way can be represented by a 4- or 8-code. Figure 11.4(c) shows the coarser boundary points represented by an 8-directional chain code. It is a simple matter to convert from an 8-code to a 4-code and vice versa (see Problems 2.15, 9.27,

a b

FIGURE 11.3

Direction numbers for
(a) 4-directional
chain code, and
(b) 8-directional
chain code.



and 9.29). For the same reason mentioned when discussing boundary tracing earlier in this section, we chose the starting point in Fig. 11.4(c) as the uppermost-leftmost point of the boundary, which gives the chain code 0766...1212. As you might suspect, the spacing of the resampling grid is determined by the application in which the chain code is used.

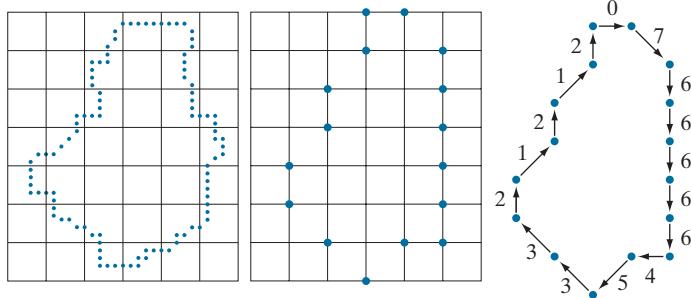
If the sampling grid used to obtain a connected digital curve is a uniform quadrilateral (see Fig. 2.19) all points of a Freeman code based on Fig. 11.3 are guaranteed to coincide with the points of the curve. The same is true if a digital curve is subsampled using the same type of sampling grid, as in Fig. 11.4(b). This is because the samples of curves produced using such grids have the same arrangement as in Fig. 11.3, so all points are reachable as we traverse a curve from one point to the next to generate the code.

The numerical value of a chain code depends on the starting point. However, the code can be normalized with respect to the starting point by a straightforward procedure: We simply treat the chain code as a circular sequence of direction numbers and redefine the starting point so that the resulting sequence of numbers forms an integer of minimum magnitude. We can normalize also for rotation (in angles that are integer multiples of the directions in Fig. 11.3) by using the *first difference* of the chain code instead of the code itself. This difference is obtained by counting the number of direction changes (in a counterclockwise direction in Fig. 11.3) that separate two adjacent elements of the code. If we treat the code as a circular sequence to normalize it with respect to the starting point, then the first element of the difference is computed by using the transition between the last and first components of the chain.

a b c

FIGURE 11.4

- (a) Digital boundary with resampling grid superimposed.
- (b) Result of resampling.
- (c) 8-directional chain-coded boundary.



For instance, the first difference of the 4-directional chain code 10103322 is 3133030. Size normalization can be achieved by altering the spacing of the resampling grid.

The normalizations just discussed are exact only if the boundaries themselves are invariant to rotation (again, in angles that are integer multiples of the directions in Fig. 11.3) and scale change, which seldom is the case in practice. For instance, the same object digitized in two different orientations will have different boundary shapes in general, with the degree of dissimilarity being proportional to image resolution. This effect can be reduced by selecting chain elements that are long in proportion to the distance between pixels in the digitized image, and/or by orienting the resampling grid along the principal axes of the object to be coded, as discussed in Section 11.3, or along its eigen axes, as discussed in Section 11.5.

EXAMPLE 11.1: Freeman chain code and some of its variations.

Figure 11.5(a) shows a 570×570 -pixel, 8-bit gray-scale image of a circular stroke embedded in small, randomly distributed specular fragments. The objective of this example is to obtain a Freeman chain code, the corresponding integer of minimum magnitude, and the first difference of the outer boundary of the stroke. Because the object of interest is embedded in small fragments, extracting its boundary would result in a noisy curve that would not be descriptive of the general shape of the object. As you know, smoothing is a routine process when working with noisy boundaries. Figure 11.5(b) shows the original image smoothed using a box kernel of size 9×9 pixels (see Section 3.5 for a discussion of spatial smoothing), and Fig. 11.5(c) is the result of thresholding this image with a global threshold obtained using Otsu's method. Note that the number of regions has been reduced to two (one of which is a dot), significantly simplifying the problem.

Figure 11.5(d) is the outer boundary of the region in Fig. 11.5(c). Obtaining the chain code of this boundary directly would result in a long sequence with small variations that are not representative of the global shape of the boundary, so we resample it before obtaining its chain code. This reduces insignificant variability. Figure 11.5(e) is the result of using a resampling grid with nodes 50 pixels apart (approximately 10% of the image width) and Fig. 11.5(f) is the result of joining the sample points by straight lines. This simpler approximation retained the principal features of the original boundary.

The 8-directional Freeman chain code of the simplified boundary is

0 0 0 0 6 0 6 6 6 6 6 6 6 4 4 4 4 4 4 2 4 2 2 2 2 2 0 2 2 0 2

The starting point of the boundary is at coordinates (2, 5) in the subsampled grid (remember from Fig. 2.19 that the origin of an image is at its top, left). This is the uppermost-leftmost point in Fig. 11.5(f). The integer of minimum magnitude of the code happens in this case to be the same as the chain code:

0 0 0 0 6 0 6 6 6 6 6 6 4 4 4 4 4 4 4 2 4 2 2 2 2 2 0 2 2 0 2

The first difference of the code is

0 0 0 6 2 6 0 0 0 0 0 0 0 6 0 0 0 0 0 6 2 6 0 0 0 0 6 2 0 6 2 6

Using this code to represent the boundary results in a significant reduction in the amount of data needed to store the boundary. In addition, working with code numbers offers a unified way to analyze the shape of a boundary, as we discuss in Section 11.3. Finally, keep in mind that the subsampled boundary can be recovered from any of the preceding codes.

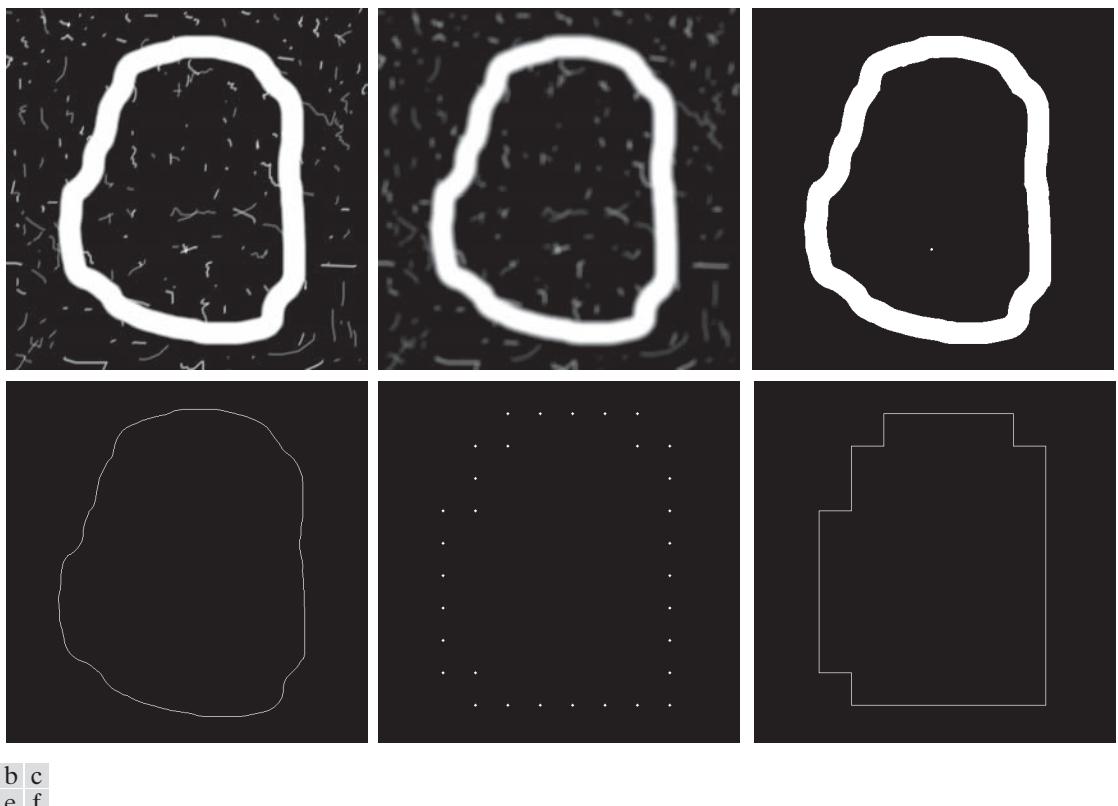


FIGURE 11.5 (a) Noisy image of size 570×570 pixels. (b) Image smoothed with a 9×9 box kernel. (c) Smoothed image, thresholded using Otsu's method. (d) Longest outer boundary of (c). (e) Subsampled boundary (the points are shown enlarged for clarity). (f) Connected points from (e).

Slope Chain Codes

Using Freeman chain codes generally requires resampling a boundary to smooth small variations, a process that implies defining a grid and subsequently assigning all boundary points to their closest neighbors in the grid. An alternative to this approach is to use *slope chain codes* (SCCs) (Bribiesca [1992, 2013]). The SCC of a 2-D curve is obtained by placing straight-line segments of equal length around the curve, with the end points of the segments touching the curve.

Obtaining an SCC requires calculating the *slope changes* between contiguous line segments, and normalizing the changes to the *continuous* (open) interval $(-1, 1)$. This approach requires defining the length of the line segments, as opposed to Freeman codes, which require defining a grid and assigning curve points to it—a much more elaborate procedure. Like Freeman codes, SCCs are independent of rotation, but a larger range of possible slope changes provides a more accurate representation under rotation than the rotational independence of the Freeman codes, which is limited to the eight directions in Fig. 11.3(b). As with Freeman codes, SCCs are independent of translation, and can be normalized for scale changes (see Problem 11.8).

Figure 11.6 illustrates how an SCC is generated. The first step is to select the length of the line segment to use in generating the code [see Fig. 11.6(b)]. Next, a starting point (the origin) is specified (for an open curve, the logical starting point is one of its end points). As Fig. 11.6(c) shows, once the origin has been selected, one end of a line segment is placed at the origin and the other end of the segment is set to coincide with the curve. This point becomes the starting point of the next line segment, and we repeat this procedure until the starting point (or end point in the case of an open curve) is reached. As the figure illustrates, you can think of this process as a sequence of identical circles (with radius equal to the length of the line segment) traversing the curve. The intersections of the circles and the curve determine the nodes of the straight-line approximation to the curve.

Once the intersections of the circles are known, we determine the slope changes between contiguous line segments. Positive and zero slope changes are normalized to the open half interval $[0, 1]$, while negative slope changes are normalized to the open interval $(-1, 0)$. Not allowing slope changes of ± 1 eliminates the implementation issues that result from having to deal with the fact that such changes result in the same line segment with opposite directions.

The sequence of slope changes is the chain that defines the SCC approximation to the original curve. For example, the code for the curve in Fig. 11.6(e) is 0.12, 0.20, 0.21, 0.11, -0.11, -0.12, -0.21, -0.22, -0.24, -0.28, -0.28, -0.31, -0.30. The accuracy of the slope changes defined in Fig. 11.6(d) is 10^{-2} , resulting in an “alphabet” of 199 possible symbols (slope changes). The accuracy can be changed, of course. For instance, and accuracy of 10^{-1} produces an alphabet of 19 symbols (see Problem 11.6). Unlike a Freeman code, there is no guarantee that the last point of the coded curve will coincide with the last point of the curve itself. However, shortening the line

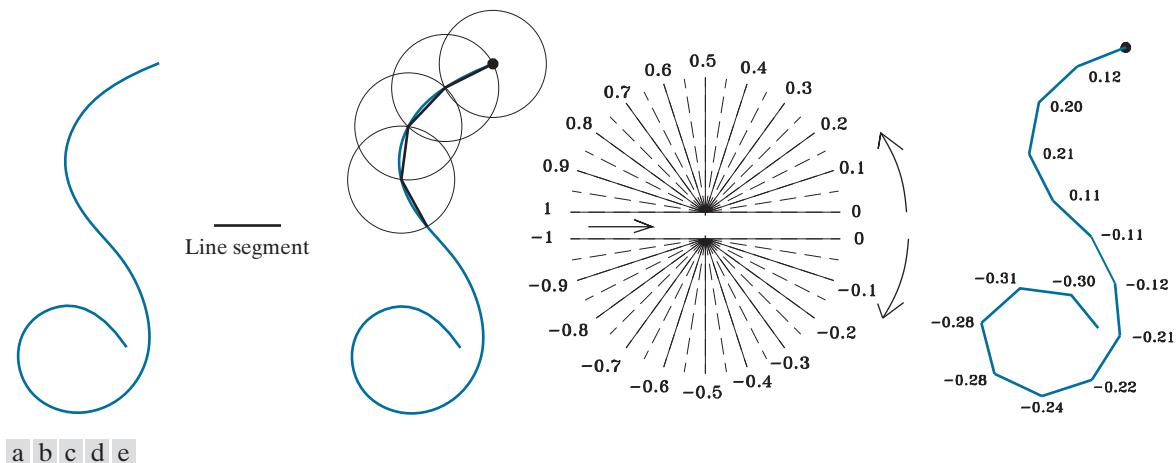


FIGURE 11.6 (a) An open curve. (b) A straight-line segment. (c) Traversing the curve using circumferences to determine slope changes; the dot is the origin (starting point). (d) Range of slope changes in the open interval $(-1, 1)$ (the arrow in the center of the chart indicates direction of travel). There can be ten subintervals between the slope numbers shown. (e) Resulting coded curve showing its corresponding numerical sequence of slope changes. (Courtesy of Professor Ernesto Bribiesca, IIMAS-UNAM, Mexico.)

length and/or increasing angle resolution often resolves the problem, because the results of computations are rounded to the nearest integer (remember we work with integer coordinates).

The *inverse* of an SCC is another chain of the same length, obtained by reversing the order of the symbols and their signs. The *mirror image* of a chain is obtained by starting at the origin and reversing the signs of the symbols. Finally, we point out that the preceding discussion is directly applicable to closed curves. Curve following would start at an arbitrary point (for example, the uppermost-leftmost point of the curve) and proceed in a clockwise or counterclockwise direction, stopping when the starting point is reached. We will illustrate an use of SSCs in Example 11.6.

BOUNDARY APPROXIMATIONS USING MINIMUM-PERIMETER POLYGONS

For an open curve, the number of segments of an exact polygonal approximation is equal to the number of points minus 1.

A digital boundary can be approximated with arbitrary accuracy by a polygon. For a closed curve, the approximation becomes exact when the number of segments of the polygon is equal to the number of points in the boundary, so each pair of adjacent points defines a segment of the polygon. The goal of a polygonal approximation is to capture the essence of the shape in a given boundary using the fewest possible number of segments. Generally, this problem is not trivial, and can turn into a time-consuming iterative search. However, approximation techniques of modest complexity are well suited for image-processing tasks. Among these, one of the most powerful is representing a boundary by a *minimum-perimeter polygon* (MPP), as defined in the following discussion.

Foundation

An intuitive approach for computing MPPs is to enclose a boundary [see Fig. 11.7(a)] by a set of concatenated cells, as in Fig. 11.7(b). Think of the boundary as a rubber band contained in the gray cells in Fig. 11.7(b). As it is allowed to shrink, the rubber band will be constrained by the vertices of the inner and outer walls of the region of the gray cells. Ultimately, this shrinking produces the shape of a polygon of minimum perimeter (with respect to this geometrical arrangement) that circumscribes the region enclosed by the cell strip, as in Fig. 11.7(c). Note in this figure that all the vertices of the MPP coincide with corners of either the inner or the outer wall.

The size of the cells determines the accuracy of the polygonal approximation. In the limit, if the size of each (square) cell corresponds to a pixel in the boundary, the maximum error in each cell between the boundary and the MPP approximation would be $\sqrt{2}d$, where d is the minimum possible distance between pixels (i.e., the distance between pixels established by the resolution of the original sampled boundary). This error can be reduced in half by forcing each cell in the polygonal approximation to be centered on its corresponding pixel in the original boundary. The objective is to use the largest possible cell size acceptable in a given application, thus producing MPPs with the fewest number of vertices. Our objective in this section is to formulate a procedure for finding these MPP vertices.

The cellular approach just described reduces the shape of the object enclosed by the original boundary, to the area circumscribed by the gray walls in Fig. 11.7(b).

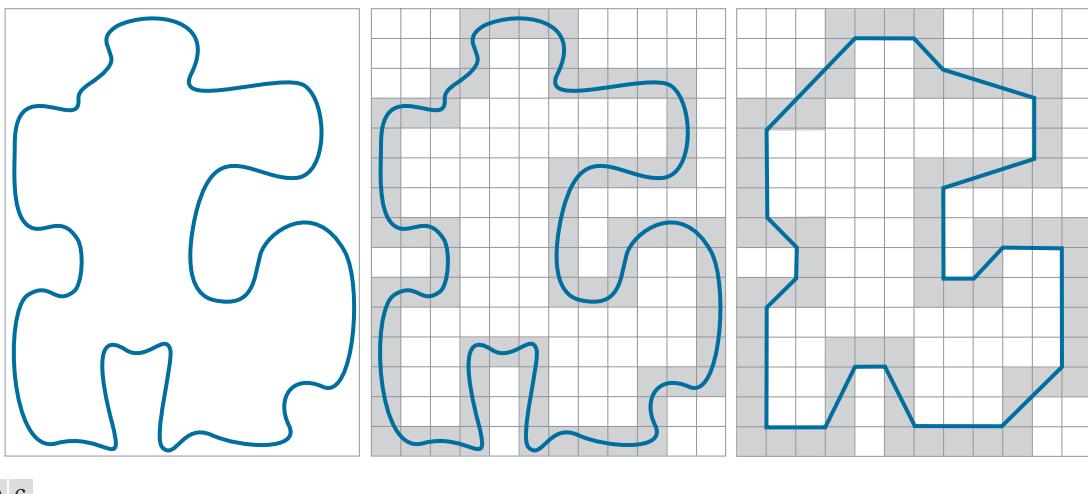


FIGURE 11.7 (a) An object boundary. (b) Boundary enclosed by cells (shaded). (c) Minimum-perimeter polygon obtained by allowing the boundary to shrink. The vertices of the polygon are created by the corners of the inner and outer walls of the gray region.

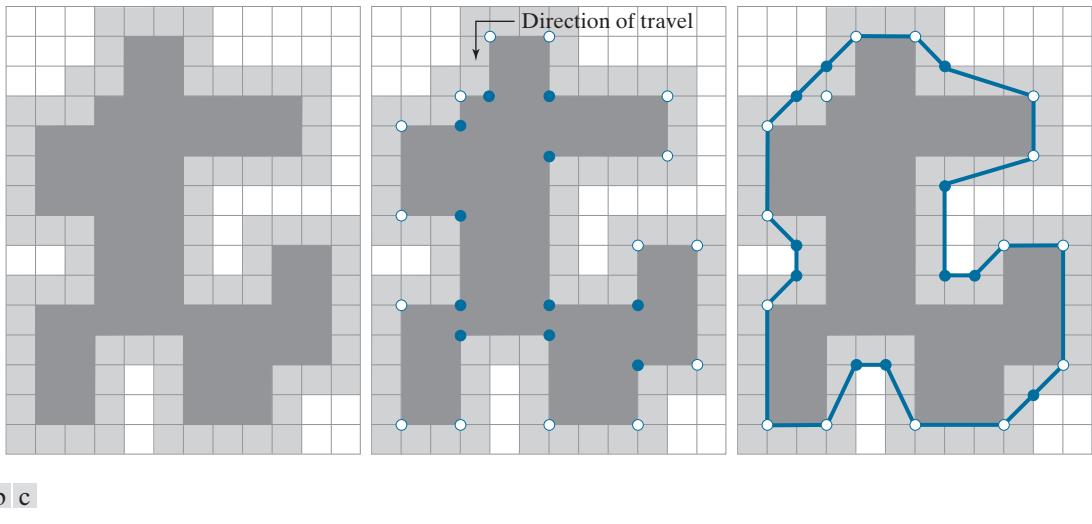
A convex vertex is the center point of a triplet of points that define an angle in the range $0^\circ < \theta < 180^\circ$. Similarly, angles of a concave vertex are in the range $180^\circ < \theta < 360^\circ$. An angle of 180° defines a degenerate vertex (i.e., segment of a straight line), which cannot be an MPP-vertex.

Figure 11.8(a) shows this shape in dark gray. Suppose that we traverse the boundary of the dark gray region in a *counterclockwise* direction. Every turn encountered in the traversal will be either a convex or a concave vertex (the angle of a vertex is defined as an *interior* angle of the boundary at that vertex). Convex and concave vertices are shown, respectively, as white and blue dots in Fig. 11.8(b). Note that these vertices are the vertices of the inner wall of the light-gray bounding region in Fig. 11.8(b), and that every concave (blue) vertex in the dark gray region has a corresponding concave “mirror” vertex in the light gray wall, located diagonally opposite the vertex. Figure 11.8(c) shows the mirrors of all the concave vertices, with the MPP from Fig. 11.7(c) superimposed for reference. We see that the vertices of the MPP coincide either with convex vertices in the inner wall (white dots) or with the mirrors of the concave vertices (blue dots) in the outer wall. Only convex vertices of the inner wall and concave vertices of the outer wall can be vertices of the MPP. Thus, our algorithm needs to focus attention only on those vertices.

MPP Algorithm

The set of cells enclosing a digital boundary [e.g., the gray cells in Fig. 11.7(b)] is called a *cellular complex*. We assume the cellular complexes to be *simply connected*, in the sense the boundaries they enclose are not self-intersecting. Based on this assumption, and letting *white* (*W*) denote convex vertices, and *blue* (*B*) denote mirrored concave vertices, we state the following observations:

1. The MPP bounded by a simply connected cellular complex is not self-intersecting.
2. Every *convex* vertex of the MPP is a *W* vertex, but not every *W* vertex of a boundary is a vertex of the MPP.



a b c

FIGURE 11.8 (a) Region (dark gray) resulting from enclosing the original boundary by cells (see Fig. 11.7). (b) Convex (white dots) and concave (blue dots) vertices obtained by following the boundary of the dark gray region in the counterclockwise direction. (c) Concave vertices (blue dots) displaced to their diagonal mirror locations in the outer wall of the bounding region; the convex vertices are not changed. The MPP (solid boundary) is superimposed for reference.

3. Every *mirrored concave* vertex of the MPP is a *B* vertex, but not every *B* vertex of a boundary is a vertex of the MPP.
4. All *B* vertices are on or outside the MPP, and all *W* vertices are on or inside the MPP.
5. The uppermost-leftmost vertex in a sequence of vertices contained in a cellular complex is always a *W* vertex of the MPP (see Problem 11.8).

These assertions can be proved formally (Sklansky et al. [1972], Sloboda et al. [1998], and Klette and Rosenfeld [2004]). However, their correctness is evident for our purposes (see Fig. 11.8), so we do not dwell on the proofs here. Unlike the angles of the vertices of the dark gray region in Fig. 11.8, the angles sustained by the vertices of the MPP are not necessarily multiples of 90°.

In the discussion that follows, we will need to calculate the orientation of triplets of points. Consider a triplet of points, (a, b, c) , and let the coordinates of these points be $a = (a_x, a_y)$, $b = (b_x, b_y)$, and $c = (c_x, c_y)$. If we arrange these points as the rows of the matrix

$$\mathbf{A} = \begin{bmatrix} a_x & a_y & 1 \\ b_x & b_y & 1 \\ c_x & c_y & 1 \end{bmatrix} \quad (11-1)$$

Then, it follows from matrix analysis that

$$\det(\mathbf{A}) = \begin{cases} > 0 & \text{if } (a, b, c) \text{ is a counterclockwise sequence} \\ 0 & \text{if the points are collinear} \\ < 0 & \text{if } (a, b, c) \text{ is a clockwise sequence} \end{cases} \quad (11-2)$$

where $\det(\mathbf{A})$ is the determinant of \mathbf{A} . In terms of this equation, movement in a counterclockwise or clockwise direction is with respect to a right-handed coordinate system (see the footnote in the discussion of Fig. 2.19). For example, using the image coordinate system from Fig. 2.19 (in which the origin is at the top left, the positive x -axis extends vertically downward, and the positive y -axis extends horizontally to the right), the sequence $a = (3, 4)$, $b = (2, 3)$, and $c = (3, 2)$ is in the counterclockwise direction. This would give $\det(\mathbf{A}) > 0$ when substituted into Eq. (11-2). It is convenient when describing the algorithm to define

$$\operatorname{sgn}(a, b, c) \equiv \det(\mathbf{A}) \quad (11-3)$$

so that $\operatorname{sgn}(a, b, c) > 0$ for a counterclockwise sequence, $\operatorname{sgn}(a, b, c) < 0$ for a clockwise sequence, and $\operatorname{sgn}(a, b, c) = 0$ when the points are collinear. Geometrically, $\operatorname{sgn}(a, b, c) > 0$ indicates that point c lies on the positive side of pair (a, b) (i.e., c lies on the positive side of the line passing through points a and b). Similarly, if $\operatorname{sgn}(a, b, c) < 0$, point c lies on the negative side of the line. Equations (11-2) and (11-3) give the same result if the sequence (c, a, b) or (b, c, a) is used because the direction of travel in the sequence is the same as for (a, b, c) . However, the geometrical interpretation is different. For example, $\operatorname{sgn}(c, a, b) > 0$ indicates that point b lies on the positive side of the line through points c and a .

To prepare the data for the MPP algorithm, we form a list of triplets consisting of a vertex label (e.g., V_0 , V_1 , etc.); the coordinates of each vertex; and an additional element denoting whether the vertex is W or B . It is important that the concave vertices be mirrored, as in Fig. 11.8(c), that the vertices be in sequential order,[†] and that the first vertex be the uppermost-leftmost vertex, which we know from property 5 is a W vertex of the MPP. Let V_0 denote this vertex. We assume that the vertices are arranged in the counterclockwise direction. The algorithm for finding MPPs uses two “crawler” points: a white crawler (W_C) and a blue crawler (B_C). W_C crawls along the convex (W) vertices, and B_C crawls along the concave (B) vertices. These two crawler points, the last MPP vertex found, and the vertex being examined are all that is necessary to implement the algorithm.

The algorithm starts by setting $W_C = B_C = V_0$ (recall that V_0 is an MPP-vertex). Then, at any step in the algorithm, let V_L denote the last MPP vertex found, and let V_k denote the current vertex being examined. One of the following three conditions can exist between V_L , V_k , and the two crawler points:

[†]Vertices of a boundary can be ordered by tracking the boundary using the boundary-following algorithm discussed earlier.

- (a) V_k is on the positive side of the line through the pair of points (V_L, W_C) , in which case $\text{sgn}(V_L, W_C, V_k) > 0$.
- (b) V_k is on the negative side of the line through pair (V_L, W_C) or is collinear with it; that is $\text{sgn}(V_L, W_C, V_k) \leq 0$. Simultaneously, V_k lies to the positive side of the line through (V_L, B_C) or is collinear with it; that is, $\text{sgn}(V_L, B_C, V_k) \geq 0$.
- (c) V_k is on the negative side of the line through pair (V_L, B_C) , in which case $\text{sgn}(V_L, B_C, V_k) < 0$.

If condition (a) holds, the next MPP vertex is W_C , and we let $V_L = W_C$; then we reinitialize the algorithm by setting $W_C = B_C = V_L$, and start with the next vertex after the newly changed V_L .

If condition (b) holds, V_k becomes a *candidate* MPP vertex. In this case, we set $W_C = V_k$ if V_k is convex (i.e., it is a W vertex); otherwise we set $B_C = V_k$. We then continue with the next vertex in the list.

If condition (c) holds, the next MPP vertex is B_C and we let $V_L = B_C$; then we reinitialize the algorithm by setting $W_C = B_C = V_L$ and start with the next vertex after the newly changed V_L .

The algorithm stops when it reaches the first vertex again, and thus has processed all the vertices in the polygon. The V_L vertices found by the algorithm are the vertices of the MPP. Klette and Rosenfeld [2004] have proved that this algorithm finds all the MPP vertices of a polygon enclosed by a simply connected cellular complex.

EXAMPLE 11.2: A numerical example showing the details of how the MPP algorithm works.

A simple example in which we can follow the algorithm step-by-step will help clarify the preceding concepts. Consider the vertices in Fig. 11.8(c). In our image coordinate system, the top-left point of the grid is at coordinates $(0,0)$. Assuming unit grid spacing, the first few (counterclockwise) vertices are:

$$V_0 (1,4) W | V_1 (2,3) B | V_2 (3,3) W | V_3 (3,2) B | V_4 (4,1) W | V_5 (7,1) W | V_6 (8,2) B | V_7 (9,2) B$$

where the triplets are separated by vertical lines, and the B vertices are mirrored, as required by the algorithm.

The uppermost-leftmost vertex is always the first vertex of the MPP, so we start by letting V_L and V_0 be equal, $V_L = V_0 = (1,4)$, and initializing the other variables: $W_C = B_C = V_L = (1,4)$.

The next vertex is $V_1 = (2,3)$. In this case we have $\text{sgn}(V_L, W_C, V_1) = 0$ and $\text{sgn}(V_L, B_C, V_1) = 0$, so condition (b) holds. Because V_1 is a B (concave) vertex, we update the blue crawler: $B_C = V_1 = (2,3)$. At this stage, we have $V_L = (1,4)$, $W_C = (1,4)$, and $B_C = (2,3)$.

Next, we look at $V_2 = (3,3)$. In this case, $\text{sgn}(V_L, W_C, V_2) = 0$, and $\text{sgn}(V_L, B_C, V_2) = 1$, so condition (b) holds. Because V_2 is W , we update the white crawler: $W_C = (3,3)$.

The next vertex is $V_3 = (3,2)$. At this junction we have $V_L = (1,4)$, $W_C = (3,3)$, and $B_C = (2,3)$. Then, $\text{sgn}(V_L, W_C, V_3) = -2$ and $\text{sgn}(V_L, B_C, V_3) = 0$, so condition (b) holds again. Because V_3 is B , we let $B_C = V_3 = (4,3)$ and look at the next vertex.

The next vertex is $V_4 = (4,1)$. We are working with $V_L = (1,4)$, $W_C = (3,3)$, and $B_C = (3,2)$. The values of sgn are $\text{sgn}(V_L, W_C, V_4) = -3$ and $\text{sgn}(V_L, B_C, V_4) = 0$. So, condition (b) holds yet again, and we let $W_C = V_4 = (4,1)$ because V_4 is a W vertex.

The next vertex is $V_5 = (7,1)$. Using the values from the previous step we obtain $\text{sgn}(V_L, W_C, V_5) = 9$, so condition (a) is satisfied. Therefore, we let $V_L = W_C = (4,1)$ (this is V_4) and reinitialize: $B_C = W_C = V_L = (4,1)$. Note that once we knew that $\text{sgn}(V_L, W_C, V_5) > 0$ we did not bother to compute the other sgn expression. Also, reinitialization means that we start fresh again by examining the next vertex following the newly found MPP vertex. In this case, that next vertex is V_5 , so we visit it again.

With $V_5 = (7,1)$, and using the new values of V_L , W_C , and B_C , it follows that $\text{sgn}(V_L, W_C, V_5) = 0$ and $\text{sgn}(V_L, B_C, V_5) = 0$, so condition (b) holds. Therefore, we let $W_C = V_5 = (7,1)$ because V_5 is a W vertex.

The next vertex is $V_6 = (8,2)$ and $\text{sgn}(V_L, W_C, V_6) = 3$, so condition (a) holds. Thus, we let $V_L = W_C = (7,1)$ and reinitialize the algorithm by setting $W_C = B_C = V_L$.

Because the algorithm was reinitialized at V_5 , the next vertex is $V_6 = (8,2)$ again. Using the results from the previous step gives us $\text{sgn}(V_L, W_C, V_6) = 0$ and $\text{sgn}(V_L, B_C, V_6) = 0$, so condition (b) holds this time. Because V_6 is B we let $B_C = V_6 = (8,2)$.

Summarizing, we have found three vertices of the MPP up to this point: $V_1 = (1,4)$, $V_4 = (4,1)$, and $V_5 = (7,1)$. Continuing as above with the remaining vertices results in the MPP vertices in Fig. 11.8(c) (see Problem 11.9). The mirrored B vertices at $(2,3), (3,2)$, and on the lower-right side at $(13,10)$, are on the boundary of the MPP. However, they are collinear and thus are not considered vertices of the MPP. Appropriately, the algorithm did not detect them as such.

EXAMPLE 11.3: Applying the MPP algorithm.

Figure 11.9(a) is a 566×566 binary image of a maple leaf, and Fig. 11.9(b) is its 8-connected boundary. The sequence in Figs. 11.9(c) through (h) shows MMP representations of this boundary using square cellular complex cells of sizes 2, 4, 6, 8, 16, and 32, respectively (the vertices in each figure were connected with straight lines to form a closed boundary). The leaf has two major features: a stem and three main lobes. The stem begins to be lost for cell sizes greater than 4×4 , as Fig. 11.9(e) shows. The three main lobes are preserved reasonably well, even for a cell size of 16×16 , as Fig. 11.9(g) shows. However, we see in Fig. 11.8(h) that by the time the cell size is increased to 32×32 , this distinctive feature has been nearly lost.

The number of points in the original boundary [Fig. 11.9(b)] is 1900. The numbers of vertices in Figs. 11.9(c) through (h) are 206, 127, 92, 66, 32, and 13, respectively. Figure 11.9(e), which has 127 vertices, retained all the major features of the original boundary while achieving a data reduction of over 90%. So here we see a significant advantage of MMPs for representing a boundary. Another important advantage is that MPPs perform boundary smoothing. As explained in the previous section, this is a usual requirement when representing a boundary by a chain code.

SIGNATURES

A signature is a 1-D functional representation of a 2-D boundary and may be generated in various ways. One of the simplest is to plot the distance from the centroid to the boundary as a function of angle, as illustrated in Fig. 11.10. The basic idea of using signatures is to reduce the boundary representation to a 1-D function that presumably is easier to describe than the original 2-D boundary.

Based on the assumptions of uniformity in scaling with respect to both axes, and that sampling is taken at equal intervals of θ , changes in the size of a shape result in changes in the amplitude values of the corresponding signature. One way to

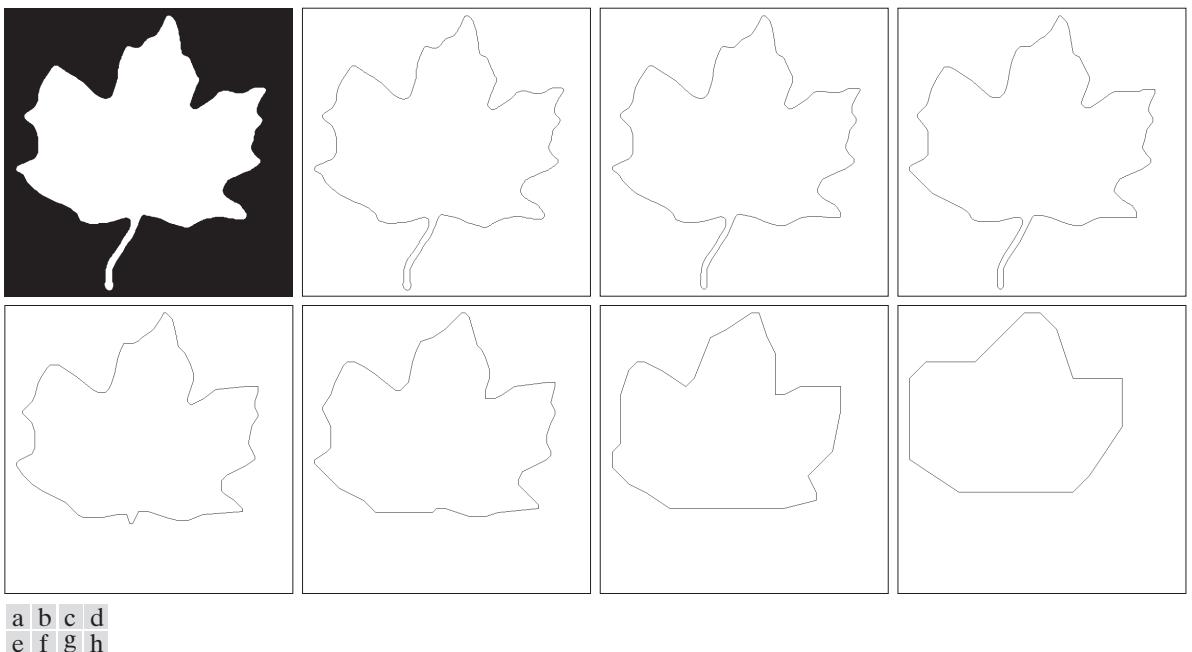


FIGURE 11.9 (a) 566×566 binary image. (b) 8-connected boundary. (c) through (h), MMPs obtained using square cells of sizes 2, 4, 6, 8, 16, and 32, respectively (the vertices were joined by straight-line segments for display). The number of boundary points in (b) is 1900. The numbers of vertices in (c) through (h) are 206, 127, 92, 66, 32, and 13, respectively. Images (b) through (h) are shown as negatives to make the boundaries easier to see.

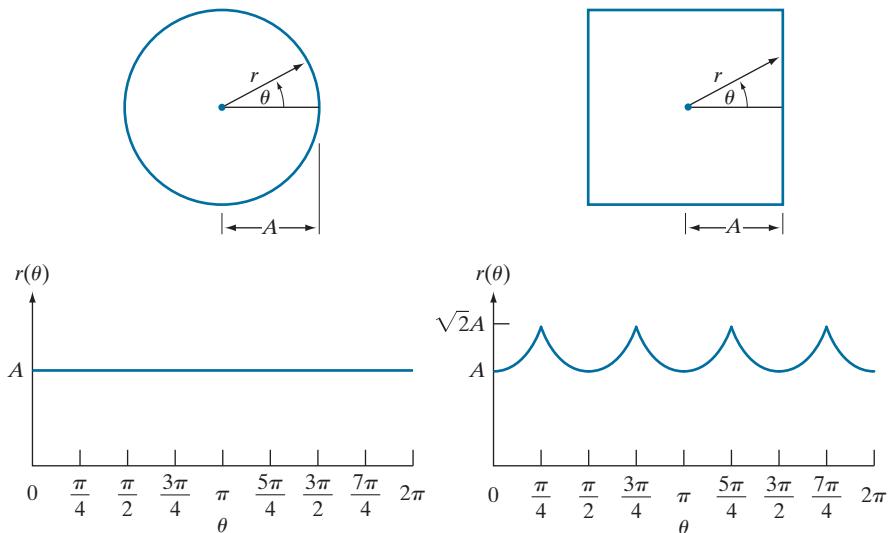
normalize for this is to scale all functions so that they always span the same range of values, e.g., $[0,1]$. The main advantage of this method is simplicity, but it has the disadvantage that scaling of the entire function depends on only two values: the minimum and maximum. If the shapes are noisy, this can be a source of significant error from object to object. A more rugged (but also more computationally intensive) approach is to divide each sample by the variance of the signature, assuming that the variance is not zero—as in the case of Fig. 11.10(a)—or so small that it creates computational difficulties. Using the variance yields a variable scaling factor that is inversely proportional to changes in size and works much as automatic volume control does. Whatever the method used, the central idea is to remove dependency on size while preserving the fundamental shape of the waveforms.

Distance versus angle is not the only way to generate a signature. For example, another way is to traverse the boundary and, corresponding to each point on the boundary, plot the angle between a line tangent to the boundary at that point and a reference line. The resulting signature, although quite different from the $r(\theta)$ curves in Fig. 11.10, carries information about basic shape characteristics. For instance, horizontal segments in the curve correspond to straight lines along the boundary because the tangent angle is constant there. A variation of this approach is to use the so-called *slope density function* as a signature. This function is a histogram of

a b

FIGURE 11.10

Distance-versus-angle signatures. In (a), $r(\theta)$ is constant. In (b), the signature consists of repetitions of the pattern $r(\theta) = A \sec \theta$ for $0 \leq \theta \leq \pi/4$, and $r(\theta) = A \csc \theta$ for $\pi/4 < \theta \leq \pi/2$.



tangent-angle values. Because a histogram is a measure of the concentration of values, the slope density function responds strongly to sections of the boundary with constant tangent angles (straight or nearly straight segments) and has deep valleys in sections producing rapidly varying angles (corners or other sharp inflections).

EXAMPLE 11.4: Signatures of two regions.

Figures 11.11(a) and (d) show two binary objects, and Figs. 11.11(b) and (e) are their boundaries. The corresponding $r(\theta)$ signatures in Figs. 11.11(c) and (f) range from 0° to 360° in increments of 1° . The number of prominent peaks in the signatures is sufficient to differentiate between the shapes of the two objects.

SKELETONS, MEDIAL AXES, AND DISTANCE TRANSFORMS

Like boundaries, skeletons are related to the shape of a region. Skeletons can be computed from a boundary by filling the area enclosed by the boundary with foreground values, and treating the result as a binary region. In other words, a skeleton is computed using the coordinates of points in the entire region, including its boundary. The idea is to reduce a region to a tree or graph by computing its skeleton. As we explained in Section 9.5 (see Fig. 9.25), the *skeleton* of a region is the set of points in the region that are equidistant from the border of the region.

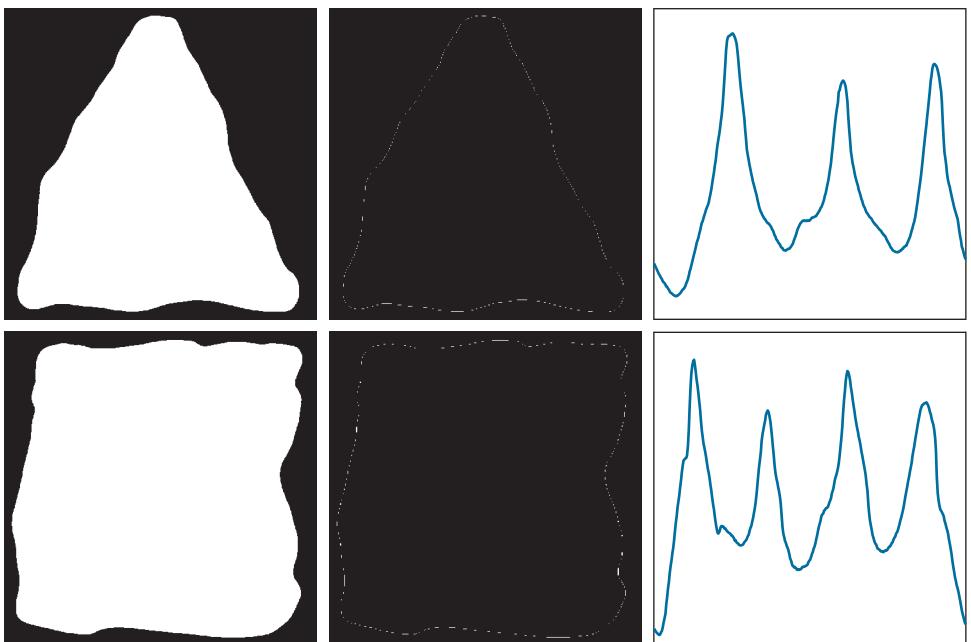
The skeleton is obtained using one of two principal approaches: (1) by successively thinning the region (e.g., using morphological erosion) while preserving end points and line connectivity (this is called *topology-preserving* thinning); or (2) by computing the *medial axis* of the region via an efficient implementation of the *medial axis transform* (MAT) proposed by Blum [1967]. We discussed thinning in Section 9.5. The MAT of a region R with border B is as follows: For each point p in R , we find its closest neighbor in B . If p has more than one such neighbor, it is said

As is true of thinning, the MAT is highly susceptible to boundary and internal region irregularities, so smoothing and other preprocessing steps generally are required to obtain a clean binary image.

a	b	c
d	e	f

FIGURE 11.11

(a) and (d) Two binary regions, (b) and (e) their external boundaries, and (c) and (f) their corresponding $r(\theta)$ signatures. The horizontal axes in (c) and (f) correspond to angles from 0° to 360° , in increments of 1° .



to belong to the medial axis of R . The concept of “closest” (and thus the resulting MAT) depends on the definition of a distance metric (see Section 2.5). Figure 11.12 shows some examples using the Euclidean distance. If the Euclidean distance is used, the resulting skeleton is the same as what would be obtained by using the maximum disks from Section 9.5. The skeleton of a region is *defined* as its medial axis.

The MAT of a region has an intuitive interpretation based on the “prairie fire” concept discussed in Section 11.3 (see Fig. 11.15). Consider an image region as a prairie of uniform, dry grass, and suppose that a fire is lit simultaneously along all the points on its border. All fire fronts will advance into the region at the same speed. The MAT of the region is the set of points reached by more than one fire front at the same time.

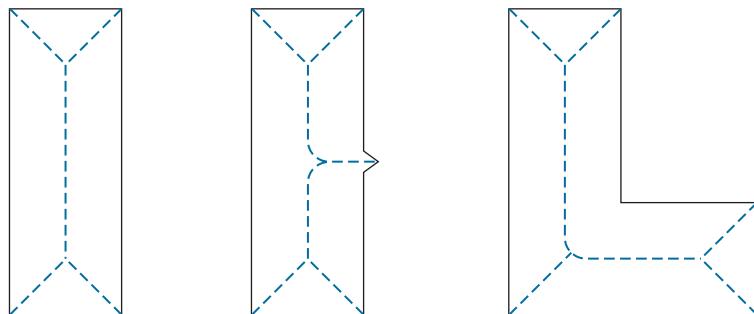
In general, the MAT comes considerably closer than thinning to producing skeletons that “make sense.” However, computing the MAT of a region requires calculating the distance from every interior point to every point on the border of the region—an impractical endeavor in most applications. Instead, the approach is to obtain the skeleton equivalently from the *distance transform*, for which numerous efficient algorithms exist.

The distance transform of a region of foreground pixels in a background of zeros is the distance from every pixel to the *nearest* nonzero valued pixel. Figure 11.13(a) shows a small binary image, and Fig. 11.13(b) is its distance transform. Observe that every 1-valued pixel has a distance transform value of 0 because its closest nonzero valued pixel is itself. For the purpose of finding skeletons equivalent to the MAT, we are interested in the distance from the pixels of a region of foreground (white)

a | b | c

FIGURE 11.12

Medial axes
(dashed) of three
simple regions.



pixels to their nearest background (zero) pixels, which constitute the region boundary. Thus, we compute the distance transform of the *complement* of the image, as Figs. 11.13(c) and (d) illustrate. By comparing Figs. 11.13(d) and 11.12(a), we see in the former that the MAT (skeleton) is equivalent to the *ridge* of the distance transform [i.e., the ridge in the image in Fig. 11.13(d)]. This ridge is the set of *local maxima* [shown bold in Fig. 11.13(d)]. Figures 11.13(e) and (f) show the same effect on a larger (414×708) binary image.

Finding approaches for computing the distance transform efficiently has been a topic of research for many years. Numerous approaches exist that can compute the distance transform with linear time complexity, $O(K)$, for a binary image with K pixels. For example, the algorithm by Maurer et al. [2003] not only can compute the distance transform in $O(K)$, it can compute it in $O(K/P)$ using P processors.

a | b
c | d
e | f

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

1.41	1	1	1	1.41
1	0	0	0	1
1	0	0	0	1
1.41	1	1	1	1.41

FIGURE 11.13

(a) A small image and (b) its distance transform. Note that all 1-valued pixels in (a) have corresponding 0's in (b). (c) A small image, and (d) the distance transform of its complement. (e) A larger image, and (f) the distance transform of its complement. The Euclidian distance was used throughout.

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

0	0	0	0	0	0	0	0	0
0	1	1	1	1	1	1	1	0
0	1	2	2	2	2	2	1	0
0	1	2	3	3	3	2	1	0
0	1	2	2	2	2	2	1	0
0	1	1	1	1	1	1	1	0
0	0	0	0	0	0	0	0	0



a | b
c | d

FIGURE 11.14

- (a) Threshholded image of blood vessels.
- (b) Skeleton obtained by thinning, shown superimposed on the image (note the spurs).
- (c) Result of 40 passes of spur removal.
- (d) Skeleton obtained using the distance transform.

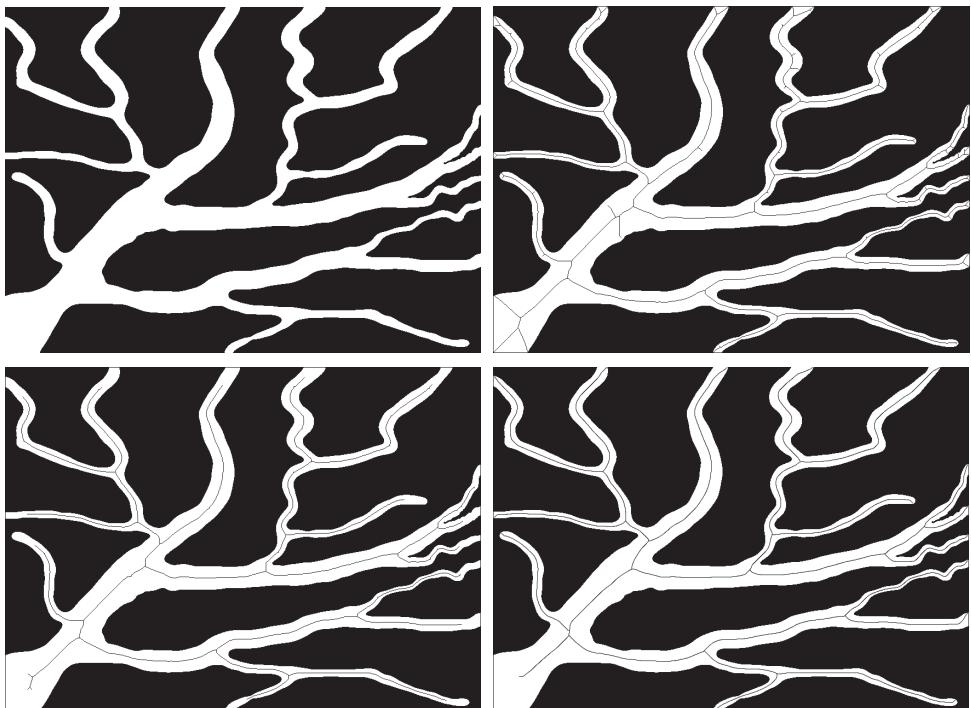
**EXAMPLE 11.5: Skeletons obtained using thinning and pruning vs. the distance transform.**

Figure 11.14(a) shows a segmented image of blood vessels, and Fig. 11.14(b) shows the skeleton obtained using morphological thinning. As we discussed in Chapter 9, thinning is characteristically accompanied by spurs, which certainly is the case here. Figure 11.14(c) shows the result of forty passes of spur removal. With the exception of the few small spurs visible on the bottom left of the image, pruning did a reasonable job of cleaning up the skeleton. One drawback of thinning is the loss of potentially important features. This was not the case here, except the pruned skeleton does not cover the full expanse of the image. Figure 11.14(d) shows the skeleton obtained using distance transform computations based on fast marching (see Lee et al. [2005] and Shi and Karl [2008]). The way the algorithm we used implements branch generation handles ambiguities such as spurs automatically.

The result in Fig. 11.14(d) is slightly superior to the result in Fig. 11.14(c), but both skeletons certainly capture the important features of the image in this case. A key advantage of the thinning approach is simplicity of implementation, which can be important in dedicated applications. Overall, distance-transform formulations tend to produce skeletons less prone to discontinuities, but overcoming the computational burden of the distance transform results in implementations that are considerably more complex than thinning.

11.3 BOUNDARY FEATURE DESCRIPTORS

We begin our discussion of feature descriptors by considering several fundamental approaches for describing region boundaries.

SOME BASIC BOUNDARY DESCRIPTORS

The *length* of a boundary is one of its simplest descriptors. The number of pixels along a boundary is an approximation of its length. For a chain-coded curve with unit spacing in both directions, the number of vertical and horizontal components plus $\sqrt{2}$ multiplied by the number of diagonal components gives its exact length. If the boundary is represented by a polygonal curve, the length is equal to the sum of the lengths of the polygonal segments.

The *diameter* of a boundary B is defined as

$$\text{diameter}(B) = \max_{i,j} [D(p_i, p_j)] \quad (11-4)$$

The major and minor axes are used also as regional descriptors.

where D is a distance measure (see Section 2.5) and p_i and p_j are points on the boundary. The value of the diameter and the orientation of a line segment connecting the two extreme points that comprise the diameter is called the *major axis* (or *longest chord*) of the boundary. That is, if the major axis is defined by points (x_1, y_1) and (x_2, y_2) , then the length and orientation of the major axis are given by

$$\text{length}_m = [(x_2 - x_1)^2 + (y_2 - y_1)^2]^{1/2} \quad (11-5)$$

and

$$\text{angle}_m = \tan^{-1} \left[\frac{y_2 - y_1}{x_2 - x_1} \right]$$

The *minor axis* (also called the *longest perpendicular chord*) of a boundary is defined as the line perpendicular to the major axis, and of such length that a box passing through the outer four points of intersection of the boundary with the two axes completely encloses the boundary. The box just described is called the *basic rectangle* or *bounding box*, and the ratio of the major to the minor axis is called the *eccentricity* of the boundary. We give some examples of this descriptor in Section 11.4.

The *curvature* of a boundary is defined as the rate of change of slope. In general, obtaining reliable measures of curvature at a point of a raw digital boundary is difficult because these boundaries tend to be locally “ragged.” Smoothing can help, but a more rugged measure of curvature is to use the difference between the slopes of adjacent boundary segments that have been represented as straight lines. Polygonal approximations are well-suited for this approach [see Fig. 11.8(c)], in which case we are concerned only with curvature at the vertices. As we traverse the polygon in the clockwise direction, a vertex point p is said to be *convex* if the change in slope at p is nonnegative; otherwise, p is said to be *concave*. The description can be refined further by using ranges for the changes of slope. For instance, p could be labeled as part of a nearly straight line segment if the absolute change of slope at that point is less than 10° , or it could be labeled as “corner-like” point if the absolute change is in the range $90^\circ, \pm 30^\circ$.

Descriptors based on changes of slope can be formulated easily by expressing a boundary in the form of a slope chain code (SSC), as discussed earlier (see Fig. 11.6). A particularly useful boundary descriptor that is easily implemented using SSCs is *tortuosity*, a measure of the twists and turns of a curve. The tortuosity, τ , of a curve

We will discuss corners in detail later in this chapter.

represented by an SCC is defined as the sum of the absolute values of the chain elements:

$$\tau = \sum_{i=1}^n |a_i| \quad (11-6)$$

where n is the number of elements in the SCC, and $|a_i|$ are the values (slope changes) of the elements in the code. The next example illustrates one use of this descriptor

EXAMPLE 11.6: Using slope chain codes to describe tortuosity.

An important measure of blood vessel morphology is its tortuosity. This metric can assist in the computer-aided diagnosis of Retinopathy of Prematurity (ROP), an eye disease that affects babies born prematurely (Bribiesca [2013]). ROP causes abnormal blood vessels to grow in the retina (see Section 2.1). This growth can cause the retina to detach from the back of the eye, potentially leading to blindness.

Figure 11.15(a) shows an image of the retina (called a *fundus image*) from a newborn baby. Ophthalmologists diagnose and make decisions about the initial treatment of ROP based on the appearance of retinal blood vessels. Dilatation and increased tortuosity of the retinal vessels are signs of highly probable ROP. Blood vessels denoted A, B, and C in Fig. 11.15 were selected to demonstrate the discriminative potential of SCCs for quantifying tortuosity (each vessel shown is a long, thin *region*, not a line segment).

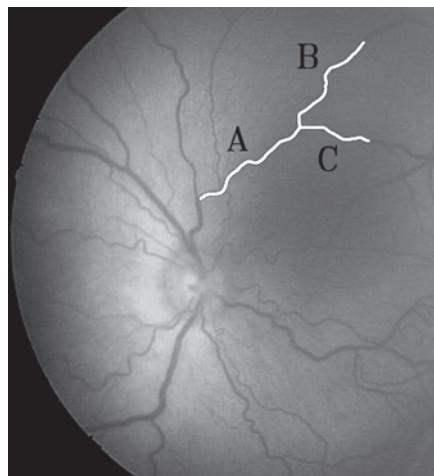
The border of each vessel was extracted and its length (number of pixels), P , was calculated. To make SCC comparisons meaningful, the three boundaries were normalized so that each would have the same number, m , of straight-line segments. The length, L , of the line segment was then computed as $L = m/P$. It follows that the number of elements of each SCC is $m - 1$. The tortuosity, τ , of a curve represented by an SCC is defined as the sum of the absolute values of the chain elements, as noted in Eq. (11-6).

The table in Fig. 11.15(b) shows values of τ for vessels A, B, and C based on 51 straight-line segments (as noted above, $n = m - 1$). The values of tortuosity are in agreement with our visual analysis of the three vessels, showing B as being slightly “busier” than A, and C as having the fewest twists and turns.

a b

FIGURE 11.15

(a) Fundus image from a prematurely born baby with ROP.
 (b) Tortuosity of vessels A, B, and C.
 (Courtesy of Professor Ernesto Bribiesca, IIMAS-UNAM, Mexico.)



Curve	n	τ
A	50	2.3770
B	50	2.5132
C	50	1.6285

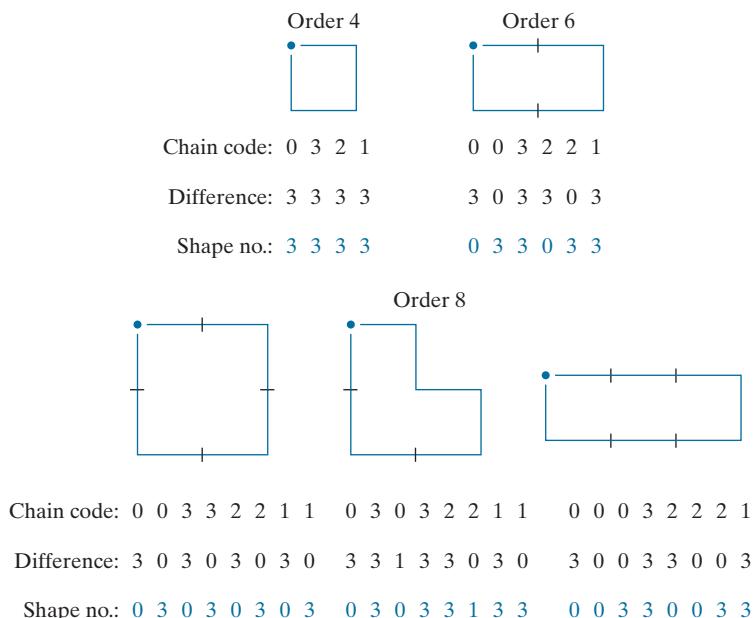
As explained in Section 11.2, the first difference of smallest magnitude makes a Freeman chain code independent of the starting point, and is insensitive to rotation in increments of 90° if a 4-directional code is used.

SHAPE NUMBERS

The shape number of a Freeman chain-coded boundary, based on the 4-directional code of Fig. 11.3(a), is defined as the first difference of smallest magnitude. The *order*, n , of a shape number is defined as the number of digits in its representation. Moreover, n is even for a closed boundary, and its value limits the number of possible different shapes. Figure 11.16 shows all the shapes of order 4, 6, and 8, along with their chain-code representations, first differences, and corresponding shape numbers. Although the first difference of a 4-directional chain code is independent of rotation (in increments of 90°), the coded boundary in general depends on the orientation of the grid. One way to normalize the grid orientation is by aligning the chain-code grid with the sides of the basic rectangle defined in the previous section.

In practice, for a desired shape order, we find the rectangle of order n whose eccentricity (defined in Section 11.4) best approximates that of the basic rectangle, and use this new rectangle to establish the grid size. For example, if $n = 12$, all the rectangles of order 12 (that is, those whose perimeter length is 12) are of sizes 2×4 , 3×3 , and 1×5 . If the eccentricity of the 2×4 rectangle best matches the eccentricity of the basic rectangle for a given boundary, we establish a 2×4 grid centered on the basic rectangle and use the procedure outlined in Section 11.2 to obtain the Freeman chain code. The shape number follows from the first difference of this code. Although the order of the resulting shape number usually equals n because of the way the grid spacing was selected, boundaries with depressions comparable to this spacing sometimes yield shape numbers of order greater than n . In this case, we specify a rectangle of order lower than n , and repeat the procedure until the resulting shape number is of order n . The order of a shape number starts at 4 and is always even because we are working with 4-connectivity and require that boundaries be closed.

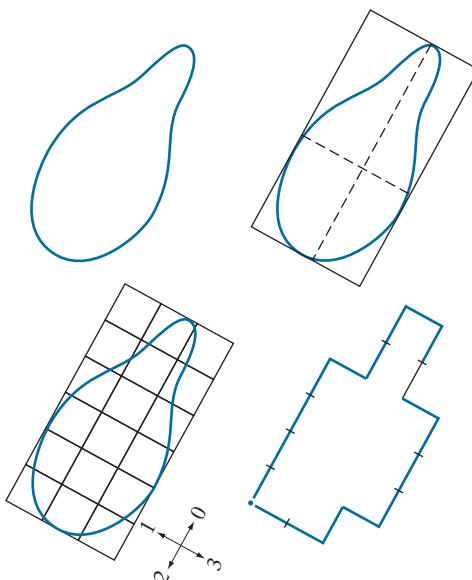
FIGURE 11.16
All shapes of order 4, 6, and 8. The directions are from Fig. 11.3(a), and the dot indicates the starting point.



a	b
c	d

FIGURE 11.17

Steps in the generation of a shape number.



Chain code: 0 0 0 0 3 0 0 3 2 2 3 2 2 2 1 2 1 1

Difference: 3 0 0 0 3 1 0 3 3 0 1 3 0 0 3 1 3 0

Shape no.: 0 0 0 3 1 0 3 3 0 1 3 0 0 3 1 3 0 3

EXAMPLE 11.7: Computing shape numbers.

Suppose that $n = 18$ is specified for the boundary in Fig. 11.17(a). To obtain a shape number of this order we follow the steps just discussed. First, we find the basic rectangle, as shown in Fig. 11.17(b). Next we find the closest rectangle of order 18. It is a 3×6 rectangle, requiring the subdivision of the basic rectangle shown in Fig. 11.17(c). The chain-code directions are aligned with the resulting grid. The final step is to obtain the chain code and use its first difference to compute the shape number, as shown in Fig. 11.17(d).

FOURIER DESCRIPTORS

We use the “conventional” axis system here for consistency with the literature. However, the same result is obtained if we use the book image coordinate system whose origin is at the top left because both are right-handed coordinate systems (see Fig. 2.19). In the latter, the rows and columns represent the real and imaginary parts of the complex number.

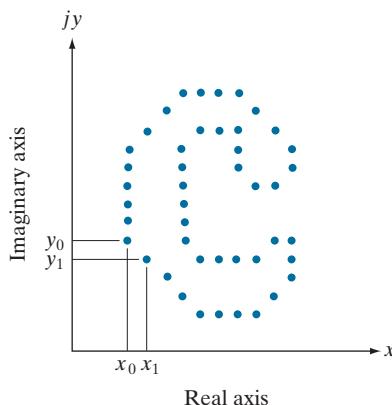
Figure 11.18 shows a digital boundary in the xy -plane, consisting of K points. Starting at an arbitrary point (x_0, y_0) , coordinate pairs $(x_0, y_0), (x_1, y_1), (x_2, y_2), \dots, (x_{K-1}, y_{K-1})$ are encountered in traversing the boundary, say, in the counterclockwise direction. These coordinates can be expressed in the form $x(k) = x_k$ and $y(k) = y_k$. Using this notation, the boundary itself can be represented as the sequence of coordinates $s(k) = [x(k), y(k)]$ for $k = 0, 1, 2, \dots, K - 1$. Moreover, each coordinate pair can be treated as a complex number so that

$$s(k) = x(k) + jy(k) \quad (11-7)$$

for $k = 0, 1, 2, \dots, K - 1$. That is, the x -axis is treated as the real axis and the y -axis as the imaginary axis of a sequence of complex numbers. Although the interpretation

FIGURE 11.18

A digital boundary and its representation as sequence of complex numbers. The points (x_0, y_0) and (x_1, y_1) are (arbitrarily) the first two points in the sequence.



of the sequence was restated, the nature of the boundary itself was not changed. Of course, this representation has one great advantage: It reduces a 2-D to a 1-D description problem.

We know from Eq. (4-44) that the discrete Fourier transform (DFT) of $s(k)$ is

$$a(u) = \sum_{k=0}^{K-1} s(k) e^{-j2\pi uk/K} \quad (11-8)$$

for $u = 0, 1, 2, \dots, K - 1$. The complex coefficients $a(u)$ are called the *Fourier descriptors* of the boundary. The inverse Fourier transform of these coefficients restores $s(k)$. That is, from Eq. (4-45),

$$s(k) = \frac{1}{K} \sum_{u=0}^{K-1} a(u) e^{j2\pi uk/K} \quad (11-9)$$

for $k = 0, 1, 2, \dots, K - 1$. We know from Chapter 4 that the inverse is identical to the original input, provided that all the Fourier coefficients are used in Eq. (11-9). However, suppose that, instead of all the Fourier coefficients, only the first P coefficients are used. This is equivalent to setting $a(u) = 0$ for $u > P - 1$ in Eq. (11-9). The result is the following *approximation* to $s(k)$:

$$\hat{s}(k) = \frac{1}{K} \sum_{u=0}^{P-1} a(u) e^{j2\pi uk/K} \quad (11-10)$$

for $k = 0, 1, 2, \dots, K - 1$. Although only P terms are used to obtain each component of $\hat{s}(k)$, parameter k still ranges from 0 to $K - 1$. That is, the *same* number of points exists in the approximate boundary, but not as many terms are used in the reconstruction of each point.

Deleting the high-frequency coefficients is the same as filtering the transform with an ideal lowpass filter. You learned in Chapter 4 that the periodicity of the DFT requires that we center the transform prior to filtering it by multiplying it by $(-1)^x$. Thus, we use this procedure when implementing Eq. (11-8), and use it again

to reverse the centering when computing the inverse in Eq. (11-10). Because of symmetry considerations in the DFT, the number of points in the boundary and its inverse must be even. This implies that the number of coefficients removed (set to 0) before the inverse is computed must be even. Because the transform is centered, we set to 0 half the number of coefficients on each end of the transform to preserve symmetry. Of course, the DFT and its inverse are computed using an FFT algorithm.

Recall from discussions of the Fourier transform in Chapter 4 that high-frequency components account for fine detail, and low-frequency components determine overall shape. Thus, the smaller we make P in Eq. (11-10), the more detail that will be lost on the boundary, as the following example illustrates.

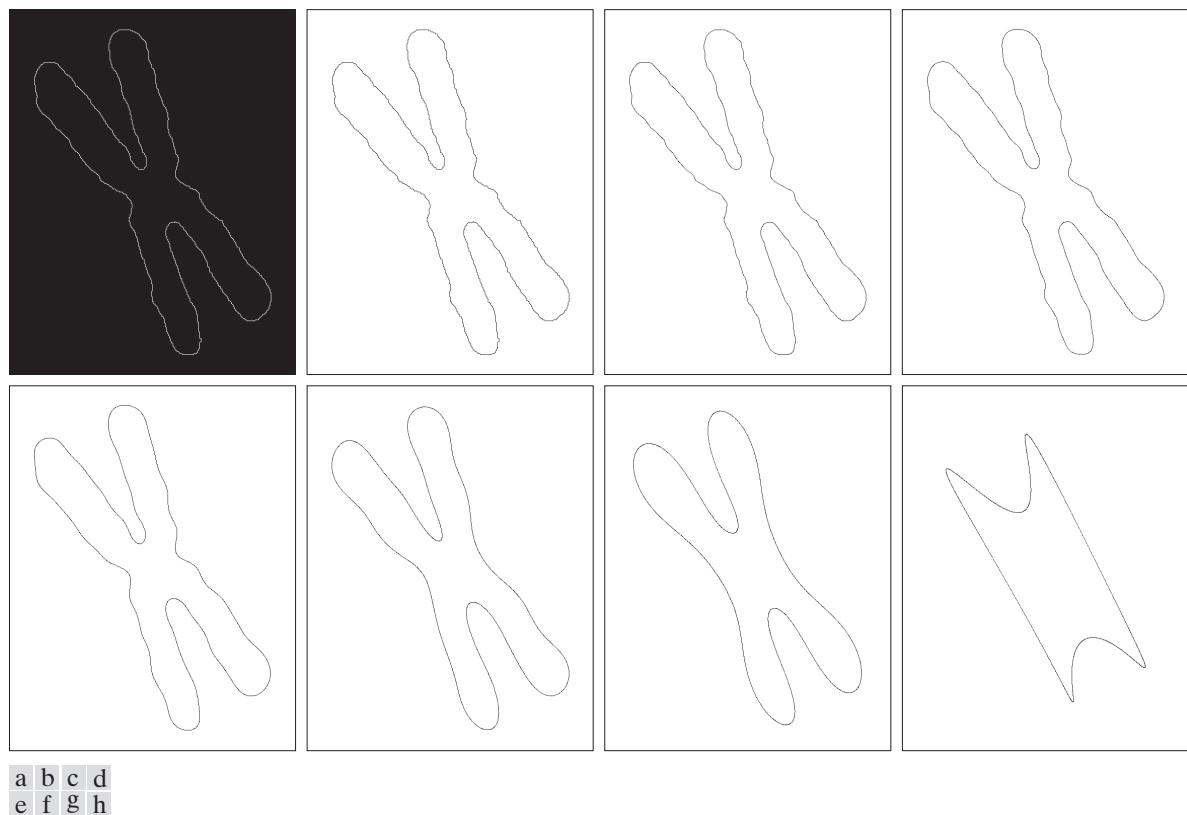
EXAMPLE 11.8: Using Fourier descriptors.

Figure 11.19(a) shows the boundary of a human chromosome, consisting of 2868 points. The corresponding 2868 Fourier descriptors were obtained using Eq. (11-8). The objective of this example is to examine the effects of reconstructing the boundary using fewer Fourier descriptors. Figure 11.19(b) shows the boundary reconstructed using one-half of the 2868 descriptors in Eq. (11-10). Observe that there is no perceptible difference between this boundary and the original. Figures 11.19(c) through (h) show the boundaries reconstructed with the number of Fourier descriptors being 10%, 5%, 2.5%, 1.25%, 0.63% and 0.28% of 2868, respectively. When rounded to the nearest even integer, these percentages are equal to 286, 144, 72, 36, 18, and 8 descriptors, respectively. The important point is that 18 descriptors, a mere six-tenths of one percent of the original 2868 descriptors, were sufficient to retain the principal shape features of the original boundary: four long protrusions and two deep bays. Figure 11.19(h), obtained with 8 descriptors, is unacceptable because the principal features are lost. Further reductions to 4 and 2 descriptors would result in an ellipse and a circle, respectively (see Problem 11.18).

As the preceding example demonstrates, a few Fourier descriptors can be used to capture the essence of a boundary. This property is valuable, because these coefficients carry shape information. Thus, forming a feature vector from these coefficients can be used to differentiate between boundary shapes, as we will discuss in Chapter 12.

We have stated several times that descriptors should be as insensitive as possible to translation, rotation, and scale changes. In cases where results depend on the order in which points are processed, an additional constraint is that descriptors should be insensitive to the starting point. Fourier descriptors are not directly insensitive to these geometrical changes, but changes in these parameters can be related to simple transformations on the descriptors. For example, consider rotation and recall from basic mathematical analysis that rotation of a point by an angle θ about the origin of the complex plane is accomplished by multiplying the point by $e^{j\theta}$. Doing so to every point of $s(k)$ rotates the entire sequence about the origin. The rotated sequence is $s(k)e^{j\theta}$, whose Fourier descriptors are

$$\begin{aligned} a_r(u) &= \sum_{k=0}^{K-1} s(k)e^{j\theta}e^{-j2\pi uk/K} \\ &= a(u)e^{j\theta} \end{aligned} \tag{11-11}$$



a b c d
e f g h

FIGURE 11.19 (a) Boundary of a human chromosome (2868 points). (b)–(h) Boundaries reconstructed using 1434, 286, 144, 72, 36, 18, and 8 Fourier descriptors, respectively. These numbers are approximately 50%, 10%, 5%, 2.5%, 1.25%, 0.63%, and 0.28% of 2868, respectively. Images (b)–(h) are shown as negatives to make the boundaries easier to see.

for $u = 0, 1, 2, \dots, K - 1$. Thus, rotation simply affects all coefficients equally by a multiplicative constant term $e^{i\theta}$.

Table 11.1 summarizes the Fourier descriptors for a boundary sequence $s(k)$ that undergoes rotation, translation, scaling, and changes in the starting point. The symbol Δ_{xy} is defined as $\Delta_{xy} = \Delta x + j\Delta y$, so the notation $s_t(k) = s(k) + \Delta_{xy}$ indicates redefining (translating) the sequence as

$$s_t(k) = [x(k) + \Delta x] + j[y(k) + \Delta y] \quad (11-12)$$

Recall from Chapter 4 that the Fourier transform of a constant is an impulse located at the origin. Recall also that an impulse $\delta(u)$ is zero everywhere, except when $u=0$.

Note that translation has no effect on the descriptors, except for $u = 0$, which has the value $\delta(0)$. Finally, the expression $s_p(k) = s(k - k_0)$ means redefining the sequence as

$$s_p(k) = x(k - k_0) + jy(k - k_0) \quad (11-13)$$

TABLE 11.1

Some basic properties of Fourier descriptors.

Transformation	Boundary	Fourier Descriptor
Identity	$s(k)$	$a(u)$
Rotation	$s_r(k) = s(k)e^{j\theta}$	$a_r(u) = a(u)e^{j\theta}$
Translation	$s_t(k) = s(k) + \Delta_{xy}$	$a_t(u) = a(u) + \Delta_{xy}\delta(u)$
Scaling	$s_s(k) = \alpha s(k)$	$a_s(u) = \alpha a(u)$
Starting point	$s_p(k) = s(k - k_0)$	$a_p(u) = a(u)e^{-j2\pi k_0 u/K}$

which changes the starting point of the sequence from $k = 0$ to $k = k_0$. The last entry in Table 11.1 shows that a change in starting point affects all descriptors in a different (but known) way, in the sense that the term multiplying $a(u)$ depends on u .

STATISTICAL MOMENTS

We will discuss moments of two variable in Section 11.4.

Statistical moments of one variable are useful descriptors applicable to 1-D renditions of 2-D boundaries, such as signatures. To see how this can be accomplished, consider Fig. 11.20 which shows the signature from Fig. 11.10(b) sampled, and treated as an ordinary discrete function $g(r)$ of one variable, r .

Suppose that we treat the *amplitude* of g as a discrete random variable z and form an amplitude histogram $p(z_i)$, $i = 0, 1, 2, \dots, A - 1$, where A is the number of discrete amplitude increments in which we divide the amplitude scale. If p is normalized so that the sum of its elements equals 1, then $p(z_i)$ is an estimate of the probability of intensity value z_i occurring. It then follows from Eq. (3-24) that the n th moment of z about its mean is

$$\mu_n(z) = \sum_{i=0}^{A-1} (z_i - m)^n p(z_i) \quad (11-14)$$

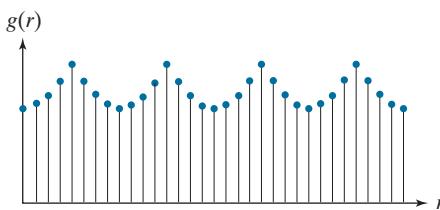
where

$$m = \sum_{i=0}^{A-1} z_i p(z_i) \quad (11-15)$$

As you know, m is the mean (average) value of z , and μ_2 is its variance. Generally, only the first few moments are required to differentiate between signatures of clearly distinct shapes.

FIGURE 11.20

Sampled signature from Fig. 11.10(b) treated as an ordinary, discrete function of one variable.



An alternative approach is to normalize the area of $g(r)$ in Fig. 11.20 to unity and treat it as a histogram. In other words, $g(r_i)$ is now treated as the probability of value r_i occurring. In this case, r is treated as the random variable and the moments are

$$\mu_n(r) = \sum_{i=0}^{K-1} (r_i - m)^n g(r_i) \quad (11-16)$$

where

$$m = \sum_{i=0}^{K-1} r_i g(r_i) \quad (11-17)$$

In these equations, K is the number of points on the boundary, and $\mu_n(r)$ is related directly to the shape of signature $g(r)$. For example, the second moment $\mu_2(r)$ measures the spread of the curve about the mean value of r , and the third moment $\mu_3(r)$ measures its symmetry with respect to the mean.

Although moments are used frequently for characterizing signatures, they are not the only descriptors used for this purpose. For instance, another approach is to compute the 1-D discrete Fourier transform of $g(r)$, obtain its spectrum, and use the first few components as descriptors. The advantage of moments over other techniques is that their implementation is straightforward and they also carry a “physical” interpretation of signature (and by implication boundary) shape. The insensitivity of this approach to rotation follows from the fact that signatures are independent of rotation, provided that the starting point is always the same along the boundary. Size normalization can be achieved by scaling the values of g and r .

11.4 REGION FEATURE DESCRIPTORS

As we did with boundaries, we begin the discussion of regional features with some basic region descriptors.

SOME BASIC DESCRIPTORS

The *major* and *minor* axes of a region, as well as the idea of a *bounding box*, are as defined earlier for boundaries. The *area* of a region is defined as the number of pixels in the region. The *perimeter* of a region is the length of its boundary. When area and perimeter are used as descriptors, they generally make sense only when they are normalized (Example 11.9 shows such a use). A more frequent use of these two descriptors is in measuring *compactness* of a region, defined as the perimeter squared over the area:

$$\text{compactness} = \frac{P^2}{A} \quad (11-18)$$

This is a dimensionless measure that is 4π for a circle (its minimum value) and 16 for a square.

A similar dimensionless measure is *circularity* (also called *roundness*), defined as

$$\text{circularity} = \frac{4\pi A}{P^2} \quad (11-19)$$

Sometimes compactness is defined as the inverse of the circularity. Obviously, these two measures are closely related.

The value of this descriptor is 1 for a circle (its maximum value) and $\pi/4$ for a square. Note that these two measures are independent of size, orientation, and translation. Another measure based on a circle is the *effective diameter*:

$$d_e = 2 \sqrt{\frac{A}{\pi}} \quad (11-20)$$

This is the diameter of a circle having the same area, A , as the region being processed. This measure is neither dimensionless nor independent of region size, but it is independent of orientation and translation. It can be normalized for size and made dimensionless by dividing it by the largest diameter expected in a given application.

In a manner analogous to the way we defined compactness and circularity relative to a circle, we define the *eccentricity* of a region relative to an ellipse as the eccentricity of an ellipse that has the same second central moments as the region. For 1-D, the second central moment is the variance. For 2-D discrete data, we have to consider the variance of each variable as well as the covariance between them. These are the components of the covariance matrix, which is estimated from samples using Eq. (11-21) below, with the samples in this case being 2-D vectors representing the coordinates of the data.

Figure 11.21(a) shows an ellipse in *standard form* (i.e., an ellipse whose major and minor axes are aligned with the coordinate axes). The eccentricity of such an ellipse is defined as the ratio of the distance between foci ($2c$ in Fig. 11.21), and the length of its major axis ($2a$), which gives the ratio $2c/2a = c/a$. That is,

$$\text{eccentricity} = \frac{c}{a} = \frac{\sqrt{a^2 - b^2}}{a} = \sqrt{1 - (b/a)^2} \quad a \geq b$$

However, we are interested in the eccentricity of an ellipse that has the same second central moments as a given 2-D region, which means that our ellipses can have arbitrary orientations. Intuitively, what we are trying to do is approximate our 2-D data by an elliptical region whose axes are aligned with the principal axes of the data, as Fig. 11.21(b) illustrates. As you will learn in Section 11.5 (see Example 11.17), the principal axes are the eigenvectors of the covariance matrix, \mathbf{C} , of the data, which is given by:

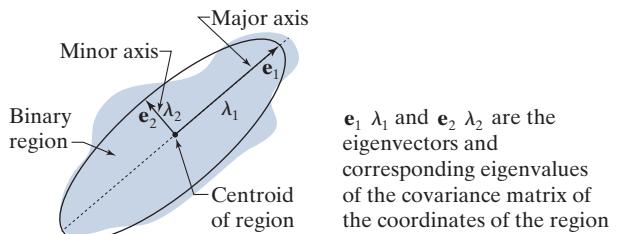
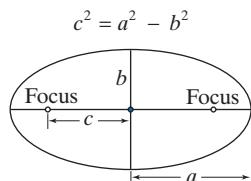
$$\mathbf{C} = \frac{1}{K-1} \sum_{k=1}^K (\mathbf{z}_k - \bar{\mathbf{z}})(\mathbf{z}_k - \bar{\mathbf{z}})^T \quad (11-21)$$

Often, you will see the constant in Eq. (11-21) written as $1/K$ instead of $1/(K-1)$. The latter is used to obtain a statistically-unbiased estimate of \mathbf{C} . For our purposes, either formulation is acceptable.

a b

FIGURE 11.21

- (a) An ellipse in standard form.
- (b) An ellipse approximating a region in arbitrary orientation.



where \mathbf{z}_k is a 2-D vector whose elements are the two spatial coordinates of a point in the region, K is the total number of points, and $\bar{\mathbf{z}}$ is the mean vector:

$$\bar{\mathbf{z}} = \frac{1}{K} \sum_{k=1}^K \mathbf{z}_k \quad (11-22)$$

The main diagonal elements of \mathbf{C} are the variances of the coordinate values of the points in the region, and the off-diagonal elements are their covariances.

An ellipse oriented in the same direction as the principal axes of the region can be interpreted as the intersection of a 2-D Gaussian function with the xy -plane. The orientation of the axes of the ellipse are also in the direction of the eigenvectors of the covariance matrix, and the distances from the center of the ellipse to its intersection with its major and minor axes is equal to the largest and smallest eigenvalues of the covariance matrix, respectively, as Fig. 11.21(b) shows. With reference to Fig. 11.21, and the equation of its eccentricity given above, we see by analogy that the eccentricity of an ellipse with the same second moments as the region is given by

$$\begin{aligned} \text{eccentricity} &= \frac{\sqrt{\lambda_2^2 - \lambda_1^2}}{\lambda_2} \\ &= \sqrt{1 - (\lambda_1/\lambda_2)^2} \quad \lambda_2 \geq \lambda_1 \end{aligned} \quad (11-23)$$

For circular regions, $\lambda_1 = \lambda_2$ and the eccentricity is 0. For a line, $\lambda_1 = 0$ and the eccentricity is 1. Thus, values of this descriptor are in the range [0,1].

EXAMPLE 11.9: Comparison of feature descriptors.

Figure 11.22 shows values of the preceding descriptors for several region shapes. None of the descriptors for the circle was exactly equal to its theoretical value because digitizing a circle introduces error into the computation, and because we approximated the length of a boundary as its number of elements. The eccentricity of the square did have an exact value of 0, because a square with no rotation aligns perfectly with the sampling grid. The other two descriptors for the square were close to their theoretical values also.

The values listed in the first two rows of Fig. 11.22 carry the same information. For example, we can tell that the star is less compact and less circular than the other shapes. Similarly, it is easy to tell from the numbers listed that the teardrop region has by far the largest eccentricity, but it is harder to differentiate from the other shapes using compactness or circularity.

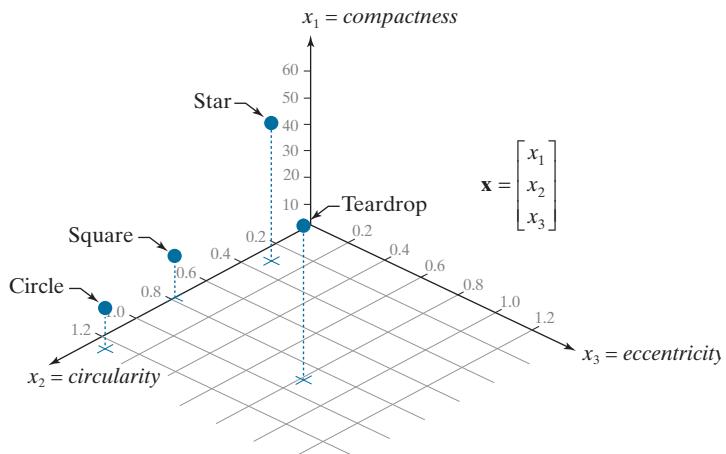
As we discussed in Section 11.1, feature descriptors typically are arranged in the form of feature vectors for subsequent processing. Figure 11.23 shows the feature space for the descriptors in Fig. 11.22.

a	b	c	d		Descriptor			
					<i>Compactness</i>	10.1701	42.2442	15.9836
					<i>Circularity</i>	1.2356	0.2975	0.7862
					<i>Eccentricity</i>	0.0411	0.0636	0
								0.8117

FIGURE 11.22
Compactness,
circularity,
and
eccentricity
of
some simple
binary regions.

FIGURE 11.23

The descriptors from Fig. 11.22 in 3-D feature space. Each dot shown corresponds to a feature vector whose components are the three corresponding descriptors in Fig. 11.22.



Each point in feature space “encapsulates” the three descriptor values for each object. Although we can tell from looking at the values of the descriptors in the figure that the circle and square are much more similar than the other two objects, note how much clearer this fact is in feature space. You can imagine that if we had multiple samples of those objects corrupted by noise, it could become difficult to differentiate between vectors (points) corresponding to squares or circles. In contrast, the star and teardrop objects are far from each other, and from the circle and square, so they are less likely to be misclassified in the presence of noise. Feature space will play an important role in Chapter 12, when we discuss image pattern classification.

EXAMPLE 11.10: Using area features.

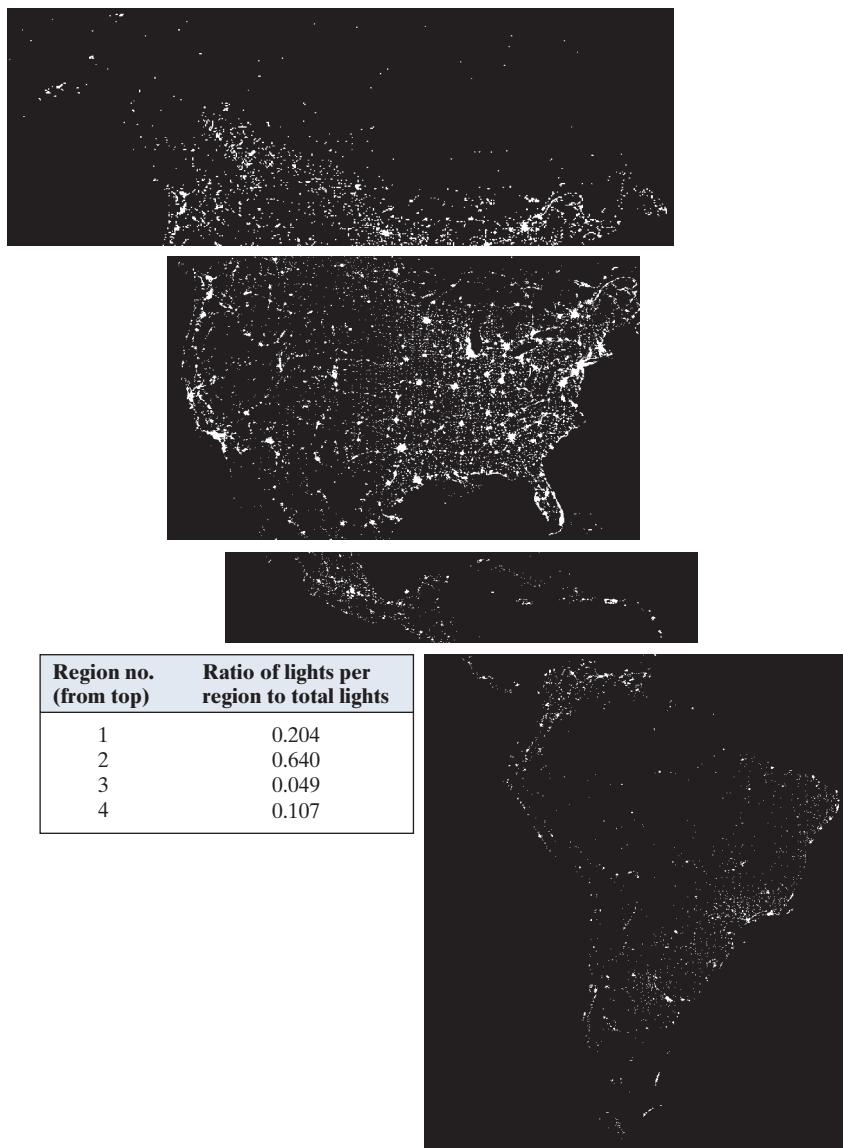
Even a simple descriptor such as normalized area can be quite useful for extracting information from images. For instance, Fig. 11.24 shows a night-time satellite infrared image of the Americas. As we discussed in Section 1.3, such images provide a global inventory of human settlements. The imaging sensors used to collect these images have the capability to detect visible and near infrared emissions, such as lights, fires, and flares. The table alongside the images shows (by region from top to bottom) the ratio of the area occupied by white (the lights) to the total light area in all four regions. A simple measurement like this can give, for example, a relative estimate by region of electrical energy consumption. The data can be refined by normalizing it with respect to land mass per region, with respect to population numbers, and so on.

TOPOLOGICAL DESCRIPTORS

Topology is the study of properties of a figure that are unaffected by any deformation, provided that there is no tearing or joining of the figure (sometimes these are called *rubber-sheet distortions*). For example, Fig. 11.25(a) shows a region with two holes. Obviously, a topological descriptor defined as the number of holes in the region will not be affected by a stretching or rotation transformation. However, the number of holes can change if the region is torn or folded. Because stretching

FIGURE 11.24

Infrared images of the Americas at night. (Courtesy of NOAA.)



affects distance, topological properties do not depend on the notion of distance or any properties implicitly based on the concept of a distance measure.

See Sections 2.5 and 9.5 regarding connected components.

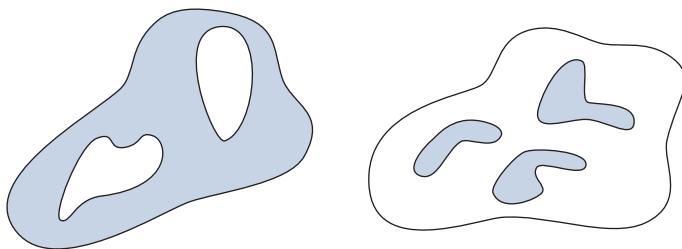
Another topological property useful for region description is the number of connected components of an image or region. Figure 11.25(b) shows a region with three connected components. The number of holes H and connected components C in a figure can be used to define the *Euler number*, E :

$$E = C - H \quad (11-24)$$

a b

FIGURE 11.25

- (a) A region with two holes.
 (b) A region with three connected components.



The Euler number is also a topological property. The regions shown in Fig. 11.26, for example, have Euler numbers equal to 0 and -1 , respectively, because the “A” has one connected component and one hole, and the “B” has one connected component but two holes.

Regions represented by straight-line segments (referred to as *polygonal networks*) have a particularly simple interpretation in terms of the Euler number. Figure 11.27 shows a polygonal network. Classifying interior regions of such a network into faces and holes is often important. Denoting the number of vertices by V , the number of edges by Q , and the number of faces by F gives the following relationship, called the *Euler formula*:

$$V - Q + F = C - H \quad (11-25)$$

which, in view of Eq. (11-24), can be expressed as

$$V - Q + F = E \quad (11-26)$$

The network in Fig. 11.27 has seven vertices, eleven edges, two faces, one connected region, and three holes; thus the Euler number is -2 (i.e., $7 - 11 + 2 = 1 - 3 = -2$).

EXAMPLE 11.11: Extracting and characterizing the largest feature in a segmented image.

Figure 11.28(a) shows a 512×512 , 8-bit image of Washington, D.C. taken by a NASA LANDSAT satellite. This image is in the near infrared band (see Fig. 1.10 for details). Suppose that we want to segment the river using only this image (as opposed to using several multispectral images, which would simplify the task, as you will see later in this chapter). Because the river is a dark, uniform region relative to the rest of the image, thresholding is an obvious approach to try. The result of thresholding the image with the highest possible threshold value before the river became a disconnected region is shown in Fig.

a b

FIGURE 11.26

- Regions with Euler numbers equal to 0 and -1 , respectively.

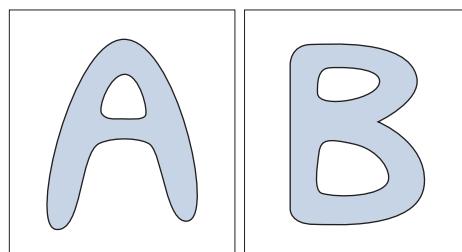
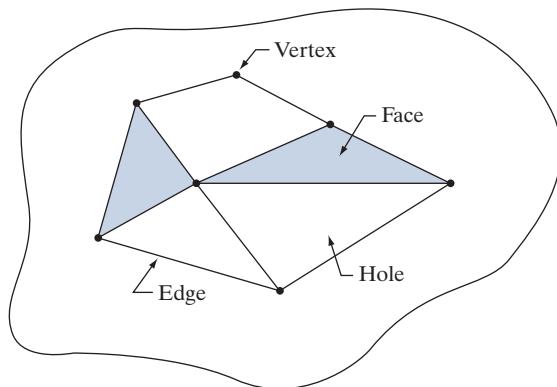


FIGURE 11.27

A region containing a polygonal network.



11.28(b). The threshold was selected manually to illustrate the point that it would be impossible in this case to segment the river by itself without other regions of the image also appearing in the thresholded result.

The image in Fig. 11.28(b) has 1591 connected components (obtained using 8-connectivity) and its Euler number is 1552, from which we deduce that the number of holes is 39. Figure 11.28(c) shows the connected component with the largest number of pixels (8479). This is the desired result, which we already know cannot be segmented by itself from the image using a threshold. Note how clean this result is. The number of holes in the region defined by the connected component just found would give us the number of land masses within the river. If we wanted to perform measurements, like the length of each branch of the river, we could use the skeleton of the connected component [Fig. 11.28(d)] to do so.

TEXTURE

An important approach to region description is to quantify its texture content. While no formal definition of texture exists, intuitively this descriptor provides measures of properties such as smoothness, coarseness, and regularity (Fig. 11.29 shows some examples). In this section, we discuss statistical and spectral approaches for describing the texture of a region. Statistical approaches yield characterizations of textures as smooth, coarse, grainy, and so on. Spectral techniques are based on properties of the Fourier spectrum and are used primarily to detect global periodicity in an image by identifying high-energy, narrow peaks in its spectrum.

Statistical Approaches

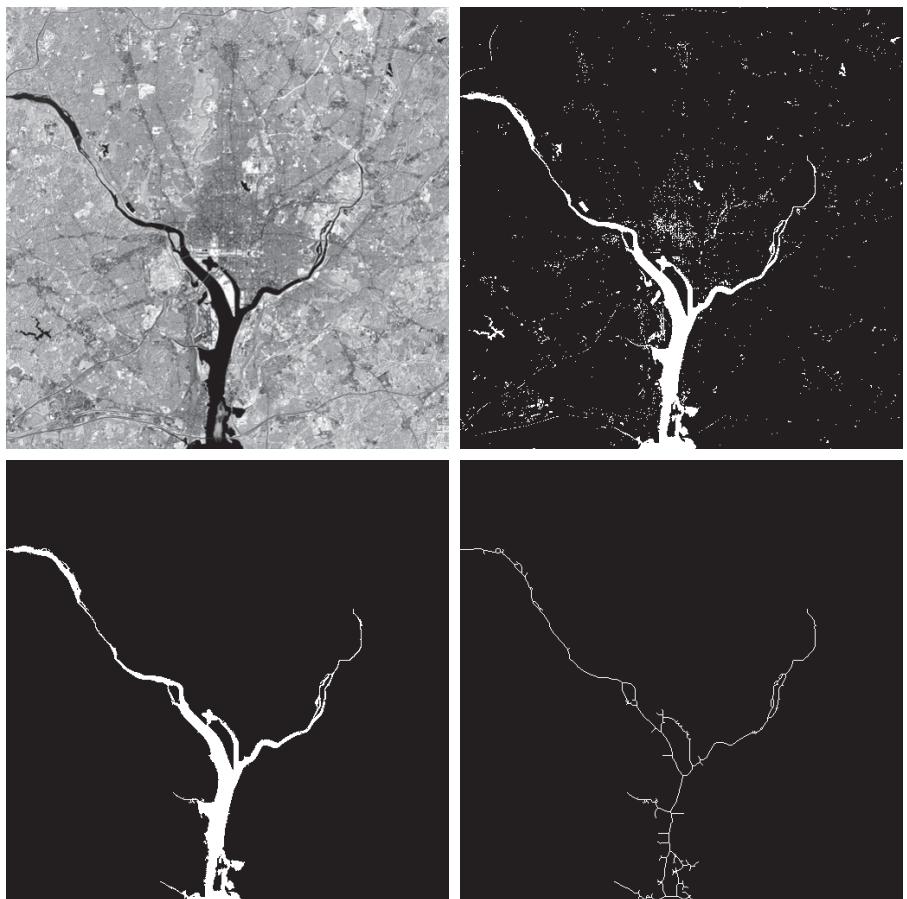
One of the simplest approaches for describing texture is to use statistical moments of the intensity histogram of an image or region. Let z be a random variable denoting intensity, and let $p(z_i)$, $i = 0, 1, 2, \dots, L - 1$, be the corresponding normalized histogram, where L is the number of distinct intensity levels. From Eq. (3-24), the n th moment of z about the mean is

$$\mu_n(z) = \sum_{i=0}^{L-1} (z_i - m)^n p(z_i) \quad (11-27)$$

a
b
c
d

FIGURE 11.28

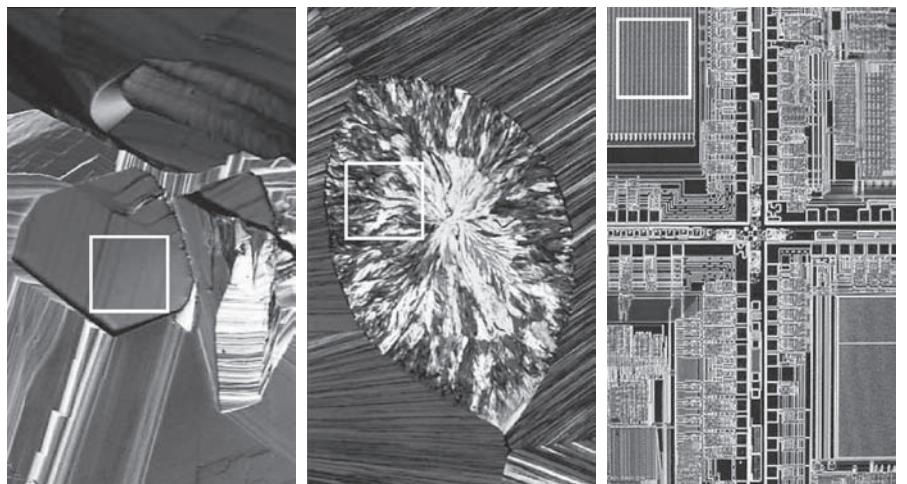
- (a) Infrared image of the Washington, D.C. area.
- (b) Thresholded image.
- (c) The largest connected component of (b).
- (d) Skeleton of (c). (Original image courtesy of NASA.)



a
b
c

FIGURE 11.29

- The white squares mark, from left to right, smooth, coarse, and regular textures. These are optical microscope images of a superconductor, human cholesterol, and a microprocessor. (Courtesy of Dr. Michael W. Davidson, Florida State University.)



where m is the mean value of z (i.e., the average intensity of the image or region):

$$m = \sum_{i=0}^{L-1} z_i p(z_i) \quad (11-28)$$

Note from Eq. (11-27) that $\mu_0 = 1$ and $\mu_1 = 0$. The second moment [the variance $\sigma^2(z) = \mu_2(z)$] is particularly important in texture description. It is a measure of intensity contrast that can be used to establish descriptors of relative intensity smoothness. For example, the measure

$$R(z) = 1 - \frac{1}{1 + \sigma^2(z)} \quad (11-29)$$

is 0 for areas of constant intensity (the variance is zero there) and approaches 1 for large values of $\sigma^2(z)$. Because variance values tend to be large for grayscale images with values, for example, in the range 0 to 255, it is a good idea to normalize the variance to the interval [0, 1] for use in Eq. (11-29). This is done simply by dividing $\sigma^2(z)$ by $(L-1)^2$ in Eq. (11-29). The standard deviation, $\sigma(z)$, also is used frequently as a measure of texture because its values are more intuitive.

For texture, typically we are interested in signs and relative magnitudes. If, in addition, normalization proves to be useful, we normalize the third and fourth moments.

As discussed in Section 2.6, the third moment, $\mu_3(z)$, is a measure of the skewness of the histogram while the fourth moment, $\mu_4(z)$, is a measure of its relative flatness. The fifth and higher moments are not so easily related to histogram shape, but they do provide further quantitative discrimination of texture content. Some useful additional texture measures based on histograms include a measure of *uniformity*, defined as

$$U(z) = \sum_{i=0}^{L-1} p^2(z_i) \quad (11-30)$$

and a measure of *average entropy* that, as you may recall from information theory, is defined as

$$e(z) = -\sum_{i=0}^{L-1} p(z_i) \log_2 p(z_i) \quad (11-31)$$

Because values of p are in the range [0, 1] and their sum equals 1, the value of descriptor U is maximum for an image in which all intensity levels are equal (maximally uniform), and decreases from there. Entropy is a measure of variability, and is 0 for a constant image.

EXAMPLE 11.12: Texture descriptors based on histograms.

Table 11.2 lists the values of the preceding descriptors for the three types of textures highlighted in Fig. 11.29. The mean describes only the average intensity of each region and is useful only as a rough idea of intensity, not texture. The standard deviation is more informative; the numbers clearly show that the first texture has significantly less variability in intensity (it is smoother) than the other two textures. The coarse texture shows up clearly in this measure. As expected, the same comments hold for R , because it measures essentially the same thing as the standard deviation. The third moment is useful for

TABLE 11.2

Statistical texture measures for the subimages in Fig. 11.29.

Texture	Mean	Standard deviation	R (normalized)	3rd moment	Uniformity	Entropy
Smooth	82.64	11.79	0.002	-0.105	0.026	5.434
Coarse	143.56	74.63	0.079	-0.151	0.005	7.783
Regular	99.72	33.73	0.017	0.750	0.013	6.674

determining the symmetry of histograms and whether they are skewed to the left (negative value) or the right (positive value). This gives an indication of whether the intensity levels are biased toward the dark or light side of the mean. In terms of texture, the information derived from the third moment is useful only when variations between measurements are large. Looking at the measure of uniformity, we again conclude that the first subimage is smoother (more uniform than the rest) and that the most random (lowest uniformity) corresponds to the coarse texture. Finally, we see that the entropy values increase as uniformity decreases, leading us to the same conclusions regarding the texture of the regions as the uniformity measure did. The first subimage has the lowest variation in intensity levels, and the coarse image the most. The regular texture is in between the two extremes with respect to both of these measures.

Measures of texture computed using only histograms carry no information regarding spatial relationships between pixels, which is important when describing texture. One way to incorporate this type of information into the texture-analysis process is to consider not only the distribution of intensities, but also the *relative positions* of pixels in an image.

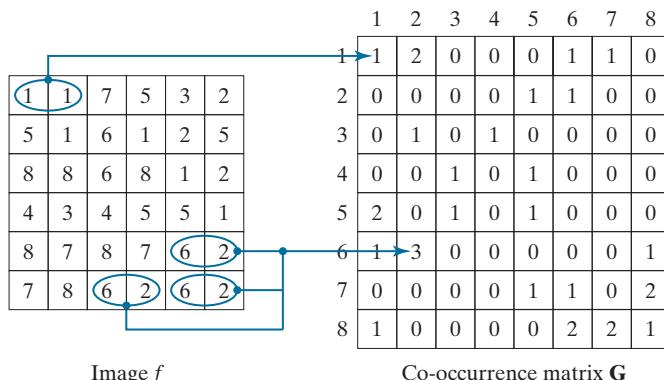
Let Q be an operator that defines the position of two pixels relative to each other, and consider an image, f , with L possible intensity levels. Let \mathbf{G} be a matrix whose element g_{ij} is the number of times that pixel pairs with intensities z_i and z_j occur in image f in the position specified by Q , where $1 \leq i, j \leq L$. A matrix formed in this manner is referred to as a *graylevel* (or *intensity*) *co-occurrence matrix*. When the meaning is clear, \mathbf{G} is referred to simply as a *co-occurrence matrix*.

Figure 11.30 shows an example of how to construct a co-occurrence matrix using $L = 8$ and a position operator Q defined as “one pixel immediately to the right” (i.e., the neighbor of a pixel is defined as the pixel immediately to its right). The array on the left is a small image and the array on the right is matrix \mathbf{G} . We see that element (1,1) of \mathbf{G} is 1, because there is only one occurrence in f of a pixel valued 1 having a pixel valued 1 immediately to its right. Similarly, element (6,2) of \mathbf{G} is 3, because there are three occurrences in f of a pixel with a value of 6 having a pixel valued 2 immediately to its right. The other elements of \mathbf{G} are similarly computed. If we had defined Q as, say, “one pixel to the right and one pixel above,” then position (1,1) in \mathbf{G} would have been 0 because there are no instances in f of a 1 with another 1 in the position specified by Q . On the other hand, positions (1,3), (1,5), and (1,7) in \mathbf{G} would all be 1’s, because intensity value 1 occurs in f with neighbors valued 3, 5, and 7 in the position specified by Q —one occurrence of each. As an exercise, you should compute all the elements of \mathbf{G} using this definition of Q .

Note that we are using the intensity range $[1, L]$ instead of the usual $[0, L-1]$. We do this so that intensity values will correspond with “traditional” matrix indexing (i.e., intensity value 1 corresponds to the first row and column indices of \mathbf{G}).

FIGURE 11.30

How to construct a co-occurrence matrix.



The number of possible intensity levels in the image determines the size of matrix \mathbf{G} . For an 8-bit image (256 possible intensity levels), \mathbf{G} will be of size 256×256 . This is not a problem when working with one matrix but, as you will see in as Example 11.13, co-occurrence matrices sometimes are used in sequences. One approach for reducing computations is to quantize the intensities into a few bands in order to keep the size of \mathbf{G} manageable. For example, in the case of 256 intensities, we can do this by letting the first 32 intensity levels equal to 1, the next 32 equal to 2, and so on. This will result in a co-occurrence matrix of size 8×8 .

The total number, n , of pixel pairs that satisfy Q is equal to the sum of the elements of \mathbf{G} ($n = 30$ in the example of Fig. 11.30). Then, the quantity

$$p_{ij} = \frac{g_{ij}}{n}$$

is an estimate of the probability that a pair of points satisfying Q will have values (z_i, z_j) . These probabilities are in the range $[0, 1]$ and their sum is 1:

$$\sum_{i=1}^K \sum_{j=1}^K p_{ij} = 1$$

where K is the row and column dimension of square matrix \mathbf{G} .

Because \mathbf{G} depends on Q , the presence of intensity texture patterns can be detected by choosing an appropriate position operator and analyzing the elements of \mathbf{G} . A set of descriptors useful for characterizing the contents of \mathbf{G} are listed in Table 11.3. The quantities used in the correlation descriptor (second row) are defined as follows:

$$m_r = \sum_{i=1}^K i \sum_{j=1}^K p_{ij}$$

$$m_c = \sum_{j=1}^K j \sum_{i=1}^K p_{ij}$$

and

TABLE 11.3

Descriptors used for characterizing co-occurrence matrices of size $K \times K$. The term p_{ij} is the ij -th term of \mathbf{G} divided by the sum of the elements of \mathbf{G} .

Descriptor	Explanation	Formula
Maximum probability	Measures the strongest response of \mathbf{G} . The range of values is $[0, 1]$.	$\max_{i,j}(p_{ij})$
Correlation	A measure of how correlated a pixel is to its neighbor over the entire image. The range of values is 1 to -1 corresponding to perfect positive and perfect negative correlations. This measure is not defined if either standard deviation is zero.	$\sum_{i=1}^K \sum_{j=1}^K \frac{(i - m_r)(j - m_c) p_{ij}}{\sigma_r \sigma_c}$ $\sigma_r \neq 0; \sigma_c \neq 0$
Contrast	A measure of intensity contrast between a pixel and its neighbor over the entire image. The range of values is 0 (when \mathbf{G} is constant) to $(K-1)^2$.	$\sum_{i=1}^K \sum_{j=1}^K (i - j)^2 p_{ij}$
Uniformity (also called Energy)	A measure of uniformity in the range $[0, 1]$. Uniformity is 1 for a constant image.	$\sum_{i=1}^K \sum_{j=1}^K p_{ij}^2$
Homogeneity	Measures the spatial closeness to the diagonal of the distribution of elements in \mathbf{G} . The range of values is $[0, 1]$, with the maximum being achieved when \mathbf{G} is a diagonal matrix.	$\sum_{i=1}^K \sum_{j=1}^K \frac{p_{ij}}{1 + i - j }$
Entropy	Measures the randomness of the elements of \mathbf{G} . The entropy is 0 when all p_{ij} 's are 0, and is maximum when the p_{ij} 's are uniformly distributed. The maximum value is thus $2 \log_2 K$.	$-\sum_{i=1}^K \sum_{j=1}^K p_{ij} \log_2 p_{ij}$

$$\sigma_r^2 = \sum_{i=1}^K (i - m_r)^2 \sum_{j=1}^K p_{ij}$$

$$\sigma_c^2 = \sum_{j=1}^K (j - m_c)^2 \sum_{i=1}^K p_{ij}$$

If we let

$$P(i) = \sum_{j=1}^K p_{ij}$$

and

$$P(j) = \sum_{i=1}^K p_{ij}$$

then the preceding equations can be written as

$$m_r = \sum_{i=1}^K i P(i)$$

$$m_c = \sum_{j=1}^K j P(j)$$

$$\sigma_r^2 = \sum_{i=1}^K (i - m_r)^2 P(i)$$

and

$$\sigma_c^2 = \sum_{j=1}^K (j - m_c)^2 P(j)$$

With reference to Eqs. (11-27), (11-28), and to their explanation, we see that m_r is in the form of a mean computed along rows of the normalized \mathbf{G} , and m_c is a mean computed along the columns. Similarly, σ_r and σ_c are in the form of standard deviations (square roots of the variances) computed along rows and columns, respectively. Each of these terms is a scalar, independently of the size of \mathbf{G} .

Keep in mind when studying Table 11.3 that “neighbors” are with respect to the way in which Q is defined (i.e., neighbors do not necessarily have to be adjacent), and also that the p_{ij} ’s are nothing more than normalized counts of the number of times that pixels having intensities z_i and z_j occur in f relative to the position specified in Q . Thus, all we are doing here is trying to find patterns (texture) in those counts.

EXAMPLE 11.13: Using descriptors to characterize co-occurrence matrices.

Figures 11.31(a) through (c) show images consisting of random, horizontally periodic (sine), and mixed pixel patterns, respectively. This example has two objectives: (1) to show values of the descriptors in Table 11.3 for the three co-occurrence matrices, \mathbf{G}_1 , \mathbf{G}_2 , and \mathbf{G}_3 , corresponding (from top to bottom) to these images; and (2) to illustrate how sequences of co-occurrence matrices can be used to detect texture patterns in an image.

Figure 11.32 shows co-occurrence matrices \mathbf{G}_1 , \mathbf{G}_2 , and \mathbf{G}_3 , displayed as images. These matrices were obtained using $L = 256$ and the position operator “one pixel immediately to the right.” The value at coordinates (i, j) in these images is the number of times that pixel pairs with intensities z_i and z_j occur in f in the position specified by Q , so it is not surprising that Fig. 11.32(a) is a random image, given the nature of the image from which it was obtained.

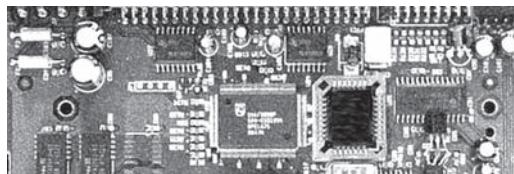
Figure 11.32(b) is more interesting. The first obvious feature is the symmetry about the main diagonal. Because of the symmetry of the sine wave, the number of counts for a pair (z_i, z_j) is the same as for the pair (z_j, z_i) , which produces a symmetric co-occurrence matrix. The nonzero elements of \mathbf{G}_2 are sparse because value differences between horizontally adjacent pixels in a horizontal sine wave are relatively small. It helps to remember in interpreting these concepts that a digitized sine wave is a staircase, with the height and width of each step depending on the frequency of the sine wave and the number of amplitude levels used in representing the function.

The structure of co-occurrence matrix \mathbf{G}_3 in Fig. 11.32(c) is more complex. High count values are grouped along the main diagonal also, but their distribution is more dense than for \mathbf{G}_2 , a property that is indicative of an image with a rich variation in intensity values, but few large jumps in intensity between adjacent pixels. Examining Fig. 11.32(c), we see that there are large areas characterized by low

a
b
c

FIGURE 11.31

Images whose pixels have
(a) random,
(b) periodic, and
(c) mixed texture patterns. Each image is of size 263×800 pixels.



variability in intensities. The high transitions in intensity occur at object boundaries, but these counts are low with respect to the moderate intensity transitions over large areas, so they are obscured by the ability of an image display to show high and low values simultaneously, as we discussed in Chapter 3.

The preceding observations are qualitative. To quantify the “content” of co-occurrence matrices, we need descriptors such as those in Table 11.3. Table 11.4 shows values of these descriptors computed for the three co-occurrence matrices in Fig. 11.32. To use these descriptors, the co-occurrence matrices must be normalized by dividing them by the sum of their elements, as discussed earlier. The entries in Table 11.4 agree with what one would expect from the images in Fig. 11.31 and their corresponding co-occurrence matrices in Fig. 11.32. For example, consider the Maximum Probability column in Table 11.4. The highest probability corresponds to the third co-occurrence matrix, which tells us that this matrix has the highest number of counts (largest number of pixel pairs occurring in the image relative to the positions in Q) than the other two matrices. This agrees with our analysis of \mathbf{G}_3 . The second column indicates that the highest correlation corresponds to \mathbf{G}_2 , which in turn tells us that the intensities in the second image are highly correlated. The repetitiveness of the sinusoidal pattern in Fig. 11.31(b) indicates why this is so. Note that the correlation for \mathbf{G}_1 is essentially zero, indicating that there is virtually no correlation between adjacent pixels, a characteristic of random images such as the image in Fig. 11.31(a).

a b c

FIGURE 11.32

256×256 co-occurrence matrices \mathbf{G}_1 , \mathbf{G}_2 , and \mathbf{G}_3 , corresponding from left to right to the images in Fig. 11.31.

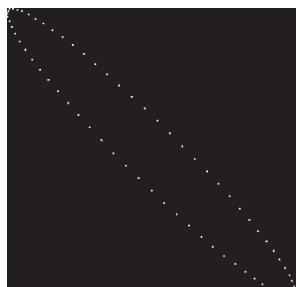
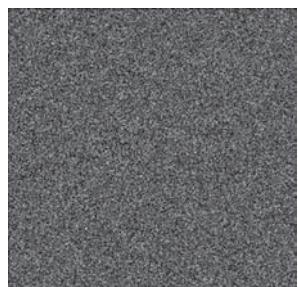


TABLE 11.4

Descriptors evaluated using the co-occurrence matrices displayed as images in Fig. 11.32.

Normalized Co-occurrence Matrix	Maximum Probability	Correlation	Contrast	Uniformity	Homogeneity	Entropy
\mathbf{G}_1/n_1	0.00006	-0.0005	10838	0.00002	0.0366	15.75
\mathbf{G}_2/n_2	0.01500	0.9650	00570	0.01230	0.0824	06.43
\mathbf{G}_3/n_3	0.06860	0.8798	01356	0.00480	0.2048	13.58

The contrast descriptor is highest for \mathbf{G}_1 and lowest for \mathbf{G}_2 . Thus, we see that the less random an image is, the lower its contrast tends to be. We can see the reason by studying the matrix displayed in Fig. 11.32. The $(i - j)^2$ terms are differences of integers for $1 \leq i, j \leq L$, so they are the same for any \mathbf{G} . Therefore, the probabilities of the elements of the normalized co-occurrence matrices are the factors that determine the value of contrast. Although \mathbf{G}_1 has the lowest maximum probability, the other two matrices have many more zero or near-zero probabilities (the dark areas in Fig. 11.32). Because the sum of the values of \mathbf{G}/n is 1, it is easy to see why the contrast descriptor tends to increase as a function of randomness.

The remaining three descriptors are explained in a similar manner. Uniformity increases as a function of the values of the probabilities squared. Thus, the less randomness there is in an image, the higher the uniformity descriptor will be, as the fifth column in Table 11.4 shows. Homogeneity measures the concentration of values of \mathbf{G} with respect to the main diagonal. The values of the denominator term $(1 + |i - j|)$ are the same for all three co-occurrence matrices, and they decrease as i and j become closer in value (i.e., closer to the main diagonal). Thus, the matrix with the highest values of probabilities (numerator terms) near the main diagonal will have the highest value of homogeneity. As we discussed earlier, such a matrix will correspond to images with a “rich” gray-level content and areas of slowly varying intensity values. The entries in the sixth column of Table 11.4 are consistent with this interpretation.

The entries in the last column of the table are measures of randomness in co-occurrence matrices, which in turn translate into measures of randomness in the corresponding images. As expected, \mathbf{G}_1 had the highest value because the image from which it was derived was totally random. The other two entries are self-explanatory. Note that the entropy measure for \mathbf{G}_1 is near the theoretical maximum of 16 ($2 \log_2 256 = 16$). The image in Fig. 11.31(a) is composed of uniform noise, so each intensity level has approximately an equal probability of occurrence, which is the condition stated in Table 11.3 for maximum entropy.

Thus far, we have dealt with single images and their co-occurrence matrices. Suppose that we want to “discover” (without looking at the images) if there are any sections in these images that contain repetitive components (i.e., periodic textures). One way to accomplish this goal is to examine the correlation descriptor for sequences of co-occurrence matrices, derived from these images by increasing the distance between neighbors. As mentioned earlier, it is customary when working with sequences of co-occurrence matrices to quantize the number of intensities in order to reduce matrix size and corresponding computational load. The following results were obtained using $L = 8$.

Figure 11.33 shows plots of the correlation descriptors as a function of horizontal “offset” (i.e., horizontal distance between neighbors) from 1 (for adjacent pixels) to 50. Figure 11.33(a) shows that all correlation values are near 0, indicating that no such patterns were found in the random image. The

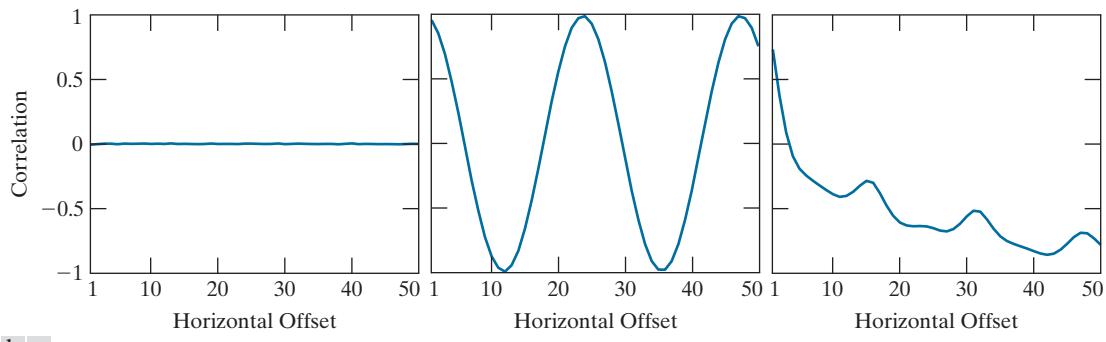


FIGURE 11.33 Values of the correlation descriptor as a function of offset (distance between “adjacent” pixels) corresponding to the (a) noisy, (b) sinusoidal, and (c) circuit board images in Fig. 11.31.

shape of the correlation in Fig. 11.33(b) is a clear indication that the input image is sinusoidal in the horizontal direction. Note that the correlation function starts at a high value, then decreases as the distance between neighbors increases, and then repeats itself.

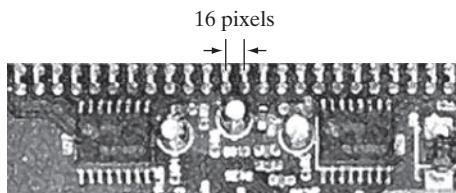
Figure 11.33(c) shows that the correlation descriptor associated with the circuit board image decreases initially, but has a strong peak for an offset distance of 16 pixels. Analysis of the image in Fig. 11.31(c) shows that the upper solder joints form a repetitive pattern approximately 16 pixels apart (see Fig. 11.34). The next major peak is at 32, caused by the same pattern, but the amplitude of the peak is lower because the number of repetitions at this distance is less than at 16 pixels. A similar observation explains the even smaller peak at an offset of 48 pixels.

Spectral Approaches

As we discussed in Section 5.4, the Fourier spectrum is ideally suited for describing the directionality of periodic or semiperiodic 2-D patterns in an image. These global texture patterns are easily distinguishable as concentrations of high-energy bursts in the spectrum. Here, we consider three features of the Fourier spectrum that are useful for texture description: (1) prominent peaks in the spectrum give the principal direction of the texture patterns; (2) the location of the peaks in the frequency plane gives the fundamental spatial period of the patterns; and (3) eliminating any periodic components via filtering leaves nonperiodic image elements, which can then be described by statistical techniques. Recall that the spectrum is symmetric about the origin, so only half of the frequency plane needs to be considered. Thus, for the

FIGURE 11.34

A zoomed section of the circuit board image showing periodicity of components.



purpose of analysis, every periodic pattern is associated with only one peak in the spectrum, rather than two.

Detection and interpretation of the spectrum features just mentioned often are simplified by expressing the spectrum in polar coordinates to yield a function $S(r, \theta)$, where S is the spectrum function, and r and θ are the variables in this coordinate system. For each direction θ , $S(r, \theta)$ may be considered a 1-D function $S_\theta(r)$. Similarly, for each frequency r , $S_r(\theta)$ is a 1-D function. Analyzing $S_\theta(r)$ for a fixed value of θ yields the behavior of the spectrum (e.g., the presence of peaks) along a radial direction from the origin, whereas analyzing $S_r(\theta)$ for a fixed value of r yields the behavior along a circle centered on the origin.

A more global description is obtained by integrating (summing for discrete variables) these functions:

$$S(r) = \sum_{\theta=0}^{\pi} S_\theta(r) \quad (11-32)$$

and

$$S(\theta) = \sum_{r=1}^{R_0} S_r(\theta) \quad (11-33)$$

where R_0 is the radius of a circle centered at the origin.

The results of Eqs. (11-32) and (11-33) constitute a pair of values $[S(r), S(\theta)]$ for each pair of coordinates (r, θ) . By varying these coordinates, we can generate two 1-D functions, $S(r)$ and $S(\theta)$, that constitute a spectral-energy description of texture for an entire image or region under consideration. Furthermore, descriptors of these functions themselves can be computed in order to characterize their behavior quantitatively. Descriptors useful for this purpose are the location of the highest value, the mean and variance of both the amplitude and axial variations, and the distance between the mean and the highest value of the function.

EXAMPLE 11.14: Spectral texture.

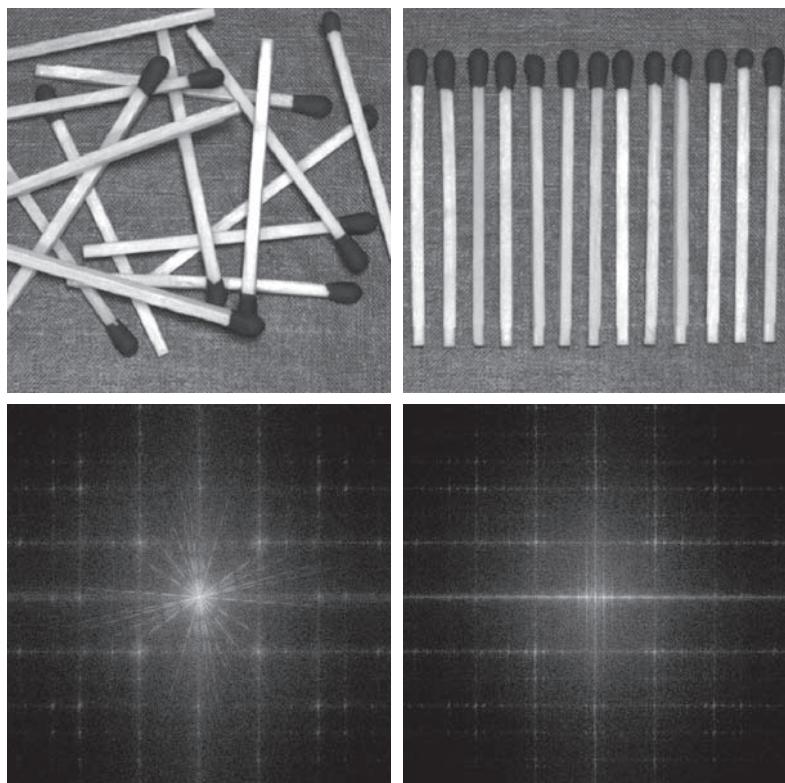
Figure 11.35(a) shows an image containing randomly distributed objects, and Fig. 11.35(b) shows an image in which these objects are arranged periodically. Figures 11.35(c) and (d) show the corresponding Fourier spectra. The periodic bursts of energy extending quadrilaterally in two dimensions in both Fourier spectra are due to the periodic texture of the coarse background material on which the objects rest. The other dominant components in the spectra in Fig. 11.35(c) are caused by the random orientation of the object edges in Fig. 11.35(a). On the other hand, the main energy in Fig. 11.35(d) not associated with the background is along the horizontal axis, corresponding to the strong vertical edges in Fig. 11.35(b).

Figures 11.36(a) and (b) are plots of $S(r)$ and $S(\theta)$ for the random objects, and similarly in (c) and (d) for the ordered objects. The plot of $S(r)$ for the random objects shows no strong periodic components (i.e., there are no dominant peaks in the spectrum besides the peak at the origin, which is the dc component). Conversely, the plot of $S(r)$ for the ordered objects shows a strong peak near $r = 15$ and a smaller one near $r = 25$, corresponding to the periodic horizontal repetition of the light (objects) and dark (background) regions in Fig. 11.35(b). Similarly, the random nature of the energy bursts in Fig. 11.35(c) is quite apparent in the plot of $S(\theta)$ in Fig. 11.36(b). By contrast, the plot in Fig. 11.36(d) shows strong energy components in the region near the origin and at 90° and 180° . This is consistent with the energy distribution of the spectrum in Fig. 11.35(d).

a b
c d

FIGURE 11.35

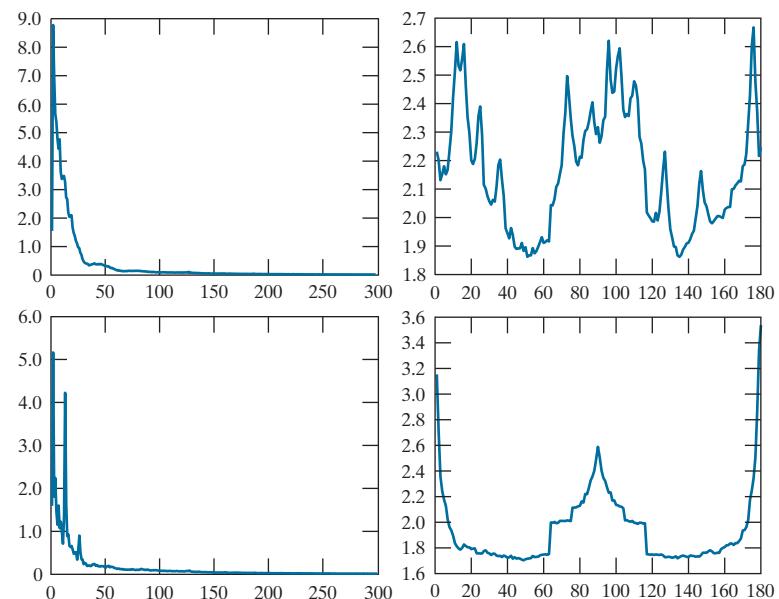
(a) and (b) Images of random and ordered objects.
(c) and (d) Corresponding Fourier spectra. All images are of size 600×600 pixels.



a b
c d

FIGURE 11.36

(a) and (b) Plots of $S(r)$ and $S(\theta)$ for Fig. 11.35(a).
(c) and (d) Plots of $S(r)$ and $S(\theta)$ for Fig. 11.35(b).
All vertical axes are $\times 10^5$.



MOMENT INVARIANTS

The 2-D *moment* of order $(p + q)$ of an $M \times N$ digital image, $f(x, y)$, is defined as

$$m_{pq} = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} x^p y^q f(x, y) \quad (11-34)$$

where $p = 0, 1, 2, \dots$ and $q = 0, 1, 2, \dots$ are integers. The corresponding *central moment* of order $(p + q)$ is defined as

$$\mu_{pq} = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} (x - \bar{x})^p (y - \bar{y})^q f(x, y) \quad (11-35)$$

for $p = 0, 1, 2, \dots$ and $q = 0, 1, 2, \dots$, where

$$\bar{x} = \frac{m_{10}}{m_{00}} \text{ and } \bar{y} = \frac{m_{01}}{m_{00}} \quad (11-36)$$

The *normalized central moment* of order $(p + q)$, denoted η_{pq} , is defined as

$$\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^\gamma} \quad (11-37)$$

where

$$\gamma = \frac{p + q}{2} + 1 \quad (11-38)$$

for $p + q = 2, 3, \dots$. A set of seven, 2-D *moment invariants* can be derived from the second and third normalized central moments:[†]

$$\phi_1 = \eta_{20} + \eta_{02} \quad (11-39)$$

$$\phi_2 = (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2 \quad (11-40)$$

$$\phi_3 = (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2 \quad (11-41)$$

$$\phi_4 = (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2 \quad (11-42)$$

[†]Derivation of these results requires concepts that are beyond the scope of this discussion. The book by Bell [1965] and the paper by Hu [1962] contain detailed discussions of these concepts. For generating moment invariants of an order higher than seven, see Flusser [2000]. Moment invariants can be generalized to n dimensions (see Mamistvalov [1998]).

$$\begin{aligned}\phi_5 = & (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12}) \left[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2 \right] \\ & + (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03}) \left[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2 \right]\end{aligned}\quad (11-43)$$

$$\begin{aligned}\phi_6 = & (\eta_{20} - \eta_{02}) \left[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2 \right] \\ & + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03})\end{aligned}\quad (11-44)$$

$$\begin{aligned}\phi_7 = & (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12}) \left[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2 \right] \\ & + (3\eta_{12} - \eta_{30})(\eta_{21} + \eta_{03}) \left[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2 \right]\end{aligned}\quad (11-45)$$

This set of moments is invariant to translation, scale change, mirroring (within a minus sign), and rotation. We can attach physical meaning to some of the low-order moment invariants. For example, ϕ_1 is the sum of two second moments with respect to the principal axes of data spread, so this moment can be interpreted as a measure of data spread. Similarly, ϕ_3 is the difference of second moments, and may be interpreted as a measure of “slenderness.” However, as the order of the moment invariants increases, the complexity of their formulation causes physical meaning to be lost. The importance of Eqs. (11-39) through (11-45) is their invariance, not their physical meaning.

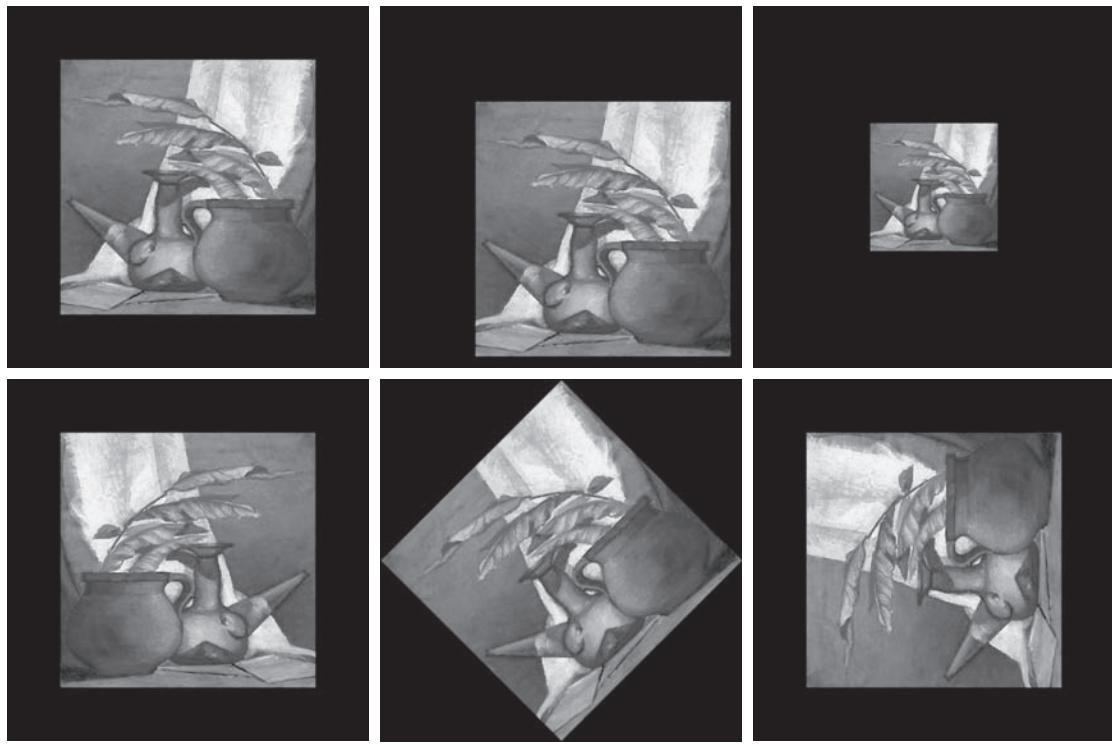
EXAMPLE 11.15: Moment invariants.

The objective of this example is to compute and compare the preceding moment invariants using the image in Fig. 11.37(a). The black (0) border was added to make all images in this example be of the same size; the zeros do not affect computation of the moment invariants. Figures 11.37(b) through (f) show the original image translated, scaled by 0.5 in both spatial dimensions, mirrored, rotated by 45°, and rotated by 90°, respectively. Table 11.5 summarizes the values of the seven moment invariants for these six images. To reduce dynamic range and thus simplify interpretation, the values shown are scaled using the expression $-\text{sgn}(\phi_i) \log_{10}(|\phi_i|)$. The absolute value is needed to handle any numbers that may be negative. The term $\text{sgn}(\phi_i)$ preserves the sign of ϕ_i , and the minus sign in front is there to handle fractions in the log computation. The idea is to make the numbers easier to interpret. Interest in this example is on the invariance and relative signs of the moments, not on their actual values. The two key points in Table 11.5 are: (1) the closeness of the values of the moments, independent of translation, scale change, mirroring and rotation; and (2) the fact that the sign of ϕ_7 is different for the mirrored image.

11.5 PRINCIPAL COMPONENTS AS FEATURE DESCRIPTORS

As we show in Example 11.17, principal components can be used also to normalize regions or boundaries for variations in size, translation, and rotation.

The material in this section is applicable to boundaries and regions. It is different from our discussion thus far, in the sense that features are based on more than one image. Suppose that we are given the three component images of a color image. The three images can be treated as a unit by expressing each group of three corresponding pixels as a vector, as discussed in Section 11.1. If we have a total of n registered



a	b	c
d	e	f

FIGURE 11.37 (a) Original image. (b)–(f) Images translated, scaled by one-half, mirrored, rotated by 45° , and rotated by 90° , respectively.

TABLE 11.5

Moment invariants for the images in Fig. 11.37.

Moment Invariant	Original Image	Translated	Half Size	Mirrored	Rotated 45°	Rotated 90°
ϕ_1	2.8662	2.8662	2.8664	2.8662	2.8661	2.8662
ϕ_2	7.1265	7.1265	7.1257	7.1265	7.1266	7.1265
ϕ_3	10.4109	10.4109	10.4047	10.4109	10.4115	10.4109
ϕ_4	10.3742	10.3742	10.3719	10.3742	10.3742	10.3742
ϕ_5	21.3674	21.3674	21.3924	21.3674	21.3663	21.3674
ϕ_6	13.9417	13.9417	13.9383	13.9417	13.9417	13.9417
ϕ_7	-20.7809	-20.7809	-20.7724	20.7809	-20.7813	-20.7809

images, then the corresponding pixels at the same spatial location in all images can be arranged as an n -dimensional vector:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad (11-46)$$

Throughout this section, the assumption is that all vectors are column vectors (i.e., matrices of order $n \times 1$). We can write them on a line of text simply by expressing them as $\mathbf{x} = (x_1, x_2, \dots, x_n)^T$, where T indicates the transpose.

You may find it helpful to review the tutorials on probability and matrix theory available on the book website.

We can treat the vectors as random quantities, just like we did when constructing an intensity histogram. The only difference is that, instead of talking about quantities like the mean and variance of the random variables, we now talk about *mean vectors* and *covariance matrices*. The mean vector of the population is defined as

$$\mathbf{m}_x = E\{\mathbf{x}\} \quad (11-47)$$

where $E\{\mathbf{x}\}$ is the expected value of \mathbf{x} , and the subscript denotes that \mathbf{m} is associated with the population of \mathbf{x} vectors. Recall that the expected value of a vector or matrix is obtained by taking the expected value of each element.

The covariance matrix of the vector population is defined as

$$\mathbf{C}_x = E\{(\mathbf{x} - \mathbf{m}_x)(\mathbf{x} - \mathbf{m}_x)^T\} \quad (11-48)$$

Because \mathbf{x} is n dimensional, \mathbf{C}_x is an $n \times n$ matrix. Element c_{ii} of \mathbf{C}_x is the variance of x_i , the i th component of the \mathbf{x} vectors in the population, and element c_{ij} of \mathbf{C}_x is the covariance between elements x_i and x_j of these vectors. Matrix \mathbf{C}_x is real and symmetric. If elements x_i and x_j are uncorrelated, their covariance is zero and, therefore, $c_{ij} = 0$, resulting in a diagonal covariance matrix.

Because \mathbf{C}_x is real and symmetric, finding a set of n orthonormal eigenvectors is always possible (Noble and Daniel [1988]). Let \mathbf{e}_i and λ_i , $i = 1, 2, \dots, n$, be the eigenvectors and corresponding eigenvalues of \mathbf{C}_x ,[†] arranged (for convenience) in descending order so that $\lambda_j \geq \lambda_{j+1}$ for $j = 1, 2, \dots, n-1$. Let \mathbf{A} be a matrix whose rows are formed from the eigenvectors of \mathbf{C}_x , arranged in descending value of their eigenvalues, so that the first row of \mathbf{A} is the eigenvector corresponding to the largest eigenvalue.

Suppose that we use \mathbf{A} as a transformation matrix to map the \mathbf{x} 's into vectors denoted by \mathbf{y} 's, as follows:

$$\mathbf{y} = \mathbf{A}(\mathbf{x} - \mathbf{m}_x) \quad (11-49)$$

The Hotelling transform is the same as the discrete Karhunen-Loëve transform, so the two names are used interchangeably in the literature.

This expression is called the *Hotelling transform*, which, as you will learn shortly, has some very interesting and useful properties.

[†]By definition, the eigenvector and eigenvalues of an $n \times n$ matrix \mathbf{C} satisfy the equation $\mathbf{C}\mathbf{e}_i = \lambda_i\mathbf{e}_i$.

It is not difficult to show (see Problem 11.25) that the mean of the \mathbf{y} vectors resulting from this transformation is zero; that is,

$$\mathbf{m}_y = E\{\mathbf{y}\} = \mathbf{0} \quad (11-50)$$

It follows from basic matrix theory that the covariance matrix of the \mathbf{y} 's is given in terms of \mathbf{A} and \mathbf{C}_x by the expression

$$\mathbf{C}_y = \mathbf{AC}_x\mathbf{A}^T \quad (11-51)$$

Furthermore, because of the way \mathbf{A} was formed, \mathbf{C}_y is a diagonal matrix whose elements along the main diagonal are the eigenvalues of \mathbf{C}_x ; that is,

$$\mathbf{C}_y = \begin{bmatrix} \lambda_1 & & & 0 \\ & \lambda_2 & & \\ & & \ddots & \\ 0 & & & \lambda_n \end{bmatrix} \quad (11-52)$$

The off-diagonal elements of this covariance matrix are 0, so the elements of the \mathbf{y} vectors are uncorrelated. Keep in mind that the λ_i are the eigenvalues of \mathbf{C}_x and that the elements along the main diagonal of a diagonal matrix are its eigenvalues (Noble and Daniel [1988]). Thus, \mathbf{C}_x and \mathbf{C}_y have the same eigenvalues.

Another important property of the Hotelling transform deals with the reconstruction of \mathbf{x} from \mathbf{y} . Because the rows of \mathbf{A} are orthonormal vectors, it follows that $\mathbf{A}^{-1} = \mathbf{A}^T$, and any vector \mathbf{x} can be recovered from its corresponding \mathbf{y} by using the expression

$$\mathbf{x} = \mathbf{A}^T \mathbf{y} + \mathbf{m}_x \quad (11-53)$$

But, suppose that, instead of using all the eigenvectors of \mathbf{C}_x , we form a matrix \mathbf{A}_k from the k eigenvectors corresponding to the k largest eigenvalues, yielding a transformation matrix of order $k \times n$. The \mathbf{y} vectors would then be k dimensional, and the reconstruction given in Eq. (11-53) would no longer be exact (this is somewhat analogous to the procedure we used in Section 11.3 to describe a boundary with a few Fourier coefficients).

The vector reconstructed by using \mathbf{A}_k is

$$\hat{\mathbf{x}} = \mathbf{A}_k^T \mathbf{y} + \mathbf{m}_x \quad (11-54)$$

It can be shown that the mean squared error between \mathbf{x} and $\hat{\mathbf{x}}$ is given by the expression

$$e_{ms} = \sum_{j=1}^n \lambda_j - \sum_{j=1}^k \lambda_j = \sum_{j=k+1}^n \lambda_j \quad (11-55)$$

Equation (11-55) indicates that the error is zero if $k = n$ (that is, if all the eigenvectors are used in the transformation). Because the λ_j 's decrease monotonically,

Eq. (11-55) also shows that the error can be minimized by selecting the k eigenvectors associated with the largest eigenvalues. Thus, the Hotelling transform is optimal in the sense that it minimizes the mean squared error between the vectors \mathbf{x} and their approximations $\hat{\mathbf{x}}$. Due to this idea of using the eigenvectors corresponding to the largest eigenvalues, the Hotelling transform also is known as the *principal components transform*.

EXAMPLE 11.16: Using principal components for image description.

Figure 11.38 shows six multispectral satellite images corresponding to six spectral bands: visible blue (450–520 nm), visible green (520–600 nm), visible red (630–690 nm), near infrared (760–900 nm), middle infrared (1550–1,750 nm), and thermal infrared (10,400–12,500 nm). The objective of this example is to illustrate how to use principal components as image features.

Organizing the images as in Fig. 11.39 leads to the formation of a six-element vector \mathbf{x} from each set of corresponding pixels in the images, as discussed earlier in this section. The images in this example are of size 564×564 pixels, so the population consisted of $(564)^2 = 318,096$ vectors from which the mean vector, covariance matrix, and corresponding eigenvalues and eigenvectors were computed. The

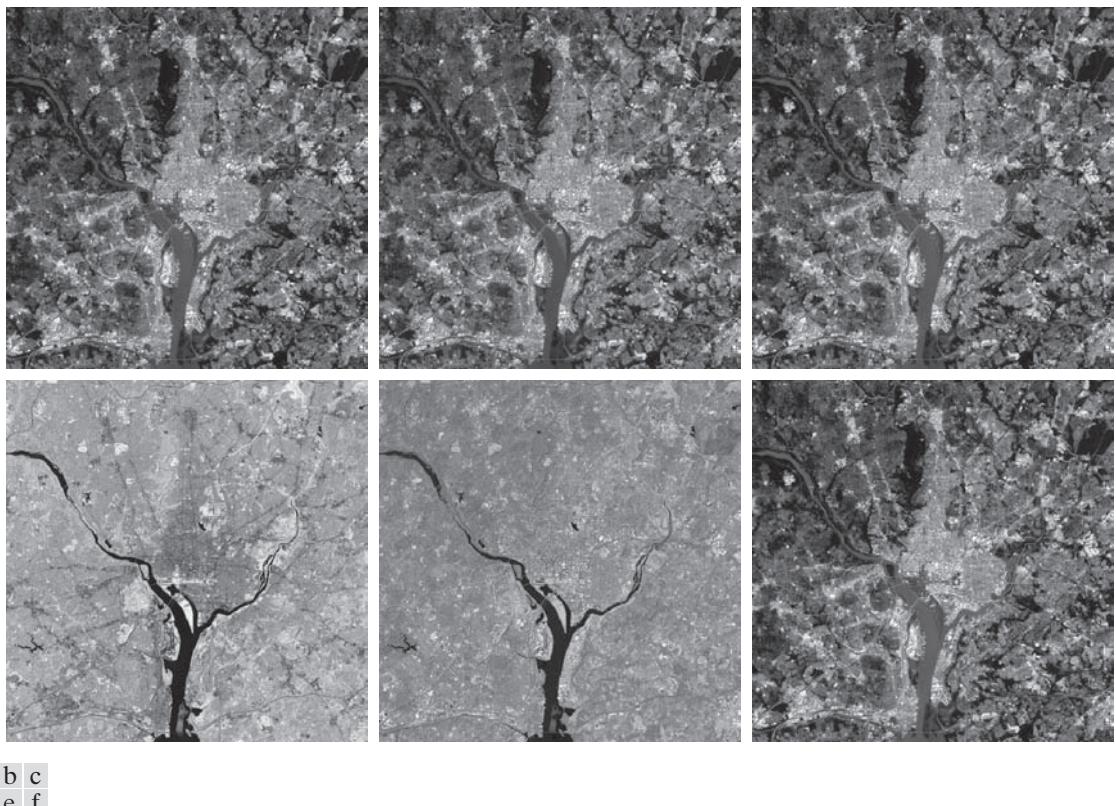
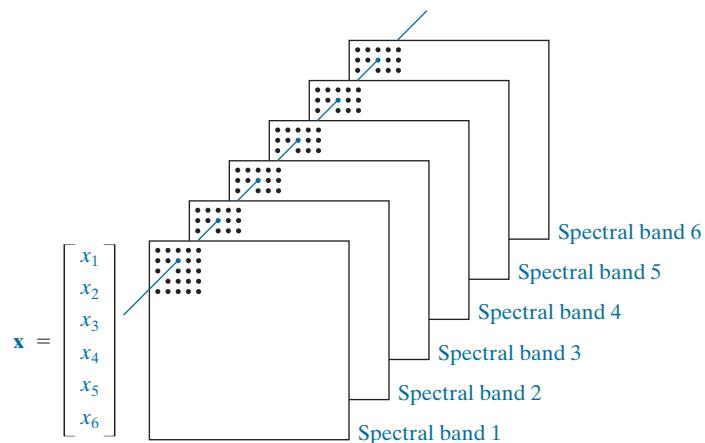


FIGURE 11.38 Multispectral images in the (a) visible blue, (b) visible green, (c) visible red, (d) near infrared, (e) middle infrared, and (f) thermal infrared bands. (Images courtesy of NASA.)

FIGURE 11.39

Forming of a feature vector from corresponding pixels in six images.



eigenvectors were then used as the rows of matrix \mathbf{A} , and a set of \mathbf{y} vectors were obtained using Eq. (11-49). Similarly, we used Eq. (11-51) to obtain \mathbf{C}_y . Table 11.6 shows the eigenvalues of this matrix. Note the dominance of the first two eigenvalues.

A set of principal component images was generated using the \mathbf{y} vectors mentioned in the previous paragraph (images are constructed from vectors by applying Fig. 11.39 in reverse). Figure 11.40 shows the results. Figure 11.40(a) was formed from the first component of the 318,096 \mathbf{y} vectors, Fig. 11.40(b) from the second component of these vectors, and so on, so these images are of the same size as the original images in Fig. 11.38. The most obvious feature in the principal component images is that a significant portion of the contrast detail is contained in the first two images, and it decreases rapidly from there. The reason can be explained by looking at the eigenvalues. As Table 11.6 shows, the first two eigenvalues are much larger than the others. Because the eigenvalues are the variances of the elements of the \mathbf{y} vectors, and variance is a measure of intensity contrast, it is not unexpected that the images formed from the vector components corresponding to the largest eigenvalues would exhibit the highest contrast. In fact, the first two images in Fig. 11.40 account for about 89% of the total variance. The other four images have low contrast detail because they account for only the remaining 11%.

According to Eqs. (11-54) and (11-55), if we used all the eigenvectors in matrix \mathbf{A} we could reconstruct the original images from the principal component images with zero error between the original and reconstructed images (i.e., the images would be identical). If the objective is to store and/or transmit the principal component images and the transformation matrix for later reconstruction of the original images, it would make no sense to store and/or transmit all the principal component images because nothing would be gained. Suppose, however, that we keep and/or transmit only the two principal component images. Then there would be significant savings in storage and/or transmission (matrix \mathbf{A} would be of size 2×6 , so its impact would be negligible).

Figure 11.41 shows the results of reconstructing the six multispectral images from the two principal component images corresponding to the largest eigenvalues. The first five images are quite close in

TABLE 11.6

Eigenvalues of \mathbf{C}_x obtained from the images in Fig. 11.38.

λ_1	λ_2	λ_3	λ_4	λ_5	λ_6
10344	2966	1401	203	94	31

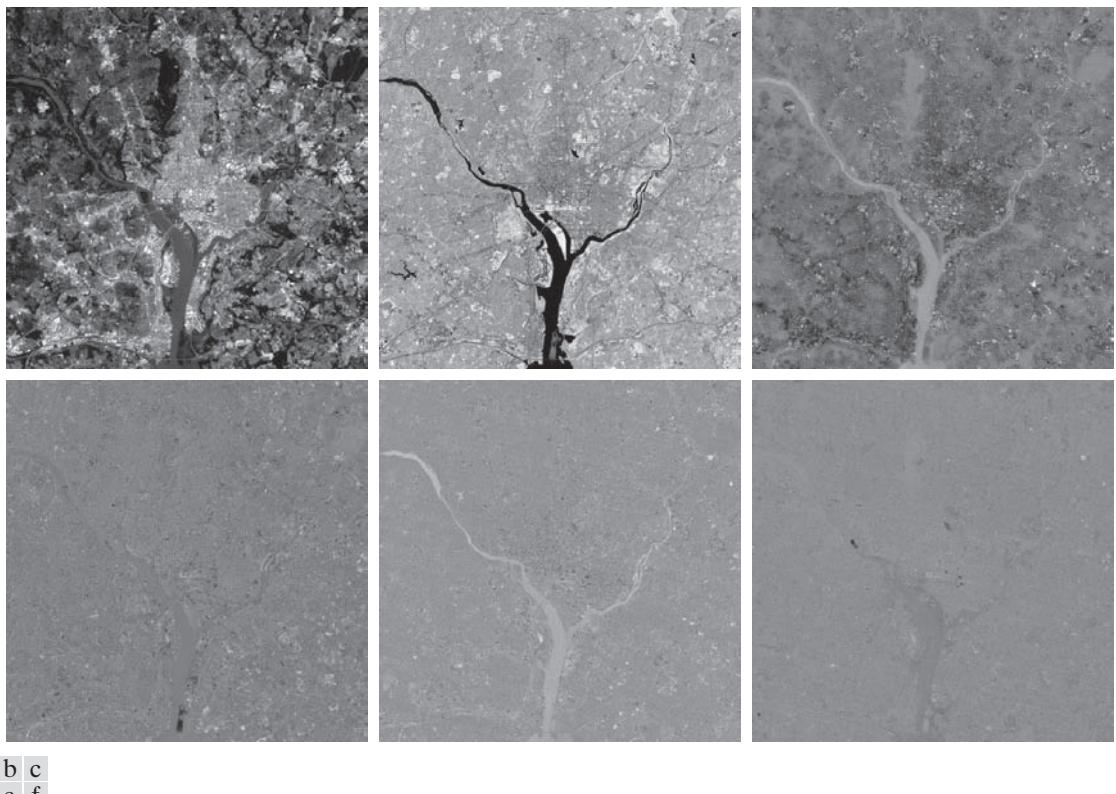


FIGURE 11.40 The six principal component images obtained from vectors computed using Eq. (11-49). Vectors are converted to images by applying Fig. 11.39 in reverse.

appearance to the originals in Fig. 11.38, but this is not true for the sixth image. The reason is that the original sixth image is actually blurry, but the two principal component images used in the reconstruction are sharp, therefore, the blurry “detail” is lost. Figure 11.42 shows the differences between the original and reconstructed images. The images in Fig. 11.42 were enhanced to highlight the differences between them. If they were shown without enhancement, the first five images would appear almost all black, with the sixth (difference) image showing the most variability.

EXAMPLE 11.17: Using principal components for normalizing for variations in size, translation, and rotation.

As we mentioned earlier in this chapter, feature descriptors should be as independent as possible of variations in size, translation, and rotation. Principal components provide a convenient way to normalize boundaries and/or regions for variations in these three variables. Consider the object in Fig. 11.43, and assume that its size, location, and orientation (rotation) are arbitrary. The points in the region (or its boundary) may be treated as 2-D vectors, $\mathbf{x} = (x_1, x_2)^T$, where x_1 and x_2 are the coordinates of any object point. All the points in the region or boundary constitute a 2-D vector population that can be used to compute the covariance matrix \mathbf{C}_x and mean vector \mathbf{m}_x . One eigenvector of \mathbf{C}_x points in the direction

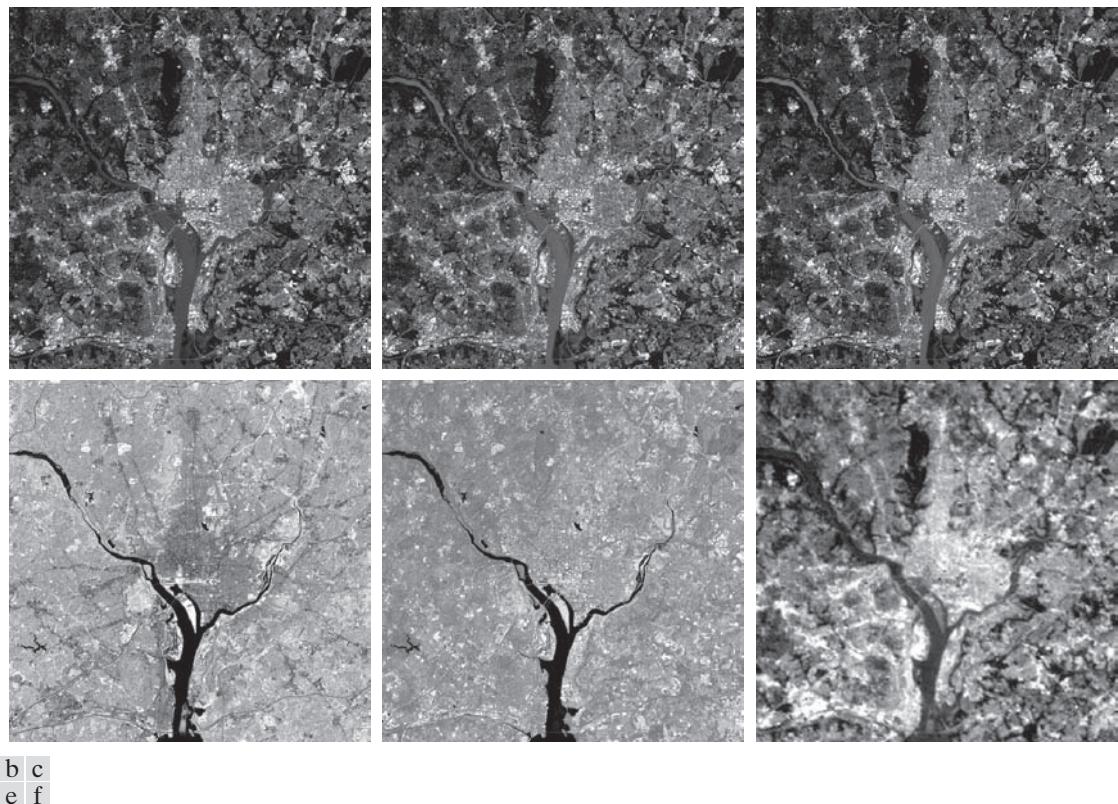
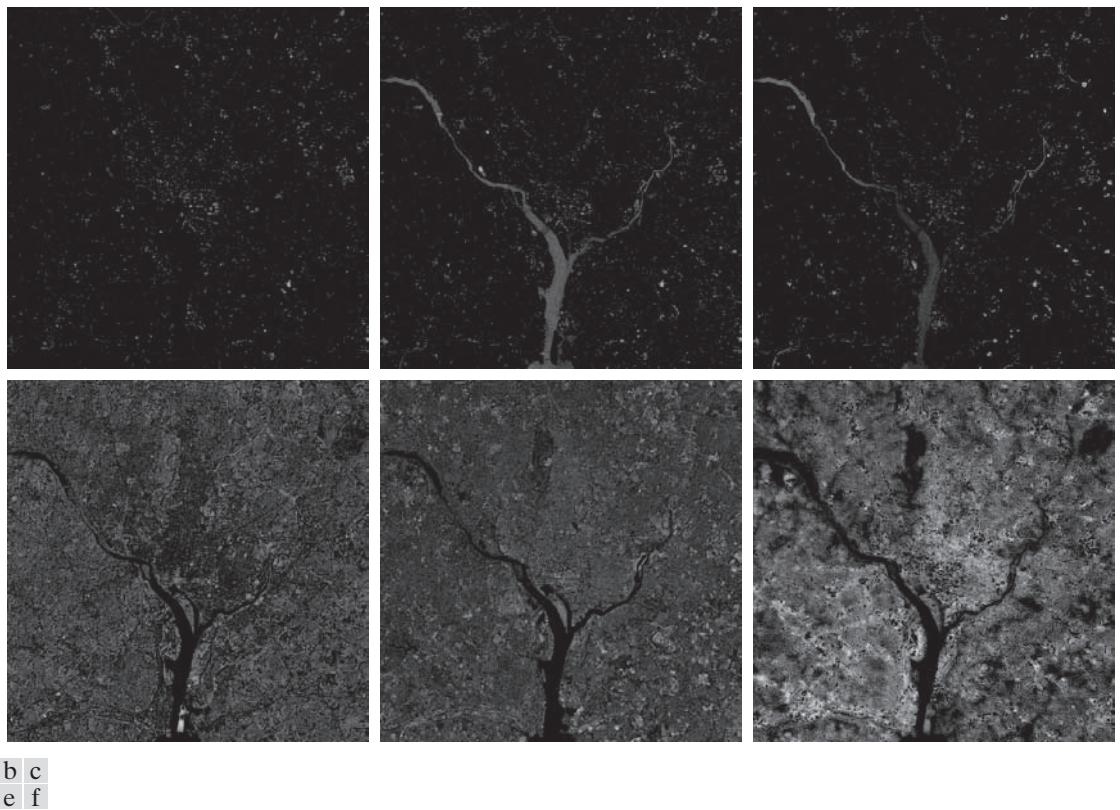


FIGURE 11.41 Multispectral images reconstructed using only the two principal component images corresponding to the two principal component vectors with the largest eigenvalues. Compare these images with the originals in Fig. 11.38.

of maximum variance (data spread) of the population, while the second eigenvector is perpendicular to the first, as Fig. 11.43(b) shows. In terms of the present discussion, the principal components transform in Eq. (11-49) accomplishes two things: (1) it establishes the center of the transformed coordinates system as the centroid (mean) of the population because \mathbf{m}_x is subtracted from each \mathbf{x} ; and (2) the \mathbf{y} coordinates (vectors) it generates are rotated versions of the \mathbf{x} 's, so that the data align with the eigenvectors. If we define a (y_1, y_2) axis system so that y_1 is along the first eigenvector and y_2 is along the second, then the geometry that results is as illustrated in Fig. 11.43(c). That is, the dominant data directions are aligned with the new axis system. The same result will be obtained regardless of the size, translation, or rotation of the object, provided that all points in the region or boundary undergo the same transformation. If we wished to size-normalize the transformed data, we would divide the coordinates by the corresponding eigenvalues.

Observe in Fig. 11.43(c) that the points in the y -axes system can have both positive and negative values. To convert all coordinates to positive values, we simply subtract the vector $(y_{1\min}, y_{2\min})^T$ from all the \mathbf{y} vectors. To displace the resulting points so that they are all greater than 0, as in Fig. 11.43(d), we add to them a vector $(a, b)^T$ where a and b are greater than 0.

Although the preceding discussion is straightforward in principle, the mechanics are a frequent source of confusion. Thus, we conclude this example with a simple manual illustration. Figure 11.44(a) shows



a b c
d e f

FIGURE 11.42 Differences between the original and reconstructed images. All images were enhanced by scaling them to the full [0, 255] range to facilitate visual analysis.

four points with coordinates $(1, 1)$, $(2, 4)$, $(4, 2)$, and $(5, 5)$. The mean vector, covariance matrix, and normalized (unit length) eigenvectors of this population are:

$$\mathbf{m}_x = \begin{bmatrix} 3 \\ 3 \end{bmatrix}, \quad \mathbf{C}_x = \begin{bmatrix} 3.333 & 2.00 \\ 2.00 & 3.333 \end{bmatrix}$$

and

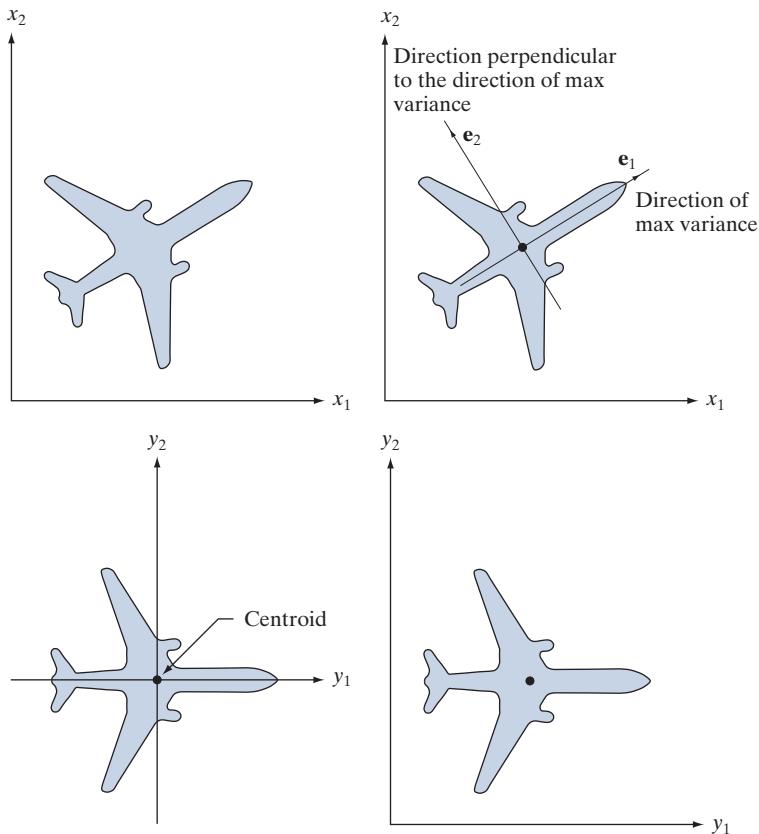
$$\mathbf{e}_1 = \begin{bmatrix} 0.707 \\ 0.707 \end{bmatrix}, \quad \mathbf{e}_2 = \begin{bmatrix} -0.707 \\ 0.707 \end{bmatrix}$$

The corresponding eigenvalues are $\lambda_1 = 5.333$ and $\lambda_2 = 1.333$. Figure 11.44(b) shows the eigenvectors superimposed on the data. From Eq. (11-49), the transformed points (the \mathbf{y} 's) are $(-2.828, 0)^T$, $(0, -1.414)^T$, $(0, 1.414)^T$, and $(2.828, 0)^T$. These points are plotted in Fig. 11.44(c). Note that they are aligned with the y -axes and that they have fractional values. When working with images, coordinate values are integers, making it necessary to round all values to their nearest integer value. Figure 11.44(d) shows the points rounded to the nearest integer and their location shifted so that all coordinate values are integers greater than 0, as in the original figure.

a	b
c	d

FIGURE 11.43

- (a) An object.
- (b) Object showing eigenvectors of its covariance matrix.
- (c) Transformed object, obtained using Eq. (11-49).
- (d) Object translated so that all its coordinate values are greater than 0.



When transforming image pixels, keep in mind that image coordinates are the same as matrix coordinates; that is, (x, y) represents (r, c) , and the origin is the top left. Axes of the principal components just illustrated are as shown in Figs. 11.43(a) and (d). You need to keep this in mind in interpreting the results of applying a principal components transformation to objects in an image.

11.6 WHOLE-IMAGE FEATURES

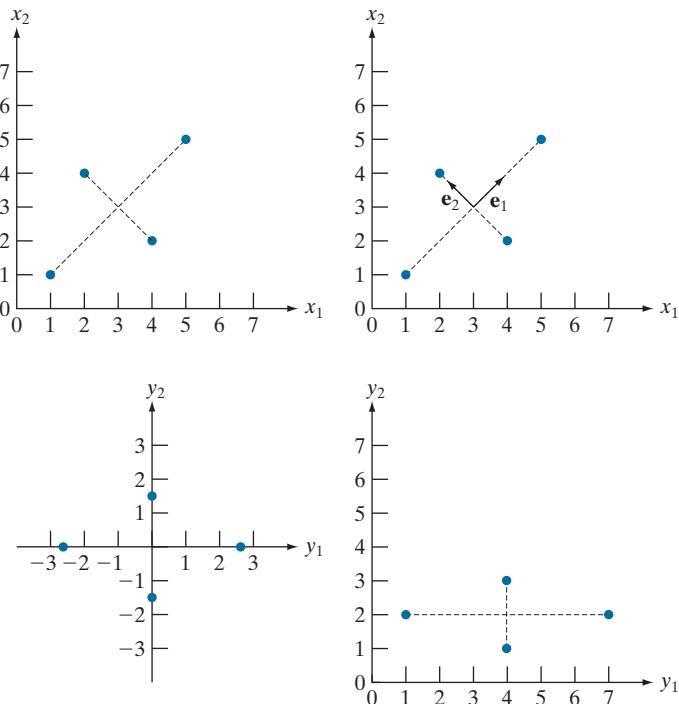
The descriptors introduced in Sections 11.2 through 11.4 are well suited for applications (e.g., industrial inspection), in which *individual* regions can be segmented reliably using methods such as the ones discussed in Chapters 10 and 11. With the exception of the application in Example 11.17, the principal components feature vectors in Section 11.5 are different from the earlier material, in the sense that they are based on multiple images. But even these descriptors are localized to sets of corresponding pixels. In some applications, such as searching image databases for matches (e.g., as in human face recognition), the variability between images is so extensive that the methods in Chapters 10 and 11 are not applicable.

a	b
c	d

FIGURE 11.44

A manual example.

- (a) Original points.
- (b) Eigenvectors of the covariance matrix of the points in (a).
- (c) Transformed points obtained using Eq. (11-49).
- (d) Points from (c), rounded and translated so that all coordinate values are integers greater than 0. The dashed lines are included to facilitate viewing. They are not part of the data.



The discussion in Sections 12.5 through 12.7 dealing with neural networks is also important in terms of processing large numbers of entire images for the purpose of characterizing their content.

Our use the term “corner” is broader than just 90° corners; it refers to features that are “corner-like.”

The state of the art in image processing is such that as the complexity of the task increases, the number of techniques suitable for addressing those tasks decreases. This is particularly true when dealing with feature descriptors applicable to entire images that are members of a large family of images. In this section, we discuss two of the principal feature detection methods currently being used for this purpose. One is based on detecting corners, and the other works with entire regions in an image. Then, in Section 11.7 we present a feature detection and description approach designed specifically to work with these types of features.

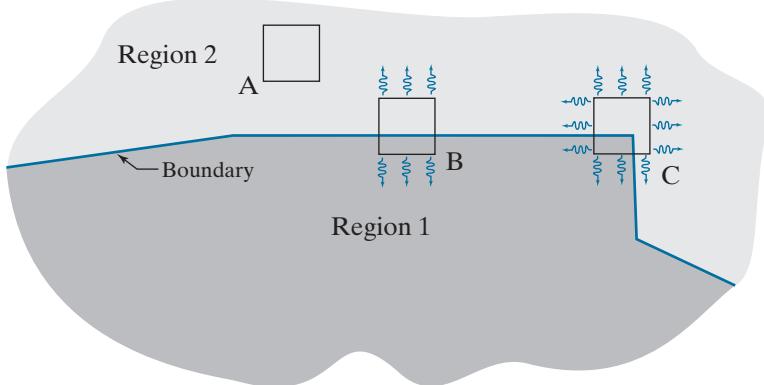
THE HARRIS-STEVENS CORNER DETECTOR

Intuitively, we think of a corner as a rapid change of direction in a curve. Corners are highly effective features because they are distinctive and reasonably invariant to viewpoint. Because of these characteristics, corners are used routinely for matching image features in applications such as tracking for autonomous navigation, stereo machine vision algorithms, and image database queries.

In this section, we discuss an algorithm for corner detection formulated by Harris and Stephens [1988]. The idea behind the *Harris-Stephens (HS) corner detector* is illustrated in Fig. 11.45. The basic approach is this: Corners are detected by running a small window over an image, as we did in Chapter 3 for spatial filtering. The detector window is designed to compute intensity changes. We are interested in three scenarios: (1) Areas of zero (or small) intensity changes in all directions, which

FIGURE 11.45

Illustration of how the Harris-Stephens corner detector operates in the three types of sub-regions indicated by A (flat), B (edge), and C (corner). The wiggly arrows indicate graphically a directional response in the detector as it moves in the three areas shown.



happens when the window is located in a constant (or nearly constant) region, as in location A in Fig. 11.45; (2) areas of changes in one direction but no (or small) changes in the orthogonal direction, which this happens when the window spans a boundary between two regions, as in location B; and (3) areas of significant changes in all directions, a condition that happens when the window contains a corner (or isolated points), as in location C. The HS corner detector is a mathematical formulation that attempts to differentiate between these three conditions.

Let f denote an image, and let $f(s,t)$ denote a patch of the image defined by the values of (s,t) . A patch of the same size, but shifted by (x,y) , is given by $f(s+x,t+y)$. Then, the weighted sum of squared differences between the two patches is given by

$$C(x,y) = \sum_s \sum_t w(s,t) [f(s+x,t+y) - f(s,t)]^2 \quad (11-56)$$

where $w(s,t)$ is a weighting function to be discussed shortly. The shifted patch can be approximated by the linear terms of a Taylor expansion

$$f(s+x,t+y) \approx f(s,t) + xf_x(s,t) + yf_y(s,t) \quad (11-57)$$

where $f_x(s,t) = \partial f / \partial x$ and $f_y(s,t) = \partial f / \partial y$, both evaluated at (s,t) . We can then write Eq. (11-56) as

$$C(x,y) = \sum_s \sum_t w(s,t) [xf_x(s,t) + yf_y(s,t)]^2 \quad (11-58)$$

This equation can be written in matrix form as

$$C(x,y) = [x \ y] \mathbf{M} \begin{bmatrix} x \\ y \end{bmatrix} \quad (11-59)$$

where

$$\mathbf{M} = \sum_s \sum_t w(s,t) \mathbf{A} \quad (11-60)$$

and

$$\mathbf{A} = \begin{bmatrix} f_x^2 & f_x f_y \\ f_x f_y & f_y^2 \end{bmatrix} \quad (11-61)$$

Matrix \mathbf{M} sometimes is called the *Harris matrix*. It is understood that its terms are evaluated at (s,t) . If $w(s,t)$ is isotropic, then \mathbf{M} is symmetric because \mathbf{A} is. The weighting function $w(s,t)$ used in the HS detector generally has one of two forms: (1) it is 1 inside the patch and 0 elsewhere (i.e., it has the shape of a box lowpass filter kernel), or (2) it is an exponential function of the form

$$w(s,t) = e^{-(s^2 + t^2)/2\sigma^2} \quad (11-62)$$

The box is used when computational speed is paramount and the noise level is low. The exponential form is used when data smoothing is important.

As illustrated in Fig. 11.45, a corner is characterized by large values in region C, in *both* spatial directions. However, when the patch spans a boundary there will also be a response in one direction. The question is: How can we tell the difference? As we discussed in Section 11.5 (see Example 11.17), the eigenvectors of a real, symmetric matrix (such as \mathbf{M} above) point in the direction of maximum data spread, and the corresponding eigenvalues are proportional to the amount of data spread in the direction of the eigenvectors. In fact, the eigenvectors are the major axes of an ellipse fitting the data, and the magnitude of the eigenvalues are the distances from the center of the ellipse to the points where it intersects the major axes. Figure 11.46 illustrates how we can use these properties to differentiate between the three cases in which we are interested.

The small image patches in Figs. 11.46(a) through (c) are representative of regions A, B, and C in Fig. 11.45. In Fig. 11.46(d), we show values of (f_x, f_y) computed using the derivative kernels $w_y = [-1 \ 0 \ 1]$ and $w_x = w_y^T$ (remember, we use the coordinate system defined in Fig. 2.19). Because we compute the derivatives at each point in the patch, variations caused by noise result in scattered values, with the spread of the scatter being directly related to the noise level and its properties. As expected, the derivatives from the flat region form a nearly circular cluster, whose eigenvalues are almost identical, yielding a nearly circular fit to the points (we label these eigenvalues as “small” in relation to the other two plots). Figure 11.46(e) shows the derivatives of the patch containing the edge. Here, the spread is greater along the x -axis, and about nearly the same as Fig. 11.46 (a) in the y -axis. Thus, eigenvalue λ_x is “large” while λ_y is “small.” Consequently, the ellipse fitting the data is elongated in the x -direction. Finally, Fig. 11.46(f) shows the derivatives of the patch containing the corner. Here, the data is spread along both directions, resulting in two large eigenvalues and a much larger and nearly circular fitting ellipse. From this we conclude that: (1) two small eigenvalues indicate nearly constant intensity; (2) one small and one large eigenvalue

As noted in Chapter 3, we do not use bold notation for vectors and matrices representing spatial kernels.

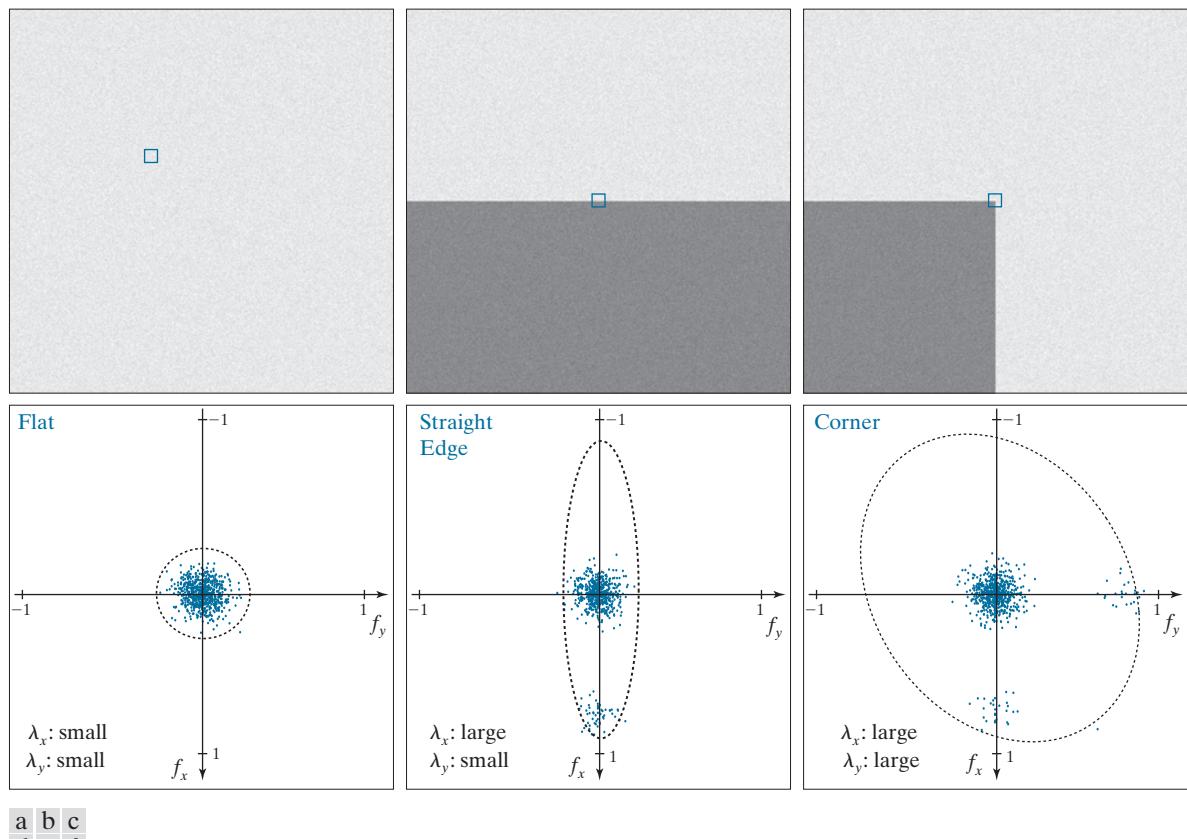


FIGURE 11.46 (a)–(c) Noisy images and image patches (small squares) encompassing image regions similar in content to those in Fig. 11.45. (d)–(f) Plots of value pairs (f_x, f_y) showing the characteristics of the eigenvalues of \mathbf{M} that are useful for detecting the presence of a corner in an image patch.

The eigenvalues of the 2×2 matrix \mathbf{M} can be expressed in a closed form (see Problem 11.31). However, their computation requires squares and square roots, which are expensive to process.

The advantage of this formulation is that the trace is the sum of the main diagonal terms of \mathbf{M} (just two numbers). The determinant of a 2×2 matrix is the product of the main diagonal elements minus the product of the cross elements. These are trivial computations.

imply the presence of a vertical or horizontal boundary; and (3) two large eigenvalues imply the presence of a corner or (unfortunately) isolated bright points.

Thus, we see that the eigenvalues of the matrix formed from derivatives in the image patch can be used to differentiate between the three scenarios of interest. However, instead of using the eigenvalues (which are expensive to compute), the HS detector utilizes a measure of corner response based on the fact that the trace of a square matrix is equal to the sum of its eigenvalues, and its determinant is equal to the product of its eigenvalues. The measure is defined as

$$\begin{aligned} R &= \lambda_x \lambda_y - k(\lambda_x + \lambda_y)^2 \\ &= \det(\mathbf{M}) - k \text{trace}^2(\mathbf{M}) \end{aligned} \quad (11-63)$$

where k is a constant to be explained shortly. Measure R has large positive values when both eigenvalues are large, indicating the presence of a corner; it has large negative values when one eigenvalue is large and the other small, indicating an edge;

and its absolute value is small when both eigenvalues are small, indicating that the image patch under consideration is flat.

Constant k is determined empirically, and its range of values depends on the implementation. For example, the MATLAB Image Processing Toolbox uses $0 < k < 0.25$. You can interpret k as a “sensitivity factor;” the smaller it is, the more likely the detector is to find corners. Typically, R is used with a threshold, T . We say that a corner at an image location has been detected only if $R > T$ for a patch at that location.

EXAMPLE 11.18: Applying the HS corner detector.

Figure 11.47(a) shows a noisy image, and Fig. 11.47(b) is the result of using the HS corner detector with $k = 0.04$ and $T = 0.01$ (the default values in our implementation). All corners of the squares were detected correctly, but the number of false detections is too high (note that all errors occurred on the right side of the image, where the difference in intensity between squares is less). Figure 11.47(c) shows

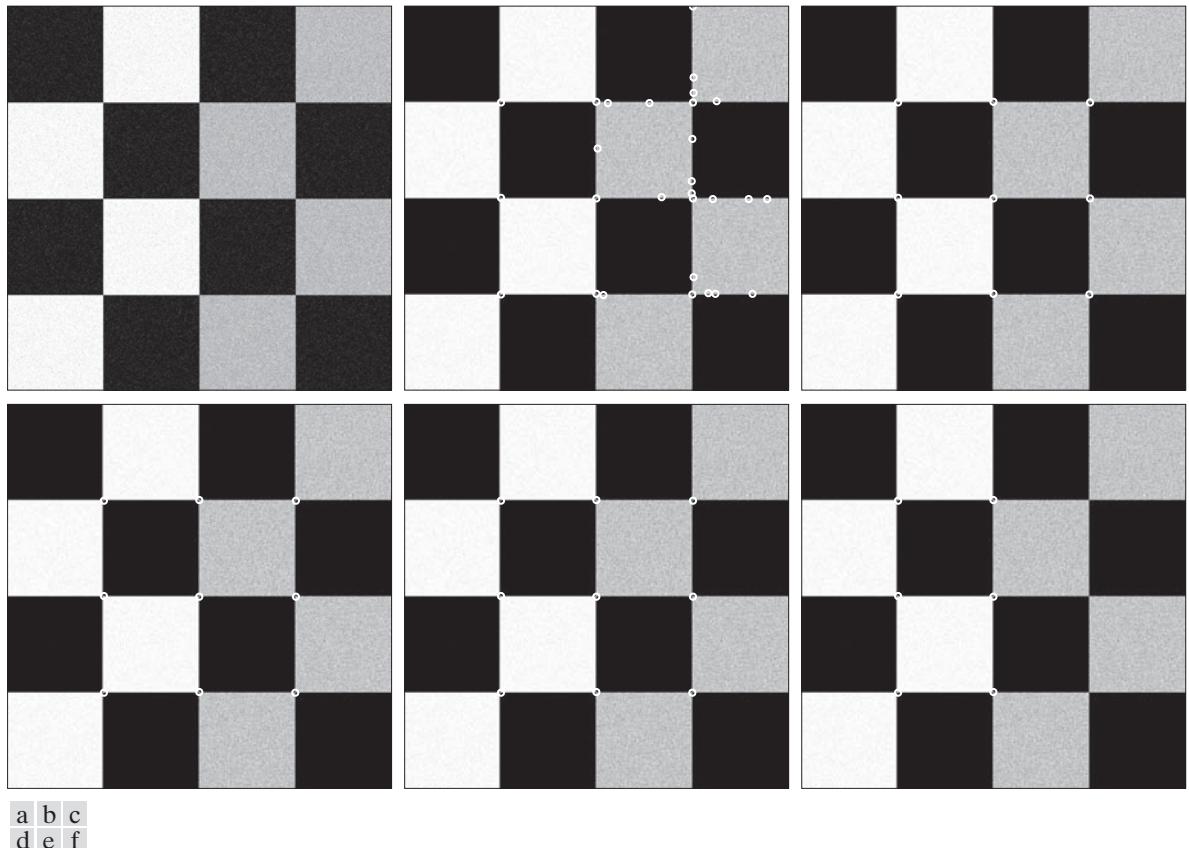
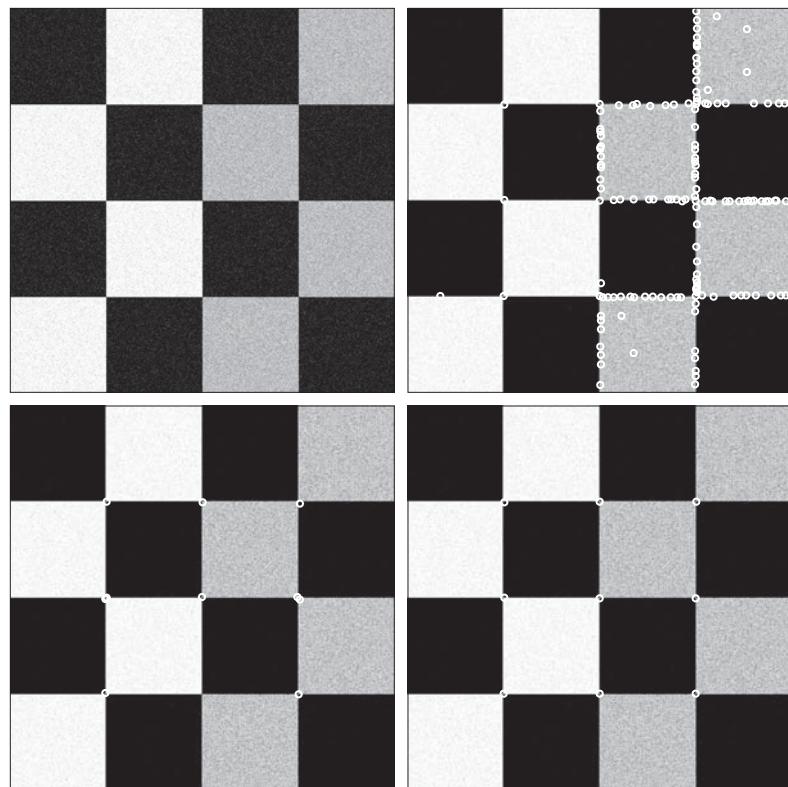


FIGURE 11.47 (a) A 600×600 image with values in the range $[0,1]$, corrupted by additive Gaussian noise with 0 mean and variance of 0.006. (b) Result of applying the HS corner detector with $k = 0.04$ and $T = 0.01$ (the defaults). Several errors are visible. (c) Result using $k = 0.1$ and $T = 0.01$. (d) Result using $k = 0.1$ and $T = 0.1$. (e) Result using $k = 0.04$ and $T = 0.1$. (f) Result using $k = 0.04$ and $T = 0.3$ (only the strongest corners on the left were detected).

a	b
c	d

FIGURE 11.48

- (a) Same as Fig. 11.47(a), but corrupted with Gaussian noise of mean 0 and variance 0.01.
 (b) Result of using the HS detector with $k = 0.04$ and $T = 0.01$ [compare with Fig. 11.47(b)].
 (c) Result with $k = 0.249$, (near the highest value in our implementation), and $T = 0.01$.
 (d) Result of using $k = 0.04$ and $T = 0.15$.



the result obtained by increasing k to 0.1 and leaving T at 0.01. This time, all corners were detected correctly. As Fig. 11.47(d) shows, increasing the threshold to $T = 0.1$ yielded the same result. In fact, using the default value of k and leaving T at 0.1 also produced the same result, as Fig. 11.47(e) shows. The point of all this is that there is considerable flexibility in the interplay between the values of k and T . Figure 11.47(f) shows the result obtained using the default value for k and using $T = 0.3$. As expected, increasing the value of the threshold eliminated some corners, yielding in this case only the corner of the squares with larger intensity differences. Increasing the value of k to 0.1 and setting T to its default value yielded the same result, as did using $k = 0.1$ and $T = 0.3$, demonstrating again the flexibility in the values chosen for these two parameters. However, as the level of noise increases, the range of possible values becomes narrower, as the results in the next paragraph illustrate.

Figure 11.48(a) shows the checkerboard corrupted by a much higher level of additive Gaussian noise (see the figure caption). Although this image does not appear much different than Fig. 11.47(a), the results using the default values of k and T are much worse than before. False corners were detected even on the left side of the image, where the intensity differences are much stronger. Figure 11.48(c) is the result of increasing k near the maximum value in our implementation (2.5) while keeping T at its default value. This time, k alone could not overcome the higher noise level. On the other hand, decreasing k to its default value and increasing T to 0.15 produced a perfect result, as Fig. 11.48(d) shows.

Figure 11.49(a) shows a more complex image with a significant number of corners embedded in various ranges of intensities. Figure 11.49(b) is the result obtained using the default values for k and T .



FIGURE 11.49 600×600 image of a building. (b) Result of applying the HS corner detector with $k = 0.04$ and $T = 0.01$ (the default values in our implementation). Numerous irrelevant corners were detected. (c) Result using $k = 0.249$ and the default value for T . (d) Result using $k = 0.17$ and $T = 0.05$. (e) Result using the default value for k and $T = 0.05$. (f) Result using the default value of k and $T = 0.07$.

As you can see, numerous detection errors occurred (see, for example, the large number of wrong corner detections in the right edge of the building). Increasing k alone had little effect on the over-detection of corners until k was near its maximum value. Using the same values as in Fig. 11.48(c) resulted in the image in 11.49(c), which shows a reduced number of erroneous corners, at the expense of missing numerous important ones in the front of the building. Reducing k to 0.17 and increasing T to 0.05 did a much better job, as Fig. 11.49(d) shows. Parameter k did not play a major role in corner detection for the building image. In fact, Figs. 11.49(e) and (f) show essentially the same level of performance obtained by reducing k to its default value of 0.04, and using $T = 0.05$ and $T = 0.07$, respectively.

Finally, Fig. 11.50 shows corner detection on a rotated image. The result in Fig. 11.50(b) was obtained using the same parameters we used in Fig. 11.49(f), showing the relative insensitivity of the method to rotation. Figures 11.49(f) and 11.50(b) show detection of at least one corner in every major structural feature of the image, such as the front door, all the windows, and the corners that define the apex of the facade. For matching purposes, these are excellent results.

a b

FIGURE 11.50

(a) Image rotated 5°.
 (b) Corners detected using the parameters used to obtain Fig. 11.49(f).



MAXIMALLY STABLE EXTREMAL REGIONS (MSERs)

The Harris-Stephens corner detector discussed in the previous section is useful in applications characterized by sharp transitions of intensities, such as the intersection of straight edges, that result in corner-like features in an image. Conversely, the maximally stable extremal regions (MSERs) introduced by Matas et al. [2002] are more “blob” oriented. As with the HS corner detector, MSERs are intended to yield whole image features for the purpose of establishing correspondence between two or more images.

We know from Fig. 2.18 that a grayscale image can be viewed as a topographic map, with the xy -axes representing spatial coordinates, and the z -axis representing intensities. Imagine that we start thresholding an 8-bit grayscale image one intensity level at a time. The result of each thresholding is a binary image in which we show the pixels at or above the threshold in white, and the pixels below the threshold as black. When the threshold, T , is 0, the result is a white image (all pixel values are at or above 0). As we start increasing T in increments of one intensity level, we will begin to see black components in the resulting binary images. These correspond to local minima in the topographic map view of the image. These black regions may begin to grow and merge, but they never get smaller from image to image. Finally, when we reach $T = 255$, the resulting image will be black (there are no pixel values above this level). Because each stage of thresholding results in a binary image, there will be one or more connected components of white pixels in each image. The set of all such components resulting from all thresholdings is the set of *extremal regions*. Extremal regions that do not change size (number of pixels) appreciably over a range of threshold values are called *maximally stable extremal regions*.

As you will see shortly, the procedure just discussed can be cast in the form of a rooted, connected tree called a *component tree*, where each level of the tree corresponds to a value of the threshold discussed in the previous paragraph. Each node of this tree represents an extremal region, R , defined as

$$\forall p \in R \text{ and } \forall q \in \text{boundary}(R) : I(p) > I(q) \quad (11-64)$$

Remember, \forall
 means “for any,” \in
 means “belonging to,”
 and a colon, :,
 is used to
 mean “it is true that.”

where I is the image under consideration, and p and q are image points. This equation indicates that an extremal region R is a region of I , with the property that the intensity of any point in the region is higher than the intensity at any point in the boundary of the region. As usual, we assume that image intensities are integers, ordered from 0 (black) to the maximum intensity (e.g., 255 for 8-bit images), which are represented by white.

MSERs are found by analyzing the nodes of the component tree. For each connected region in the tree, we compute a stability measure, ψ , defined as

$$\psi(R_j^{T+n\Delta T}) = \frac{|R_i^{T+(n-1)\Delta T}| - |R_k^{T+(n+1)\Delta T}|}{|R_j^{T+n\Delta T}|} \quad (11-65)$$

where $|R|$ is the size of the area (number of pixels) of connected region R , T is a threshold value in the range $T \in [\min(I), \max(I)]$, and ΔT is a specified threshold increment. Regions $R_i^{T+(n-1)\Delta T}$, $R_j^{T+n\Delta T}$, and $R_k^{T+(n+1)\Delta T}$ are connected regions obtained at threshold levels $T + (n-1)\Delta T$, $T + n\Delta T$, and $T + (n+1)\Delta T$, respectively. In terms of the component tree, regions R_i and R_k are respectively the *parent* and *child* of region R_j . Because $T + (n-1)\Delta T < T + (n+1)\Delta T$, we are guaranteed that $|R_i^{T+(n-1)\Delta T}| \geq |R_k^{T+(n+1)\Delta T}|$. It then follows from Eq. (11-65) that $\psi \geq 0$. MSREs are the regions corresponding to the nodes in the tree that have a stability value that is a *local minimum* along the path of the tree containing that region. What this means in practice is that maximally stable regions are regions whose sizes do not change appreciably across two, $2\Delta T$ neighboring thresholded images.

Figure 11.51 illustrates the concepts just introduced. The grayscale image at the top consists of some simple regions of constant intensity, with values in the range [0, 255]. Based on the explanation of Eqs. (11-64) and (11-65), we used the threshold $T = 10$, which is in the range $[\min(I) = 5, \max(I) = 225]$. Choosing $\Delta T = 50$ segmented all the different regions of the image. The column of binary images on the left contains the results of thresholding the grayscale image with the threshold values shown. The resulting component tree is on the right. Note that the tree is shown “root up,” which is the way you would normally program it.

All the squares in the grayscale image are of the same size (area); therefore, regardless of the image size, we can normalize the size of each square to 1. For example, if the image is of size 400×400 pixels, the size of each square is $100 \times 100 = 10^4$ pixels. Normalizing the size to 1 means that size 1 corresponds to 10^4 pixels (one square), size 2 corresponds to 2×10^4 pixels (two squares), and so forth. You can arrive at the same conclusion by noticing that the ratio in Eq. (11-65) eliminates the common 10^4 factor.

The component tree in Fig. 11.51 is a good summary of how the MSER algorithm works. The first level is the result of thresholding I with $T + \Delta T = 60$. There is only one connected component (white pixels) in the thresholded image on the left. The size of the connected component is 11 normalized units. As mentioned above, each node of a component tree, denoted by a subscripted R , contains *one* connected component consisting of white pixels. The next level in the tree is formed from the

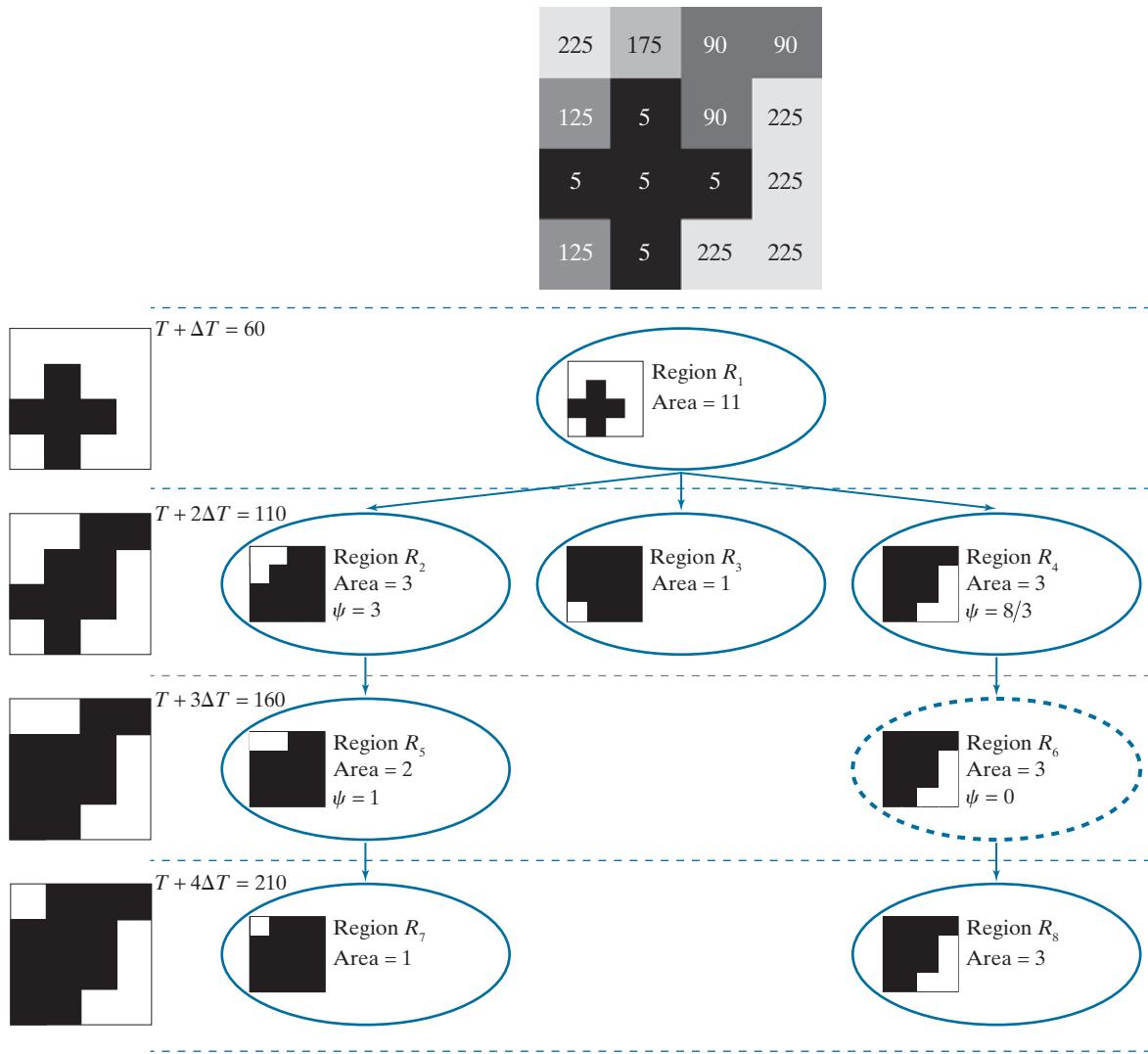


FIGURE 11.51 Detecting MSERs. Top: Grayscale image. Left: Thresholded images using $T = 10$ and $\Delta T = 50$. Right: Component tree, showing the individual regions. Only one MSER was detected (see dashed tree node on the rightmost branch of the tree). Each level of the tree is formed from the thresholded image on the left, at that same level. Each node of the tree contains one extremal region (connected component) shown in white, and denoted by a subscripted R .

regions in the binary image obtained by thresholding I using $T + 2\Delta T = 110$. As you can see on the left, this image has three connected components, so we create three nodes in the component tree at the level of the thresholded image. Similarly, the binary image obtained by thresholding I with $T + 3\Delta T = 160$ has two connected

components, so we create two nodes in the tree at this level. These two connected components are children of the connected components in the previous level, so we place the new nodes in the same path as their respective parents. The next level of the tree is explained in the same manner. Note that the center node in the previous level had no children, so that path of the tree ends in the second level.

Because we need to check size variations between parent and child regions to determine stability, only the two middle regions (corresponding to threshold values of 110 and 160) are relevant in this example. As you can see in our component tree, only R_6 has a parent and child of similar size (the sizes are identical in this case). Therefore, region R_6 is the only MSER detected in this case. Observe that if we had used a single global threshold to detect the brightest regions, region R_7 would have been detected also (an undesirable result in this context). Thus, we see that although MSERs are based on intensity, they also depend on the nature of the background surrounding a region. In this case, R_6 was surrounded by a darker background than R_7 , and the darker background was thresholded earlier in the tree, allowing the size of R_6 to remain constant over the two, $2\Delta T$ neighboring range required for detection as an MSER.

In our example, it was easy to detect an MSER as the only region that did not change size, which gave a stability factor 0. A value of zero automatically implies that an MSER has been found because the parent and child regions are of the same size. When working with more complex images, the values of stability factors seldom are zero because of variations in intensity caused by variables such as illumination, viewpoint, and noise. The concept of a local minimum mentioned earlier is simply a way of saying that MSERs are extremal regions that do change size significantly over a $2\Delta T$ thresholding range. What is considered a “significant” change depends on the application.

It is not unusual for numerous MSERs to be detected, many of which may not be meaningful because of their size. One way to control the number of regions detected is by the choice of ΔT . Another is to label as insignificant any region whose size is not in a specified size range. We illustrate this in Example 11.19.

Matas et al. [2002] indicate that MSERs are affine-covariant (see Section 11.1). This follows directly from the fact that area ratios are preserved under affine transformations, which in turn implies that for an affine transformation the original and transformed regions are related by that transformation. We illustrate this property in Figs. 11.54 and 11.55.

Finally, keep in mind that the preceding MSER formulation is designed to detect bright regions with darker surroundings. The same formulation applied to the negative (in the sense defined in Section 3.2) of an image will detect dark regions with lighter surroundings. If interest lies in detecting both types of regions simultaneously, we form the union of both sets of MSERs.

EXAMPLE 11.19: Extracting MSERs from grayscale images.

Figure 11.52(a) shows a slice image from a CT scan of a human head, and Fig. 11.52(b) shows the result of smoothing Fig. 11.52(a) with a box kernel of size 15×15 elements. Smoothing is used routinely as a

a	b
c	d

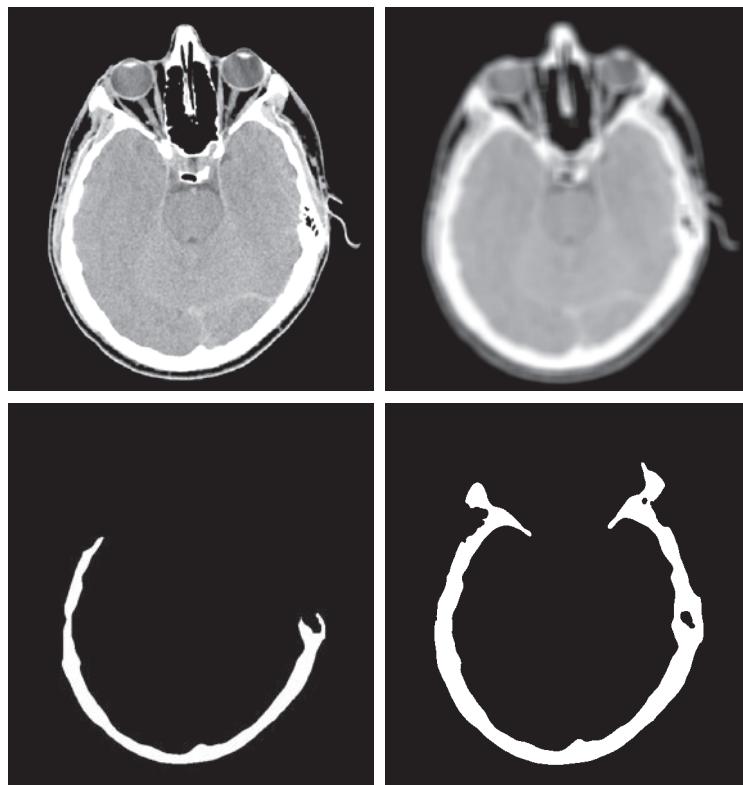
FIGURE 11.52

(a) 600×570 CT slice of a human head. (b) Image smoothed with a box kernel of size 15×15 elements. (c) A extremal region along the path of the tree containing one MSER.

(d) The MSER.

(All MSER regions were limited to the range 10,260–34,200 pixels, corresponding to a range between 3% and 10% of image size.)

(Original image courtesy of Dr. David R. Pickens, Vanderbilt University.)



preprocessing step when ΔT is relatively small. In this case, we used $T = 0$ and $\Delta T = 10$. This increment was small enough to require smoothing for proper MSER detection. In addition, we used a “size filter,” in the sense that the size (area) of an MSER had to be between 10,262 and 34,200 pixels; these size limits are 3% and 10% of the size of the image, respectively.

Figure 11.53 illustrates MSER detection on a more complex image. We used less blurring (a 5×5 box kernel) in this image because it has more fine detail. We used the same T and ΔT as in Fig. 11.52, and a valid MSER size in the range 10,000 to 30,000 pixels, corresponding approximately to 3% and 8% of image size, respectively. Two MSERs were detected using these parameters, as Figs. 11.53(c) and (d) show. The composite MSER, shown in Fig. 11.53(e), is a good representation of the front of the building.

Figure 11.54 shows the behavior under rotation of the MSERs detected in Fig. 11.53. Figure 11.54(a) is the building image rotated 5° in the counterclockwise direction. The image was cropped after rotation to eliminate the resulting black areas (see Fig. 2.41), which would change the nature of the image data and thus influence the results. Figure 11.54(b) is the result of performing the same smoothing as in Fig. 11.53, and Fig. 11.54(c) is the composite MSER detected using the same parameters as in Fig. 11.53(e). As you can see, the composite MSER of the rotated image corresponds quite closely to the MSER in Fig. 11.53(e).

Finally, Fig. 11.55 shows the behavior of the MSER detector under scale changes. Figure 11.55(a) is the building image scaled to 0.5 of its original dimensions, and Fig. 11.55(b) shows the image smoothed with a correspondingly smaller box kernel of size 3×3 . Because the image area is now one-fourth the size



FIGURE 11.53 (a) Building image of size 600×600 pixels. (b) Image smoothed using a 5×5 box kernel. (c) and (d) MSERs detected using $T = 0$, $\Delta T = 10$, and MSER size range between 10,000 and 30,000 pixels, corresponding approximately to 3% and 8% of the area of the image. (e) Composite image.

of the original area, we reduced the valid MSER range by one-fourth to 2500–7500 pixels. Other than these changes, we used the same parameters as in Fig. 11.53. Figure 11.55(c) shows the resulting MSER. As you can see, this figure is quite close to the full-size result in Fig. 11.53(e).

11.7 SCALE-INVARIANT FEATURE TRANSFORM (SIFT)

SIFT is an algorithm developed by Lowe [2004] for extracting invariant features from an image. It is called a *transform* because it transforms image data into scale-invariant coordinates relative to local image features. SIFT is by far the most complex feature detection and description approach we discuss in this chapter.

As you progress through this section, you will notice the use of a significant number of experimentally determined parameters. Thus, unlike most of the formulations of individual approaches we have discussed thus far, SIFT is strongly heuristic. This is a consequence of the fact that our current knowledge is insufficient to tell us how



FIGURE 11.54 (a) Building image rotated 5° counterclockwise. (b) Smoothed image using the same kernel as in Fig. 11.53(b). (c) Composite MSER detected using the same parameters we used to obtain Fig. 11.53(e). The MSERs of the original and rotated images are almost identical.

to assemble a set of reasonably well-understood individual methods into a “system” capable of addressing problems that cannot be solved by any single known method acting alone. Thus, we are forced to determine experimentally the interplay between the various parameters controlling the performance of more complex systems.

When images are similar in nature (same scale, similar orientation, etc), corner detection and MSERs are suitable as whole image features. However, in the presence of variables such as scale changes, rotation, changes in illumination, and changes in viewpoint, we are forced to use methods like SIFT.

SIFT features (called *keypoints*) are invariant to image scale and rotation, and are robust across a range of affine distortions, changes in 3-D viewpoint, noise, and changes of illumination. The input to SIFT is an image. Its output is an n -dimensional feature vector whose elements are the invariant feature descriptors. We begin our discussion by analyzing how scale invariance is achieved by SIFT.



FIGURE 11.55 (a) Building image reduced to half-size. (b) Image smoothed with a 3×3 box kernel. (c) Composite MSER obtained with the same parameters as Fig. 11.53(e), but using a valid MSER region size range of 2,500–7,500 pixels.

SCALE SPACE

The first stage of the SIFT algorithm is to find image locations that are invariant to scale change. This is achieved by searching for stable features across all possible scales, using a function of scale known as *scale space*, which is a multi-scale representation suitable for handling image structures at different scales in a consistent manner. The idea is to have a formalism for handling the fact that objects in unconstrained scenes will appear in different ways, depending on the scale at which images are captured. Because these scales may not be known beforehand, a reasonable approach is to work with all relevant scales simultaneously. Scale space represents an image as a one-parameter *family* of smoothed images, with the objective of simulating the loss of detail that would occur as the scale of an image decreases. The parameter controlling the smoothing is referred to as the *scale parameter*.

In SIFT, Gaussian kernels are used to implement smoothing, so the scale parameter is the standard deviation. The reason for using Gaussian kernels is based on work performed by Lindberg [1994], who showed that the only smoothing kernel that meets a set of important constraints, such as linearity and shift-invariance, is the Gaussian lowpass kernel. Based on this, the scale space, $L(x, y, \sigma)$, of a grayscale image, $f(x, y)$,[†] is produced by convolving f with a variable-scale Gaussian kernel, $G(x, y, \sigma)$:

As in Chapter 3, “★” indicates spatial convolution.

$$L(x, y, \sigma) = G(x, y, \sigma) \star f(x, y) \quad (11-66)$$

where the scale is controlled by parameter σ , and G is of the form

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2 + y^2)/2\sigma^2} \quad (11-67)$$

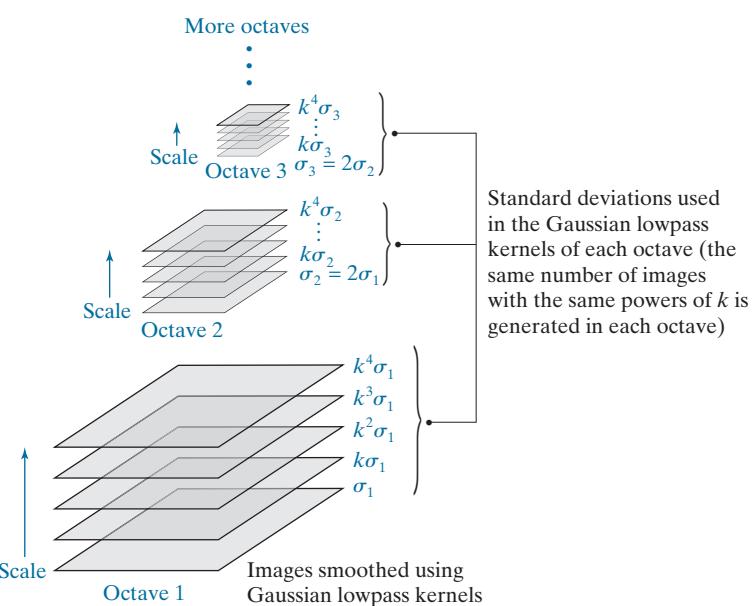
The input image $f(x, y)$ is successively convolved with Gaussian kernels having standard deviations $\sigma, k\sigma, k^2\sigma, k^3\sigma, \dots$ to generate a “stack” of Gaussian-filtered (smoothed) images that are separated by a constant factor k , as shown in the lower left of Fig. 11.56.

SIFT subdivides scale space into *octaves*, with each octave corresponding to a doubling of σ , just as an octave in music theory corresponds to doubling the frequency of a sound signal. SIFT further subdivides each octave into an integer number, s , of intervals, so that an interval of 1 consists of two images, an interval of 2 consists of three images, and so forth. It then follows that the value used in the Gaussian kernel that generates the image corresponding to an octave is $k^s\sigma = 2\sigma$ which means that $k = 2^{1/s}$. For example, for $s = 2$, $k = \sqrt{2}$, and the input image is successively smoothed using standard deviations of $\sigma, (\sqrt{2})\sigma$, and $(\sqrt{2})^2\sigma$, so that the third image (i.e., the octave image for $s = 2$) in the sequence is filtered using a Gaussian kernel with standard deviation $(\sqrt{2})^2\sigma = 2\sigma$.

[†]Experimental results reported by Lowe [2004] suggest that smoothing the original image using a Gaussian kernel with $\sigma = 0.5$ and then doubling its size by linear (nearest-neighbor) interpolation improves the number of stable features detected by SIFT. This preprocessing step is an integral part of the algorithm. Images are assumed to have values in the range $[0, 1]$.

FIGURE 11.56

Scale space, showing three octaves. Because $s = 2$ in this case, each octave has five smoothed images. A Gaussian kernel was used for smoothing, so the space parameter is σ .



The preceding discussion indicates that the number of smoothed images generated in an octave is $s + 1$. However, as you will see in the next section, the smoothed images in scale space are used to compute differences of Gaussians [see Eq. (10-32)] which, in order to cover a full octave, implies that an additional two images past the octave image are required, giving a total of $s + 3$ images. Because the octave image is always the $(s + 1)$ th image in the stack (counting from the bottom), it follows that this image is the third image from the top in the expanded sequence of $s + 3$ images. Each octave in Fig. 11.56 contains five images, indicating that $s = 2$ was used in this case.

The *first* image in the *second* octave is formed by downsampling the original image (by skipping every other row and column), and then smoothing it using a kernel with twice the standard deviation used in the first octave (i.e., $\sigma_2 = 2\sigma_1$). Subsequent images in that octave are smoothed using σ_2 , with the same sequence of values of k as in the first octave (this is denoted by dots in Fig. 11.56). The same basic procedure is then repeated for subsequent octaves. That is, the *first* image of the *new* octave is formed by: (1) downsampling the original image enough times to achieve half the size of the image in the previous octave, and (2) smoothing the downsampled image with a new standard deviation that is twice the standard deviation of the previous octave. The rest of the images in the new octave are obtained by smoothing the downsampled image with the new standard deviation multiplied by the same sequence of values of k as before.

When $k = \sqrt{2}$, we can obtain the first image of a new octave without having to smooth the downsampled image. This is because, for this value of k , the kernel used to smooth the first image of every octave is the same as the kernel used to smooth

Instead of repeatedly downsampling the original image, we can carry the previously downsampled image, and downsample it by 2 to obtain the image required for the next octave.

the *third* image from the top of the previous octave. Thus, the first image of a new octave can be obtained directly by downsampling that third image of the previous octave by 2. The result will be the same (see Problem 11.36). The third image from the top of any octave is called the *octave image* because the standard deviation used to smooth it is twice (i.e., $k^2 = 2$) the value of the standard deviation used to smooth the first image in the octave.

Figure 11.57 uses grayscale images to further illustrate how scale space is constructed in SIFT. Because each octave is composed of five images, it follows that we are again using $s = 2$. We chose $\sigma_1 = \sqrt{2}/2 = 0.707$ and $k = \sqrt{2} = 1.414$ for this example so that the numbers would result in familiar multiples. As in Fig. 11.56, the images going up scale space are blurred by using Gaussian kernels with progressively larger standard deviations, and the first image of the second and subsequent octaves is obtained by downsampling the octave image from the previous octave by 2. As you can see, the images become significantly more blurred (and consequently lose more fine detail) as they go up both in scale as well as in octave. The images in the third octave show significantly fewer details, but their gross appearance is unmistakably that of the same structure.

DETECTING LOCAL EXTREMA

SIFT initially finds the locations of keypoints using the Gaussian filtered images, then refines the locations and validity of those keypoints using two processing steps.

Finding the Initial Keypoints

Keypoint locations in scale space are found initially by SIFT by detecting extrema in the difference of Gaussians of two adjacent scale-space images in an octave, convolved with the input image that corresponds to that octave. For example, to find keypoint locations related to the first two levels of octave 1 in scale space, we look for extrema in the function

$$D(x, y, \sigma) = [G(x, y, k\sigma) - G(x, y, \sigma)] \star f(x, y) \quad (11-68)$$

It follows from Eq. (11-66) that

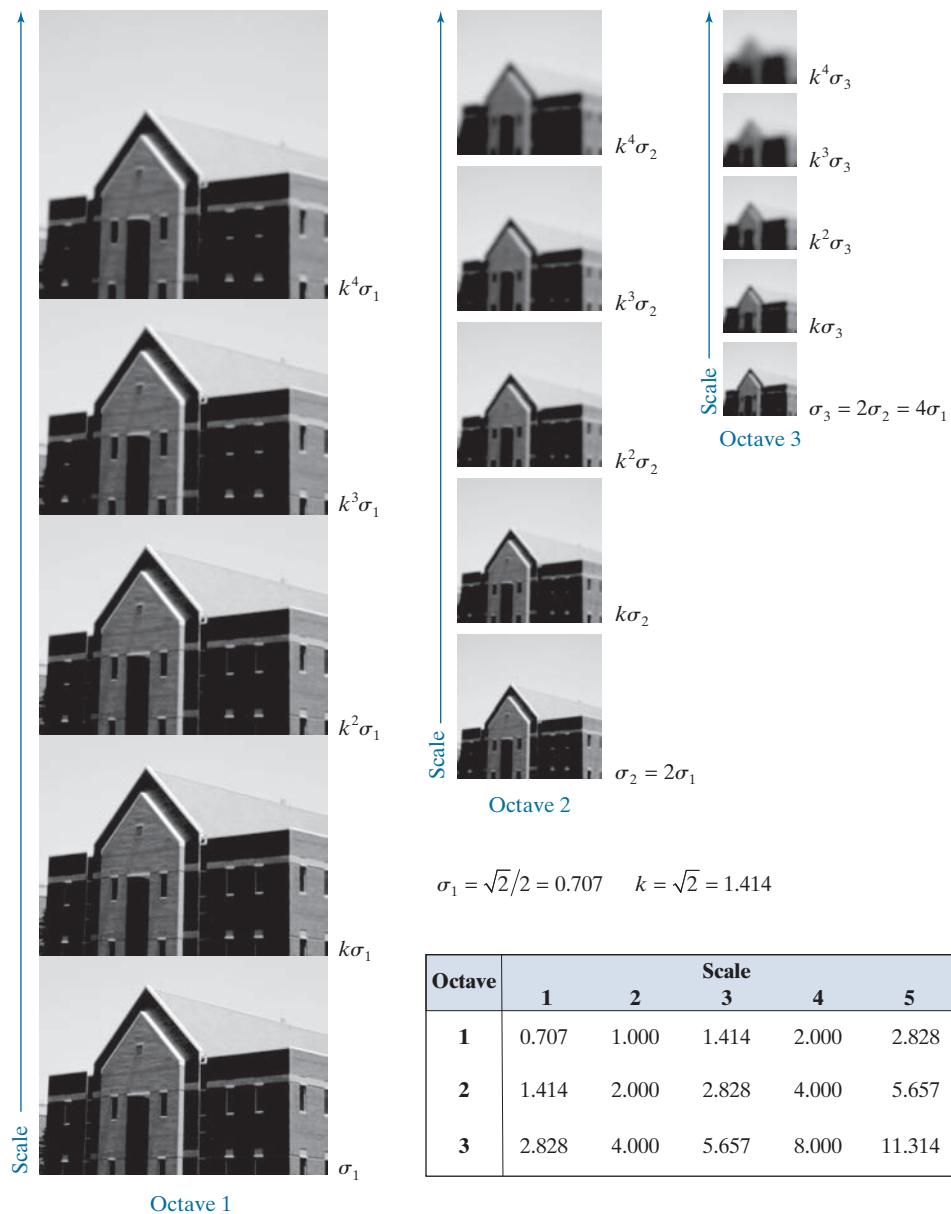
$$D(x, y, \sigma) = L(x, y, k\sigma) - L(x, y, \sigma) \quad (11-69)$$

In other words, all we have to do to form function $D(x, y, \sigma)$ is subtract the first two images of octave 1. Recall from the discussion of the Marr-Hildreth edge detector (Section 10.2) that the difference of Gaussians is an approximation to the Laplacian of a Gaussian (LoG). Therefore, Eq. (11-69) is nothing more than an approximation to Eq. (10-30). The key difference is that SIFT looks for extrema in $D(x, y, \sigma)$, whereas the Marr-Hildreth detector would look for the zero crossings of this function.

Lindberg [1994] showed that true scale invariance in scale space requires that the LoG be normalized by σ^2 (i.e., that $\sigma^2 \nabla^2 G$ be used). It can be shown (see Problem 11.34) that

FIGURE 11.57

Illustration using images of the first three octaves of scale space in SIFT. The entries in the table are values of standard deviation used at each scale of each octave. For example the standard deviation used in scale 2 of octave 1 is $k\sigma_1$, which is equal to 1.0. (The images of octave 1 are shown slightly overlapped to fit in the figure space.)



$$G(x, y, k\sigma) - G(x, y, \sigma) \approx (k - 1)\sigma^2 \nabla^2 G \quad (11-70)$$

Therefore, DoGs already have the necessary scaling “built in.” The factor $(k - 1)$ is constant over all scales, so it does not influence the process of locating extrema in scale space. Although Eqs. (11-68) and (11-69) are applicable to the first two images

of octave 1, the same form of these equations is applicable to any two images from any octave, provided that the appropriate downsampled image is used, and the DoG is computed from two adjacent images in the octave.

Figure 11.58 illustrates the concepts just discussed, using the building image from Fig. 11.57. A total of $s + 2$ difference functions, $D(x, y, \sigma)$, are formed in each octave from all adjacent pairs of Gaussian-filtered images in that octave. These difference functions can be viewed as images, and one sample of such an image is shown for each of the three octaves in Fig. 11.58. As you might expect from the results in Fig. 11.57, the level of detail in these images decreases the further up we go in scale space.

Figure 11.59 shows the procedure used by SIFT to find extrema in a $D(x, y, \sigma)$ image. At each location (shown in black) in a $D(x, y, \sigma)$ image, the value of the pixel at that location is compared to the values of its eight neighbors in the current image and its nine neighbors in the images above and below. The point is selected as an extremum (maximum or minimum) point if its value is larger than the values of all its neighbors, or smaller than all of them. No extrema can be detected in the first (last) scale of an octave because it has no lower (upper) scale image of the same size.

Improving the Accuracy of Keypoint Locations

When a continuous function is sampled, its true maximum or minimum may actually be located between sample points. The usual approach used to get closer to the true

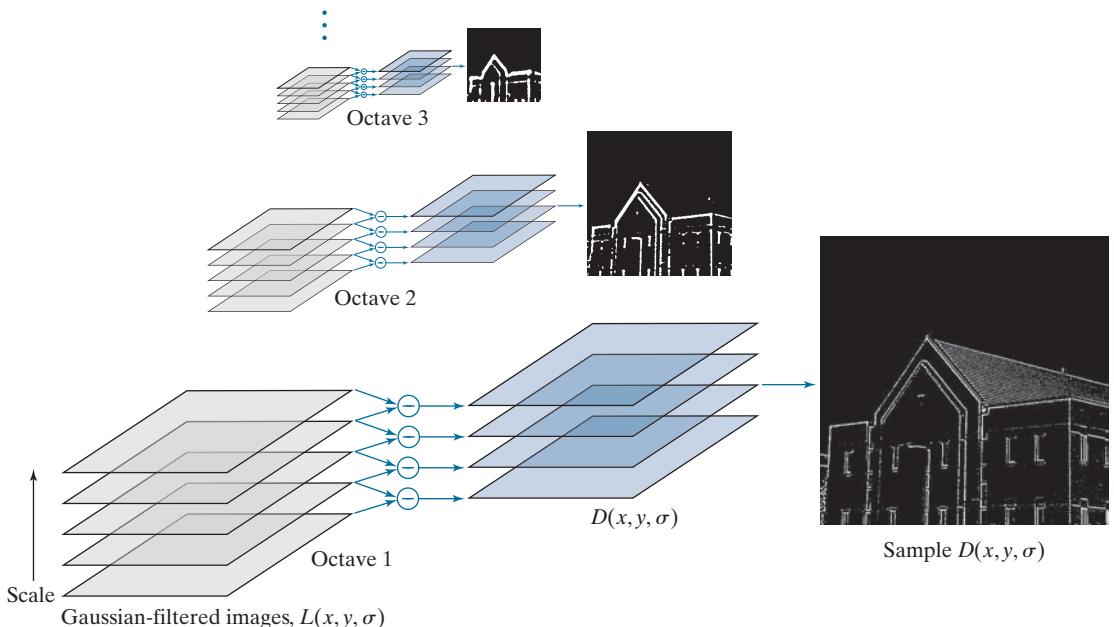
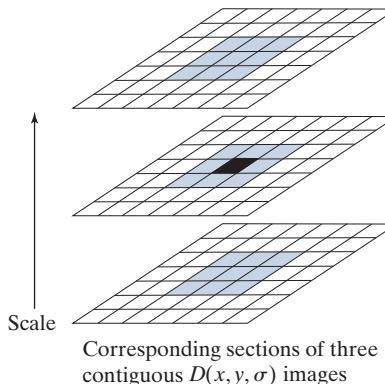


FIGURE 11.58 How Eq. (11-69) is implemented in scale space. There are $s + 3$ $L(x, y, \sigma)$ images and $s + 2$ corresponding $D(x, y, \sigma)$ images in each octave.

FIGURE 11.59

Extrema (maxima or minima) of the $D(x, y, \sigma)$ images in an octave are detected by comparing a pixel (shown in black) to its 26 neighbors (shown shaded) in 3×3 regions at the current and adjacent scale images.



extremum (to achieve subpixel accuracy) is to fit an interpolating function at each extremum point found in the digital function, then look for an improved extremum location in the interpolated function. SIFT uses the linear and quadratic terms of a Taylor series expansion of $D(x, y, \sigma)$, shifted so that the origin is located at the sample point being examined. In vector form, the expression is

$$\begin{aligned} D(\mathbf{x}) &= D + \left(\frac{\partial D}{\partial \mathbf{x}} \right)^T \mathbf{x} + \frac{1}{2} \mathbf{x}^T \frac{\partial}{\partial \mathbf{x}} \left(\frac{\partial D}{\partial \mathbf{x}} \right) \mathbf{x} \\ &= D + (\nabla D)^T \mathbf{x} + \frac{1}{2} \mathbf{x}^T \mathbf{H} \mathbf{x} \end{aligned} \quad (11-71)$$

where D and its derivatives are evaluated at the sample point, $\mathbf{x} = (x, y, \sigma)^T$ is the offset from that point, ∇ is the familiar gradient operator,

$$\nabla D = \frac{\partial D}{\partial \mathbf{x}} = \begin{bmatrix} \partial D / \partial x \\ \partial D / \partial y \\ \partial D / \partial \sigma \end{bmatrix} \quad (11-72)$$

and \mathbf{H} is the *Hessian matrix*

$$\mathbf{H} = \begin{bmatrix} \partial^2 D / \partial x^2 & \partial^2 D / \partial x \partial y & \partial^2 D / \partial x \partial \sigma \\ \partial^2 D / \partial y \partial x & \partial^2 D / \partial y^2 & \partial^2 D / \partial y \partial \sigma \\ \partial^2 D / \partial \sigma \partial x & \partial^2 D / \partial \sigma \partial y & \partial^2 D / \partial \sigma^2 \end{bmatrix} \quad (11-73)$$

The location of the extremum, $\hat{\mathbf{x}}$, is found by taking the derivative of Eq. (11-71) with respect to \mathbf{x} and setting it to zero, which gives us (see Problem 11.37):

Because D and its derivatives are evaluated at the sample point, they are constants with respect to \mathbf{x} .

$$\hat{\mathbf{x}} = -\mathbf{H}^{-1}(\nabla D) \quad (11-74)$$

The Hessian and gradient of D are approximated using differences of neighboring points, as we did in Section 10.2. The resulting 3×3 system of linear equations is easily solved computationally. If the offset $\hat{\mathbf{x}}$ is greater than 0.5 in any of its three dimensions, we conclude that the extremum lies closer to another sample point, in which case the sample point is changed and the interpolation is performed about that point instead. The final offset $\hat{\mathbf{x}}$ is added to the location of its sample point to obtain the interpolated estimate of the location of the extremum.

The function value at the extremum, $D(\hat{\mathbf{x}})$, is used by SIFT for rejecting unstable extrema with low contrast, where $D(\hat{\mathbf{x}})$ is obtained by substituting Eq. (11-74) into Eq. (11-71), giving (see Problem 11.37):

$$D(\hat{\mathbf{x}}) = D + \frac{1}{2} (\nabla D)^T \hat{\mathbf{x}} \quad (11-75)$$

In the experimental results reported by Lowe [2004], any extrema for which $D(\hat{\mathbf{x}})$ was less than 0.03 was rejected, based on all image values being in the range $[0, 1]$. This eliminates keypoints that have low contrast and/or are poorly localized.

Eliminating Edge Responses

Recall from Section 10.2 that using a difference of Gaussians yields edges in an image. But keypoints of interest in SIFT are “corner-like” features, which are significantly more localized. Thus, intensity transitions caused by edges are eliminated. To quantify the difference between edges and corners, we can look at local curvature. An edge is characterized by high curvature in one direction, and low curvature in the orthogonal direction. Curvature at a point in an image can be estimated from the 2×2 Hessian matrix evaluated at that point. Thus, to estimate local curvature of the DoG at any level in scalar space, we compute the Hessian matrix of D at that level:

$$\mathbf{H} = \begin{bmatrix} \partial^2 D / \partial x^2 & \partial^2 D / \partial x \partial y \\ \partial^2 D / \partial y \partial x & \partial^2 D / \partial y^2 \end{bmatrix} = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{yx} & D_{yy} \end{bmatrix} \quad (11-76)$$

where the form on the right uses the same notation as the \mathbf{A} term [Eq. (11-61)] of the Harris matrix (but note that the main diagonals are different). The eigenvalues of \mathbf{H} are proportional to the curvatures of D . As we explained in connection with the Harris-Stephens corner detector, we can avoid direct computation of the eigenvalues by formulating tests based on the trace and determinant of \mathbf{H} , which are equal to the sum and product of the eigenvalues, respectively. To use notation different from the HS discussion, let α and β be the eigenvalues of \mathbf{H} with the largest and smallest magnitude, respectively. Using the relationship between the eigenvalues of \mathbf{H} and its trace and determinant we have (remember, \mathbf{H} is symmetric and of size 2×2):

$$\begin{aligned} \text{Tr}(\mathbf{H}) &= D_{xx} + D_{yy} = \alpha + \beta \\ \text{Det}(\mathbf{H}) &= D_{xx}D_{yy} - (D_{xy})^2 = \alpha\beta \end{aligned} \quad (11-77)$$

If you display an image as a topographic map (see Fig. 2.18), edges will appear as ridges that have low curvature along the ridge and high curvature perpendicular to it.

As with the HS corner detector, the advantage of this formulation is that the trace and determinants of 2×2 matrix \mathbf{H} are easy to compute. See the margin note in Eq. (11-63).

If the determinant is negative, the curvatures have different signs and the keypoint in question cannot be an extremum, so it is discarded.

Let r denote the ratio of the largest to the smallest eigenvalue. Then $\alpha = r\beta$ and

$$\frac{[\text{Tr}(\mathbf{H})]^2}{\text{Det}(\mathbf{H})} = \frac{(\alpha + \beta)^2}{\alpha\beta} = \frac{(r\beta + \beta)^2}{r\beta^2} = \frac{(r + 1)^2}{r} \quad (11-78)$$

which depends on the ratio of the eigenvalues, rather than their individual values. The minimum of $(r + 1)^2/r$ occurs when the eigenvalues are equal, and it increases with r . Therefore, to check that the ratio of principal curvatures is below some threshold, r , we only need to check

$$\frac{[\text{Tr}(\mathbf{H})]^2}{\text{Det}(\mathbf{H})} < \frac{(r + 1)^2}{r} \quad (11-79)$$

which is a simple computation. In the experimental results reported by Lowe [2004], a value of $r = 10$ was used, meaning that keypoints with ratios of curvature greater than 10 were eliminated.

Figure 11.60 shows the SIFT keypoints detected in the building image using the approach discussed in this section. Keypoints for which $D(\hat{\mathbf{x}})$ in Eq. (11-75) was less than 0.03 were rejected, as were keypoints that failed to satisfy Eq. (11-79) with $r = 10$.

KEYPOINT ORIENTATION

At this point in the process, we have computed keypoints that SIFT considers stable. Because we know the location of each keypoint in scale space, we have achieved *scale independence*. The next step is to assign a consistent orientation to each keypoint based on local image properties. This allows us to represent a keypoint relative to its orientation and thus achieve *invariance to image rotation*. SIFT uses a

FIGURE 11.60

SIFT keypoints detected in the building image. The points were enlarged slightly to make them easier to see.



See Section 10.2 regarding computation of the gradient magnitude and angle.

straightforward approach for this. The scale of the keypoint is used to select the Gaussian smoothed image, L , that is closest to that scale. In this way, all orientation computations are performed in a scale-invariant manner. Then, for each image sample, $L(x, y)$, at this scale, we compute the gradient magnitude, $M(x, y)$, and orientation angle, $\theta(x, y)$, using pixel differences:

$$M(x, y) = \left[(L(x+1, y) - L(x-1, y))^2 + (L(x, y+1) - L(x, y-1))^2 \right]^{\frac{1}{2}} \quad (11-80)$$

and

$$\theta(x, y) = \tan^{-1} \left[(L(x, y+1) - L(x, y-1)) / (L(x+1, y) - L(x-1, y)) \right] \quad (11-81)$$

A histogram of orientations is formed from the gradient orientations of sample points in a neighborhood of each keypoint. The histogram has 36 bins covering the 360° range of orientations on the image plane. Each sample added to the histogram is weighed by its gradient magnitude, and by a circular Gaussian function with a standard deviation 1.5 times the scale of the keypoint.

Peaks in the histogram correspond to dominant local directions of local gradients. The highest peak in the histogram is detected and any other local peak that is within 80% of the highest peak is used also to create another keypoint with that orientation. Thus, for the locations with multiple peaks of similar magnitude, there will be multiple keypoints created at the same location and scale, but with *different* orientations. SIFT assigns multiple orientations to only about 15% of points with multiple orientations, but these contribute significant to image matching (to be discussed later and in Chapter 12). Finally, a parabola is fit to the three histogram values closest to each peak to interpolate the peak position for better accuracy.

Figure 11.61 shows the same keypoints as Fig. 11.60 superimposed on the image and showing keypoint orientations as arrows. Note the consistency of orientation

FIGURE 11.61

The keypoints from Fig. 11.60 superimposed on the original image. The arrows indicate keypoint orientations.



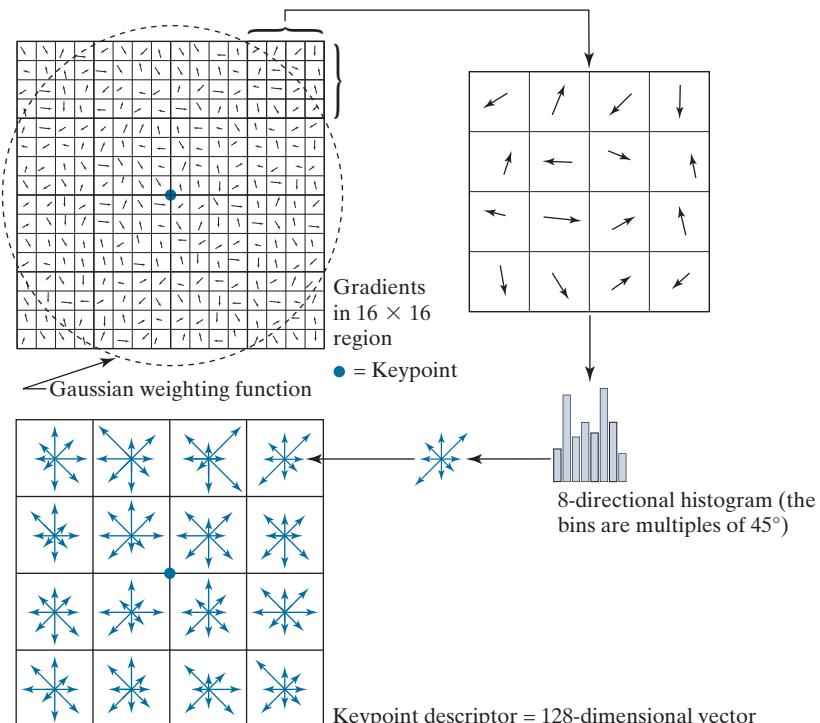
of similar sets of keypoints in the image. For example, observe the keypoints on the right, vertical corner of the building. The lengths of the arrows vary, depending on illumination and image content, but their direction is unmistakably consistent. Plots of keypoint orientations generally are quite cluttered and are not intended for general human interpretation. The value of keypoint orientation is in image matching, as we will illustrate later in our discussion.

KEYPOINT DESCRIPTORS

The procedures discussed up to this point are used for assigning an image location, scale, and orientation to each keypoint, thus providing invariance to these three variables. The next step is to compute a descriptor for a local region around each keypoint that is highly distinctive, but is at the same time as invariant as possible to changes in scale, orientation, illumination, and image viewpoint. The idea is to be able to use these descriptors to identify matches (similarities) between local regions in two or more images.

The approach used by SIFT to compute descriptors is based on experimental results suggesting that local image gradients appear to perform a function similar to what human vision does for matching and recognizing 3-D objects from different viewpoints (Lowe [2004]). Figure 11.62 summarizes the procedure used by SIFT to generate the descriptors associated with each keypoint. A region of size 16×16

FIGURE 11.62
Approach used to compute a keypoint descriptor.



pixels is centered on a keypoint, and the gradient magnitude and direction are computed at each point in the region using pixel differences. These are shown as randomly oriented arrows in the upper-left of the figure. A Gaussian weighting function with standard deviation equal to one-half the size of the region is then used to assign a weight that multiplies the magnitude of the gradient at each point. The Gaussian weighting function is shown as a circle in the figure, but it is understood that it is a bell-shaped surface whose values (weights) decrease as a function of distance from the center. The purpose of this function is to reduce sudden changes in the descriptor with small changes in the position of the function.

Because there is one gradient computation for each point in the region surrounding a keypoint, there are $(16)^2$ gradient directions to process for each keypoint. There are 16 directions in each 4×4 subregion. The top-rightmost subregion is shown zoomed in the figure to simplify the explanation of the next step, which consists of quantizing all gradient orientations in the 4×4 subregion into eight possible directions differing by 45° . Rather than assigning a directional value as a full count to the bin to which it is closest, SIFT performs interpolation that *distributes* a histogram entry among *all* bins proportionally, depending on the distance from that value to the center of each bin. This is done by multiplying each entry into a bin by a weight of $1 - d$, where d is the shortest distance from the value to the center of a bin, measured in the units of the histogram spacing, so that the maximum possible distance is 1. For example, the center of the first bin is at $45^\circ/2 = 22.5^\circ$, the next center is at $22.5^\circ + 45^\circ = 67.5^\circ$, and so on. Suppose that a particular directional value is 22.5° . The distance from that value to the center of the first histogram bin is 0, so we would assign a full entry (i.e., a count of 1) to that bin in the histogram. The distance to the next center would be greater than 0, so we would assign a fraction of a full entry, that is $1 * (1 - d)$, to that bin, and so forth for all bins. In this way, every bin gets a proportional fraction of a count, thus avoiding “boundary” effects in which a descriptor changes abruptly as a small change in orientation causes it to be assigned from one bin to another.

Figure 11.62 shows the eight directions of a histogram as a small cluster of vectors, with the length of each vector being equal to the value of its corresponding bin. Sixteen histograms are computed, one for each 4×4 subregion of the 16×16 region surrounding a keypoint. A descriptor, shown on the lower left of the figure, then consists of a 4×4 array, each containing eight directional values. In SIFT, this descriptor data is organized as a 128-dimensional vector.

In order to achieve orientation invariance, the coordinates of the descriptor and the gradient orientations are rotated relative to the keypoint orientation. In order to reduce the effects of illumination, a feature vector is normalized in two stages. First, the vector is normalized to unit length by dividing each component by the vector norm. A change in image contrast resulting from each pixel value being multiplied by a constant will multiply the gradients by the same constant, so the change in contrast will be cancelled by the first normalization. A brightness change caused by a constant being added to each pixel will not affect the gradient values because they are computed from pixel differences. Therefore, the descriptor is invariant to affine changes in illumination. However, nonlinear illumination changes resulting, for example, from camera saturation, can also occur. These types of changes can cause large variations in the relative magnitudes of some of the gradients, but they

are less likely to affect gradient orientation. SIFT reduces the influence of large gradient magnitudes by thresholding the values of the normalized feature vector so that all components are below the experimentally determined value of 0.2. After thresholding, the feature vector is renormalized to unit length.

SUMMARY OF THE SIFT ALGORITHM

As the material in the preceding sections shows, SIFT is a complex procedure consisting of many parts and empirically determined constants. The following is a step-by-step summary of the method.

As indicated at the beginning of this section, smoothing and doubling the size of the input image is assumed. Input images are assumed to have values in the range [0, 1].

- 1. Construct the scale space.** This is done using the procedure outlined in Figs. 11.56 and 11.57. The parameters that need to be specified are σ , s , (k is computed from s), and the number of octaves. Suggested values are $\sigma = 1.6$, $s = 2$, and three octaves.
- 2. Obtain the initial keypoints.** Compute the difference of Gaussians, $D(x, y, \sigma)$, from the smoothed images in scale space, as explained in Fig. 11.58 and Eq. (11-69). Find the extrema in each $D(x, y, \sigma)$ image using the method explained in Fig. 11.59. These are the initial keypoints.
- 3. Improve the accuracy of the location of the keypoints.** Interpolate the values of $D(x, y, \sigma)$ via a Taylor expansion. The improved key point locations are given by Eq. (11-74).
- 4. Delete unsuitable keypoints.** Eliminate keypoints that have low contrast and/or are poorly localized. This is done by evaluating D from Step 3 at the improved locations, using Eq. (11-75). All keypoints whose values of D are lower than a threshold are deleted. A suggested threshold value is 0.03. Keypoints associated with edges are deleted also, using Eq. (11-79). A value of 10 is suggested for r .
- 5. Compute keypoint orientations.** Use Eqs. (11-80) and (11-81) to compute the magnitude and orientation of each keypoint using the histogram-based procedure discussed in connection with these equations.
- 6. Compute keypoint descriptors.** Use the method summarized in Fig. 11.62 to compute a feature (descriptor) vector for each keypoint. If a region of size 16×16 around each keypoint is used, the result will be a 128-dimensional feature vector for each keypoint.

The following example illustrates the power of this algorithm.

EXAMPLE 11.20: Using SIFT for image matching.

We illustrate the performance of the SIFT algorithm by using it to find the number of matches between an image of a building and a subimage formed by extracting part of the right corner edge of the building. We also show results for rotated and scaled-down versions of the image and subimage. This type of process can be used in applications such as finding correspondences between two images for the purpose of image registration, and for finding instances of an image in a database of images.

Figure 11.63(a) shows the keypoints for the building image (this is the same as Fig. 11.61), and the keypoints for the subimage, which is a separate, much smaller image. The keypoints were computed

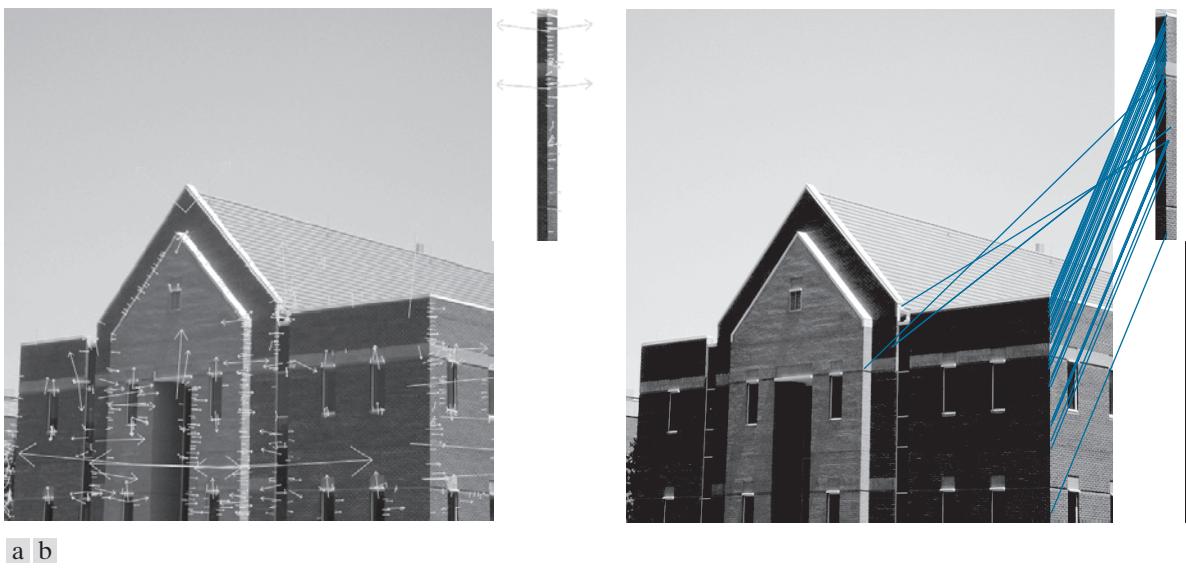


FIGURE 11.63 (a) Keypoints and their directions (shown as gray arrows) for the building image and for a section of the right corner of the building. The subimage is a separate image and was processed as such. (b) Corresponding key points between the building and the subimage (the straight lines shown connect pairs of matching points). Only three of the 36 matches found are incorrect.

using SIFT *independently* for each image. The building shows 643 keypoints and the subimage 54 keypoints. Figure 11.63(b) shows the matches found by SIFT between the image and subimage; 36 keypoint matches were found and, as the figure shows, only three were incorrect. Considering the large number of initial keypoints, you can see that keypoint descriptors offer a high degree of accuracy for establishing correspondences between images.

Figure 11.64(a) shows keypoints for the building image after it was rotated by 5° counterclockwise, and for a subimage extracted from its right corner edge. The rotated image is smaller than the original because it was cropped to eliminate the constant areas created by rotation (see Fig. 2.41). Here, SIFT found 547 keypoints for the building and 49 for the subimage. A total of 26 matches were found and, as Fig. 11.64(b) shows, only two were incorrect.

Figure 11.65 shows the results obtained using SIFT on an image of the building reduced to half the size in both spatial directions. When SIFT was applied to the downsampled image and a corresponding subimage, no matches were found. This was remedied by brightening the reduced image slightly by manipulating the intensity gamma. The subimage was extracted from this image. Despite the fact that SIFT has the capability to handle some degree of changes in intensity, this example indicates that performance can be improved by enhancing the contrast of an image prior to processing. When working with a database of images, histogram specification (see Chapter 3) is an excellent tool for normalizing the intensity of all images using the characteristics of the image being queried. SIFT found 195 keypoints for the half-size image and 24 keypoints for the corresponding subimage. A total of seven matches were found between the two images, of which only one was incorrect.

The preceding two figures illustrate the insensitivity of SIFT to rotation and scale changes, but they are not ideal tests because the reason for seeking insensitivity to these variables in the first place is

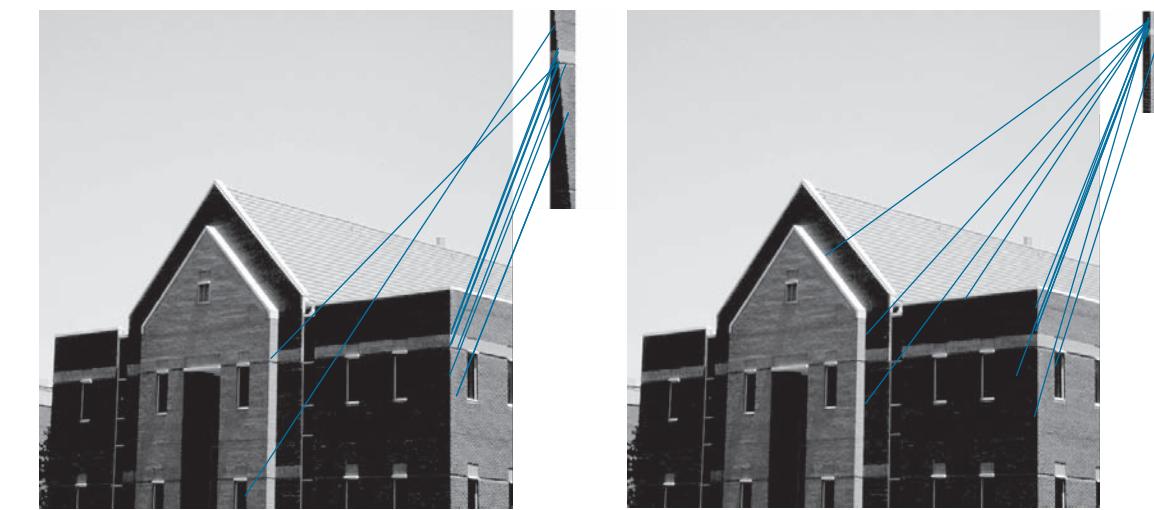


FIGURE 11.64 (a) Keypoints for the rotated (by 5°) building image and for a section of the right corner of the building. The subimage is a separate image and was processed as such. (b) Corresponding keypoints between the corner and the building. Of the 26 matches found, only two are in error.

that we do not always know *a priori* when images have been acquired under different conditions and geometrical arrangements. A more practical test is to compute features for a prototype image and test them against unknown samples. Figure 11.66 shows the results of such tests. Figure 11.66(a) is the original building image, for which SIFT features vectors were already computed (see Fig. 11.63). SIFT was used to compare the rotated subimage from Fig. 11.64(a) against the original, unrotated image. As Fig. 11.66(a) shows, 10 matches were found, of which two were incorrect. These are excellent results, considering the relatively small size of the subimage, and the fact that it was rotated. Figure 11.66(b) shows the results of matching the half-sized subimage against the original image. Eleven matches were found,



FIGURE 11.65 (a) Keypoints for the half-sized building and a section of the right corner. (b) Corresponding keypoints between the corner and the building. Of the seven matches found, only one is in error.



a b

FIGURE 11.66 (a) Matches between the original building image and a rotated version of a segment of its right corner. Ten matches were found, of which two are incorrect. (b) Matches between the original image and a half-scaled version of a segment of its right corner. Here, 11 matches were found, of which four were incorrect.

of which four were incorrect. Again, these are good results, considering the fact that significant detail was lost in the subimage when it was rotated or reduced in size. If asked in both cases: Based solely on the matches found by SIFT, from which part of the building did the two subimages come? The obvious answer in both is that the subimages are from the right corner of the building. The preceding two tests illustrate the adaptability of SIFT to variations in rotation and scale.

Summary, References, and Further Reading

Feature extraction is a fundamental process in the operation of most automated image processing applications. As indicated by the range of feature detection and description techniques covered in this chapter, the choice of one method over another is determined by the problem under consideration. The objective is to choose feature descriptors that “capture” essential differences between objects, or classes of objects, while maintaining as much independence as possible to changes in variables such as location, scale, orientation, illumination, and viewing angle.

The Freeman chain code discussed in Section 11.2 was first proposed by Freeman [1961, 1974], while the slope chain code is due to Bribiesca [2013]. See Klette and Rosenfeld [2004] regarding the minimum-perimeter polygon algorithm. For additional reading on signatures see Ballard and Brown [1982]. The medial axis transform is generally credited to Blum [1967]. For efficient computation of the Euclidean distance transform used for skeletonizing see Maurer et al. [2003].

For additional reading on the basic boundary feature descriptors in Section 11.3, see Rosenfeld and Kak [1982]. The discussion on shape numbers is based on the work of Bribiesca and Guzman [1980]. For additional reading on Fourier descriptors, see the early paper by Zahn and Roskies [1972]. For an example of current uses of this technique, see Sikic and Konjicila [2016]. The discussion on statistical moments as boundary descriptors is from basic probability (for example, see Montgomery and Runger [2011]).

For additional reading on the basic region descriptors discussed in Section 11.4, see Rosenfeld and Kak [1982]. For further introductory reading on texture, see Haralick and Shapiro [1992] and Shapiro and Stockman [2001].

Our discussion of moment-invariants is based on Hu [1962]. For generating moments of arbitrary order, see Flusser [2000].

Hotelling [1933] was the first to derive and publish the approach that transforms discrete variables into uncorrelated coefficients (Section 11.5). He referred to this technique as the method of *principal components*. His paper gives considerable insight into the method and is worth reading. Principal components are still used widely in numerous fields, including image processing, as evidenced by Xiang et al. [2016]. The corner detector in Section 11.6 is from Harris and Stephens [1988], and our discussion of MSERs is based on Matas et al. [2002]. The SIFT material in Section 11.7 is from Lowe [2004]. For details on the software aspects of many of the examples in this chapter, see Gonzalez, Woods, and Eddins [2009].

Problems

Solutions to the problems marked with an asterisk (*) are in the DIP4E Student Support Package (consult the book website: www.ImageProcessingPlace.com).

11.1 Do the following:

- (a)* Provide all the missing steps in Fig. 11.1. Show your results using the same format as in that figure.
- (b) When applied to binary regions, the boundary-following algorithm in Section 11.2 typically yields boundaries that are one pixel thick, but this is not always the case. Give a small image example in which the boundary is thicker than one pixel in at least one place.

11.2 With reference to the Moore boundary-following algorithm explained in Section 11.2, answer the following, using the same grid as in Fig. 11.2 to identify boundary points in your explanation [remember, the origin is at (1,1), instead of our usual (0,0)]. Include the position of points *b* and *c* at each point you mention.

- (a)* Give the coordinates in Fig. 11.2(a) at which the algorithm starts and ends. What would it do when it arrived at the end point of the boundary?
- (b) How would the algorithm behave when it arrived at the intersection point in Fig. 11.2(b) for the first time, and then for the second time?

11.3 Answer the following:

- (a)* Does normalizing the Freeman chain code of a closed curve so that the starting point is the smallest integer always give a unique starting point?

- (b) Does a chain-coded closed curve always have an even number of segments? If your answer is yes, prove it. If it is no, give an example.

- (c) Find the normalized starting point of the code 11076765543322.

11.4 Do the following:

- (a)* Show that the first difference of a chain code normalizes it to rotation, as explained in Section 11.2.
- (b) Compute the first difference of the code 0101030303323232212111.

11.5 Answer the following:

- (a)* Given a one-pixel-thick, open or closed, 4-connected simple (does not intersect itself) digital curve, can a slope chain code be formulated so that it behaves exactly as a Freeman chain code? If your answer is no, explain why. If your answer is yes, explain how you would do it, detailing any assumptions you need to make for your answer to hold.
- (b) Repeat (a) for an 8-connected curve.
- (c) How would you normalize a slope chain code for scale changes?

11.6* Explain why a slope chain code with an angle accuracy of 10^{-1} produces 19 symbols.

11.7 Let *L* be the length of the straight-line segments used in a slope chain code. Assume that *L* is such that an integral number of line segments fit the

curve under consideration. Assume also that the angle accuracy is high enough so that it may be considered infinite for your purposes, answer the following:

- (a)* What is the tortuosity of a square boundary of size $d \times d$?
- (b)* What is the tortuosity of a circle of radius r ?
- (c) What is the tortuosity of a closed convex curve?

11.8* Advance an argument that explains why the uppermost-leftmost point of a digital closed curve has the property that a polygonal approximation to the curve has a convex vertex at that point.

11.9 With reference to Example 11.2, start with vertex V_7 and apply the MPP algorithm through, and including, V_{11} .

11.10 Do the following:

- (a)* Explain why the rubber-band polygonal approximation approach discussed in Section 11.2 yields a polygon with minimum perimeter for a convex curve.
- (b) Show that if each cell corresponds to a pixel on the boundary, the maximum possible error in that cell is $\sqrt{2}d$, where d is the minimum possible horizontal or vertical distance between adjacent pixels (i.e., the distance between lines in the sampling grid used to produce the digital image).

11.11 Explain how the MPP algorithm in Section 11.2 behaves under the following conditions:

- (a)* One-pixel wide, one-pixel deep indentations.
- (b)* One-pixel wide, two-or-more pixel deep indentations.
- (c) One-pixel wide, n -pixel long protrusions.

11.12 Do the following.

- (a)* Plot the signature of a square boundary using the tangent-angle method discussed in Section 11.2.
- (b) Repeat (a) for the slope density function. Assume that the square is aligned with the x - and y -axes, and let the x -axis be the reference line. Start at the corner closest to the origin.

11.13 Find an expression for the signature of each of

the following boundaries, and plot the signatures.

- (a)* An equilateral triangle.
- (b) A rectangle.
- (c) An ellipse

11.14 Do the following:

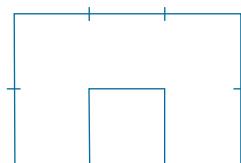
- (a)* With reference to Figs. 11.11(c) and (f), give a word description of an algorithm for counting the peaks in the two waveforms. Such an algorithm would allow us to differentiate between triangles and rectangles.
- (b) How can you make your solution independent of scale changes? You may assume that the scale changes are the same in both directions.

11.15 Draw the medial axis of:

- (a)* A circle.
- (b)* A square.
- (c) An equilateral triangle.

11.16 For the figure shown,

- (a)* What is the order of the shape number?
- (b) Obtain the shape number.



11.17* The procedure discussed in Section 11.3 for using Fourier descriptors consists of expressing the coordinates of a contour as complex numbers, taking the DFT of these numbers, and keeping only a few components of the DFT as descriptors of the boundary shape. The inverse DFT is then an approximation to the original contour. What class of contour shapes would have a DFT consisting of real numbers, and how would the axis system in Fig. 11.18 have to be set up to obtain those real numbers?

11.18 Show that if you use only two Fourier descriptors ($u = 0$ and $u = 1$) to reconstruct a boundary with Eq. (11-10), the result will always be a circle. (Hint: Use the parametric representation of a circle in the complex plane, and express the equation of a circle in polar coordinates.)

- 11.19*** Give the smallest number of statistical moment descriptors needed to differentiate between the signatures of the figures in Fig. 11.10.
- 11.20** Give two boundary shapes that have the same mean and third statistical moment descriptors, but different second moments.
- 11.21*** Propose a set of descriptors capable of differentiating between the shapes of the characters 0, 1, 8, 9, and X . (*Hint:* Use topological descriptors in conjunction with the convex hull.)
- 11.22** Consider a binary image of size 200×200 pixels, with a vertical black band extending from columns 1 to 99 and a vertical white band extending from columns 100 to 200.
- (a) Obtain the co-occurrence matrix of this image using the position operator “one pixel to the right.”
 - (b)* Normalize this matrix so that its elements become probability estimates, as explained in Section 11.4.
 - (c) Use your matrix from (b) to compute the six descriptors in Table 11.3.
- 11.23** Consider a checkerboard image composed of alternating black and white squares, each of size $m \times m$ pixels. Give a position operator that will yield a diagonal co-occurrence matrix.
- 11.24** Obtain the gray-level co-occurrence matrix of an array pattern of alternating single 0's and 1's (starting with 0) if:
- (a)* The position operator Q is defined as “one pixel to the right.”
 - (b) The position operator Q is defined as “two pixels to the right.”
- 11.25** Do the following.
- (a)* Prove the validity of Eqs. (11-50) and (11-51).
 - (b) Prove the validity of Eq. (11-52).
- 11.26*** We mentioned in Example 11.16 that a credible job could be done of reconstructing approximations to the six original images by using only the two principal-component images associated with the largest eigenvalues. What would be the mean squared error incurred in doing so? Express your answer as a percentage of the maximum possible error.
- 11.27** For a set of images of size 64×64 , assume that the covariance matrix given in Eq. (11-52) is the identity matrix. What would be the mean squared error between the original images and images reconstructed using Eq. (11-54) with only half of the original eigenvectors?
- 11.28** Under what conditions would you expect the major axes of a boundary, defined in the discussion of Eq. (11-4), to be equal to the eigen axes of that boundary?
- 11.29*** You are contracted to design an image processing system for detecting imperfections on the inside of certain solid plastic wafers. The wafers are examined using an X-ray imaging system, which yields 8-bit images of size 512×512 . In the absence of imperfections, the images appear uniform, having a mean intensity of 100 and variance of 400. The imperfections appear as blob-like regions in which about 70% of the pixels have excursions in intensity of 50 intensity levels or less about a mean of 100. A wafer is considered defective if such a region occupies an area exceeding 20×20 pixels in size. Propose a system based on texture analysis for solving this problem.
- 11.30** With reference to Fig. 11.46, answer the following:
- (a)* What is the cause of nearly identical clusters near the origin in Figs. 11.46(d)-(f).
 - (b) Look carefully, and you will see a single point near coordinates $(0.8, 0.8)$ in Fig. 11.46(f). What caused this point?
 - (c) The results in Fig. 11.46(d)-(e) are for the small image patches shown in Figs. 11.46(a)-(b). What would the results look like if we performed the computations over the entire image, instead of limiting the computation to the patches?
- 11.31** When we discussed the Harris-Stephens corner detector, we mentioned that there is a closed-form formula for computing the eigenvalues of a 2×2 matrix.
- (a)* Given matrix $\mathbf{M} = [a \ b; c \ d]$, give the general formula for finding its eigenvalues. Express your formula in terms of the trace and determinant of \mathbf{M} .
 - (b) Give the formula for *symmetric* matrices of

size 2×2 in terms of its four elements, without using the trace nor the determinant.

- 11.32*** With reference to the component tree in Fig. 11.51, assume that any pixels extending past the border of the small image are 0. Is region R_1 an extremal region? Explain.

- 11.33** With reference to the discussion of maximally stable extremal regions in Section 11.6, can the root of a component tree contain an MSER? Explain.

- 11.34*** The well known heat-diffusion equation of a temperature function $g(x,y,z,t)$ of three spatial variables, (x,y,z) , is given by $\partial g/\partial t - \alpha \nabla^2 g = 0$, where α is the thermal diffusivity and ∇^2 is the Laplacian operator. In terms of our discussion of SIFT, the form of this equation is used to establish a relationship between the difference of Gaussians and the scaled Laplacian, $\sigma^2 \nabla^2$. Show how this can be done to derive Eq. (11-70).

- 11.35** With reference to the SIFT algorithm discussed in Section 11.7, assume that the input image is square, of size $M \times M$ (with $M = 2^n$), and let the number of intervals per octave be $s = 2$.

- (a) How many smoothed images will there be in each octave?
- (b)* How many octaves could be generated before it is no longer possible to down-sample the image by 2?
- (c) If the standard deviation used to smooth the first image in the first octave is σ , what are the values of standard deviation used to smooth the first image in each of the remaining octaves in (b)?

- 11.36** Advance an argument showing that smoothing an image and then downsampling it by 2 gives the same result as first downsampling the image by 2 and then smoothing it with the same kernel. By downsampling we mean skipping every other row and column. (*Hint:* Consider the fact that convolution is a linear process.)

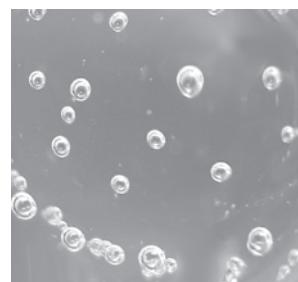
- 11.37** Do the following:

- (a)* Show how to obtain Eq. (11-74) from Eq. (11-71).
- (b) Show how Eq. (11-75) follows from Eqs. (11-74) and (11-71).

- 11.38** A company that bottles a variety of industrial chemicals employs you to design an approach for detecting when bottles of their product are not full. As they move along a conveyor line past an automatic filling and capping station, the bottles appear as shown in the following image. A bottle is considered imperfectly filled when the level of the liquid is below the midway point between the bottom of the neck and the shoulder of the bottle. The shoulder is defined as the intersection of the sides and slanted portions of the bottle. The bottles move at a high rate of speed, but the company has an imaging system equipped with an illumination flash front end that effectively stops motion, so you will be given images that look very close to the sample shown here. Based on the material you have learned up to this point, propose a solution for detecting bottles that are not filled properly. State clearly all assumptions that you make and that are likely to impact the solution you propose.



- 11.39** Having heard about your success with the bottle inspection problem, you are contacted by a fluids company that wishes to automate bubble-counting in certain processes for quality control. The company has solved the imaging problem and can obtain 8-bit images of size 700×700 pixels, such as the one shown in the figure below.



Each image represents an area of 7cm^2 . The company wishes to do two things with each

image: (1) Determine the ratio of the area occupied by bubbles to the total area of the image; and (2) count the number of distinct bubbles. Based on the material you have learned up to this point, propose a solution to this problem. In

your report, state the physical dimensions of the smallest bubble your solution can detect. State clearly all assumptions that you make and that are likely to impact the solution you propose.

12

Image Pattern Classification

One of the most interesting aspects of the world is that it can be considered to be made up of patterns.

A pattern is essentially an arrangement. It is characterized by the order of the elements of which it is made, rather than by the intrinsic nature of these elements.

Norbert Wiener

Preview

We conclude our coverage of digital image processing with an introduction to techniques for image pattern classification. The approaches developed in this chapter are divided into three principal categories: classification by *prototype matching*, classification based on an *optimal statistical* formulation, and classification based on *neural networks*. The first two approaches are used extensively in applications in which the nature of the data is well understood, leading to an effective pairing of features and classifier design. These approaches often rely on a great deal of engineering to define features and elements of a classifier. Approaches based on neural networks rely less on such knowledge, and lend themselves well to applications in which pattern class characteristics (e.g., features) are learned by the system, rather than being specified *a priori* by a human designer. The focus of the material in this chapter is on principles, and on how they apply specifically in image pattern classification.

Upon completion of this chapter, readers should:

- Understand the meaning of patterns and pattern classes, and how they relate to digital image processing.
- Be familiar with the basics of minimum-distance classification.
- Know how to apply image correlation techniques for template matching.
- Understand the concept of string matching.
- Be familiar with Bayes classifiers.
- Understand perceptrons and their history.
- Be familiar with the concept of learning from training samples.
- Understand neural network architectures.
- Be familiar with the concept of deep learning in fully connected and deep convolutional neural networks. In particular, be familiar with the importance of the latter in digital image processing.

12.1 BACKGROUND

Humans possess the most sophisticated pattern recognition capabilities in the known biological world. By contrast, the capabilities of current recognition machines pale in comparison with tasks humans perform routinely, from being able to interpret the meaning of complex images, to our ability for generalizing knowledge stored in our brains. But recognition machines play an important, sometimes even crucial role in everyday life. Imagine what modern life would be like without machines that read barcodes, process bank checks, inspect the quality of manufactured products, read fingerprints, sort mail, and recognize speech.

In image pattern recognition, we think of a *pattern* as a spatial arrangement of features. A *pattern class* is a set of patterns that share some common properties. Pattern recognition by machine encompasses techniques for automatically assigning patterns to their respective classes. That is, given a pattern or sets of patterns whose class is unknown, the job of a pattern recognition system is to assign a class label to each of its input patterns.

There are four main stages involved in recognition: (1) sensing, (2) preprocessing, (3) feature extraction, and (4) classification. In terms of image processing, sensing is concerned with generating signals in a spatial (2-D) or higher-dimensional format. We covered numerous aspects of image sensing in Chapter 1. Preprocessing deals with techniques for tasks such as noise reduction, enhancement, restoration, and segmentation, as discussed in earlier chapters. You learned about feature extraction in Chapter 11. Classification, the focus of this chapter, deals with using a set of features as the basis for assigning class labels to unknown input image patterns.

In the following section, we will discuss three basic approaches used for image pattern classification: (1) classification based on matching unknown patterns against specified prototypes, (2) optimum statistical classifiers, and (3) neural networks. One way to characterize the differences between these approaches is in the level of “engineering” required to transform raw data into formats suitable for computer processing. Ultimately, recognition performance is determined by the discriminative power of the features used.

In classification based on prototypes, the objective is to make the features so unique and easily detectable that classification itself becomes a simple task. A good example of this are bank-check processors, which use stylized font styles to simplify machine processing (we will discuss this application in Section 12.3).

In the second category, classification is cast in decision-theoretic, statistical terms, and the classification approach is based on selecting parameters that can be shown to yield optimum classification performance in a statistical sense. Here, emphasis is placed on both the features used, and the design of the classifier. We will illustrate this approach in Section 12.4 by deriving the Bayes pattern classifier, starting from basic principles.

In the third category, classification is performed using neural networks. As you will learn in Sections 12.5 and 12.6, neural networks can operate using engineered features too, but they have the unique ability of being able to generate, on their own, representations (features) suitable for recognition. These systems can accomplish this using raw data, without the need for engineered features.

One characteristic shared by the preceding three approaches is that they are based on parameters that must be either specified or learned from patterns that represent the recognition problem we want to solve. The patterns can be *labeled*, meaning that we know the class of each pattern, or *unlabeled*, meaning that the data are known to be patterns, but the class of each pattern is unknown. A classic example of labeled data is the character recognition problem, in which a set of character samples is collected and the identity of each character is recorded as a label from the group 0 through 9 and *a* through *z*. An example of unlabeled data is when we are seeking clusters in a data set, with the aim of utilizing the resulting cluster centers as being prototypes of the pattern classes contained in the data.

When working with a labeled data, a given data set generally is subdivided into three subsets: a *training set*, a *validation set*, and a *test set* (a typical subdivision might be 50% training, and 25% each for the validation and test sets). The process by which a training set is used to generate classifier parameters is called *training*. In this mode, a classifier is given the class label of each pattern, the objective being to make adjustments in the parameters if the classifier makes a mistake in identifying the class of the given pattern. At this point, we might be working with several candidate designs. At the end of training, we use the validation set to compare the various designs against a performance objective. Typically, several iterations of training/validation are required to establish the design that comes closest to meeting the desired objective. Once a design has been selected, the final step is to determine how it will perform “in the field.” For this, we use the test set, which consists of patterns that the system has never “seen” before. If the training and validation sets are truly representative of the data the system will encounter in practice, the results of training/validation should be close to the performance using the test set. If training/validation results are acceptable, but test results are not, we say that training/validation “over fit” the system parameters to the available data, in which case further work on the system architecture is required. Of course all this assumes that the given data are truly representative of the problem we want to solve, and that the problem in fact can be solved by available technology.

A system that is designed using training data is said to undergo *supervised learning*. If we are working with unlabeled data, the system learns the pattern classes themselves while in an *unsupervised* learning mode. In this chapter, we deal only with supervised learning. As you will see in this and the next chapter, supervised learning covers a broad range of approaches, from applications in which a system learns parameters of features whose form is fixed by a designer, to systems that utilize *deep learning* and large sets of raw data sets to learn, *on their own*, the features required for classification. These systems accomplish this task without a human designer having to specify the features, *a priori*.

After a brief discussion in the next section of how patterns are formed, and on the nature of patterns classes, we will discuss in Section 12.3 various approaches for prototype-based classification. In Section 12.4, we will start from basic principles and derive the equations of the Bayes classifier, an approach characterized by optimum classification performance on an average basis. We will also discuss supervised training of a Bayes classifier based on the assumption of multivariate Gaussian

Because the examples in this chapter are intended to demonstrate basic principles and are not large scale, we dispense with validation and subdivide the pattern data into training and test sets.

Generally, we associate the concept of deep learning with large sets of data. These ideas are discussed in more detail later in this section and next.

distributions. Starting with Section 12.5, we will spend the rest of the chapter discussing neural networks. We will begin Section 12.5 with a brief introduction to perceptrons and some historical facts about machine learning. Then, we will introduce the concept of deep neural networks and derive the equations of backpropagation, the method of choice for training deep neural nets. These networks are well-suited for applications in which input patterns are vectors. In Section 12.6, we will introduce deep convolutional neural networks, which currently are the preferred approach when the system inputs are digital images. After deriving the backpropagation equations used for training convolutional nets, we will give several examples of applications involving classes of images of various complexities. In addition to working directly with image inputs, deep convolutional nets are capable of learning, on their own, image features suitable for classification. This is accomplished starting with raw image data, as opposed to the other classification methods discussed in Sections 12.3 and 12.4, which rely on “engineered” features whose form, as noted earlier, is specified a priori by a human designer.

12.2 PATTERNS AND PATTERN CLASSES

In image pattern classification, the two principal pattern arrangements are quantitative and structural. *Quantitative patterns* are arranged in the form of *pattern vectors*. *Structural patterns* typically are composed of symbols, arranged in the form of *strings*, *trees*, or, less frequently, as *graphs*. Most of the work in this chapter is based on pattern vectors, but we will discuss structural patterns briefly at the end of this section, and give an example at the end of Section 12.3.

PATTERN VECTORS

Pattern vectors are represented by lowercase letters, such as \mathbf{x} , \mathbf{y} , and \mathbf{z} , and have the form

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad (12-1)$$

where each component, x_i , represents the i th feature descriptor, and n is the total number of such descriptors. We can express a vector in the form of a column, as in Eq. (12-1), or in the equivalent row form $\mathbf{x} = (x_1, x_2, \dots, x_n)^T$, where T indicates transposition. A pattern vector may be “viewed” as a point in n -dimensional Euclidean space, and a pattern class may be interpreted as a “hypercloud” of points in this *pattern space*. For the purpose of recognition, we like for our pattern classes to be grouped tightly, and as far away from each other as possible.

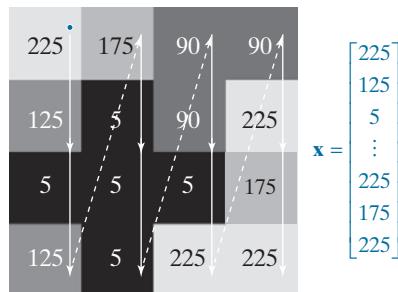
Pattern vectors can be formed directly from image pixel intensities by vectorizing the image using, for example, linear indexing, as in Fig. 12.1. A more common approach is for pattern elements to be features. An early example is the work of Fisher [1936] who, close to a century ago, reported the use of what then was a new

We discussed linear indexing in Section 2.4 (see Fig. 2.22).

a b

FIGURE 12.1

Using linear indexing to vectorize a grayscale image.



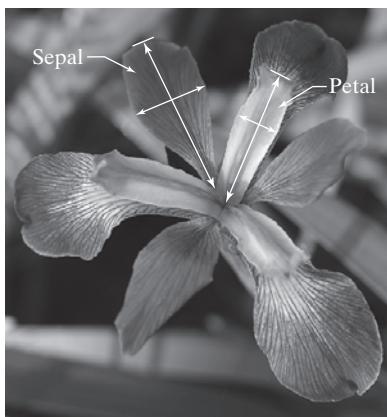
Sepals are the undergrowth beneath the petals.

technique called *discriminant analysis* to recognize three types of iris flowers (*Iris setosa*, *virginica*, and *versicolor*). Fisher described each flower using four features: the length and width of the petals, and similarly for the sepals (see Fig. 12.2). This leads to the 4-D vectors shown in the figure. A set of these vectors, obtained for fifty samples of each flower gender, constitutes the three famous *Fisher iris pattern classes*. Had Fisher been working today, he probably would have added spectral colors and shape features to his measurements, yielding vectors of higher dimensionality. We will be working with the original iris data set later in this chapter.

A higher-level representation of patterns is based on feature descriptors of the types you learned in Chapter 11. For instance, pattern vectors formed from descriptors of boundary shape are well-suited for applications in controlled environments, such as industrial inspection. Figure 12.3 illustrates the concept. Here, we are interested in classifying different types of noisy shapes, a sample of which is shown in the figure. If we represent an object by its signature, we would obtain 1-D signals of the form shown in Fig. 12.3(b). We can express a signature as a vector by sampling its amplitude at increments of θ , then forming a vector by letting $x_i = r(\theta_i)$, for $i = 0, 1, 2, \dots, n$. Instead of using “raw” sampled signatures, a more common approach is to compute some function, $x_i = g(r(\theta_i))$, of the signature samples and use them to form vectors. You learned in Section 11.3 several approaches to do this, such as statistical moments.

FIGURE 12.2

Petal and sepal width and length measurements (see arrows) performed on iris flowers for the purpose of data classification. The image shown is of the *Iris virginica* gender. (Image courtesy of USDA.)



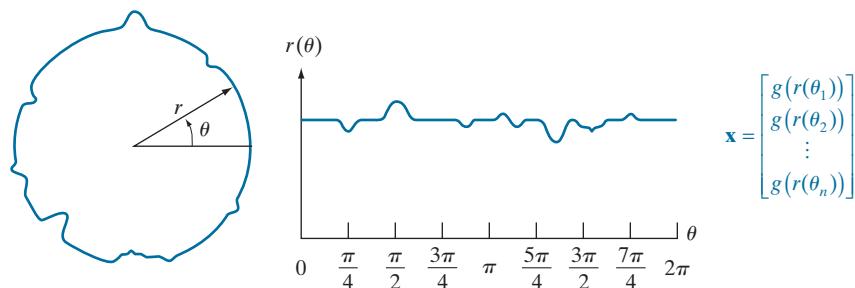
$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$$

- x_1 = Petal width
- x_2 = Petal length
- x_3 = Sepal width
- x_4 = Sepal length

a b

FIGURE 12.3

(a) A noisy object boundary, and (b) its corresponding signature.



$$\mathbf{x} = \begin{bmatrix} g(r(\theta_1)) \\ g(r(\theta_2)) \\ \vdots \\ g(r(\theta_n)) \end{bmatrix}$$

Vectors can be formed also from features of both boundary and regions. For example, the objects in Fig. 12.4 can be represented by 3-D vectors whose components capture shape information related to both boundary and region properties of single binary objects. Pattern vectors can be used also to represent properties of image regions. For example, the elements of the 6-D vector in Fig. 12.5 are texture measures based on the feature descriptors in Table 11.3. Figure 12.6 shows an example in which pattern vector elements are features that are invariant to transformations, such as image rotation and scaling (see Section 11.4).

When working with sequences of registered images, we have the option of using pattern vectors formed from corresponding pixels in those images (see Fig. 12.7). Forming pattern vectors in this way implies that recognition will be based on information extracted from the same spatial location across the images. Although this may seem like a very limiting approach, it is ideally suited for applications such as recognizing regions in multispectral images, as you will see in Section 12.4.

When working with entire images as units, we need the detail afforded by vectors of much-higher dimensionality, such as those we discussed in Section 11.7 in connection with the SIFT algorithm. However, a more powerful approach when working with entire images is to use deep convolutional neural networks. We will discuss neural nets in detail in Sections 12.5 and 12.6.

STRUCTURAL PATTERNS

Pattern vectors are not suitable for applications in which objects are represented by structural features, such as strings of symbols. Although they are used much less than vectors in image processing applications, patterns containing structural descriptions of objects are important in applications where shape is of interest. Figure 12.8 shows an example. The boundaries of the bottles were approximated by a polygon

a b c d

FIGURE 12.4

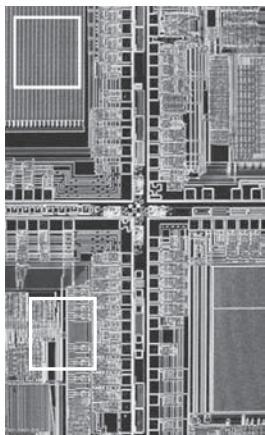
Pattern vectors whose components capture both boundary and regional characteristics.



$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \begin{aligned} x_1 &= \text{compactness} \\ x_2 &= \text{circularity} \\ x_3 &= \text{eccentricity} \end{aligned}$$

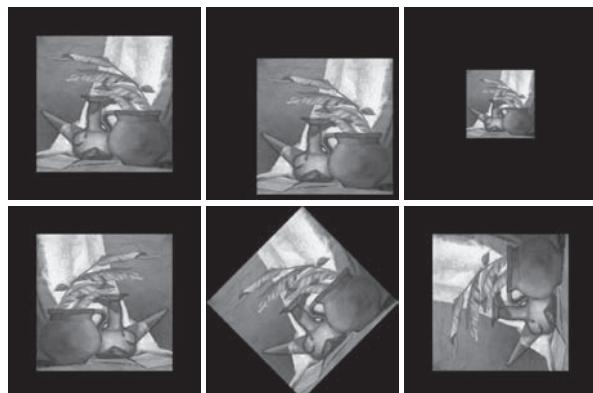
FIGURE 12.5

An example of pattern vectors based on properties of subimages. See Table 11.3 for an explanation of the components of \mathbf{x} .



$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix}$$

x_1 = max probability
 x_2 = correlation
 x_3 = contrast
 x_4 = uniformity
 x_5 = homogeneity
 x_6 = entropy

FIGURE 12.6 Feature vectors with components that are invariant to transformations such as rotation, scaling, and translation. The vector components are moment invariants.

The ϕ 's are moment invariants

Images in spectral bands 1–3

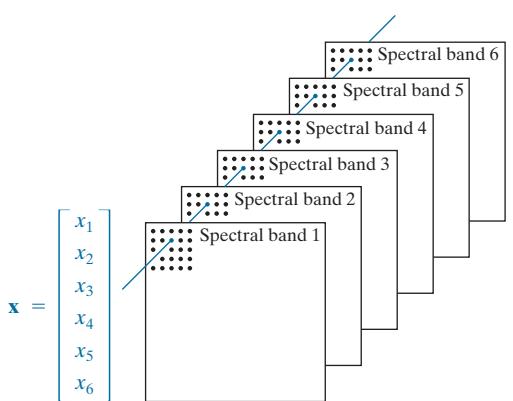
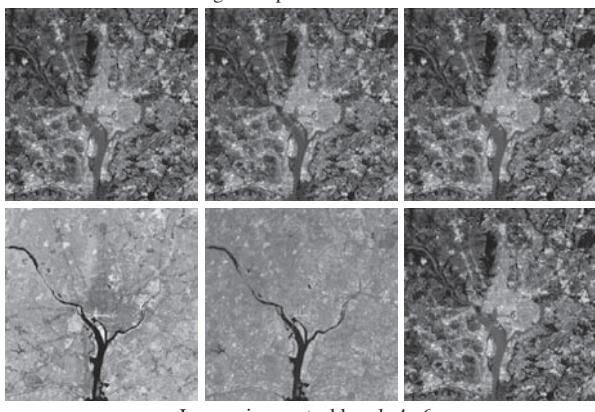
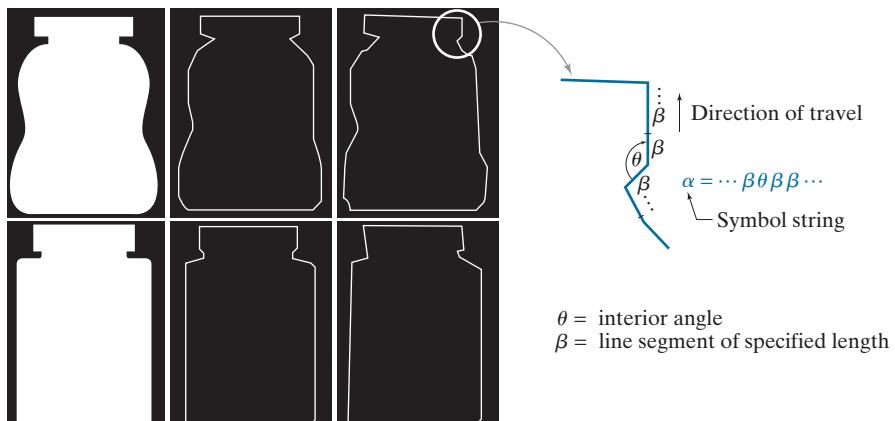
**FIGURE 12.7** Pattern (feature) vectors formed by concatenating corresponding pixels from a set of registered images. (Original images courtesy of NASA.)

FIGURE 12.8

Symbol string generated from a polygonal approximation of the boundaries of medicine bottles.



using the approach explained in Section 11.2. The boundary is subdivided into line segments (denoted by β in the figure), and the interior angle, θ , is computed at each intersection of two line segments. A string of sequential symbols is generated as the boundary is traversed in the counterclockwise direction, as the figure shows. *Strings* of this form are structural patterns, and the objective, as you will see in Section 12.3, is to match a given string against stored string prototypes.

A *tree* is another structural representation, suitable for higher-level descriptions of an entire image in terms of its component regions. Basically, most hierarchical ordering schemes lead to tree structures. For example, Fig. 12.9 shows a satellite image of a heavily built downtown area and surrounding residential areas. Let the symbol \$ represent the root of a tree. The (upside down) tree shown in the figure was obtained using the structural relationship “composed of.” Thus, the root of the tree represents the entire image. The next level indicates that the image is composed of a downtown and residential areas. In turn, the residential areas are composed of housing, highways, and shopping malls. The next level down in the tree further describes the housing and highways. We can continue this type of subdivision until we reach the limit of our ability to resolve different regions in the image.

12.3 PATTERN CLASSIFICATION BY PROTOTYPE MATCHING

Prototype matching involves comparing an unknown pattern against a set of prototypes, and assigning to the unknown pattern the class of the prototype that is the most “similar” to the unknown. Each prototype represents a unique pattern class, but there may be more than one prototype for each class. What distinguishes one matching method from another is the measure used to determine similarity.

MINIMUM-DISTANCE CLASSIFIER

The minimum-distance classifier is also referred to as the *nearest-neighbor classifier*.

One of the simplest and most widely used prototype matching methods is the *minimum-distance classifier* which, as its name implies, computes a distance-based measure between an unknown pattern vector and each of the class prototypes. It then assigns the unknown pattern to the class of its closest prototype. The prototype

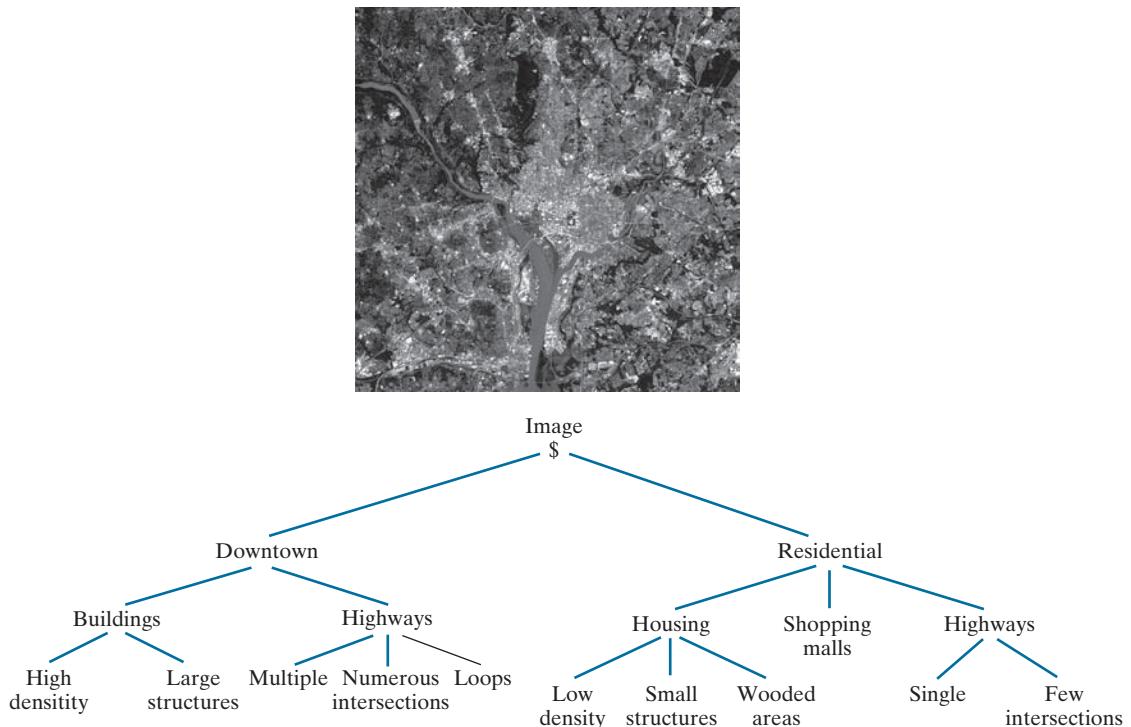


FIGURE 12.9 Tree representation of a satellite image showing a heavily built downtown area (Washington, D.C.) and surrounding residential areas. (Original image courtesy of NASA.)

vectors of the minimum-distance classifier usually are the mean vectors of the various pattern classes:

$$\mathbf{m}_j = \frac{1}{n_j} \sum_{\mathbf{x} \in c_j} \mathbf{x} \quad j = 1, 2, \dots, N_c \quad (12-2)$$

where n_j is the number of pattern vectors used to compute the j th mean vector, c_j is the j th pattern class, and N_c is the number of classes. If we use the Euclidean distance to determine similarity, the minimum-distance classifier computes the distances

$$D_j(\mathbf{x}) = \|\mathbf{x} - \mathbf{m}_j\| \quad j = 1, 2, \dots, N_c \quad (12-3)$$

where $\|\mathbf{a}\| = (\mathbf{a}^T \mathbf{a})^{1/2}$ is the Euclidean norm. The classifier then assigns an unknown pattern \mathbf{x} to class c_i if $D_i(\mathbf{x}) < D_j(\mathbf{x})$ for $j = 1, 2, \dots, N_c$, $j \neq i$. Ties [i.e., $D_i(\mathbf{x}) = D_j(\mathbf{x})$] are resolved arbitrarily.

It is not difficult to show (see Problem 12.2) that selecting the smallest distance is equivalent to evaluating the functions

$$d_j(\mathbf{x}) = \mathbf{m}_j^T \mathbf{x} - \frac{1}{2} \mathbf{m}_j^T \mathbf{m}_j \quad j = 1, 2, \dots, N_c \quad (12-4)$$

and assigning an unknown pattern \mathbf{x} to the class whose prototype yielded the *largest* value of d . That is, \mathbf{x} is assigned to class c_i , if

$$d_i(\mathbf{x}) > d_j(\mathbf{x}) \quad j = 1, 2, \dots, N_c; j \neq i \quad (12-5)$$

When used for recognition, functions of this form are referred to as *decision* or *discriminant functions*.

The *decision boundary* separating class c_i from c_j is given by the values of \mathbf{x} for which

$$d_i(\mathbf{x}) = d_j(\mathbf{x}) \quad (12-6)$$

or, equivalently, by values of \mathbf{x} for which

$$d_i(\mathbf{x}) - d_j(\mathbf{x}) = 0 \quad (12-7)$$

The decision boundaries for a minimum-distance classifier follow directly from this equation and Eq. (12-4):

$$\begin{aligned} d_{ij}(\mathbf{x}) &= d_i(\mathbf{x}) - d_j(\mathbf{x}) \\ &= (\mathbf{m}_i - \mathbf{m}_j)^T \mathbf{x} - \frac{1}{2} (\mathbf{m}_i - \mathbf{m}_j)^T (\mathbf{m}_i + \mathbf{m}_j) = 0 \end{aligned} \quad (12-8)$$

The boundary given by Eq. (12-8) is the perpendicular bisector of the line segment joining \mathbf{m}_i and \mathbf{m}_j (see Problem 12.3). In 2-D (i.e., $n = 2$), the perpendicular bisector is a line, for $n = 3$ it is a plane, and for $n > 3$ it is called a *hyperplane*.

EXAMPLE 12.1: Illustration of the minimum-distance classifier for two classes in 2-D.

Figure 12.10 shows scatter plots of petal width and length values for the classes Iris versicolor and Iris setosa. As mentioned in the previous section, pattern vectors in the iris database consists of four measurements for each flower. We show only two here so that you can visualize the pattern classes and the decision boundary between them. We will work with the complete database later in this chapter.

We denote the Iris versicolor and setosa data as classes c_1 and c_2 , respectively. The means of the two classes are $\mathbf{m}_1 = (4.3, 1.3)^T$ and $\mathbf{m}_2 = (1.5, 0.3)^T$. It then follows from Eq. (12-4) that

$$\begin{aligned} d_1(\mathbf{x}) &= \mathbf{m}_1^T \mathbf{x} - \frac{1}{2} \mathbf{m}_1^T \mathbf{m}_1 \\ &= 4.3x_1 + 1.3x_2 - 10.1 \end{aligned}$$

and

$$\begin{aligned} d_2(\mathbf{x}) &= \mathbf{m}_2^T \mathbf{x} - \frac{1}{2} \mathbf{m}_2^T \mathbf{m}_2 \\ &= 1.5x_1 + 0.3x_2 - 1.17 \end{aligned}$$

From Eq. (12-8), the equation of the boundary is

$$\begin{aligned} d_{12}(\mathbf{x}) &= d_1(\mathbf{x}) - d_2(\mathbf{x}) \\ &= 2.8x_1 + 1.0x_2 - 8.9 = 0 \end{aligned}$$

FIGURE 12.10

Decision boundary of a minimum distance classifier (based on two measurements) for the classes of Iris versicolor and Iris setosa. The dark dot and square are the means of the two classes.

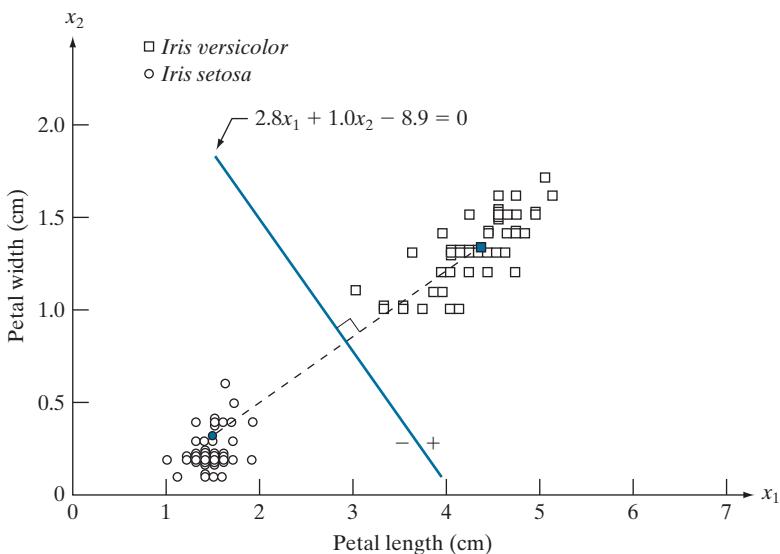


Figure 12.10 shows a plot of this boundary. Substituting any pattern vector from class c_1 into this equation would yield $d_{12}(\mathbf{x}) > 0$. Conversely, any pattern from class c_2 would give $d_{12}(\mathbf{x}) < 0$. Thus, given an unknown pattern \mathbf{x} belonging to one of these two classes, the sign of $d_{12}(\mathbf{x})$ would be sufficient to determine the class to which that pattern belongs.

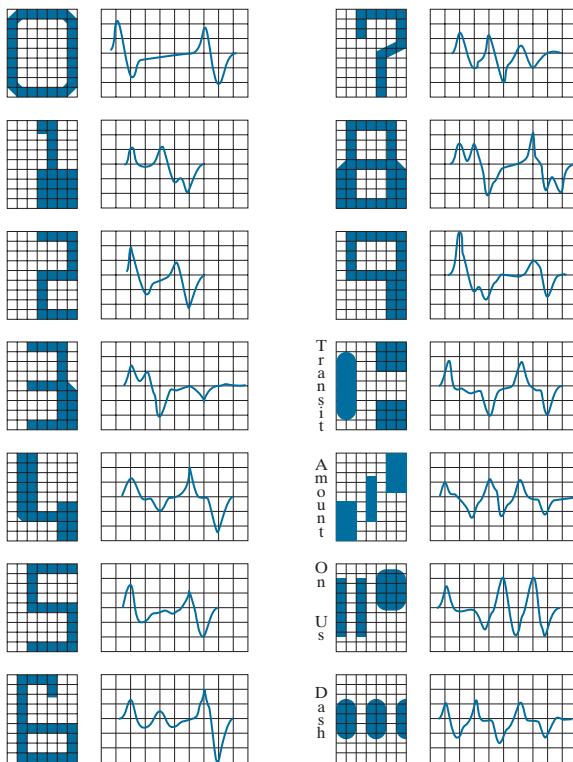
The minimum-distance classifier works well when the distance between means is large compared to the spread or randomness of each class with respect to its mean. In Section 12.4 we will show that the minimum-distance classifier yields optimum performance (in terms of minimizing the average loss of misclassification) when the distribution of each class about its mean is in the form of a spherical “hypercloud” in n -dimensional pattern space.

As noted earlier, one of the keys to accurate recognition performance is to specify features that are effective discriminators between classes. As a rule, the better the features are at meeting this objective, the better the recognition performance will be. In the case of the minimum-distance classifier this implies wide separation between means and tight grouping of the classes.

Systems based on the Banker's Association E-13B font character are a classic example of how highly engineered features can be used in conjunction with a simple classifier to achieve superior results. In the mid-1940s, bank checks were processed manually, which was a laborious, costly process prone to mistakes. As the volume of check writing increased in the early 1950s, banks became keenly interested in automating this task. In the middle 1950s, the E-13B font and the system that reads it became the standard solution to the problem. As Fig. 12.11 shows, this font set consists of 14 characters laid out on a 9×7 grid. The characters are stylized to maximize the difference between them. The font was designed to be compact and readable by humans, but the overriding purpose was that the characters should be readable by machine, quickly, and with very high accuracy.

FIGURE 12.11

The American Bankers Association E-13B font character set and corresponding waveforms.



Appropriately, recognition of magnetized characters is referred to as *Magnetic Ink Character Recognition (MICR)*.

In addition to a stylized font design, the operation of the reading system is further enhanced by printing each character using an ink that contains finely ground magnetic material. To improve character detectability in a check being read, the ink is subjected to a magnetic field that accentuates each character against the background. The stylized design further enhances character detectability. The characters are scanned in a horizontal direction with a single-slit reading head that is narrower but taller than the characters. As a check passes through the head, the sensor produces a 1-D electrical signal (a signature) that is conditioned to be proportional to the rate of increase or decrease of the character area under the head. For example, consider the waveform of the number 0 in Fig. 12.11. As a check moves to the right past the head, the character area seen by the sensor begins to increase, producing a positive derivative (a positive rate of change). As the right leg of the character begins to pass under the head, the character area seen by the sensor begins to decrease, producing a negative derivative. When the head is in the middle zone of the character, the area remains nearly constant, producing a zero derivative. This waveform repeats itself as the other leg of the character enters the head. The design of the font ensures that the waveform of each character is distinct from all others. It also ensures that the peaks and zeros of each waveform occur approximately on the vertical lines of the background grid on which these waveforms are displayed, as the figure shows. The E-13B font has the property that sampling the waveforms only at these (nine)

points yields enough information for their accurate classification. The effectiveness of these highly engineered features is further refined by the magnetized ink, which results in clean waveforms with almost no scatter.

Designing a minimum-distance classifier for this application is straightforward. We simply store the sample values of each waveform at the vertical lines of the grid, and let each set of the resulting samples be represented as a 9-D prototype vector, \mathbf{m}_j , $j = 1, 2, \dots, 14$. When an unknown character is to be classified, the approach is to scan it in the manner just described, express the grid samples of the waveform as a 9-D vector, \mathbf{x} , and identify its class by selecting the class of the prototype vector that yields the highest value in Eq. (12-4). We do not even need a computer to do this. Very high classification speeds can be achieved with analog circuits composed of resistor banks (see Problem 12.4).

The most important lesson in this example is that a recognition problem often can be made trivial if we can control the environment in which the patterns are generated. The development and implementation of the E13-B font reading system is a striking example of this fact. On the other hand, this system would be inadequate if we added the requirement that it has to recognize the textual content and signature written on each check. For this, we need systems that are significantly more complex, such as the convolutional neural networks we will discuss in Section 12.6.

USING CORRELATION FOR 2-D PROTOTYPE MATCHING

We introduced the basic idea of spatial correlation and convolution in Section 3.4, and used these concepts extensively in Chapter 3 for spatial filtering. From Eq. (3-34), we know that correlation of a kernel w with an image $f(x, y)$ is given by

$$(w \star f)(x, y) = \sum_s \sum_t w(s, t)f(x + s, y + t) \quad (12-9)$$

where the limits of summation are taken over the region shared by w and f . This equation is evaluated for all values of the displacement variables x and y so all elements of w visit every pixel of f . As you know, correlation has its highest value(s) in the region(s) where f and w are equal or nearly equal. In other words, Eq. (12-9) finds locations where w *matches* a region of f . But this equation has the drawback that the result is sensitive to changes in the amplitude of either function. In order to normalize correlation to amplitude changes in one or both functions, we perform matching using the *correlation coefficient* instead:

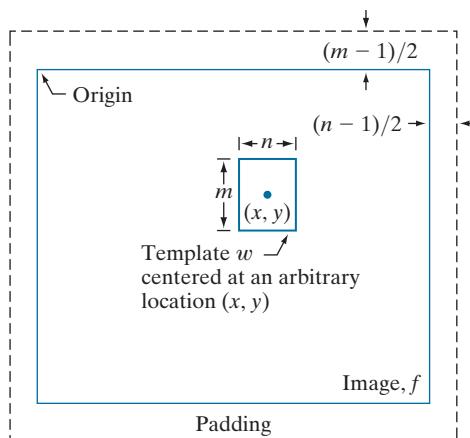
$$\gamma(x, y) = \frac{\sum_s \sum_t [w(s, t) - \bar{w}] [f(x + s, y + t) - \bar{f}_{xy}]}{\left\{ \sum_s \sum_t [w(s, t) - \bar{w}]^2 \sum_s \sum_t [f(x + s, y + t) - \bar{f}_{xy}]^2 \right\}^{1/2}} \quad (12-10)$$

where the limits of summation are taken over the region shared by w and f , \bar{w} is the average value of the kernel (computed only once), and \bar{f}_{xy} is the average value of f in the region coincident with w . In image correlation work, w is often referred to as a *template* (i.e., a prototype subimage) and correlation is referred to as *template matching*.

To be formal, we should refer to correlation (and the correlation coefficient) as *cross-correlation* when the functions are different, and as *autocorrelation* when they are the same. However, it is customary to use the generic term *correlation* and *correlation coefficient*, except when the distinction is important (as in deriving equations, in which it makes a difference which is being applied).

FIGURE 12.12

The mechanics of template matching.



It can be shown (see Problem 12.5) that $\gamma(x, y)$ has values in the range $[-1, 1]$ and is thus normalized to changes in the amplitudes of w and f . The maximum value of γ occurs when the normalized w and the corresponding normalized region in f are identical. This indicates *maximum correlation* (the best possible match). The *minimum* occurs when the two normalized functions exhibit the least similarity in the sense of Eq. (12-10).

Figure 12.12 illustrates the mechanics of the procedure just described. The border around image f is padding, as explained in Section 3.4. In template matching, values of correlation when the center of the template is past the border of the image generally are of no interest, so the padding is limited to half the kernel width.

The template in Fig. 12.12 is of size $m \times n$, and it is shown with its center at an arbitrary location (x, y) . The value of the correlation coefficient at that point is computed using Eq. (12-10). Then, the center of the template is incremented to an adjacent location and the procedure is repeated. Values of the correlation coefficient $\gamma(x, y)$ are obtained by moving the center of the template (i.e., by incrementing x and y) so the center of w visits every pixel in f . At the end of the procedure, we look for the maximum in $\gamma(x, y)$ to find where the best match occurred. It is possible to have multiple locations in $\gamma(x, y)$ with the same maximum value, indicating several matches between w and f .

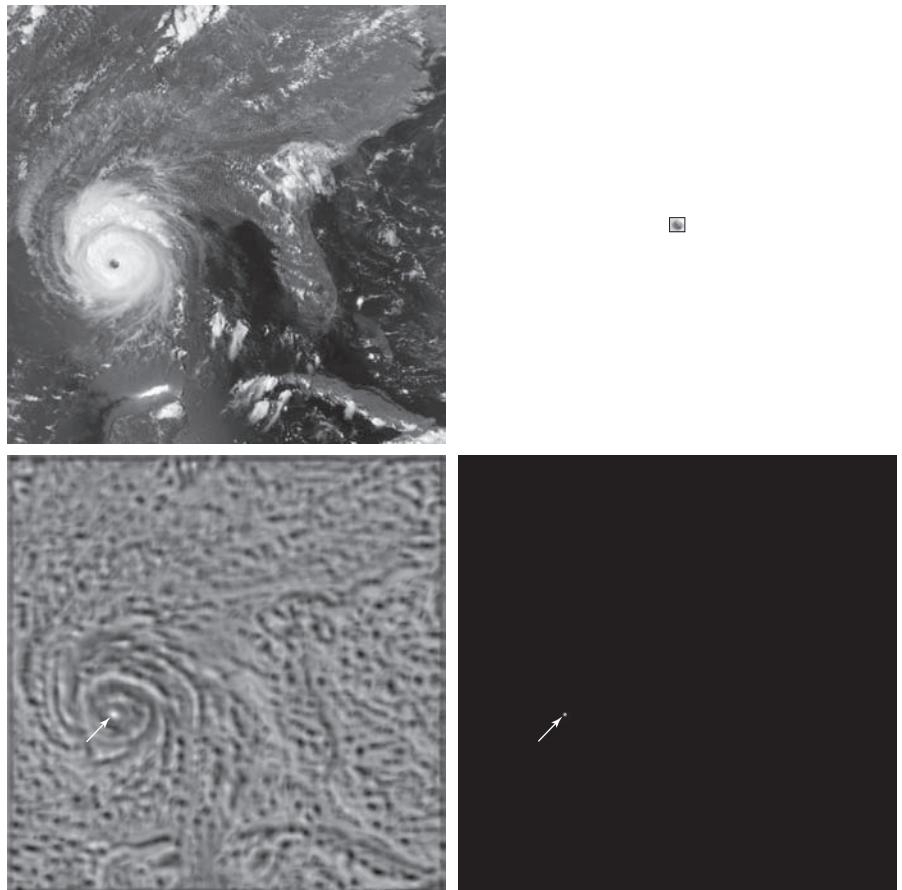
EXAMPLE 12.2: Matching by correlation.

Figure 12.13(a) shows a 913×913 satellite image of 1992 Hurricane Andrew, in which the eye of the storm is clearly visible. We want to use correlation to find the location of the best match in Fig. 12.13(a) of the template in Fig. 12.13(b), which is a 31×31 subimage of the eye of the storm. Figure 12.13(c) shows the result of computing the correlation coefficient in Eq. (12-10) for all values of x and y in the original image. The size of this image was 943×943 pixels due to padding (see Fig. 12.12), but we cropped it to the size of the original image for display. The intensity in this image is proportional to the correlation values, and all negative correlations were clipped at 0 (black) to simplify the visual analysis of the image. The area of highest correlation values appears as a small white region in this image. The brightest point in this region matches with the center of the eye of the storm. Figure 12.13(d) shows as a

a	b
c	d

FIGURE 12.13

- (a) 913×913 satellite image of Hurricane Andrew.
 (b) 31×31 template of the eye of the storm.
 (c) Correlation coefficient shown as an image (note the brightest point, indicated by an arrow).
 (d) Location of the best match (identified by the arrow). This point is a single pixel, but its size was enlarged to make it easier to see.
 (Original image courtesy of NOAA.)



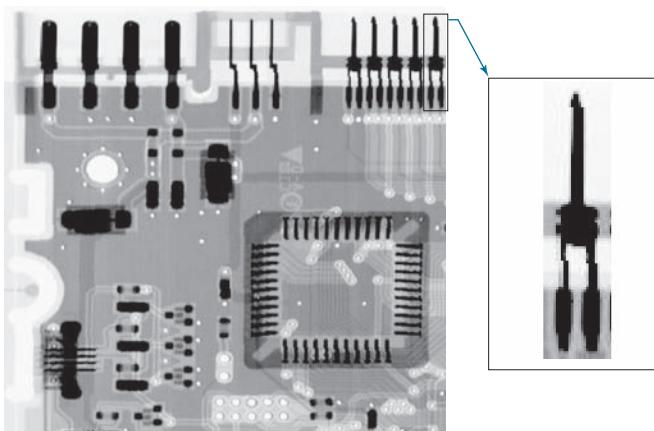
white dot the location of this maximum correlation value (in this case there was a unique match whose maximum value was 1), which we see corresponds closely with the location of the eye in Fig. 12.13(a).

MATCHING SIFT FEATURES

We discussed the scale-invariant feature transform (SIFT) in Section 11.7. SIFT computes a set of invariant features that can be used for matching between known (prototype) and unknown images. The SIFT implementation in Section 11.7 yields 128-dimensional feature vectors for each local region in an image. SIFT performs matching by looking for correspondences between sets of stored feature vector prototypes and feature vectors computed for an unknown image. Because of the large number of features involved, searching for exact matches is computationally intensive. Instead, the approach is to use a best-bin-first method that can identify the nearest neighbors with high probability using only a limited amount of computation (see Lowe [1999], [2004]). The search is further simplified by looking for clusters of potential solutions using the generalized Hough transform proposed by Ballard [1981]. We

FIGURE 12.14

Circuit board image of size 948×915 pixels, and a subimage of one of the connectors. The subimage is of size 212×128 pixels, shown zoomed on the right for clarity. (Original image courtesy of Mr. Joseph E. Pascente, Lixi, Inc.)



know from the discussion in Section 10.2 that the Hough transform simplifies looking for data patterns by utilizing bins that reduce the level of detail with which we look at a data set. We already discussed the SIFT algorithm in Section 11.7. The focus in this section is to further illustrate the capabilities of SIFT for prototype matching.

Figure 12.14 shows the circuit board image we have used several times before. The small rectangle enclosing the rightmost connector on the top of the large image identifies an area from which an image of the connector was extracted. The small image is shown zoomed for clarity. The sizes of the large and small images are shown in the figure caption. Figure 12.15 shows the keypoints found by SIFT, as explained in Section 11.7. They are visible as faint lines on both images. The zoomed view of the subimage shows them a little clearer. It is important to note that the keypoints for the image and subimage were found independently by SIFT. The large image had 2714 keypoints, and the small image had 35.

Figure 12.16 shows the matches between keypoints found by SIFT. A total of 41 matches were found between the two images. Because there are only 35 keypoints

FIGURE 12.15

Keypoints found by SIFT. The large image has 2714 keypoints (visible as faint gray lines). The subimage has 35 keypoints. This is a separate image, and SIFT found its keypoints independently of the large image. The zoomed section is shown for clarity.

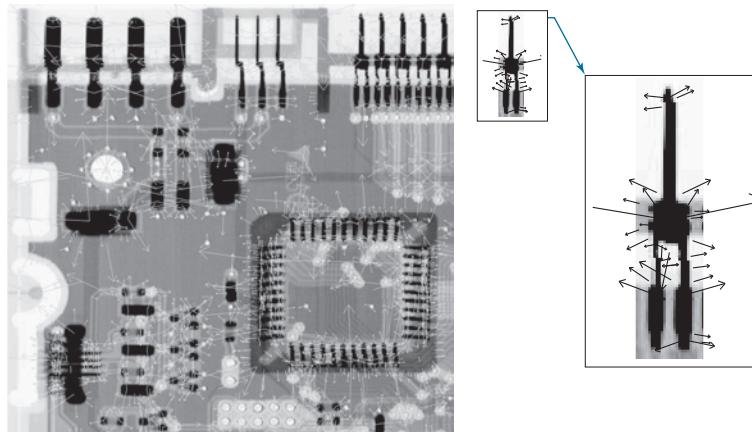
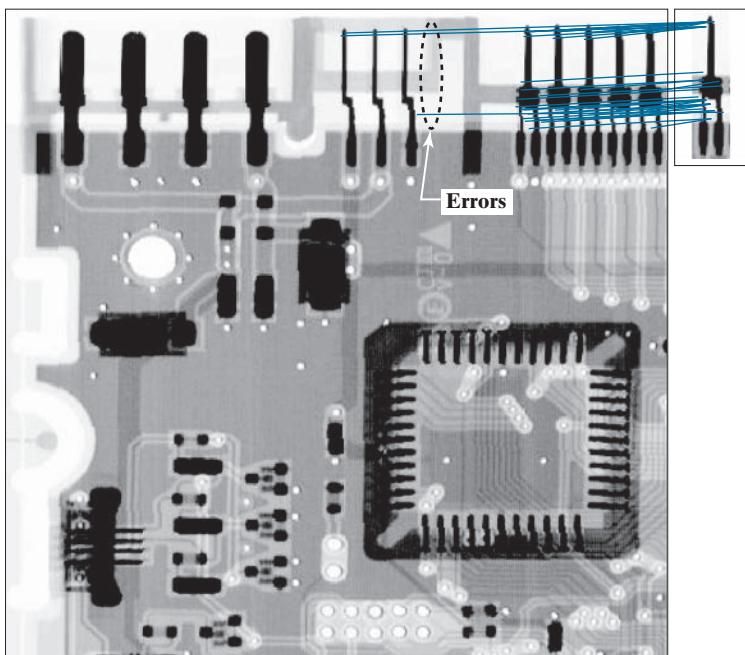


FIGURE 12.16

Matches found by SIFT between the large and small images. A total of 41 matching pairs were found. They are shown connected by straight lines. Only three of the matches were “real” errors (labeled “Errors” in the figure).



in the small image, obviously at least six matches are either incorrect, or there are multiple matches. Three of the errors are clearly visible as matches with connectors in the middle of the large image. However, if you compare the shape of the connectors in the middle of the large image, you can see that they are virtually identical to *parts* of the connectors on the right. Therefore, these errors can be explained on that basis. The other three extra matches are easier to explain. All connectors on the top right of the circuit board are identical, and we are comparing one of them against the rest. There is no way for a system to tell the difference between them. In fact, by looking at the connecting lines, we can see that the matches are between the subimage and all five connectors. These in fact are correct matches between the subimage and other connectors that are identical to it.

MATCHING STRUCTURAL PROTOTYPES

The techniques discussed up to this point deal with patterns quantitatively, and largely ignore any structural relationships inherent in pattern shapes. The methods discussed in this section seek to achieve pattern recognition by capitalizing precisely on these types of relationships. In this section, we introduce two basic approaches for the recognition of boundary shapes based on string representations, which are the most practical approach in structural pattern recognition.

Matching Shape Numbers

A procedure similar in concept to the minimum-distance classifier introduced earlier for pattern vectors can be formulated for comparing region boundaries that are

described by shape numbers. With reference to the discussion in Section 11.3, the *degree of similarity*, k , between two region boundaries, is defined as the largest order for which their shape numbers still coincide. For example, let a and b denote shape numbers of closed boundaries represented by 4-directional chain codes. These two shapes have a degree of similarity k if

Parameter j starts at 4 and is always even because we are working with 4-connectivity, and we require that boundaries be closed.

$$\begin{aligned} s_j(a) &= s_j(b) && \text{for } j = 4, 6, 8, \dots, k; \text{ and} \\ s_j(a) &\neq s_j(b) && \text{for } j = k + 2, k + 4, \dots \end{aligned} \quad (12-11)$$

where s indicates shape number, and the subscript indicates shape order. The *distance* between two shapes a and b is defined as the inverse of their degree of similarity:

$$D(a,b) = \frac{1}{k} \quad (12-12)$$

This expression satisfies the following properties:

$$\begin{aligned} D(a,b) &\geq 0 \\ D(a,b) &= 0 \quad \text{if and only if } a = b \\ D(a,c) &\leq \max[D(a,b), D(b,c)] \end{aligned} \quad (12-13)$$

Either k or D may be used to compare two shapes. If the degree of similarity is used, the larger k is, the more similar the shapes are (note that k is infinite for identical shapes). The reverse is true when Eq. (12-12) is used.

EXAMPLE 12.3: Matching shape numbers.

Suppose we have a shape, f , and want to find its closest match in a set of five shape prototypes, denoted by a, b, c, d , and e , as shown in Fig. 12.17(a). The search may be visualized with the aid of the *similarity tree* in Fig. 12.17(b). The root of the tree corresponds to the lowest possible degree of similarity, which is 4. Suppose shapes are identical up to degree 8, with the exception of shape a , whose degree of similarity with respect to all other shapes is 6. Proceeding down the tree, we find that shape d has degree of similarity 8 with respect to all others, and so on. Shapes f and c match uniquely, having a higher degree of similarity than any other two shapes. Conversely, if a had been an unknown shape, all we could have said using this method is that a was similar to the other five shapes with degree of similarity 6. The same information can be summarized in the form of the *similarity matrix* in Fig. 12.17(c).

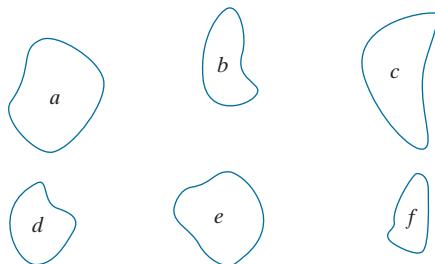
String Matching

Suppose two region boundaries, a and b , are coded into strings of symbols, denoted as $a_1a_2\dots a_n$ and $b_1b_2\dots b_m$, respectively. Let α represent the number of matches between the two strings, where a match occurs in the k th position if $a_k = b_k$. The number of symbols that do not match is

$$\beta = \max(|a|, |b|) - \alpha \quad (12-14)$$

**FIGURE 12.17**

- (a) Shapes.
(b) Similarity tree.
(c) Similarity matrix.
(Bribiesca and Guzman.)



Degree	a	b	c	d	e	f
4	∞	6	6	6	6	6
6		∞	8	8	10	8
8			∞	8	8	12
10				∞	8	8
12					∞	8
14						∞

where $|\arg|$ is the length (number of symbols) of string in the argument. It can be shown that $\beta = 0$ if and only if a and b are identical (see Problem 12.7).

An effective measure of similarity is the ratio

$$R = \frac{\alpha}{\beta} = \frac{\alpha}{\max(|a|, |b|) - \alpha} \quad (12-15)$$

We see that R is infinite for a perfect match and 0 when none of the corresponding symbols in a and b match ($\alpha = 0$ in this case). Because matching is done symbol by symbol, the starting point on each boundary is important in terms of reducing the amount of computation required to perform a match. Any method that normalizes to, or near, the same starting point is helpful if it provides a computational advantage over brute-force matching, which consists of starting at arbitrary points on each string, then shifting one of the strings (with wraparound) and computing Eq. (12-15) for each shift. The largest value of R gives the best match.

Refer to Section 11.2 for examples of how the starting point of a curve can be normalized.

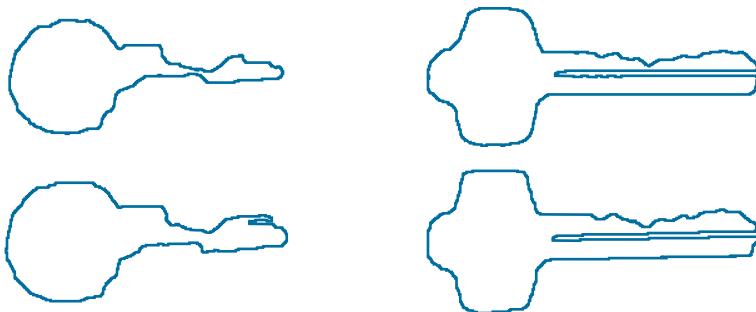
EXAMPLE 12.4: String matching.

Figures 12.18(a) and (b) show sample boundaries from each of two object classes, which were approximated by a polygonal fit (see Section 11.2). Figures 12.18(c) and (d) show the polygonal approximations

a	b
c	d
e	f
g	

FIGURE 12.18

(a) and (b) sample boundaries of two different object classes; (c) and (d) their corresponding polygonal approximations; (e)–(g) tabulations of R .
(Sze and Yang.)



R	1.a	1.b	1.c	1.d	1.e	1.f	R	2.a	2.b	2.c	2.d	2.e	2.f
1.a	∞						2.a	∞					
1.b	16.0	∞					2.b	33.5	∞				
1.c	9.6	26.3	∞				2.c	4.8	5.8	∞			
1.d	5.1	8.1	10.3	∞			2.d	3.6	4.2	19.3	∞		
1.e	4.7	7.2	10.3	14.2	∞		2.e	2.8	3.3	9.2	18.3	∞	
1.f	4.7	7.2	10.3	8.4	23.7	∞	2.f	2.6	3.0	7.7	13.5	27.0	∞

R	1.a	1.b	1.c	1.d	1.e	1.f
2.a	1.24	1.50	1.32	1.47	1.55	1.48
2.b	1.18	1.43	1.32	1.47	1.55	1.48
2.c	1.02	1.18	1.19	1.32	1.39	1.48
2.d	1.02	1.18	1.19	1.32	1.29	1.40
2.e	0.93	1.07	1.08	1.19	1.24	1.25
2.f	0.89	1.02	1.02	1.24	1.22	1.18

corresponding to the boundaries in Figs. 12.18(a) and (b), respectively. Strings were formed from the polygons by computing the interior angle, θ , between segments as each polygon was traversed clockwise. Angles were coded into one of eight possible symbols, corresponding to multiples of 45° ; that is, $\alpha_1 : 0^\circ < \theta \leq 45^\circ$; $\alpha_2 : 45^\circ < \theta \leq 90^\circ$; ...; $\alpha_8 : 315^\circ < \theta \leq 360^\circ$.

Figure 12.18(e) shows the results of computing the measure R for six samples of object 1 against themselves. The entries are values of R and, for example, the notation 1.c refers to the third string from object class 1. Figure 12.18(f) shows the results of comparing the strings of the second object class against themselves. Finally, Fig. 12.18(g) shows the R values obtained by comparing strings of one class against the other. These values of R are significantly smaller than any entry in the two preceding tabulations. This indicates that the R measure achieved a high degree of discrimination between the two classes of objects. For example, if the class of string 1.a had been unknown, the *smallest* value of R resulting from comparing this string against sample (prototype) strings of class 1 would have been 4.7 [see Fig. 12.18(e)]. By contrast, the *largest* value in comparing it against strings of class 2 would have been 1.24 [see Fig. 12.18(g)]. This result would have led to the conclusion that string 1.a is a member of object class 1. This approach to classification is analogous to the minimum-distance classifier introduced earlier.

12.4 OPTIMUM (BAYES) STATISTICAL CLASSIFIERS

In this section, we develop a probabilistic approach to pattern classification. As is true in most fields that deal with measuring and interpreting physical events, probability considerations become important in pattern recognition because of the randomness under which pattern classes normally are generated. As shown in the following discussion, it is possible to derive a classification approach that is optimal in the sense that, on average, it yields the lowest probability of committing classification errors (see Problem 12.12).

DERIVATION OF THE BAYES CLASSIFIER

The probability that a pattern vector \mathbf{x} comes from class c_i is denoted by $p(c_i/\mathbf{x})$. If the pattern classifier decides that \mathbf{x} came from class c_j when it actually came from c_i it incurs a *loss* (to be defined shortly), denoted by L_{ij} . Because pattern \mathbf{x} may belong to any one of N_c possible classes, the average loss incurred in assigning \mathbf{x} to class c_j is

$$r_j(\mathbf{x}) = \sum_{k=1}^{N_c} L_{kj} p(c_k/\mathbf{x}) \quad (12-16)$$

Quantity $r_j(\mathbf{x})$ is called the *conditional average risk* or *loss* in decision-theory terminology.

We know from Bayes' rule that $p(a/b) = [p(a)p(b/a)]/p(b)$, so we can write Eq. (12-16) as

$$r_j(\mathbf{x}) = \frac{1}{p(\mathbf{x})} \sum_{k=1}^{N_c} L_{kj} p(\mathbf{x}/c_k) P(c_k) \quad (12-17)$$

where $p(\mathbf{x}/c_k)$ is the probability density function (PDF) of the patterns from class c_k , and $P(c_k)$ is the probability of occurrence of class c_k (sometimes $P(c_k)$ is referred to as the *a priori*, or simply the *prior, probability*). Because $1/p(\mathbf{x})$ is positive and common to all the $r_j(\mathbf{x})$, $j = 1, 2, \dots, N_c$, it can be dropped from Eq. (12-17) without affecting the relative order of these functions from the smallest to the largest value. The expression for the average loss then reduces to

$$r_j(\mathbf{x}) = \sum_{k=1}^{N_c} L_{kj} p(\mathbf{x}/c_k) P(c_k) \quad (12-18)$$

Given an unknown pattern, the classifier has N_c possible classes from which to choose. If the classifier computes $r_1(\mathbf{x}), r_2(\mathbf{x}), \dots, r_{N_c}(\mathbf{x})$ for each pattern \mathbf{x} and assigns the pattern to the class with the smallest loss, the total average loss with respect to all decisions will be minimum. The classifier that minimizes the total average loss is called the *Bayes classifier*. This classifier assigns an unknown pattern \mathbf{x} to class c_i if $r_i(\mathbf{x}) < r_j(\mathbf{x})$ for $j = 1, 2, \dots, N_c$; $j \neq i$. In other words, \mathbf{x} is assigned to class c_i if

$$\sum_{k=1}^{N_c} L_{ki} p(\mathbf{x}/c_k) P(c_k) < \sum_{q=1}^{N_c} L_{qj} p(\mathbf{x}/c_q) P(c_q) \quad (12-19)$$

for all $j; j \neq i$. The loss for a correct decision generally is assigned a value of 0, and the loss for any incorrect decision usually is assigned a value of 1. Then, the loss function becomes

$$L_{ij} = 1 - \delta_{ij} \quad (12-20)$$

where $\delta_{ij} = 1$ if $i = j$, and $\delta_{ij} = 0$ if $i \neq j$. Equation (12-20) indicates a loss of unity for incorrect decisions and a loss of zero for correct decisions. Substituting Eq. (12-20) into Eq. (12-18) yields

$$\begin{aligned} r_j(\mathbf{x}) &= \sum_{k=1}^{N_c} (1 - \delta_{kj}) p(\mathbf{x}/c_k) P(c_k) \\ &= p(\mathbf{x}) - p(\mathbf{x}/c_j) P(c_j) \end{aligned} \quad (12-21)$$

The Bayes classifier then assigns a pattern \mathbf{x} to class c_i if, for all $j \neq i$,

$$p(\mathbf{x}) - p(\mathbf{x}/c_i) P(c_i) < p(\mathbf{x}) - p(\mathbf{x}/c_j) P(c_j) \quad (12-22)$$

or, equivalently, if

$$p(\mathbf{x}/c_i) P(c_i) > p(\mathbf{x}/c_j) P(c_j) \quad j = 1, 2, \dots, N_c; j \neq i \quad (12-23)$$

Thus, the Bayes classifier for a 0-1 loss function computes *decision functions* of the form

$$d_j(\mathbf{x}) = p(\mathbf{x}/c_j) P(c_j) \quad j = 1, 2, \dots, N_c \quad (12-24)$$

and assigns a pattern to class c_i if $d_i(x) > d_j(x)$ for all $j \neq i$. This is exactly the same process described in Eq. (12-5), but we are now dealing with decision functions that have been shown to be optimal in the sense that they minimize the average loss in misclassification.

For the optimality of Bayes decision functions to hold, the probability density functions of the patterns in each class, as well as the probability of occurrence of each class, must be known. The latter requirement usually is not a problem. For instance, if all classes are equally likely to occur, then $P(c_j) = 1/N_c$. Even if this condition is not true, these probabilities generally can be inferred from knowledge of the problem. Estimating the probability density functions $p(\mathbf{x}/c_j)$ is more difficult. If the pattern vectors are n -dimensional, then $p(\mathbf{x}/c_j)$ is a function of n variables. If the form of $p(\mathbf{x}/c_j)$ is not known, estimating it requires using multivariate estimation methods. These methods are difficult to apply in practice, especially if the number of representative patterns from each class is not large, or if the probability density functions are not well behaved. For these reasons, uses of the Bayes classifier often are based on assuming an analytic expression for the density functions. This in turn reduces the problem to one of estimating the necessary parameters from sample patterns from each class using training patterns. By far, the most prevalent form assumed for $p(\mathbf{x}/c_j)$ is the Gaussian probability density function. The closer this assumption is to reality, the closer the Bayes classifier approaches the minimum average loss in classification.

BAYES CLASSIFIER FOR GAUSSIAN PATTERN CLASSES

You may find it helpful to review the tutorial on probability available in the book website.

To begin, let us consider a 1-D problem ($n = 1$) involving two pattern classes ($N_c = 2$) governed by Gaussian densities, with means m_1 and m_2 , and standard deviations σ_1 and σ_2 , respectively. From Eq. (12-24) the Bayes decision functions have the form

$$\begin{aligned} d_j(x) &= p(x/c_j)P(c_j) \\ &= \frac{1}{\sqrt{2\pi}\sigma_j} e^{-\frac{(x-m_j)^2}{2\sigma_j^2}} P(c_j) \quad j = 1, 2 \end{aligned} \quad (12-25)$$

where the patterns are now scalars, denoted by x . Figure 12.19 shows a plot of the probability density functions for the two classes. The boundary between the two classes is a single point, x_0 , such that $d_1(x_0) = d_2(x_0)$. If the two classes are equally likely to occur, then $P(c_1) = P(c_2) = 1/2$, and the decision boundary is the value of x_0 for which $p(x_0/c_1) = p(x_0/c_2)$. This point is the intersection of the two probability density functions, as shown in Fig. 12.19. Any pattern (point) to the right of x_0 is classified as belonging to class c_1 . Similarly, any pattern to the left of x_0 is classified as belonging to class c_2 . When the classes are not equally likely to occur, x_0 moves to the left if class c_1 is more likely to occur or, conversely, it moves to the right if class c_2 is more likely to occur. This result is to be expected, because the classifier is trying to minimize the loss of misclassification. For instance, in the extreme case, if class c_2 never occurs, the classifier would never make a mistake by always assigning all patterns to class c_1 (that is, x_0 would move to negative infinity).

In the n -dimensional case, the Gaussian density of the vectors in the j th pattern class has the form

$$p(\mathbf{x}/c_j) = \frac{1}{(2\pi)^{n/2} |\mathbf{C}_j|^{1/2}} e^{-\frac{1}{2}(\mathbf{x} - \mathbf{m}_j)^T \mathbf{C}_j^{-1} (\mathbf{x} - \mathbf{m}_j)} \quad (12-26)$$

where each density is specified completely by its mean vector \mathbf{m}_j and covariance matrix \mathbf{C}_j , which are defined as

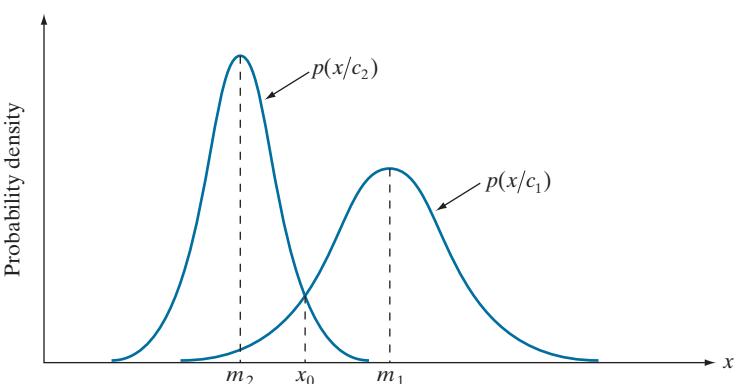


FIGURE 12.19
Probability density functions for two 1-D pattern classes. Point x_0 (at the intersection of the two curves) is the Bayes decision boundary if the two classes are equally likely to occur.

$$\mathbf{m}_j = E_j \{ \mathbf{x} \} \quad (12-27)$$

and

$$\mathbf{C}_j = E_j \left\{ (\mathbf{x} - \mathbf{m}_j)(\mathbf{x} - \mathbf{m}_j)^T \right\} \quad (12-28)$$

where $E_j \{ \cdot \}$ is the expected value of the argument over the patterns of class c_j . In Eq. (12-26), n is the dimensionality of the pattern vectors, and $|\mathbf{C}_j|$ is the determinant of matrix \mathbf{C}_j . Approximating the expected value E_j by the sample average yields an estimate of the mean vector and covariance matrix:

$$\mathbf{m}_j = \frac{1}{n_j} \sum_{\mathbf{x} \in c_j} \mathbf{x} \quad (12-29)$$

and

$$\mathbf{C}_j = \frac{1}{n_j} \sum_{\mathbf{x} \in c_j} \mathbf{x} \mathbf{x}^T - \mathbf{m}_j \mathbf{m}_j^T \quad (12-30)$$

where n_j is the number of sample pattern vectors from class c_j and the summation is taken over these vectors. We will give an example later in this section of how to use these two expressions.

The covariance matrix is symmetric and positive semidefinite. Its k th diagonal element is the variance of the k th element of the pattern vectors. The kj th off-diagonal matrix element is the covariance of elements x_k and x_j in these vectors. The multivariate Gaussian density function reduces to the product of the univariate Gaussian density of each element of \mathbf{x} when the off-diagonal elements of the covariance matrix are zero, which happens when the vector elements x_k and x_j are uncorrelated.

From Eq. (12-24), the Bayes decision function for class c_j is $d_j(\mathbf{x}) = p(\mathbf{x}/c_j)P(c_j)$. However, the exponential form of the Gaussian density allows us to work with the natural logarithm of this decision function, which is more convenient. In other words, we can use the form

$$\begin{aligned} d_j(\mathbf{x}) &= \ln \left[p(\mathbf{x}/c_j)P(c_j) \right] \\ &= \ln p(\mathbf{x}/c_j) + \ln P(c_j) \end{aligned} \quad (12-31)$$

This expression is equivalent to Eq. (12-24) in terms of classification performance because the logarithm is a monotonically increasing function. That is, the numerical order of the decision functions in Eqs. (12-24) and (12-31) is the same. Substituting Eq. (12-26) into Eq. (12-31) yields

$$d_j(\mathbf{x}) = \ln P(c_j) - \frac{n}{2} \ln 2\pi - \frac{1}{2} \ln |\mathbf{C}_j| - \frac{1}{2} \left[(\mathbf{x} - \mathbf{m}_j)^T \mathbf{C}_j^{-1} (\mathbf{x} - \mathbf{m}_j) \right] \quad (12-32)$$

As noted in Section 6.7 [see Eq. (6-49)], the square root of the rightmost term in this equation is called the *Mahalanobis distance*.

The term $(n/2)\ln 2\pi$ is the same for all classes, so it can be eliminated from Eq. (12-32), which then becomes

$$d_j(\mathbf{x}) = \ln P(c_j) - \frac{1}{2} \ln |\mathbf{C}_j| - \frac{1}{2} \left[(\mathbf{x} - \mathbf{m}_j)^T \mathbf{C}_j^{-1} (\mathbf{x} - \mathbf{m}_j) \right] \quad (12-33)$$

for $j = 1, 2, \dots, N_c$. This equation gives the Bayes decision functions for Gaussian pattern classes under the condition of a 0-1 loss function.

The decision functions in Eq. (12-33) are hyperquadrics (quadratic functions in n -dimensional space), because no terms higher than the second degree in the components of \mathbf{x} appear in the equation. Clearly, then, the best that a Bayes classifier for Gaussian patterns can do is to place a second-order decision boundary between each pair of pattern classes. If the pattern populations are truly Gaussian, no other boundary would yield a lesser average loss in classification.

If all covariance matrices are equal, then $\mathbf{C}_j = \mathbf{C}$ for $j = 1, 2, \dots, N_c$. By expanding Eq. (12-33), and dropping all terms that do not depend on j , we obtain

$$d_j(\mathbf{x}) = \ln P(c_j) + \mathbf{x}^T \mathbf{C}^{-1} \mathbf{m}_j - \frac{1}{2} \mathbf{m}_j^T \mathbf{C}^{-1} \mathbf{m}_j \quad (12-34)$$

which are linear decision functions (hyperplanes) for $j = 1, 2, \dots, N_c$.

If, in addition, $\mathbf{C} = \mathbf{I}$, where \mathbf{I} is the identity matrix, and also if the classes are equally likely (i.e., $P(c_j) = 1/N_c$ for all j), then we can drop the term $\ln P(c_j)$ because it would be the same for all values of j . Equation (12-34) then becomes

$$d_j(\mathbf{x}) = \mathbf{m}_j^T \mathbf{x} - \frac{1}{2} \mathbf{m}_j^T \mathbf{m}_j \quad j = 1, 2, \dots, N_c \quad (12-35)$$

which we recognize as the decision functions for a minimum-distance classifier [see Eq. (12-4)]. Thus, as mentioned earlier, the minimum-distance classifier is optimum in the Bayes sense if (1) the pattern classes follow a Gaussian distribution, (2) all covariance matrices are equal to the identity matrix, and (3) all classes are equally likely to occur. Gaussian pattern classes satisfying these conditions are spherical clouds of identical shape in n dimensions (called *hyperspheres*). The minimum-distance classifier establishes a hyperplane between every pair of classes, with the property that the hyperplane is the perpendicular bisector of the line segment joining the center of the pair of hyperspheres. In 2-D, the patterns are distributed in circular regions, and the boundaries become lines that bisect the line segment joining the center of every pair of such circles.

EXAMPLE 12.5: A Bayes classifier for 3-D patterns.

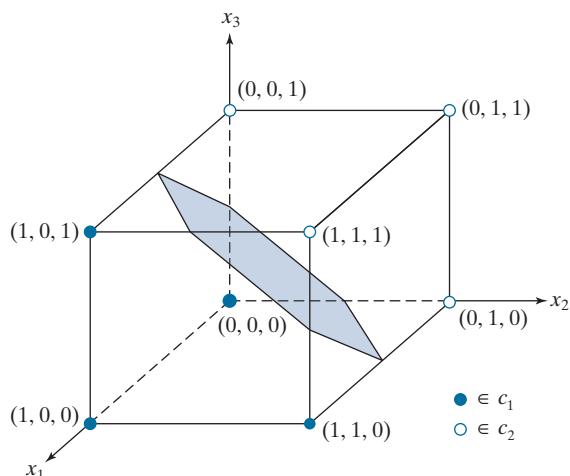
We illustrate the mechanics of the preceding development using the simple patterns in Fig. 12.20. We assume that the patterns are samples from two Gaussian populations, and that the classes are equally likely to occur. Applying Eq. (12-29) to the patterns in the figure results in

$$\mathbf{m}_1 = \frac{1}{3} \begin{bmatrix} 3 \\ 1 \\ 1 \end{bmatrix} \text{ and } \mathbf{m}_2 = \frac{1}{3} \begin{bmatrix} 1 \\ 3 \\ 3 \end{bmatrix}$$

And, from Eq. (12-30),

FIGURE 12.20

Two simple pattern classes and the portion of their Bayes decision boundary (shaded) that intersects the cube.



$$\mathbf{C}_1 = \mathbf{C}_2 = \frac{1}{16} \begin{bmatrix} 3 & 1 & 1 \\ 1 & 3 & -1 \\ 1 & -1 & 3 \end{bmatrix}$$

The inverse of this matrix is

$$\mathbf{C}_1^{-1} = \mathbf{C}_2^{-1} = \begin{bmatrix} 8 & -4 & -4 \\ -4 & 8 & 4 \\ -4 & 4 & 8 \end{bmatrix}$$

Next, we obtain the decision functions. Equation (12-34) applies because the covariance matrices are equal, and we are assuming that the classes are equally likely:

$$d_j(\mathbf{x}) = \mathbf{x}^T \mathbf{C}^{-1} \mathbf{m}_j - \frac{1}{2} \mathbf{m}_j^T \mathbf{C}^{-1} \mathbf{m}_j$$

Carrying out the vector-matrix expansion, we obtain the two decision functions:

$$d_1(\mathbf{x}) = 4x_1 - 1.5 \quad \text{and} \quad d_2(\mathbf{x}) = -4x_1 + 8x_2 + 8x_3 - 5.5$$

The decision boundary separating the two classes is then

$$d_1(\mathbf{x}) - d_2(\mathbf{x}) = 8x_1 - 8x_2 - 8x_3 + 4 = 0$$

Figure 12.20 shows a section of this planar surface. Note that the classes were separated effectively.

EXAMPLE 12.6: Classification of multispectral data using a Bayes classifier.

As discussed in Sections 1.3 and 11.5, a multispectral scanner responds to selected bands of the electromagnetic energy spectrum, such as the bands: 0.45–0.52, 0.53–0.61, 0.63–0.69, and 0.78–0.90 microns. These ranges are in the visible blue, visible green, visible red, and near infrared bands, respectively. A region on the ground scanned using these multispectral bands produces four digital images of the region,

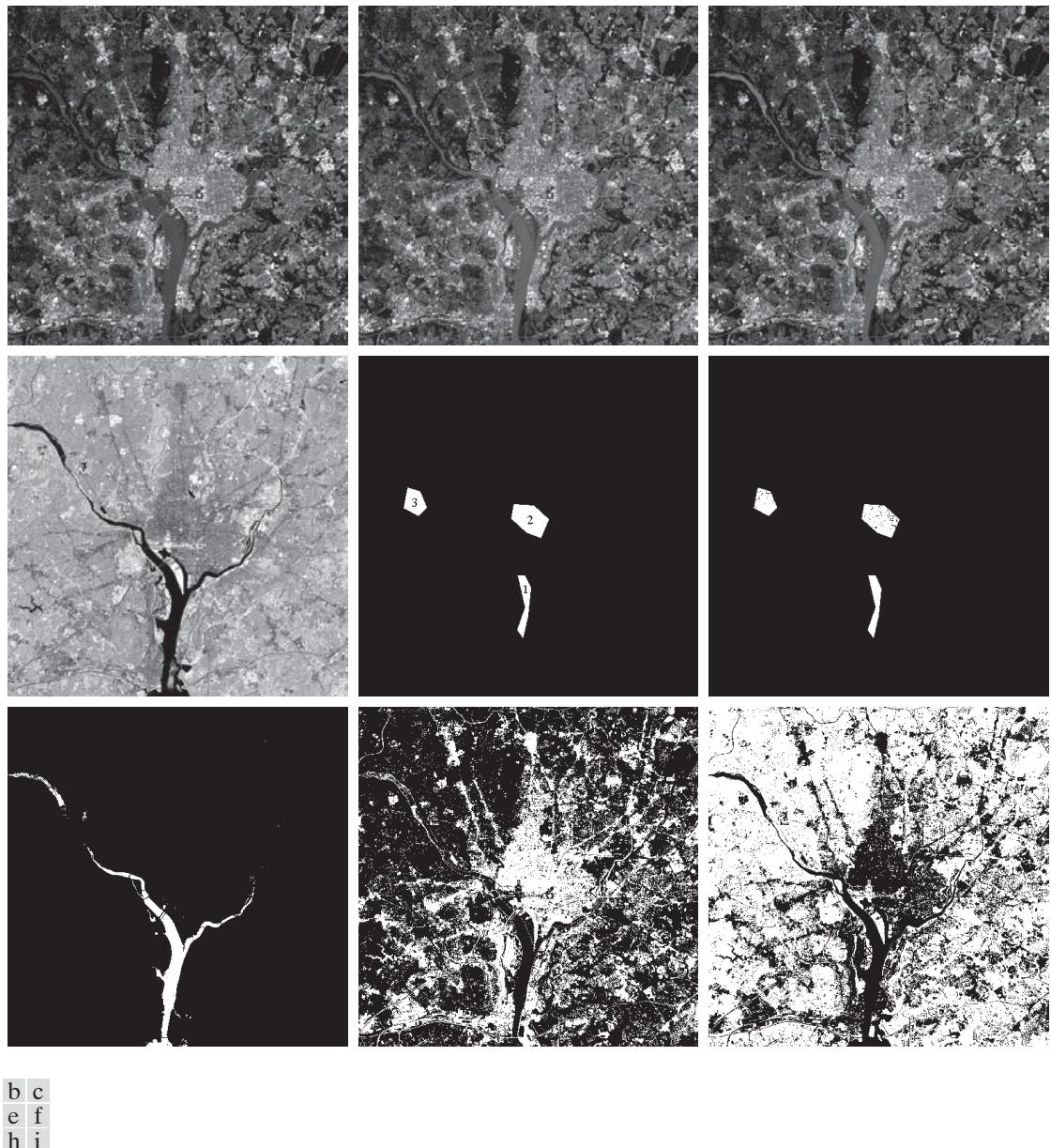
one for each band. If the images are registered spatially, they can be visualized as being stacked one behind the other, as illustrated in Fig. 12.7. As we explained in that figure, every point on the ground in this example can be represented by a 4-D pattern vector of the form $\mathbf{x} = (x_1, x_2, x_3, x_4)^T$, where x_1 is a shade of blue, x_2 a shade of green, and so on. If the images are of size 512×512 pixels, each stack of four multispectral images can be represented by 266,144 four-dimensional pattern vectors. As noted previously, the Bayes classifier for Gaussian patterns requires estimates of the mean vector and covariance matrix for each class. In remote sensing applications, these estimates are obtained using training multispectral data whose classes are known from each region of interest (this knowledge sometimes is referred to as *ground truth*). The resulting vectors are then used to estimate the required mean vectors and covariance matrices, as in Example 12.5.

Figures 12.21(a) through (d) show four 512×512 multispectral images of the Washington, D.C. area, taken in the bands mentioned in the previous paragraph. We are interested in classifying the pixels in these images into one of three pattern classes: *water*, *urban development*, or *vegetation*. The masks in Fig. 12.21(e) were superimposed on the images to extract samples representative of these three classes. Half of the samples were used for training (i.e., for estimating the mean vectors and covariance matrices), and the other half were used for independent testing to assess classifier performance. We assume that the a priori probabilities are equal, $P(c_j) = 1/3$; $j = 1, 2, 3$.

Table 12.1 summarizes the classification results we obtained with the training and test data sets. The percentage of training and test pattern vectors recognized correctly was about the same with both data sets, indicating that the learned parameters did not over-fit the parameters to the training data. The largest error in both cases was with patterns from the urban area. This is not unexpected, as vegetation is present there also (note that no patterns in the vegetation or urban areas were misclassified as water). Figure 12.21(f) shows as black dots the training and test patterns that were misclassified, and as white dots the patterns that were classified correctly. No black dots are visible in region 1, because the seven misclassified points are very close to the boundary of the white region. You can compute from the numbers in the table that the correct recognition rate was 96.4% for the training patterns, and 96.1% for the test patterns.

Figures 12.21(g) through (i) are more interesting. Here, we let the system classify *all* image pixels into one of the three categories. Figure 12.21(g) shows in white all pixels that were classified as water. Pixels not classified as water are shown in black. We see that the Bayes classifier did an excellent job of determining which parts of the image were water. Figure 12.21(h) shows in white all pixels classified as urban development; observe how well the system performed in recognizing urban features, such as the bridges and highways. Figure 12.21(i) shows the pixels classified as vegetation. The center area in Fig. 12.21(h) shows a high concentration of white pixels in the downtown area, with the density decreasing as a function of distance from the center of the image. Figure 12.21(i) shows the opposite effect, indicating the least vegetation toward the center of the image, where urban development is the densest.

We mentioned in Section 10.3 when discussing Otsu's method that thresholding may be viewed as a Bayes classification problem, which optimally assigns patterns to two or more classes. In fact, as the previous example shows, pixel-by-pixel classification may be viewed as a segmentation that partitions an image into two or more possible types of regions. If only one single variable (e.g., intensity) is used, then Eq. (12-24) becomes an optimum function that similarly partitions an image based on the intensity of its pixels, as we did in Section 10.3. Keep in mind that optimality requires that the PDF and a priori probability of each class be known. As we



a	b	c
d	e	f
g	h	i

FIGURE 12.21 Bayes classification of multispectral data. (a)–(d) Images in the visible blue, visible green, visible red, and near infrared wavelength bands. (e) Masks for regions of water (labeled 1), urban development (labeled 2), and vegetation (labeled 3). (f) Results of classification; the black dots denote points classified incorrectly. The other (white) points were classified correctly. (g) All image pixels classified as water (in white). (h) All image pixels classified as urban development (in white). (i) All image pixels classified as vegetation (in white).

TABLE 12.1

Bayes classification of multispectral image data. Classes 1, 2, and 3 are water, urban, and vegetation, respectively.

Training Patterns							Test Patterns				
Class	No. of Samples	Classified into Class			% Correct	Class	No. of Samples	Classified into Class			% Correct
		1	2	3				1	2	3	
1	484	482	2	0	99.6	1	483	478	3	2	98.9
2	933	0	885	48	94.9	2	932	0	880	52	94.4
3	483	0	19	464	96.1	3	482	0	16	466	96.7

have mentioned previously, estimating these densities is not a trivial task. If assumptions have to be made (e.g., as in assuming Gaussian densities), then the degree of optimality achieved in classification depends on how close the assumptions are to reality.

12.5 NEURAL NETWORKS AND DEEP LEARNING

The principal objectives of the material in this section and in Section 12.6 are to present an introduction to deep neural networks, and to derive the equations that are the foundation of deep learning. We will discuss two types of networks. In this section, we focus attention on multilayer, fully connected neural networks, whose inputs are pattern vectors of the form introduced in Section 12.2. In Section 12.6, we will discuss convolutional neural networks, which are capable of accepting images as inputs. We follow the same basic approach in presenting the material in these two sections. That is, we begin by developing the equations that describe how an input is mapped through the networks to generate the outputs that are used to classify that input. Then, we derive the equations of backpropagation, which are the tools used to train both types of networks. We give examples in both sections that illustrate the power of deep neural networks and deep learning for solving complex pattern classification problems.

BACKGROUND

The essence of the material that follows is the use of a multitude of elemental non-linear computing elements (called *artificial neurons*), organized as networks whose interconnections are similar in some respects to the way in which neurons are interconnected in the visual cortex of mammals. The resulting models are referred to by various names, including *neural networks*, *neurocomputers*, *parallel distributed processing models*, *neuromorphic systems*, *layered self-adaptive networks*, and *connectionist models*. Here, we use the name *neural networks*, or *neural nets* for short. We use these networks as vehicles for adaptively learning the parameters of decision functions via successive presentations of training patterns.

Interest in neural networks dates back to the early 1940s, as exemplified by the work of McCulloch and Pitts [1943], who proposed neuron models in the form of

binary thresholding devices, and stochastic algorithms involving sudden 0–1 and 1–0 changes of states, as the basis for modeling neural systems. Subsequent work by Hebb [1949] was based on mathematical models that attempted to capture the concept of learning by reinforcement or association.

During the mid-1950s and early 1960s, a class of so-called *learning machines* originated by Rosenblatt [1959, 1962] caused a great deal of excitement among researchers and practitioners of pattern recognition. The reason for the interest in these machines, called *perceptrons*, was the development of mathematical proofs showing that perceptrons, when trained with linearly separable training sets (i.e., training sets separable by a hyperplane), would converge to a solution in a finite number of iterative steps. The solution took the form of parameters (coefficients) of hyperplanes that were capable of correctly separating the classes represented by patterns of the training set.

Unfortunately, the expectations following discovery of what appeared to be a well-founded theoretical model of learning soon met with disappointment. The basic perceptron, and some of its generalizations, were inadequate for most pattern recognition tasks of practical significance. Subsequent attempts to extend the power of perceptron-like machines by considering multiple layers of these devices lacked effective training algorithms, such as those that had created interest in the perceptron itself. The state of the field of learning machines in the mid-1960s was summarized by Nilsson [1965]. A few years later, Minsky and Papert [1969] presented a discouraging analysis of the limitation of perceptron-like machines. This view was held as late as the mid-1980s, as evidenced by comments made by Simon [1986]. In this work, originally published in French in 1984, Simon dismisses the perceptron under the heading “Birth and Death of a Myth.”

More recent results by Rumelhart, Hinton, and Williams [1986] dealing with the development of new training algorithms for multilayers of perceptron-like units have changed matters considerably. Their basic method, called *backpropagation* (*backprop* for short), provides an effective training method for multilayer networks. Although this training algorithm cannot be shown to converge to a solution in the sense of the proof for the single-layer perceptron, backpropagation is capable of generating results that have revolutionized the field of pattern recognition.

The approaches to pattern recognition we have studied up to this point rely on human-engineered techniques to transform raw data into formats suitable for computer processing. The methods of feature extraction we studied in Chapter 11 are examples of this. Unlike these approaches, neural networks can use backpropagation to automatically learn representations suitable for recognition, starting with raw data. Each layer in the network “refines” the representation into more abstract levels. This type of multilayered learning is commonly referred to as *deep learning*, and this capability is one of the underlying reasons why applications of neural networks have been so successful. As we noted at the beginning of this section, practical implementations of deep learning generally are associated with large data sets.

Of course, these are not “magical” systems that assemble themselves. Human intervention is still required for specifying parameters such as the number of layers, the number of artificial neurons per layer, and various coefficients that are problem

dependent. Teaching proper recognition to a complex multilayer neural network is not a science; rather, it is an art that requires considerable knowledge and experimentation on the part of the designer. Countless applications of pattern recognition, especially in constrained environments, are best handled by more “traditional” methods. A good example of this is stylized font recognition. It would be senseless to develop a neural network to recognize the E-13B font we studied in Fig. 12.11. A minimum-distance classifier implemented on a hard-wired architecture is the ideal solution to this problem, provided that interest is limited to reading only the E-13B font printed on bank checks. On the other hand, neural networks have proved to be the ideal solution if the scope of application is expanded to require that all relevant text written on checks, including cursive script, be read with high accuracy.

Deep learning has shined in applications that defy other methods of solution. In the two decades following the introduction of backpropagation, neural networks have been used successfully in a broad range of applications. Some of them, such as speech recognition, have become an integral part of everyday life. When you speak into a smart phone, the nearly flawless recognition is performed by a neural network. This type of performance was unachievable just a few years ago. Other applications from which you benefit, perhaps without realizing it, are smart filters that learn user preferences for rerouting spam and other junk mail from email accounts, and the systems that read zip codes on postal mail. Often, you see television clips of vehicles navigating autonomously, and robots that are capable of interacting with their environment. Most are solutions based on neural networks. Less familiar applications include the automated discovery of new medicines, the prediction of gene mutations in DNA research, and advances in natural language understanding.

Although the list of practical uses of neural nets is long, applications of this technology in image pattern classification has been slower in gaining popularity. As you will learn shortly, using neural nets in image processing is based principally on neural network architectures called *convolutional neural nets* (denoted by *CNNs* or *ConvNets*). One of the earliest well-known applications of CNNs is the work of LeCun et al. [1989] for reading handwritten U.S. postal zip codes. A number of other applications followed shortly thereafter, but it was not until the results of the 2012 ImageNet Challenge were published (e.g., see Krizhevsky, Sutskever, and Hinton [2012]) that CNNs became widely used in image pattern recognition. Today, this is the approach of choice for addressing complex image recognition tasks.

The neural network literature is vast and rapidly evolving, so as usual, our approach is to focus on fundamentals. In this and the following sections, we will establish the foundation of how neural nets are trained, and how they operate after training. We will begin by briefly discussing perceptrons. Although these computing elements are not used per se in current neural network architectures, the operations they perform are almost identical to artificial neurons, which are the basic computing units of neural nets. In fact, an introduction to neural networks would be incomplete without a discussion of perceptrons. We will follow this discussion by developing in detail the theoretical foundation of backpropagation. After developing the basic backpropagation equations, we will recast them in matrix form, which

reduces the training and operation of neural nets to a simple, straightforward cascade of matrix multiplications.

After studying several examples of fully connected neural nets, we will follow a similar approach in developing the foundation of CNNs, including how they differ from fully connected neural nets, and how their training is different. This is followed by several examples of how CNNs are used for image pattern classification.

THE PERCEPTRON

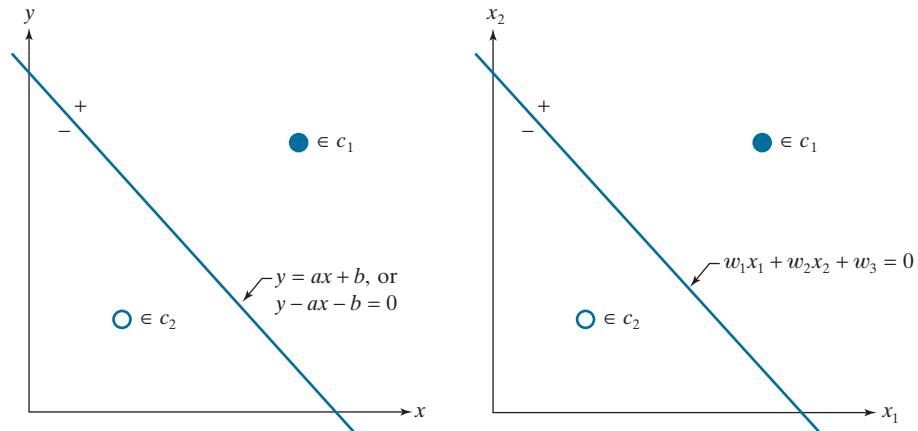
A single perceptron unit learns a linear boundary between two linearly separable pattern classes. Figure 12.22(a) shows the simplest possible example in two dimensions: two pattern classes, consisting of a single pattern each. A linear boundary in 2-D is a straight line with equation $y = ax + b$, where coefficient a is the *slope* and b is the *y-intercept*. Note that if $b = 0$, the line goes through the origin. Therefore, the function of parameter b is to displace the line from the origin without affecting its slope. For this reason, this “floating” coefficient that is not multiplied by a coordinate is often referred to as the *bias*, the *bias coefficient*, or the *bias weight*.

We are interested in a line that separates the two classes in Fig. 12.22. This is a line positioned in such a way that pattern (x_1, y_1) from class c_1 lies on one side of the line, and pattern (x_2, y_2) from class c_2 lies on the other. The locus of points (x, y) that are *on* the line, satisfy the equation $y - ax - b = 0$. It then follows that any point on one side of the line would yield a positive value when its coordinates are plugged into this equation, and conversely for a point on the other side.

Generally, we work with patterns in much higher dimensions than two, so we need more general notation. Points in n dimensions are vectors. The components of a vector, x_1, x_2, \dots, x_n , are the coordinates of the point. For the coefficients of the boundary separating the two classes, we use the notation $w_1, w_2, \dots, w_n, w_{n+1}$, where w_{n+1} is the bias. The general equation of our line using this notation is $w_1x_1 + w_2x_2 + w_3 = 0$ (we can express this equation in slope-intercept form as $x_2 + (w_1/w_2)x_1 + w_3/w_2 = 0$). Figure 12.22(b) is the same as (a), but using this notation. Comparing the two figures, we see that $y = x_2$, $x = x_1$, $a = w_1/w_2$, and $b = w_3/w_2$. Equipped with our more

FIGURE 12.22

- (a) The simplest two-class example in 2-D, showing one possible decision boundary out of an infinite number of such boundaries.
 (b) Same as (a), but with the decision boundary expressed using more general notation.



general notation, we say that an arbitrary point (x_1, x_2) is on the positive side of a line if $w_1x_1 + w_2x_2 + w_3 > 0$, and conversely for any point on the negative side. For points in 3-D, we work with the equation of a plane, $w_1x_1 + w_2x_2 + w_3x_3 + w_4 = 0$, but would perform exactly the same test to see if a point lies on the positive or negative side of the plane. For a point in n dimensions, the test would be against a *hyperplane*, whose equation is

$$w_1x_1 + w_2x_2 + \cdots + w_nx_n + w_{n+1} = 0 \quad (12-36)$$

This equation is expressed in summation form as

$$\sum_{i=1}^n w_i x_i + w_{n+1} = 0 \quad (12-37)$$

or in vector form as

$$\mathbf{w}^T \mathbf{x} + w_{n+1} = 0 \quad (12-38)$$

where \mathbf{w} and \mathbf{x} are n -dimensional *column* vectors and $\mathbf{w}^T \mathbf{x}$ is the dot (inner) product of the two vectors. Because the inner product is commutative, we can express Eq. (12-38) in the equivalent form $\mathbf{x}^T \mathbf{w} + w_{n+1} = 0$. We refer to \mathbf{w} as a *weight vector* and, as above, to w_{n+1} as a *bias*. Because the bias is a weight that is always multiplied by 1, sometimes we avoid repetition by using the term *weights, coefficients, or parameters* when referring to the bias and the elements of a weight vector collectively.

Stating the class separation problem in general form we say that, given any pattern vector \mathbf{x} from a vector population, we want to find a set of weights with the property

$$\mathbf{w}^T \mathbf{x} + w_{n+1} = \begin{cases} > 0 & \text{if } \mathbf{x} \in c_1 \\ < 0 & \text{if } \mathbf{x} \in c_2 \end{cases} \quad (12-39)$$

Finding a line that separates two *linearly separable* pattern classes in 2-D can be done by inspection. Finding a separating plane by visual inspection of 3-D data is more difficult, but it is doable. For $n > 3$, finding a separating hyperplane by inspection becomes impossible in general. We have to resort instead to an algorithm to find a solution. The perceptron is an implementation of such an algorithm. It attempts to find a solution by iteratively stepping through the patterns of each of two classes. It starts with an arbitrary weight vector and bias, and is guaranteed to converge in a finite number of iterations if the classes are linearly separable.

The perceptron algorithm is simple. Let $\alpha > 0$ denote a *correction increment* (also called the *learning increment* or the *learning rate*), let $\mathbf{w}(1)$ be a vector with arbitrary values, and let $w_{n+1}(1)$ be an arbitrary constant. Then, do the following for $k = 2, 3, \dots$: For a pattern vector, $\mathbf{x}(k)$, at step k ,

- 1)** If $\mathbf{x}(k) \in c_1$ and $\mathbf{w}^T(k)\mathbf{x}(k) + w_{n+1}(k) \leq 0$, let

$$\begin{aligned} \mathbf{w}(k+1) &= \mathbf{w}(k) + \alpha \mathbf{x}(k) \\ w_{n+1}(k+1) &= w_{n+1}(k) + \alpha \end{aligned} \quad (12-40)$$

It is customary to associate $>$ with class c_1 and $<$ with class c_2 , but the sense of the inequality is arbitrary, provided that you are consistent. Note that this equation implements a *linear decision function*.

Linearly separable classes satisfy Eq. (12-39). That is, they are separable by single hyperplanes.

2) If $\mathbf{x}(k) \in c_2$ and $\mathbf{w}^T(k)\mathbf{x}(k) + w_{n+1}(k) \geq 0$, let

$$\begin{aligned}\mathbf{w}(k+1) &= \mathbf{w}(k) - \alpha\mathbf{x}(k) \\ w_{n+1}(k+1) &= w_{n+1}(k) - \alpha\end{aligned}\quad (12-41)$$

3) Otherwise, let

$$\begin{aligned}\mathbf{w}(k+1) &= \mathbf{w}(k) \\ w_{n+1}(k+1) &= w_{n+1}(k)\end{aligned}\quad (12-42)$$

The correction in Eq. (12-40) is applied when the pattern is from class c_1 and Eq. (12-39) does not give a positive response. Similarly, the correction in Eq. (12-41) is applied when the pattern is from class c_2 and Eq. (12-39) does not give a negative response. As Eq. (12-42) shows, no change is made when Eq. (12-39) gives the correct response.

The notation in Eqs. (12-40) through (12-42) can be simplified if we add a 1 at the end of every pattern vector and include the bias in the weight vector. That is, we define $\mathbf{x} \triangleq [x_1, x_2, \dots, x_n, 1]^T$ and $\mathbf{w} \triangleq [w_1, w_2, \dots, w_n, w_{n+1}]^T$. Then, Eq. (12-39) becomes

$$\mathbf{w}^T \mathbf{x} = \begin{cases} > 0 & \text{if } \mathbf{x} \in c_1 \\ < 0 & \text{if } \mathbf{x} \in c_2 \end{cases} \quad (12-43)$$

where both vectors are now $(n+1)$ -dimensional. In this formulation, \mathbf{x} and \mathbf{w} are referred to as *augmented* pattern and weight vectors, respectively. The algorithm in Eqs. (12-40) through (12-42) then becomes: For any pattern vector, $\mathbf{x}(k)$, at step k

1') If $\mathbf{x}(k) \in c_1$ and $\mathbf{w}^T(k)\mathbf{x}(k) \leq 0$, let

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \alpha\mathbf{x}(k) \quad (12-44)$$

2') If $\mathbf{x}(k) \in c_2$ and $\mathbf{w}^T(k)\mathbf{x}(k) \geq 0$, let

$$\mathbf{w}(k+1) = \mathbf{w}(k) - \alpha\mathbf{x}(k) \quad (12-45)$$

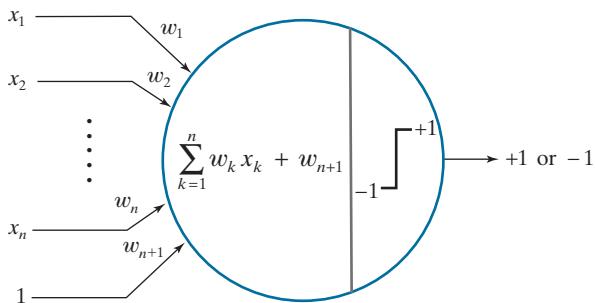
3') Otherwise, let

$$\mathbf{w}(k+1) = \mathbf{w}(k) \quad (12-46)$$

where the starting weight vector, $\mathbf{w}(1)$, is arbitrary and, as above, α is a positive constant. The procedure implemented by Eqs. (12-40)–(12-42) or (12-44)–(12-46) is called the *perceptron training algorithm*. The *perceptron convergence theorem* states that the algorithm is guaranteed to converge to a solution (i.e., a separating hyperplane) in a finite number of steps if the two pattern classes are linearly separable (see Problem 12.15). Normally, Eqs. (12-44)–(12-46) are the basis for implementing the perceptron training algorithm, and we will use it in the following paragraphs of this section. However, the notation in Eqs. (12-40)–(12-42), in which the bias is

FIGURE 12.23

Schematic of a perceptron, showing the operations it performs.



shown separately, is more prevalent in neural networks, so you need to be familiar with it as well.

Figure 12.23 shows a schematic diagram of the perceptron. As you can see, all this simple “machine” does is form a *sum of products* of an input pattern using the weights and bias found during training. The output of this operation is a scalar value that is then passed through an *activation function* to produce the unit’s output. For the perceptron, the activation function is a thresholding function (we will consider other forms of activation when we discuss neural networks). If the thresholded output is a +1, we say that the pattern belongs to class c_1 . Otherwise, a -1 indicates that the pattern belongs to class c_2 . Values 1 and 0 sometimes are used to denote the two possible states of the output.

Note that the perceptron model implements Eq. (12-39), which is in the form of a decision function.

EXAMPLE 12.7: Using the perceptron algorithm to learn a decision boundary.

We illustrate the steps taken by a perceptron in learning the coefficients of a linear boundary by solving the mini problem in Fig. 12.22. To simplify manual computations, let the pattern vector furthest from the origin be $\mathbf{x} = [3 \ 3 \ 1]^T$, and the other be $\mathbf{x} = [1 \ 1 \ 1]^T$, where we augmented the vectors by appending a 1 at the end, as discussed earlier. To match the figure, let these two patterns belong to classes c_1 and c_2 , respectively. Also, assume the patterns are “cycled” through the perceptron in that order during training (one complete iteration through all patterns of the training is called an *epoch*). To start, we let $\alpha = 1$ and $\mathbf{w}(1) = \mathbf{0} = [0 \ 0 \ 0]^T$; then,

For $k = 1$, $\mathbf{x}(1) = [3 \ 3 \ 1]^T \in c_1$, and $\mathbf{w}(1) = [0 \ 0 \ 0]^T$. Their inner product is zero,

$$\mathbf{w}^T(1)\mathbf{x}(1) = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 3 \\ 3 \\ 1 \end{bmatrix} = 0$$

so Step 1' of the second version of the training algorithm applies:

$$\mathbf{w}(2) = \mathbf{w}(1) + \alpha\mathbf{x}(1) = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} + (1) \begin{bmatrix} 3 \\ 3 \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \\ 3 \\ 1 \end{bmatrix}$$

For $k = 2$, $\mathbf{x}(2) = [1 \ 1 \ 1]^T \in c_2$ and $\mathbf{w}(2) = [3 \ 3 \ 1]^T$. Their inner product is

$$\mathbf{w}^T(2)\mathbf{x}(2) = [3 \ 3 \ 1] \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = 7$$

The result is positive when it should have been negative, so Step 2' applies:

$$\mathbf{w}(3) = \mathbf{w}(2) - \alpha \mathbf{x}(2) = \begin{bmatrix} 3 \\ 3 \\ 1 \end{bmatrix} - (1) \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \\ 0 \end{bmatrix}$$

We have gone through a complete training epoch with at least one correction, so we cycle through the training set again.

For $k = 3$, $\mathbf{x}(3) = [3 \ 3 \ 1]^T \in c_1$, and $\mathbf{w}(3) = [2 \ 2 \ 0]^T$. Their inner product is positive (i.e., 6) as it should be because $\mathbf{x}(3) \in c_1$. Therefore, Step 3' applies and the weight vector is not changed:

$$\mathbf{w}(4) = \mathbf{w}(3) = \begin{bmatrix} 2 \\ 2 \\ 0 \end{bmatrix}$$

For $k = 4$, $\mathbf{x}(4) = [1 \ 1 \ 1]^T \in c_2$, and $\mathbf{w}(4) = [2 \ 2 \ 0]^T$. Their inner product is positive (i.e., 4) and it should have been negative, so Step 2' applies:

$$\mathbf{w}(5) = \mathbf{w}(4) - \alpha \mathbf{x}(4) = \begin{bmatrix} 2 \\ 2 \\ 0 \end{bmatrix} - (1) \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}$$

At least one correction was made, so we cycle through the training patterns again. For $k = 5$, we have $\mathbf{x}(5) = [3 \ 3 \ 1]^T \in c_1$, and, using $\mathbf{w}(5)$, we compute their inner product to be 5. This is positive as it should be, so Step 3' applies and we let $\mathbf{w}(6) = \mathbf{w}(5) = [1 \ 1 \ -1]^T$. Following this procedure just discussed, you can show (see Problem 12.13) that the algorithm converges to the solution weight vector

$$\mathbf{w} = \mathbf{w}(12) = \begin{bmatrix} 1 \\ 1 \\ -3 \end{bmatrix}$$

which gives the decision boundary

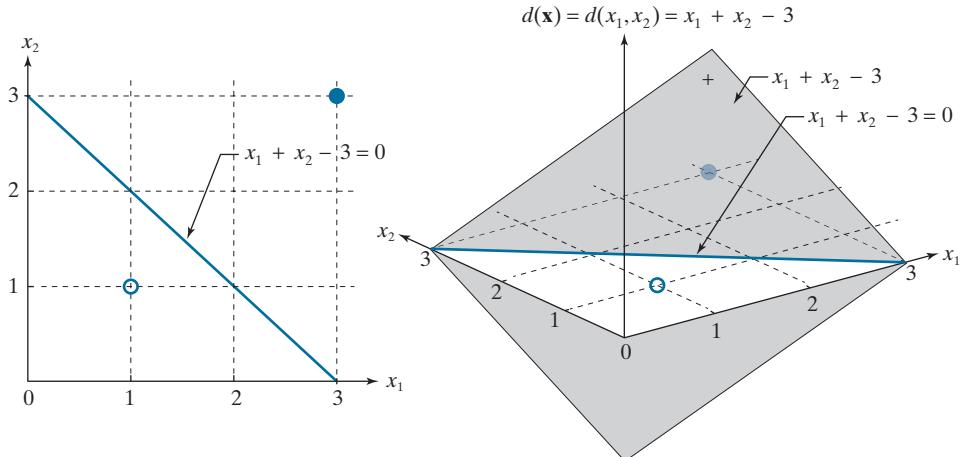
$$x_1 + x_2 - 3 = 0$$

Figure 12.24(a) shows the boundary defined by this equation. As you can see, it clearly separates the patterns of the two classes. In terms of the terminology we used in the previous section, the *decision surface* learned by the perceptron is $d(\mathbf{x}) = d(x_1, x_2) = x_1 + x_2 - 3$, which is a plane. As before, the *decision boundary* is the locus of points such that $d(\mathbf{x}) = d(x_1, x_2) = 0$, which is a line. Another way to visualize this boundary is that it is the intersection of the decision surface (a plane) with the $x_1 x_2$ -plane, as Fig. 12.24(b) shows. All points (x_1, x_2) such that $d(x_1, x_2) > 0$ are on the positive side of the boundary, and vice versa for $d(x_1, x_2) < 0$.

a b

FIGURE 12.24

- (a) Segment of the decision boundary learned by the perceptron algorithm.
 (b) Section of the decision surface. The decision boundary is the intersection of the decision surface with the x_1x_2 -plane.

**EXAMPLE 12.8: Using the perceptron to classify two sets of iris data measurements.**

In Fig. 12.10 we showed a reduced set of the iris database in two dimensions, and mentioned that the only class that was separable from the others is the class of Iris setosa. As another illustration of the perceptron, we now find the full decision boundary between the Iris setosa and the Iris versicolor classes. As we mentioned when discussing Fig. 12.10, these are 4-D data sets. Letting $\alpha = 0.5$, and starting with all parameters equal to zero, the perceptron converged in only four epochs to the solution weight vector $\mathbf{w} = [0.65, 2.05, -2.60, -1.10, 0.50]^T$, where the last element is w_{n+1} .

In practice, linearly separable pattern classes are rare, and a significant amount of research effort during the 1960s and 1970s went into developing techniques for dealing with nonseparable pattern classes. With recent advances in neural networks, many of those methods have become items of mere historical interest, and we will not dwell on them here. However, we mention briefly one approach because it is relevant to the discussion of neural networks in the next section. The method is based on minimizing the error between the actual and desired response at any training step.

Let r denote the response we want the perceptron to have for any pattern during training. The output of our perceptron is either $+1$ or -1 , so these are the two possible values that r can have. We want to find the augmented weight vector, \mathbf{w} , that minimizes the mean squared error (MSE) between the desired and actual responses of the perceptron. The function should be differentiable and have a unique minimum. The function of choice for this purpose is a quadratic of the form

$$E(\mathbf{w}) = \frac{1}{2}(r - \mathbf{w}^T \mathbf{x})^2 \quad (12-47)$$

The $1/2$ is used to cancel out the 2 that will result from taking the derivative of this expression. Also, remember that $\mathbf{w}^T \mathbf{x}$ is a scalar.

where E is our error measure, \mathbf{w} is the weight vector we are seeking, \mathbf{x} is any pattern from the training set, and r is the response we desire for that pattern. Both \mathbf{w} and \mathbf{x} are augmented vectors.

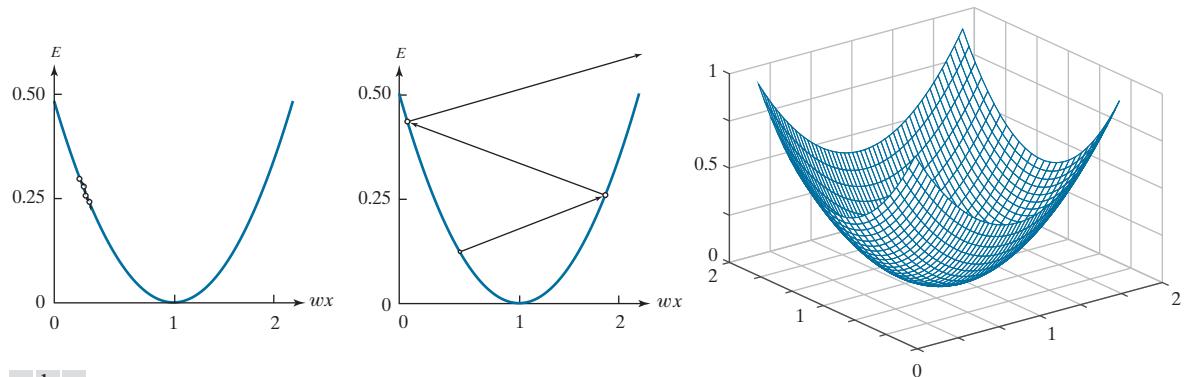


FIGURE 12.25 Plots of E as a function of wx for $r = 1$. (a) A value of α that is too small can slow down convergence. (b) If α is too large, large oscillations or divergence may occur. (c) Shape of the error function in 2-D.

We find the minimum of $E(\mathbf{w})$ using an iterative gradient descent algorithm, whose form is

Note that the right side of this equation is the gradient of $E(\mathbf{w})$.

$$\mathbf{w}(k+1) = \mathbf{w}(k) - \alpha \left[\frac{\partial E(\mathbf{w})}{\partial \mathbf{w}} \right]_{\mathbf{w}=\mathbf{w}(k)} \quad (12-48)$$

where the starting weight vector is arbitrary, and $\alpha > 0$.

Figure 12.25(a) shows a plot of E for scalar values, w and x , of \mathbf{w} and \mathbf{x} . We want to move w incrementally so $E(w)$ approaches a minimum, which implies that E should stop changing or, equivalently, that $\partial E(w)/\partial w = 0$. Equation (12-48) does precisely this. If $\partial E(w)/\partial w > 0$, a portion of this quantity (determined by the value of the learning increment α) is subtracted from $w(k)$ to create a new, updated value $w(k+1)$, of the weight. The opposite happens if $\partial E(w)/\partial w < 0$. If $\partial E(w)/\partial w = 0$, the weight is unchanged, meaning that we have arrived at a minimum, which is the solution we are seeking. The value of α determines the relative magnitude of the correction in weight value. If α is too small, the step changes will be correspondingly small and the weight would move slowly toward convergence, as Fig. 12.25(a) illustrates. On the other hand, choosing α too large could cause large oscillations on either side of the minimum, or even become unstable, as Fig. 12.25(b) illustrates. There is no general rule for choosing α . However, a logical approach is to start small and experiment by increasing α to determine its influence on a particular set of training patterns. Figure 12.25(c) shows the shape of the error function for two variables.

Because the error function is given analytically and it is differentiable, we can express Eq. (12-48) in a form that does not require computing the gradient explicitly at every step. The partial of $E(\mathbf{w})$ with respect to \mathbf{w} is

$$\frac{\partial E(\mathbf{w})}{\partial \mathbf{w}} = -(r - \mathbf{w}^T \mathbf{x}) \mathbf{x} \quad (12-49)$$

Substituting this result into Eq. (12-48) yields

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \alpha [r(k) - \mathbf{w}^T(k)\mathbf{x}(k)]\mathbf{x}(k) \quad (12-50)$$

which is in terms of known or easily computable terms. As before, $\mathbf{w}(1)$ is arbitrary.

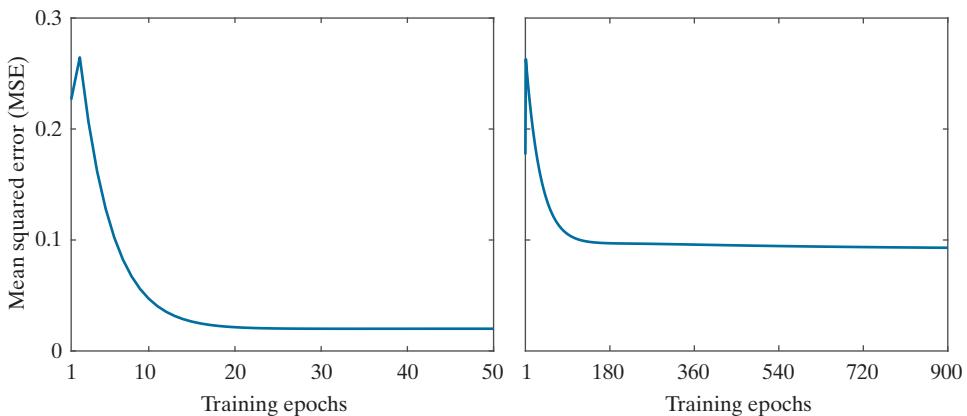
Widrow and Stearns [1985] have shown that it is necessary (but not sufficient) for α to be in the range $0 < \alpha < 2$ for the algorithm in Eq. (12-50) to converge. A typical range for α is $0.1 < \alpha < 1.0$. Although the proof is not shown here, the algorithm converges to a solution that minimizes the mean squared error over the patterns of the training set. For this reason, the algorithm is often referred to as the *least-mean-squared-error* (LMSE) algorithm. In practice, we say that the algorithm has converged when the error decreases below a specified threshold. The solution at convergence may not be a hyperplane that fully partitions two linearly separable classes. That is, a mean-square-error solution does not imply a solution in the sense of the perceptron training theorem. This uncertainty is the price of using an algorithm whose convergence is independent of the linear separability of the pattern classes.

EXAMPLE 12.9: Using the LMSE algorithm.

It will be interesting to compare the performance of the LMSE algorithm using the same set of separable iris data as in Example 12.8. Figure 12.26(a) is a plot of the error [Eq. (12-47)] as a function of epoch for 50 epochs, using Eq. (12-50) (with $\alpha = 0.001$) to obtain the weights (we started with $\mathbf{w}(1) = \mathbf{0}$). Each epoch of training consisted of sequentially updating the weights, one pattern at a time, and computing Eq. (12-47) for each weight and the corresponding pattern. At the end of the epoch, the errors were added and divided by 100 (the total number of patterns) to obtain the mean squared error (MSE). This yielded one point of the curve of Fig. 12.26(a). After increasing and then decreasing rapidly, no appreciable difference in error occurred after about 20 epochs. For example, the error at the end of the 50th epoch was 0.02 and, at the end of 1,000 epochs, it was 0.0192. Getting smaller error values is possible by further decreasing α , but at the expense of slower decay in the error, as noted in Fig. 12.25. Keep in mind also that MSE is not directly proportional to correct recognition rate.

a b

FIGURE 12.26
MSE as a function of epoch for:
(a) the linearly separable Iris classes (setosa and versicolor); and (b) the linearly nonseparable Iris classes (versicolor and virginica).



The weight vector at the end of 50 epochs of training was $\mathbf{w} = [0.098 \ 0.357 \ -0.548 \ -0.255 \ 0.075]^T$. All patterns were classified correctly into their two respective classes using this vector. That is, although the MSE did not become zero, the resulting weight vector was able to classify all the patterns correctly. But keep in mind that the LMSE algorithm does not always achieve 100% correct recognition of linearly separable classes.

As noted earlier, only the Iris setosa samples are linearly separable from the others. But the Iris versicolor and virginica samples are not. The perceptron algorithm would not converge when presented with these data, whereas the LMSE algorithm does. Figure 12.26(b) is the MSE as a function of training epoch for these two data sets, obtained using the same values for $\mathbf{w}(1)$ and α as in (a). This time, it took 900 epochs for the MSE to stabilize at 0.09, which is much higher than before. The resulting weight vector was $\mathbf{w} = [0.534 \ 0.584 \ -0.878 \ -1.028 \ 0.651]^T$. Using this vector resulted in seven misclassification errors out of 100 patterns, giving a recognition rate of 93%.

A classic example used to show the limitations of single linear decision boundaries (and hence single perceptron units) is the XOR classification problem. The table in Fig. 12.27(a) shows the definition of the XOR operator for two variables. As you can see, the XOR operation produces a logical true (1) value when either of the variables (but not both) is true; otherwise, the result is false (0). The XOR two-class pattern classification problem is set up by letting each pair of values A and B be a point in 2-D space, and letting the true (1) XOR values define one class, and the false (0) values define the other. In this case, we assigned the class c_1 label to patterns $\{(0, 0), (1, 1)\}$, and the c_2 label to patterns $\{(1, 0), (0, 1)\}$. A classifier capable of solving the XOR problem must respond with a value, say, 1, when a pattern from class c_1 is presented, and a different value, say, 0 or -1 , when the input pattern is from class c_2 . You can tell by inspection of Fig. 12.27(b) that a single linear decision boundary (a straight line) cannot separate the two classes correctly. This means that we cannot solve the problem with a single perceptron. The simplest linear boundary consists of two straight lines, as Fig. 12.27(b) shows. A more complex, nonlinear, boundary capable of solving the problem is a quadratic function, as in Fig. 12.27(c).

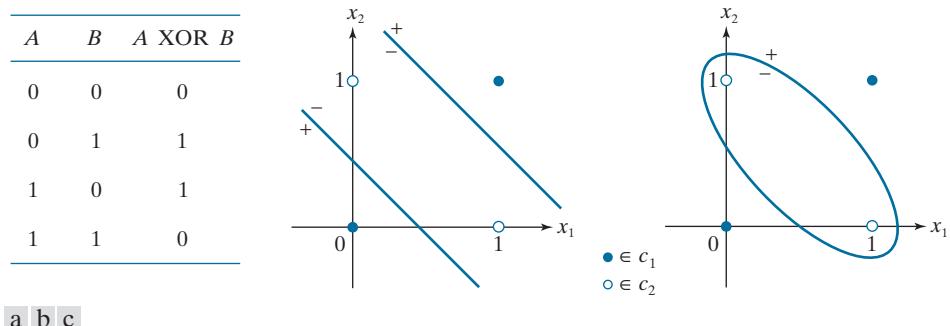
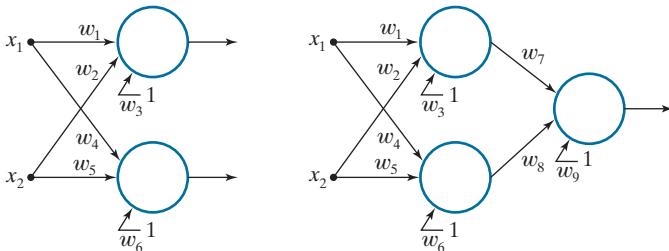


FIGURE 12.27 The XOR classification problem in 2-D. (a) Truth table definition of the XOR operator. (b) 2-D pattern classes formed by assigning the XOR truth values (1) to one pattern class, and false values (0) to another. The simplest decision boundary between the two classes consists of two straight lines. (c) Nonlinear (quadratic) boundary separating the two classes.

a b

FIGURE 12.28

(a) Minimum perceptron solution to the XOR problem in 2-D. (b) A solution that implements the XOR truth table in Fig. 12.27(a).



Natural questions at this point are: Can more than one perceptron solve the XOR problem? If so, what is the minimum number of units required? We know that a single perceptron can implement one straight line, and we need to implement two lines, so the obvious answers are: yes to the first question, and two units to the second. Figure 12.28(a) shows the solution for two variables, which requires a total of six coefficients because we need two lines. The solution coefficients are such that, for either of the two patterns from class c_1 , one output is true (1) and the other is false (0). The opposite condition must hold for either pattern from class c_2 . This solution requires that we analyze two outputs. If we want to implement the truth table, meaning that a single output should give the same response as the XOR function [the third column in Fig. 12.27(a)], then we need one additional perceptron. Figure 12.28(b) shows the architecture for this solution. Here, one perceptron in the first layer maps any input from one class into a 1, and the other perceptron maps a pattern from the other class into a 0. This reduces the four possible inputs into two outputs, which is a two-point problem. As you know from Fig. 12.24, a single perceptron can solve this problem. Therefore, we need three perceptrons to implement the XOR table, as in Fig. 12.28(b).

With a little work, we could determine by inspection the coefficients needed to implement either solution in Fig. 12.28. However, rather than dwell on that, we focus attention in the following section on a more general, layered architecture, of which the XOR solution is a trivial, special case.

MULTILAYER FEEDFORWARD NEURAL NETWORKS

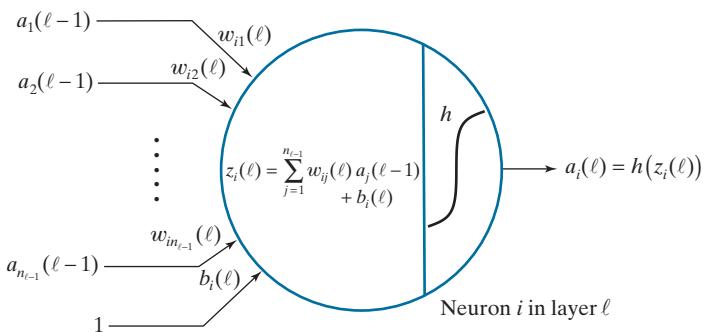
In this section, we discuss the architecture and operation of multilayer neural networks, and derive the equations of backpropagation used to train them. We then give several examples illustrating the capabilities of neural nets

Model of an Artificial Neuron

Neural networks are interconnected perceptron-like computing elements called *artificial neurons*. These neurons perform the same computations as the perceptron, but they differ from the latter in how they process the result of the computations. As illustrated in Fig. 12.23, the perceptron uses a “hard” thresholding function that outputs two values, such as +1 and -1, to perform classification. Suppose that in a network of perceptrons, the output before thresholding of one of the perceptrons is infinitesimally greater than zero. When thresholded, this very small signal will be turned into a +1. But a similarly small signal with the opposite sign would cause

FIGURE 12.29

Model of an artificial neuron, showing all the operations it performs. The “ ℓ ” is used to denote a particular layer in a layered network.



a large swing in value from +1 to -1. Neural networks are formed from layers of computing units, in which the output of one unit affects the behavior of all units following it. The perceptron’s sensitivity to the sign of small signals can cause serious stability problems in an interconnected system of such units, making perceptrons unsuitable for layered architectures.

The solution is to change the characteristic of the activation function from a hard-limiter to a smooth function. Figure 12.29 shows an example based on using the activation function

$$h(z) = \frac{1}{1 + e^{-z}} \quad (12-51)$$

where z is the result of the computation performed by the neuron, as shown in Fig. 12.29. Except for more complicated notation, and the use of a smooth function rather than a hard threshold, this model performs the same *sum-of-products* operations as in Eq. (12-36) for the perceptron. Note that the *bias* term is denoted by b instead

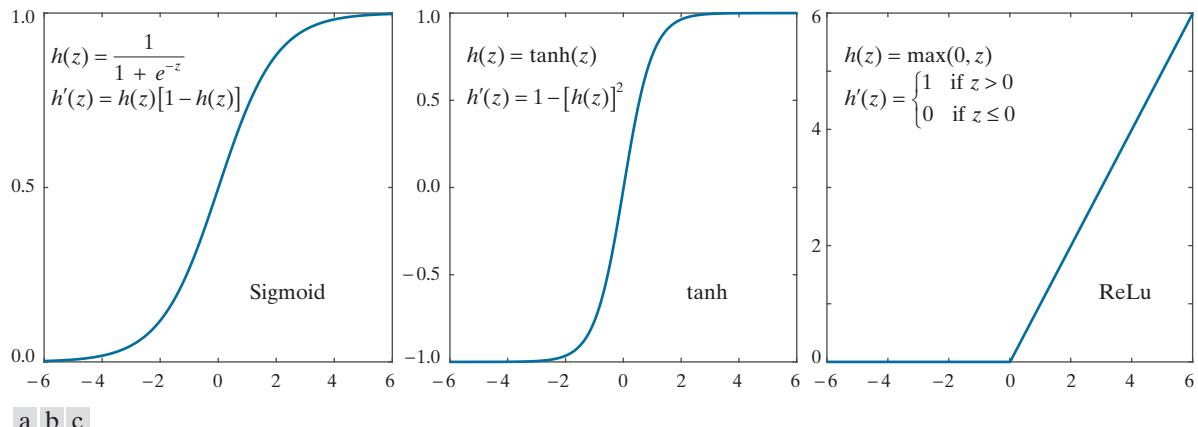


FIGURE 12.30 Various activation functions. (a) Sigmoid. (b) Hyperbolic tangent (also has a sigmoid shape, but it is centered about 0 in both dimensions). (c) Rectifier linear unit (ReLU).

of w_{n+1} , as we do the perceptron. It is customary to use different notation, typically b , in neural networks to denote the bias term, so we are following convention. The more complicated notation used in Fig. 12.29, which we will explain shortly, is needed because we will be dealing with multilayer arrangements with several neurons per layer. We use the symbol “ ℓ ” to denote layers.

As you can see by comparing Figs. 12.29 and 12.23, we use variable z to denote the sum-of-products computed by the neuron. The output of the unit, denoted by a , is obtained by passing z through h . We call h the *activation function*, and refer to its output, $a = h(z)$, as the *activation value* of the unit. Note in Fig. 12.29 that the inputs to a neuron are activation values from neurons in the previous layer. Figure 12.30(a) shows a plot of $h(z)$ from Eq. (12-51). Because this function has the shape of a sigmoid function, the unit in Fig. 12.29 is sometimes called an *artificial sigmoid neuron*, or simply a *sigmoid neuron*. Its derivative has a very nice form, expressible in terms of $h(z)$ [see Problem 12.16(a)]:

$$h'(z) = \frac{\partial h(z)}{\partial z} = h(z)[1 - h(z)] \quad (12-52)$$

Figures 12.30(b) and (c) show two other forms of $h(z)$ used frequently. The hyperbolic tangent also has the shape of a sigmoid function, but it is symmetric about both axes. This property can help improve the convergence of the backpropagation algorithm to be discussed later. The function in Fig. 12.30(c) is called the *rectifier function*, and a unit using it is referred to a *rectifier linear unit* (ReLU). Often, you see the function itself referred to as the ReLU *activation function*. Experimental results suggest that this function tends to outperform the other two in deep neural networks.

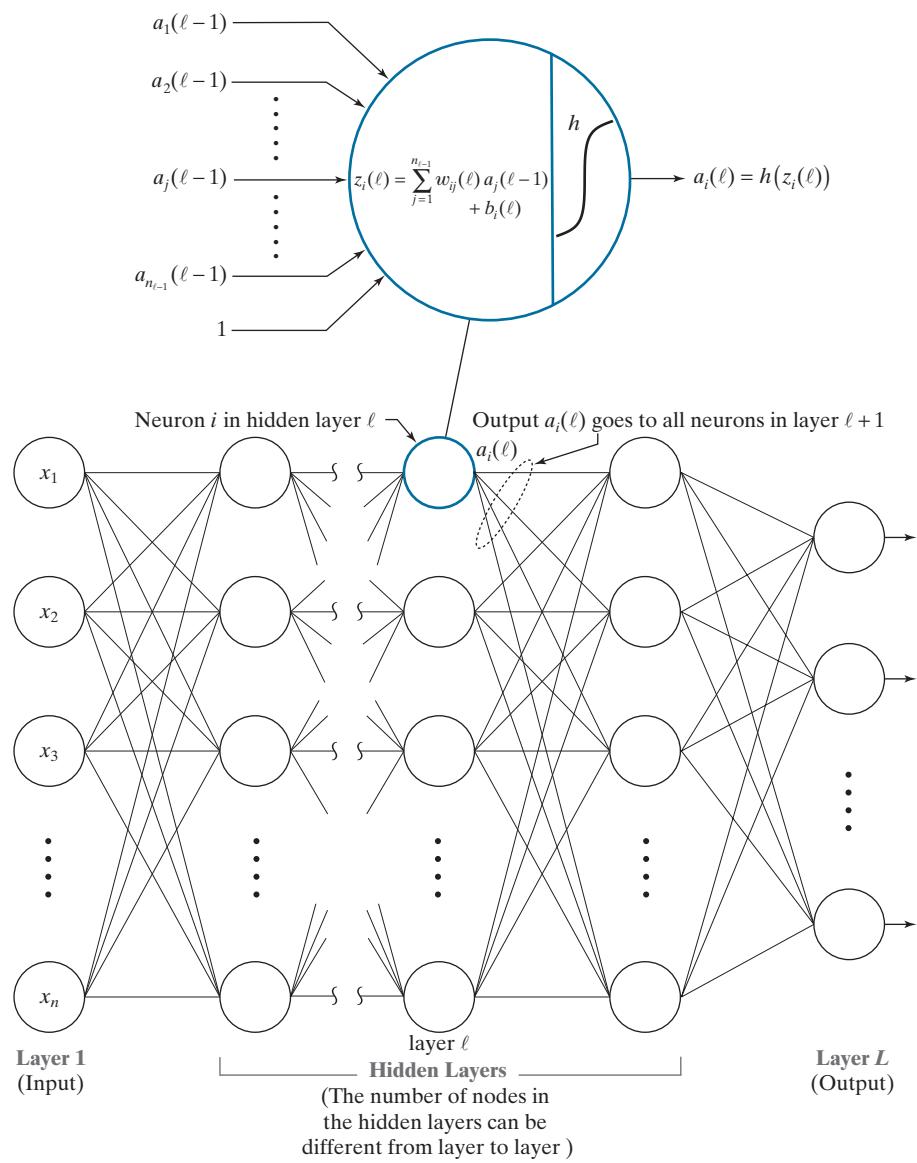
Interconnecting Neurons to Form a Fully Connected Neural Network

Figure 12.31 shows a generic diagram of a multilayer neural network. A *layer* in the network is the set of nodes (neurons) in a column of the network. As indicated by the zoomed node in Fig. 12.31, all the nodes in the network are artificial neurons of the form shown in Fig. 12.29, except for the input layer, whose nodes are the components of an input pattern vector \mathbf{x} . Therefore, the outputs (activation values) of the first layer are the values of the elements of \mathbf{x} . The outputs of all other nodes are the activation values of neurons in a particular layer. Each layer in the network can have a different number of nodes, but each node has a *single* output. The multiple lines shown at the outputs of the neurons in Fig. 12.31 indicate that the output of every node is connected to the input of all nodes in the next layer, to form a *fully connected* network. We also require that there be no loops in the network. Such networks are called *feedforward networks*. Fully connected, feedforward neural nets are the only types of networks considered in this section.

We obviously know the values of the nodes in the first layer, and we can observe the values of the output neurons. All others are *hidden neurons*, and the layers that contain them are called *hidden layers*. Generally, we call a neural net with a single hidden layer a *shallow neural network*, and refer to network with two or more hidden layers as a *deep neural network*. However, this terminology is not universal, and

FIGURE 12.31

General model of a feedforward, fully connected neural net. The neuron is the same as in Fig. 12.29. Note how the output of each neuron goes to the input of all neurons in the following layer, hence the name *fully connected* for this type of architecture.



sometimes you will see the words “shallow” and “deep” used subjectively to denote networks with a “few” and with “many” layers, respectively.

We used the notation in Eq. (12-37) to label all the inputs and weights of a perceptron. In a neural network, the notation is more complicated because we have to account for neuron weights, inputs, and outputs within a layer, and also from layer to layer. Ignoring layer notation for a moment, we denote by w_{ij} the weight that associates the link connecting the *output* of neuron j to the *input* of neuron i . That is,

the first subscript denotes the neuron that *receives* the signal, and the second refers to the neuron that *sends* the signal. Because i precedes j alphabetically, it would seem to make more sense for i to send and for j to receive. The reason we use the notation as stated is to avoid a matrix transposition in the equation that describes propagation of signals through the network. This notation is convention, but there is no doubt that it is confusing, so special care is necessary to keep the notation straight.

Remember, a bias is a weight that is always multiplied by 1.

Because the biases depend only on the neuron containing it, a single subscript that associates a bias with a neuron is sufficient. For example, we use b_i to denote the bias value associated with the i th neuron in a given layer of the network. Our use of b instead of w_{n+1} (as we did for perceptrons) follows notational convention used in neural networks. The weights, biases, and activation function(s) completely define a neural network. Although the activation function of any neuron in a neural network could be different from the others, there is no convincing evidence to suggest that there is anything to be gained by doing so. We assume in all subsequent discussions that the same form of activation function is used in all neurons.

Let ℓ denote a layer in the network, for $\ell = 1, 2, \dots, L$. With reference to Fig. 12.31, $\ell = 1$ denotes the input layer, $\ell = L$ is the output layer, and all other values of ℓ denote hidden layers. The number of neurons in layer ℓ is denoted n_ℓ . We have two options to include layer indexing in the parameters of a neural network. We can do it as a superscript, for example, w_{ij}^ℓ and b_i^ℓ ; or we can use the notation $w_{ij}(\ell)$ and $b_i(\ell)$. The first option is more prevalent in the literature on neural network. We use the second option because it is more consistent with the way we describe iterative expressions in the book, and also because you may find it easier to follow. Using this notation, the output (activation value) of neuron k in layer ℓ is denoted $a_k(\ell)$.

Keep in mind that our objective in using neural networks is the same as for perceptrons: to determine the class membership of unknown input patterns. The most common way to perform pattern classification using a neural network is to assign a class label to each output neuron. Thus, a neural network with n_L outputs can classify an unknown pattern into one of n_L classes. The network assigns an unknown pattern vector \mathbf{x} to class c_k if output neuron k has the largest activation value; that is, if $a_k(L) > a_j(L)$, $j = 1, 2, \dots, n_L$; $j \neq k$.[†]

In this and the following section, the number of outputs of our neural networks will always equal the number of classes. But this is not a requirement. For instance, a network for classifying two pattern classes could be structured with a single output (Problem 12.17 illustrates such a case) because all we need for this task is two states, and a single neuron is capable of that. For three and four classes, we need three and four states, respectively, which can be achieved with two output neurons. Of course, the problem with this approach is that we would need additional logic to decipher the output combinations. It is simply more practical to have one neuron per output, and let the neuron with the highest output value determine the class of the input.

[†] Instead of a sigmoid or similar function in the final output layer, you will sometimes see a *softmax function* used instead. The concept is the same as we explained earlier, but the activation values in a softmax implementation are given by $a_i(L) = \exp[z_i(L)] / \sum_{j,k} \exp[z_j(L)]$, where the summation is over all outputs. In this formulation, the sum of all activations is 1, thus giving the outputs a probabilistic interpretation.

FORWARD PASS THROUGH A FEEDFORWARD NEURAL NETWORK

A *forward pass* through a neural network maps the input layer (i.e., values of \mathbf{x}) to the output layer. The values in the output layer are used for determining the class of an input vector. The equations developed in this section explain how a feedforward neural network carries out the computations that result in its output. Implicit in the discussion in this section is that the network parameters (weights and biases) are known. The important results in this section will be summarized in Table 12.2 at the end of our discussion, but understanding the material that gets us there is important when we discuss training of neural nets in the next section.

The Equations of a Forward Pass

The outputs of the layer 1 are the components of input vector \mathbf{x} :

$$a_j(1) = x_j \quad j = 1, 2, \dots, n_1 \quad (12-53)$$

where $n_1 = n$ is the dimensionality of \mathbf{x} . As illustrated in Figs. 12.29 and 12.31, the computation performed by neuron i in layer ℓ is given by

$$z_i(\ell) = \sum_{j=1}^{n_{\ell-1}} w_{ij}(\ell) a_j(\ell-1) + b_i(\ell) \quad (12-54)$$

for $i = 1, 2, \dots, n_\ell$ and $\ell = 2, \dots, L$. Quantity $z_i(\ell)$ is called the *net* (or *total*) *input* to neuron i in layer ℓ , and is sometimes denoted by net_i . The reason for this terminology is that $z_i(\ell)$ is formed using *all* outputs from layer $\ell - 1$. The output (activation value) of neuron i in layer ℓ is given by

$$a_i(\ell) = h(z_i(\ell)) \quad i = 1, 2, \dots, n_\ell \quad (12-55)$$

where h is an activation function. The value of network *output* node i is

$$a_i(L) = h(z_i(L)) \quad i = 1, 2, \dots, n_L \quad (12-56)$$

Equations (12-53) through (12-56) describe all the operations required to map the input of a fully connected feedforward network to its output.

EXAMPLE 12.10: Illustration of a forward pass through a fully connected neural network.

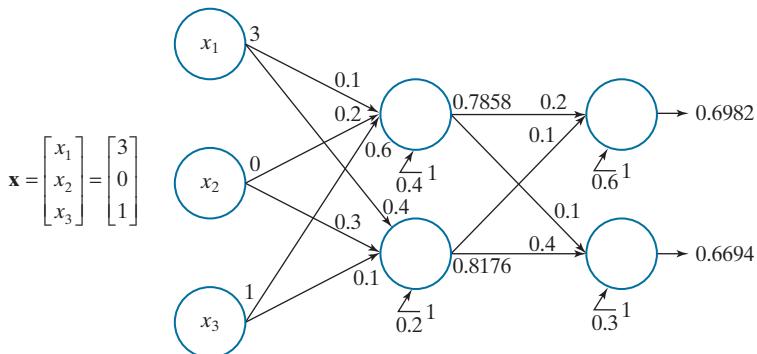
It will be helpful to consider a simple numerical example. Figure 12.32 shows a three-layer neural network consisting of the input layer, one hidden layer, and the output layer. The network accepts three inputs, and has two outputs. Thus, this network is capable of classifying 3-D patterns into one of two classes.

The numbers shown above the arrow heads on each input to a node are the weights of that node associated with the outputs from the nodes in the preceding layer. Similarly, the number shown in the output of each node is the activation value, a , of that node. As noted earlier, there is only one output value for each node, but it is routed to the input of every node in the next layer. The inputs associated with the 1's are bias values.

Let us look at the computations performed at each node, starting with the first (top) node in layer 2. We use Eq. (12-54) to compute the net input, $z_1(2)$, for that node:

FIGURE 12.32

A small, fully connected, feedforward net with labeled weights, biases, and outputs. The activation function is a sigmoid.



$$z_1(2) = \sum_{j=1}^3 w_{1j}(2) a_j(1) + b_1(2) = (0.1)(3) + (0.2)(0) + (0.6)(1) + 0.4 = 1.3$$

We obtain the output of this node using Eqs. (12-51) and (12-55):

$$a_1(2) = h(z_1(2)) = \frac{1}{1 + e^{-1.3}} = 0.7858$$

A similar computation gives the value for the output of the second node in the second layer,

$$z_2(2) = \sum_{j=1}^3 w_{2j}(2) a_j(1) + b_2(2) = (0.4)(3) + (0.3)(0) + (0.1)(1) + 0.2 = 1.5$$

and

$$a_2(2) = h(z_2(2)) = \frac{1}{1 + e^{-1.5}} = 0.8176$$

We use the outputs of the nodes in layer 2 to obtain the net values of the neurons in layer 3:

$$z_1(3) = \sum_{j=1}^2 w_{1j}(3) a_j(2) + b_1(3) = (0.2)(0.7858) + (0.1)(0.8176) + 0.6 = 0.8389$$

The output of this neuron is

$$a_1(3) = h(z_1(3)) = \frac{1}{1 + e^{-0.8389}} = 0.6982$$

Similarly,

$$z_2(3) = \sum_{j=1}^2 w_{2j}(3) a_j(2) + b_2(3) = (0.1)(0.7858) + (0.4)(0.8176) + 0.3 = 0.7056$$

and

$$a_2(3) = h(z_2(2)) = \frac{1}{1 + e^{-0.7056}} = 0.6694$$

If we were using this network to classify the input, we would say that pattern \mathbf{x} belongs to class c_1 because $a_1(L) > a_2(L)$, where $L = 3$ and $n_L = 2$ in this case.

Matrix Formulation

The details of the preceding example reveal that there are numerous individual computations involved in a pass through a neural network. If you wrote a computer program to automate the steps we just discussed, you would find the code to be very inefficient because of all the required loop computations, the numerous node and layer indexing you would need, and so forth. We can develop a more elegant (and computationally faster) implementation by using matrix operations. This means writing Eqs. (12-53) through (12-55) as follows.

First, note that the number of outputs in layer 1 is always of the same dimension as an input pattern, \mathbf{x} , so its matrix (vector) form is simple:

$$\mathbf{a}(1) = \mathbf{x} \quad (12-57)$$

Next, we look at Eq. (12-54). We know that the summation term is just the inner product of two vectors [see Eqs. (12-37) and (12-38)]. However, this equation has to be evaluated for all nodes in every layer past the first. This implies that a loop is required if we do the computations node by node. The solution is to form a matrix, $\mathbf{W}(\ell)$, that contains *all* the weights in layer ℓ . The structure of this matrix is simple—each of its *rows* contains the weights for one of the nodes in layer ℓ :

With reference to our earlier discussion on the order of the subscripts i and j , if we had let i be the sending node and j the receiver, this matrix would have to be transposed.

$$\mathbf{W}(\ell) = \begin{bmatrix} w_{11}(\ell) & w_{12}(\ell) & \dots & w_{1n_{\ell-1}}(\ell) \\ w_{21}(\ell) & w_{22}(\ell) & \dots & w_{2n_{\ell-1}}(\ell) \\ \vdots & \vdots & \ddots & \\ w_{n_\ell 1}(\ell) & w_{n_\ell 2}(\ell) & \dots & w_{n_\ell n_{\ell-1}}(\ell) \end{bmatrix} \quad (12-58)$$

Then, we can obtain all the sum-of-products computations, $z_i(\ell)$, for layer ℓ simultaneously:

$$\mathbf{z}(\ell) = \mathbf{W}(\ell)\mathbf{a}(\ell-1) + \mathbf{b}(\ell) \quad \ell = 2, 3, \dots, L \quad (12-59)$$

where $\mathbf{a}(\ell-1)$ is a column vector of dimension $n_{\ell-1} \times 1$ containing the outputs of layer $\ell-1$, $\mathbf{b}(\ell)$ is a column vector of dimension $n_\ell \times 1$ containing the bias values of all the neurons in layer ℓ , and $\mathbf{z}(\ell)$ is an $n_\ell \times 1$ column vector containing the net input values, $z_i(\ell)$, $i = 1, 2, \dots, n_\ell$, to all the nodes in layer ℓ . You can easily verify that Eq. (12-59) is dimensionally correct.

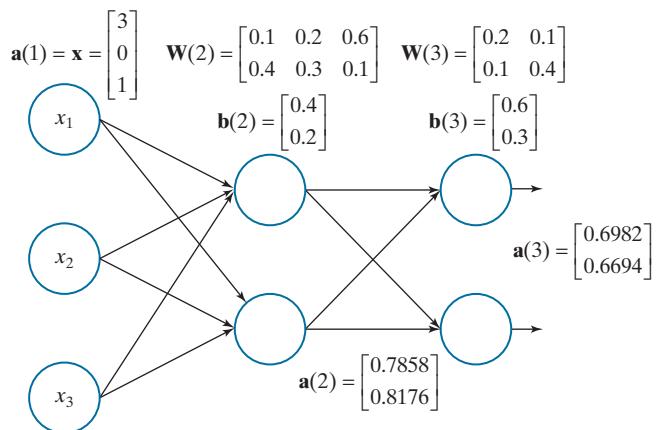
Because the activation function is applied to each net input independently of the others, the outputs of the network at any layer can be expressed in vector form as:

$$\mathbf{a}(\ell) = h[\mathbf{z}(\ell)] = \begin{bmatrix} h(z_1(\ell)) \\ h(z_2(\ell)) \\ \vdots \\ h(z_{n_\ell}(\ell)) \end{bmatrix} \quad (12-60)$$

Implementing Eqs. (12-57) through (12-60) requires just a series of matrix operations, with no loops.

FIGURE 12.33

Same as Fig. 12.32, but using matrix labeling.



EXAMPLE 12.11: Redoing Example 12.10 using matrix operations.

Figure 12.33 shows the same neural network as in Fig. 12.32, but with all its parameters shown in matrix form. As you can see, the representation in Fig. 12.33 is more compact. Starting with

$$\mathbf{a}(1) = \begin{bmatrix} 3 \\ 0 \\ 1 \end{bmatrix}$$

it follows that

$$\mathbf{z}(2) = \mathbf{W}(2)\mathbf{a}(1) + \mathbf{b}(2) = \begin{bmatrix} 0.1 & 0.2 & 0.6 \\ 0.4 & 0.3 & 0.1 \end{bmatrix} \begin{bmatrix} 3 \\ 0 \\ 1 \end{bmatrix} + \begin{bmatrix} 0.4 \\ 0.2 \end{bmatrix} = \begin{bmatrix} 1.3 \\ 1.5 \end{bmatrix}$$

Then,

$$\mathbf{a}(2) = h[\mathbf{z}(2)] = \begin{bmatrix} h(z_1(2)) \\ h(z_2(2)) \end{bmatrix} = \begin{bmatrix} h(1.3) \\ h(1.5) \end{bmatrix} = \begin{bmatrix} 0.7858 \\ 0.8176 \end{bmatrix}$$

With $\mathbf{a}(2)$ as input to the next layer, we obtain

$$\mathbf{z}(3) = \mathbf{W}(3)\mathbf{a}(2) + \mathbf{b}(3) = \begin{bmatrix} 0.2 & 0.1 \\ 0.1 & 0.4 \end{bmatrix} \begin{bmatrix} 0.7858 \\ 0.8176 \end{bmatrix} + \begin{bmatrix} 0.6 \\ 0.3 \end{bmatrix} = \begin{bmatrix} 0.8389 \\ 0.7056 \end{bmatrix}$$

and, as before,

$$\mathbf{a}(3) = h[\mathbf{z}(3)] = \begin{bmatrix} h(z_1(3)) \\ h(z_2(3)) \end{bmatrix} = \begin{bmatrix} h(0.8389) \\ h(0.7056) \end{bmatrix} = \begin{bmatrix} 0.6982 \\ 0.6694 \end{bmatrix}$$

The clarity of the matrix formulation over the indexed notation used in Example 12.10 is evident.

Equations (12-57) through (12-60) are a significant improvement over node-by-node computations, but they apply only to one pattern. To classify multiple pattern vectors, we would have to loop through each pattern using the same set of matrix equations per loop iteration. What we are after is one set of matrix equations

capable of processing *all* patterns in a single forward pass. Extending Eqs. (12-57) through (12-60) to this more general formulation is straightforward. We begin by arranging all our input pattern vectors as *columns* of a single matrix, \mathbf{X} , of dimension $n \times n_p$ where, as before, n is the dimensionality of the vectors and n_p is the number of pattern vectors. It follows from Eq. (12-57) that

$$\mathbf{A}(1) = \mathbf{X} \quad (12-61)$$

where each column of matrix $\mathbf{A}(1)$ contains the initial activation values (i.e., the vector values) for one pattern. This is a straightforward extension of Eq. (12-57), except that we are now dealing with an $n \times n_p$ matrix instead of an $n \times 1$ vector.

The parameters of a network do not change because we are processing more pattern vectors, so the weight matrix is as given in Eq. (12-58). This matrix is of size $n_\ell \times n_{\ell-1}$. When $\ell = 2$, we have that $\mathbf{W}(2)$ is of size $n_2 \times n$, because n_1 is always equal to n . Then, extending the product term of Eq. (12-59) to use $\mathbf{A}(2)$ instead of $\mathbf{a}(2)$, results in the matrix product $\mathbf{W}(2)\mathbf{A}(2)$, which is of size $(n_2 \times n)(n \times n_p) = n_2 \times n_p$. To this, we have to add the bias vector for the second layer, which is of size $n_2 \times 1$. Obviously, we cannot add a matrix of size $n_2 \times n_p$ and a vector of size $n_2 \times 1$. However, as is true of the weight matrices, the bias vectors do not change because we are processing more pattern vectors. We just have to account for one identical bias vector, $\mathbf{b}(2)$, per input vector. We do this by creating a matrix $\mathbf{B}(2)$ of size $n_2 \times n_p$, formed by concatenating column vector $\mathbf{b}(2)$ n_p times, horizontally. Then, Eq. (12-59) written in matrix becomes $\mathbf{Z}(2) = \mathbf{W}(2)\mathbf{A}(1) + \mathbf{B}(2)$. Matrix $\mathbf{Z}(2)$ is of size $n_2 \times n_p$; it contains the computation performed by Eq. (12-59), but for *all* input patterns. That is, each column of $\mathbf{Z}(2)$ is exactly the computation performed by Eq. (12-59) for one input pattern.

The concept just discussed applies to the transition from any layer to the next in the neural network, provided that we use the weights and bias appropriate for a particular location in the network. Therefore, the full matrix version of Eq. (12-59) is

$$\mathbf{Z}(\ell) = \mathbf{W}(\ell)\mathbf{A}(\ell-1) + \mathbf{B}(\ell) \quad (12-62)$$

where $\mathbf{W}(\ell)$ is given by Eq. (12-58) and $\mathbf{B}(\ell)$ is an $n_\ell \times n_p$ matrix whose columns are duplicates of $\mathbf{b}(\ell)$, the bias vector containing the biases of the neurons in layer ℓ .

All that remains is the matrix formulation of the output of layer ℓ . As Eq. (12-60) shows, the activation function is applied independently to each element of the vector $\mathbf{z}(\ell)$. Because each column of $\mathbf{Z}(\ell)$ is simply the application of Eq. (12-60) corresponding to a particular input vector, it follows that

$$\mathbf{A}(\ell) = h[\mathbf{Z}(\ell)] \quad (12-63)$$

where activation function h is applied to each element of matrix $\mathbf{Z}(\ell)$.

Summarizing the dimensions in our matrix formulation, we have: \mathbf{X} and $\mathbf{A}(1)$ are of size $n \times n_p$, $\mathbf{Z}(\ell)$ is of size $n_\ell \times n_p$, $\mathbf{W}(\ell)$ is of size $n_\ell \times n_{\ell-1}$, $\mathbf{A}(\ell-1)$ is of

TABLE 12.2

Steps in the matrix computation of a forward pass through a fully connected, feedforward multilayer neural net.

Step	Description	Equations
Step 1	Input patterns	$\mathbf{A}(1) = \mathbf{X}$
Step 2	Feedforward	For $\ell = 2, \dots, L$, compute $\mathbf{Z}(\ell) = \mathbf{W}(\ell)\mathbf{A}(\ell-1) + \mathbf{B}(\ell)$ and $\mathbf{A}(\ell) = h(\mathbf{Z}(\ell))$
Step 3	Output	$\mathbf{A}(L) = h(\mathbf{Z}(L))$

size $n_{\ell-1} \times n_p$, $\mathbf{B}(\ell)$ is of size $n_\ell \times n_p$, and $\mathbf{A}(\ell)$ is of size $n_\ell \times n_p$. Table 12.2 summarizes the matrix formulation for the forward pass through a fully connected, feed-forward neural network for all pattern vectors. Implementing these operations in a matrix-oriented language like MATLAB is a trivial undertaking. Performance can be improved significantly by using dedicated hardware, such as one or more graphics processing units (GPUs).

The equations in Table 12.2 are used to classify each of a set of patterns into one of n_L pattern classes. Each column of output matrix $\mathbf{A}(L)$ contains the activation values of the n_L output neurons for a specific pattern vector. The class membership of that pattern is given by the location of the output neuron with the highest activation value. Of course, this assumes we know the weights and biases of the network. These are obtained during training using backpropagation, as we explain next.

USING BACKPROPAGATION TO TRAIN DEEP NEURAL NETWORKS

A neural network is defined completely by its weights, biases, and activation function. Training a neural network refers to using one or more sets of training patterns to estimate these parameters. During training, we know the desired response of every output neuron of a multilayer neural net. However, we have no way of knowing what the values of the outputs of hidden neurons should be. In this section, we develop the equations of *backpropagation*, the tool of choice for finding the value of the weights and biases in a multilayer network. This *training by backpropagation* involves four basic steps: (1) inputting the pattern vectors; (2) a forward pass through the network to classify all the patterns of the training set and determine the classification error; (3) a backward (backpropagation) pass that feeds the output error back through the network to compute the changes required to update the parameters; and (4) updating the weights and biases in the network. These steps are repeated until the error reaches an acceptable level. We will provide a summary of all principal results derived in this section at the end of the discussion (see Table 12.3). As you will see shortly, the principal mathematical tool needed to derive the equations of backpropagation is the chain rule from basic calculus.

The Equations of Backpropagation

Given a set of training patterns and a multilayer feedforward neural network architecture, the approach in the following discussion is to find the network parameters

that minimize an *error* (also called *cost* or *objective*) *function*. Our interest is in classification performance, so we define the error function for a neural network as the average of the differences between desired and actual responses. Let \mathbf{r} denote the desired response for a given pattern vector, \mathbf{x} , and let $\mathbf{a}(L)$ denote the actual response of the network to that input. For example, in a ten-class recognition application, \mathbf{r} and $\mathbf{a}(L)$ would be 10-D column vectors. The ten components of $\mathbf{a}(L)$ would be the ten outputs of the neural network, and the components of \mathbf{r} would be zero, except for the element corresponding to the class of \mathbf{x} , which would be 1. For example, if the input training pattern belongs to class 6, the 6th element of \mathbf{r} would be 1 and the rest would be 0's.

The activation values of neuron j in the output layer is $a_j(L)$. We define the error of that neuron as

$$E_j = \frac{1}{2} (r_j - a_j(L))^2 \quad (12-64)$$

for $j = 1, 2, \dots, n_L$, where r_j is the desired response of output neuron $a_j(L)$ for a given pattern \mathbf{x} . The output error with respect to a single \mathbf{x} is the sum of the errors of all output neurons with respect to that vector:

$$\begin{aligned} E &= \sum_{j=1}^{n_L} E_j = \frac{1}{2} \sum_{j=1}^{n_L} (r_j - a_j(L))^2 \\ &= \frac{1}{2} \| \mathbf{r} - \mathbf{a}(L) \|^2 \end{aligned} \quad (12-65)$$

See Eqs. (2-50) and (2-51) regarding the Euclidean vector norm.

When the meaning is clear, we sometimes include the bias term in the word "weights."

where the second line follows from the definition of the Euclidean vector norm. The *total network output error* over all training patterns is defined as the sum of the errors of the individual patterns. We want to find the weights that minimize this total error. As we did for the LMSE perceptron, we find the solution using gradient descent. However, unlike the perceptron, we have no way for computing the gradients of the weights in the hidden nodes. The beauty of backpropagation is that we can achieve an equivalent result by propagating the output error back into the network.

The key objective is to find a scheme to adjust all weights in a network using training patterns. In order to do this, we need to know how E changes with respect to the weights in the network. The weights are contained in the expression for the net input to each node [see Eq. (12-54)], so the quantity we are after is $\partial E / \partial z_j(\ell)$ where, as defined in Eq. (12-54), $z_j(\ell)$ is the net input to node j in layer ℓ . In order to simplify the notation later, we use the symbol $\delta_j(\ell)$ to denote $\partial E / \partial z_j(\ell)$. Because backpropagation starts with the output and works backward from there, we look first at

We use " j " generically to mean any node in the network. We are not concerned at the moment with inputs to, or outputs from, a node.

$$\delta_j(L) = \frac{\partial E}{\partial z_j(L)} \quad (12-66)$$

We can express this equation in terms of the output $a_j(L)$ using the chain rule:

$$\begin{aligned}\delta_j(L) &= \frac{\partial E}{\partial z_j(L)} = \frac{\partial E}{\partial a_j(L)} \frac{\partial a_j(L)}{\partial z_j(L)} = \frac{\partial E}{\partial a_j(L)} \frac{\partial h(z_j(L))}{\partial z_j(L)} \\ &= \frac{\partial E}{\partial a_j(L)} h'(z_j(L))\end{aligned}\quad (12-67)$$

where we used Eq. (12-56) to obtain the last expression in the first line. This equation gives us the value of $\delta_j(L)$ in terms of quantities that can be observed or computed. For example, if we use Eq. (12-64) as our error measure, and Eq. (12-52) for $h'(z_j(x))$, then

$$\delta_j(L) = h(z_j(L)) [1 - h(z_j(L))] [a_j(L) - r_j] \quad (12-68)$$

where we interchanged the order of the terms. The $h(z_j(L))$ are computed in the forward pass, $a_j(L)$ can be observed in the output of the network, and r_j is given along with \mathbf{x} during training. Therefore, we can compute $\delta_j(L)$.

Because the relationship between the net input and the output of any neuron in any layer (except the first) is the same, the form of Eq. (12-66) is valid for any node j in any hidden layer:

$$\delta_j(\ell) = \frac{\partial E}{\partial z_j(\ell)} \quad (12-69)$$

This equation tells us how E changes with respect to a change in the net input to any neuron in the network. What we want to do next is express $\delta_j(\ell)$ in terms of $\delta_j(\ell+1)$. Because we will be proceeding backward in the network, this means that if we have this relationship, then we can start with $\delta_j(L)$ and find $\delta_j(L-1)$. We then use this result to find $\delta_j(L-2)$, and so on until we arrive at layer 2. We obtain the desired expression using the chain rule (see Problem 12.25):

$$\begin{aligned}\delta_j(\ell) &= \frac{\partial E}{\partial z_j(\ell)} = \sum_i \frac{\partial E}{\partial z_i(\ell+1)} \frac{\partial z_i(\ell+1)}{\partial a_j(\ell)} \frac{\partial a_j(\ell)}{\partial z_j(\ell)} \\ &= \sum_i \delta_i(\ell+1) \frac{\partial z_i(\ell+1)}{\partial a_j(\ell)} h'(z_j(\ell)) \\ &= h'(z_j(\ell)) \sum_i w_{ij}(\ell+1) \delta_i(\ell+1)\end{aligned}\quad (12-70)$$

for $\ell = L-1, L-2, \dots, 2$, where we used Eqs. (12-55) and (12-69) to obtain the middle line, and Eq. (12-54), plus some rearranging to obtain the last line.

The preceding development tells us how we can start with the error in the output (which we can compute) and obtain how that error changes as function of the net inputs to every node in the network. This is an intermediate step toward our final objective, which is to obtain expressions for $\partial E / \partial w_{ij}(\ell)$ and $\partial E / \partial b_i(\ell)$ in terms of $\delta_j(\ell) = \partial E / \partial z_j(\ell)$. For this, we use the chain rule again:

$$\begin{aligned}
 \frac{\partial E}{\partial w_{ij}(\ell)} &= \frac{\partial E}{\partial z_i(\ell)} \frac{\partial z_i(\ell)}{\partial w_{ij}(\ell)} \\
 &= \delta_i(\ell) \frac{\partial z_i(\ell)}{\partial w_{ij}(\ell)} \\
 &= a_j(\ell - 1) \delta_i(\ell)
 \end{aligned} \tag{12-71}$$

where we used Eq. (12-54), Eq. (12-69), and interchanged the order of the results to clarify matrix formulations later in our discussion. Similarly (see Problem 12.26),

$$\frac{\partial E}{\partial b_i(\ell)} = \delta_i(\ell) \tag{12-72}$$

Now we have the rate of change of E with respect to the network weights and biases in terms of quantities we can compute. The last step is to use these results to update the network parameters using gradient descent:

$$\begin{aligned}
 w_{ij}(\ell) &= w_{ij}(\ell) - \alpha \frac{\partial E(\ell)}{\partial w_{ij}(\ell)} \\
 &= w_{ij}(\ell) - \alpha \delta_i(\ell) a_j(\ell - 1)
 \end{aligned} \tag{12-73}$$

and

$$\begin{aligned}
 b_i(\ell) &= b_i(\ell) - \alpha \frac{\partial E}{\partial b_i(\ell)} \\
 &= b_i(\ell) - \alpha \delta_i(\ell)
 \end{aligned} \tag{12-74}$$

for $\ell = L - 1, L - 2, \dots, 2$, where the a 's are computed in the forward pass, and the δ 's are computed during backpropagation. As with the perceptron, α is the learning rate constant used in gradient descent. There are numerous approaches that attempt to find optimal learning rates, but ultimately this is a problem-dependent parameter that involves experimenting. A reasonable approach is to start with a small value of α (e.g., 0.01), then experiment with vectors from the training set to determine a suitable value in a given application. Remember, α is used only during training, so it has no effect on post-training operating performance.

Matrix Formulation

As with the equations that describe the forward pass through a neural network, the equations of backpropagation developed in the previous discussion are excellent for describing how the method works at a fundamental level, but they are clumsy when it comes to implementation. In this section, we follow a procedure similar to the one we used for the forward pass to develop the matrix equations for backpropagation.

As before, we arrange all the pattern vectors as columns of matrix \mathbf{X} , and package the weights of layer ℓ as matrix $\mathbf{W}(\ell)$. We use $\mathbf{D}(\ell)$ to denote the matrix equivalent of $\delta(\ell)$, the vector containing the errors in layer ℓ . Our first step is to find an expression for $\mathbf{D}(L)$. We begin at the output and proceed backward, as before. From Eq. (12-67),

$$\delta(L) = \begin{bmatrix} \delta_1(L) \\ \delta_2(L) \\ \vdots \\ \delta_{n_L}(L) \end{bmatrix} = \begin{bmatrix} \frac{\partial E}{\partial a_1(L)} h'(z_1(L)) \\ \frac{\partial E}{\partial a_2(L)} h'(z_2(L)) \\ \vdots \\ \frac{\partial E}{\partial a_{n_L}(L)} h'(z_{n_L}(L)) \end{bmatrix} = \begin{bmatrix} \frac{\partial E}{\partial a_1(L)} \\ \frac{\partial E}{\partial a_2(L)} \\ \vdots \\ \frac{\partial E}{\partial a_{n_L}(L)} \end{bmatrix} \odot \begin{bmatrix} h'(z_1(L)) \\ h'(z_2(L)) \\ \vdots \\ h'(z_{n_L}(L)) \end{bmatrix} \quad (12-75)$$

where, as defined in Section 2.6, “ \odot ” denotes elementwise multiplication (of two vectors in this case). We can write the vector on the left of this symbol as $\partial E / \partial \mathbf{a}(L)$, and the vector on the right as $h'(\mathbf{z}(L))$. Then, we can write Eq. (12-75) as

$$\delta(L) = \frac{\partial E}{\partial \mathbf{a}(L)} \odot h'(\mathbf{z}(L)) \quad (12-76)$$

This $n_L \times 1$ column vector contains the activation values of all the output neurons for *one* pattern vector. The only error function we use in this chapter is a quadratic function, which is given in vector form in Eq. (12-65). The partial of that quadratic function with respect to $\mathbf{a}(L)$ is $(\mathbf{a}(L) - \mathbf{r})$ which, when substituted into Eq. (12-76), gives us

$$\delta(L) = (\mathbf{a}(L) - \mathbf{r}) \odot h'(\mathbf{z}(L)) \quad (12-77)$$

Column vector $\delta(L)$ accounts for one pattern vector. To account for *all* n_p patterns simultaneously we form a matrix $\mathbf{D}(\ell)$, whose columns are the $\delta(L)$ from Eq. (12-77), evaluated for a specific pattern vector. This is equivalent to writing Eq. (12-77) directly in matrix form as

$$\mathbf{D}(L) = (\mathbf{A}(L) - \mathbf{R}) \odot h'(\mathbf{Z}(L)) \quad (12-78)$$

Each column of $\mathbf{A}(L)$ is the network output for one pattern. Similarly, each column of \mathbf{R} is a binary vector with a 1 in the location corresponding to the class of a particular pattern vector, and 0's elsewhere, as explained earlier. Each column of the difference $(\mathbf{A}(L) - \mathbf{R})$ contains the components of $\|\mathbf{a} - \mathbf{r}\|$. Therefore, squaring the elements of a column, adding them, and dividing by 2 is the same as computing the error measure defined in Eq. (12-65), for one pattern. Adding all the column computations gives an average measure of error for all the patterns. Similarly, the columns of matrix $h'(\mathbf{Z}(L))$ are values of the net inputs to all output neurons, with

each column corresponding to one pattern vector. All matrices in Eq. (12-78) are of size $n_L \times n_p$.

Following a similar line of reasoning, we can express Eq. (12-70) in matrix form as

$$\mathbf{D}(\ell) = (\mathbf{W}^T(\ell+1)\mathbf{D}(\ell+1)) \odot h'(\mathbf{Z}(\ell)) \quad (12-79)$$

It is easily confirmed by dimensional analysis that the matrix $\mathbf{D}(\ell)$ is of size $n_\ell \times n_p$ (see Problem 12.27). Note that Eq. (12-79) uses the weight matrix transposed. This reflects the fact that the inputs to layer ℓ are coming from layer $\ell+1$, because in backpropagation we move in the direction opposite of a forward pass.

We complete the matrix formulation by expressing the weight and bias update equations in matrix form. Considering the weight matrix first, we can tell from Eqs. (12-70) and (12-73) that we are going to need matrices $\mathbf{W}(\ell)$, $\mathbf{D}(\ell)$, and $\mathbf{A}(\ell-1)$. We already know that $\mathbf{W}(\ell)$ is of size $n_\ell \times n_{\ell-1}$ and that $\mathbf{D}(\ell)$ is of size $n_\ell \times n_p$. Each column of matrix $\mathbf{A}(\ell-1)$ is the set of outputs of the neurons in layer $\ell-1$ for one pattern vector. There are n_p patterns, so $\mathbf{A}(\ell-1)$ is of size $n_{\ell-1} \times n_p$. From Eq. (12-73) we infer that \mathbf{A} post-multiplies \mathbf{D} , so we are also going to need $\mathbf{A}^T(\ell-1)$, which is of size $n_p \times n_{\ell-1}$. Finally, recall that in a matrix formulation, we construct a matrix $\mathbf{B}(\ell)$ of size $n_\ell \times n_p$ whose columns are copies of vector $\mathbf{b}(\ell)$, which contains all the biases in layer ℓ .

Next, we look at updating the biases. We know from Eq. (12-74) that each element $b_i(\ell)$ of $\mathbf{b}(\ell)$ is updated as $b_i(\ell) = b_i(\ell) - \alpha \delta_i(\ell)$, for $i = 1, 2, \dots, n_\ell$. Therefore, $\mathbf{b}(\ell) = \mathbf{b}(\ell) - \alpha \boldsymbol{\delta}(\ell)$. But this is for *one* pattern, and the columns of $\mathbf{D}(\ell)$ are the $\boldsymbol{\delta}(\ell)$'s for *all* patterns in the training set. This is handled in a matrix formulation by using the *average* of the columns of $\mathbf{D}(\ell)$ (this is the average error over all patterns) to update $\mathbf{b}(\ell)$.

Putting it all together results in the following two equations for updating the network parameters:

$$\mathbf{W}(\ell) = \mathbf{W}(\ell) - \alpha \mathbf{D}(\ell) \mathbf{A}^T(\ell-1) \quad (12-80)$$

and

$$\mathbf{b}(\ell) = \mathbf{b}(\ell) - \alpha \sum_{k=1}^{n_p} \boldsymbol{\delta}_k(\ell) \quad (12-81)$$

where $\boldsymbol{\delta}_k(\ell)$ is the k th column of matrix $\mathbf{D}(\ell)$. As before, we form matrix $\mathbf{B}(\ell)$ of size $n_\ell \times n_p$ by concatenating $\mathbf{b}(\ell)$ n_p times in the horizontal direction:

$$\mathbf{B}(\ell) = \text{concatenate}_{n_p \text{ times}} \{ \mathbf{b}(\ell) \} \quad (12-82)$$

As we mentioned earlier, backpropagation consists of four principal steps: (1) inputting the patterns, (2) a forward pass, (3) a backpropagation pass, and (4) a parameter update step. The process begins by specifying the initial weights and biases as (small) random numbers. Table 12.3 summarizes the matrix formulations of these four steps. During training, these steps are repeated for a number of specified epochs, or until a predefined measure of error is deemed to be small enough.

TABLE 12.3

Matrix formulation for training a feedforward, fully connected multilayer neural network using backpropagation. Steps 1–4 are for one epoch of training. \mathbf{X} , \mathbf{R} , and the learning rate parameter α , are provided to the network for training. The network is initialized by specifying weights, $\mathbf{W}(1)$, and biases, $\mathbf{B}(1)$, as small random numbers.

Step	Description	Equations
Step 1	Input patterns	$\mathbf{A}(1) = \mathbf{X}$
Step 2	Forward pass	For $\ell = 2, \dots, L$, compute: $\mathbf{Z}(\ell) = \mathbf{W}(\ell)\mathbf{A}(\ell-1) + \mathbf{B}(\ell)$; $\mathbf{A}(\ell) = h(\mathbf{Z}(\ell))$; $h'(\mathbf{Z}(\ell))$; and $\mathbf{D}(L) = (\mathbf{A}(L) - \mathbf{R}) \odot h'(\mathbf{Z}(L))$
Step 3	Backpropagation	For $\ell = L-1, L-2, \dots, 2$, compute $\mathbf{D}(\ell) = (\mathbf{W}^T(\ell+1)\mathbf{D}(\ell+1)) \odot h'(\mathbf{Z}(\ell))$
Step 4	Update weights and biases	For $\ell = 2, \dots, L$, let $\mathbf{W}(\ell) = \mathbf{W}(\ell) - \alpha \mathbf{D}(\ell) \mathbf{A}^T(\ell-1)$, $\mathbf{b}(\ell) = \mathbf{b}(\ell) - \alpha \sum_{k=1}^{n_p} \delta_k(\ell)$, and $\mathbf{B}(\ell) = \text{concatenate}_{n_p \text{ times}} \{\mathbf{b}(\ell)\}$, where the $\delta_k(\ell)$ are the columns of $\mathbf{D}(\ell)$

There are two major types of errors in which we are interested. One is the *classification error*, which we compute by counting the number of patterns that were misclassified and dividing by the total number of patterns in the training set. Multiplying the result by 100 gives the percentage of patterns misclassified. Subtracting the result from 1 and multiplying by 100 gives the percent correct recognition. The other is the *mean squared error* (MSE), which is based on actual values of E . For the error defined in Eq. (12-65), this value is obtained (for one pattern) by squaring the elements of a column of the matrix $(\mathbf{A}(L) - \mathbf{R})$, adding them, and dividing by the result by 2 (see Problem 12.28). Repeating this operation for all columns and dividing the result by the number of patterns in \mathbf{X} gives the MSE over the entire training set.

EXAMPLE 12.12: Using a fully connected neural net to solve the XOR problem.

Figure 12.34(a) shows the XOR classification problem discussed previously (the coordinates were chosen to center the patterns for convenience in indexing, but the spatial relationships are as before). Pattern matrix \mathbf{X} and class membership matrix \mathbf{R} are:

$$\mathbf{X} = \begin{bmatrix} 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix}; \quad \mathbf{R} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

We specified a neural network having three layers, with two nodes each (see Fig. 12.35). This is the smallest network consistent with our architecture in Fig. 12.31. Comparing it to the minimum perceptron arrangements in Fig. 12.28(a), we see that our neural network performs the same basic function, in the sense that it has two inputs and two outputs.

We used $\alpha = 1.0$, an initial set of Gaussian random weights of zero mean and standard deviation of 0.02, and the activation function in Eq. (12-51). We then trained the network for 10,000 epochs (we used a large number of epochs to get close to the values in the \mathbf{R} ; we discuss below solutions with fewer epochs). The resulting weights and biases were:

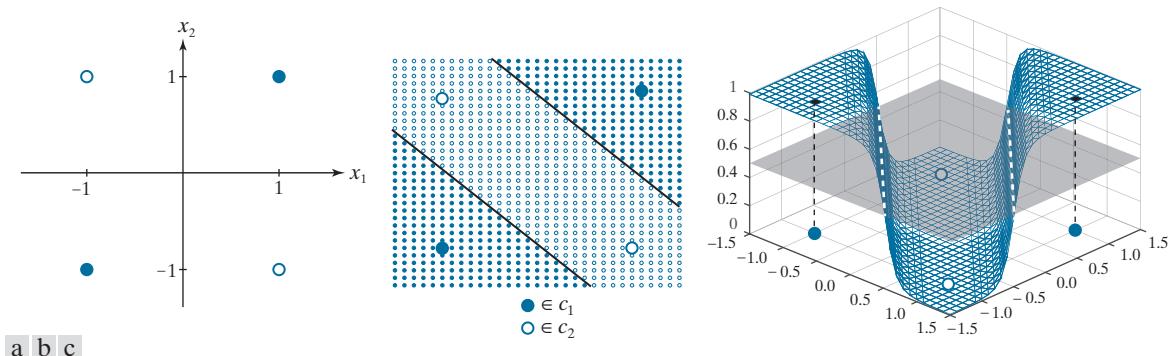


FIGURE 12.34 Neural net solution to the XOR problem. (a) Four patterns in an XOR arrangement. (b) Results of classifying additional points in the range -1.5 to 1.5 in increments of 0.1 . All solid points were classified as belonging to class c_1 and all open circles were classified as belonging to class c_2 . Together, the two lines separating the regions constitute the decision boundary [compare with Fig. 12.27(b)]. (c) Decision surface, shown as a mesh. The decision boundary is the pair of dashed, white lines in the intersection of the surface and a plane perpendicular to the vertical axis, intersecting that axis at 0.5 . (Figure (c) is shown in a different perspective than (b) in order to make all four patterns visible.)

$$\mathbf{W}(2) = \begin{bmatrix} 4.792 & 4.792 \\ 4.486 & 4.486 \end{bmatrix}; \quad \mathbf{b}(2) = \begin{bmatrix} 4.590 \\ -4.486 \end{bmatrix}; \quad \mathbf{W}(3) = \begin{bmatrix} -9.180 & 9.429 \\ 9.178 & -9.427 \end{bmatrix}; \quad \mathbf{b}(3) = \begin{bmatrix} 4.420 \\ -4.419 \end{bmatrix}$$

Figure 12.35 shows the neural net based on these values.

When presented with the four training patterns after training was completed, the results at the two outputs should have been equal to the values in \mathbf{R} . Instead, the values were close:

$$\mathbf{A}(3) = \begin{bmatrix} 0.987 & 0.990 & 0.010 & 0.010 \\ 0.013 & 0.010 & 0.990 & 0.990 \end{bmatrix}$$

These weights and biases, along with the sigmoid activation function, completely specify our trained neural network. To test its performance with values other than the training patterns, which we know it classifies correctly, we created a set of 2-D test patterns by subdividing the pattern space into increments of 0.1 , from -1.5 to 1.5 in both directions, and classified the resulting points using a forward pass through

FIGURE 12.35
Neural net used to solve the XOR problem, showing the weights and biases learned via training using the equations in Table 12.3.

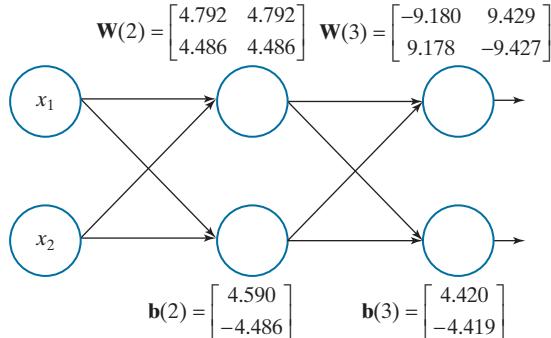
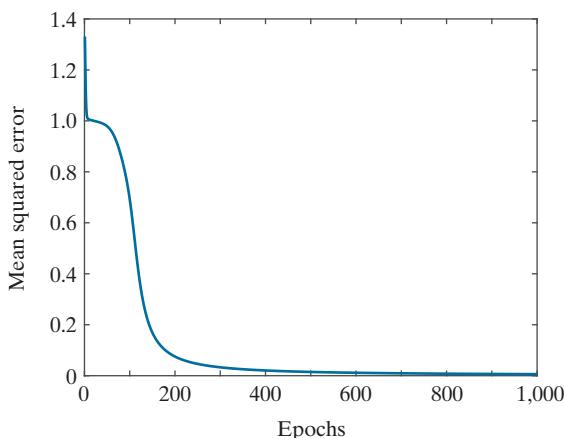


FIGURE 12.36

MSE as a function of training epochs for the XOR pattern arrangement.



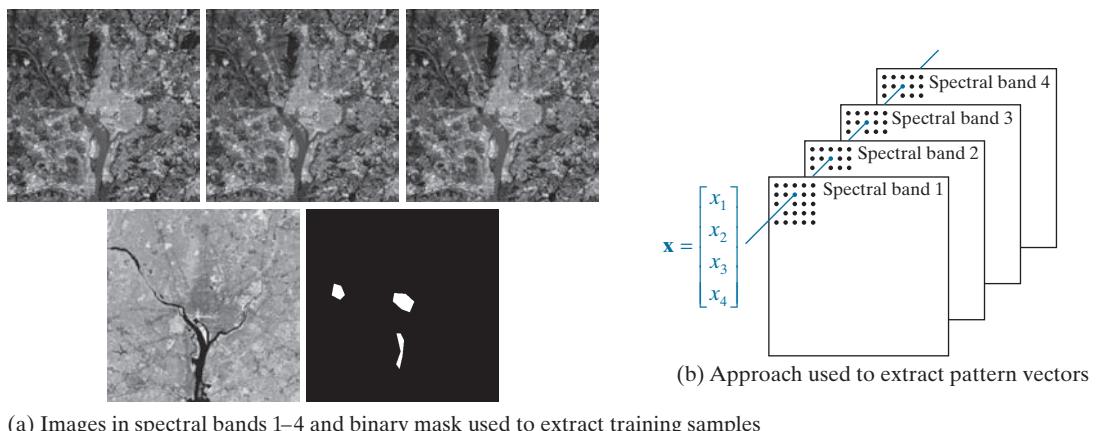
the network. If the activation value of output node 1 was greater than the activation value of output node 2, the pattern was assigned to class c_1 ; otherwise, it was assigned to class c_2 . Fig. 12.34(b) is a plot of the results. Solid dots are points classified into to class c_1 , and white dots were classified as belonging to class c_2 . The boundaries between these two regions (shown as solid black lines) are precisely the boundaries in Fig. 12.27(b). Thus, our small neural network found the simplest boundary between the two classes, and thus performed the same function as the perceptron arrangement in Fig. 12.28(a).

Figure 12.34(c) shows the decision surface. This figure is analogous to Fig. 12.24(b), but it intersects the plane twice because the patterns are not linearly separable. Our decision boundary is the intersection of the decision surface with a plane perpendicular to the vertical axis, and intersecting that axis at 0.5. This is because the range of values in the output nodes is in the [0, 1] range, and we assign a pattern to the class for which one the two outputs had the largest value. The plane is shown shaded in the figure, and the decision boundary is shown as dashed white lines. We adjusted the viewing perspective of Fig. 12.34(c) so you can see all the XOR points.

Because classification in this case is based on selecting the largest output, we do not need the outputs to be so close to 1 and 0 as we showed above, provided they are greater for the patterns of class c_1 and conversely for the patterns of class c_2 . This means that we can train the network using fewer epochs and still achieve correct recognition. For example, correct classification of the XOR patterns can be achieved using the parameters learned with as few as 150 epochs. Figure 12.36 shows the reason why this is possible. By the end of the 1000th epoch, the mean squared error has decreased almost to zero, so we would expect it to decrease very little from there for 10,000 epochs. We know from the preceding results that the neural net performed flawlessly using the weights learned with 10,000 epochs. Because the error for 1,000 and 10,000 epochs is close, we can expect the weights to be close as well. At 150 epochs, the error has decreased by close to 90% from its maximum, so the probability that the weights would perform well should be reasonably high, which was true in this case.

EXAMPLE 12.13: Using neural nets to classify multispectral image data.

In this example, we compare the recognition performance of the Bayes classifier we discussed in Section 12.4 and the multilayer neural nets discussed in this section. The objective here is the same as in Example 12.6: to classify the pixels of multispectral image data into three pattern classes: *water*, *urban*,



(a) Images in spectral bands 1–4 and binary mask used to extract training samples

FIGURE 12.37 (a) Starting with the leftmost image: blue, green, red, near infrared, and binary mask images. In the mask, the lower region is for water, the center region is for the urban area, and the left mask corresponds to vegetation. All images are of size 512×512 pixels. (b) Approach used for generating 4-D pattern vectors from a stack of the four multispectral images. (Multispectral images courtesy of NASA.)

and *vegetation*. Figure 12.37 shows the four multispectral images used in the experiment, the masks used to extract the training and test samples, and the approach used to generate the 4-D pattern vectors.

As in Example 12.6, we extracted a total of 1900 training pattern vectors and 1887 test pattern vectors (see Table 12.1 for a listing of vectors by class). After preliminary runs with the training data to establish that the mean squared error was decreasing as a function of epoch, we determined that a neural net with one hidden layer of two nodes achieved stable learning with $\alpha = 0.001$ and 1,000 training epochs. Keeping those two parameters fixed, we varied the number of nodes in the internal layer, as listed in Table 12.4. The objective of these preliminary runs was to determine the smallest neural net that would give the best recognition rate. As you can see from the results in the table, [4 3 3] is clearly the architecture of choice in this case. Figure 12.38 shows this neural net, along with the parameters learned during training.

After the basic architecture was defined, we kept the learning rate constant at $\alpha = 0.001$ and varied the number of epochs to determine the best recognition rate with the architecture in Fig. 12.38. Table 12.5 shows the results. As you can see, the recognition rate improved slowly as a function of epoch, reaching a plateau at around 50,000 epochs. In fact, as Fig. 12.39 shows, the MSE decreased quickly up to about 800 training epochs and decreased slowly after that, explaining why the correct recognition rate changed so little after about 2,000 epochs. Similar results were obtained with $\alpha = 0.01$, but decreasing

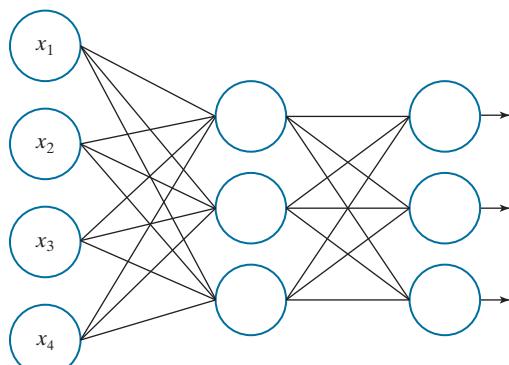
TABLE 12.4

Recognition rate as a function of neural net architecture for $\alpha = 0.001$ and 1,000 training epochs. The network architecture is defined by the numbers in brackets. The first and last number inside each bracket refer to the number of input and output nodes, respectively. The inner entries give the number of nodes in each hidden layer.

Network Architecture	[4 2 3]	[4 3 3]	[4 4 3]	[4 5 3]	[4 2 2 3]	[4 4 3 3]	[4 4 4 3]	[4 10 3 3]	[4 10 10 3]
Recognition Rate	95.8%	96.2%	95.9%	96.1%	74.6%	90.8%	87.1%	84.9%	89.7%

FIGURE 12.38

Neural net architecture used to classify the multispectral image data in Fig. 12.37 into three classes: water, urban, and vegetation. The parameters shown were obtained in 50,000 epochs of training using $\alpha = 0.001$.



$$\mathbf{W}(2) = \begin{bmatrix} 2.393 & 1.020 & 1.249 & -15.965 \\ 6.599 & -2.705 & -0.912 & 14.928 \\ 8.745 & 0.270 & 3.358 & 1.249 \end{bmatrix} \quad \mathbf{W}(3) = \begin{bmatrix} 4.093 & -10.563 & -3.245 \\ 7.045 & 9.662 & 6.436 \\ -7.447 & 3.931 & -6.619 \end{bmatrix}$$

$$\mathbf{b}(2) = [4.920 \quad -2.002 \quad -3.485]^T \quad \mathbf{b}(3) = [3.277 \quad -14.982 \quad 1.582]^T$$

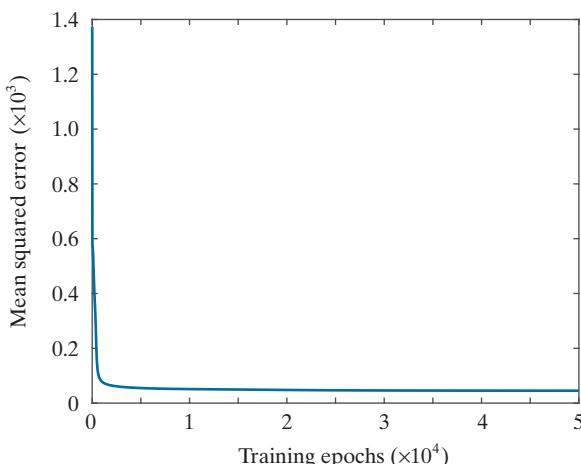
TABLE 12.5

Recognition performance on the training set as a function of training epochs. The learning rate constant was $\alpha = 0.001$ in all cases.

Training Epochs	1,000	10,000	20,000	30,000	40,000	50,000	60,000	70,000	80,000
Recognition Rate	95.3%	96.6%	96.7%	96.8%	96.9%	97.0%	97.0%	97.0%	97.0%

FIGURE 12.39

MSE for the network architecture in Fig. 12.38 as a function of the number of training epochs. The learning rate parameter was $\alpha = 0.001$ in all cases.



this parameter to $\alpha = 0.1$ resulted in a drop of the best correct recognition rate to 49.1%. Based on the preceding results, we used $\alpha = 0.001$ and 50,000 epochs to train the network.

The parameters in Fig. 12.38 were the result of training. The recognition rate for the training data using these parameters was 97%. We achieved a recognition rate of 95.6% on the test set using the same parameters. The difference between these two figures, and the 96.4% and 96.2%, respectively, obtained for the same data with the Bayes classifier (see Example 12.6), are statistically insignificant.

The fact that our neural networks achieved results comparable to those obtained with the Bayes classifier is not surprising. It can be shown (Duda, Hart, and Stork [2001]) that a three-layer neural net, trained by backpropagation using a sum of errors squared criterion, approximates the Bayes decision functions in the limit, as the number of training samples approaches infinity. Although our training sets were small, the data were well behaved enough to yield results that are close to what theory predicts.

12.6 DEEP CONVOLUTIONAL NEURAL NETWORKS

Up to this point, we have organized pattern features as vectors. Generally, this assumes that the form of those features has been specified (i.e., “engineered” by a human designer) and extracted from images prior to being input to a neural network (Example 12.13 is an illustration of this approach). But one of the strengths of neural networks is that they are capable of learning pattern features directly from training data. What we would like to do is input a set of training images directly into a neural network, and have the network learn the necessary features *on its own*. One way to do this would be to convert images to vectors directly by organizing the pixels based on a linear index (see Fig. 12.1), and then letting each element (pixel) of the linear index be an element of the vector. However, this approach does not utilize any spatial relationships that may exist between pixels in an image, such as pixel arrangements into corners, the presence of edge segments, and other features that may help to differentiate one image from another. In this section, we present a class of neural networks called *deep convolutional neural networks* (*CNNs* or *ConvNets* for short) that accept images as inputs and are ideally suited for automatic learning and image classification. In order to differentiate between CNNs and the neural nets we studied in Section 12.5, we will refer to the latter as “fully connected” neural networks.

A BASIC CNN ARCHITECTURE

In the following discussion, we use a *LeNet* architecture (see references at the end of this chapter) to introduce convolutional nets. We do this for two main reasons: First, the LeNet architecture is reasonably simple to understand. This makes it ideal for introducing basic CNN concepts. Second, our real interest is in deriving the equations of backpropagation for convolutional networks, a task that is simplified by the intuitiveness of LeNets.

The CNN in Fig. 12.40 contains all the basic elements of a LeNet architecture, and we use it without loss of generality. A key difference between this architecture and the neural net architectures we studied in the previous section is that inputs to CNNs are 2-D arrays (images), while inputs to our fully connected neural networks are vectors. However, as you will see shortly, the computations performed by both networks are very similar: (1) a sum of products is formed, (2) a bias value is added,

To simplify the explanation of the CNN in Fig. 12.40, we focus attention initially on a single image input. Multiple input images are a trivial extension we will consider later in our discussion.

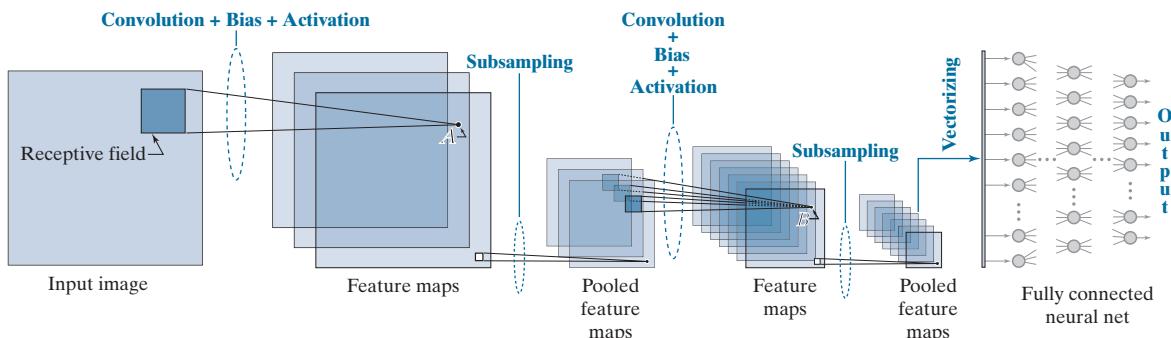


FIGURE 12.40 A CNN containing all the basic elements of a LeNet architecture. Points A and B are specific values to be addressed later in this section. The last pooled feature maps are vectorized and serve as the input to a fully connected neural network. The class to which the input image belongs is determined by the output neuron with the highest value.

(3) the result is passed through an activation function, and (4) the activation value becomes a single input to a following layer.

Despite the fact that the computations performed by CNNs and fully connected neural nets are similar, there are some basic differences between the two, beyond their input formats being 2-D versus vectors. An important difference is that CNNs are capable of learning 2-D features directly from raw image data, as mentioned earlier. Because the tools for systematically engineering comprehensive feature sets for complex image recognition tasks do not exist, having a system that can learn its own image features from raw image data is a crucial advantage of CNNs. Another major difference is in the way in which layers are connected. In a fully connected neural net, we feed the output of *every* neuron in a layer directly into the input of *every* neuron in the next layer. By contrast, in a CNN we feed into every input of a layer, a *single* value, determined by the convolution (hence the name *convolutional* neural net) over a spatial neighborhood in the output of the previous layer. Therefore, CNNs are not fully connected in the sense defined in the last section. Another difference is that the 2-D arrays from one layer to the next are subsampled to reduce sensitivity to translational variations in the input. These differences and their meaning will become clear as we look at various CNN configurations in the following discussion.

Basics of How a CNN Operates

As noted above, the type of neighborhood processing in CNNs is spatial convolution. We explained the mechanics of spatial convolution in Fig. 3.29, and expressed it mathematically in Eq. (3-35). As that equation shows, convolution computes a sum of products between pixels and a set of kernel weights. This operation is carried out at every spatial location in the input image. The result at each location (x, y) in the input is a scalar value. Think of this value as the output of a neuron in a layer of a fully connected neural net. If we add a bias and pass the result through an activation function (see Fig. 12.29), we have a complete analogy between the

We will discuss in the next subsection the exact form of neural computations in a CNN, and show they are equivalent in form to the computations performed by neurons in a fully connected neural net.

basic computations performed by a CNN and those performed by the neural nets discussed in the previous section.

These remarks are summarized in Fig. 12.40, the leftmost part of which shows a neighborhood at one location in the input image. In CNN terminology, these neighborhoods are called *receptive fields*. All a receptive field does is select a region of pixels in the input image. As the figure shows, the first operation performed by a CNN is convolution, whose values are generated by moving the receptive field over the image and, at each location, forming a sum of products of a set of weights and the pixels contained in the receptive field. The set of weights, arranged in the shape of the receptive field, is a *kernel*, as in Chapter 3. The number of spatial increments by which a receptive field is moved is called the *stride*. Our spatial convolutions in previous chapters had a stride of one, but that is not a requirement of the equations themselves. In CNNs, an important motivation for using strides greater than one is data reduction. For example, changing the stride from one to two reduces the image resolution by one-half in each spatial dimension, resulting in a three-fourths reduction in the amount of data per image. Another important motivation is as a substitute for subsampling which, as we discuss below, is used to reduce system sensitivity to spatial translation.

To each convolution value (sum of products) we add a bias, then pass the result through an activation function to generate a single value. Then, this value is fed to the corresponding (x, y) location in the input of the next layer. When repeated for all locations in the input image, the process just explained results in a 2-D set of values that we store in next layer as a 2-D array, called a *feature map*. This terminology is motivated by the fact that the role performed by convolution is to extract features such as edges, points, and blobs from the input (remember, convolution is the basis of spatial filtering, which we used in Chapter 3 for tasks such as smoothing, sharpening, and computing edges in an image). The same weights and a single bias are used to generate the convolution (feature map) values corresponding to all locations of the receptive field in the input image. This is done to cause the same feature to be detected at all points in the image. Using the same weights and bias for this purpose is called *weight* (or *parameter*) *sharing*.

Figure 12.40 shows three feature maps in the first layer of the network. The other two feature maps are generated in the manner just explained, but using a *different* set of weights and bias for each feature map. Because each set of weights and bias is different, each feature map generally will contain a different set of features, all extracted from the same input image. The feature maps are referred to collectively as a *convolutional layer*. Thus, the CNN in Fig. 12.40 has two convolutional layers.

The process after convolution and activation is *subsampling* (also called *pooling*), which is motivated by a model of the mammal visual cortex proposed by Hubel and Wiesel [1959]. Their findings suggest that parts of the visual cortex consist of *simple* and *complex* cells. The simple cells perform feature extraction, while the complex cells combine (aggregate) those features into a more meaningful whole. In this model, a reduction in spatial resolution appears to be responsible for achieving translational invariance. Pooling is a way of modeling this reduction in dimensionality. When training a CNN with large image databases, pooling has the additional

In the terminology of Chapter 3, a feature map is a spatially filtered image.

adjacency is not a requirement of pooling per se. We assume it here for simplicity and because this is an approach that is used frequently.

advantage of reducing the volume of data being processed. You can think of the results of subsampling as producing *pooled feature maps*. In other words, a pooled feature map is a feature map of reduced spatial resolution. Pooling is done by subdividing a feature map into a set of small (typically 2×2) regions, called *pooling neighborhoods*, and replacing *all* elements in such a neighborhood by a *single* value. We assume that pooling neighborhoods are *adjacent* (i.e., they do not overlap). There are several ways to compute the pooled values; collectively, the different approaches are called *pooling methods*. Three common pooling methods are: (1) *average pooling*, in which the values in each neighborhood are replaced by the average of the values in the neighborhood; (2) *max-pooling*, which replaces the values in a neighborhood by the maximum value of its elements; and (3) L_2 pooling, in which the resulting pooled value is the square root of the sum of the neighborhood values squared. There is one pooled feature map for each feature map. The pooled feature maps are referred to collectively as a *pooling layer*. In Fig. 12.40 we used 2×2 pooling so each resulting pooled map is one-fourth the size of the preceding feature map. The use of receptive fields, convolution, parameter sharing, and pooling are characteristics unique to CNNs.

Because feature maps are the result of spatial convolution, we know from Chapter 3 that they are simply filtered images. It then follows that pooled feature maps are filtered images of lower resolution. As Fig. 12.40 illustrates, the pooled feature maps in the first layer become the inputs to the next layer in the network. But, whereas we showed a single image as an input to the first layer, we now have multiple pooled feature maps (filtered images) that are inputs into the second layer.

To see how these multiple inputs to the second layer are handled, focus for a moment on one pooled feature map. To generate the values for the first feature map in the second convolutional layer, we perform convolution, add a bias, and use activation, as before. Then, we change the kernel and bias, and repeat the procedure for the second feature map, still using the same input. We do this for every remaining feature map, changing the kernel weights and bias for each. Then, we consider the next pooled feature map input and perform the same procedure (convolution, plus bias, plus activation) for every feature map in the second layer, using yet another set of different kernels and biases. When we are finished, we will have generated three values for the same location in every feature map, with one value coming from the corresponding location in each of the three inputs. The question now is: How do we combine these three individual values into one? The answer lies in the fact that convolution is a linear process, from which it follows that the three individual values are combined into one by superposition (that is, by adding them).

In the first layer, we had one input image and three feature maps, so we needed three kernels to complete all required convolutions. In the second layer, we have three inputs and seven feature maps, so the total number of kernels (and biases) needed is $3 \times 7 = 21$. Each feature map is pooled to generate a corresponding pooled feature map, resulting in seven pooled feature maps. In Fig. 12.40, there are only two layers, so these seven pooled feature maps are the outputs of the last layer.

As usual, the ultimate objective is to use features for classification, so we need a classifier. As Fig. 12.40 shows, in a CNN we perform classification by feeding the

You could interpret the convolution with several input images as 3-D convolution, but with movement only in the spatial (x and y) directions. The result would be identical to summing individual convolutions with each image separately, as we do here.

The parameters of the fully connected neural net are learned during training of the CNN, to be discussed shortly.

value of the last pooled layer into a fully connected neural net, the details of which you learned in Section 12.5. But the outputs of a CNN are 2-D arrays (i.e., filtered images of reduced resolution), whereas the inputs to a fully connected net are vectors. Therefore, we have to *vectorize* the 2-D pooled feature maps in the last layer. We do this using linear indexing (see Fig. 12.1). Each 2-D array in the last layer of the CNN is converted into a vector, then all resulting vectors are concatenated (vertically for a column) to form a single vector. This vector propagates through the neural net, as explained in Section 12.5. In any given application, the number of outputs in the fully connected net is equal to the number of pattern classes being classified. As before, the output with the highest value determines the class of the input.

EXAMPLE 12.14: Receptive fields, pooling neighborhoods, and their corresponding feature maps.

The top row of Fig. 12.41 shows a numerical example of the relative sizes of feature maps and pooled feature maps as a function of the sizes of receptive fields and pooling neighborhoods. The input image is of size 28×28 pixels, and the receptive field is of size 5×5 . If we require that the receptive field be contained in the image during convolution, you know from Section 3.4 that the resulting convolution array (feature map) will be of size 24×24 . If we use a pooling neighborhood of size 2×2 , the resulting pooled feature maps will be of size 12×12 , as the figure shows. As noted earlier, we assume that pooling neighborhoods do not overlap.

As an analogy with fully connected neural nets, think of each element of a 2-D array in the top row of Fig. 12.41 as a neuron. The outputs of the neurons in the input are pixel values. The neurons in the feature map of the first layer have output values generated by convolving with the input image a kernel whose size and shape are the same as the receptive field, and whose coefficients are learned during training. To each convolution value we add a bias and pass the result through an activation function to generate the output value of the corresponding neuron in the feature map. The output values of the neurons in the pooled feature maps are generated by pooling the output values of the neurons in the feature maps.

The second row in Fig. 12.41 illustrates visually how feature maps and pooled feature maps look based on the input image shown in the figure. The kernel shown is as described in the previous paragraph, and its weights (shown as intensity values) were learned from sample images using the training of the CNN described later in Example 12.17. Therefore, the nature of the learned features is determined by the learned kernel coefficients. Note that the contents of the feature maps are specific features detected by convolution. For example, some of the features emphasize edges in the character. As mentioned earlier, the pooled features are lower-resolution versions of this effect.

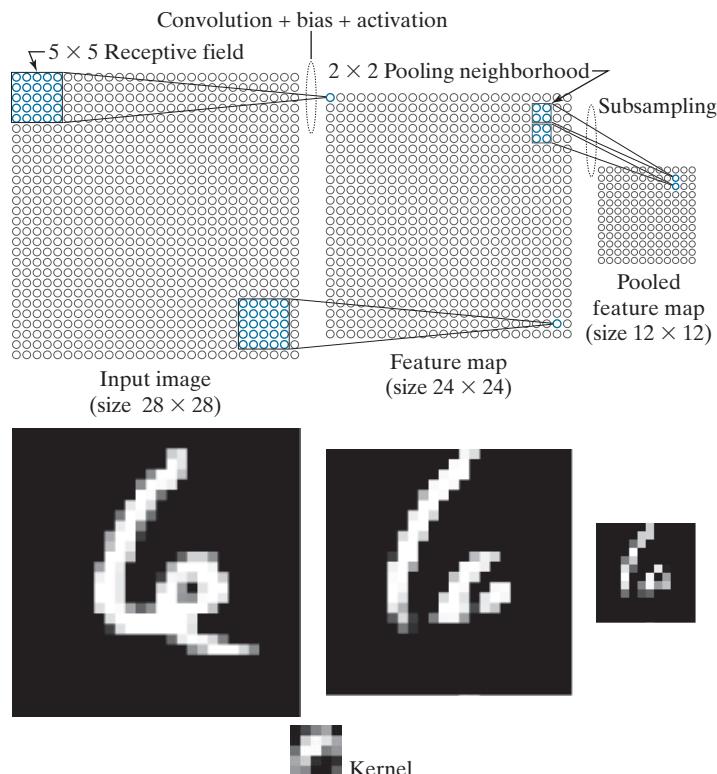
EXAMPLE 12.15: Graphical illustration of the functions performed by the components of a CNN.

Figure 12.42 shows the 28×28 image from Fig. 12.41, input into an expanded version of the CNN architecture from Fig. 12.40. The expanded CNN, which we will discuss in more detail in Example 12.17, has six feature maps in the first layer, and twelve in the second. It uses receptive fields of size 5×5 , and pooling neighborhoods of size 2×2 . Because the receptive fields are of size 5×5 , the feature maps in the first layer are of size 24×24 , as we explained in Example 12.14. Each feature map has its own set of weights and bias, so we will need a total of $(5 \times 5) \times 6 + 6 = 156$ parameters (six kernels with twenty-five weights each, and six biases) to generate the feature maps in the first layer. The top row of Fig. 12.43(a) shows the kernels with the weights learned during training of the CNN displayed as images, with intensity being proportional to kernel values.

FIGURE 12.41

Top row: How the sizes of receptive fields and pooling neighborhoods affect the sizes of feature maps and pooled feature maps.

Bottom row: An image example. This figure is explained in more detail in Example 12.17. (Image courtesy of NIST.)



Because we used pooling neighborhoods of size 2×2 , the pooled feature maps in the first layer of Fig. 12.42 are of size 12×12 . As we discussed earlier, the number of feature maps and pooled feature maps is the same, so we will have six arrays of size 12×12 acting as inputs to the twelve feature maps in the second layer (the number of feature maps generally is different from layer to layer). Each feature map will have its own set of weights and bias, so will need a total of $6 \times (5 \times 5) \times 12 + 12 = 1812$

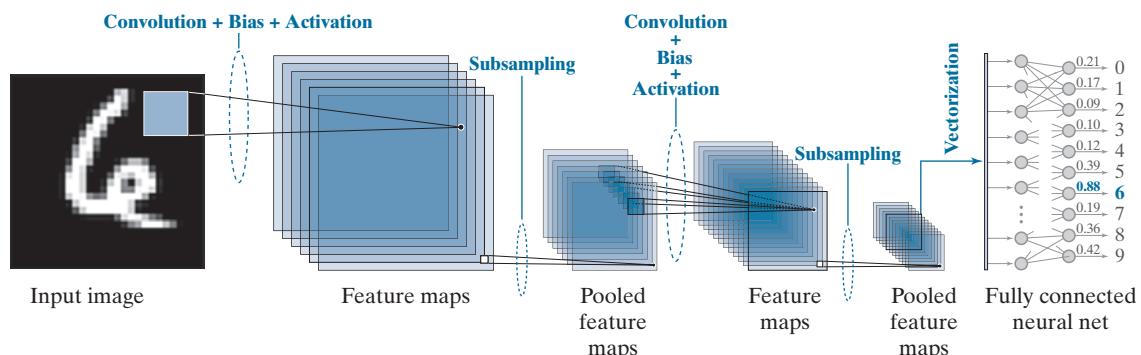


FIGURE 12.42 Numerical example illustrating the various functions of a CNN, including recognition of an input image. A sigmoid activation function was used throughout.

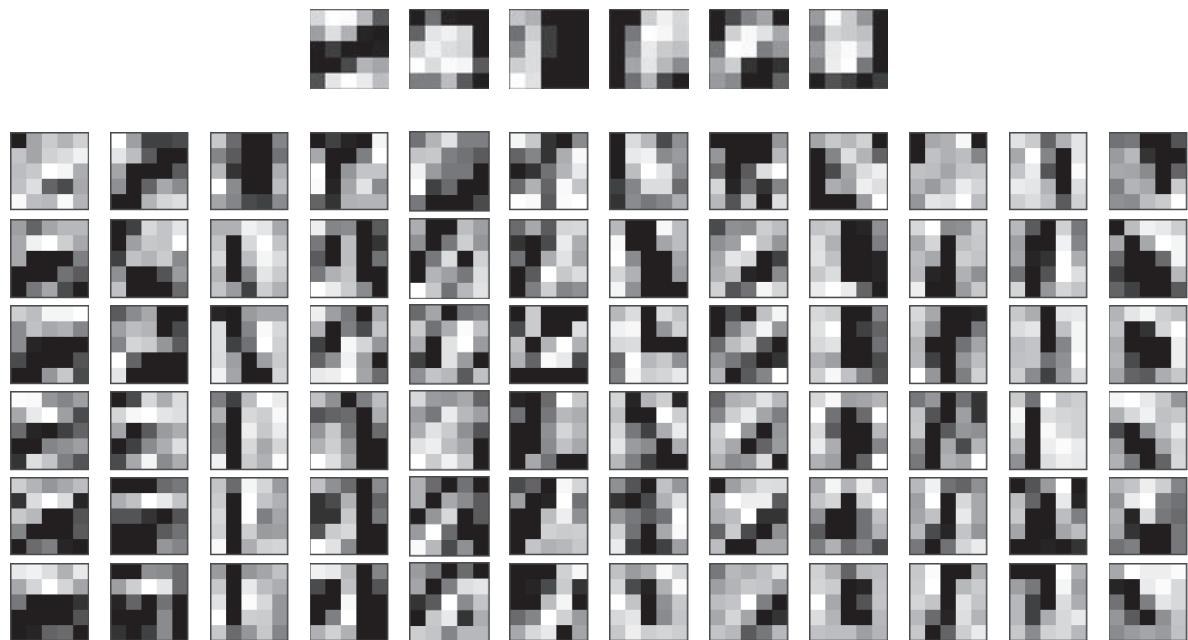


FIGURE 12.43 Top: The weights (shown as images of size 5×5) corresponding to the six feature maps in the first layer of the CNN in Fig. 12.42. Bottom: The weights corresponding to the twelve feature maps in the second layer.

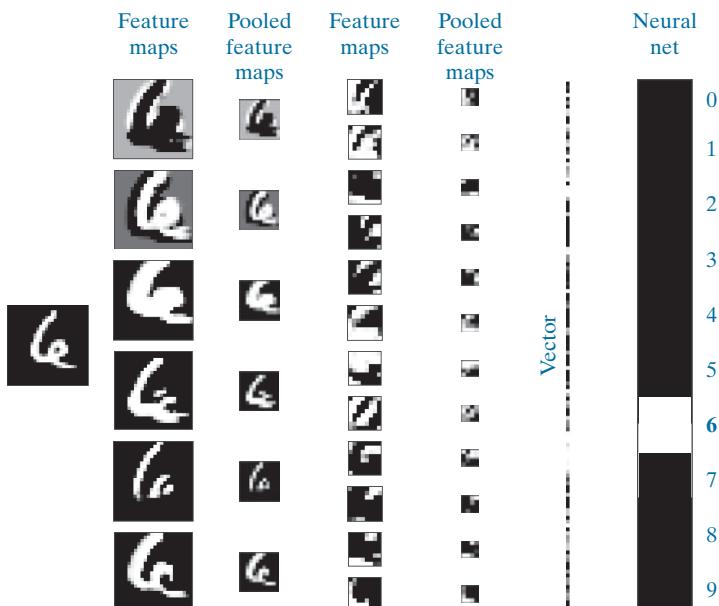
parameters to generate the feature maps in the second layer (i.e., twelve sets of six kernels with twenty-five weights each, plus twelve biases). The bottom part of Fig. 12.43 shows the kernels as images. Because we are using receptive fields of size 5×5 , the feature maps in the second layer are of size 8×8 . Using 2×2 pooling neighborhoods resulted in pooled feature maps of size 4×4 in the second layer.

As we discussed earlier, the pooled feature maps in the last layer have to be vectorized to be able to input them into the fully connected neural net. Each pooled feature map resulted in a column vector of size 16×1 . There are 12 of these vectors which, when concatenated vertically, resulted in a single vector of size 192×1 . Therefore, our fully connected neural net has 192 input neurons. There are ten numeral classes, so there are 10 output neurons. As you will see later, we obtained excellent performance by using a neural net with no hidden layers, so our complete neural net had a total of 192 input neurons and 10 output neurons. For the input character shown in Fig. 12.42, the highest value in the output of the fully connected neural net was in the seventh neuron, which corresponds to the class of 6's. Therefore, the input was recognized properly. This is shown in bold text in the figure.

Figure 12.44 shows graphically what the feature maps look like as the input image propagates through the CNN. Consider the feature maps in the first layer. If you look at each map carefully, you will notice that it highlights a different characteristic of the input. For example, the map on the top of the first column highlights the two principal edges on the top of the character. The second map highlights the edges of the entire inner region, and the third highlights a “blob-like” nature of the digit, almost as if it had been blurred by a lowpass kernel. The other three images show other features. Although the pooled feature maps are lower-resolution versions of the original feature maps, they still retained the key characteristics of the features in the latter. If you look at the first two feature maps in the second layer, and compare them with the first two in the first layer, you can see that they could be interpreted as higher-

FIGURE 12.44

Visual summary of an input image propagating through the CNN in Fig. 12.42. Shown as images are all the results of convolution (feature maps) and pooling (pooled feature maps) for both layers of the network. (Example 12.17 contains more details about this figure.)



level abstractions of the top part of the character, in the sense that they show an area flanked on both sides by areas of opposite intensity. These abstractions are not always easy to analyze visually, but as you will see in later examples, they can be very effective. The vectorized version of the last pooled layer is self-explanatory. The output of the fully connected neural net shows dark for low values and white for the highest value, indicating that the input was properly recognized as a number 6. Later in this section, we will show that the simple CNN architecture in Fig. 12.42 is capable of recognizing the correct class of over 70,000 numerical samples with nearly perfect accuracy.

Neural Computations in a CNN

Recall from Fig. 12.29 that the basic computation performed by an artificial neuron is a sum of products between weights and values from a previous layer. To this we add a bias and call the result the *net (total) input* to the neuron, which we denoted by z_i . As we showed in Eq. (12-54), the sum involved in generating z_i is a single sum. The computations performed in a CNN to generate a single value in a feature map is 2-D convolution. As you learned in Chapter 3, this is a double sum of products between the coefficients of a kernel and the corresponding elements of the image array overlapped by the kernel. With reference to Fig. 12.40, let w denote a kernel formed by arranging the weights in the shape of the receptive field we discussed in connection with that figure. For notational consistency with Section 12.5, let $a_{x,y}$ denote image or pooled feature values, depending on the layer. The convolution value at any point (x,y) in the input is given by

$$w \star a_{x,y} = \sum_l \sum_k w_{l,k} a_{x-l,y-k} \quad (12-83)$$

where l and k span the dimensions of the kernel. Suppose that w is of size 3×3 . Then, we can then expand this equation into the following sum of products:

$$\begin{aligned} w \star a_{x,y} &= w \star a_{x,y} = \sum_l \sum_k w_{l,k} a_{x-l,y-k} \\ &= w_{1,1} a_{x-1,y-1} + w_{1,2} a_{x-1,y-2} + \cdots + w_{3,3} a_{x-3,y-3} \end{aligned} \quad (12-84)$$

We could relabel the subscripts on w and a , and write instead

$$\begin{aligned} w \star a_{x,y} &= w_1 a_1 + w_2 a_2 + \cdots + w_9 a_9 \\ &= \sum_{i=1}^9 w_i a_i \end{aligned} \quad (12-85)$$

The results of Eqs. (12-84) and (12-85) are identical. If we add a bias to the latter equation and call the result z we have

$$\begin{aligned} z &= \sum_{j=1}^9 w_j a_j + b \\ &= w \star a_{x,y} + b \end{aligned} \quad (12-86)$$

The form of the first line of this equation is identical to Eq. (12-54). Therefore, we conclude that if we add a bias to the spatial convolution computation performed by a CNN at any fixed position (x, y) in the input, the result can be expressed in a form identical to the computation performed by an artificial neuron in a fully connected neural net. We need the x, y only to account for the fact that we are working in 2-D. If we think of z as the net input to a neuron, the analogy with the neurons discussed in Section 12.5 is completed by passing z through an activation function, h , to get the output of the neuron:

$$a = h(z) \quad (12-87)$$

This is exactly how the value of any point in a feature map (such as the point labeled A in Fig. 12.40) is computed.

Now consider point B in that figure. As mentioned earlier, its value is given by adding three convolution equations:

$$\begin{aligned} w_{l,k}^{(1)} \star a_{x,y}^{(1)} + w_{l,k}^{(2)} \star a_{x,y}^{(2)} + w_{l,k}^{(3)} \star a_{x,y}^{(3)} &= \sum_l \sum_k w_{l,k}^{(1)} a_{x-l,y-k}^{(1)} + \\ &\quad \sum_l \sum_k w_{l,k}^{(2)} a_{x-l,y-k}^{(2)} + \sum_l \sum_k w_{l,k}^{(3)} a_{x-l,y-k}^{(3)} \end{aligned} \quad (12-88)$$

where the superscripts refer to the three pooled feature maps in Fig. 12.40. The values of l, k, x , and y are the same in all three equations because all three kernels are of the same size and they move in unison. We could expand this equation and obtain a sum of products that is lengthier than for point A in Fig. 12.40, but we could still relabel all terms and obtain a sum of products that involves only one summation, *exactly* as before.

The preceding result tells us that the equations used to obtain the value of an element of any feature map in a CNN can be expressed in the form of the computation performed by an artificial neuron. This holds for any feature map, regardless of how many convolutions are involved in the computation of the elements of that feature map, in which case we would simply be dealing with the sum of more convolution equations. The implication is that we can use the basic form of Eqs. (12-86) and (12-87) to describe how the value of an element in any feature map of a CNN is obtained. This means we do not have to account explicitly for the number of different pooled feature maps (and hence the number of different kernels) used in a pooling layer. The result is a significant simplification of the equations that describe forward and backpropagation in a CNN.

Multiple Input Images

The values of $a_{x,y}$ just discussed are pixel values in the first layer but, in layers past the first, $a_{x,y}$ denotes values of pooled features. However, our equations do not differentiate based on what these variables actually represent. For example, suppose we replace the input to Fig. 12.40 with three images, such as the three components of an RGB image. The equations for the value of point A in the figure would now have the same form as those we stated for point B —only the weights and biases would be different. Thus, the results in the previous discussion for one input image are applicable directly to multiple input images. We will give an example of a CNN with three input images later in our discussion.

THE EQUATIONS OF A FORWARD PASS THROUGH A CNN

We concluded in the preceding discussion that we can express the result of convolving a kernel, w , and an input array with values $a_{x,y}$, as

$$\begin{aligned} z_{x,y} &= \sum_l \sum_k w_{l,k} a_{x-l,y-k} + b \\ &= w \star a_{x,y} + b \end{aligned} \quad (12-89)$$

where l and k span the dimensions of the kernel, x and y span the dimensions of the input, and b is a bias. The corresponding value of $a_{x,y}$ is

$$a_{x,y} = h(z_{x,y}) \quad (12-90)$$

But this $a_{x,y}$ is different from the one we used to compute Eq. (12-89), in which $a_{x,y}$ represents values from the previous layer. Thus, we are going to need additional notation to differentiate between layers. As in fully connected neural nets, we use ℓ for this purpose, and write Eqs. (12-89) and (12-90) as

$$\begin{aligned} z_{x,y}(\ell) &= \sum_l \sum_k w_{l,k}(\ell) a_{x-l,y-k}(\ell-1) + b(\ell) \\ &= w(\ell) \star a_{x,y}(\ell-1) + b(\ell) \end{aligned} \quad (12-91)$$

As noted earlier, a kernel is formed by organizing the weights in the shape of a corresponding receptive field. Also keep in mind that w and $a_{x,y}$ represent all the weights and corresponding values in a set of input images or pooled features.

and

$$a_{x,y}(\ell) = h(z_{x,y}(\ell)) \quad (12-92)$$

for $\ell = 1, 2, \dots, L_c$, where L_c is the number of convolutional layers, and $a_{x,y}(\ell)$ denotes the values of pooled features in convolutional layer ℓ . When $\ell = 1$,

$$a_{x,y}(0) = \{\text{values of pixels in the input image(s)}\} \quad (12-93)$$

When $\ell = L_c$,

$$a_{x,y}(L_c) = \{\text{values of pooled features in last layer of the CNN}\} \quad (12-94)$$

Note that ℓ starts at 1 instead of 2, as we did in Section 12.5. The reason is that we are naming layers, as in “convolutional layer ℓ .” It would be confusing to start at convolutional layer 2. Finally, we note that the pooling does not require any convolutions. The only function of pooling is to reduce the spatial dimensions of the feature map preceding it, so we do not include explicit pooling equations here.

Equations (12-91) through (12-94) are all we need to compute all values in a forward pass through the convolutional section of a CNN. As described in Fig. 12.40, the values of the pooled features of the last layer are vectorized and fed into a fully connected feedforward neural network, whose forward propagation is explained in Eqs. (12-54) and (12-55) or, in matrix form, in Table 12.2.

THE EQUATIONS OF BACKPROPAGATION USED TO TRAIN CNNs

As you saw in the previous section, the feedforward equations of a CNN are similar to those of a fully connected neural net, but with multiplication replaced by convolution, and notation that reflects the fact that CNNs are not fully connected in the sense defined in Section 12.5. As you will see in this section, the equations of backpropagation also are similar in many respects to those in fully connected neural nets.

As in the derivation of backpropagation in Section 12.5, we start with the definition of how the output error of our CNN changes with respect to each neuron in the network. The form of the error is the same as for fully connected neural nets, but now it is a function of x and y instead of j :

$$\delta_{x,y}(\ell) = \frac{\partial E}{\partial z_{x,y}(\ell)} \quad (12-95)$$

As in Section 12.5, we want to relate this quantity to $\delta_{xy}(\ell + 1)$, which we again do using the chain rule:

$$\delta_{x,y}(\ell) = \frac{\partial E}{\partial z_{x,y}(\ell)} = \sum_u \sum_v \frac{\partial E}{\partial z_{u,v}(\ell + 1)} \frac{\partial z_{u,v}(\ell + 1)}{\partial z_{x,y}(\ell)} \quad (12-96)$$

where u and v are any two variables of summation over the range of possible values of z . As noted in Section 12.5, these summations result from applying the chain rule.

By definition, the first term of the double summation of Eq. (12-96) is $\delta_{x,y}(\ell+1)$. So, we can write this equation as

$$\delta_{x,y}(\ell) = \frac{\partial E}{\partial z_{x,y}(\ell)} = \sum_u \sum_v \delta_{u,v}(\ell+1) \frac{\partial z_{u,v}(\ell+1)}{\partial z_{x,y}(\ell)} \quad (12-97)$$

Substituting Eq. (12-92) into Eq. (12-91), and using the resulting $z_{u,v}$ in Eq. (12-97), we obtain

$$\delta_{x,y}(\ell) = \sum_u \sum_v \delta_{u,v}(\ell+1) \frac{\partial}{\partial z_{x,y}(\ell)} \left[\sum_l \sum_k w_{l,k}(\ell+1) h(z_{u-l,v-k}(\ell)) + b(\ell+1) \right] \quad (12-98)$$

The derivative of the expression inside the brackets is zero unless $u-l=x$ and $v-k=y$, and because the derivative of $b(\ell+1)$ with respect to $z_{x,y}(\ell)$ is zero. But, if $u-l=x$ and $v-k=y$, then $l=u-x$ and $k=v-y$. Therefore, taking the indicated derivative of the expression in brackets, we can write Eq. (12-98) as

$$\delta_{x,y}(\ell) = \sum_u \sum_v \delta_{u,v}(\ell+1) \left[\sum_{u-x} \sum_{v-y} w_{u-x,v-y}(\ell+1) h'(z_{x,y}(\ell)) \right] \quad (12-99)$$

Values of x , y , u , and v are specified outside of the terms inside the brackets. Once the values of these variables are fixed, $u-x$ and $v-y$ inside the brackets are simply two *constants*. Therefore, the double summation evaluates to $w_{u-x,v-y}(\ell+1)h'(z_{x,y}(\ell))$, and we can write Eq. (12-99) as

$$\begin{aligned} \delta_{x,y}(\ell) &= \sum_u \sum_v \delta_{u,v}(\ell+1) w_{u-x,v-y}(\ell+1) h'(z_{x,y}(\ell)) \\ &= h'(z_{x,y}(\ell)) \sum_u \sum_v \delta_{u,v}(\ell+1) w_{u-x,v-y}(\ell+1) \end{aligned} \quad (12-100)$$

The double sum expression in the second line of this equation is in the form of a convolution, but the displacements are the negatives of those in Eq. (12-91). Therefore, we can write Eq. (12-100) as

$$\delta_{x,y}(\ell) = h'(z_{x,y}(\ell)) [\delta_{x,y}(\ell+1) \star w_{-x,-y}(\ell+1)] \quad (12-101)$$

The negatives in the subscripts indicate that w is *reflected* about both spatial axes. This is the same as rotating w by 180° , as we explained in connection with Eq. (3-35). Using this fact, we finally arrive at an expression for the error at a layer ℓ by writing Eq. (12-101) equivalently as

$$\delta_{x,y}(\ell) = h'(z_{x,y}(\ell)) [\delta_{x,y}(\ell+1) \star \text{rot180}(w_{x,y}(\ell+1))] \quad (12-102)$$

The 180° rotation is for each 2-D kernel in a layer.

But the kernels do not depend on x and y , so we can write this equation as

$$\delta_{x,y}(\ell) = h'(z_{x,y}(\ell)) \left[\delta_{x,y}(\ell+1) \star \text{rot180}(w(\ell+1)) \right] \quad (12-103)$$

As in Section 12.5, our final objective is to compute the change in E with respect to the weights and biases. Following a similar procedure as above, we obtain

$$\begin{aligned} \frac{\partial E}{\partial w_{l,k}} &= \sum_x \sum_y \frac{\partial E}{\partial z_{x,y}(\ell)} \frac{\partial z_{x,y}(\ell)}{\partial w_{l,k}} \\ &= \sum_x \sum_y \delta_{x,y}(\ell) \frac{\partial z_{x,y}(\ell)}{\partial w_{l,k}} \\ &= \sum_x \sum_y \delta_{x,y}(\ell) \frac{\partial}{\partial w_{l,k}} \left[\sum_l \sum_k w_{l,k}(\ell) h(z_{x-l,y-k}(\ell-1)) + b(\ell) \right] \quad (12-104) \\ &= \sum_x \sum_y \delta_{x,y}(\ell) h(z_{x-l,y-k}(\ell-1)) \\ &= \sum_x \sum_y \delta_{x,y}(\ell) a_{x-l,y-k}(\ell-1) \end{aligned}$$

where the last line follows from Eq. (12-92). This line is in the form of a convolution but, comparing it to Eq. (12-91), we see there is a sign reversal between the summation variables and their corresponding subscripts. To put it in the form of a convolution, we write the last line of Eq. (12-104) as

$$\begin{aligned} \frac{\partial E}{\partial w_{l,k}} &= \sum_x \sum_y \delta_{x,y}(\ell) a_{-(l-x),-(k-y)}(\ell-1) \\ &= \delta_{l,k}(\ell) \star a_{-l,-k}(\ell-1) \\ &= \delta_{l,k}(\ell) \star \text{rot180}(a(\ell-1)) \quad (12-105) \end{aligned}$$

Similarly (see Problem 12.32),

$$\frac{\partial E}{\partial b(\ell)} = \sum_x \sum_y \delta_{x,y}(\ell) \quad (12-106)$$

Using the preceding two expressions in the gradient descent equations (see Section 12.5), it follows that

$$\begin{aligned} w_{l,k}(\ell) &= w_{l,k}(\ell) - \alpha \frac{\partial E}{\partial w_{l,k}} \\ &= w_{l,k}(\ell) - \alpha \delta_{l,k}(\ell) \star \text{rot180}(a(\ell-1)) \quad (12-107) \end{aligned}$$

and

$$\begin{aligned} b(\ell) &= b(\ell) - \alpha \frac{\partial E}{\partial b(\ell)} \\ &= b(\ell) - \alpha \sum_x \sum_y \delta_{x,y}(\ell) \end{aligned} \quad (12-108)$$

Equations (12-107) and (12-108) update the weights and bias of each convolution layer in a CNN. As we have mentioned before, it is understood that the $w_{l,k}$ represents *all* the weights of a layer. The variables l and k span the spatial dimensions of the 2-D kernels, all of which are of the same size.

In a forward pass, we went from a convolution layer to a pooled layer. In backpropagation, we are going in the opposite direction. But the pooled feature maps are smaller than their corresponding feature maps (see Fig. 12.40). Therefore, when going in the reverse direction, we *upsample* (e.g., by pixel replication) each pooled feature map to match the size of the feature map that generated it. Each pooled feature map corresponds to a unique feature map, so the path of backpropagation is clearly defined.

With reference to Fig. 12.40, backpropagation starts at the output of the fully connected neural net. We know from Section 12.5 how to update the weights of this network. When we get to the “interface” between the neural net and the CNN, we have to reverse the vectorization method used to generate input vectors. That is, before we can proceed with backpropagation using Eqs. (12-107) and (12-108), we have to *regenerate* the individual pooled feature maps from the single vector propagated back by the fully connected neural net.

We summarized in Table 12.3 the backpropagation steps for a fully connected neural net. Table 12.6 summarizes the steps for performing backpropagation in the CNN architecture in Fig. 12.40. The procedure is repeated for a specified number of

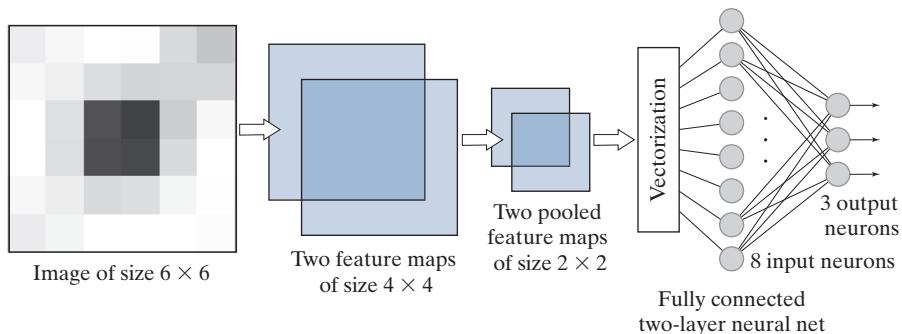
TABLE 12.6

The principal steps used to train a CNN. The network is initialized with a set of small random weights and biases. In backpropagation, a vector arriving (from the fully connected net) at the output pooling layer must be converted to 2-D arrays of the same size as the pooled feature maps in that layer. Each pooled feature map is upsampled to match the size of its corresponding feature map. The steps in the table are for one epoch of training.

Step	Description	Equations
Step 1	Input images	$a(0)$ = the set of image pixels in the input to layer 1
Step 2	Forward pass	For each neuron corresponding to location (x, y) in each feature map in layer ℓ compute: $z_{x,y}(\ell) = w(\ell) \star a_{x,y}(\ell-1) + b(\ell) \quad \text{and} \quad a_{x,y}(\ell) = h(z_{x,y}(\ell)); \ell = 1, 2, \dots, L_c$
Step 3	Backpropagation	For each neuron in each feature map in layer ℓ compute: $\delta_{x,y}(\ell) = h'(z_{x,y}(\ell)) [\delta_{x,y}(\ell+1) \star \text{rot180}(w(\ell+1))]; \ell = L_c - 1, L_c - 2, \dots, 1$
Step 4	Update parameters	Update the weights and bias for each feature map using $w_{l,k}(\ell) = w_{l,k}(\ell) - \alpha \delta_{l,k}(\ell) \star \text{rot180}(a(\ell-1)) \quad \text{and}$ $b(\ell) = b(\ell) - \alpha \sum_x \sum_y \delta_{x,y}(\ell); \ell = 1, 2, \dots, L_c$

FIGURE 12.45

CNN with one convolutional layer used to learn to recognize the images in Fig. 12.46.



epochs, or until the output error of the neural net reaches an acceptable value. The error is computed exactly as we did in Section 12.5. It can be the mean squared error, or the recognition error. Keep in mind that the weights in $w(\ell)$ and the bias value $b(\ell)$ are *different* for each feature map in layer ℓ .

EXAMPLE 12.16: Teaching a CNN to recognize some simple images.

We begin our illustrations of CNN performance by teaching the CNN in Fig. 12.45 to recognize the small 6×6 images in Fig. 12.46. As you can see on the left of this figure, there are three samples each of images of a horizontal stripe, a small centered square, and a vertical stripe. These images were used as the training set. On the right are noisy samples of images in these three categories. These were used as the test set.

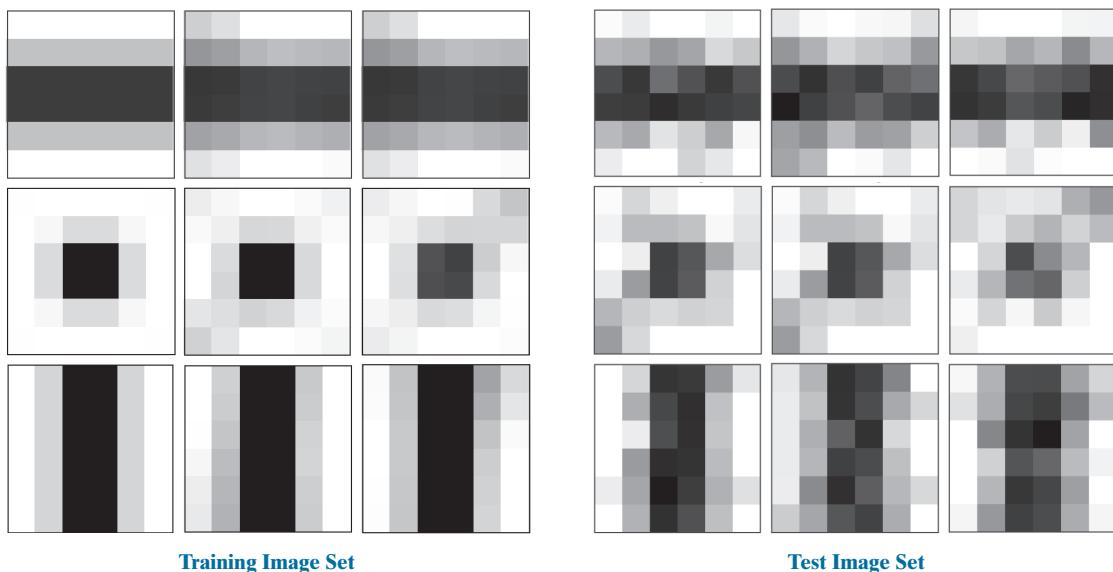
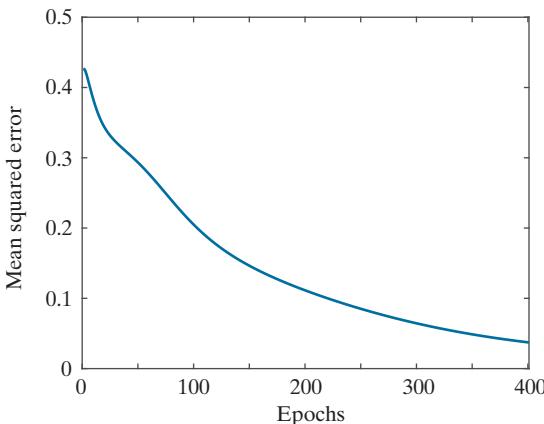


FIGURE 12.46 Left: Training images. Top row: Samples of a dark horizontal stripe. Center row: Samples of a centered dark square. Bottom row: Samples of a dark vertical stripe. Right: Noisy samples of the three categories on the left, created by adding Gaussian noise of zero mean and unit variance to the samples on the left. (All images are 8-bit grayscale images.)

FIGURE 12.47

Training MSE as a function of epoch for the images in Fig. 12.46. Perfect recognition of the training and test sets was achieved after approximately 100 epochs, despite the fact that the MSE was relatively high there.



As Fig. 12.45 shows, the inputs to our system are single images. We used a receptor field of size 3×3 , which resulted in feature maps of size 4×4 . There are two feature maps, which means we need two kernels of size 3×3 , and two biases. The pooled feature maps were generated using average pooling in neighborhoods of size 2×2 . This resulted in two pooled feature maps of size 2×2 , because the feature maps are of size 4×4 . The two pooled maps contain eight total elements which were organized as an 8-D column vector to vectorize the output of the last layer. (We used linear indexing of each image, then concatenated the two resulting 4-D vectors into a single 8-D vector.) This vector was then fed into the fully connected neural net on the right, which consists of the input layer and a three-neuron output layer, one neuron per class. Because this network has no hidden layers, it implements linear decision functions (see Problem 12.18). To train the system, we used $\alpha = 1.0$ and ran the system for 400 epochs. Figure 12.47 is a plot of the MSE as a function of epoch. Perfect recognition of the training set was achieved after approximately 100 epochs of training, despite the fact that the MSE was relatively high there. Recognition of the test set was 100% as well. The kernel and bias values learned by the system were:

$$\mathbf{w}_1 = \begin{bmatrix} 3.0132 & 1.1808 & -0.0945 \\ 0.9718 & 0.7087 & -0.9093 \\ 0.7193 & 0.0230 & -0.8833 \end{bmatrix}, b_1 = -0.2990 \quad \mathbf{w}_2 = \begin{bmatrix} -0.7388 & 1.8832 & 4.1077 \\ -1.0027 & 0.3908 & 2.0357 \\ -1.2164 & -1.1853 & -0.1987 \end{bmatrix}, b_2 = -0.2834$$

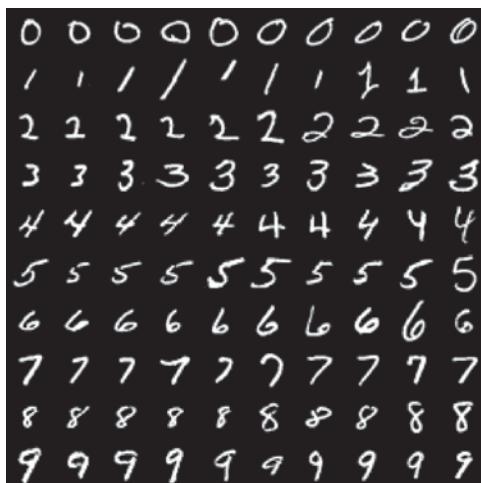
It is important that the CNN learned these parameters automatically from the raw training images. No features in the sense discussed in Chapter 11 were employed.

EXAMPLE 12.17: Using a large training set to teach a CNN to recognize handwritten numerals.

In this example, we look at a more practical application using a database containing 60,000 training and 10,000 test images of handwritten numeric characters. The content of this database, called the *MNIST database*, is similar to a database from NIST (National Institute of Standards and Technology). The former is a “cleaned up” version of the latter, in which the characters have been centered and formatted into grayscale images of size 28×28 pixels. Both databases are freely available online. Figure 12.48 shows examples of typical numeric characters available in the databases. As you can see, there is

FIGURE 12.48

Samples similar to those available in the NIST and MNIST databases. Each character subimage is of size 28×28 pixels. (Individual images courtesy of NIST.)



significant variability in the characters—and this is just a small sampling of the 70,000 characters available for experimentation.

Figure 12.49 shows the architecture of the CNN we trained to recognize the ten digits in the MNIST database. We trained the system for 200 epochs using $\alpha = 1.0$. Figure 12.50 shows the training MSE as a function of epoch for the 60,000 training images in the MNIST database.

Training was done using mini batches of 50 images at a time to improve the learning rate (see the discussion in Section 12.7). We also classified all images of the training set and all images of the test set after each epoch of training. The objective of doing this was to see how quickly the system was learning the characteristics of the data. Figure 12.51 shows the results. A high level of correct recognition performance was achieved after relatively few epochs for both data sets, with approximately 98% correct recognition achieved after about 40 epochs. This is consistent with the training MSE in Fig. 12.50, which dropped quickly, then began a slow descent after about 40 epochs. Another 160 epochs of training were required for the system to achieve recognition of about 99.9%. These are impressive results for such a small CNN.

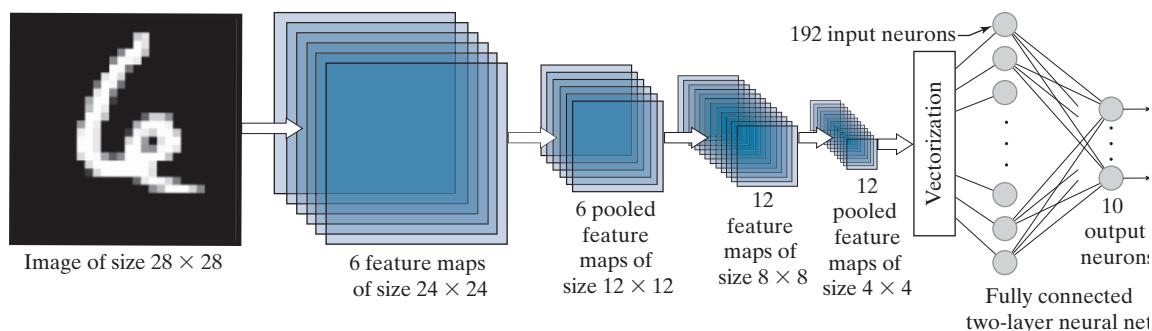


FIGURE 12.49 CNN used to recognize the ten digits in the MNIST database. The system was trained with 60,000 numerical character images of the same size as the image shown on the left. This architecture is the same as the architecture we used in Fig. 12.42. (Image courtesy of NIST.)

FIGURE 12.50

Training mean squared error as a function of epoch for the 60,000 training digit images in the MNIST database.

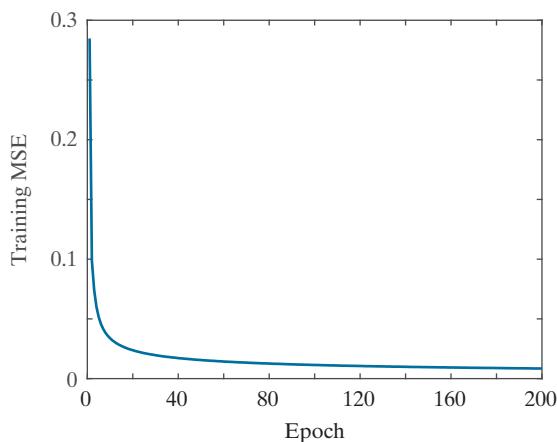


Figure 12.52 shows recognition performance on each digit class for both the training and test sets. The most revealing feature of these two graphs is that the CNN did equally as well on both sets of data. This is a good indication that the training was successful, and that it generalized well to digits it had not seen before. This is an example of the neural network not “over-fitting” the data in the training set.

Figure 12.53 shows the values of the kernels for the first feature map, displayed as intensities. There is one input image and six feature maps, so six kernels are required to generate the feature maps of the first layer. The dimensions of the kernels are the same as the receptive field, which we set at 5×5 . Thus, the first image on the left in Fig. 12.53 is the 5×5 kernel corresponding to the first feature map. Figure 12.54 shows the kernels for the second layer. In this layer, we have six inputs (which are the pooled maps of the first layer) and twelve feature maps, so we need a total of $6 \times 12 = 72$ kernels and biases to generate the twelve feature maps in the second layer. Each column of Fig. 12.54 shows the six

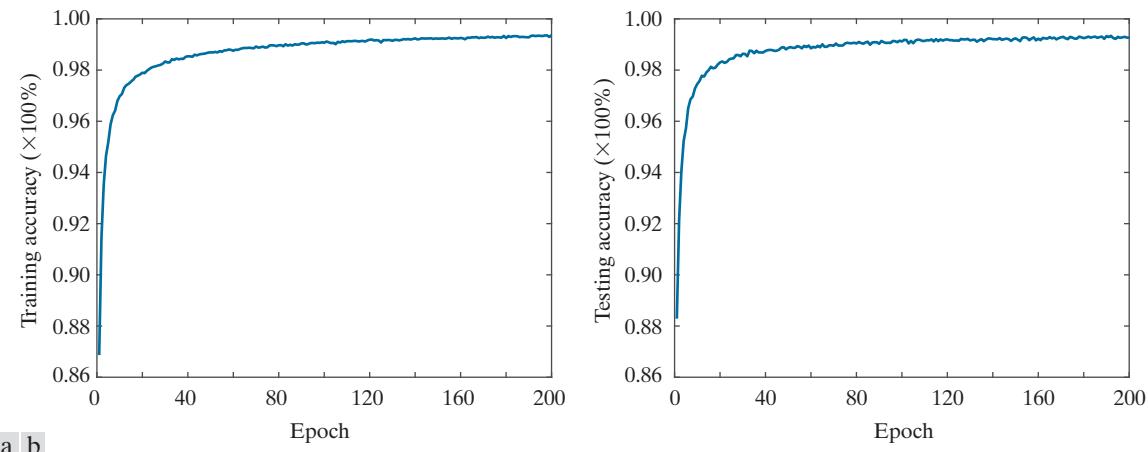


FIGURE 12.51 (a) Training accuracy (percent correct recognition of the training set) as a function of epoch for the 60,000 training images in the MNIST database. The maximum achieved was 99.36% correct recognition. (b) Accuracy as a function of epoch for the 10,000 test images in the MNIST database. The maximum correct recognition rate was 99.13%.

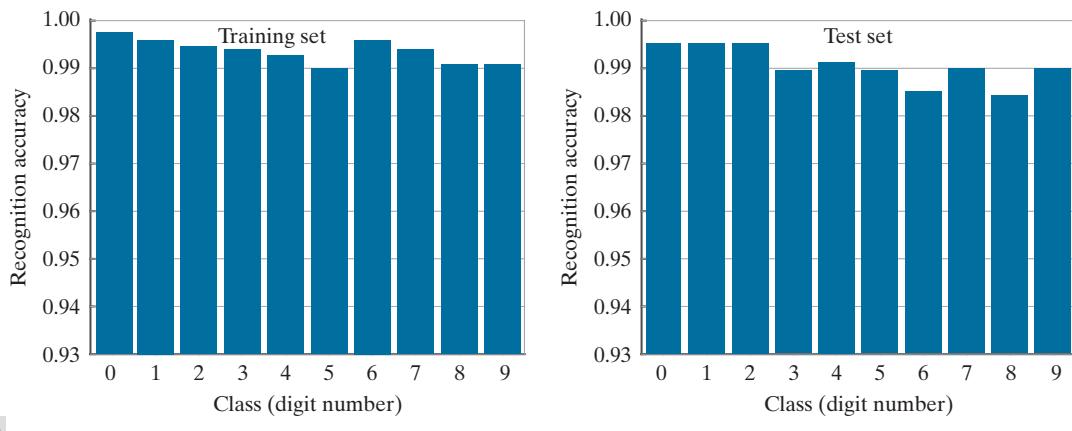


FIGURE 12.52 (a) Recognition accuracy of training set by image class. Each bar shows a number between 0 and 1. When multiplied by 100%, these numbers give the correct recognition percentage for that class. (b) Recognition results per class in the test set. In both graphs the recognition rate is above 98%.

5×5 kernels corresponding to one of the feature maps in the second layer. We used 2×2 pooling in both layers, resulting in a 50% reduction of each of the two spatial dimensions of the feature maps.

Finally, it is of interest to visualize how one input image proceeds through the network, using the kernels learned during training. Figure 12.55 shows an input digit image from the test set, and the computations performed by the CNN at each layer. As before, we display numerical results as intensities.

Consider the results of convolution in the first layer. If you look at each resulting feature map carefully, you will notice that it highlights a different characteristic of the input. For example, the feature map on the top of the first column highlights the two vertical edges on the top of the character. The second highlights the edges of the entire inner region, and the third highlights a “blob-like” feature of the digit, as if it had been blurred by a lowpass kernel. The other three feature maps show other features. If you now look at the first two feature maps in the second layer, and compare them with the first feature map in the first layer, you can see that they could be interpreted as higher-level abstractions of the top of the character, in the sense that they show a dark area flanked on each side by white areas. Although these abstractions are not always easy to analyze visually, this example clearly demonstrates that they can be very effective. And, remember the important fact that our simple system learned these features automatically from 60,000 training images. This capability is what makes convolutional networks so powerful when it comes to image pattern classification. In the next example, we will consider even more complex images, and show some of the limitations of our simple CNN architecture.

EXAMPLE 12.18: Using a large image database to teach a CNN to recognize natural images.

In this example, we trained the same CNN architecture as in Fig. 12.49, but using the RGB color images in Fig. 12.56. These images are representative of those found in the CIFAR-10 database, a popular database used to test the performance of image classification systems. Our objective was to test the limitations of the CNN architecture in Fig. 12.49 by training it with data that is significantly more complex than the MNIST images in Example 12.17. The only difference between the architecture needed to

FIGURE 12.53

Kernels of the first layer after 200 epochs of training, shown as images.

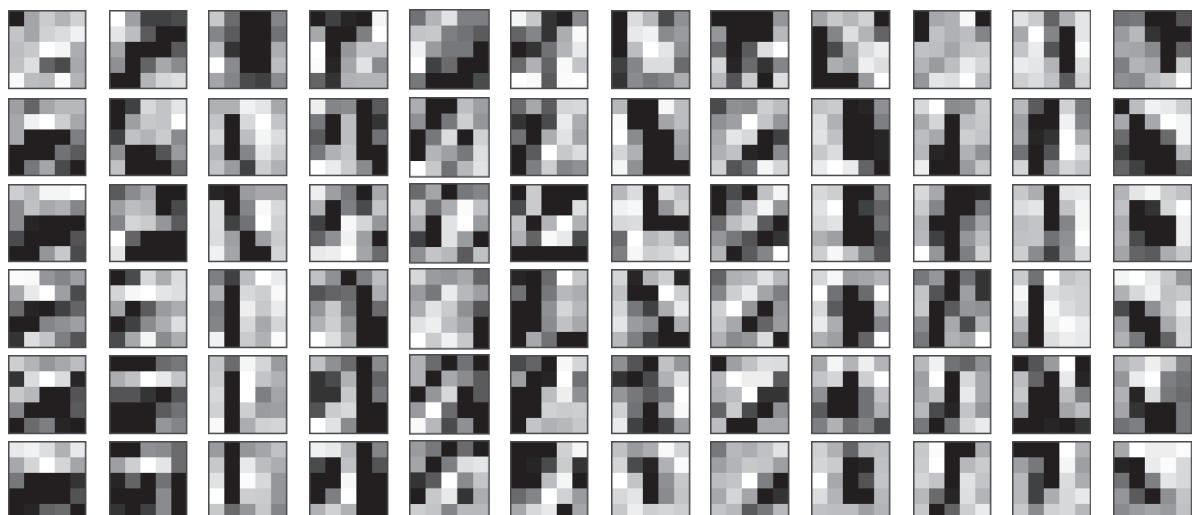
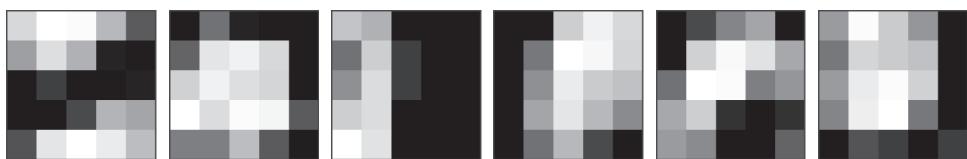


FIGURE 12.54 Kernels of the second layer after 200 epochs of training, displayed as images of size 5×5 . There are six inputs (pooled feature maps) into the second layer. Because there are twelve feature maps in the second layer, the CNN learned the weights of $6 \times 12 = 72$ kernels.

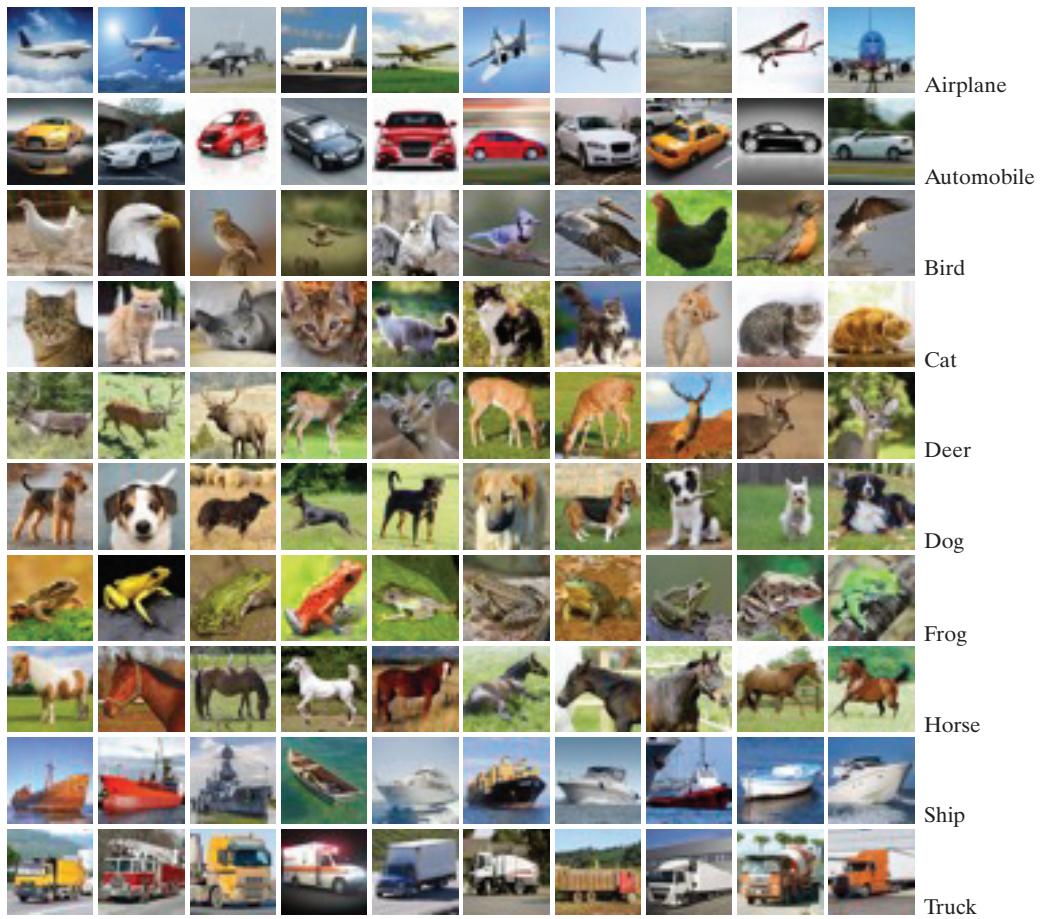
FIGURE 12.55

Results of a forward pass for one digit image through the CNN in Fig. 12.49 after training. The feature maps were generated using the kernels from Figs. 12.53 and 12.54, followed by pooling. The neural net is the two-layer neural network from Fig. 12.49. The output high value (in white) indicates that the CNN recognized the input properly. (This figure is the same as Fig. 12.44.)

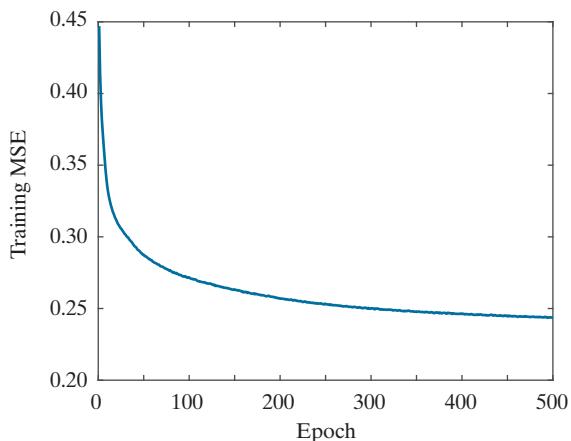


FIGURE 12.56

Mini images of size 32×32 pixels, representative of the 50,000 training and 10,000 test images in the CIFAR-10 database (the 10 stands for ten classes). The class names are shown on the right. (Images courtesy of Pearson Education.)

**FIGURE 12.57**

Training mean squared error as a function of the number of epochs for a training set of 50,000 CIFAR-10 images.



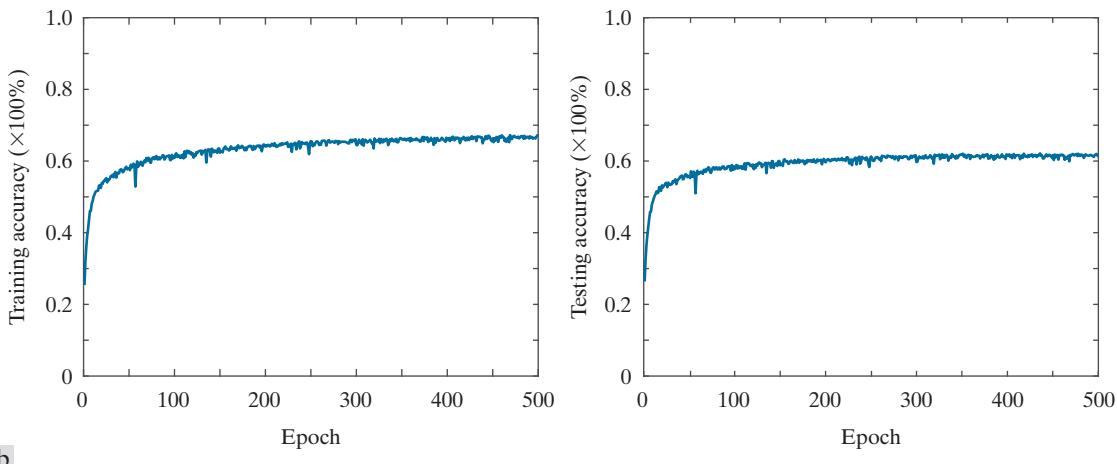


FIGURE 12.58 (a) Training accuracy (percent correct recognition of the training set) as a function of epoch for the 50,000 training images in the CIFAR-10 database. (b) Accuracy as a function of epoch for the 10,000 CIFAR-10 test images.

process the CIFAR-10 images, and the architecture in Fig. 12.49, is that the CIFAR-10 images are RGB color images, and hence have three channels. We worked with these input images using the approach explained in the subsection entitled Multiple Input Images, on page 973.

We trained the modified CNN for 500 epochs using the 50,000 training images of the CIFAR-10 database. Figure 12.57 is a plot of the mean squared error as a function of epoch during the training phase. Observe that the MSE begins to plateau at a value of approximately 0.25. In contrast, the MSE plot in Fig. 12.50 for the MNIST data achieved a much lower final value. This is not unexpected, given that the CIFAR-10 images are significantly more complex, both in the objects of interest as well as their backgrounds. The lower expected recognition performance of the training set is confirmed by the training-accuracy plotted in Fig. 12.58(a) as a function of epoch. The recognition rate leveled-off around 68% for the training data and about 61% for the test data. Although these results are not nearly as good as those obtained for the MNIST data, they are consistent with what we would expect from a very basic network. It is possible to achieve over 96% accuracy on this database (see Graham [2015]), but that requires a more complex network and a different pooling strategy.

Figure 12.59 shows the recognition accuracy per class for the training and test image sets. With a few exceptions, the highest recognition rate in both the training and test sets was achieved for engineered objects, and the lowest was for small animals. Frogs were an exception, caused most likely by the fact that frog size and shape are more consistent than they are, for example, in dogs and birds. As you can see in Fig. 12.59, if the small animals were removed from the list, recognition performance on the rest of the images would have been considerably higher.

Figures 12.60 and Fig. 12.61 show the kernels of the first and second layers. Note that each column in Fig. 12.60 has three 5×5 kernels. This is because there are three input channels to the CNN in this example. If you look carefully at the columns in Fig. 12.60, you can detect a similarity in the arrangement and values of the coefficients. Although it is not obvious what the kernels are detecting, it is clear that they are consistent in each column, and that all columns are quite different from each other, indicating a capability to detect different features in the input images. We show Fig. 12.61 for completeness only,

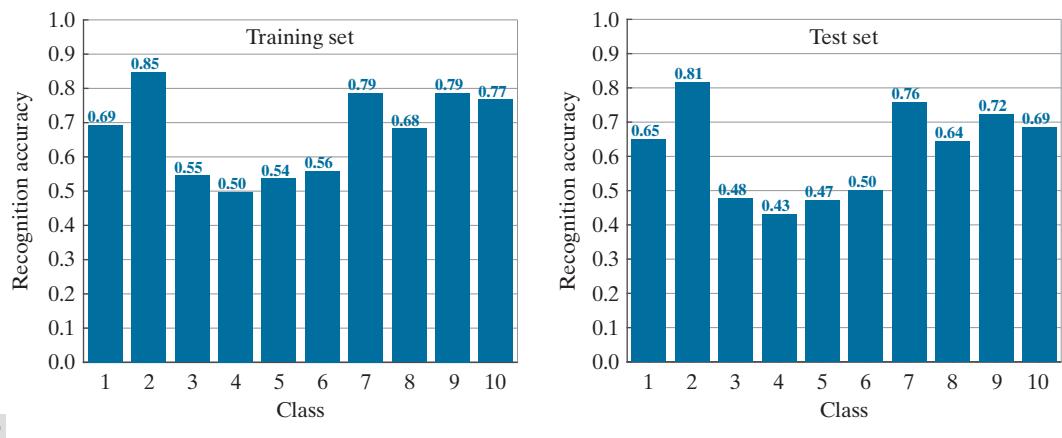
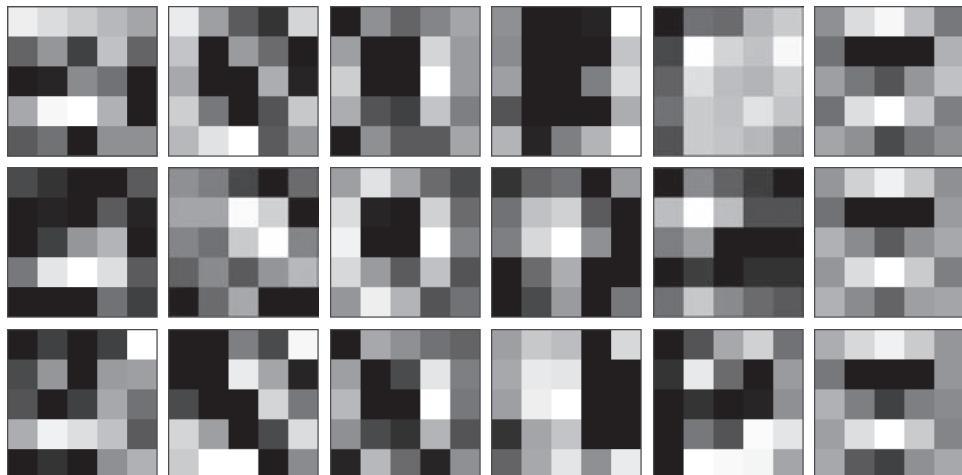


FIGURE 12.59 (a) CIFAR-10 recognition rate of training set by image class. Each bar shows a number between 0 and 1. When multiplied by 100%, these numbers give the correct recognition percentage for that class. (b) Recognition results per class in the test set.

as there is little we can infer that deep into the network, especially at this small scale, and considering the complexity of the images in the training set. Finally, Fig. 12.62 shows a complete recognition pass through the CNN using the weights in Figs. 12.60 and 12.61. The input shows the three color channels of the RGB image in the seventh column of the first row in Fig. 12.56. The feature maps in the first column, show the various features extracted from the input. The second column shows the pooling results, zoomed to the size of the features maps for clarity. The third and fourth columns show the results in the second layer, and the fifth column shows the vectorized output. Finally, the last column shows the result of recognition, with white representing a high output, and the others showing much smaller values. The input image was properly recognized as belonging to class 1.

FIGURE 12.60

Weights of the kernels of the first convolution layer after 500 epochs of training.



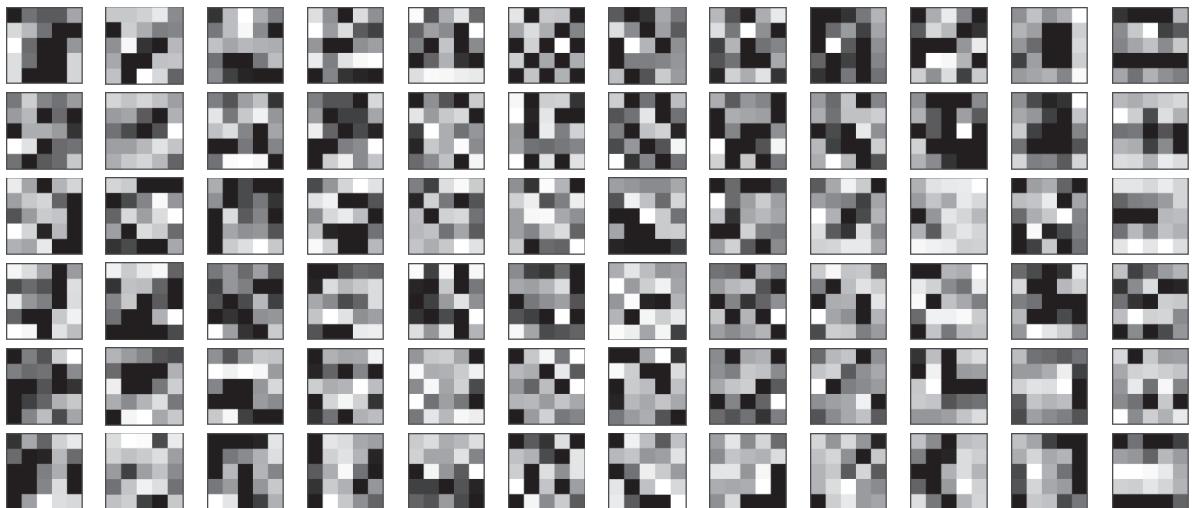


FIGURE 12.61 Weights of the kernels of the second convolution layer after 500 epochs of training. The interpretation of these kernels is the same as in Fig. 12.54.

12.7 SOME ADDITIONAL DETAILS OF IMPLEMENTATION

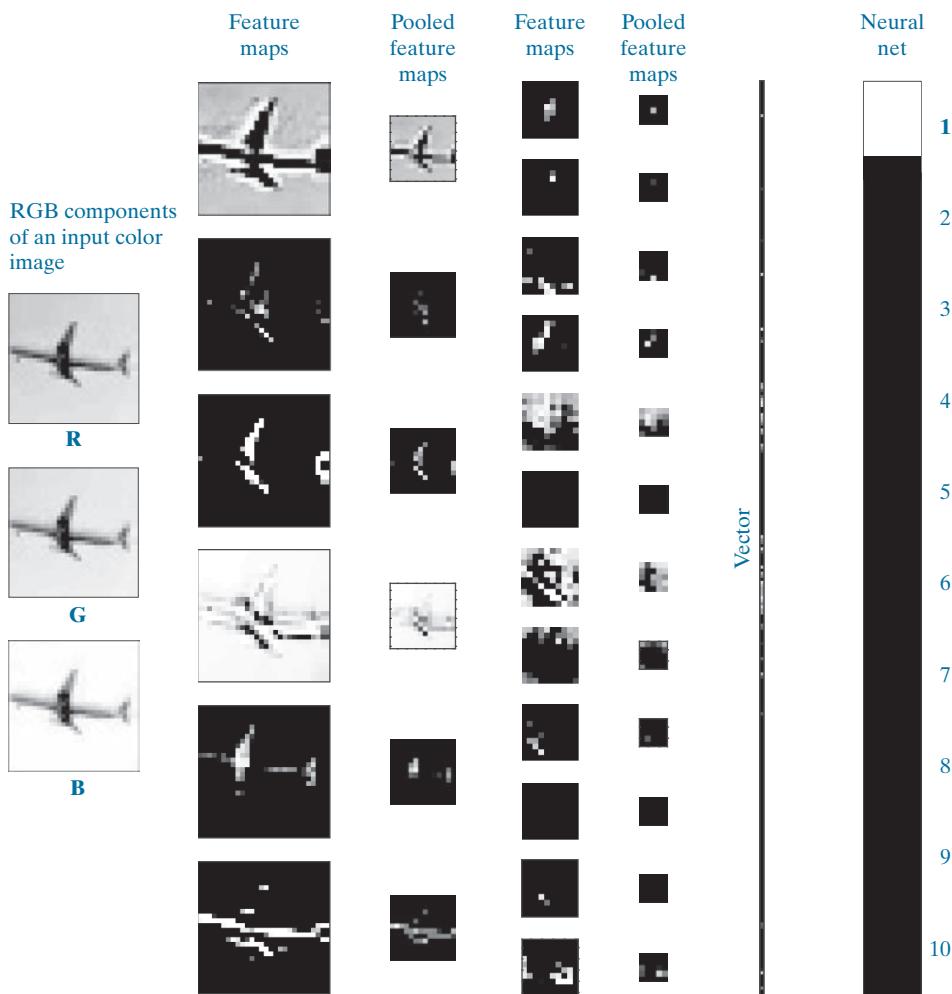
We mentioned in the previous section that neural (including convolutional) nets have the ability to learn features directly from training data, thus reducing the need for “engineered” features. While this is a significant advantage, it does not imply that the design of a neural network is free of human input. On the contrary, designing complex neural networks requires significant skill and experimentation.

In the last two sections, our focus was on the development of fundamental concepts in neural nets, with an emphasis on the derivation of backpropagation for both fully connected and convolutional nets. Backpropagation is the backbone of neural net design, but there are other important considerations that influence how well a neural net learns, and then generalizes to patterns it has not seen before. In this section, we discuss briefly some important aspects in the design of fully connected and convolutional neural networks.

One of the first questions when designing a neural net architecture is how many layers to specify for the network. Theoretically, the *universality approximation theorem* (Cybenko [1989]) tells us that, under mild conditions, arbitrarily complex decision functions can be *approximated* by a continuous feedforward neural network with a single hidden layer. Although the theorem does not tell us how to compute the parameters of that single hidden layer, it does indicate that structurally simple neural nets can be very powerful. You have seen this in some of the examples in the last two sections. Experimental evidence suggests that deep neural nets (i.e., networks with two or more hidden layers) are better than a single hidden layer network at learning abstract representations, which typically is the main point of learning. There is no such thing as an algorithm to determine the “optimum” number of layers to use in a neural network. Therefore, specifying the number of layers generally

FIGURE 12.62

Graphical illustration of a forward pass through the trained CNN. The purpose was to recognize one input image from the set in Fig. 12.56. As the output shows, the image was recognized correctly as belonging to class 1, the class of airplanes. (Original image courtesy of Pearson Education.)



is determined by a combination of experience and experimentation. “Starting small” is a logical approach to this problem. The more layers a network has, the higher the probability that backpropagation will run into problems such as so-called *vanishing gradients*, where gradient values are so small that gradient descent ceases to be effective. In convolutional networks, we have the added issue that the size of the inputs decreases as the images propagate through the network. There are two causes for this. The first is a natural size reduction caused by convolution itself, with the amount of reduction being proportional to the size of the receptive fields. One solution is to use *padding* prior to performing convolution operations, as we discussed in Section 3.4. The second (and most significant) cause of size reduction is pooling. The minimum pooling neighborhood is of size 2×2 , which reduces the size of feature maps by three-quarters at each layer. A solution that helps is to *upsample* the input

images, but this must be done with care because the relative sizes of features of interest would increase proportionally, thus influencing the size selected for receptive fields.

After the number of layers has been specified, the next task is to specify the number of neurons per layer. We always know how many neurons are needed in the first and last layers, but the number of neurons for internal layer is also an open question with no theoretical “best” answer. If the objective is to keep the number of layers as small as possible, the power of the network is increased to some degree by increasing the number of neurons per layer.

The main aspects of specifying the architecture of a neural network are completed by specifying the activation function. In this chapter, we worked with sigmoid functions for consistency between examples, but there are applications in which hyperbolic tangent and ReLU activation functions are superior in terms of improving training performance.

Once a network architecture has been specified, training is the central aspect of making the architecture useful. Although the networks we discussed in this chapter are relatively simple, networks applied to very large-scale problems can have millions of nodes and require large blocks of time to train. When available, the parameters of a *pretrained* network are an ideal starting point for further training, or for validating recognition performance. Another central theme in training neural nets is the use of GPUs to accelerate matrix operations.

An issue often encountered in training is *over-fitting*, in which recognition of the training set is acceptable, but the recognition rate on samples not used for training is much lower. That is, the net is not able to *generalize* what it learned and apply it to inputs it has not encountered before. When additional training data is not available, the most common approach is to artificially enlarge the training set using transformations such as geometric distortions and intensity variations. The transformations are carried out while preserving the class membership of the transformed patterns. Another major approach is to use *dropout*, a technique that randomly drops nodes with their connections from a neural network during training. The idea is to change the architecture slightly to prevent the net from adapting too much to a fixed set of parameters (see Srivastava et al. [2014]).

In addition to computational speed, another important aspect of training is efficiency. Simple things, such as shuffling the input patterns at the beginning of each training epoch can reduce or eliminate the possibility of “cycling,” in which parameter values repeat at regular intervals. *Stochastic gradient descent* is another important training refinement in which, instead of using the entire training set, samples are selected randomly and input into the network. You can think of this as dividing the training set into *mini-batches*, and then choosing a single sample from each mini-batch. This approach often results in speedier convergence during training.

In addition to the above topics, a paper by LeCun et al. [2012] is an excellent overview of the types of considerations introduced in the preceding discussion. In fact, the breadth spanned by these topics is extensive enough to be the subject of an entire book (see Montavon et al. [2012]). The neural net architectures we discussed were by necessity limited in scope. You can get a good idea of the practical requirements of implementing practical networks by reading a paper by Krizhevsky, Sutskever, and Hinton [2012], which summarizes the design and implementation of a large-scale, deep convolutional neural network. There are a multitude of designs that have

been implemented over the past decade, including commercial and free implementations. A quick internet search will reveal a multitude of available architectures.

Summary, References, and Further Reading

Background material for Sections 12.1 through 12.4 are the books by Theodoridis and Koutroumbas [2006], by Duda, Hart, and Stork [2001], and by Tou and Gonzalez [1974]. For additional reading on the material on matching shape numbers see Bribiesca and Guzman [1980]. On string matching, see Sze and Yang [1981]. A significant portion of this chapter was devoted to neural networks. This is a reflection of the fact that neural nets, and in particular convolutional neural nets, have made significant strides in the past decade in solving image pattern classifications problems. As in the rest of the book, our presentation of this topic focused on fundamentals, but the topics covered were thoroughly developed. What you have learned in this chapter is a solid foundation for much of the work being conducted in this area. As we mentioned earlier, the literature on neural nets is vast, and quickly growing. As a starting point, a basic book by Nielsen [2015] provides an excellent introduction to the topic. The more advanced book by Goodfellow, Bengio, and Courville [2016] provides more depth into the mathematical underpinning of neural nets. Two classic papers worth reading are by Rumelhart, Hinton, and Williams [1986], and by LeCun, Bengio, and Haffner [1998]. The LeNet architecture we discussed in Section 12.6 was introduced in the latter reference, and it is still a foundation for image pattern classification. A recent survey article by LeCun, Bengio, and Hinton [2015] gives an interesting perspective on the scope of applicability of neural nets in general. The paper by Krizhevsky, Sutskever, and Hinton [2012] was one of the most important catalysts leading to the significant increase in the present interest on convolutional networks, and on their applicability to image pattern classification. This paper is also a good overview of the details and techniques involved in implementing a large-scale convolutional neural network. For details on the software aspects of many of the examples in this chapter, see Gonzalez, Woods, and Eddins [2009].

Problems

Solutions to the problems marked with an asterisk (*) are in the DIP4E Student Support Package (consult the book website: www.ImageProcessingPlace.com).

12.1 Do the following:

- (a)* Compute the decision functions of a minimum distance classifier for the patterns in Fig. 12.10. You may obtain the required mean vectors by (careful) inspection.
- (b) Sketch the decision boundary implemented by the decision functions in (a).

12.2* Show that Eqs. (12-3) and (12-4) perform the same function in terms of pattern classification.

12.3 Show that the boundary given by Eq. (12-8) is the perpendicular bisector of the line joining the n -dimensional points \mathbf{m}_i and \mathbf{m}_j .

12.4* Show how the minimum distance classifier discussed in connection with Fig. 12.11 could be implemented by using N_c resistor banks (N_c is the number of classes), a summing junction at

each bank (for summing currents), and a maximum selector capable of selecting the maximum value of N_c decision functions in order to determine the class membership of a given input.

12.5* Show that the correlation coefficient of Eq. (12-10) has values in the range $[-1, 1]$. (Hint: Express γ in vector form.)

12.6 Show that the distance measure $D(a,b)$ in Eq. (12-12) satisfies the properties in Eq. (12-13).

12.7* Show that $\beta = \max(|a|, |b|) - \alpha$ in Eq. (12-14) is 0 if and only if a and b are identical strings.

12.8 Carry out the manual computations that resulted in the mean vector and covariance matrices in Example 12.5.

12.9* The following pattern classes have Gaussian probability density functions:

$$\begin{aligned}c_1 &: \{(0,0)^T, (2,0)^T, (2,2)^T, (0,2)^T\} \\c_2 &: \{(4,4)^T, (6,4)^T, (6,6)^T, (4,6)^T\}\end{aligned}$$

- (a) Assume that $P(c_1) = P(c_2) = 1/2$ and obtain the equation of the Bayes decision boundary between these two classes.
 (b) Sketch the boundary.

12.10 Repeat Problem 12.9, but use the following pattern classes:

$$\begin{aligned}c_1 &: \{(-1,0)^T, (0,-1)^T, (1,0)^T, (0,1)^T\} \\c_2 &: \{(-2,0)^T, (0,-2)^T, (2,0)^T, (0,2)^T\}\end{aligned}$$

Note that the classes are not linearly separable.

12.11 With reference to the results in Table 12.1, compute the overall correct recognition rate for the patterns of the training set. Repeat for the patterns of the test set.

12.12* We derived the Bayes decision functions

$$d_j(\mathbf{x}) = p(\mathbf{x}/c_j)P(c_j), j = 1, 2, \dots, N_c$$

using a 0-1 loss function. Prove that these decision functions minimize the probability of error. (*Hint:* The probability of error $p(e)$ is $1 - p(c)$, where $p(c)$ is the probability of being correct. For a pattern vector \mathbf{x} belonging to class c_i , $p(\mathbf{x}/c_i) = p(c_i/\mathbf{x})$. Find $p(c)$ and show that $p(c)$ is maximum [$p(e)$ is minimum] when $p(\mathbf{x}/c_i)P(c_i)$ is maximum.)

12.13 Finish the computations started in Example 12.7.

12.14* The perceptron algorithm given in Eqs. (12-44) through (12-46) can be expressed in a more concise form by multiplying the patterns of class c_2 by -1 , in which case the correction steps in the algorithm become $\mathbf{w}(k+1) = \mathbf{w}(k)$, if $\mathbf{w}^T(k)\mathbf{y}(k) > 0$, and $\mathbf{w}(k+1) = \mathbf{w}(k) + \alpha\mathbf{y}(k)$ otherwise, where we use \mathbf{y} instead of \mathbf{x} to make it clear that the patterns of class c_2 were multiplied by -1 . This is one of several perceptron algorithm formulations that can be derived starting from the general gradient descent equation

$$\mathbf{w}(k+1) = \mathbf{w}(k) - \alpha \left[\frac{\partial J(\mathbf{w}, \mathbf{y})}{\partial \mathbf{w}} \right]_{\mathbf{w}=\mathbf{w}(k)}$$

where $\alpha > 0$, $J(\mathbf{w}, \mathbf{y})$ is a criterion function, and the partial derivative is evaluated at $\mathbf{w} = \mathbf{w}(k)$. Show that the perceptron algorithm in the prob-

lem statement can be obtained from this general gradient descent procedure by using the criterion function

$$J(\mathbf{w}, \mathbf{y}) = \frac{1}{2}(|\mathbf{w}^T \mathbf{y}| - \mathbf{w}^T \mathbf{y})$$

(*Hint:* The partial derivative of $\mathbf{w}^T \mathbf{y}$ with respect to \mathbf{w} is \mathbf{y} .)

12.15* Prove that the perceptron training algorithm given in Eqs. (12-44) through (12-46) converges in a finite number of steps if the training pattern sets are linearly separable. [*Hint:* Multiply the patterns of class c_2 by -1 and consider a non-negative threshold, T_0 so that the perceptron training algorithm (with $\alpha = 1$) is expressed in the form $\mathbf{w}(k+1) = \mathbf{w}(k)$, if $\mathbf{w}^T(k)\mathbf{y}(k) > T_0$, and $\mathbf{w}(k+1) = \mathbf{w}(k) + \alpha\mathbf{y}(k)$ otherwise. You may need to use the Cauchy-Schwartz inequality: $\|\mathbf{a}\|^2 \|\mathbf{b}\|^2 \geq (\mathbf{a}^T \mathbf{b})^2$.]

12.16 Derive equations of the derivatives of the following activation functions:

- (a) The sigmoid activation function in Fig. 12.30(a).
 (b) The hyperbolic tangent activation function in Fig. 12.30(b).
 (c)* The ReLU activation function in Fig. 12.30(c).

12.17* Specify the structure, weights, and bias(es) of the smallest neural network capable of performing exactly the same function as a minimum distance classifier for two pattern classes in n -dimensional space. You may assume that the classes are tightly grouped and are linearly separable.

12.18 What is the decision boundary implemented by a neural network with n inputs, a single output neuron, and no hidden layers? Explain.

12.19 Specify the structure, weights, and bias of a neural network capable of performing exactly the same function as a Bayes classifier for two pattern classes in n -dimensional space. The classes are Gaussian with different means but equal covariance matrices.

12.20 Answer the following:

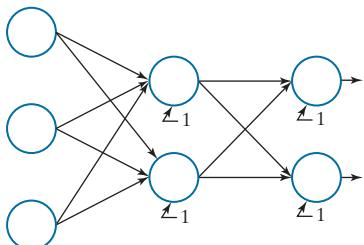
- (a)* Under what conditions are the neural networks in Problems 12.17 and 12.19 identical?
 (b) Suppose you specify a neural net architecture identical to the one in Problem 12.17. Would training by backpropagation yield the same

weights and bias as that network if trained with a sufficiently large number of samples? Explain.

- 12.21** Two pattern classes in two dimensions are distributed in such a way that the patterns of class c_1 lie randomly along a circle of radius r_1 . Similarly, the patterns of class c_2 lie randomly along a circle of radius r_2 , where $r_2 = 2r_1$. Specify the structure of a neural network with the minimum number of layers and nodes needed to classify properly the patterns of these two classes.

- 12.22*** If two classes are linearly separable, we can train a perceptron starting with weights and a bias that are all zero, and we would still get a solution. Can you do the same when training a neural network by backpropagation? Explain.

- 12.23** Label the outputs, weights, and biases for every node in the following neural network using the general notation introduced in Fig. 12.31.



- 12.24** Answer the following:

- (a) The last element of the input vector in Fig. 12.32 is 1. Is this vector augmented? Explain.
- (b) Repeat the calculations in Fig. 12.32, but using weight matrices that are 100 times the values of those used in the figure.
- (c)* What can you conclude in general from your results in (b)?

- 12.25** Answer the following:

- (a)* The chain rule in Eq. (12-70) shows three terms. However, you are probably more familiar with chain rule expressions that have two terms. Show that if you start with the expression

$$\delta_j(\ell) = \frac{\partial E}{\partial z_j(\ell)} = \sum_i \frac{\partial E}{\partial z_i(\ell+1)} \frac{\partial z_i(\ell+1)}{\partial z_j(\ell)}$$

you can arrive at the result in Eq. (12-70).

- (b) Show how the middle term in the third line of Eq. (12-70) follows from the middle term in the second.

- 12.26** Show the validity of Eq. (12-72). (*Hint:* Use the chain rule.)

- 12.27*** Show that the dimensions of matrix $\mathbf{D}(\ell)$ in Eq. (12-79) are $n_\ell \times n_p$. (*Hint:* Some of the parameters in that equation are computed in forward propagation, so you already know their dimensions.)

- 12.28** With reference to the discussion following Eq. (12-82), explain why the error for one pattern is obtained by squaring the elements of one column of matrix $(\mathbf{A}(L) - \mathbf{R})$, adding them, and dividing the result by 2.

- 12.29*** The matrix formulation in Table 12.3 contains all patterns as columns of a single matrix \mathbf{X} . This is ideal in terms of speed and economy of implementation. It is also well suited when training is done using mini-batches. However, there are applications in which the large number of training vectors is too large to hold in memory, and it becomes more practical to loop through each pattern using the vector formulation. Compose a table similar to Table 12.3, but using individual patterns, \mathbf{x} , instead of matrix \mathbf{X} .

- 12.30** Consider a CNN whose inputs are RGB color images of size 512×512 pixels. The network has two convolutional layers. Using this information, answer the following:

- (a)* You are told that the spatial dimensions of the feature maps in the first layer are 504×504 , and that there are 12 feature maps in the first layer. Assuming that no padding is used, and that the kernels used are square, and of an odd size, what are the spatial dimensions of these kernels?

- (b) If subsampling is done using neighborhoods of size 2×2 , what are the spatial dimensions of the pooled feature maps in the first layer?

- (c) What is the depth (number) of the pooled feature maps in the first layer?

- (d) The spatial dimensions of the convolution kernels in the second layer are 3×3 . Assuming no padding, what are the sizes of the feature maps in the second layer?

- (e) You are told that the number of feature maps

in the second layer is 6, and that the size of the pooling neighborhoods is again 2×2 . What are the dimensions of the vectors that result from vectorizing the last layer of the CNN? Assume that vectorization is done using linear indexing.

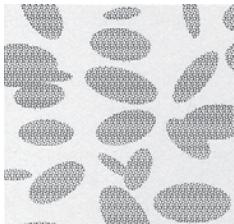
- 12.31** Suppose the input images to a CNN are padded to compensate for the size reduction caused by convolution and subsampling (pooling). Let P denote the thickness of the padding border, let V denote the width of the (square) input images, let S denote the stride, and let F denote the width of the (square) receptive field.
- Show that the number, N , of neurons in each row in the resulting feature map is

$$N = \frac{V + 2P - F}{S} + 1$$

- How would you interpret a result using this equation that is not an integer?

12.32* Show the validity of Eq. (12-106).

- 12.33** An experiment produces binary images of blobs that are nearly elliptical in shape, as the following example image shows. The blobs are of three sizes, with the average values of the principal axes of the ellipses being $(1.3, 0.7)$, $(1.0, 0.5)$, and $(0.75, 0.25)$. The dimensions of these axes vary $\pm 10\%$ about their average values.



Develop an image processing system capable of rejecting incomplete or overlapping ellipses, then classifying the remaining single ellipses into one of the three given size classes. Show your solution in block diagram form, giving specific details regarding the operation of each block. Solve the classification problem using a minimum distance classifier, indicating clearly how you would go about obtaining training samples, and how you would use these samples to train the classifier.

- 12.34** A factory mass-produces small American flags for sporting events. The quality assurance team has observed that, during periods of peak production, some printing machines have a tendency to drop (randomly) between one and three stars and one or two entire stripes. Aside from these errors, the flags are perfect in every other way. Although the flags containing errors represent a small percentage of total production, the plant manager decides to solve the problem. After much investigation, she concludes that automatic inspection using image processing techniques is the most economical approach. The basic specifications are as follows: The flags are approximately 7.5 cm by 12.5 cm in size. They move lengthwise down the production line (individually, but with a $\pm 15\%$ variation in orientation) at approximately 50 cm/s, with a separation between flags of approximately 5 cm. In all cases, “approximately” means $\pm 5\%$. The plant manager employs you to design an image processing system for each production line. You are told that cost and simplicity are important parameters in determining the viability of your approach. Design a complete system based on the model of Fig. 1.23. Document your solution (including assumptions and specifications) in a brief (but clear) written report addressed to the plant manager. You can use any of the methods discussed in the book.

This page intentionally left blank

Bibliography

- Abidi, M. A. and Gonzalez, R. C. (eds.) [1992]. *Data Fusion in Robotics and Machine Intelligence*, Academic Press, New York.
- Abramson, N. [1963]. *Information Theory and Coding*, McGraw-Hill, New York.
- Achanta, R., et al. [2012]. “SLIC Superpixels Compared to State-of-the-Art Superpixel Methods,” *IEEE Trans. Pattern Anal. Mach. Intell.* vol. 34, no. 11, pp. 2274–2281.
- Ahmed, N., Natarajan, T., and Rao, K. R. [1974]. “Discrete Cosine Transforms,” *IEEE Trans. Comp.*, vol. C-23, pp. 90–93.
- Andrews, H. C. [1970]. *Computer Techniques in Image Processing*, Academic Press, New York.
- Andrews, H. C. and Hunt, B. R. [1977]. *Digital Image Restoration*, Prentice Hall, Englewood Cliffs, NJ.
- Antonini, M., Barlaud, M., Mathieu, P., and Daubechies, I. [1992]. “Image Coding Using Wavelet Transform,” *IEEE Trans. Image Process.*, vol. 1, no. 2, pp. 205–220.
- Ascher, R.N. and Nagy, G. [1974]. “A Means for Achieving a High Degree of Compaction on Scan-Digitized Printed Text,” *IEEE Transactions on Comp.*, C-23, pp. 1174–1179.
- Ballard, D. H. [1981]. “Generalizing the Hough Transform to Detect Arbitrary Shapes,” *Pattern Recognition*, vol. 13, no. 2, pp. 111–122.
- Ballard, D. H. and Brown, C. M. [1982]. *Computer Vision*, Prentice Hall, Englewood Cliffs, NJ.
- Basart, J. P., Chacklackal, M. S., and Gonzalez, R. C. [1992]. “Introduction to Gray-Scale Morphology,” in *Advances in Image Analysis*, Y. Mahdavieh and R. C. Gonzalez (eds.), SPIE Press, Bellingham, WA, pp. 306–354.
- Basart, J. P. and Gonzalez, R. C. [1992]. “Binary Morphology,” in *Advances in Image Analysis*, Y. Mahdavieh and R. C. Gonzalez (eds.), SPIE Press, Bellingham, WA, pp. 277–305.
- Bell, E.T. [1965]. *Men of Mathematics*, Simon & Schuster, New York.
- Berger, T. [1971]. *Rate Distortion Theory*, Prentice Hall, Englewood Cliffs, NJ.
- Beucher, S. and Meyer, F. [1992]. “The Morphological Approach of Segmentation: The Watershed Transformation,” in *Mathematical Morphology in Image Processing*, E. Dougherty (ed.), Marcel Dekker, New York.
- Beyerer, J., Puente Leon, F. and Frese, C. [2016]. *Machine Vision—Automated Visual Inspection: Theory, Practice, and Applications*, Springer-Verlag, Berlin, GermanyNew York.
- Blahut, R. E. [1987]. *Principles and Practice of Information Theory*, Addison-Wesley, Reading, MA.
- Bleau, A. and Leon, L. J. [2000]. “Watershed-Based Segmentation and Region Merging,” *Computer Vision and Image Understanding*, vol. 77, no. 3, pp. 317–370.
- Blum, H. [1967]. “A Transformation for Extracting New Descriptors of Shape,” in *Models for the Perception of Speech and Visual Form*, Wathen-Dunn,W. (ed.), MIT Press, Cambridge, MA.

- Born, M. and Wolf, E. [1999]. *Principles of Optics: Electromagnetic Theory of Propagation, Interference and Diffraction of Light*, 7th ed., Cambridge University Press, Cambridge, UK.
- Bracewell, R. N. [1995]. *Two-Dimensional Imaging*, Prentice Hall, Upper Saddle River, NJ.
- Bracewell, R. N. [2003]. *Fourier Analysis and Imaging*, Springer, New York.
- Bribiesca, E. [1992]. "A Geometric Structure for Two-Dimensional Shapes and Three Dimensional Surfaces," *Pattern Recognition*, vol. 25, pp. 483–496.
- Bribiesca, E. [2013]. "A Measure of Tortuosity Based on Chain Coding," *Pattern Recognition*, vol. 46, pp. 716–724.
- Bribiesca, E. and Guzman, A. [1980]. "How to Describe Pure Form and How to Measure Differences in Shape Using Shape Numbers," *Pattern Recognition*, vol. 12, no. 2, pp. 101–112.
- Brigham, E. O. [1988]. *The Fast Fourier Transform and its Applications*, Prentice Hall, Upper Saddle River, NJ.
- Bronson, R. and Costa, G. B. [2009]. *Matrix Methods: Applied Linear Algebra*, 3rd ed., Academic Press/Elsevier, Burlington, MA.
- Burrus, C. S., Gopinath, R. A., and Guo, H. [1998]. *Introduction to Wavelets and Wavelet Transforms*, Prentice Hall, Upper Saddle River, NJ, pp. 250–251.
- Buzug, T. M. [2008]. *Computed Tomography: From Photon Statistics to Modern Cone-Beam CT*, Springer-Verlag, Berlin, Germany.
- Cannon, T. M. [1974]. "Digital Image Deblurring by Non-Linear Homomorphic Filtering," Ph.D. thesis, University of Utah.
- Canny, J. [1986]. "A Computational Approach for Edge Detection," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 8, no. 6, pp. 679–698.
- Caselles, V., Kimmel, R., and Sapiro, G. [1997]. "Geodesic Active Contours," *Int'l J. Comp. Vision*, vol. 22, no. 1, pp. 61–79.
- Castleman, K. R. [1996]. *Digital Image Processing*, 2nd ed., Prentice Hall, Upper Saddle River, NJ.
- Chakrabarti, I., et al. [2015]. *Motion Estimation for Video Coding*, Springer Int'l Publishing, Cham, Switzerland.
- Champeney, D. C. [1987]. *A Handbook of Fourier Theorems*, Cambridge University Press, London, UK.
- Chan, T. F. and Vese, L. A. [2001]. "Active Contours Without Edges," *IEEE Trans. Image Process.*, vol. 10, no. 2, pp. 266–277.
- Cheng, Y., Hu, X., Wang, J., Wang, Y., and Tamura, S. [2015]. "Accurate Vessel Segmentation with Constrained B-Snake," *IEEE Trans. Image Process.* vol. 24, no. 8, pp. 2440–2455.
- Choromanska, A., et al. [2015]. "The Loss Surfaces of Multilayer Networks," *Proc. 18th Int'l Conference Artificial Intell. and Statistics (AISTATS)*, vol. 38, pp. 192–204.
- Clarke, R. J. [1985]. *Transform Coding of Images*, Academic Press, New York.
- Cohen, A., Daubechies, I., and Feauveau, J.-C. [1992]. "Biorthogonal Bases of Compactly Supported Wavelets," *Commun. Pure and Appl. Math.*, vol. 45, pp. 485–560.
- Coifman, R. R. and Wickerhauser, M. V. [1992]. "Entropy-Based Algorithms for Best Basis Selection," *IEEE Tran. Information Theory*, vol. 38, no. 2, pp. 713–718.

- Coltuc, D., Bolon, P., and Chassery, J-M [2006]. "Exact Histogram Specification," *IEEE Trans. Image Process.*, vol. 15, no. 5, pp. 1143–1152.
- Cornsweet, T. N. [1970]. *Visual Perception*, Academic Press, New York.
- Cox, I., Kilian, J., Leighton, F., and Shamoon, T. [1997]. "Secure Spread Spectrum Watermarking for Multimedia," *IEEE Trans. Image Process.*, vol. 6, no. 12, pp. 1673–1687.
- Cox, I., Miller, M., and Bloom, J. [2001]. *Digital Watermarking*, Morgan Kaufmann (Elsevier), New York.
- Cybenco, G. [1989]. "Approximation by Superposition of a Sigmoidal Function," *Math. Control Signals Systems*, vol. 2, no. 4, pp. 303–314.
- D. N. Joanes and C. A. Gill. [1998]. "Comparing Measures of Sample Skewness and Kurtosis". *The Statistician*, vol 47, no. 1, pp. 183–189.
- Danielsson, P. E. and Seger, O. [1990]. "Generalized and Separable Sobel Operators," in *Machine Vision for Three-Dimensional Scenes*, Herbert Freeman (ed.), Academic Press, New York.
- Daubechies, I. [1988]. "Orthonormal Bases of Compactly Supported Wavelets," *Commun. On Pure and Appl. Math.*, vol. 41, pp. 909–996.
- Daubechies, I. [1990]. "The Wavelet Transform, Time-Frequency Localization and Signal Analysis," *IEEE Transactions on Information Theory*, vol. 36, no. 5, pp. 961–1005.
- Daubechies, I. [1992]. *Ten Lectures on Wavelets*, Society for Industrial and Applied Mathematics, Philadelphia, PA.
- Daubechies, I. [1993]. "Orthonormal Bases of Compactly Supported Wavelets II, Variations on a Theme," *SIAM J. Mathematical Analysis*, vol. 24, no. 2, pp. 499–519.
- Daubechies, I. [1996]. "Where Do We Go from Here?—A Personal Point of View," *Proc. IEEE*, vol. 84, no. 4, pp. 510–513.
- Delgado-Gonzalo, R., Uhlmann, V., and Unser, M. [2015]. "Snakes on a Plane: A Perfect Snap for Bioimage Analysis," *IEEE Signal Proc. Magazine*, vol. 32, no. 1, pp. 41–48.
- de Moura, C. A. and Kubrusky, C. S. (eds.) [2013]. *The Courant-Friedrichs-Lowy (CLF) Condition*, Springer, New York.
- Drew, M. S., Wei, J., and Li, Z.-N. [1999]. "Illumination Invariant Image Retrieval and Video Segmentation," *Pattern Recognition*, vol. 32, no. 8, pp. 1369–1388.
- Duda, R. O., Hart, P. E., and Stork, D. G. [2001]. *Pattern Classification*, John Wiley & Sons, New York.
- Eng, H.-L. and Ma, K.-K. [2001]. "Noise Adaptive Soft-Switching Median Filter," *IEEE Trans. Image Process.*, vol. 10, no. 2, pp. 242–251.
- Eng, H.-L. and Ma, K.-K. [2006]. "A Switching Median Filter With Boundary Discriminative Noise Detection for Extremely Corrupted Images," *IEEE Trans. Image Process.*, vol. 15, no. 6, pp. 1506–1516.
- Federal Bureau of Investigation [1993]. *WSQ Gray-Scale Fingerprint Image Compression Specification*, IAFIS-IC-0110v2, Washington, DC.