CHAPTER 18

FEATURE EXTRACTION AND REPRESENTATION

WHAT WILL WE LEARN?

- What is feature extraction and why is it a critical step in most computer vision and image processing solutions?
- Which types of features can be extracted from an image and how is this usually done?
- How are the extracted features usually represented for further processing?

18.1 INTRODUCTION

This chapter discusses methods and techniques for representing and describing an image and its objects or regions of interest. Most techniques presented in this chapter assume that an image has undergone segmentation.[1] The common goal of feature extraction and representation techniques is to convert the segmented objects into representations that better describe their main features and attributes. The type and complexity of the resulting representation depend on many factors, such as the type of image (e.g., binary, grayscale, or color), the level of granularity (entire image or

[1] Image segmentation was the topic of Chapter 16.

Practical Image and Video Processing Using MATLAB®. By Oge Marques.

individual regions) desired, and the context of the application that uses the results (e.g., a two-class pattern classifier that tells circular objects from noncircular ones or an image retrieval system that retrieves images judged to be similar to an example image).

Feature extraction is the process by which certain features of interest within an image are detected and represented for further processing. It is a critical step in most computer vision and image processing solutions because it marks the transition from pictorial to nonpictorial (alphanumerical, usually quantitative) data representation. The resulting representation can be subsequently used as an input to a number of pattern recognition and classification techniques,[2] which will then label, classify, or recognize the semantic contents of the image or its objects.

There are many ways an image (and its objects) can be represented for image analysis purposes. In this chapter, we present several representative techniques for feature extraction using a broad range of image properties.

18.2 FEATURE VECTORS AND VECTOR SPACES

A *feature vector* is a $n \times 1$ array that encodes the n features (or measurements) of an image or object. The array contents may be symbolic (e.g., a string containing the name of the predominant color in the image), numerical (e.g., an integer expressing the area of an object, in pixels), or both. In the remaining part of our discussion, we shall focus exclusively on numerical feature vectors.

Mathematically, a numerical feature vector $\mathbf{x}$ is given by

$$\mathbf{x} = (x_1, x_2, \ldots, x_n)^T \tag{18.1}$$

where n is the total number of features and T indicates the *transpose* operation.

The feature vector is a compact representation of an image (or object within the image), which can be associated with the notion of a *feature space*, an n-dimensional *hyperspace* that allows the visualization (for $n < 4$) and interpretation of the feature vectors' contents, their relative distances, and so on.[3]

■ EXAMPLE 18.1

Suppose that the objects in Figure 18.1a have been represented by their area and perimeter, which have been computed as follows:

Object	Area	Perimeter
Square (Sq)	1024	124
Large circle (LC)	3209	211
Small circle (SC)	797	105

[2]Pattern classification techniques will be discussed in Chapter 19.

[3]In Chapter 19, we shall discuss how feature vectors are used by pattern classifiers.

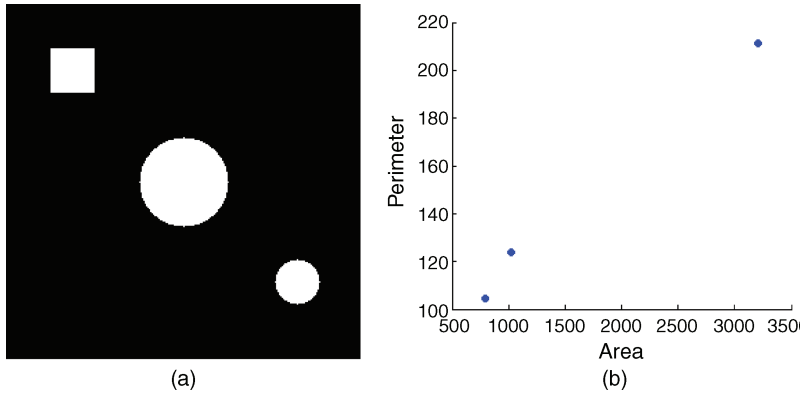

FIGURE 18.1 Test image (a) and resulting 2D feature vectors (b).

The resulting feature vectors will be as follows:
$\mathbf{Sq} = (1024, 124)^T$
$\mathbf{LC} = (3209, 211)^T$
$\mathbf{SC} = (797, 105)^T$

Figure 18.1b shows the three feature vectors plotted in a 2D graph whose axes are the selected features, namely, *area* and *perimeter*.

In MATLAB

In MATLAB, feature vectors are usually represented using cell arrays and structures.[4]

18.2.1 Invariance and Robustness

A common requirement for feature extraction and representation techniques is that the features used to represent an image be invariant to rotation, scaling, and translation, collectively known as *RST*. RST invariance ensures that a machine vision system will still be able to recognize objects even when they appear at different size, position within the image, and angle (relative to a horizontal reference). Clearly, this requirement is application dependent. For example, if the goal of a machine vision system is to ensure that an integrated circuit (IC) is present at the correct position and orientation on a printed circuit board, rotation invariance is no longer a requirement; on the contrary, being able to tell that the IC is upside down is an integral part of the system's functionality.

As you may recall from our discussion in Chapter 1, the goal of many machine vision systems is to emulate—to the highest possible degree—the human visual system's ability to recognize objects and scenes under a variety of circumstances. In order

[4]Refer to Chapter 3—particularly Tutorial 3.2—if you need to refresh these concepts.

to achieve such ability, the feature extraction and representation stage of a machine vision system should ideally provide a representation that is not only RST invariant but also *robust* to other aspects, such as poor spatial resolution, nonuniform lighting, geometric distortions (caused by different viewing angles), and noise. This is a big challenge for machine vision systems designers, which may require careful feature selection as well as pre- and postprocessing techniques to be fully accomplished.

18.3 BINARY OBJECT FEATURES

In this section, we present several useful features for binary objects. A binary object, in this case, is a connected region within a binary image $f(x, y)$, which will be denoted as O_i, $i > 0$.

Mathematically, we can define a function $O_i(x, y)$ as follows:

$$O_i(x, y) = \begin{cases} 1 & \text{if } f(x, y) \in O_i \\ 0 & \text{otherwise} \end{cases} \tag{18.2}$$

In MATLAB

The IPT function `bwlabel` – introduced in Chapter 2 and also discussed in Chapter 13 – implements the operation $i \times O_i(x, y)$ by labeling regions of connected pixels in a binary image: pixels labeled 0 correspond to the background; pixels labeled 1 and higher correspond to the connected components in the image.

18.3.1 Area

The area of the ith object O_i, measured in pixels, is given by

$$A_i = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} O_i(x, y) \tag{18.3}$$

18.3.2 Centroid

The coordinates of the centroid (also known as *center of area*) of object O_i, denoted $(\bar{x}_i, \bar{y}_i)$, are given by

$$\bar{x}_i = \frac{1}{A_i} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} x O_i(x, y) \tag{18.4}$$

and

$$\bar{y}_i = \frac{1}{A_i} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} y O_i(x, y) \tag{18.5}$$

where A_i is the area of object O_i, as defined in equation (18.3).

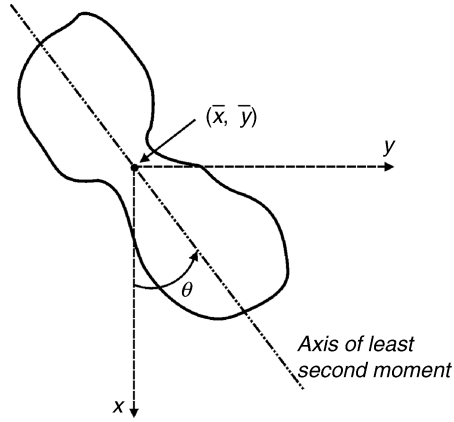

FIGURE 18.2 Axis of least second moment.

18.3.3 Axis of Least Second Moment

The axis of least second moment is used to provide information about the object's orientation relative to the coordinate plan of the image. It can be described as the axis of least inertia, that is, the line about which it takes the least amount of energy to rotate the object.

By convention, the angle θ represents the angle between the vertical axis and the axis of least second moment, measured counterclockwise. Figure 18.2 illustrates this concept. Note that the origin of the coordinate system has been moved to the center of area of the object.

Mathematically, θ can be calculated using equation (18.6):

$$\tan(2\theta_i) = 2 \times \frac{\sum_{x=0}^{M-1} \sum_{y=0}^{N-1} x O_i(x, y)}{\sum_{x=0}^{M-1} \sum_{y=0}^{N-1} x^2 O_i(x, y) - \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} y^2 O_i(x, y)} \tag{18.6}$$

18.3.4 Projections

The horizontal and vertical projections of a binary object—$h_i(x)$ and $v_i(y)$, respectively—are obtained by equations (18.7) and (18.8):

$$h_i(x) = \sum_{x=0}^{M-1} O_i(x, y) \tag{18.7}$$

and

$$v_i(y) = \sum_{y=0}^{N-1} O_i(x, y) \tag{18.8}$$

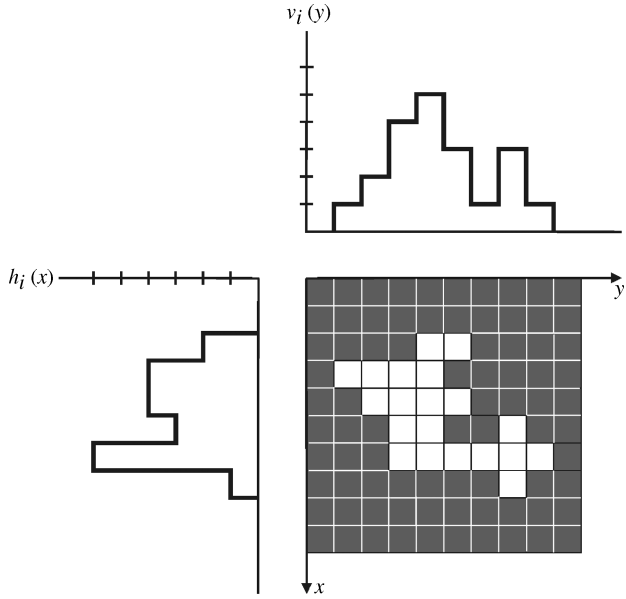

FIGURE 18.3 Horizontal and vertical projections.

Projections are very useful and compact shape descriptors. For example, the height and width of an object without holes can be computed as the maximum value of the object's vertical and horizontal projections, respectively, as illustrated in Figure 18.3.

The equations for the coordinates of the center of area of the object (equations (18.4) and (18.5)) can be rewritten as a function of the horizontal and vertical projections as follows:

$$\bar{x}_i = \frac{1}{A_i} \sum_{x=0}^{M-1} x h_i(x) \tag{18.9}$$

and

$$\bar{y}_i = \frac{1}{A_i} \sum_{y=0}^{N-1} y v_i(y) \tag{18.10}$$

18.3.5 Euler Number

The Euler number of an object or image (E) is defined as the number of connected components (C) minus the number of holes (H):

$$E = C - H \tag{18.11}$$

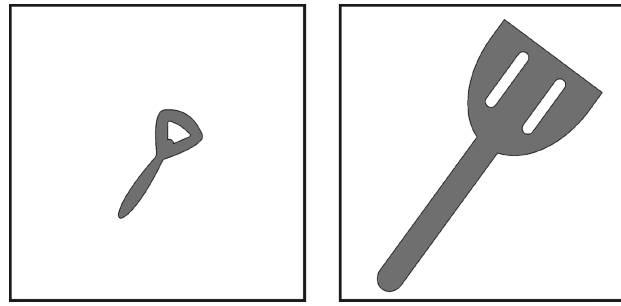

FIGURE 18.4 Examples of two regions with Euler numbers equal to 0 and −1, respectively.

Alternatively, the Euler number can be expressed as the difference between the number of *convexities* and the number of *concavities* in the image or region. The Euler number is considered a *topological descriptor*, because it is unaffected by deformations (such as *rubber sheet* distortions), provided that they do not fill holes or break connected components apart. Figure 18.4 shows examples of two different objects and their Euler numbers.

18.3.6 Perimeter

The perimeter of a binary object O_i can be calculated by counting the number of object pixels (whose value is 1) that have one or more background pixels (whose value is 0) as their neighbors. An alternative method consists in first extracting the edge (contour) of the object and then counting the number of pixels in the resulting border. Due to some inevitable imperfections in the digitization process (e.g., jagged curve outlines and serrated edges), the value of perimeter computed using either method is not 100% accurate; it has been suggested [Umb05] that those values should be multiplied by $\pi/4$ for better accuracy.

In MATLAB

The IPT function `bwperim`—introduced in Chapter 13—computes the perimeter of objects in a binary image using either 4- (which is the default) or 8-connectivity criteria.

18.3.7 Thinness Ratio

The thinness ratio T_i of a binary object O_i is a figure of merit that relates the object's area and its perimeter by equation (8.12):

$$T_i = \frac{4\pi A_i}{P_i^2} \tag{18.12}$$

where A_i is the area and P_i is the perimeter.

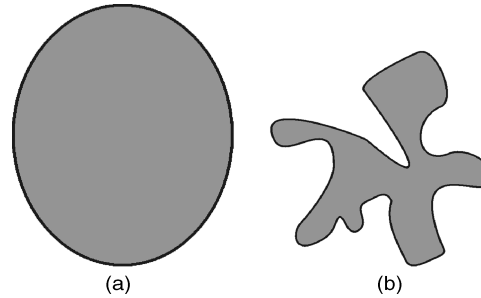

FIGURE 18.5 Examples of a compact (a) and a noncompact (b) regions.

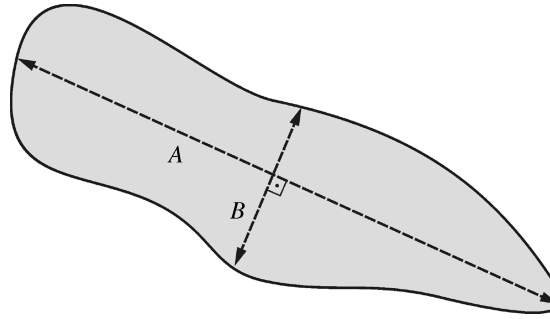

FIGURE 18.6 Eccentricity (A/B) of a region.

The thinness ratio is often used as a measure of roundness. Since the maximum value for T_i is 1 (which corresponds to a perfect circle), for a generic object the higher its thinness ratio, the more round it is. This figure of merit can also be used as a measure of regularity and its inverse, $1/T_i$ is sometimes called *irregularity* or *compactness* ratio. Figure 18.5 shows an example of a compact and a noncompact region.

18.3.8 Eccentricity

The eccentricity of an object is defined as the ratio of the major and minor axes of the object (Figure 18.6).

18.3.9 Aspect Ratio

The aspect ratio (AR) is a measure of the relationship between the dimensions of the bounding box of an object. It is given by

$$\mathrm{AR} = \frac{x_{\max} - x_{\min} + 1}{y_{\max} - y_{\min} + 1} \tag{18.13}$$

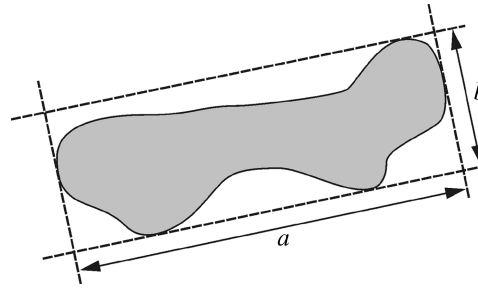

FIGURE 18.7 Elongatedness (a/b) of a region.

where $(x_{\min}, y_{\min})$ and $(x_{\max}, y_{\max})$ are the coordinates of the top left and bottom right corners of the bounding box surrounding an object, respectively.

It should be noted that the aspect ratio is not rotation invariant and cannot be used to compare rotated objects against one another unless it is normalized somehow, for example, by computing the AR after aligning the axis of least second moment to the horizontal direction. Such normalized representations of the aspect ratio are also referred to as the *elongatedness* of the object (Figure 18.7).

18.3.10 Moments

The 2D *moment of order (p + q)* of a digital image $f(x, y)$ is defined as

$$m_{pq} = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} x^p y^q f(x, y) \tag{18.14}$$

where M and N are the image height and width, respectively, and p and q are positive nonzero integers.

Central moments are the translation-invariant equivalent of moments. They are defined as

$$\mu_{pq} = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} (x - \bar{x})^p (y - \bar{y})^q f(x, y) \tag{18.15}$$

where

$$\bar{x} = \frac{m_{10}}{m_{00}} \quad \text{and} \quad \bar{y} = \frac{m_{01}}{m_{00}} \tag{18.16}$$

The *normalized central moments* are defined as

$$\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^{\gamma}} \tag{18.17}$$

TABLE 18.1 RST-Invariant Moments

$$\phi_1 = \eta_{20} + \eta_{02}$$
$$\phi_2 = (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2$$
$$\phi_3 = (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2$$
$$\phi_4 = (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2$$
$$\phi_5 = (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})\left[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2\right] + (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})\left[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2\right]$$
$$\phi_6 = (\eta_{30} - \eta_{02})\left[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2\right] + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03})$$
$$\phi_7 = (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})\left[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2\right] - (\eta_{30} - 3\eta_{12})(\eta_{21} + \eta_{03})\left[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2\right]$$

where

$$\gamma = \frac{p+q}{2} + 1 \tag{18.18}$$

for $(p + q) > 1$.

A set of seven *RST-invariant moments*, ϕ_1–ϕ_7, originally proposed by Hu in 1962 [Hu62], can be derived from second- and third-order normalized central moments. They are listed in Table 18.1.

In MATLAB

The IPT function `regionprops` measures a set of properties for each labeled region `L`. One or more properties from Table 18.2 can be specified as parameters. You will explore `regionprops` in Tutorial 18.1.

18.4 BOUNDARY DESCRIPTORS

The descriptors described in Section 18.3 are collectively referred to as *region based*. In this section, we will look at *contour-based* representation and description techniques. These techniques assume that the contour (or *boundary*) of an object can be represented in a convenient coordinate system (Cartesian—the most common, polar, or tangential) and rely exclusively on boundary pixels to describe the region or object. Object boundaries can be represented by different techniques, ranging from simple polygonal approximation methods to more elaborated techniques involving piecewise polynomial interpolations such as B-spline curves.

The techniques described in this section assume that the pixels belonging to the boundary of the object (or region) can be traced, starting from any background pixel, using an algorithm known as *bug tracing* that works as follows: as soon as the conceptual bug crosses into a boundary pixel, it makes a left turn and moves to the next pixel; if that pixel is a boundary pixel, the bug makes another left turn, otherwise it turns right; the process is repeated until the bug is back to the starting point. As the

TABLE 18.2 Properties of Labeled Regions

Property	Description
`'Area'`	The actual number of pixels in the region
`'BoundingBox'`	The smallest rectangle containing the region, specified by the coordinates of the top left corner and the length along each dimension
`'Centroid'`	A vector that specifies the center of mass of the region
`'ConvexHull'`	The smallest convex polygon that can contain the region
`'ConvexImage'`	Binary image that specifies the convex hull, that is, the image for which with all pixels within the hull are set to *true*
`'ConvexArea'`	The number of pixels in `'ConvexImage'`
`'Eccentricity'`	The eccentricity of the ellipse that has the same second moments as the region. The eccentricity is a value in the [0,1] range (0 for a circle, 1 for a line segment). It is defined as the ratio of the distance between the foci of the ellipse and its major axis length
`'EquivDiameter'`	The diameter of a circle with the same area as the region, calculated as $\sqrt{(4 * \text{Area}/\pi)}$
`'EulerNumber'`	The difference between the number of objects in the region and the number of holes in these objects
`'Extent'`	The proportion of the pixels in the bounding box that are also in the region, that is, the ratio between the area of region and the area of the bounding box
`'Extrema'`	An 8×2 matrix containing the coordinates of the extrema points in the region
`'FilledArea'`	The number of pixels in `FilledImage`
`'FilledImage'`	A binary image of the same size as the bounding box of the region, where all *true* pixels correspond to the region and all holes have been filled in
`'Image'`	A binary image of the same size as the bounding box of the region, where all *true* pixels correspond to the region, and all other pixels are set to *false*
`'MajorAxisLength'`	The length (in pixels) of the major axis of the ellipse that has the same normalized second central moments as the region
`'MaxIntensity'`	The value of the pixel with the greatest intensity in the region
`'MeanIntensity'`	The mean of all the intensity values in the region
`'MinIntensity'`	The value of the pixel with the lowest intensity in the region
`'MinorAxisLength'`	The length (in pixels) of the minor axis of the ellipse that has the same normalized second central moments as the region

(*Continued*)

TABLE 18.2 (*Continued*)

Property	Description
`'Orientation'`	The angle (in degrees ranging from -90° to 90°) between the x-axis and the major axis of the ellipse that has the same second-moments as the region
`'Perimeter'`	Vector containing the perimeter of each region in the image
`'PixelIdxList'`	Vector containing the linear indices of the pixels in the region
`'PixelList'`	Matrix specifying the locations of pixels in the region
`'PixelValues'`	Vector containing the values of all pixels in a region
`'Solidity'`	The proportion of the pixels in the convex hull that are also in the region. Computed as `Area/ConvexArea`
`'SubarrayIdx'`	Vector of linear indices of nonzero elements returned in `Image`
`'WeightedCentroid'`	Vector of coordinates specifying the center of the region based on location and intensity value

conceptual bug follows the contour, it builds a list of coordinates of the boundary pixels being visited.

In MATLAB

The IPT function `bwtraceboundary` traces objects in a binary image. The coordinates of the starting point and the initial search direction (N, NE, E, SE, S, SW, W, or NW) must be passed as parameters. Optionally, the type of connectivity (4 or 8) and the direction of the tracing (clockwise or counterclockwise) can also be passed as arguments. The `bwtraceboundary` function returns an array containing the row and column coordinates of the boundary pixels.

■ EXAMPLE 18.2

Figure 18.8a shows an example of using `bwtraceboundary` to trace the first 50 pixels of the boundary of the leftmost object in the input image, starting from the top left corner and heading south, searching in counterclockwise direction. Figure 18.8b repeats the process, this time tracing the first 40 pixels of the boundary of *all* objects in the image, heading east, and searching in clockwise direction. The boundaries are shown in red.

In MATLAB

The IPT function `bwboundaries` traces region boundaries in binary image. It can trace the exterior boundaries of objects, as well as boundaries of holes inside these

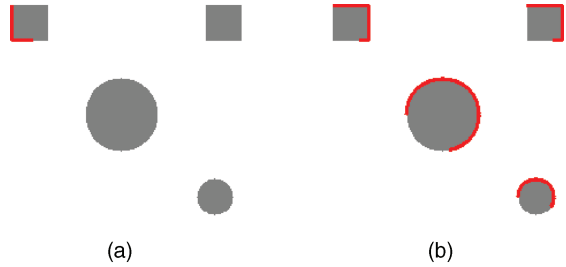

FIGURE 18.8 Tracing boundaries of objects.

objects. It can also trace the boundaries of children of parent objects in an image. The `bwboundaries` function returns a cell array (`B`) in which each cell contains the row and column coordinates of a boundary pixel. Optionally, it also returns a label matrix `L` where each object and hole is assigned a unique label, the total number of objects found (`N`), and an adjacency matrix used to represent parent–child–hole dependencies.

■ EXAMPLE 18.3

Figure 18.9a shows an example of using `bwboundaries` to trace the objects (represented in red) and holes (represented in green) in the input image using the syntax `[B,L,N] = bwboundaries(BW);`. Figure 18.9b repeats the process, this time using 4-connectivity and searching only for object boundaries, that is, no holes, in the input image using the syntax `[B,L,N] = bwboundaries(BW, 4, 'noholes');`.

18.4.1 Chain Code, Freeman Code, and Shape Number

Chain codes are alternative methods for tracing and describing a contour. A chain code is a boundary representation technique by which a contour is represented as a sequence of straight line segments of specified length (usually 1) and direction. The simplest chain code mechanism, also known as *crack code*, consists of assigning a

FIGURE 18.9 Tracing boundaries of objects and holes.

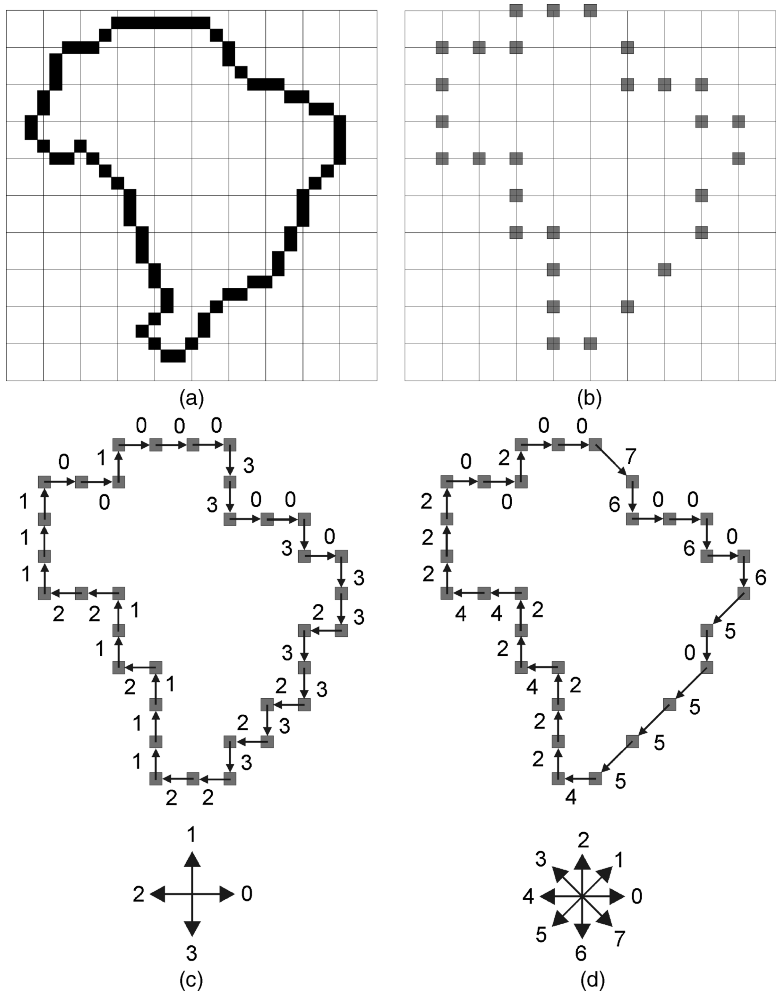

FIGURE 18.10 Chain code and Freeman code for a contour: (a) original contour; (b) subsampled version of the contour; (c) chain code representation; (d) Freeman code representation.

number to the direction followed by a bug tracking algorithm as follows: right (0), down (1), left (2), and up (3). Assuming that the total number of boundary points is p (the perimeter of the contour), the array C (of size p), where $C(p) = 0, 1, 2, 3$, contains the chain code of the boundary. A modified version of the basic chain code, known as the *Freeman code*, uses eight directions instead of four. Figure 18.10 shows an example of a contour, its chain code, and its Freeman code.

Once the chain code for a boundary has been computed, it is possible to convert the resulting array into a rotation-invariant equivalent, known as the *first difference*,

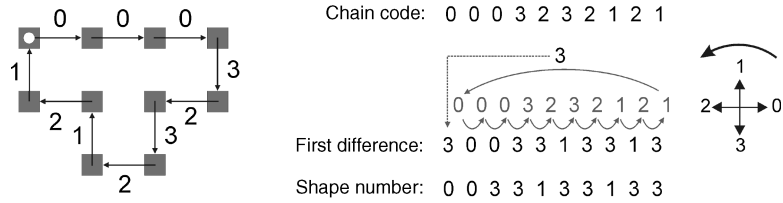

FIGURE 18.11 Chain code, first differences, and shape number.

which is obtained by encoding the number of direction changes, expressed in multiples of 90° (according to a predefined convention, for example, counterclockwise), between two consecutive elements of the Freeman code. The first difference of smallest magnitude—obtained by treating the resulting array as a circular array and rotating it cyclically until the resulting numerical pattern results in the smallest possible number—is known as the *shape number* of the contour. The shape number is rotation invariant and insensitive to the starting point used to compute the original sequence. Figure 18.11 shows an example of a contour, its chain code, first differences, and shape number.

In MATLAB

The IPT[5] does not have built-in functions for computing the chain code, Freeman code, and shape number of a contour. Refer to Chapter 12 of [McA04] or Chapter 11 of [GWE04] for alternative implementations.

18.4.2 Signatures

A signature is a 1D representation of a boundary, usually obtained by representing the boundary in a polar coordinate system and computing the distance r between each pixel along the boundary and the centroid of the region, and the angle θ subtended between a straight line connecting the boundary pixel to the centroid and a horizontal reference (Figure 18.12, top). The resulting plot of all computed values for $0 \leq \theta \leq 2\pi$ (Figure 18.12, bottom) provides a concise representation of the boundary that is translation invariant can be made rotation invariant (if the same starting point is always selected), but is *not* scaling invariant.

Figure 18.13 illustrates the effects of noise on the signature of a contour.

In MATLAB

The IPT does not have a built-in function for computing the signature of a boundary. Refer to Chapter 11 of [GWE04] for an alternative implementation.

[5] At the time of this writing, the latest version of the IPT is Version 7.2 (R2011a).

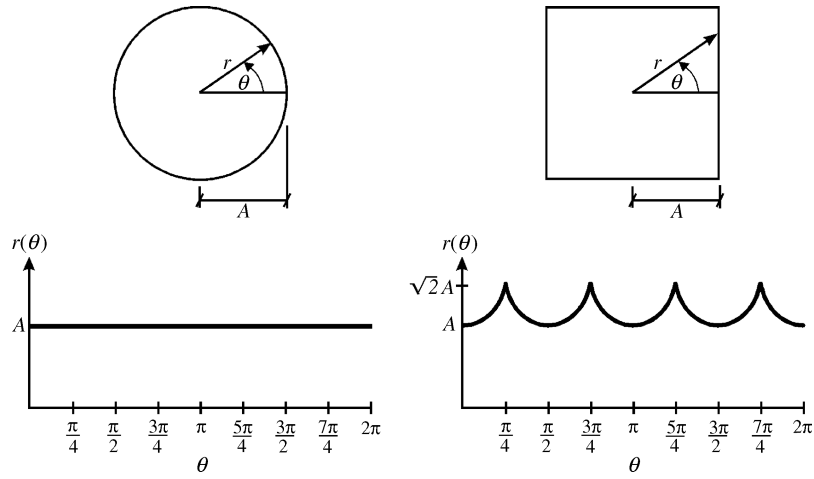

FIGURE 18.12 Distance × angle signatures for two different objects. Redrawn from [GW08].

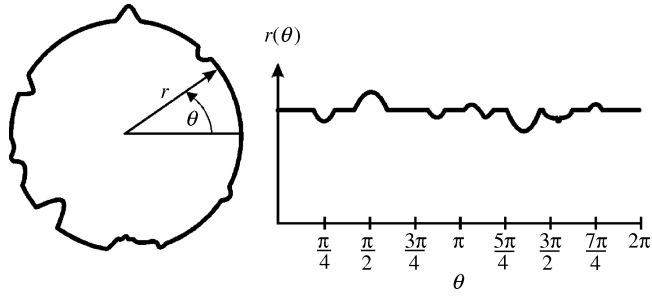

FIGURE 18.13 Effect of noise on signatures for two different objects. Redrawn from [GW08].

18.4.3 Fourier Descriptors

The idea behind Fourier descriptors is to traverse the pixels belonging to a boundary, starting from an arbitrary point, and record their coordinates. Each value in the resulting list of coordinate pairs $(x_0, y_0), (x_1, y_1), \ldots, (x_{K-1}, y_{K-1})$ is then interpreted as a complex number $x_k + jy_k$, for $k = 0, 1, \ldots, K-1$. The discrete Fourier transform (DFT)[6] of this list of complex numbers is the *Fourier descriptor* of the boundary. The inverse DFT restores the original boundary. Figure 18.14 shows a K-point digital boundary in the xy plane and the first two coordinate pairs, (x_0, y_0) and (x_1, y_1).

[6]The discrete Fourier transform was introduced in Chapter 11.

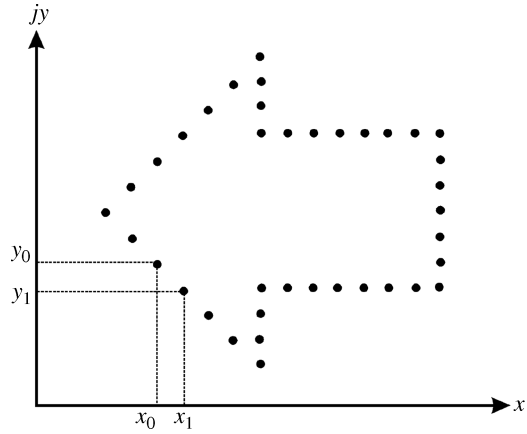

FIGURE 18.14 Fourier descriptor of a boundary.

In MATLAB

The IPT does not have a built-in function for computing the Fourier descriptor of a boundary, but it is not too difficult to create your own function. The basic procedure is to compute the boundaries of the region, convert the resulting coordinates into complex numbers (e.g., using the `complex` function), and apply the `fft` function to the resulting 1D array of complex numbers. Refer to Chapter 12 of [McA04] or Chapter 11 of [GWE04] for possible implementations.

One of the chief advantages of using Fourier descriptors is their ability to represent the essence of the corresponding boundary using very few coefficients. This property is directly related to the ability of the low-order coefficients of the DFT to preserve the main aspects of the boundary, while the high-order coefficients encode the fine details.

■ EXAMPLE 18.4

Figure 18.15 shows an example of a boundary containing 1467 points (part (a)) and the results of reconstructing it (applying the DFT followed by IDFT) using a variable number of terms. Part (b) shows the results of reconstructing with the same number of points for the sake of verification. Parts (c)–(f) show the results for reconstruction with progressively fewer points: 734, 366, 36, and 15, respectively. The chosen values for parts (c)–(f) correspond to approximately 50%, 25%, 2.5%, and 1% of the total number of points, respectively. A close inspection of the results shows that the image reconstructed using 25% of the points is virtually identical to the one obtained with 50% or 100% of the points. The boundary reconstructed using 2.5% of the points still preserves much of its overall shape, and the results using only 1% of the total number of points can be deemed unacceptable.

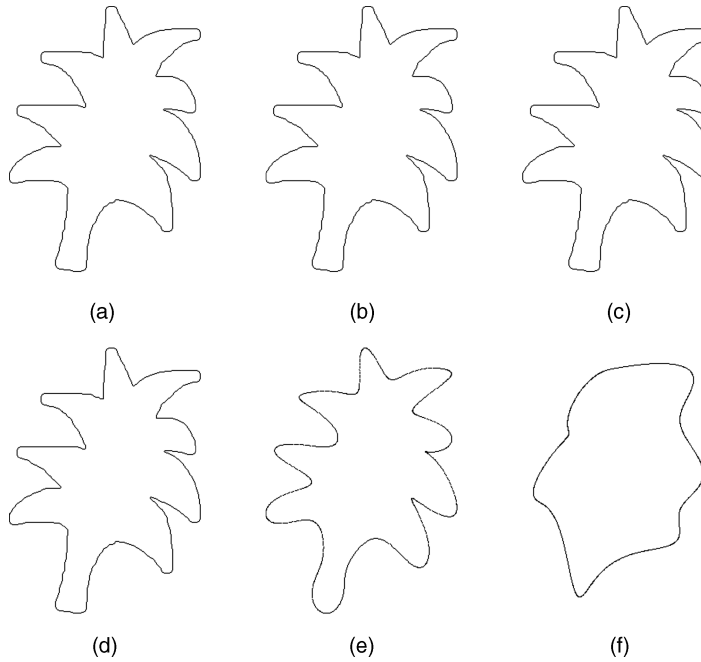

FIGURE 18.15 Example of boundary reconstruction using Fourier descriptors: (a) original image; (b–f) reconstructed image using 100%, 50%, 25%, 2.5%, and 1% of the total number of points, respectively.

18.5 HISTOGRAM-BASED (STATISTICAL) FEATURES

Histograms provide a concise and useful representation of the intensity levels in a grayscale image. In Chapter 9, we learned how to compute and display an image's histogram and we used histogram-based techniques for image enhancement. In Chapter 16, we extended that discussion to color images. In this section, we are interested in using histograms (or numerical descriptors that can be derived from them) as features[7] that describe an image (or its objects).

The simplest histogram-based descriptor is the *mean* gray value of an image, representing its average intensity m and given by

$$m = \sum_{j=0}^{L-1} r_j p(r_j) \tag{18.19}$$

where r_j is the jth gray level (out of a total of L possible values), whose probability of occurrence is $p(r_j)$.

[7]Histogram-based features are also referred to as *amplitude features* in the literature.

The mean gray value can also be computed directly from the pixel values from the original image $f(x, y)$ of size $M \times N$ as follows:

$$m = \frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y) \tag{18.20}$$

The mean is a very compact descriptor (one floating-point value per image or object) that provides a measure of the overall brightness of the corresponding image or object. It is also RST invariant. On the negative side, it has very limited expressiveness and discriminative power.

The *standard deviation* σ of an image is given by

$$\sigma = \sqrt{\sum_{j=0}^{L-1} (r_j - m)^2 p(r_j)} \tag{18.21}$$

where m is given by equation (18.19) or equation (18.20).

The square of the standard deviation is the *variance*, which is also known as the *normalized second-order moment* of the image.

The standard deviation provides a concise representation of the overall contrast. Similar to the mean, it is compact and RST invariant, but has limited expressiveness and discriminative power.

The *skew* of a histogram is a measure of its asymmetry about the mean level. It is defined as

$$\text{skew} = \frac{1}{\sigma^3} \sum_{j=0}^{L-1} (r_j - m)^3 p(r_j) \tag{18.22}$$

where σ is the standard deviation given by equation (18.21).

The sign of the skew indicates whether the histogram's tail spreads to the right (positive skew) or to the left (negative skew). The skew is also known as the *normalized third-order moment* of the image.

If the image's mean value (m), standard deviation (σ), and *mode*—defined as the histogram's highest peak—are known, the skew can be calculated as follows:

$$\text{skew} = \frac{m - \text{mode}}{\sigma} \tag{18.23}$$

The *energy* descriptor provides another measure of how the pixel values are distributed along the gray-level range: images with a single constant value have maximum energy (i.e., energy = 1); images with few gray levels will have higher energy than the ones with many gray levels. The energy descriptor can be calculated as

$$\text{energy} = \sum_{j=0}^{L-1} [p(r_j)]^2 \tag{18.24}$$

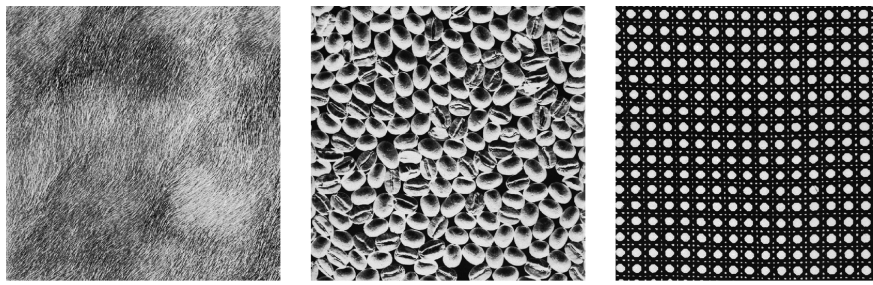

FIGURE 18.16 Example of images with smooth (a), coarse (b), and regular (c) texture. Images from the Brodatz textures data set. Courtesy of http://tinyurl.com/brodatz.

Histograms also provide information about the complexity of the image, in the form of *entropy* descriptor. The higher the entropy, the more complex the image.[8] Entropy and energy tend to vary inversely with one another. The mathematical formulation for entropy is

$$\text{entropy} = -\sum_{j=0}^{L-1} p(r_j) \log_2[p(r_j)] \tag{18.25}$$

Histogram-based features and their variants are usually employed as texture descriptors, as we shall see in Section 18.6.

18.6 TEXTURE FEATURES

Texture can be a powerful descriptor of an image (or one of its regions). Although there is not a universally agreed upon definition of texture, image processing techniques usually associate the notion of texture with image (or region) properties such as *smoothness* (or its opposite, *roughness*), *coarseness*, and *regularity*. Figure 18.16 shows one example of each and Figure 18.17 shows their histograms.

There are three main approaches to describe texture properties in image processing: structural, spectral, and statistical. In this book, we will focus on the statistical approaches, due to their popularity, usefulness, and ease of computing.[9]

One of the simplest set of statistical features for texture description consists of the following histogram-based descriptors of the image (or region): mean, variance (or its square root, the standard deviation), skew, energy (used as a measure of *uniformity*),

[8]This information also has implications for image coding and compression schemes (Chapter 17): the amount of coding redundancy that can be exploited by such schemes is inversely proportional to the entropy.

[9]Refer to "Learn More About It" section at the end of the chapter for references on structural and spectral methods for describing texture properties.

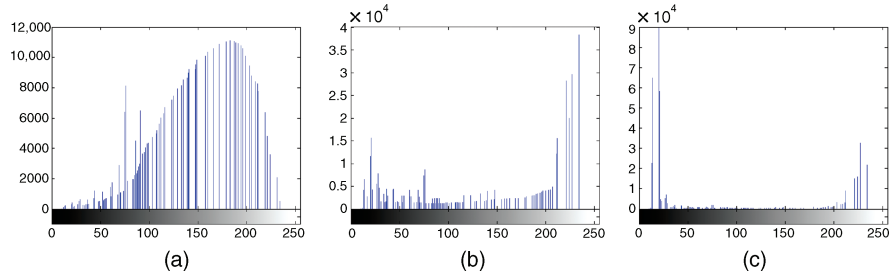

FIGURE 18.17 Histograms of images in Figure 18.16.

and entropy, all of which were introduced in Section 18.5. The variance is sometimes used as a normalized descriptor of roughness (R), defined as

$$R = 1 - \frac{1}{1 + \sigma^2} \tag{18.26}$$

where σ^2 is the normalized (to a [0, 1] interval) variance. $R = 0$ for areas of constant intensity, that is, smooth texture.

■ EXAMPLE 18.5

Table 18.3 shows the values of representative statistical texture descriptors for the three images in Figure 18.16, whose histograms are shown in Figure 18.17.

As expected, the regular texture has the highest uniformity (and lowest entropy) of all three. Moreover, the coarse texture shows a higher value for normalized roughness than the smooth texture, which is also consistent with our expectations.

Histogram-based texture descriptors are limited by the fact that the histogram does not carry any information about the spatial relationships among pixels. One way to circumvent this limitation consists in using an alternative representation for the pixel values that encodes their relative position with respect to one another. One such representation is the *gray-level cooccurrence matrix* $\mathbf{G}$, defined as a matrix whose element $g(i, j)$ represents the number of times that pixel pairs with intensities z_i and z_j occur in image $f(x, y)$ in the position specified by an operator $\mathbf{d}$. The vector $\mathbf{d}$ is known as *displacement vector*:

$$\mathbf{d} = (d_x, d_y) \tag{18.27}$$

TABLE 18.3 Statistical Texture Descriptors for the Three Images in Figure 18.16

Texture	Mean	Standard deviation	Roughness R	Skew	Uniformity	Entropy
Smooth	147.1459	47.9172	0.0341	−0.4999	0.0190	5.9223
Coarse	138.8249	81.1479	0.0920	−1.9095	0.0306	5.8405
Regular	79.9275	89.7844	0.1103	10.0278	0.1100	4.1181

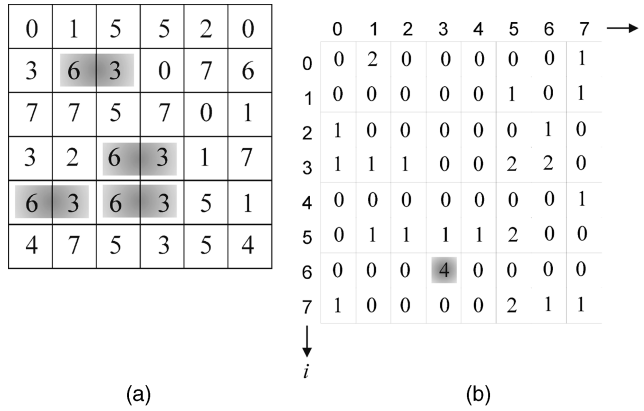

FIGURE 18.18 An image (a) and its cooccurrence matrix for $\mathbf{d} = (0, 1)$ (b).

where d_x and d_y are the displacements, in pixels, along the rows and columns, of the image, respectively.

Figure 18.18 shows an example of gray-level cooccurrence matrix for $\mathbf{d} = (0, 1)$. The array on the left is an image $F(x, y)$ of size 4×4 and $L = 8$ (L is the total number of gray levels). The array on the right is the gray-level cooccurrence matrix $\mathbf{G}$, using a convention $0 \leq i, j < L$. Each element in $\mathbf{G}$ corresponds to the number of occurrences of a pixel of gray-level i occurs to the left of a pixel of gray-level j. For example, since the value 6 appears to the left of the value 3 in the original image four times, the value of $g(6, 3)$ is equal to 4.

The contents of $\mathbf{G}$ clearly depend on the choice of $\mathbf{d}$. If we had chosen $\mathbf{d} = (1, 0)$ (which can be interpreted as "one pixel below"), the resulting gray-level cooccurrence matrix for the same image in Figure 18.18a would be the one in Figure 18.19.

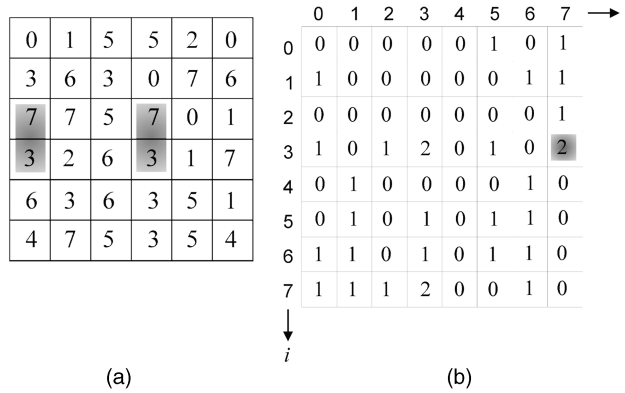

FIGURE 18.19 An image (a) and its cooccurrence matrix for $\mathbf{d} = (1, 0)$ (b).

The gray-level cooccurrence matrix can be normalized as follows:

$$N_g(i, j) = \frac{g(i, j)}{\sum_i \sum_j g(i, j)} \tag{18.28}$$

where $N_g(i, j)$ is the normalized gray-level cooccurrence matrix. Since all values of $N_g(i, j)$ lie between 0 and 1, they can be thought of as the probability that a pair of points satisfying **d** will have values (z_i, z_j).

Cooccurrence matrices can be used to represent the texture properties of an image. Instead of using the entire matrix, more compact descriptors are preferred. These are the most popular texture-based features that can be computed from a normalized gray-level cooccurrence matrix $N_g(i, j)$:

$$\text{Maximum probability} = \max_{i,j} N_g(i, j) \tag{18.29}$$

$$\text{Energy} = \sum_i \sum_j N_g^2(i, j) \tag{18.30}$$

$$\text{Entropy} = -\sum_i \sum_j N_g(i, j) \log_2 N_g(i, j) \tag{18.31}$$

$$\text{Contrast} = \sum_i \sum_j (i - j)^2 N_g(i, j) \tag{18.32}$$

$$\text{Homogeneity} = \sum_i \sum_j \frac{N_g(i, j)}{1 + |i - j|} \tag{18.33}$$

$$\text{Correlation} = \frac{\sum_i \sum_j (i-\mu_i)(j-\mu_j) N_g(i,j)}{\sigma_i \sigma_j} \tag{18.34}$$

where μ_i, μ_j are the means and σ_i, σ_j are the standard deviations of the row and column sums $N_g(i)$ and $N_g(j)$, defined as

$$N_g(i) = \sum_i N_g(i, j) \tag{18.35}$$

$$N_g(j) = \sum_j N_g(i, j) \tag{18.36}$$

■ EXAMPLE 18.6

Table 18.4 shows the values of representative statistical texture descriptors for the three images in Figure 18.16, whose histograms are shown in Figure 18.17. These descriptors were obtained by computing the normalized gray-level cooccurrence matrices with $\mathbf{d} = (0, 1)$.

As expected, the regular texture has the highest uniformity, highest homogeneity, and lowest entropy of all three. Although distinguishing between the coarse and smooth textures using these descriptors is possible (e.g., the image with coarse texture shows significantly higher correlation, uniformity, and homogeneity than the image with smooth texture), it is not as intuitive.

TABLE 18.4 Statistical Texture Descriptors for the Three Images in Figure 18.16

Texture	Max Probability	Correlation	Contrast	Uniformity (Energy)	Homogeneity	Entropy
Smooth	0.0013	0.5859	33.4779	0.0005	0.0982	7.9731
Coarse	0.0645	0.9420	14.5181	0.0088	0.3279	6.8345
Regular	0.1136	0.9267	13.1013	0.0380	0.5226	4.7150

18.7 TUTORIAL 18.1: FEATURE EXTRACTION AND REPRESENTATION

Goal

The goal of this tutorial is to learn how to use MATLAB to extract features from binary images and use these features to recognize objects within the image.

Objectives

- Learn how to use the `regionprops` function to extract features from binary objects.
- Learn how to perform feature selection and use the selected features to implement a simple, application-specific, heuristic classifier.

What You Will Need

- Test images `TPTest1.png`, `shapes23.png`, and `Test3.png`.

Procedure

1. Load test image `TPTest1.png` (Figure 18.20a) and display its contents.

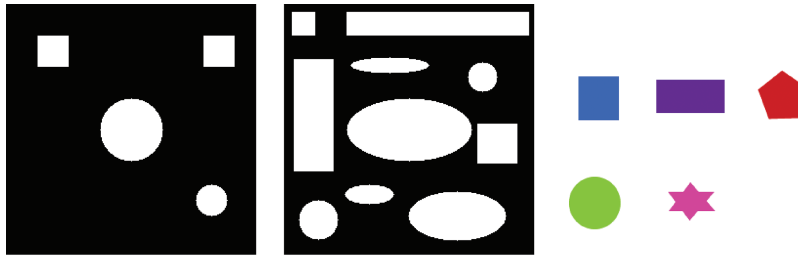

FIGURE 18.20 Test images for this tutorial: (a) steps 1–6; (b) step 7; (c) step 11.

```
J = imread('TPTest1.png');
imshow(J)
```

2. Use `bwboundaries` to display the boundaries of the objects in the test image.

```
[B,L] = bwboundaries(J);
figure; imshow(J); hold on;
for k=1:length(B),
    boundary = B{k};
    plot(boundary(:,2),boundary(:,1),'g','LineWidth',2);
end
```

3. Use `bwlabel` to label the connected regions (i.e., objects) in the test image, pseudocolor them, and display each of them with an associated numerical label.

```
[L, N] = bwlabel(J);
RGB = label2rgb(L,'hsv',[.5 .5 .5], 'shuffle');

figure; imshow(RGB); hold on;
for k=1:length(B),
    boundary = B{k};
    plot(boundary(:,2),boundary(:,1),'w','LineWidth',2);
    text(boundary(1,2)-11,boundary(1,1)+11,num2str(k),'Color','y',...
       'FontSize',14,'FontWeight','bold');
end
```

Question 1 What is the value of `N` returned by `bwlabel`? Does it make sense to you?

4. Use `regionprops` to extract the following binary features for each object in the image (top left square, top right square, small circle, big circle): area, centroid, orientation, Euler number, eccentricity, aspect ratio, perimeter, and thinness ratio.
5. Organize the feature values and object names in a table (see Table 18.5), for easier comparative analysis.

TABLE 18.5 Table for Feature Extraction Results

Object	Area	Centroid (row, col)	Orientation (degrees)	Euler number	Eccentricity	Aspect ratio	Perimeter	Thiness ratio
Top left square								
Big circle								
Small circle								
Top right square								

```
stats = regionprops(L,'all');
temp = zeros(1,N);
for k = 1:N
    % Compute thinness ratio
    temp(k) = 4*pi*stats(k,1).Area / (stats(k,1).Perimeter)^2;
    stats(k,1).ThinnessRatio = temp(k);

    % Compute aspect ratio
    temp(k) = (stats(k,1).BoundingBox(3))/(stats(k,1).BoundingBox(4));
    stats(k,1).AspectRatio = temp(k);
end
```

Question 2 Do the results obtained for the extracted features correspond to your expectations? Explain.

Question 3 Which of the extracted features have the **best** discriminative power to help tell squares from circles? Explain.

Question 4 Which of the extracted features have the **worst** discriminative power to help tell squares from circles? Explain.

Question 5 Which of the extracted features are ST invariant, that is, robust to changes in size and translation? Explain.

Question 6 If you had to use only one feature to distinguish squares from circles, in a ST-invariant way, which feature would you use? Why?

6. Plot the 2D feature vectors obtained using the area and thinness ratio of each object.

```
areas = zeros(1,N);
for k = 1:N
    areas(k) = stats(k).Area;
end

TR = zeros(1,N);
for k = 1:N
    TR(k) = stats(k).ThinnessRatio;
end

cmap = colormap(lines(16))
for k = 1:N
    scatter(areas(k), TR(k), [], cmap(k,:), 'filled'), ...
        ylabel('Thinness Ratio'), xlabel('Area')
    hold on
end
```

7. Repeat steps 1–6 for a different test image, `Test3.png` (Figure 18.20b).
8. Write MATLAB code to implement a heuristic three-class classifier capable of discriminating *squares* from *circles* from *unknown* shapes. *Hints*: Use a subset of features with enough discriminative power and encode your solution using `if-else-if` statements. Use the code snippet below to get started.[10]

```
name = cell(1,N);
for k = 1:N
    if (TR(k) > 0.9)
        name{1,k}='circle';
    else if (TR (k) > 0.8)
            name{1,k} = 'square';
        else
            name{1,k} = 'other';
        end
    end
end
```

9. Test your solution using the `TPTest1.png` and `Test3.png` test images.
10. Test your solution using different test images.
11. Extend your classifier to be able to process color images, for example, `shapes23.png` (Figure 18.20c).

WHAT HAVE WE LEARNED?

- Feature extraction is the process by which certain features of interest within an image are detected and represented for further processing. It is a critical step in most computer vision and image processing solutions because it marks the transition from pictorial to nonpictorial (alphanumerical, usually quantitative) data representation. From a pragmatic standpoint, having extracted meaningful features from an image enables the use of a vast array of pattern recognition and classification techniques to be applied to the resulting data.
- The types of features that can be extracted from an image depend on the type of image (e.g., binary, gray-level, or color), the level of granularity (entire image or individual regions) desired, and the context of the application. Several representative techniques for feature extraction using pixel intensity, texture, and relative positioning have been described in this chapter.
- Once the features have been extracted, they are usually represented in an alphanumerical way for further processing. The actual representation depends on the technique used: chain codes are normally treated as arrays of numbers,

[10]This solution, although naive and inelegant, works for both test images used so far. In Chapter 19, you will learn how to build better classifiers.

whereas quantitative features (e.g., measures of an object's area, perimeter, and number of holes) are usually encoded into a *feature vector*.

LEARN MORE ABOUT IT

- Polygonal approximation techniques are described in Chapter 11 of [GW08], Chapter 11 of [GWE04], Section 5.3 of [SOS00], and Section 8.2 of [SHB08].
- Section 11.1 of [GW08] and Section 17.6 of [Pra07] discuss boundary following algorithms in greater detail.
- Many structural, spectral, and statistical texture descriptors have been proposed in the literature and entire books have been written on the topic of texture analysis, among them are [MXS08], [Pet06], and [TT90].

18.8 PROBLEMS

18.1 Compute the first difference for the chain code: 0103212111021103.

18.2 Sketch the signature plots for the following geometrical figures:

(a) Rectangle

(b) Isosceles triangle

(c) Ellipse

(d) 6-point star

18.3 Write a MATLAB function to compute the gray-level cooccurrence matrix for an image $f(x, y)$ and a displacement vector **d**, which should be passed as parameters.