CHAPTER

Introduction to texture description, segmentation, and classification

8

CHAPTER OUTLINE HEAD

8.1 Overview

This chapter is concerned with how we can use many of the feature extraction and description techniques presented earlier to characterize regions in an image. The aim here is to describe how we can collect together measurements for purposes of recognition, using texture by way of introduction and as a vehicle for using **feature extraction** in **recognition**. We shall also use it as a mechanism to introduce distance measures which describe how different features appear to be by their measurements.

We shall first look at what is meant by texture and then how we can use Fourier transform techniques, statistics, and region measures to describe it. We shall then look at how the measurements provided by these techniques, the

Table 8.1 Overview of Chapter 8

Main Topic	Subtopics	Main Points
Texture description	What is image texture and how do we determine sets of **numbers** that allow us to be able to **recognize** it	Feature extraction: Fourier transform, *cooccurrence matrices,* and regions; modern techniques: *local binary patterns* (LPB) and *uniform LBP*; feature *descriptions*: energy, entropy, and inertia
Distance measures	How different do (texture) features appear to be, from the measurements we have made; different ways of measuring distance and understanding dissimilarity	Distance metrics: *Manhattan* city block and *Euclidean* (L_1 and L_2 distances), *Mahalanobis*, *Bhattacharyya*, and cosine; construction, visualization, and the confusion matrix
Texture classification	How do we **associate** the numbers we have derived with those that we have already **stored** for known examples	*k-nearest neighbor* rule, *support vector machines*, and other **classification** approaches
Texture segmentation	How do we find **regions** of texture within images	Convolution, tiling, and thresholding

description of the texture, can be collected together to recognize it. Finally, we shall label an image according to the texture found within it, to give a segmentation into classes known to exist within the image. Since we could be recognizing shapes described by Fourier descriptors, region measures, or by other feature extraction and description approaches, the material is actually general and could be applied for purposes of recognition to measures other than texture (Table 8.1).

8.2 What is texture?

Texture is actually a very nebulous concept, often attributed to human perception, as either the feel or the appearance of (woven) fabric. Everyone has their own interpretation as to the nature of texture; there is no mathematical definition for texture, it simply exists. By way of reference, let us consider one of the dictionary definitions (Oxford Concise English Dictionary, 4th Ed, 1958):

> "***texture*** n., & v.t. ***1.*** n. *arrangement of threads etc. in textile fabric. characteristic feel due to this; arrangement of small constituent parts, perceived structure,* (of *skin, rock, soil, organic tissue, literary work, etc.); representation of structure and detail of objects in art; ...*"

That covers quite a lot. If we change "threads" for "pixels," the definition could apply to images (except for the bit about artwork). Essentially, texture can be what

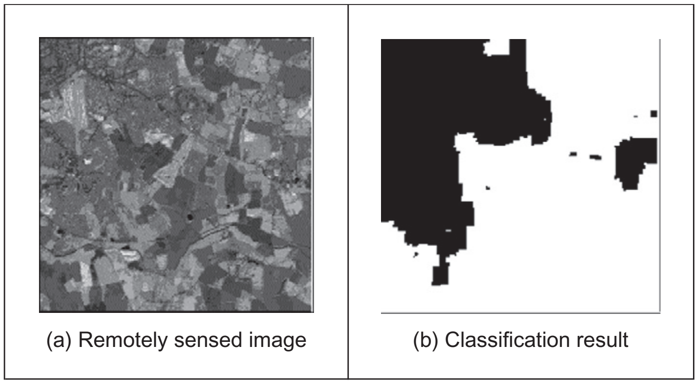


FIGURE 8.1

Example texture analysis.

we define it to be. Why might we want to do this? By way of example, analysis of remotely sensed images is now a major application of image-processing techniques. In such analysis, pixels are labeled according to the categories of a required application, such as whether the ground is farmed or urban in land-use analysis or water for estimation of surface analysis. An example of remotely sensed image is given in Figure 8.1(a) which is of an urban area (in the top left) and some farmland. Here, the image resolution is low, and each pixel corresponds to a large area of the ground. Square groups of pixels have then been labeled either as urban or as farmland according to their texture properties as shown in Figure 8.1(b) where black represents the area classified as urban and white is for the farmland. In this way, we can assess the amount of area that urban areas occupy. As such, we have used **real** textures to label pixels, the perceived textures of the urban and farming areas.

As an alternative definition of texture, we can consider it as being defined by the database of images that researchers use to test their algorithms. Many texture researchers have used a database of pictures of textures (Brodatz, 1968), produced for artists and designers, rather than for digital image analysis. Parts of three of the Brodatz texture images are given in Figure 8.2. Here, the French canvas (Brodatz index D20) shown in Figure 8.2(a) is a detail of Figure 8.2(b) (Brodatz index D21), taken at four times the magnification. The beach sand shown in Figure 8.2(c) (Brodatz index D29) is clearly of a different texture to that of cloth. Given the diversity of texture, there are now many databases available on the Web, at the sites given in Chapter 1 or at this book's web site. Alternatively, we can define texture as a quantity for which texture extraction algorithms provide meaningful results. One study (Karru et al., 1996) suggests

"The answer to the question 'is there any texture in the image?' depends not only on the input image, but also on the goal for which the image texture is used and the textural features that are extracted from the image."

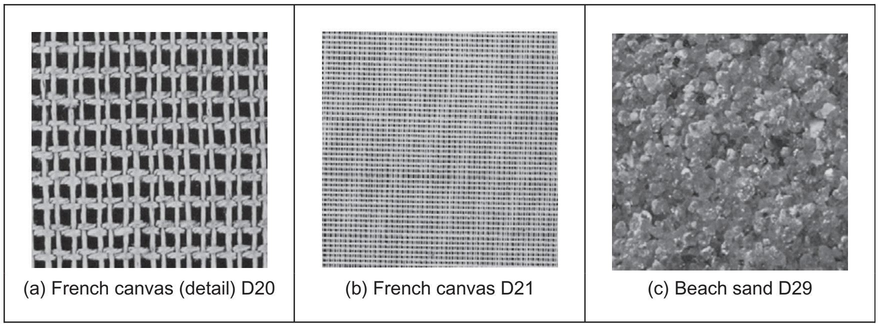


FIGURE 8.2

Three Brodatz textures.

As we shall find, texture analysis has a rich history in image processing and computer vision, and there is even a book devoted to texture analysis (Petrou and Sevilla, 2006). Despite this, approaches which synthesize texture are relatively recent. This is of course motivated also by graphics, and the need to include texture to improve the quality of the rendered scenes (Heckbert, 1986). By way of example, one well-known approach to texture synthesis is to use a Markov random field (Efros and Leung, 1999), but we shall not dwell on that here.

Essentially, there is no unique definition of texture and there are many ways to describe and extract it. It is a very large and exciting field of research and there continues to be many new developments.

Clearly, images will usually contain samples of more than one texture. Accordingly, we would like to be able to **describe** texture (*texture descriptions* are measurements which characterize a texture) and then to *classify* it (classification attributes the correct class label to a set of measurements) and then, perhaps to segment an image according to its texture content. We have used similar classification approaches to characterize the shape descriptions in the previous chapter. Actually these are massive fields of research that move on to the broad subject of pattern recognition. We shall look at an introduction here; later references will point you to topics of particular interest and to some of the more recent developments. The main purpose of this introduction is to show how the measurements can be collected together to recognize objects. Texture is used as the vehicle for this since it is a region-based property that has not as yet been covered. Since texture itself is an enormous subject, you will find plenty of references to established approaches and to surveys of the field. First, we shall look at approaches to deriving the features (measurements) which can be used to describe textures. Broadly, these can be split into **structural** (transform-based), **statistical**, and **combination** approaches. Clearly, the frequency content of an image will reflect its texture; we shall start with Fourier. First, though, we shall consider some of the required properties of the descriptions.

8.3 Texture description

8.3.1 Performance requirements

The purpose of texture description is to derive some measurements that can be used to classify a particular texture. As such, there are **invariance** requirements on the measurements, as there were for shape description. Actually, the invariance requirements for feature extraction, namely invariance to position, scale, and rotation, can apply equally to texture extraction. After all texture is a feature, albeit a rather nebulous one as opposed to the definition of a shape. Clearly, we require **position** invariance: the measurements describing a texture should not vary with the position of the analyzed section (of a larger image). Also, we require **rotation** invariance but this is not as strong a requirement as position invariance; the definition of texture does not imply knowledge of orientation but could be presumed to. The least strong requirement is that of **scale** for this depends primarily on application. Consider using texture to analyze forests in remotely sensed images. Scale invariance would imply that closely spaced young trees should give the same measure as widely spaced mature trees. This should be satisfactory if the purpose is only to analyze foliage cover. It would be unsatisfactory if the purpose was to measure age for purposes of replenishment, since a scale-invariant measure would be of little use as it could not, in principle, distinguish between young trees and old ones.

Unlike feature extraction, texture description rarely depends on edge extraction since one main purpose of edge extraction is to remove reliance on overall **illumination** level. The higher-order invariants, such as perspective invariance, are rarely applied to texture description. This is perhaps because many applications are like remotely sensed imagery or are in constrained industrial application where the camera geometry can be controlled.

8.3.2 Structural approaches

The most basic approach to texture description is to generate the Fourier transform of the image and then to group the transform data in some way so as to obtain a set of measurements. Naturally, the size of the set of measurements is smaller than the size of the image's transform. In Chapter 2, we saw how the transform of a set of horizontal lines was a set of vertical spatial frequencies (since the point spacing varies along the vertical axis). Here, we must remember that for display we rearrange the Fourier transform so that the d.c. component (the zero-frequency component) is at the center of the presented image.

The transforms of the three Brodatz textures of Figure 8.2 are shown in Figure 8.3. Figure 8.3(a) shows a collection of frequency components which are then replicated with the same structure (consistent with the Fourier transform) in Figure 8.3(b). (Figure 8.3(a) and (b) also shows the frequency scaling property of the Fourier transform: greater magnification reduces the high-frequency content.)

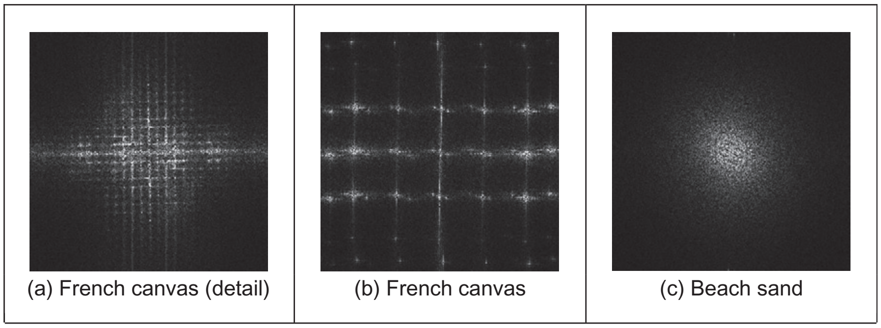


FIGURE 8.3

Fourier transforms of the three Brodatz textures.

Figure 8.3(c) is clearly different in that the structure of the transform data is spread in a different manner to that of Figure 8.3(a) and (b). Naturally, these images have been derived by application of the FFT which we shall denote as

$$\mathbf{FP} = \Im(\mathbf{P}) \tag{8.1}$$

where $\mathbf{FP}_{u,v}$ and $\mathbf{P}_{x,y}$ are the transform (spectral) and pixel data, respectively. One clear advantage of the Fourier transform is that it possesses shift invariance (Section 2.6.1): the transform of a bit of (large and uniform) cloth will be the same, whatever segment we inspect. This is consistent with the observation that phase is of little use in Fourier-based texture systems (Pratt, 1992), so the modulus of the transform (its magnitude) is usually used. The transform is of the same size as the image, even though conjugate symmetry of the transform implies that we do not need to use all its components as measurements. As such we can **filter** the Fourier transform (Section 2.8) so as to select those frequency components deemed to be of interest to a particular application. Alternatively, it is convenient to collect the magnitude transform data in different ways to achieve a reduced set of measurements. First though, the transform data can be normalized by the sum of the squared values of each magnitude component (excepting the zero-frequency components, those for $u = 0$ and $v = 0$), so that the magnitude data is invariant to linear shifts in illumination to obtain normalized Fourier coefficients **NFP** as

$$\mathbf{NFP}_{u,v} = \frac{|\mathbf{FP}_{u,v}|}{\sqrt{\sum_{(u\neq 0)\wedge(v\neq 0)} |\mathbf{FP}_{u,v}|^2}} \tag{8.2}$$

The denominator is then a measure of total power, so the magnitude data becomes invariant to linear shifts. Alternatively, histogram equalization

(Section 3.3.3) can provide such invariance but is more complicated than using Eq. (8.2). The spectral data can then be described by the *entropy*, h, as

$$h = \sum_{u=1}^{N} \sum_{v=1}^{N} \mathbf{NFP}_{u,v} \log(\mathbf{NFP}_{u,v}) \tag{8.3}$$

which gives compression weighted by a measure of the information content. A uniformly distributed image would have zero entropy, so the entropy measures by how much the image differs from a uniform distribution. Another measure is the *energy*, e, as

$$e = \sum_{u=1}^{N} \sum_{v=1}^{N} (\mathbf{NFP}_{u,v})^2 \tag{8.4}$$

which gives priority to larger items (by virtue of the squaring function). The measure is then appropriate when it is the larger values which are of interest; it will be of little use when the measure has a uniform distribution. Another measure is the *inertia*, i, defined as

$$i = \sum_{u=1}^{N} \sum_{v=1}^{N} (u-v)^2 \mathbf{NFP}_{u,v} \tag{8.5}$$

which emphasizes components which have a large separation. As such, each measure describes a different facet of the underlying data. These measures are shown for the three Brodatz textures in Code 8.1. In a way, they are like the shape descriptions in the previous chapter: the measures should be the same for the same object and should differ for a different one. Here, the texture measures are actually different for each of the textures. Perhaps the detail in the French canvas (Code 8.1(a)) could be made to give a closer measure to that of the full resolution (Code 8.1(b)) by using the frequency scaling property of the Fourier transform, as discussed in Section 2.6.3. The beach sand clearly gives a different set of measures from the other two (Code 8.1(c)). In fact, the beach sand in Code 8.1(c) would appear to be more similar to the French canvas in Code 8.1(b), since the inertia and energy measures are much closer than those for Code 8.1(a) (only the entropy measure in Code 8.1(a) is closest to Code 8.1(b)). This is consistent with the images: each of the beach sand and French canvas has a large proportion of higher frequency information, since each is a finer texture than that of the detail in the French canvas.

```
entropy(FD20)=-253.11
inertia(FD20)=5.55·10^5
energy(FD20)=5.41
```
(a) French canvas (detail)

```
entropy(FD21)=-196.84
inertia(FD21)=6.86·10^5
energy(FD21)=7.49
```
(b) French canvas

```
entropy(FD29)=-310.61
inertia(FD29)=6.38·10^5
energy(FD29)=12.37
```
(c) Beach sand

CODE 8.1

Measures of the Fourier transforms of the three Brodatz textures.

By Fourier analysis, the measures are inherently **position** invariant. Clearly, the entropy, inertia, and energy are relatively immune to **rotation**, since order is not important in their calculation. Also, the measures can be made **scale** invariant, as a consequence of the frequency scaling property of the Fourier transform. Finally, the measurements (by virtue of the normalization process) are inherently invariant to linear changes in **illumination**. Naturally, the descriptions will be subject to noise. In order to handle large datasets we need a larger set of measurements (larger than the three given here) in order to better discriminate between different textures. Other measures can include:

1. the energy in the major peak,
2. the Laplacian of the major peak,
3. the largest horizontal frequency magnitude,
4. the largest vertical frequency magnitude.

Among others, these are elements of Liu's features (Liu and Jernigan, 1990) chosen in a way aimed to give Fourier transform-based measurements good performance in noisy conditions.

Naturally, there are many other transforms and these can confer different attributes in analysis. The wavelet transform is very popular since it allows for localization in time and frequency (Laine and Fan, 1993; Lu et al., 1997). Other approaches use the Gabor wavelet (Bovik et al., 1990; Jain and Farrokhnia, 1991; Daugman and High, 1993; Dunn et al., 1994), as introduced in Section 2.7.3. One comparison between Gabor wavelets and tree- and pyramidal-structured wavelets suggested that Gabor has the greater descriptional ability, at penalty of greater computational complexity (Pichler et al., 1996; Grigorescu et al., 2002). There has also been interest in Markov random fields (Gimmel'farb and Jain, 1996; Wu and Wei, 1996). Others, such as Walsh transform (where the basis functions are 1s and 0s), appear yet to await application in texture description, no doubt due to basic properties. In fact, one survey (Randen and Husoy, 2000) includes the use of Fourier, wavelet, and discrete cosine transforms (Section 2.7.1) for texture characterization. These approaches are structural in nature: an image is viewed in terms of a transform applied to a whole image as such exposing its **structure**. This is like the dictionary definition of an arrangement of parts. Another part of the dictionary definition concerned **detail**: this can of course be exposed by analysis of the high-frequency components, but these can be prone to noise. An alternative way to analyze the detail is to consider the **statistics** of an image.

8.3.3 Statistical approaches

The most famous statistical approach is the *cooccurrence matrix*. This was the result of the first approach to describe, and then classify, image texture (Haralick et al., 1973). It remains popular today by virtue of good performance. The cooccurrence matrix contains elements that are counts of the number of pixel pairs for

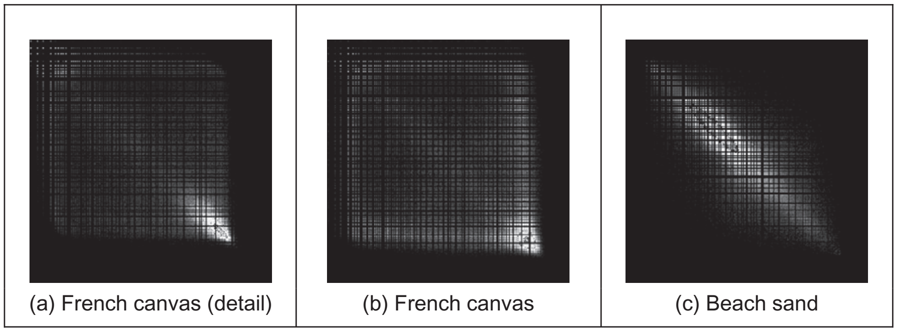


FIGURE 8.4

Cooccurrence matrices of the three Brodatz textures.

specific brightness levels, when separated by some distance and at some relative inclination. For brightness levels $b1$ and $b2$, the cooccurrence matrix $\mathbf{C}$ is

$$\mathbf{C}_{b1,b2} = \sum_{x=1}^{N}\sum_{y=1}^{N}(\mathbf{P}_{x,y} = b1) \wedge (\mathbf{P}_{x',y'} = b2) \tag{8.6}$$

where $\wedge$ denotes the logical AND operation and where the x coordinate x' is the offset given by the specified distance d and inclination θ by

$$x' = x + d\cos(\theta) \quad \forall (d \in 1,\ \max(d)) \wedge (\theta \in 0, 2\pi) \tag{8.7}$$

and the y coordinate y' is

$$y' = y + d\sin(\theta) \quad \forall (d \in 1,\ \max(d)) \wedge (\theta \in 0, 2\pi) \tag{8.8}$$

When Eq. (8.6) is applied to an image, we obtain a square, symmetric, matrix whose dimensions equal the number of gray levels in the picture. The cooccurrence matrices for the three Brodatz textures of Figure 8.2 are shown in Figure 8.4. In the cooccurrence matrix generation, the maximum distance was one pixel and the directions were set to select the four nearest neighbors of each point. Now the result for the two samples of French canvas (Figure 8.4(a) and (b)) appear to be much more similar and quite different to the cooccurrence matrix for sand (Figure 8.4(c)). As such, the cooccurrence matrix looks like it can better expose the underlying nature of texture than can the Fourier description. This is because the cooccurrence measures spatial relationships between brightness, as opposed to frequency content. This clearly gives alternative results. To generate results faster, the number of gray levels can be reduced by brightness scaling of the whole image, reducing the dimensions of the cooccurrence matrix, but this reduces discriminatory ability.

These matrices have been achieved by the implementation in Code 8.2. The subroutine `tex_cc` generates the cooccurrence matrix of an image `im` given a maximum distance `d` and a number of directions `dirs`. If `d` and `dirs` are set to 1 and 4, respectively (as was used to generate the results in Figure 8.4), then the cooccurrence will be evaluated from a point and its four nearest neighbors. First, the cooccurrence matrix is cleared. Then, for each point in the image and for each value of distance and relative inclination (and so long as the two points are within the image), the element of the cooccurrence matrix indexed by the brightness of the two points is incremented. There is a dummy operation after the incrementing process: this has been introduced for layout reasons (otherwise the Mathcad code would stretch out sideways, too far). Finally, the completed cooccurrence matrix is returned. Note that even though the cooccurrence matrix is symmetric, this factor cannot be used to speed its production.

```
tex_cc(im,dist,dirs):= | for x∈0..maxbri
                       |   for y∈0..maxbri
                       |     $cocc_{y,x}$←0
                       | for x∈0..cols(im)-1
                       |   for y∈0..rows(im)-1
                       |     for r∈1..dist
                       |        for θ∈0,$\frac{2\cdot\pi}{dirs}$..2·π
                       |          | xc←floor(x+r·cos(θ))
                       |          | yc←floor(y+r·sin(θ))
                       |          | if(0≤yc)·(yc<rows(im))·(0≤xc)·(xc<cols(im))
                       |          |   | $cocc_{im_{y,x},im_{yc,xc}}$←$cocc_{im_{y,x},im_{yc,xc}}$+1
                       |          |   | I←1
                       | cocc
```

CODE 8.2

Cooccurrence matrix generation.

Again, we need measurements that describe these matrices. We shall use the measures of entropy, inertia, and energy defined earlier. The results are shown in Code 8.3. Unlike visual analysis of the cooccurrence matrices, the difference between the measures of the three textures is less clear: classification from them will be discussed later. Clearly, the cooccurrence matrices have been reduced to only three different measures. In principle, these measurements are again invariant to linear shift in illumination (by virtue of brightness comparison) and to rotation (since order is of no consequence in their description and rotation only affects cooccurrence by discretization effects). As with Fourier, scale can affect the structure of the cooccurrence matrix, but the description can be made scale invariant. Gray level difference statistics (a first-order measure) were later added to improve descriptional capability (Weska et al., 1976). Other statistical approaches include the statistical feature matrix (Wu and Chen, 1992) with the advantage of faster generation.

entropy(CCD20)=7.052·10^5 inertia(CCD20)=5.166·10^8 energy(CCD20)=5.16·10^8	entropy(CCD21)=5.339·10^5 inertia(CCD21)=1.528·10^9 energy(CCD21)=3.333·10^7	entropy(CCD29)=6.445·10^5 inertia(CCD29)=1.139·10^8 energy(CCD29)=5.315·10^7
(a) French canvas (detail)	(b) French canvas	(c) Beach sand

CODE 8.3

Measures of cooccurrence matrices of the three Brodatz textures.

8.3.4 Combination approaches

The previous approaches have assumed that we can represent textures by purely structural or purely statistical description combined in some appropriate manner. Since texture is not an exact quantity, and is more a nebulous one, there are naturally many alternative descriptions. One approach (Chen et al., 1995) suggested that texture combines geometrical structures (say in patterned cloth) with statistical ones (say in carpet) and has been shown to give good performance in comparison with other techniques and by using the **whole** Brodatz dataset. The technique is called statistical geometric features (SGF), reflecting the basis of its texture description. This is not a dominant texture characterization: the interest here is that we shall now see the earlier **shape measures** in action, describing texture. Essentially, geometric features are derived from images and then described by using statistics. The geometric quantities are actually derived from $NB-1$ binary images $\mathbf{B}$ which are derived from the original image $\mathbf{P}$ (which has NB brightness levels). These binary images are given by

$$\mathbf{B}(\alpha)_{x,y} = \left| \begin{array}{ll} 1, & \text{if } \mathbf{P}_{x,y} \geq \alpha \\ 0, & \text{otherwise} \end{array} \right. \quad \forall \alpha \in 1, NB \tag{8.9}$$

Then, the points in each binary region are connected into regions of 1s and 0s. Four geometrical measures are made on these data. First, in each binary plane, the number of regions of 1s and 0s (the number of connected sets of 1s and 0s) is counted to give $NOC1$ and $NOC0$. Then, in each plane, each of the connected regions is described by its **irregularity** which is a local shape measure of a region $\mathbf{R}$ of connected 1s giving irregularity $I1$ defined by

$$I1(\mathbf{R}) = \frac{1+\sqrt{\pi}\max\limits_{i\in\mathbf{R}}\sqrt{(x_i-\overline{x})^2+(y_i-\overline{y})^2}}{\sqrt{N(\mathbf{R})}} - 1 \tag{8.10}$$

where x_i and y_i are the coordinates of points within the region, $\overline{x}$ and $\overline{y}$ are the region's centroid (its mean x and y coordinates), and N is the number of points within (i.e., the area of) the region. The irregularity of the connected 0s, $I0(\mathbf{R})$ is similarly defined. When this is applied to the regions of 1s and 0s, it gives two further geometric measures, $IRGL1(i)$ and $IRGL0(i)$, respectively. To balance the contributions from different regions, the irregularity of the regions of 1s in a particular plane is formed as a weighted sum $WI1(\alpha)$ as

$$WI1(\alpha) = \frac{\sum\limits_{\mathbf{R} \in \mathbf{B}(\alpha)} N(\mathbf{R})I(\mathbf{R})}{\sum\limits_{\mathbf{R} \in \mathbf{P}} N(\mathbf{R})} \tag{8.11}$$

giving a single irregularity measure for each plane. Similarly, the weighted irregularity of the connected 0s is $WI0$. Together with the two counts of connected regions, $NOC1$ and $NOC0$, the weighted irregularities give the four geometric measures in SGF. The statistics are derived from these four measures. The derived statistics are the maximum value of each measure across all binary planes, M. Using $m(\alpha)$ to denote any of the four measures, the maximum is

$$M = \max_{\alpha i \in 1,NB} (m(\alpha)) \tag{8.12}$$

the average $\overline{m}$ is

$$\overline{m} = \frac{1}{255} \sum_{\alpha=1}^{NB} m(\alpha) \tag{8.13}$$

the sample mean $\overline{s}$ is

$$\overline{s} = \frac{1}{\sum\limits_{\alpha=1}^{NB} m(\alpha)} \sum_{\alpha=1}^{NB} \alpha m(\alpha) \tag{8.14}$$

and the final statistic is the sample standard deviation, ssd, as

$$\text{ssd} = \sqrt{\frac{1}{\sum\limits_{\alpha=1}^{NB} m(\alpha)} \sum_{\alpha=1}^{NB} (\alpha - \overline{s})^2 m(\alpha)} \tag{8.15}$$

The irregularity measure can be replaced by **compactness** (Section 7.3.1), but compactness varies with rotation, though this was not found to influence results much (Chen et al., 1995).

In order to implement these measures, we need to derive the sets of connected 1s and 0s in each of the binary planes. This can be achieved by using a version of the `connect` routine in hysteresis thresholding (Section 4.2.1.5). The reformulation is necessary because the `connect` routine just labels connected points, whereas the irregularity measures require a list of points in the connected region so that the centroid (and hence the maximum distance of a point from the centroid) can be calculated. The results for four of the measures (for the region of 1s, the maximum and average values of the number of connected regions and of the weighted irregularity) are shown in Code 8.4. Again, the set of measures is different for each texture. Of note the last measure, $\overline{m}(WI1)$, does not appear to offer much discriminatory capability here, whereas the measure $M(WI1)$ appears to be a much more potent descriptor. Classification, or **discrimination**, is to select which class the measures refer to.

$M(NOC1)$=52.0 $\bar{m}(NOC1)$=8.75 $M(WI1)$=1.50 $\bar{m}(WI1)$=0.40 (a) French canvas (detail)	$M(NOC1)$=178 $\bar{m}(NOC1)$=11.52 $M(WI1)$=1.42 $\bar{m}(WI1)$=0.35 (b) French canvas	$M(NOC1)$=81 $\bar{m}(NOC1)$=22.14 $M(WI1)$=1.00 $\bar{m}(WI1)$=0.37 (c) Beach sand

CODE 8.4

Four of the SGF measures of the three Brodatz textures.

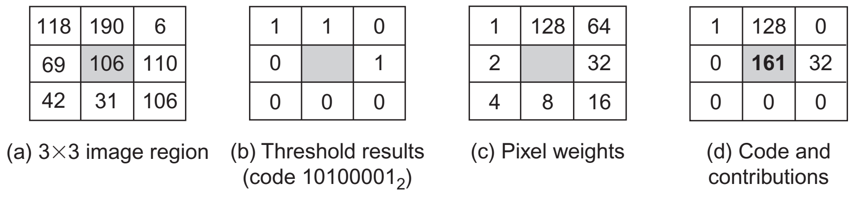
8.3.5 Local binary patterns

The *local binary pattern* (LBP) texture description is a relatively recent approach and it has rapidly gained favor in the research community due to its attractive performance capabilities. There was an early approach (Ojala et al., 1996) which gives the concept, and this was then refined over some time to give the most recent approach (Ojala et al., 2002). (It derives originally from the University of Oulu which must be the closest university to the Santa theme park—but that is mere digression.) We shall progress from where it started, the basic LBP to the current version since the most recent approach would be rather a mind stretch without this progression. Essentially, for a 3×3 region, the basic LBP is derived by comparing the center point with its neighbors, to derive a code which is stored at the center point. For points P and $\mathbf{P}_x$ the process depends on thresholding, which is the function

$$s(x) = \left| \begin{array}{ll} 1, & \text{if } \mathbf{P}_x > P \\ 0, & \text{otherwise} \end{array} \right. \tag{8.16}$$

The code is derived from binary weighting applied to result of thresholding (which is equivalent to thresholding the points neighboring the center point and then unwrapping the code as a binary code). So the code LBP for a point P with eight neighbors x is

$$\text{LBP} = \sum_{x\in 1,8} s(x)\times 2^{x-1} \tag{8.17}$$



118	190	6
69	106	110
42	31	106

(a) 3×3 image region

1	1	0
0		1
0	0	0

(b) Threshold results (code 10100001_2)

1	128	64
2		32
4	8	16

(c) Pixel weights

1	128	0
0	**161**	32
0	0	0

(d) Code and contributions

FIGURE 8.5

Constructing a local binary pattern code.

This is shown in Figure 8.5 where point LBP is the center point and the eight values for x address its eight immediate neighbors. For the 3×3 patch in Figure 8.5(a), the value of the center point is exceeded three times, so there are three 1s in the resulting code in Figure 8.5(b). When this is unwrapped clockwise from the top left point (and the top middle point is the most significant bit) the resulting code is 10100001_2. When this is considered as a binary code, with weightings shown in Figure 8.5(c), we arrive at a final value LBP = 161 (Figure 8.5(d)). Naturally, the thresholding process, the unwrapping, and the weighting can be achieved in different ways, but it is essential that it is consistent across the whole image. The code P now encodes the local intensity structure: the local binary pattern.

The basic LBP code was complemented by two local measures: **contrast** and **variance**. The former of these was computed from the difference between points encoded as a "1" and those encoded as a "0," the variance was computed from the four neighbor pixels aiming to reflect pattern correlation as well as contrast. Of these two complementary measures, contrast was found to add most to discriminatory capability.

The LBP approach then determines a histogram of the codes derived for an entire image and this histogram describes the texture. The approach is inherently **translation** invariant by its formulation: a texture which is shifted should achieve the same histogram of LBP codes. By virtue of its formulation, the basic process is not **scale** or **rotation** invariant, as in the case of rotation, a different weighting will be applied to the point comparisons resulting in a different code value. The histogram of LBP values for the French canvas texture Figure 8.6(a) is shown in Figure 8.6(b). The histogram is also shown for a version of the French canvas image which has been shifted leftward (Figure 8.6(c)) by 40 pixels (cyclic shift and so the image wraps—as in Section 2.6.1) and the resulting histogram (Figure 8.6(d)) appears very similar in structure, as expected. There is in fact some difference between points on the two histograms since we are dealing with real images—but it is at most a difference of around 20 which is very small given that some of the histogram values are counts of over 4×10^3 pixels and thus cannot be determined by visual inspection of the histograms.

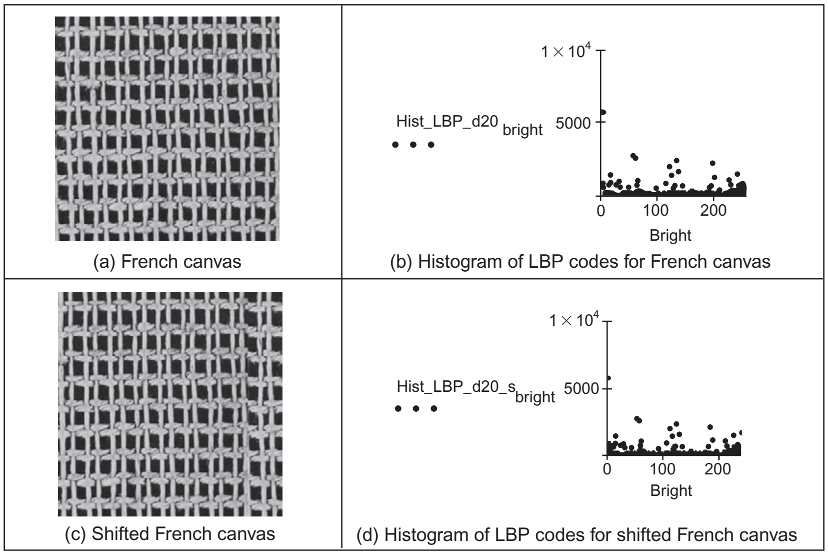


FIGURE 8.6

Shift invariant local binary pattern histograms.

The next consideration is scale invariance. This requires consideration of points at a greater distance. If the space is sampled in a circular manner, and P points are derived at radius R, then the coordinate equations for $i \in (1,P)$ are

$$x(i) = \begin{bmatrix} x_0 + R\cos\left(\frac{2\pi}{P}i\right) \\ y_0 + R\sin\left(\frac{2\pi}{P}i\right) \end{bmatrix} \tag{8.18}$$

As in the Hough transform for circles (Section 5.5.3), Bresenham's algorithm offers a more efficacious method for generating circle. As we can now have a different number of points within the code, the code generation for a scale invariant LBP LBP_S is

$$LBP\_S(P,R) = \sum_{i \in 1,P} s(x(i))2^{i-1} \tag{8.19}$$

The patterns for radial sampling for different values of P and R are shown in Figure 8.7, where Figure 8.7(a) is the sampling for a circle with 8 points radius 1 and is equivalent to the earlier 3×3 patch in Figure 8.5(a), Figure 8.7(b) is for a radius 2 also with 8 points, and Figure 8.7(c) is yet larger. All show the effect of discretization on low-resolution generation of circular patterns and it is more

$\mathbf{c} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix}$	$\mathbf{c} = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}$	$\mathbf{c} = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \end{pmatrix}$
(a) $\mathbf{c}$:= Circle (8,1)	(b) $\mathbf{c}$:= Circle (8,2)	(c) $\mathbf{c}$:= Circle (16,3)

FIGURE 8.7

Sampling in a radial pattern for (P, R).

usual to use **interpolation** to determine point values rather than the nearest pixel's value.

The rotation-invariant arrangement then shifts the derived code so as to achieve a minimum integer, as in the rotation-invariant chain code Section 7.2.2 (except the LBP is a pattern of 1s and 0s, not integers), and the rotation invariant LBP *LBP_R* is then

$$LBP\_R(P,R) = \min\left\{ \mathrm{ROR}\left(\sum_{i \in 1,P} s(x(i))2^{i-1} \right) \right\} \tag{8.20}$$

where ROR( ) is the (circular) rotation operator. For the sampling arrangement of *LBP_S*(8,1) (Figure 8.7(a)), the approach was found to determine 36 individual patterns for which the occurrence frequencies varied greatly and, given the coarse angular measure used in that arrangement, the technique was found to lack discriminatory ability and so a more potent approach was required.

In order to achieve better discriminatory ability, it was noted that some basic patterns dominated discriminatory ability and the occurrence of these patterns dominates texture description capability (Ojala et al., 2002). For the sampling arrangement of *LBP_S*(8,1) (Figure 8.7(a)) and denoting the output of thresholding relative to the central point as black ("0") or white ("1"), then we achieve the arrangements given in Figure 8.8. In these, patterns 0–8 correspond to basic features: pattern 0 represents the thresholding for a bright spot (all surrounding points have a lower value) and pattern 8 represents a dark spot (all points are brighter). Patterns 1–7 represent lines of varying degrees of curvature: pattern 1 represents the end of a line (a termination), pattern 2 a sharp point, and pattern 4 represents an edge. These are called the *uniform binary patterns* and are characterized by having at most two transitions of "1–0" (or vice versa) when progressing around the circular pattern. The remaining patterns (those which have no label) have more than two transitions of "1–0" in a circular progression and are called **nonuniform**. There are more nonuniform patterns available than those

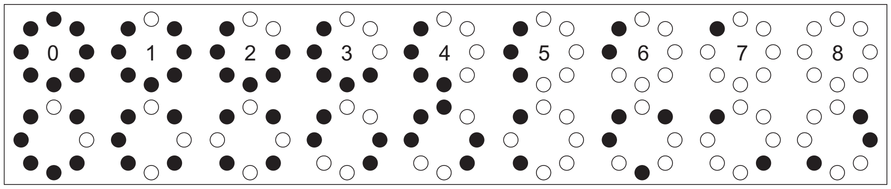


FIGURE 8.8

Rotation invariant binary patterns for *LBP_S*(8,1).

shown in the second row of Figure 8.8. These patterns can occur at any rotation, so the LBP can be arranged to detect them in a rotation-invariant manner.

For the *uniform* LBP approach, first we need to detect whether the patterns are uniform or not, and this is achieved using an operator U which counts the number of transitions of "1–0" (or vice versa):

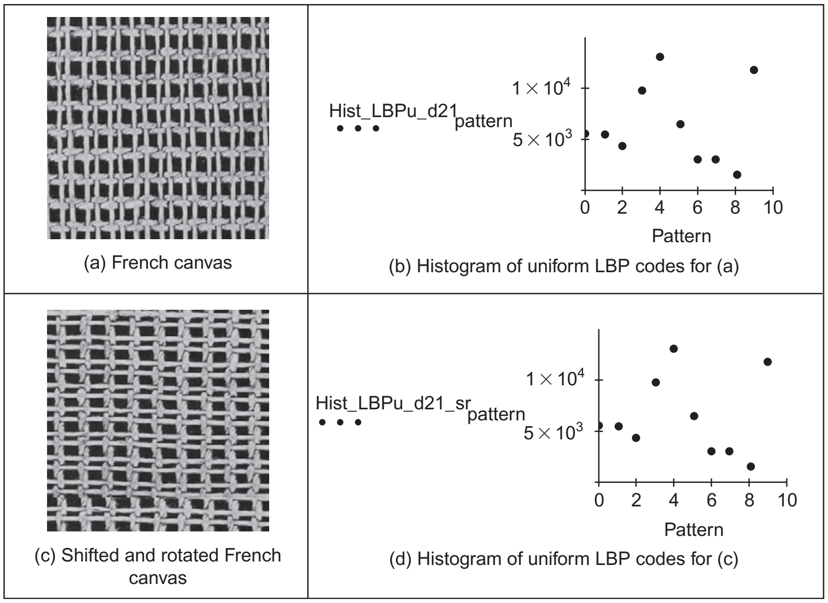
$$U(LBP\_S(P,R)) = |s(x(0)) - s(x(P))| + \sum_{i \in 1,P-1} |s(x(i)) - s(x(i+1))| \qquad (8.21)$$

For the arrangement LBP_S(8,1), the patterns 0–8 thus have a maximum value of $U = 2$ ($U = 0$ for patterns 0 and 8 and for all others $U = 2$). We then need to determine a code for each of the uniform patterns. Since these are rotation invariant, this can be achieved simply by counting the number of bits that are set in the pattern. These are only counted for the uniform patterns; the nonuniform patterns are all set to the same code value (the easiest value is to exceed by one the number of patterns to be expected for that sampling arrangement). For the patterns in Figure 8.8 which are for the sampling arrangement LBP_S(8,1), there are codes 0–8 so we can lump together the nonuniform patterns to a code value of 9. For N patterns ranging from 0 to $N-1$, we can then define the rotation-invariant code as

$$LBP\_U(P,R) = \begin{vmatrix} \sum_{i \in 1,P} s(x(i)), & \text{if } U < 3 \\ N, & \text{otherwise} \end{vmatrix} \qquad (8.22)$$

We then derive a histogram of occurrence of these basic features and we describe a texture by the frequency of occurrence of local basic structures, and this has proved to be a very popular way for describing texture.

Applying the uniform LBP description to the earlier texture D20 and its shifted version, we again achieve a similar histogram of code values. In this histogram, the most popular codes are those for the line structures, which is entirely consistent with the image from which the histogram was derived. Note also that the codes 0–8 dominate the representation, as expected. The results for a shifted and rotated version of texture D20 are shown in Figure 8.9. Here, we see in Figure 8.9(a) the original texture and Figure 8.9(c) its shifted and rotated version. The description of the original texture is given in Figure 8.9(b) and its shifted and rotated version is in Figure 8.9(d). Visually there is little difference between the



(a) French canvas

(b) Histogram of uniform LBP codes for (a)

(c) Shifted and rotated French canvas

(d) Histogram of uniform LBP codes for (c)

FIGURE 8.9

Uniform local binary pattern histograms.

histograms in Figure 8.9(b) and (d) but there is actually some slight difference, of less than 100 count values, which is considerably smaller than the count values (10^4) exposed by the uniform LBP technique.

To apply the technique over different scales, the histograms obtained at each scale can be concatenated. The LBP becomes multiscaled since it can classify any texture pattern which is repeated within one of the neighborhoods. Two-dimensional similarity metrics are used to classify textures using this method because each texture class has a histogram for each scale. The preferred measure for the dissimilarity L, between the concatenated histograms of a sample S and a model M, is:

$$L(S,M) = -\sum_{h=1}^{H}\sum_{n=1}^{N_h} \frac{T_{hs}S_{hn}}{\sum_h T_{hs}} \ln \frac{T_{hm}M_{hn}}{\sum_h T_{hm}} \tag{8.23}$$

where S_{hn} and M_{hn} are the probabilities of the nth bin in the hth sample and model histogram, respectively; N_h is the number of patterns in the H histograms; and T_{hs} and T_{hm} are the total number of entries in the sample and model histograms, respectively.

The original presentation of the uniform LBP technique (Ojala et al., 2002) contains an extensive experimental evaluation on texture databases (including Brodatz), considers more of the ramifications of the representation and implementation of the uniform LBP approach, and contains links to (Matlab) code and data. There have been many extensions and applications of the local binary pattern technique. One particular use is in biometrics, face detection, and recognition (e.g., Ahonen et al., 2006). There is also a book by its originators (Pietikäinen et al., 2011) giving more complete treatment of the LBP approach and describing many of the variants now available.

8.3.6 Other approaches

There have been many approaches to texture and we have so far concentrated on the themes of approaches. Of other themes, one early approach was to develop (Gaussian) *random field models* (Cross and Jain, 1983; Chellappa and Chatterjee, 1985). In these, the approach was to determine a model to describe texture and to use the model parameters for classification—thus affording a way of texture synthesis. The approach was theoretically elegant but computationally unattractive. Julesz introduced the term *texton* for elementary units of texture perception (Julesz, 1981), analogous to a phoneme in speech recognition, and this was used to determine an operational definition of textons and an algorithm for partitioning the image into disjoint regions of coherent brightness and texture (Malik et al., 2001). More recently, techniques have been extended to *dynamic textures* (Szummer and Picard, 1999) which are those textures which exhibit temporal motion such as "wavy water, rising steam, and fire" like the random field models; these were described using a statistical model allowing for recognition and for synthesis, with impressive results. It is a large field though and there are many derivatives of these themes and combinations. These are ably described elsewhere (Petrou and Sevilla, 2006).

8.4 Classification

8.4.1 Distance measures

In application, usually we have a description of a texture **sample** and we want to find which element of a database best matches that sample. This is *classified* as to associate the appropriate *class label* (type of texture) with the test sample by using the measurements that describe it. One way to make the association is by finding the member of the class (the sample of a known texture) with measurements which differ by the least amount from the test sample's measurements. As such, we assign class by analyzing distance.

There is a selection of formulae to measure distance. This is reflected by the aphorism "how long is a piece of string" which in colloquial English is used to reflect uncertainty. It is an appropriate phrase here, since the measure of distance depends on what we want to achieve, and how we want to achieve it: there are many ways to measure the length of a piece of string. The first measure is the length between the endpoints, here equivalent to measuring the distance between two points using a ruler. This is the *Euclidean distance*, and for sets of points, the

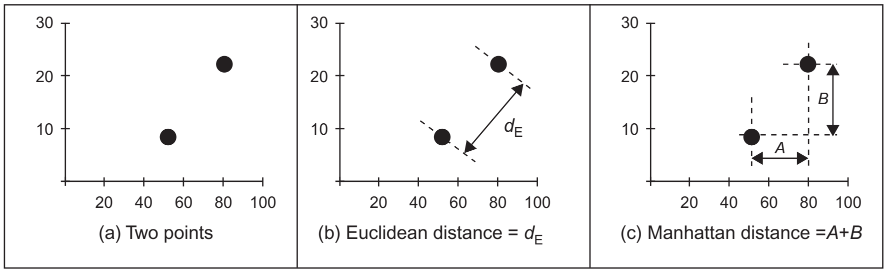


FIGURE 8.10

Distance measures in a 2D plane.

difference d between the M descriptions of a sample, $\mathbf{s}$, and the description of a known texture, $\mathbf{k}$, is

$$d_E = \sqrt{\sum_{i=1}^{M} (\mathbf{s}_i - \mathbf{k}_i)^2} \tag{8.24}$$

which is also called the L_2 *norm* (or L_2 *distance*). This measures the length of a straight line between two points as shown in Figure 8.10. In a 2D plane, for a point $\mathbf{p}1 = (\mathbf{p}1_x, \mathbf{p}1_y)$ and another point $\mathbf{p}2 = (\mathbf{p}2_x, \mathbf{p}2_y)$,

$$d_E = \sqrt{(\mathbf{p}1_x - \mathbf{p}2_x)^2 + (\mathbf{p}1_y - \mathbf{p}2_y)^2} \tag{8.25}$$

The distance measured rather depends on the nature of the property being measured, so there are alternative distance metrics. These include the L_1 *norm* which is the sum of the modulus of the differences between the measurements:

$$d_M = \sum_{i=1}^{M} |\mathbf{s}_i - \mathbf{k}_i| \tag{8.26}$$

This is also called the *Manhattan* distance or *taxicab* measure, by virtue of the analogy to distance in an urban neighborhood. There, the distance traveled is along the streets since you cannot go through buildings. In a rectilinear urban system (i.e., an arrangement of perpendicular streets), it doesn't matter which path you take (except in New York where you can take Broadway which cuts across on a diagonal across the north–south and east–west street systems, and as such is like the Euclidean distance). So for the points $\mathbf{p}1$ and $\mathbf{p}2$ in a 2D plane, the Manhattan distance is the distance along the x axis, added to the distance along the y axis, as shown in Figure 8.10. (It is also called the L_1 *distance* or L_1 *norm*.)

$$d_M = |\mathbf{p}1_x - \mathbf{p}2_x| + |\mathbf{p}1_y - \mathbf{p}2_y| \tag{8.27}$$

These distance measures are illustrated for two points in a 2D plane in Figure 8.10. This is rather difficult to visualize for multidimensional data, so for the multidimensional case illustrated in Code 8.5, the Euclidean distance is measured in a 4D space.

```
Point 1:pv1 := (52.0 8.75 1.50 0.40)

Point 2:pv2 := (81 22.14 1.00 0.37)

Distance measure:
    dvE(vector1, vector2) := $\sqrt{\sum_{i=0}^{cols(vector1)-1} (vector1_{0,i} - vector2_{0,i})^2}$

Euclidean distance: dvE(pv1,pv2) = 31.946
```

CODE 8.5

Illustrating the multidimensional Euclidean distance.

For **groups** of points, we need to be able to handle sets of measurements. These can be derived, say, by applying the same texture description process to a different image of the same cloth. We then have multiple vectors of measurements, which we shall here store as a matrix, as given in Code 8.6. In this, there are two sets of measured feature vectors, `mat1` and `mat2`. Here two of the measures for `mat2` appear different from those for `mat1` so these sets of measurements could be those taken from images of different textures (different classes of texture). These both have five sets of four different measurements. The Euclidean distance can then be the average distance over the sets of measurements. So we work out the Euclidean distance for each row and then average over these five values.

```
Data:
    mat1 := $\begin{pmatrix} 52 & 8.75 & 1.5 & 0.4 \\ 51.9 & 8.8 & 1.5 & 0.39 \\ 52.2 & 8.75 & 1.49 & 0.41 \\ 52 & 8.76 & 1.51 & 0.32 \\ 51.5 & 8.82 & 1.46 & 0.4 \end{pmatrix}$
    mat2 := $\begin{pmatrix} 81 & 22.14 & 1.0 & 0.37 \\ 80.5 & 22.28 & 1.1 & 0.39 \\ 81.5 & 22.16 & 1.05 & 0.41 \\ 80.9 & 22.18 & 1.11 & 0.32 \\ 81 & 22.12 & 1.16 & 0.4 \end{pmatrix}$

Operator:
    averaged_dvE(m1,m2) := | distance ← 0
                           | for j∈0..rows(m1)-1
                           |     distance ← distance + $\sqrt{\sum_{i=0}^{cols(m1)-1} (m1_{j,i} - m2_{j,i})^2}$
                           | $\frac{distance}{rows(m1)}$

Result: averaged_dvE(mat1,mat2) = 32.004
```

CODE 8.6

Illustrating the Euclidean distance for groups of points.

Naturally, the Euclidean distance between one set of points and itself is zero. The distance between set 1 and set 2 should be the same as the distance between set 2 and set 1. This is summarized as a *confusion matrix*, which shows the difference between the different sets of measurements. This is shown in Code 8.7 and the confusion matrix has zeros on the leading diagonal and is symmetric, as expected.

```
Construction:
             confusion (m1,m2):= ( averaged_dvE(m1,m1) averaged_dvE(m1,m2) )
                                 ( averaged_dvE(m2,m1) averaged_dvE(m2,m2) )

Result:
    confusion (mat1,mat2)= (   0    32.004 )
                           ( 32.004   0    )
```

CODE 8.7

Construction of a confusion matrix for two classes.

Another way to estimate the distance is by using a matrix formulation. A more esoteric formulation is to use matrix norms. The advantages here are that there can be fast algorithms available for matrix computation. Another way to measure the distance would be to measure the distance between the means (the centers) of the clusters. As with the matrix formulations, that is a rather incomplete measure since it does not include the cluster spread. To obtain a distance measure between clusters, where the measure reflects not only the cluster spacing but also the cluster spread, we can use the *Mahalanobis distance* measure. This is shown in Figure 8.11, where we have two sets of measures (points denoted by +) of vectors $\mathbf{p}_1$ and $\mathbf{p}_2$. There are two cases of the second cluster, one which is tightly clustered (low variance), indicated by the solid line, and one which is spread out (high variance), indicated by the dotted line (the data points are omitted for the large variance case). If the variance in the second case was sufficiently high, the cluster for $\mathbf{p}_1$ would intersect with the cluster for $\mathbf{p}_2$ and so there would appear to be little difference between them (implying that the classes are the same). The Euclidean

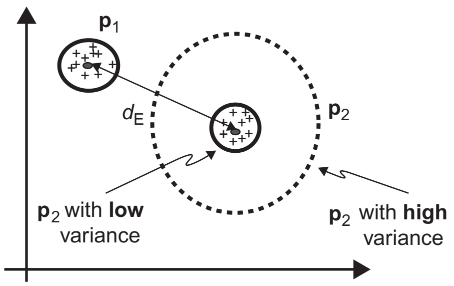


FIGURE 8.11

Illustrating the Mahalanobis distance measure.

distance between the means will remain the same whatever the cluster spread, and so is not affected by this; conversely, the Euclidean distance only measures where the center of mass is, and not the spread. The Mahalanobis distance includes the variance and so is a more perceptive measure of distance. The Mahalanobis distance between $\mathbf{p}_1$ and $\mathbf{p}_2$ would in this case chance with the variance of $\mathbf{p}_2$.

For sets of vector points $\mathbf{p}_i = (\mathbf{m}1_i, \mathbf{m}2_i, \mathbf{m}3_i, \ldots, \mathbf{m}N_i)^{\mathrm{T}}$ which have mean values $\boldsymbol{\mu} = (\mu 1, \mu 2, \mu 3, \ldots, \mu N)^{\mathrm{T}}$ and covariance matrix Σ, the Mahalanobis distance is defined as

$$d_{\mathrm{MAH}} = \sqrt{(\mathbf{p}-\boldsymbol{\mu})^{\mathrm{T}} \sum{}^{-1} (\mathbf{p}-\boldsymbol{\mu})} \tag{8.28}$$

where the covariance matrix is formed of elements which express the variance as

$$\sum\nolimits_{ij} = \mathrm{E}[(\mathbf{p}i - \boldsymbol{\mu}i)(\mathbf{p}j - \boldsymbol{\mu}j)] \tag{8.29}$$

where E denotes the expected value. Since our main consideration is the distance between two sets of feature vectors, we shall formulate this measure for feature vectors which are stored as rows (each sample is a row vector, each column contains the value of a particular measurement)

$$d_{\mathrm{MAH}_{i,j}} = \sqrt{(\mathbf{p}_i - \mathbf{p}_j)\mathbf{P}^{-1}(\mathbf{p}_i - \mathbf{p}_j)^{\mathrm{T}}} \tag{8.30}$$

where the covariance matrix $\mathbf{P}$ is

$$\mathbf{P} = \frac{\mathbf{P}_i + \mathbf{P}_j}{2} \tag{8.31}$$

where the individual covariance matrices $\mathbf{P}_i$ are

$$\mathbf{P}_i = \frac{1}{N}(\mathbf{p}_i - \boldsymbol{\mu}_i)^{\mathrm{T}}(\mathbf{p}_i - \boldsymbol{\mu}_i) \tag{8.32}$$

In this way, the distance is scaled by the variance. So the distance measure reflects the distributions of the data, which is ignored in the Euclidean distance formulation. The formulation does rather depend on the structure used for the data. The formulation here is consistent with the feature vectors delivered by texture extraction approaches (it can also be the same for feature vectors delivered by other applications, such as biometrics—that is how general it is).

The calculation of the covariance matrix is illustrated for two classes (two sets of feature vectors) in Code 8.8. Here, routine `get_mean` calculates the average of each column and supplies this as a row vector of four values. The mean for each column is then subtracted from each column element in routine `subtract_mean`. The routine `cov` implements Eq. (8.31); the routine `covariance` implements Eq. (8.32). The final row shows the calculated covariance matrix for the two sets of measurements (`mat1` and `mat2`) given earlier. The diagonal elements reflect the variance in each measure; the off-diagonal measurements reflect the correlation between the different measurements. The properties of this matrix are symmetric and positive definite. When assessing implementation, test for symmetry when developing code check by subtracting the transpose (and the result should be zero); positive semi-definiteness can be assessed by an eigenvector calculation, but it is simpler to submit an example to a mathematical package and check that Cholesky decomposition can be performed on it (if it cannot, then the matrix is not positive definite).

```
Find mean:
 get_mean(matrix):= | for i∈0,1..cols(matrix)−1
                    |   $\text{meanv}_{0,i} \leftarrow \dfrac{\sum_{j=0}^{\text{rows(matrix)}-1} \text{matrix}_{j,i}}{\text{rows(matrix)}}$
                    | meanv
Remove mean:
 subtract_mean(matrix):= | means ← get_mean(matrix)
                         | for i∈0..cols(matrix)−1
                         |   for j∈0..rows(matrix)−1
                         |     $\text{matrix}_{j,i} \leftarrow \text{matrix}_{j,i} - \text{means}_{0,i}$
                         | matrix
Evaluate covariance:
 $\text{covariance(matrix)} := \dfrac{1}{\text{rows(matrix)}} \cdot \left(\text{subtract\_mean(matrix)}^T \cdot \text{subtract\_mean(matrix)}\right)$
Evaluate mean covariance:
 $\text{cov(m1,m2)} := \dfrac{\text{covariance(m1)} + \text{covariance(m2)}}{2}$
Result:
```

$$\text{cov(mat1, mat2)} = \begin{pmatrix} 0.078 & -9\times10^{-3} & -1.28\times10^{-3} & -1.24\times10^{-3} \\ -9\times10^{-3} & 1.964\times10^{-3} & -5.8\times10^{-5} & 3.4\times10^{-5} \\ -1.28\times10^{-3} & -5.8\times10^{-5} & 1.64\times10^{-3} & -1.6\times10^{-4} \\ 1.24\times10^{-3} & 3.4\times10^{-5} & -1.64\times10^{-4} & 1.04\times10^{-3} \end{pmatrix}$$

CODE 8.8

Constructing the covariance matrix

Determining the Mahalanobis distance is illustrated in Code 8.9 for the two classes mat1 and mat2. This is slightly different from Eq. (8.30). This is because Eq. (8.30) is the conventional expression. Here, we are using vectors of measurements and we have multiple samples of these measurements. As such, the distance measure we are interested in is the sum of the values on the leading diagonal (the *trace* of the matrix, hence the use of the operator tr). The implementation is the same in all other respects.

Data: $\mathtt{diff(m1,m2)} := \mathtt{m1-m2}$

Operator: $$\mathtt{dMaha(m1,m2)} := \frac{\sqrt{\mathtt{tr}\left[\mathtt{diff(m1,m2)}\cdot(\mathtt{cov(m1,m2)})^{-1}\cdot\mathtt{diff(m1,m2)}^{T}\right]}}{\mathtt{rows(m1)}}$$

Result: $\mathtt{dMaha(mat1,mat2)} = 260.059$

CODE 8.9

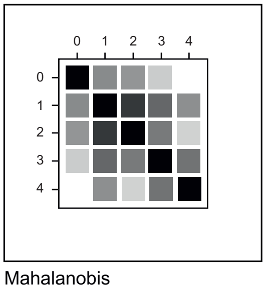
The Mahalanobis distance between two classes.

The confusion matrix derived by the Mahalanobis distance is little different from that derived by Euclidean distance, the change is that the values on the off-diagonal will be larger. To assess its effect, we need multiple samples of multiple classes and we need to evaluate the confusion matrix over these. This is shown in Code 8.10. Here we have five classes: `mat1`, `mat2`, `mat3`, `mat4`, and `mat5`. The confusion matrices are now 5×5, showing the confusion between every class and the other four. The leading diagonal is still zero, since each class is the same as itself. The off-diagonal elements show the difference between the different classes. As the distance becomes larger, the cell becomes brighter. The confusion matrix by the Euclidean distance has more difficulty distinguishing between classes 3, 4, and 5 than that for the Mahalanobis distance, since the lower right-hand corner is largely dark, whereas in the Mahalanobis distance only the leading diagonal is dark. This is because the Mahalanobis distance uses the structure of the data (its variance) to determine distance.

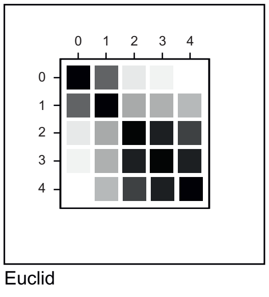
$$\mathtt{mat1} := \begin{pmatrix} 52 & 8.75 & 1.5 & 0.4 \\ 51.9 & 8.8 & 1.5 & 0.39 \\ 52.2 & 8.75 & 1.49 & 0.41 \\ 52 & 8.76 & 1.51 & 0.32 \\ 51.5 & 8.82 & 1.46 & 0.4 \end{pmatrix} \quad \mathtt{mat2} := \begin{pmatrix} 81 & 22.14 & 1.0 & 0.37 \\ 80.5 & 22.28 & 1.1 & 0.39 \\ 81.5 & 22.16 & 1.05 & 0.41 \\ 80.9 & 22.18 & 1.11 & 0.32 \\ 81 & 22.12 & 1.16 & 0.4 \end{pmatrix} \quad \mathtt{mat3} := \begin{pmatrix} 89 & 32.14 & 1.0 & 0.37 \\ 70.5 & 32.28 & 1.1 & 0.39 \\ 101.5 & 32.16 & 1.05 & 0.41 \\ 180.9 & 32.18 & 1.11 & 0.32 \\ 161 & 32.12 & 1.16 & 0.4 \end{pmatrix}$$

$$\mathtt{mat4} := \begin{pmatrix} 89 & 42.14 & 1.0 & 0.37 \\ 70.5 & 42.28 & 1.1 & 0.39 \\ 101.5 & 42.16 & 1.05 & 0.41 \\ 180.9 & 42.18 & 1.11 & 0.32 \\ 161 & 42.12 & 1.16 & 0.4 \end{pmatrix} \quad \mathtt{mat5} := \begin{pmatrix} 89 & 52.13 & 1.0 & 0.37 \\ 70.5 & 52.28 & 1.1 & 0.39 \\ 101.5 & 52.16 & 1.05 & 0.41 \\ 180.9 & 52.18 & 1.11 & 0.32 \\ 161 & 52.12 & 1.16 & 0.4 \end{pmatrix}$$

Data samples



(a) By Mahalanobis distance



(b) By Euclidean distance

CODE 8.10

Confusion matrices by different distance measures.

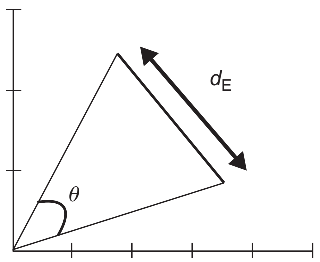


FIGURE 8.12

Cosine and Euclidean distance measures.

There are many other distance measures. The *Bhattacharyya distance*

$$d_B = -\ln \sum_{i=1}^{M} \sqrt{\mathbf{s}_i \times \mathbf{k}_i} \tag{8.33}$$

gives smaller precedence to larger distances by using the logarithm to compress them, but this appears to be used less, like other metrics such as the *Matusita difference*. There is a measure which emphasizes **direction** rather than **distance**. This is the *cosine distance* measure

$$d_C = \cos(\theta) = \frac{\mathbf{p}1 \cdot \mathbf{p}2}{|\mathbf{p}1||\mathbf{p}2|} \tag{8.34}$$

where "$\cdot$" represents the scalar product of the two vectors **p**1 and **p**2 (the inner product) and | | denotes the length of each vector. The angle θ is that between the two vectors to the points, as shown in Figure 8.12. Note that in this case, **similarity** is when $\theta \to 0$ (so $d_C \to 1$) which is different for the other cases wherein similarity is reflected by a minimum of the distance measure. Essentially, the measure of distance depends on the technique used to make a measurement and the nature of the data itself. As ever, try a selection and use the one best suited to a particular application.

8.4.2 The *k*-nearest neighbor rule

We then need to use the distance measure to determine the class to be associated with the data points. If we have M measurements of N known samples of textures and we have O samples of each, we have an M-dimensional *feature space* that contains the $N \times O$ points. If we select the point, in the feature space, which is closest to the current sample, then we have selected the sample's *nearest neighbor*. This is shown in Figure 8.13, where we have a 2D feature space produced by the two measures made on each sample, measure 1 and measure 2. Each sample gives different values for these measures but the samples of different classes give rise to clusters in the feature space where each cluster is associated with a single class. In Figure 8.13, we have seven samples of two known textures: Class A and

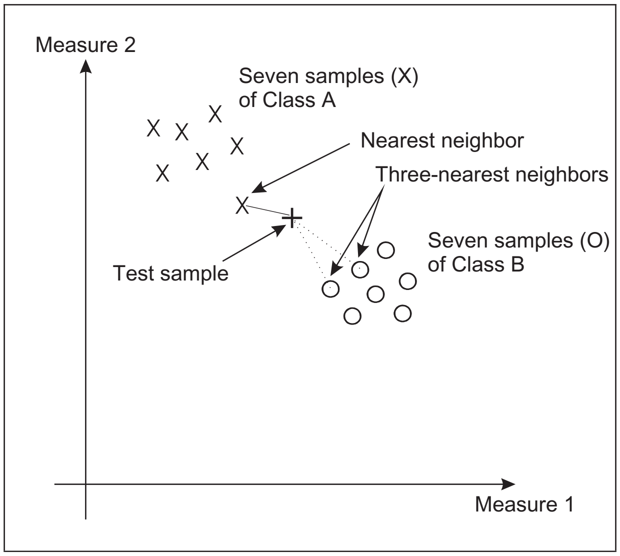


FIGURE 8.13

Feature space and classification.

Class B depicted by X and O, respectively. We want to classify a test sample, depicted by +, as belonging either to Class A or to Class B (i.e., we assume that the training data contains representatives of all possible classes). Its nearest neighbor, the sample with least distance, is one of the samples of Class A, so we could then say that our test appears to be another sample of Class A (i.e., the class label associated with it is Class A). Clearly, the clusters will be far apart for measures that have good discriminatory ability, whereas the clusters will be overlapped for measures that have poor discriminatory ability, i.e., how we can choose measures for particular tasks. Before that, let us look at how best to associate a class label with our test sample.

Classifying a test sample as the training sample it's closest to in feature space is actually a specific case of a general classification rule known as the *k-nearest neighbor rule*. In this rule, the class selected is the **mode** of the sample's nearest k neighbors. By the k-nearest neighbor rule, for $k = 3$, we select the nearest three neighbors (those three with the least distance) and their mode, the maximally represented **class**, is attributed to the sample. In Figure 8.13, the three-nearest neighbor is actually Class B since the three nearest samples contain one from Class A (its nearest neighbor) and two from Class B. Since there are two elements of Class B, the sample is attributed to this class by the three-nearest neighbor rule. As such, selection from more than one point introduces a form of feature space smoothing and allows the classification decision not to be affected by noisy *outlier* points. Clearly, this smoothing has greater effect for larger values of k.

(Further details concerning a modern view of the *k*-nearest neighbor rule can be found in Michie et al. (1994).)

A Mathcad implementation of the *k*-nearest neighbor rule is given in Code 8.11. The arguments are `test` (the vector of measurements of the test sample), `data` (the list of vectors of measurements of all samples), `size` (the value of *k*), and `no`. The final parameter `no` dictates the structure of the presented data and is the number of classes within that data. The training data is presumed to have been arranged so that samples of each class are all stored together. For two classes in the training data, `no` = 2, where each occupies one half (the same situation as in Figure 8.13). If `no` = 3, then there are three classes, each occupying one-third of the complete dataset and the first third contains the first class, the second third contains samples of another class, while the remaining third contains samples of the final class. In application, first the distances between the current sample, `test`, and all other samples are evaluated by using the function `distance`. Then, the *k*-nearest neighbors are selected to form a vector of distances `min`, these are the *k* neighbors which are closest (in the feature space) to the sample test. The number of feature space splits `fsp` is the spacing between the classes in the `data`. The class that occurs the most number of times in the set of `size`-nearest neighbors is then returned as the *k*-nearest neighbor, by incrementing the class number to which each of the *k* neighbors is associated. (If no such decision is possible, i.e., there is no maximally represented class, the technique can be arranged to return the class of the nearest neighbor, by default. An alternative way of stating this is that the default class for *k*-NN is the 1-NN when all *k* classes are different.)

```
k_nn(test,data,size,no):=
        for i∈0..rows(data)-1
            dist_i←0
            for j∈0..cols(data)-1
              dist_i←distance(test,data,i)
        for i∈0..size-1
            posmin←coord(min(dist),dist)
            dist_posmin←max(dist)+1
            min_i←posmin
              rows(data)
        fsp←  ----------
                 no
        for j∈1..no
          class_j←0
        for i∈0..size-1
          for j∈1..no
            class_j←class_j+1 if [min_i≥(j-1)·fsp]·(min_i<j·fsp)
        test_class←coord(max(class),class)
        test_class
```

CODE 8.11

Implementing the *k*-nearest neighbor rule.

The result of testing the *k*-nearest neighbor routine is illustrated on synthetic data in Code 8.12. Here there are two different datasets. The first (Code 8.12(a)) has three classes of which there are three samples (each sample is a row of data, so this totals nine rows) and each sample is made up of three measurements (the three columns). As this is synthetic data, it can be seen that each class is quite distinct: the first class is for measurements and is based on the studies of Ahonen et al. (2006), Bishop (1996), and Bovik et al. (1990); the second class is based on Brodatz (1968), Chen et al. (1995), and Cross and Jain (1983); and the third is based on Cross and Jain (1983), Chen et al. (1995), and Bovik et al. (1990). A small amount of noise has been added to the measurements. We then want to see the class associated with a test sample with measurements (Brodatz, 1968; Cross and Jain, 1983; Chen et al., 1995) (Code 8.12(b)). Naturally, the one-nearest neighbor (Code 8.12(c)) associates it with the class with the closest measurements which is Class 2 as the test sample's nearest neighbor is the fourth row of data. (The result is either Class 1, Class 2, or Class 3.) The three-nearest neighbor (Code 8.12(d)) is again Class 2 as the nearest three neighbors are the fourth, fifth, and sixth rows and each of these is from Class 2.

$\text{population1:=}\begin{bmatrix} 1 & 2 & 3 \\ 1.1 & 2 & 3.1 \\ 1 & 2.1 & 3 \\ 4 & 6 & 8 \\ 3.9 & 6.1 & 8.1 \\ 4.1 & 5.9 & 8.2 \\ 8.8 & 6.1 & 2.8 \\ 7.8 & 5.9 & 3.3 \\ 8.8 & 6.4 & 3.1 \end{bmatrix}$	$\text{population2:=}\begin{bmatrix} 2 & 4 & 6 & 8 \\ 2.1 & 3.9 & 6.2 & 7.8 \\ 2.3 & 3.6 & 5.8 & 8.3 \\ 2.5 & 4.5 & 6.5 & 8.5 \\ 3.4 & 4.4 & 6.6 & 8.6 \\ 2.3 & 4.6 & 6.4 & 8.5 \end{bmatrix}$
(a) 3 classes, 3 samples, 3 features	(e) 2 classes, 3 samples, 4 features
`test_point1:=(4 6 8)`	`test_point2:=(2.5 3.8 6.4 8.3)`
(b) First test sample	(f) Second test sample
`k_nn(test_point1,population1,1,3)=2`	`k_nn(test_point2,population2,1,2)=1`
(c) 1-nearest neighbor	(g) 1-nearest neighbor
`k_nn(test_point1,population1,3,3)=2`	`k_nn(test_point2,population2,3,2)=2`
(d) 3-nearest neighbor	(h) 3-nearest neighbor

CODE 8.12

Applying the *k*-nearest neighbor rule to synthetic data.

The second dataset (Code 8.12(e)) is two classes with three samples each made up of four measures. The test sample (Code 8.12(f)) is actually associated with Class 1 by the one-nearest neighbor (Code 8.12(g)) but with Class 2 for the three-nearest neighbor (Code 8.12(h)). This is because the test sample is actually closest to the sample in the third row. After the third row, the next two closest samples are in the fourth and sixth rows. As the nearest neighbor is in a different class (Class 1) to that of the next two nearest neighbors (Class 2); a different result has occurred when there is more **smoothing** in the feature space (when the value of k is increased).

The Brodatz database actually contains 112 textures, but few descriptions have been evaluated on the whole database, usually concentrating on a subset. It has been shown that the SGF description can afford better classification capability than the cooccurrence matrix and the Fourier transform features (described by Liu's features) (Chen et al., 1995). For experimental procedure, the Brodatz pictures were scanned into 256×256 images which were split into 16 64×64 subimages. Nine of the subimages were selected at random and results were classified using *leave-one-out cross-validation* (Lachenbruch and Mickey, 1968). **Leave-one-out** refers to a procedure where one of the samples is selected as the test sample, the others form the training data (this is the leave-one-out rule). **Cross validation** is where the test is repeated for all samples: each sample becomes the test data once. In the comparison, the eight optimal Fourier transform features were used (Liu and Jernigan, 1990), and the five most popular measures from the cooccurrence matrix. The correct classification rate, the number of samples attributed to the correct class, showed better performance by the combination of statistical and geometric features (86%), as opposed to use of single measures. The enduring capability of the cooccurrence approach was reflected by their (65%) performance in comparison with Fourier (33%—whose poor performance is rather surprising). An independent study (Walker and Jackway, 1996) has confirmed the experimental advantage of SGF over the cooccurrence matrix, based on a (larger) database of 117 cervical cell specimen images. Another study (Ohanian and Dubes, 1992) concerned the features which optimized the classification rate and compared cooccurrence, fractal-based, Markov Random field, and Gabor-derived features. By analysis on synthetic and real imagery, via the k-nearest neighbor rule, the results suggested that cooccurrence offered the best overall performance. Wavelets (Porter and Canagarajah, 1997), Gabor wavelets, and Gaussian Markov random fields have been compared (on a limited subset of the Brodatz database) to show that the wavelet-based approach had the best overall classification performance (in noise as well) together with the smallest computational demand.

8.4.3 Other classification approaches

Classification is the process by which we attribute a class label to a set of measurements. Essentially, this is the heart of pattern recognition: intuitively, there must be many approaches. These include **statistical** and **structural** approaches: a review can be found in Shalkoff (1992) and a more modern view in Cherkassky and Mulier (1998). There are some newer texts, such as Forsyth and Ponce

(2002), Szeliski (2011), and Prince (2012), which specifically include learning in computer vision and they offer much greater depth than this brief survey.

One major approach is to use a *neural network* which is a common alternative to using a classification rule. Essentially, modern approaches are based on *multi-layer perceptrons* with *artificial neural networks* in which the computing elements aim to mimic properties of neurons in the human brain. These networks require **training**, typically by error back-propagation, aimed to minimize classification error on the training data. At this point, the network should have learnt how to recognize the **test** data (they aim to learn its structure): the output of a neural network can be arranged to be class labels. Approaches using neural nets (Muhamad and Deravi, 1994) show how texture metrics can be used with neural nets as classifiers, another uses cascaded neural nets for texture extraction (Shang and Brown, 1994). Neural networks are within a research field that has shown immense growth in the past two decades, further details may be found in Michie et al. (1994) and Bishop (1996) (often a student's favorite), and more targeted at vision in Zhou and Chellappa (1992). *Support vector machines* (SVMs) (Vapnik, 1995) are one of the most popular approaches to data modeling and classification, more recently subsumed within *Kernel methods* (Shawe-Taylor and Cristianini, 2004). Their advantages include excellent **generalization** capability which concerns the ability to classify correctly samples which are not within feature space used for training. SVMs have naturally found application in texture classification (Kim et al., 2002). Interest in biometrics has focused on combining different classifiers, such as face and speech, and there are promising approaches to accommodate this (Kittler, 1998; Kittler et al., 1998).

Also, there are methods aimed to improve classification capability by pruning the data which does not contribute to the classification decision. Guided ways which investigate the potency of measures for analysis are known as *feature (subset) selection*. *PCA* (Chapter 12, Appendix 3) can reduce dimensionality, orthogonalize, and remove redundant data. There is also *linear discriminant analysis* (also called *canonical analysis*) to improve class separability while concurrently reducing cluster size (it is formulated to concurrently minimize the within-class distance and to maximize the between-class distance). There are also algorithms aimed at choosing a reduced set of features for classification: feature selection for improved discriminatory ability; a comparison can be found in Jain and Zongker (1997). Alternatively, the basis functionals can be chosen in such a way as to improve classification capability.

8.5 Segmentation

In order to *segment* an image according to its texture, we can measure the texture in a chosen region and then classify it. This is equivalent to **template convolution** but where the result applied to pixels is the class to which they belong, as

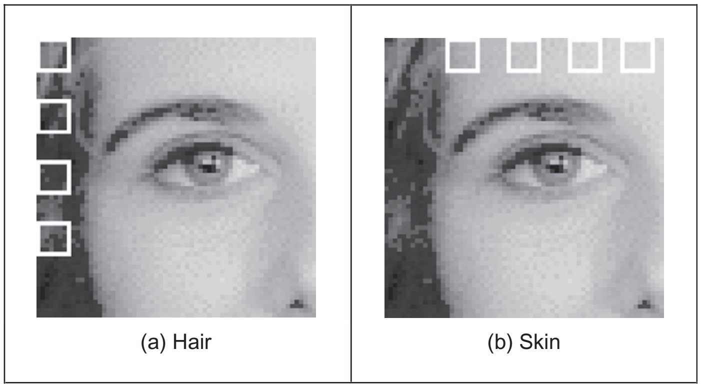


FIGURE 8.14

Training regions for classification.

opposed to the usual result of template convolution. Here, we shall use a 7×7 template size: the texture measures will be derived from the 49 points within the template. First though, we need data from which we can make a classification decision, the training data. Naturally, this depends on a chosen application. Here, we shall consider the problem of segmenting the eye image into regions of **hair** and **skin**.

This is a two-class problem for which we need samples of each class, samples of skin and hair. We will take samples of each of the two classes; the classification decision is as shown in Figure 8.13. The texture measures are the energy, entropy, and inertia of the cooccurrence matrix of the 7×7 region, so the feature space is 3D. The training data is derived from the regions of hair and skin, as shown in Figure 8.14(a) and (b), respectively. The first half of this data is the samples of hair and the other half is samples of the skin, as required for the *k*-nearest neighbor classifier of Code 8.11.

We can then segment the image by classifying each pixel according to the description obtained from its 7×7 region. Clearly, the training samples of each class should be classified correctly. The result is shown in Figure 8.15(a). Here, the top left corner is first (correctly) classified as hair, and the top row of the image is classified as hair until the skin commences (note that the border inherent in template convolution reappears). In fact, much of the image appears to be classified as expected. The eye region is classified as hair, but this is a somewhat arbitrary decision, it is simply that hair is the closest texture feature. Also, some of the darker regions of skin are classified as hair, perhaps the result of training on regions of brighter skin.

Naturally, this is a computationally demanding process. An alternative approach is to simply classify regions as opposed to pixels. This is the tiled approach, with the result shown in Figure 8.15(b). The resolution is clearly very

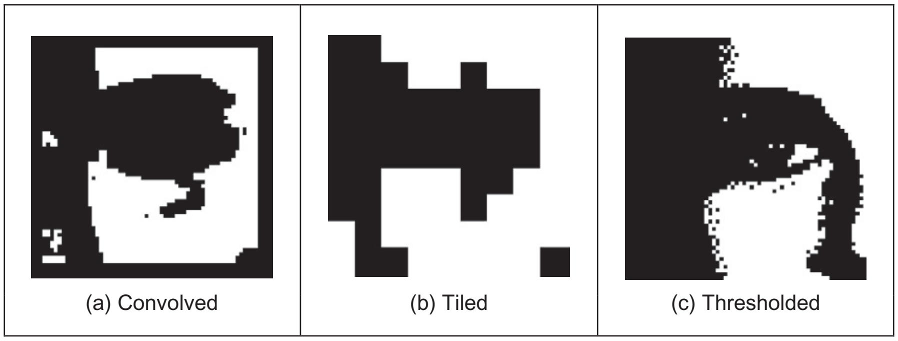


FIGURE 8.15

Segmenting the eye image into two classes.

poor: the image has effectively been reduced to a set of 7×7 regions, but it is much **faster** requiring only 2% of the computation of the convolution approach.

A comparison with the result achieved by uniform thresholding is given, for comparison, in Figure 8.15(c). This is equivalent to pixel segmentation by brightness alone. Clearly, there are no regions where the hair and skin are mixed and in some ways the result appears superior. This is in part due to the simplicity in implementation of texture segmentation. But the result of thresholding depends on **illumination** level and on appropriate choice of the threshold value. The texture segmentation method is completely **automatic** and the measures are known to have **invariance** properties to illumination, as well as other factors. Also, in uniform thresholding there is no extension possible to separate **more** classes (except perhaps to threshold at differing brightness levels).

8.6 Further reading

Clearly, there is much further reading in the area of texture description, classification, and segmentation, as evidenced by the volume of published work in this area. The best place to start for texture is Maria Petrou's book (Petrou and Sevilla, 2006). There is one fairly comprehensive—but dated—survey (Reed and du Buf, 1993). An updated review of Tuceryan and Jain (1998) has a wide bibliography. Zhang and Tan (2002) offer review of the approaches which are invariant to rotation, translation, and to affine or projective transforms, but texture is a large field of work to survey with many applications. Even though it is a large body of work, it is still only a subset of the field of pattern recognition. In fact, reviews of pattern recognition give many pointers to this fascinating and extensive field (e.g., Jain et al., 2000).

There is also a vast body of work on pattern classification, for this is the sphere of machine learning. Beyond the texts already cited here, have a look at the Proceedings of the Neural Information Processing Conference: NIPS. This is the top international conference in machine learning and you will find state-of-the-art papers there. For vision-based learning approaches, many of the conferences listed earlier in Section 1.6.1 will have new papers in these fields.

8.7 References

Ahonen, T., Hadid, A., Pietikäinen, M., 2006. Face description with local binary patterns: application to face recognition. IEEE Trans. PAMI 28 (12), 2037−2041.

Bishop, C.M., 1996. Neural Networks for Pattern Recognition. Oxford University Press, Oxford.

Bovik, A.C., Clark, M., Geisler, W.S., 1990. Multichannel texture analysis using localised spatial filters. IEEE Trans. PAMI 12 (1), 55−73.

Brodatz, P., 1968. Textures: A Photographic Album for Artists and Designers. Reinhold, New York, NY.

Chellappa, R., Chatterjee, S., 1985. Classification of textures using Gaussian Markov random fields. IEEE Trans. ASSP 33 (4), 959−963.

Chen, Y.Q., Nixon, M.S., Thomas, D.W., 1995. Texture classification using statistical geometric features. Pattern Recog. 28 (4), 537−552.

Cherkassky, V., Mulier, F., 1998. Learning from Data. Wiley, New York, NY.

Cross, G.R., Jain, A.K., 1983. Markov random field texture models. IEEE Trans. PAMI 5 (1), 25−39.

Daugman, J., High, G., 1993. Confidence visual recognition of persons using a test of statistical independence. IEEE Trans. PAMI 18 (8), 1148−1161.

Dunn, D., Higgins, W.E., Wakely, J., 1994. Texture segmentation using 2-D Gabor elementary functions. IEEE Trans. PAMI 16 (2), 130−149.

Efros, A., Leung, T., 1999. Texture synthesis by non-parametric sampling. Proc. ICCV, 1033−1038.

Forsyth, D., Ponce, J., 2002. Computer Vision: A Modern Approach. Prentice Hall, Upper Saddle River, NJ.

Gimmel'farb, G.L., Jain, A.K., 1996. On retrieving textured images from an image database. Pattern Recog. 28 (12), 1807−1817.

Grigorescu, S.E., Petkov, N., Kruizinga, P., 2002. Comparison of texture features based on Gabor filters. IEEE Trans. IP 11 (10), 1160−1167.

Haralick, R.M., Shanmugam, K., Dinstein, I., 1973. Textural features for image classification. IEEE Trans. SMC 2, 610−621.

Heckbert, P.S., 1986. Survey of texture mapping. IEEE Comput. Graphics Appl. 6, 56−67.

Jain, A.K., Farrokhnia, F., 1991. Unsupervised texture segmentation using Gabor filters. Pattern Recog. 24 (12), 1186−1191.

Jain, A.K., Zongker, D., 1997. Feature selection: evaluation, application and small sample performance. IEEE Trans. PAMI 19 (2), 153−158.

Jain, A.K., Duin, R.P.W., Mao, J., 2000. Statistical pattern recognition: a review. IEEE Trans. PAMI 22 (1), 4−37.

Julesz, B., 1981. Textons the elements of texture and perception, and their interactions. Nature 290, 91−97.

Karru, K., Jain, A.K., Bolle, R., 1996. Is there any texture in an image? Pattern Recog. 29 (9), 1437−1446.

Kim, K.I., Jung, K., Park, S.H., Kim, H.J., 2002. Support vector machines for texture classification. IEEE Trans. PAMI 24 (11), 1542−1550.

Kittler, J., 1998. Combining classifiers: a theoretical framework. Pattern Anal. Appl. 1 (1), 18−27.

Kittler, J., Hatef, M., Duin, R.P.W., Matas, J., 1998. On combining classifiers. IEEE Trans. PAMI 20 (3), 226−239.

Lachenbruch, P.A., Mickey, M.R., 1968. Estimation of error rates in discriminant analysis. Technometrics 10, 1−11.

Laine, A., Fan, J., 1993. Texture classification via wavelet pattern signatures. IEEE Trans. PAMI 15 (11), 1186−1191.

Liu, S.S., Jernigan, M.E., 1990. Texture analysis and discrimination in additive noise. CVGIP 49, 52−67.

Lu, C.S., Chung, P.C., Chen, C.F., 1997. Unsupervised texture segmentation via wavelet transform. Pattern Recog. 30 (5), 729−742.

Malik, J., Belongie, S., Leung, T., Shi, J., 2001. Textons, contour and texture analysis for image segmentation. Int. J. Comput. Vision 43 (1), 7−27.

Michie, D., Spiegelhalter, D.J., Taylor, C.C. (Eds.), 1994. Machine Learning, Neural and Statistical Classification. Ellis Horwood, Hemel Hempstead.

Muhamad, A.K., Deravi, F., 1994. Neural networks for the classification of image texture. Eng. Appl. Artif. Intell. 7 (4), 381−393.

Ohanian, P.P., Dubes, R.C., 1992. Performance evaluation for four classes of textural features. Pattern Recog. 25 (8), 819−833.

Ojala, T., Pietikäinen, M., Harwood, D., 1996. A comparative study of texture measures with classification based on featured distribution. Pattern Recog. 29 (1), 51−59.

Ojala, T., Pietikäinen, M., Mäenpää, T., 2002. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. IEEE Trans. PAMI 24 (7), 971−987.

Petrou, M., Sevilla, O.G., 2006. *Image Processing: Dealing with Texture*. Wiley.

Pichler, O., Teuner, A., Hosticka, B.J., 1996. A comparison of texture feature extraction using adaptive Gabor filtering, pyramidal and tree structured wavelet transforms. Pattern Recog. 29 (5), 733−742.

Pietikäinen, M., Hadid, A., Zhao, G., Ahonen, T., 2011. *Computer Vision Using Local Binary Patterns*. Springer.

Porter, R., Canagarajah, N., 1997. Robust rotation-invariant texture classification: wavelet, Gabor filter and GRMF based schemes. IEE Proc. Vision Image Signal Process. 144 (3), 180−188.

Pratt, W.K., 1992. *Digital Image Processing*. Wiley.

Prince, S.J.D., 2012. Computer Vision Models, Learning, and Inference. Cambridge University Press, Cambridge.

Randen, T., Husoy, J.H., 2000. Filtering for texture classification: a comparative study. IEEE Trans. PAMI 21 (4), 291−310.

Reed, T.R., du Buf, H., 1993. A review of recent texture segmentation and feature extraction techniques. CVGIP: Image Understand. 57 (3), 359−372.

Shalkoff, R.J., 1992. Pattern Recognition—Statistical. Structural and Neural Approaches. Wiley, New York, NY.

Shang, C.G., Brown, K., 1994. Principal features-based texture classification with *neural networks*. Pattern Recog. 27 (5), 675−687.

Shawe-Taylor, J., Cristianini, N., 2004. *Kernel Methods for Pattern Analysis*. Cambridge University Press.

Szeliski, R., 2011. Computer Vision: Algorithms and Applications. Springer Verlag, London.

Szummer, M., Picard, R.W., 1999. Temporal texture modelling. Proc. ICIP 3, 823−826.

Tuceryan, M., Jain, A.K., 1998. Texture analysis. In: Chen, C.H., Pau, L.F., Wang, P.S.P. (Eds.), The Handbook of Pattern Recognition and Computer Vision, second ed. World Scientific Publishing, Singapore, pp. 207−248.

Vapnik, V., 1995. The Nature of Statistical Learning Theory. Springer-Verlag, New York, NY.

Walker, R.F., Jackway, P.T., 1996. Statistical geometric features—extensions for cytological texture analysis. Proceedings of the Thirteenth ICPR, Vienna, II (Track B), pp. 790−794.

Weska, J.S., Dyer, C.R., Rosenfeld, A., 1976. A comparative study of texture measures for terrain classification. IEEE Trans. SMC 6 (4), 269−285.

Wu, C.M., Chen, Y.C., 1992. Statistical feature matrix for texture analysis. CVGIP: Graphical Models Image Process. 54, 407−419.

Wu, W., Wei, S., 1996. Rotation and gray-scale transform-invariant texture classification using spiral resampling, subband decomposition and hidden Markov model. IEEE Trans. IP 5 (10), 1423−1434.

Zhang, J., Tan, T., 2002. Brief review of invariant texture analysis methods. Pattern Recog. 35, 735−747.

Zhou, Y-T, Chellappa, R., 1992. Artificial Neural Networks for Computer Vision. Springer, New York, NY.