3 Image Processing

3.1 Overview

We shall now start to process digital images. First, we shall describe the brightness variation in an image using its histogram. We shall then look at operations which manipulate the image so as to change the histogram, and at processes that shift and scale the result (making the image brighter or dimmer, in different ways). We shall also consider thresholding techniques that turn an image from grey level into binary. These are called single point operations. After, we shall move to group operations where the group is those points found inside a template. Some of the most common operations on the groups of points are statistical, providing images where each point is the result of, say, averaging the neighbourhood of each point in the original image. We shall see how the statistical operations can reduce noise in the image, which is of benefit to the feature extraction techniques to be considered later. As such, these basic operations are usually for pre-processing for later feature extraction or to improve display quality. Finally, morphological operators process an image according to shape, starting with binary and moving to grey level operations.

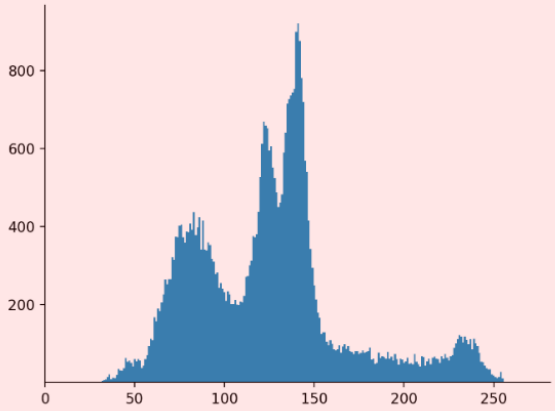
Main topic	Sub topics	Main points
Image Description	Portray variation in image brightness content as a graph/histogram.	*Histograms*, image *contrast*.
Point Operations	Calculate new image points as a function of the point at the same place in the original image. The functions can be mathematical, or can be computed from the image itself and will change the image's histogram. Finally, thresholding turns an image from grey level to a binary (black and white) representation.	*Histogram* manipulation; *intensity mapping*: addition, inversion, scaling, logarithm, exponent. *Intensity normalisation*; *histogram equalisation*. *Thresholding* and *optimal thresholding*.
Group Operations	Calculate new image points as a function of the neighbourhood of the point at the same place in the original image. The functions can be statistical including: mean (average); median and mode. Advanced filtering techniques, including feature preservation.	*Template convolution* (inc. frequency domain implementation). *Statistical operators*: *direct averaging*, *median* filter, and *mode* filter. *Non-local means*, a*nisotropic diffusion,* and *bilateral filter* for image smoothing. Other operators: *force field* and *image ray* transforms.
Image morphology and operators	Morphological operators that process an image according to shape, starting with binary and moving to grey level operations.	*Mathematical morphology*: *hit or miss transform*, *erosion*, *dilation* (inc. grey level operators) and *Minkowski* operators.

Table 3.1 Overview of Chapter 3

3.2 Histograms

The intensity *histogram* shows how individual brightness levels are occupied in an image; the *image contrast* is measured by the range of brightness levels. The histogram plots the number of pixels with a particular brightness level against the brightness level. For 8-bit pixels, the brightness ranges from zero (black) to 255 (white). Figure 3.1 shows an image and its histogram. The histogram, Figure 3.1(b), shows that not all the grey levels are used and the lowest and highest intensity levels are far apart, reflecting good contrast. Note that the image contains many light grey pixels that produce the wide lower peak in the histogram. If the image was darker, overall, the histogram would be concentrated towards black. If the image was brighter, but with lower contrast, then the histogram would be thinner and concentrated near the whiter brightness levels.

(a) image



(b) histogram

Figure 3.1 An Image and its Histogram

This histogram shows us that we have not used all available grey levels. Accordingly, we can stretch the image to use them all, and the image would become clearer. This is essentially cosmetic attention to make the image's appearance better. Making the appearance better, especially in view of later processing, is the focus of many basic image processing operations, as will be covered in this Chapter. The histogram can also reveal if there is much noise in the image, if the ideal histogram is known. We might want to remove this noise, not only to improve the appearance of the image, but to ease the task of (and to present the target better for) later feature extraction techniques. This Chapter concerns these basic operations which can improve the appearance and quality of images.

3.3 Point Operators

3.3.1 Basic Point Operations

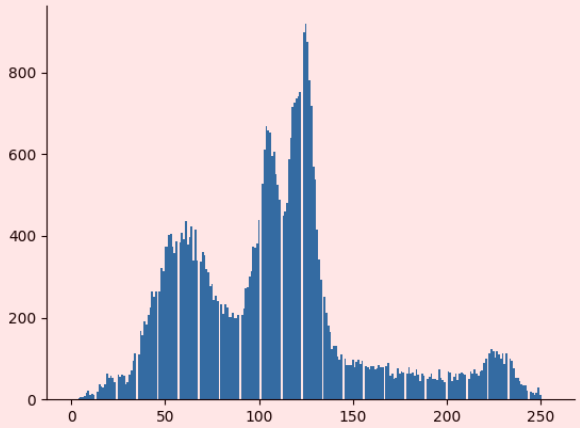
The most basic operations in image processing are *point operations* where each pixel value is replaced with a new value obtained from the old one. If we want to increase the brightness to stretch the contrast we can simply multiply all pixel values by a scalar, say by 2 to double the range. Conversely, to reduce the contrast (though this is not usual) we can divide all point values by a

scalar. If the overall brightness is controlled by a *level*, l, (e.g. the brightness of global light) and the range is controlled by a gain, k, the brightness of the points in a new picture, **N**, can be related to the brightness in old picture, **O**, by:

$$\mathbf{N}_{x,y} = k \times \mathbf{O}_{x,y} + l \qquad \forall x, y \in 1, N \tag{3.1}$$

This is a point operator that replaces the brightness at points in the picture according to a linear brightness relation. The level controls overall brightness and is the minimum value of the output picture. The gain controls the contrast, or range, and if the gain is greater than unity, the output range will be increased, this process is illustrated in Figure 3.2. So the image, processed by k = 1.1 and l = -10 will become brighter, Figure 3.2(a), and with better contrast, though in this case the brighter points are mostly set near to white (255). These factors can be seen in its histogram, Figure 3.2(b).

(a) brighter image



(b) histogram of brighter image

Figure 3.2 Brightening an Image

The basis of the implementation of point operators was given earlier, for inversion in Code 1.2. The stretching process can be displayed as a mapping between the input and output ranges, according to the specified relationship, as in Figure 3.3. Figure 3.3(a) is a mapping where the output is a direct copy of the input (this relationship is the dotted line in Figures 3.3(c) and (d)); Figure 3.3(b) is the mapping for *brightness inversion* where dark parts in an image become bright and vice versa. Figure 3.3(c) is the mapping for *addition* and Figure 3.3(d) is the mapping for *multiplication* (or *division*, if the slope was less than that of the input). In these mappings, if the mapping produces values that are smaller than the expected minimum (say negative when zero represents black), or larger than a specified maximum, then a *clipping* process can be used to set the output values to a chosen level. For example, if the relationship between input and output aims to produce output points with intensity values greater than 255, as used for white, the output value can be set to white for these points, as it is in Figure 3.2.

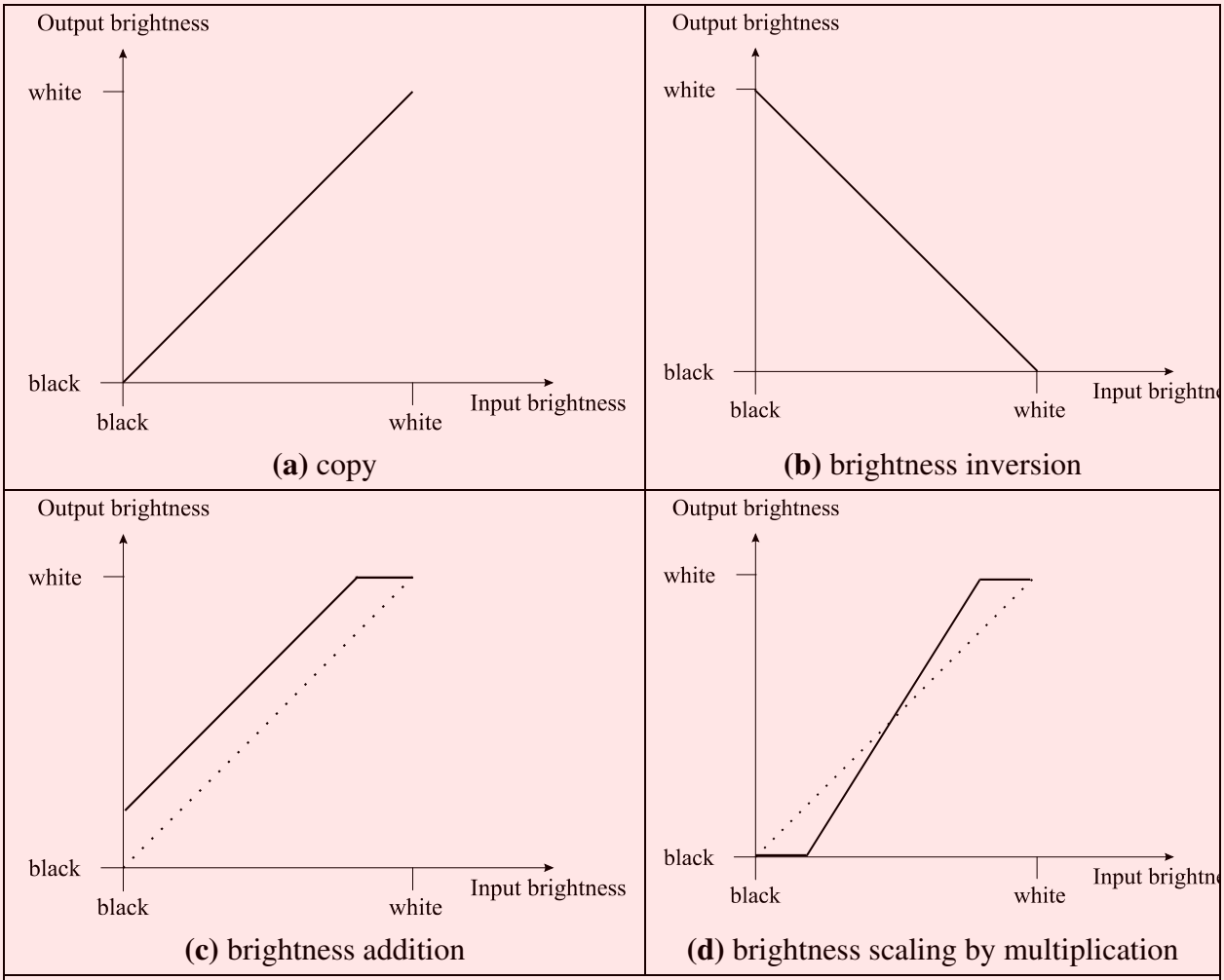


Figure 3.3 Intensity Mappings

Finally, rather than simple multiplication we can use arithmetic functions such as the logarithm to reduce the range or the exponent to increase it. This can be used, say, to equalise the response of a camera, or to compress the range of displayed brightness levels. If the camera has a known exponential performance, and outputs a value for brightness which is proportional to the exponential of the brightness of the corresponding point in the scene of view, the application of a *logarithmic point operator* will restore the original range of brightness levels. The implementation of point operators for arithmetic functions is illustrated in Code 3.1. This code simply computes the logarithm or exponential value of the value at each pixel position.

```
for x,y in itertools.product(range(0, width), range(0, height)):
    # Set the pixels in the Logarithmic
    outputLogarithmicImage[y,x] = 20 * log(inputImage[y,x] * 100.0)

    # Set the pixels in the Exponential image
    outputExponentialImage[y,x] = 20 * exp(inputImage[y,x] / 100.0)
```

Code 3.1 Point Operations

The effect of replacing brightness by a scaled version of its natural logarithm (implemented as $N_{x,y} = 20\ln(100O_{x,y})$) is shown in Figure 3.4(a); the effect of a scaled version of the exponent (implemented as $N_{x,y} = 20\exp(O_{x,y}/100)$) is shown in Figure 3.4(b). The scaling factors were chosen to ensure that the resulting image can be displayed since the logarithm or exponent greatly reduce or magnify, pixel values, respectively. This can be seen in the results: Figure 3.4(a) is dark with a small range of brightness levels whereas Figure 3.4(b) is much brighter, with greater contrast.

Naturally, application of the logarithmic point operator will change any multiplicative changes in brightness to become additive. As such, the logarithmic operator can find application in reducing the effects of multiplicative intensity change. The logarithm operator is often used to compress Fourier transforms for display purposes. This is because the d.c. component can be very large, or the contrast too large, to allow the other points to be seen.

(a) logarithmic compression

(b) exponential expansion

Figure 3.4 Applying Exponential and Logarithmic Point Operators

In hardware, point operators can be implemented using *look-up-tables* (LUTs). LUTs give an output that is programmed, and stored, in a table entry that corresponds to a particular input value. If the brightness response of the camera is known, it is possible to pre-program an LUT to make the camera response equivalent to a uniform or flat response across the range of brightness levels (in software, this can be implemented as an array or by using a map associative container).

3.3.2 Histogram Normalisation

Popular techniques to stretch the range of intensities include *histogram* (*intensity*) *normalisation*. Here, the original histogram is stretched, and shifted, to cover all the 256 available levels. If the original histogram of an old picture **O** starts at $\mathbf{O}min$ and extends up to $\mathbf{O}max$ brightness levels, then we can scale up the image so that the pixels in the new picture **N** lie between a minimum output level $\mathbf{N}min$ and a maximum level $\mathbf{N}max$, simply by scaling up the input intensity levels according to:

$$\mathbf{N}_{x,y} = \frac{\mathbf{N}max - \mathbf{N}min}{\mathbf{O}max - \mathbf{O}min} \times \left(\mathbf{O}_{x,y} - \mathbf{O}min\right) + \mathbf{N}min \qquad \forall x, y \in 1, N \qquad (3.2)$$

Code 3.2 gives an implementation of intensity normalisation. The code uses an output ranging from $\mathbf{N}min = 0$ to $\mathbf{N}max = 255$ that is the maximum range for images that use a byte per pixel. This is scaled by the input range that is determined from the maximum and minimum values are returned by the `imageMaxMin` function. Each point in the picture is then scaled as in Equation 3.2. Note that Matlab's `imagesc` function appears to effect normalisation.

```
# Maximum and range
maxVal, miniVal = imageMaxMin(inputImage)
brightRange = float(maxVal - miniVal)

# Set the pixels in the output image
for x,y in itertools.product(range(0, width), range(0, height)):

    # Normalize the pixel value according to the range
```

```
outputNormalizedImage[y,x] = ((inputImage[y,x] - miniVal) * 255.0 / brightRange)
```

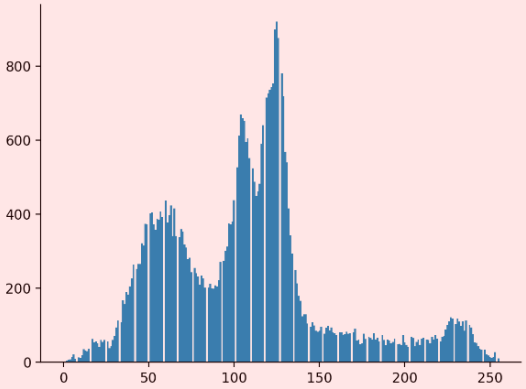
Code 3.2 Intensity Normalisation

The normalisation process is illustrated in Figure 3.5, and can be compared with the original image and histogram in Figure 3.1. An intensity-normalised version of the image is shown in Figure 3.5(a) which now has better contrast and appears better to human vision. Its histogram, Figure 3.5(b), shows that the intensity now ranges across all available levels (there is actually one black pixel!).

3.3.3 Histogram Equalisation

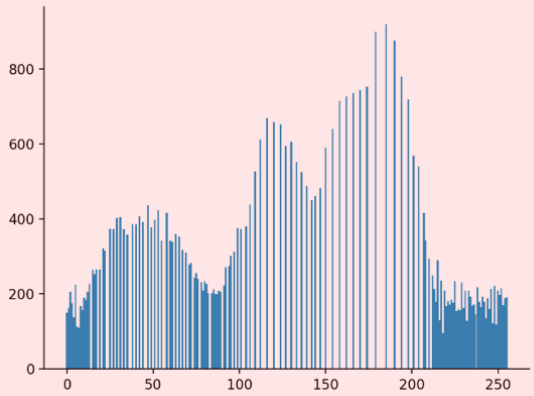
Histogram equalisation is a non-linear process aimed to highlight image brightness in a way particularly suited to human visual analysis. Histogram equalisation aims to change a picture in such a way as to produce a picture with a flatter histogram, where all levels are equiprobable. In order to develop the operator, we can first inspect the histograms. For a range of M levels then the histogram plots the points per level against level. For the input (old) and the output (new) image, the number of points per level is denoted as $\mathbf{O}(l)$ and $\mathbf{N}(l)$ (for $0<l<M$), respectively. For square images, there are N^2 points in the input and the output image, so the sum of points per level in each should be equal:

(a) intensity normalised image



(b) histogram of intensity normalised image

(c) histogram equalised image



(d) histogram of histogram equalised image

Figure 3.5 Illustrating Intensity Normalisation and Histogram Equalisation

$$\sum_{l=0}^{M} \mathbf{O}(l) = \sum_{l=0}^{M} \mathbf{N}(l) \tag{3.3}$$

Also, this should be the same for an arbitrarily chosen level p, since we are aiming for an output picture with a uniformly flat histogram. So the cumulative histogram up to level p should be transformed to cover up to the level q in the new histogram:

$$\sum_{l=0}^{p} \mathbf{O}(l) = \sum_{l=0}^{q} \mathbf{N}(l) \tag{3.4}$$

Since the output histogram is uniformly flat, the cumulative histogram up to level p should be a fraction of the overall sum. So the number of points per level in the output picture is the ratio of the number of points to the range of levels in the output image:

$$\mathbf{N}(l) = \frac{N^2}{\mathbf{N}max - \mathbf{N}min} \tag{3.5}$$

So the cumulative histogram of the output picture is:

$$\sum_{l=0}^{q} \mathbf{N}(l) = q \times \frac{N^2}{\mathbf{N}max - \mathbf{N}min} \tag{3.6}$$

By Equation 3.4 this is equal to the cumulative histogram of the input image, so that:

$$q \times \frac{N^2}{\mathbf{N}max - \mathbf{N}min} = \sum_{l=0}^{p} \mathbf{O}(l) \tag{3.7}$$

This gives a mapping for the output pixels at level q, from the input pixels at level p as:

$$q = \frac{\mathbf{N}max - \mathbf{N}min}{N^2} \times \sum_{l=0}^{p} \mathbf{O}(l) \tag{3.8}$$

This gives a mapping function that provides an output image that has an approximately flat histogram. The mapping function is given by phrasing Equation 3.8 as an equalising function (E) of the level (q) and the image ($\mathbf{O}$) as

$$E(q, \mathbf{O}) = \frac{\mathbf{N}max - \mathbf{N}min}{N^2} \times \sum_{l=0}^{p} \mathbf{O}(l) \tag{3.9}$$

The output image is then

$$\mathbf{N}_{x,y} = E(\mathbf{O}_{x,y}, \mathbf{O}) \tag{3.10}$$

```
function equalised = equalise(image)
%get dimensions
[rows,cols]=size(image);
%specify range of levels
range=255;
%and the number of points
number=cols*rows;

%initialise the image histogram
hist(1:256)=0;

%work out the histogram
for x = 1:cols %address all columns
  for y = 1:rows %address all rows
    hist(image(y,x)+1)=hist(image(y,x)+1)+1;
  end
end;

%evaluate the cumulative histogram
sum=0;
for i=1:256
```

```
  sum=sum+hist(i);
  cumhist(i)=floor(sum*range/number); %Eq. 3.9
end

%map using the cumulative histogram
for x = 1:cols %address all columns
  for y = 1:rows %address all rows
    equalised(y,x)=cumhist(image(y,x)); %Eq 3.10
  end
end
```

Code 3.3 Histogram Equalisation

The result of equalising an image is shown in Figures 3.5(c) and (d). The intensity equalised image, Figure 3.5(c) has much better defined features than in the original version (Figure 3.1). The histogram, Figure 3.5(d), reveals the non-linear mapping process whereby white and black are not assigned equal weight, as they were in intensity normalisation. Accordingly, more pixels are mapped into the darker region and the brighter intensities become better spread, consistent with the aims of histogram equalisation.

Its performance can be very convincing since it is well mapped to the properties of human vision. If a linear brightness transformation is applied to the original image then the equalised histogram will be the same. If we replace pixel values with ones computed according to Equation 3.1, the result of histogram equalisation will not change. An alternative interpretation is that if we equalise images (prior to further processing) then we need not worry about any brightness transformation in the original image. This is to be expected, since the linear operation of the brightness change in Equation 3.2 does not change the overall shape of the histogram, only its size and position. However, noise in the image acquisition process will affect the shape of the original histogram, and hence the equalised version. So the equalised histogram of a picture will not be the same as the equalised histogram of a picture with some noise added to it. You cannot avoid noise in electrical systems, however well you design a system to reduce its effect. Accordingly, histogram equalisation finds little use in generic image processing systems, except for display, though it can be potent in specialised applications. For these reasons, intensity normalisation is often preferred when a picture's histogram requires manipulation.

In implementation, the function `equalise` in Code 3.3, we shall use an output range where $\mathbf{N}min = 0$ and $\mathbf{N}max = 255$. The implementation first determines the cumulative histogram for each level of the brightness histogram. This is then used as a look up table for the new output brightness at that level. The look up table is used to speed implementation of Equation 3.9, since it can be precomputed from the image to be equalised.

An alternative argument also against the use of histogram equalisation is that it is a non-linear process and is irreversible. We cannot return to the original picture after equalisation, and we cannot separate the histogram of an unwanted picture. On the other hand, intensity normalisation is a linear process and we can return to the original image, should we need to, or separate pictures - if required. Note that there have been extensions to histogram equalisation, and *adaptive histogram equalisation* with some extensions [Pizer87] has proved particularly enduring.

3.3.4 Thresholding

The last point operator of major interest is called *thresholding*. This operator selects pixels which have a particular value, or are within a specified range. It can be used to find objects within a picture if their brightness level (or range) is known. This implies that the object's brightness must be known as well. There are two main forms: uniform and adaptive thresholding. In *uniform thresholding*, pixels above a specified level are set to white, those below the specified level are set to black.

$$\mathbf{N}_{x,y} = \begin{cases} 255 & if \quad \mathbf{O}_{x,y} > threshold \\ 0 & otherwise \end{cases} \tag{3.11}$$

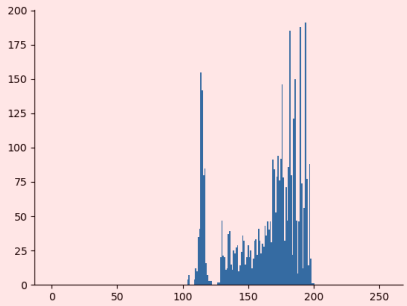
As shown in Code 3.4 the implementation of thresholding sets the value of the output image by an `if` condition according to the threshold parameter.

```
for x,y in itertools.product(range(0, width), range(0, height)):
    if inputImage[y,x] > threshold:
        outputImage[y,x] = 255
    else:
        outputImage[y,x] = 0
```

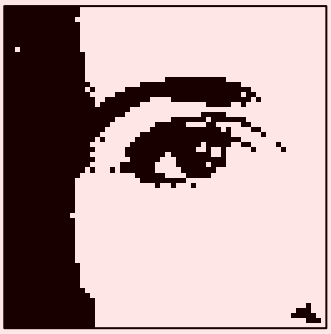
Code 3.4 Image Thresholding

Given the original eye image, Figure 3.6 shows a thresholded image where all pixels above 160 brightness levels are set to white, and those below 160 brightness levels are set to black. By this process, the parts pertaining to the facial skin are separated from the background; the cheeks, forehead and other bright areas are separated from the hair and eyes. This can therefore provide a way of isolating points of interest.

(a) image



(b) histogram



(c) thresholded image

Figure 3.6 Thresholding the Eye Image

Uniform thresholding clearly requires knowledge of the grey level, or the target features might not be selected in the thresholding process. If the level is not known, histogram equalisation or intensity normalisation can be used, but with the restrictions on performance stated earlier. This is, of course, a problem of image interpretation. These problems can only be solved by simple approaches, such as thresholding, for very special cases.

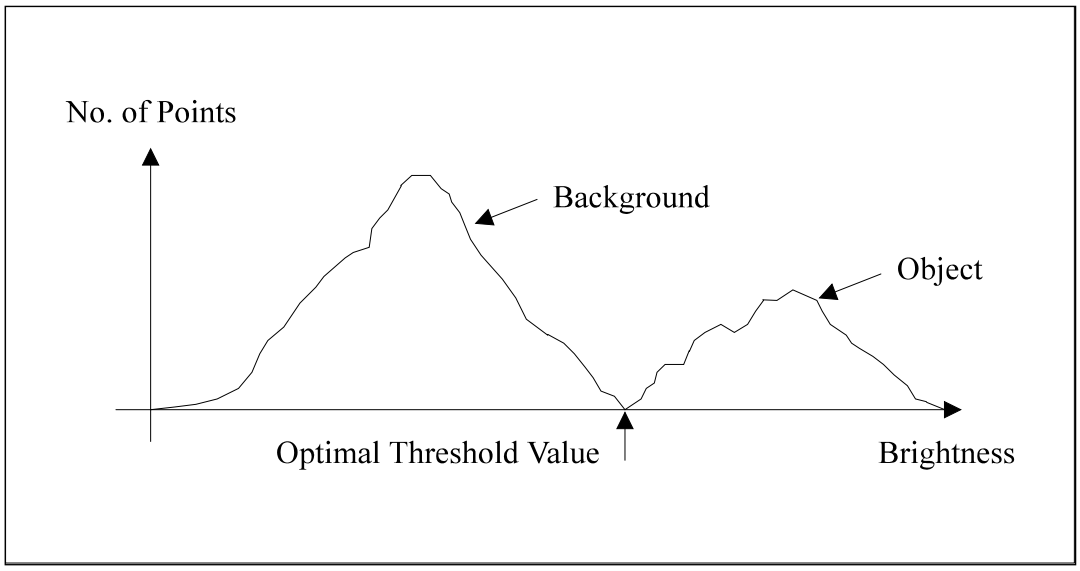


Figure 3.7 Optimal Thresholding

There are more advanced techniques, known as *optimal thresholding*. These usually seek to select a value for the threshold that separates an object from its background. This suggests that the object has a different range of intensities to the background, in order that an appropriate threshold can be chosen, as illustrated in Figure 3.7. *Otsu's method* [Otsu79] is one of the most popular techniques of optimal thresholding; there have been surveys [Sahoo88, Lee90, Glasbey93] which compare the performance different methods can achieve. Essentially, Otsu's technique maximises the likelihood that the threshold is chosen so as to split the image between an object and its background. This is achieved by selecting a threshold that gives the best separation of classes, for all pixels in an image. The basis is use of the normalised histogram where the number of points at each level is divided by the total number of points in the original image. As such, this represents a probability distribution for the intensity levels as

$$p(l) = \frac{\mathbf{O}(l)}{N^2} \tag{3.12}$$

This can be used to compute then zero- and first-order cumulative moments of the normalised histogram up to the k^{th} level as

$$\omega(k) = \sum_{l=1}^{k} p(l) \tag{3.13}$$

and

$$\mu(k) = \sum_{l=1}^{k} l \cdot p(l) \tag{3.14}$$

The total mean level μ_{T} of the image is given by

$$\mu_{\mathrm{T}} = \sum_{l=1}^{N\max} l \cdot p(l) \tag{3.15}$$

The variance of the class separability (the similarity between the variables of the same class) is then the ratio

$$\sigma_B^2(k) = \frac{\left(\mu_{\mathrm{T}} \cdot \omega(k) - \mu(k)\right)^2}{\omega(k)\left(1-\omega(k)\right)} \qquad \forall k \in 1, N\max \tag{3.16}$$

The optimal threshold is the level for which the variance of class separability is at its maximum, namely the optimal threshold T_{opt} is that for which the variance

$$\sigma_B^2\left(T_{\text{opt}}\right) = \max_{1 \le k < N\max} \left(\sigma_B^2(k)\right) \tag{3.17}$$

The implementation of the optimal thresholding is shown in Code 3.5. The code finds the optimum threshold in two stages. First, it uses the histogram of the input image to compute the moments in Equations 3.13 and 3.14. A second step computes the separability measure in Equation 3.16. The code keeps all the separability values for display, but in practice we only require to keep the maximum value.

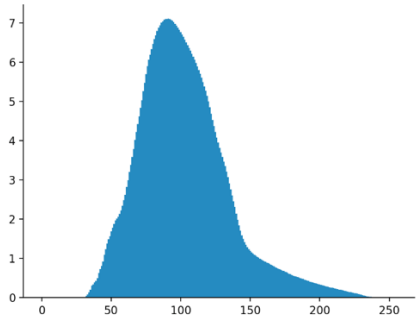
```
# Obtain histograms
normalization = 1.0 / float(width * height)
w[0] = normalization * inputHistogram[0]
for level in range(1, 256):
    w[level] = w[level-1] + normalization * inputHistogram[level]
    m[level] = m[level-1] + level * normalization * inputHistogram[level]

# Look for the maximum
maximumLevel = 0
for level in range(0, 256):
    if w[level] * (float(level) - w[level]) != 0:
        separability[level] = float(pow( ( m[255] * w[level] - m[level]), 2)      \
                                    / (w[level] * (float(level) - w[level])))

        if separability[level] > separability[maximumLevel]:
            maximumLevel = level
```

Code 3.5 Optimal Thresholding

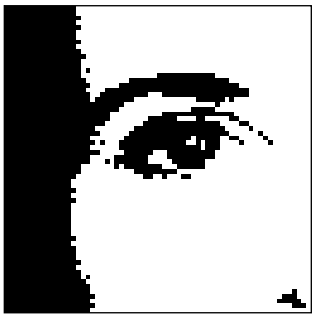
Figure 3.**8** shows an example of optimal thresholding obtained with Code 3.5. Figure 3.**8**(a) shows the separability computed for each potential threshold. Low values represent thresholds that produce two regions with high inter class variance and high values minimize the interclass variance. The threshold for the maximum value is used to produce the thresholded image in Figure 3.**8**(b). In this figure the pixels are divided in two classes whose variance is minimum compared with any other possible threshold.



(a) separability

(b) thresholded image

Figure 3.8 Optimal Thresholding Example



(a) thresholding at level 160

(b) thresholding by Otsu (level = 127)

Figure 3.9 Thresholding the Eye Image: Manual and Automatic

A comparison of uniform thresholding with optimal thresholding is given in Figure 3.9 for the eye image. The threshold selected by Otsu's operator is actually lower than the value selected manually, and so the thresholded image does omit some detail around the eye, especially in the eyelids. However, the selection by Otsu is automatic, as opposed to manual and this can be to application advantage in automated vision. Consider for example the need to isolate the human figure in Figure 3.10(a). This can be performed automatically by Otsu as shown in Figure 3.10(b). Note however that there are some extra points, due to illumination, which have appeared in the resulting image together with the human subject. It is easy to remove the isolated points, as we will see later, but more difficult to remove the connected ones. In this instance, the size of the human shape could be used as information to remove the extra points though you might like to suggest other factors that could lead to their removal.

(a) walking subject

(b) automatic thresholding by Otsu

Figure 3.10 Thresholding an Image of a Walking Subject

Also, we have so far considered global techniques, methods that operate on the entire image. There are also locally adaptive techniques that are often used to binarise document images prior to character recognition. As mentioned before, surveys of thresholding are available, and one approach [Rosin01] targets thresholding of images whose histogram is unimodal (has a single peak). One survey [Trier95] compares global and local techniques with reference to document image analysis. These techniques are often used in statistical pattern recognition: the thresholded object is classified according to its statistical properties. However, these techniques find less use in image interpretation, where a common paradigm is that there is more than one object in the scene, such as Figure 3.6 where the thresholding operator has selected many objects of potential interest. As such, only uniform thresholding is used in many vision applications, since objects are often occluded (hidden), and many objects have similar ranges of pixel intensity. Accordingly, more