CHAPTER 9

HISTOGRAM PROCESSING

WHAT WILL WE LEARN?

- What is the histogram of an image?
- How can the histogram of an image be computed?
- How much information does the histogram provide about the image?
- What is histogram equalization and what happens to an image whose histogram is equalized?
- How can the histogram be modified through direct histogram specification and what happens to an image when we do it?
- What other histogram modification techniques can be applied to digital images and what is the result of applying such techniques?

9.1 IMAGE HISTOGRAM: DEFINITION AND EXAMPLE

The histogram of a monochrome image is a graphical representation of the frequency of occurrence of each gray level in the image. The data structure that stores the frequency values is a 1D array of numerical values, h, whose individual elements store the number (or percentage) of image pixels that correspond to each possible gray level.

Each individual histogram entry can be expressed mathematically as

$$h(k) = n_k = \text{card}\{(x, y) | f(x, y) = k\} \tag{9.1}$$

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

Here, $k = 0, 1, ..., L - 1$, where L is the number of gray levels of the digitized image, and card$\{\cdots\}$ denotes the cardinality of a set, that is, the number of elements in that set (n_k).

A normalized histogram can be mathematically defined as

$$p(r_k) = \frac{n_k}{n} \tag{9.2}$$

where n is the total number of pixels in the image and $p(r_k)$ is the probability (percentage) of the kth gray level (r_k).

Histograms are normally represented using a bar chart, with one bar per gray level, in which the height of the bar is proportional to the number (or percentage) of pixels that correspond to that particular gray level.

In MATLAB

MATLAB's IPT has a built-in function to calculate and display the histogram of a monochrome image: `imhist`. Alternatively, other MATLAB plotting functions such as `bar`, `plot`, and `stem` can also be used to display histograms. In Tutorial 9.1 (page 188), you will have a chance to experiment with all of them.

■ EXAMPLE 9.1

Table 9.1 shows the pixel counts for a hypothetical image containing 128×128 pixels, with eight gray levels. The number of pixels that correspond to a given gray level is indicated in the second column and the corresponding percentages (probabilities), $p(r_k)$, are given in the third column. Its bar graph representation is shown in Figure 9.1.

Each value of $p(r_k)$ represents the percentage of pixels in the image whose gray level is r_k. In other words, a histogram can be interpreted as a probability mass function of a random variable (r_k) and as such it follows all the axioms and theorems of elementary probability theory. For instance, it is easy to verify from Table 9.1 that the sum of the values for $p(r_k)$ is 1, as expected.

TABLE 9.1 Example of a Histogram

Gray Level (r_k)	n_k	$p(r_k)$
0	1120	0.068
1	3214	0.196
2	4850	0.296
3	3425	0.209
4	1995	0.122
5	784	0.048
6	541	0.033
7	455	0.028
Total	16,384	1.000

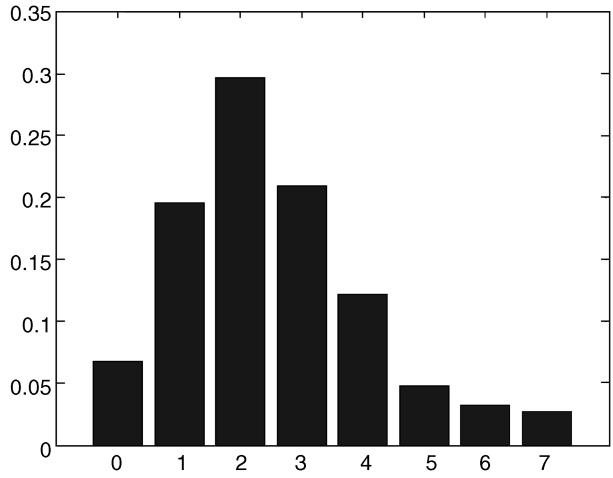

FIGURE 9.1 Example of histogram for an image with eight gray levels.

9.2 COMPUTING IMAGE HISTOGRAMS

To compute the histogram of an 8-bit (256 gray levels) monochrome image, an array of 256 elements (each of which acts as a counter) is created and initialized with zeros. The image is then read, one pixel at a time, and for each pixel the array position corresponding to its gray level is incremented.[1] After the whole image is processed, each array element will contain the number of pixels whose gray level corresponds to the element's index. These values can then be normalized, dividing each of them by the total number of pixels in the image.

For images with more than 8 bits per pixel, it is not practical to map each possible gray level k ($0 \leq k \leq K$) to an array element: the resulting array would be too large and unwieldy. Instead, we use a technique known as *binning*, by which an array of B elements—where B is the number of histogram *bins* (or *buckets*), $B \leq K$—is created and initialized with zeros. In this case, each bin $h(j)$ stores the number of pixels having values within the interval $r_j \leq r < r_{j+1}$ and equation (9.1) can be rewritten as

$$h(j) = \text{card}\{(x, y) | r_j \leq f(x, y) < r_{j+1}\} \quad \text{for } 0 \leq j < B \tag{9.3}$$

The lower limit of each bin can be obtained by the following expression:

$$r_j = j \cdot \frac{K}{B} = j \cdot k_B \tag{9.4}$$

where k_B is the length of each interval.

[1]Keep in mind that arrays in MATLAB are 1 based, and therefore, in MATLAB, the array element with index $k + 1$ will store the number or percentage of pixels whose value is k, where $k \geq 0$.

9.3 INTERPRETING IMAGE HISTOGRAMS

Histograms provide an easy, practical, and straightforward way of evaluating image attributes, such as overall contrast and average brightness. The histogram of a predominantly dark image contains a concentration of bars on the lower end of the gray-level range (Figure 9.2c), whereas the histogram for a bright image is mostly concentrated on the opposite end (Figure 9.2d). For a low contrast image, the histogram is clustered within a narrow range of gray levels (Figure 9.2a), whereas a high contrast image usually exhibits a bimodal histogram with clear separation between the two predominant modes (Figure 9.2b).[2]

■ EXAMPLE 9.2

Figure 9.2 shows four examples of monochrome images and their corresponding histograms (obtained using the `imhist` function in MATLAB).

The histogram in Figure 9.2a shows that the pixels are grouped around intermediate gray-level values (mostly in the [100, 150] range), which corresponds to an image with low contrast. Figure 9.2b shows a typical bimodal histogram, one that has two distinctive hills, a more pronounced one in the dark region (background) and the other smaller one in the light region of the histogram (foreground objects). In these situations, it can be said that the corresponding image has high contrast since the two modes are well spaced from each other.[3] In part (c), the histogram exhibits a large concentration of pixels in the lower gray levels, which corresponds to a mostly dark image. Finally, in Figure 9.2d, the pixel values are grouped close to the higher gray-level values, which corresponds to a bright image.

It is important to note that even though a histogram carries significant qualitative and quantitative information about the corresponding image (e.g., minimum, average, and maximum gray-level values, dominance of bright or dark pixels, etc.), other qualitative conclusions can be reached only upon examining the image itself (e.g., overall quality of the image, presence or absence of noise, etc.). A quick analysis of Figure 9.2 should be enough to prove this statement true. Moreover, although a histogram provides the frequency distribution of gray levels in an image, it tells us nothing about the spatial distribution of the pixels whose gray levels are represented in the histogram (see Problem 9.8).

Histograms have become a popular tool for conveying image statistics and helping determine certain problems in an image. Their usefulness can be demonstrated by the

[2]In many cases, those two modes (hills) correspond to background and one or more objects of interest in the foreground, for example, Figure 9.2b.

[3]It is interesting to note that the concepts of *high* and *low* contrast in this case are only related to the average spacing between groups of histogram bars. The expression "good contrast," on the other hand, should be used with caution, for it often refers to one's subjective opinion about the image quality. Such quality judgments usually cannot be made based upon looking at the histogram alone.

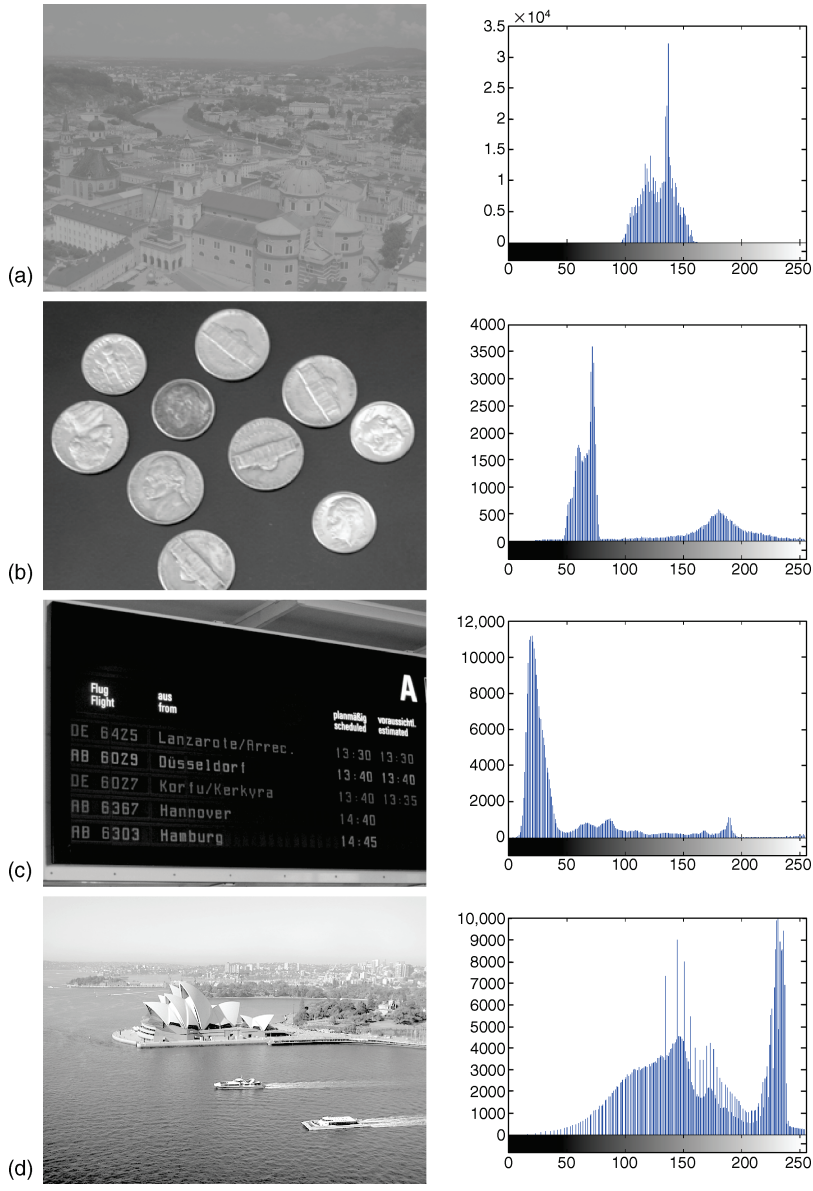

FIGURE 9.2 Examples of images and corresponding histograms. Original image in part (b): courtesy of MathWorks.

fact that many contemporary digital cameras have an optional real-time histogram overlay in the viewfinder to prevent taking underexposed or overexposed pictures.

Histograms can be used whenever a statistical representation of the gray-level distribution in an image or video frame is desired. Histograms can also be used to

enhance or modify the characteristics of an image, particularly its contrast. Some of these techniques, generally called *histogram modification (or modeling) techniques*, are histogram equalization, histogram specification (matching), histogram stretching (input cropping), and histogram shrinking (output cropping). They are described in more detail later.

9.4 HISTOGRAM EQUALIZATION

Histogram equalization is a technique by which the gray-level distribution of an image is changed in such a way as to obtain a uniform (flat) resulting histogram, in which the percentage of pixels of every gray level is the same. To perform histogram equalization, it is necessary to use an auxiliary function, called the *transformation function*, $T(r)$. Such transformation function must satisfy two criteria [GW08]:

1. $T(r)$ must be a monotonically increasing function in the interval $0 \leq r \leq L-1$.
2. $0 \leq T(r) \leq L-1$ for $0 \leq r \leq L-1$.

The most usual transformation function is the *cumulative distribution function (cdf)* of the original probability mass function, given by

$$s_k = T(r_k) = \sum_{j=0}^{k} \frac{n_j}{n} = \sum_{j=0}^{k} p(r_j) \tag{9.5}$$

where s_k is the new (mapped) gray level for all pixels whose original gray level used to be r_k.

The inverse of this function is given by

$$r_k = T^{-1}(s_k) \quad \text{for} \quad k = 0, 1, \ldots, L-1 \tag{9.6}$$

Even though the inverse of the cdf is not needed during the equalization process, it will be used during the direct histogram specification method, described later in this chapter.

In MATLAB

MATLAB's IPT has a built-in function to perform histogram equalization of a monochrome image: `histeq`. For the sake of histogram equalization, the syntax for `histeq` is usually `J = histeq(I,n)`, where `n` (whose default value is 64)[4] is the number of desired gray levels in the output image. In addition to histogram equalization, this function can also be used to perform histogram matching (Section 9.5), as it will be seen in Tutorial 9.2.

[4]Caution: This value is *not* the same as the default value for the imhist function that uses $n = 256$ as a default.

■ EXAMPLE 9.3

Assume the histogram data from Table 9.1 and its graphic representation (Figure 9.1). Calculate the equalized histogram using the cdf of the original probability mass function as a transformation function and plot the resulting histogram.

Solution

Using the *cdf* as the transformation function, we can calculate

$$s_0 = T(r_0) = \sum_{j=0}^{0} p(r_j) = p(r_0) = 0.068 \tag{9.7}$$

Similarly,

$$s_1 = T(r_1) = \sum_{j=0}^{1} p(r_j) = p(r_0) + p(r_1) = 0.264 \tag{9.8}$$

and $s_2 = 0.560$, $s_3 = 0.769$, $s_4 = 0.891$, $s_5 = 0.939$, $s_6 = 0.972$, and $s_7 = 1$. The transformation function is plotted in Figure 9.3.

Since the image was quantized with only eight gray levels, each value of s_k must be rounded to the closest valid (multiple of 1/7) value. Thus, $s_0 \simeq 0$, $s_1 \simeq 2$, $s_2 \simeq 4$, $s_3 \simeq 5$, $s_4 \simeq 6$, $s_5 \simeq 7$, $s_6 \simeq 7$, and $s_7 \simeq 7$.

The above values indicate that the original histogram bars must be shifted (and occasionally grouped) according to the following mapping: the original level $r_0 = 0$ should be mapped to the new level $s_0 = 0$, which means the corresponding bar should not change. The 3214 pixels whose original gray level was (1/7) should be mapped to $s_1 = 2(1/7)$. Similarly, those pixels whose gray level was 2 should be mapped to

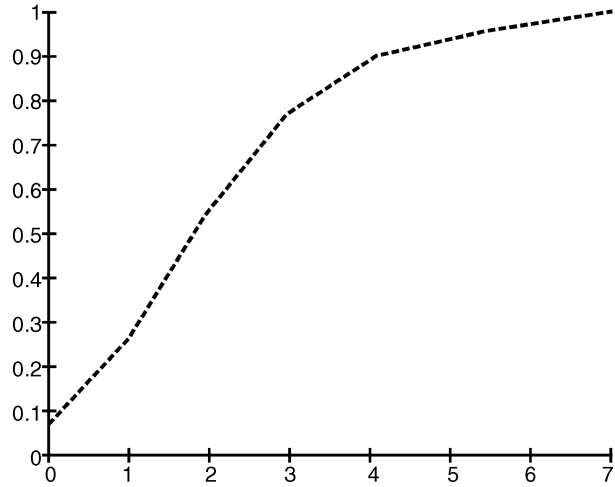

FIGURE 9.3 Transformation function used for histogram equalization.

TABLE 9.2 Equalized Histogram: Values

Gray Level (s_k)	n_k	$p(s_k)$
0	1120	0.068
1	0	0.000
2	3214	0.196
3	0	0.000
4	4850	0.296
5	3425	0.209
6	1995	0.122
7	1780	0.109
Total	16,384	1.000

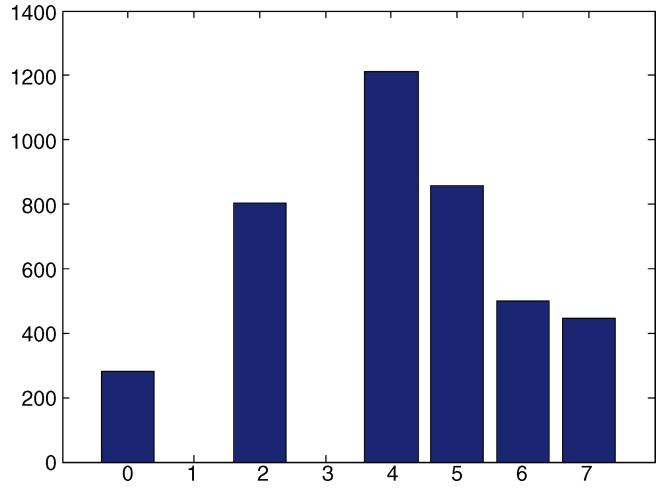

FIGURE 9.4 Equalized histogram—graph.

4, those with $r = 3$ should be mapped to 5, and those with gray level 4 should be mapped to 6. Finally, the three bins that correspond to pixels with gray levels 5, 6, and 7 should be added and mapped to 7.

The resulting (equalized) histogram is shown in Figure 9.4 and the corresponding values are presented in Table 9.2. It should be noted that in the equalized histogram (Figure 9.4), the pixels are more evenly distributed along the gray scale than in the original one (Figure 9.1). Although clearly not a perfectly flat result, it should be interpreted as "the best possible result that can be obtained for this particular image using this transformation function."

■ EXAMPLE 9.4

Figure 9.5 shows an example (obtained using the `histeq` function in MATLAB) of employing histogram equalization to improve the contrast of a 600 × 800

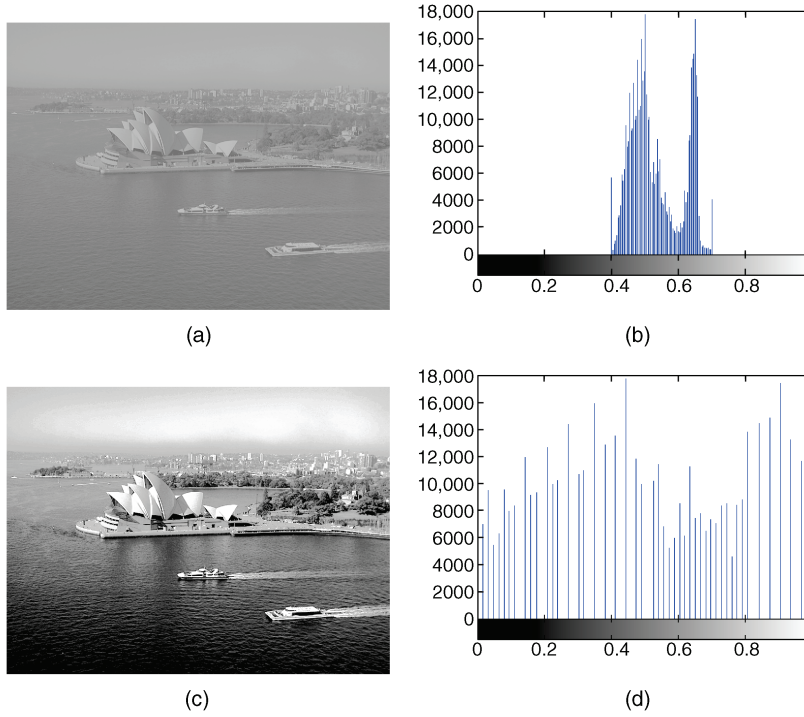

FIGURE 9.5 Use of histogram equalization to improve image contrast.

image with 256 gray levels. Part (a) shows the original image, whose histogram is plotted in Figure 9.5b. Part (d) shows the equalized histogram, corresponding to the image in Figure 9.5c. You may have noticed that the image after equalization exhibits a little bit of *false contouring* (see Chapter 5), particularly in the sky portion of the image. This can be seen as an inevitable "side effect" of the histogram equalization process.

The histogram equalization algorithm described above is *global*; that is, once the mapping (transformation) function is computed, it is applied (as a lookup table) to all pixels of the input image. When the goal is to enhance details in small areas within an image, it is sometimes necessary to apply a *local* version of the algorithm. The local variant of histogram equalization consists of adopting a rectangular (usually square) sliding window (also called a *tile*) and moving it across the entire image. For each image pixel (aligned with the center of the sliding window), the histogram in the neighborhood delimited by the window is computed, the mapping function is calculated, and the reference pixel is remapped to the new value as determined by the transformation function. This process is clearly much more computationally expensive than its global variant and often results in a noisy-looking output image.

■ EXAMPLE 9.5

Figure 9.6 shows a comparison between local and global histogram equalization (obtained using the `adapthisteq` and `histeq` functions in MATLAB, respectively). Parts (a) and (b) show the original image and its histogram, parts (c) and (d) show the results of global histogram equalization, and parts (e) and (f) show the results of local histogram equalization that preserves the bimodal nature of the original histogram while still improving the contrast of the image.

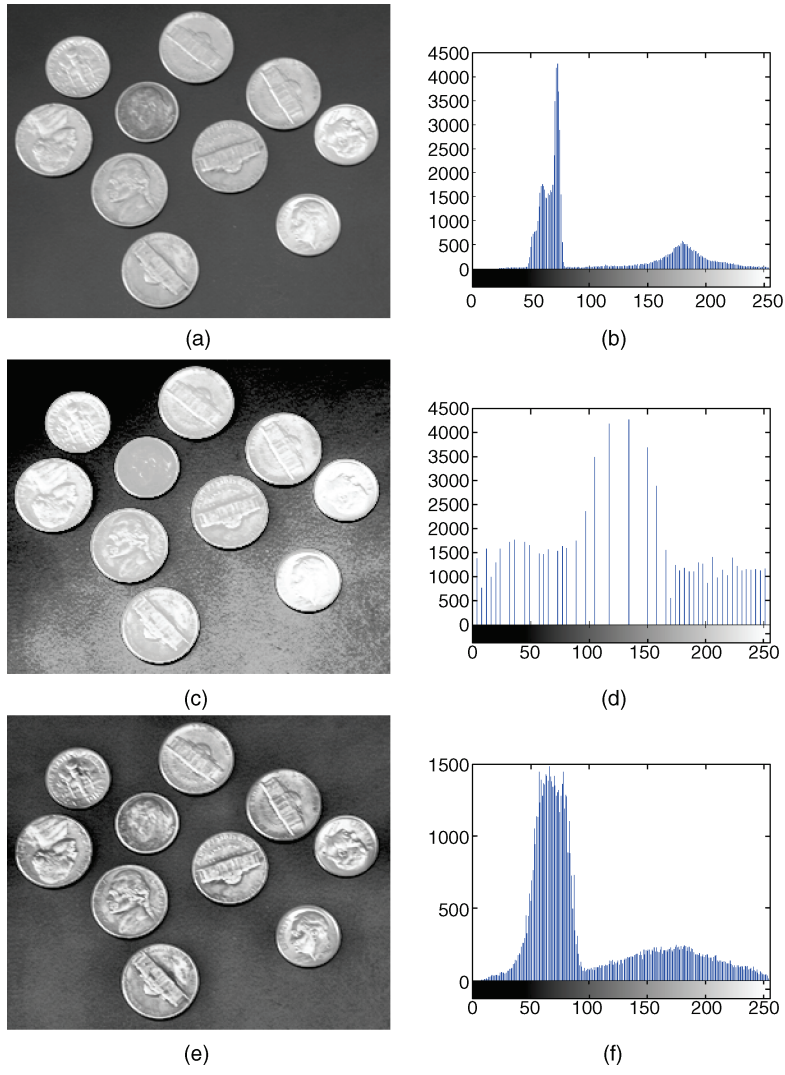

FIGURE 9.6 Global versus local histogram equalization. Original image: courtesy of MathWorks.

In MATLAB

MATLAB's IPT has a built-in function to perform local histogram equalization of a monochrome image, `adapthisteq`, which, unlike `histeq`, operates on small data regions (tiles), rather than the entire image. You will learn how to use this function in Tutorial 9.2.

9.5 DIRECT HISTOGRAM SPECIFICATION

Despite its usefulness in contrast enhancement, histogram equalization is a rather inflexible technique. Its only modifiable parameter is the choice of transformation function, which is normally chosen to be the cdf of the original probability mass function (a convenient choice since the information needed to compute the transformation function can be extracted directly from the pixel values). There might be situations, however, in which one wants to be able to perform specific changes on the original histogram. In these situations, a useful technique is the direct histogram specification, also known as *histogram matching*.

Given an image (and its original histogram) and the desired resulting histogram, the direct histogram specification consists of the following:

1. Equalizing the original image's histogram using the cdf as a transformation function:

$$s_k = T(r_k) = \sum_{j=0}^{k} \frac{n_j}{n} = \sum_{j=0}^{k} p(r_j) \tag{9.9}$$

2. Equalizing the desired probability mass function (in other words, equalizing the desired histogram):

$$v_k = G(z_k) = \sum_{j=0}^{k} p(z_j) \tag{9.10}$$

3. Applying the inverse transformation function

$$z = G^{-1}(s) \tag{9.11}$$

to the values obtained in step 1.

In MATLAB

The `histeq` function introduced previously can also be used for histogram matching. In this case, the syntax for `histeq` usually changes to `J = histeq(I,h)`, where `h` (a 1D array of integers) represents the specified histogram.

TABLE 9.3 Desired Histogram

Gray Level (s_k)	n_k	$p(s_k)$
0	0	0.0
1	0	0.0
2	0	0.0
3	1638	0.1
4	3277	0.2
5	6554	0.4
6	3277	0.2
7	1638	0.1
Total	16,384	1.0

■ EXAMPLE 9.6

Let us use the histogram from Table 9.1 one more time. Assume that we want to modify this histogram in such a way as to have the resulting pixel distribution as shown in Table 9.3 and plotted in Figure 9.7a. Following the steps outlined above, calculate and plot the resulting histogram that best matches the desired characteristics.

Solution

The equalized histogram has already been calculated before and its results are shown in Table 9.2.

The next step consists in obtaining the cdf of the desired probability mass function. Using equation (9.10), we find the following values:

$v_0 = 0$, $v_1 = 0$, $v_2 = 0$, $v_3 = 0.1$, $v_4 = 0.3$, $v_5 = 0.7$, $v_6 = 0.9$, and $v_7 = 1$.

The last step—and the most difficult to understand when studying this technique for the first time—is obtaining the inverse function. Since we are dealing with discrete values, the inverse function can be obtained simply by searching, for each value of s_k, the closest value of v_k. For instance, for $s_1 = 2/7 \simeq 0.286$, the closest value of v_k is $v_4 = G(z_4) = 0.3$. In inverse function notation, $G^{-1}(0.3) = z_4$. Therefore, pixels

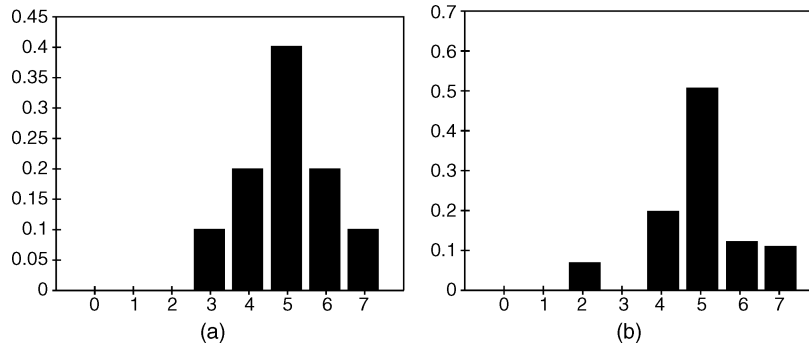

FIGURE 9.7 Histogram matching: (a) desired (specified) histogram; (b) resulting histogram.

TABLE 9.4 Direct Histogram Specification: Summary

k	$p(r_k)$	s_k ($\times 1/7$)	Maps to	v_k	$p(z_k)$
0	0.068	0	z_2	0.00	0.000
1	0.196	2	z_4	0.00	0.000
2	0.296	4	z_5	0.00	0.000
3	0.209	5	z_5	0.10	0.100
4	0.122	6	z_6	0.30	0.200
5	0.048	7	z_7	0.70	0.400
6	0.033	7	z_7	0.90	0.200
7	0.028	7	z_7	1.00	0.100

that were shifted to gray level s_1 after the original histogram equalization should be mapped to gray level z_4. In other words, the 3214 pixels whose original gray level was 1/7 and that were remapped to gray level $s_1 = 2/7$ during the equalization step should now be shifted again to gray level $z_4 = 4/7$. Similarly, for the remaining values of s_k, the following mappings should be performed:

$s_0 = 0 \rightarrow z_2$
$s_1 = 2/7 \simeq 0.286 \rightarrow z_4$
$s_2 = 4/7 \simeq 0.571 \rightarrow z_5$
$s_3 = 5/7 \simeq 0.714 \rightarrow z_5$
$s_4 = 6/7 \simeq 0.857 \rightarrow z_6$
$s_5 = 1 \rightarrow z_7$
$s_6 = 1 \rightarrow z_7$
$s_7 = 1 \rightarrow z_7$

In this case, we have assumed that the algorithm for obtaining the inverse of the transformation function for a given value of s_k would search through the values of v_k and store the index of the most recent (i.e., *last*) value whose absolute difference ($|v_k - s_k|$) is the smallest so far. If the algorithm used another way of breaking ties, s_0 could map to z_0 or z_1 in this example. Table 9.4 summarizes the original and desired histograms, their corresponding cdfs, and the mapping process described above.

Table 9.5 presents the numerical values for the resulting histogram, where $\hat{p}(z_k)$—which was obtained by adding the contents of all rows in column $p(r_k)$ in Table 9.4 that match to each z_k—corresponds to the best possible approximation to the specified $p(z_k)$. For an easy visual comparison between the desired and the resulting histograms, they are plotted side by side in Figure 9.7.

You will probably agree that the resulting histogram approaches, within certain limits, the desired (specified) one.

■ EXAMPLE 9.7

Figure 9.8 shows an example (obtained using the `histeq` function in MATLAB) of direct histogram specification applied to a 600 × 800 image with 256 gray levels.

TABLE 9.5 Resulting Histogram

z_k ($\times 1/7$)	$\hat{p}(z_k)$
0	0.000
1	0.000
2	0.068
3	0.000
4	0.196
5	0.505
6	0.122
7	0.109
Total	1

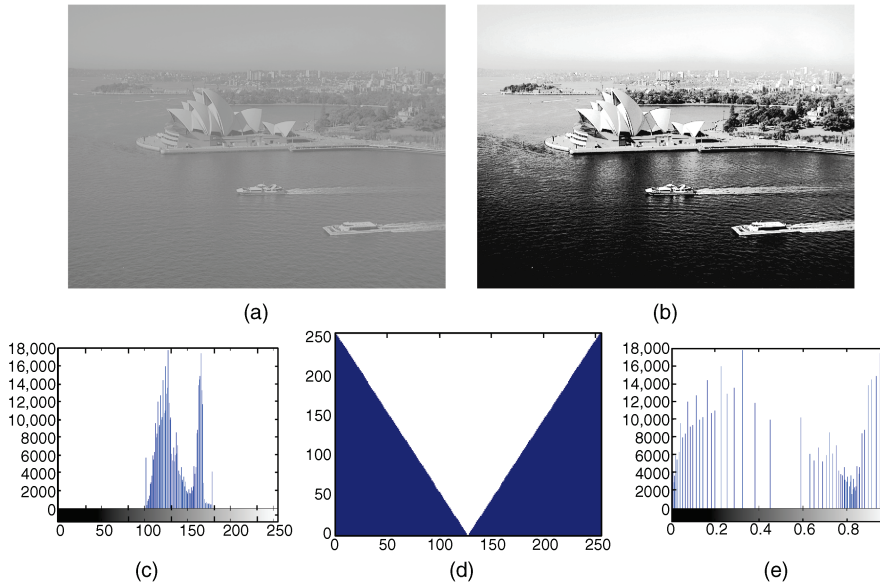

FIGURE 9.8 Histogram matching: (a) original image; (b) resulting image; (c) original histogram; (d) desired histogram; (e) resulting histogram.

9.6 OTHER HISTOGRAM MODIFICATION TECHNIQUES

In this section, we present other techniques by which a histogram can be processed to achieve a specific goal, such as brightness increase (or decrease) or contrast improvement. These techniques parallel some of the point transformation functions presented in Chapter 8, with the main difference being that whereas in Chapter 8 our focus was on the transformation function and its effect on the input image, in this chapter we also inspect the effect on the image's histogram.

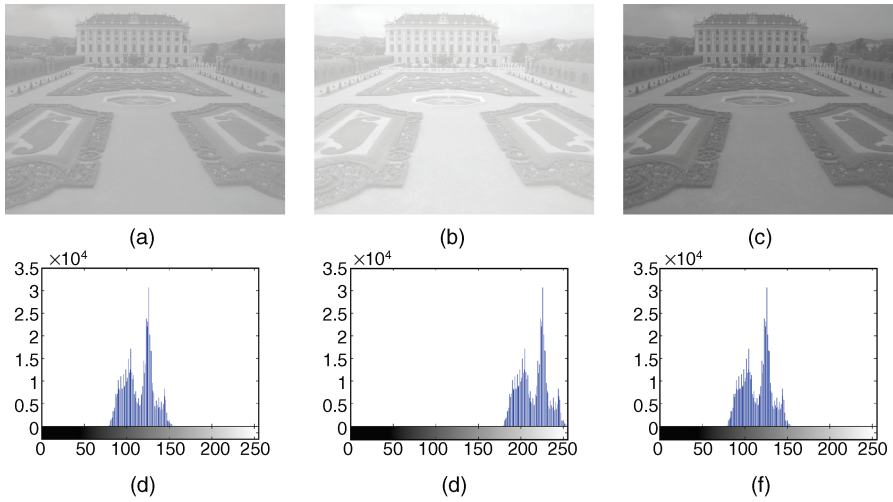

FIGURE 9.9 Histogram sliding: (a) original image; (b) result of sliding to the right by 50; (c) result of sliding to the left by 50; (d–f) histograms corresponding to images in (a)–(c).

9.6.1 Histogram Sliding

This technique consists of simply adding or subtracting a constant brightness value to all pixels in the image. The overall effect will be an image with comparable contrast properties, but higher or lower (respectively) average brightness.

In MATLAB

The `imadd` and `imsubtract` functions introduced earlier in the book can be used for histogram sliding.

■ EXAMPLE 9.8

Figure 9.9 shows an example of histogram sliding using `imadd` and `imsubtract`.

9.6.2 Histogram Stretching

This technique—also known as *input cropping*—consists of a linear transformation that expands (stretches) part of the original histogram so that its nonzero intensity range $[r_{\min},\ r_{\max}]$ occupies the full dynamic gray scale, $[0,\ L-1]$. Mathematically, each input intensity value, r, is mapped to an output value, s, according to the following linear mapping function:

$$s = \frac{r - r_{\min}}{r_{\max} - r_{\min}} \times (L-1) \tag{9.12}$$

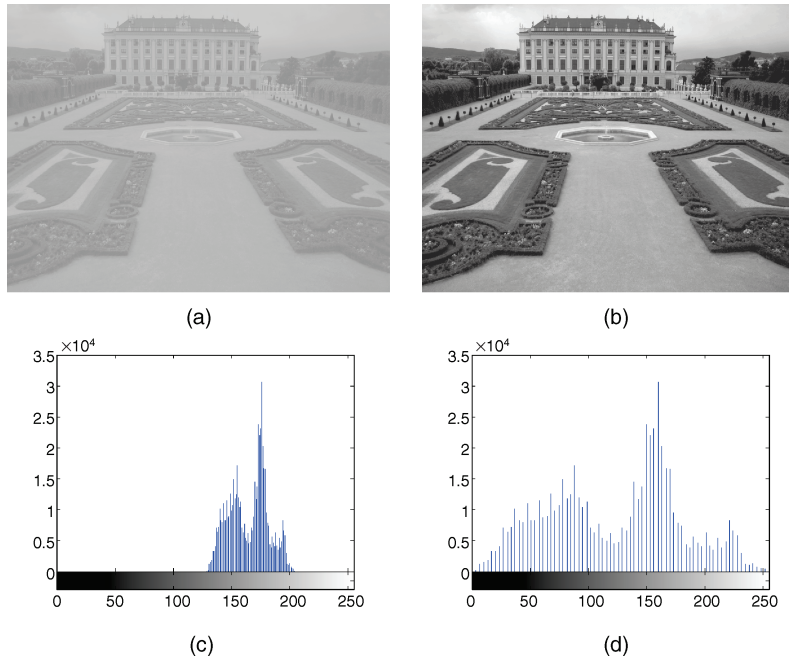

FIGURE 9.10 Example of using histogram stretching to improve contrast: (a) original image ($r_{\min} = 129$, $r_{\max} = 204$); (b) result of stretching using equation (9.12); (c and d) histograms corresponding to images in (a) and (b).

You may have noticed that equation (9.12) is identical to equation (8.4), introduced during our discussion of autocontrast in Chapter 8.

Histogram stretching increases contrast without modifying the shape of the original histogram. It is effective only in cases where the gray-level range of an image is compressed to a narrow range around the middle portion of the dynamic grayscale range. It will not be effective if applied to a poor contrast image whose histogram's lowest and highest occupied bins are close to the minimum and maximum of the full range, such as the one in Figure 9.2c.

■ EXAMPLE 9.9

Figure 9.10 shows an example (using the `imadjust` function in MATLAB) of using the histogram stretching technique to enhance image contrast.

9.6.3 Histogram Shrinking

This technique—also known as *output cropping*—modifies the original histogram in such a way as to compress its dynamic grayscale range, $[r_{\min}, r_{\max}]$, into a narrower

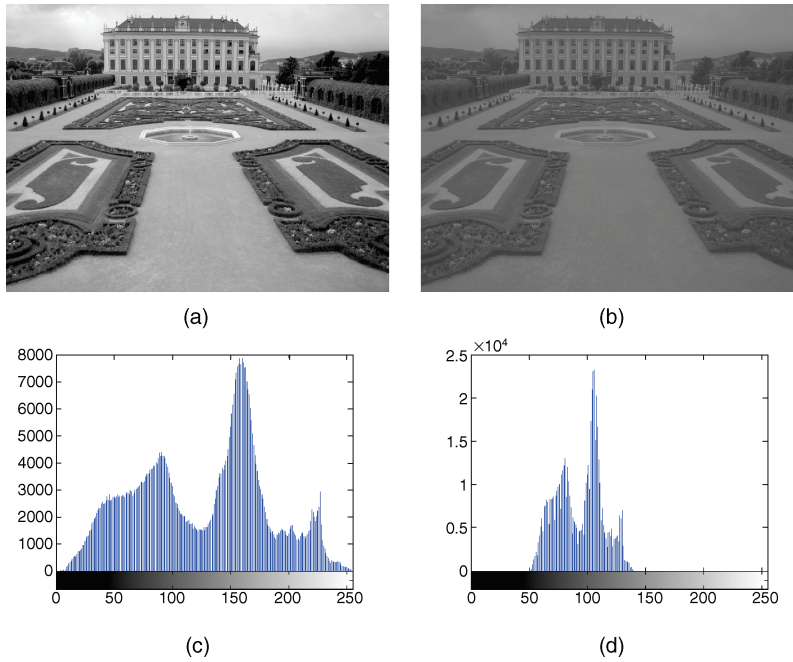

FIGURE 9.11 Example of using histogram shrinking to reduce contrast: (a) original image; (b) result of shrinking using equation (9.13) with $r_{\min} = 4$, $r_{\max} = 254$, $s_{\min} = 49$, and $s_{\max} = 140$; (c and d) histograms corresponding to images in (a) and (b).

gray scale, between $s_{\min}$ and $s_{\max}$. As a consequence, the resulting image contrast is reduced.

Mathematically,

$$s = \left[\frac{s_{\max} - s_{\min}}{r_{\max} - r_{\min}}\right](r - r_{\min}) + s_{\min} \tag{9.13}$$

■ EXAMPLE 9.10

Figure 9.11 shows an example (using the `imadjust` function in MATLAB) of applying the histogram shrinking technique to a monochrome image.

In MATLAB

MATLAB's IPT has a built-in function to perform histogram stretching and shrinking (among other operations): `imadjust`. You will learn how to use this function in Tutorial 9.3.

9.7 TUTORIAL 9.1: IMAGE HISTOGRAMS

Goal

The goal of this tutorial is to use MATLAB and IPT to calculate and display image histograms.

Objectives

- Learn how to use the IPT function `imhist`.
- Learn how other MATLAB plotting techniques can be used to view and analyze histogram data.

Procedure

Let us begin exploring the `imhist` function that is responsible for computing and displaying the histogram of an image.

1. Display an image and its histogram.

```
I = imread('circuit.tif');
figure, subplot(2,2,1), imshow(I), title('Image')
subplot(2,2,2), imhist(I,256), axis tight, title('Histogram')
```

2. The previous step displayed the default histogram for the image—a histogram with 256 bins. Let us see what happens if we change this value to 64 and 32.

```
subplot(2,2,3), imhist(I,64), axis tight, ...
    title('Histogram with 64 bins')
subplot(2,2,4), imhist(I,32), axis tight, ...
    title('Histogram with 32 bins')
```

You may have noticed that we set the axis to `tight` when displaying histograms. This adjusts the axis limits to the range of the data.

Question 1 Explain the drastic change of the Y-axis values when the histogram is displayed with fewer bins.

There may be a need to postprocess the histogram data or display it using other plotting techniques. To do this, we need the values for each bin of the histogram. The following step illustrates this procedure.

3. Get the values of each bin in the histogram for later use.

```
c = imhist(I,32);
```

We can now use the values in `c` to display histogram using other plotting techniques. Naturally, the plot of a histogram displays the count of each bin, but it may be more relevant to plot each bin's percentage. This can be done by normalizing the data, as shown in the next step.

4. Normalize the values in `c`.

```
c_norm = c / numel(I);
```

Question 2 What does the function `numel` do?

Question 3 Write a one line MATLAB statement that will verify that the sum of the normalized values add to 1.

5. Close any open figures.
6. Display the histogram data using a bar chart.

```
figure, subplot(1,2,1), bar_1 = bar(c);
set(gca, 'XLim', [0 32], 'YLim', [0 max(c)]);
```

In the previous step, we saw how the bar chart can be customized. In MATLAB, almost every object you create can be customized. When we create the bar chart, there is an *axes* object and a bar chart object displayed on the *axes* object. Here, the variable `bar_1` is set to the *bar chart* object so that we can reference it later for further customization. The `set` function allows us to change settings of a particular object. The first parameter of the `set` function is the object you wish to customize. In this case, the first object we customize is `gca`, which stands for *get current axes*. Here, we set the limits of the *X* and *Y* axes. Even though the limits have been set, the graph is still ambiguous because the tick marks on the *X* and *Y* axes do not reflect the limits.

7. Set the tick marks to reflect the limits of the graph.

```
set(gca, 'XTick', [0:8:32], 'YTick', ...
    [linspace(0,7000,8) max(c)]);
```

Now the tick marks reflect the limits of the data. We used the set function to change settings of the *current axes*, but we can just as easily use it to customize the bar chart.

8. Use the `set` function to change the color of the bar chart. Also, give the chart a title.

```
set(bar_1, 'FaceColor', 'r'), title('Bar Chart')
```

Question 4 How would we change the width of the bars in a bar chart?

Notice in the previous step how we used the bar chart object `bar_1` when changing settings. Similarly, we can display the normalized bar chart on the same figure using `subplot`.

9. Display the normalized bar chart and customize its display.

```
subplot(1,2,2), bar_2 = bar(c_norm);
set(gca, 'XTick', [0:8:32], 'YTick', ...
    [linspace(0,0.09,10) max(c_norm)])
xlim([0 32]), ylim([0 max(c_norm)])
title('Normalized Bar Chart')
set(bar_2, 'FaceColor', 'g')
```

Here, we made similar modifications as before. You may have noticed that we used `xlim` and `ylim` functions to set the limits of the axes. Sometimes there is more than one way to accomplish the same task, and this is an example of just that.*Stem charts* are similar to bar charts.

10. Close any open figures.
11. Display stem charts for both standard and normalized histogram data.

```
figure,
subplot(1,2,1), stem(c,'fill','MarkerFaceColor','red'), ...
    axis tight, title('Stem Chart')
subplot(1,2,2), stem(c_norm,'fill','MarkerFaceColor','red'), ...
    axis tight, title('Normalized Stem Chart')
```

In the previous step, we set visual properties of the stem charts by specifying the settings directly in the stem function call—we filled the marker and colored it red. We could have just as easily set a variable equal to the stem plot object and used the `set` function to make the changes.

Question 5 Explore the properties of stem charts. How can we make the lines dotted instead of solid?

Question 6 Alter the axes limits and tick marks to reflect the data being displayed in the stem plot.

The `plot` function will display the data by connecting each point with a straight line.

12. Display a plot graph for both standard and normalized histogram data.

```
figure, subplot(1,2,1), plot(c), axis auto, title('Plot Graph')
subplot(1,2,2), plot(c_norm), axis auto, ...
    title('Normalized Plot Graph')
```

Question 7 Explore the properties of plot graphs. In the above code, the points for each bin are visually lost within the graph line. How can we make the points bolder so that they are more visible?

9.8 TUTORIAL 9.2: HISTOGRAM EQUALIZATION AND SPECIFICATION

Goal

The goal of this tutorial is to learn how to use the IPT for (global and local) histogram equalization and histogram specification (matching).

Objectives

- Explore the process of histogram equalization.
- Learn how to use the `histeq` function.
- Learn how to perform histogram specification (matching).
- Explore the *Interactive Histogram Matching* demo.
- Learn how to perform local histogram equalization with the `adapthisteq` function.

What You Will Need

`ihmdemo.m`—interactive Histogram Matching demo M-file

Procedure

Let us begin by using the function `histeq` to perform histogram equalization on our own images, and by using the `imhist` function, we can view the histogram of the original and the adjusted image.

1. Display the image `pout` and its histogram.

```
I = imread('pout.tif');
figure, subplot(2,2,1), imshow(I), ...
    title('Original Image')
subplot(2,2,2), imhist(I), ...
    title('Original Histogram')
```

2. Use the `histeq` function to perform histogram equalization.

```
I_eq = histeq(I,256);
```

Question 1 Why must we include the second parameter (256) in the `histeq` function call?

3. Display the equalized image and its histogram.

```
subplot(2,2,3), imshow(I_eq), title('Equalized Image')
subplot(2,2,4), imhist(I_eq), title('Equalized Histogram')
```

Question 2 What is the effect of histogram equalization on images with low contrast?

4. Close any open figures and clear all workspace variables.
5. Execute the following code to perform histogram equalization on the `tire` image.

```
I = imread('tire.tif'); I_eq = histeq(I,256);
figure, subplot(2,2,1), imshow(I), title('Original Image')
subplot(2,2,2), imhist(I), title('Original Histogram')
subplot(2,2,3), imshow(I_eq), title('Equalized Image')
subplot(2,2,4), imhist(I_eq), title('Equalized Histogram')
```

Question 3 Based on the tire image's original histogram, what can be said about its overall brightness?

Question 4 How did histogram equalization affect the overall image brightness in this case?

Histogram equalization does not always perform well. As we will see in the next steps, it depends on the original image.

6. Close any open figures and clear all workspace variables.
7. Perform histogram equalization on the `eight` image.

```
I = imread('eight.tif'); I_eq = histeq(I,256);
figure, subplot(2,2,1), imshow(I), title('Original Image')
subplot(2,2,2), imhist(I), title('Original Histogram')
subplot(2,2,3), imshow(I_eq), title('Equalized Image')
subplot(2,2,4), imhist(I_eq), title('Equalized Histogram')
```

Question 5 Why was there such a loss in image quality after histogram equalization?

The transformation function for histogram equalization is simply the cdf of the original image.

8. Display the normalized cdf for the eight.tif image.

```
I_hist = imhist(I); tf = cumsum(I_hist); tf_norm = tf / max(tf);
figure, plot(tf_norm), axis tight
```

Question 6 What does the `cumsum` function do in the previous step?

9. The transformation function can also be obtained without using the `cumsum` function.

```
[newmap, T] = histeq(I);
figure, plot(T)
```

As we have learned, the histogram equalization process attempts to flatten the image histogram. Histogram specification (also known as *histogram matching*) tries to match the image histogram to a specified histogram. The `histeq` function can also be used for this operation.

10. Close any open figures and clear all workspace variables.
11. Prepare a subplot and display original image and its histogram.

```
img1 = imread('pout.tif');
figure, subplot(3,3,1), imshow(img1), title('Original Image')
subplot(3,3,2), imhist(img1), title('Original Histogram')
```

12. Display the image after histogram equalization for comparison.

```
img1_eq = histeq(img1); m1 = ones(1,256)*0.5;
subplot(3,3,4), imshow(img1_eq), title('Equalized Image')
subplot(3,3,5), imhist(img1_eq), title('Equalized Histogram')
subplot(3,3,6), plot(m1), title('Desired Histogram Shape'), ...
    ylim([0 1]), xlim([1 256])
```

13. Display matched image where the desired histogram shape is a straight line from (0, 0) to (1, 1).

```
m2 = linspace(0,1,256); img2 = histeq(img1,m2);
subplot(3,3,7), imshow(img2), title('Matched Image')
subplot(3,3,8), imhist(img2), title('Matched Histogram')
subplot(3,3,9), plot(m2), title('Desired Histogram Shape'), ...
    ylim([0 1]), xlim([1 256])
```

As we can see from the previous steps, performing histogram specification means we must generate a function that represents the shape of the desired histogram. The *Interactive Histogram Matching* demo (developed by Jeremy Jacob and available at

the book's companion web site) shows us how creating a desired histogram shape can be an interactive process.

14. Close any open figures and clear all workspace variables.
15. Run the *Interactive Histogram Matching* demo.

```
ihmdemo
```

16. Experiment with creating your own desired histogram shape. To create new points on the function curve, click the curve at the desired location. To move a point, press and drag the point. To delete a point, simply click it.

Question 7 What does the *Continuous Update* checkbox do?

Question 8 How do the different interpolation methods change the shape of the desired histogram curve?

Question 9 How can the demo be loaded with a different image?

Local histogram equalization is performed by the `adapthisteq` function. This function performs contrast limited adaptive histogram equalization (CLAHE) and operates on small data regions (called *tiles*), whose size can be passed as a parameter.

17. Perform local histogram equalization on the `coins` image.

```
I = imread('coins.png');
I_eq = histeq(I,256);
I_leq = adapthisteq(I,'ClipLimit',0.1);
figure, subplot(3,2,1), imshow(I), title('Original Image')
subplot(3,2,2), imhist(I), title('Original Histogram')
subplot(3,2,3), imshow(I_eq), title('Equalized Image')
subplot(3,2,4), imhist(I_eq), title('Equalized Histogram')
subplot(3,2,5), imshow(I_leq), ...
    title('Local Histogram Equalization')
subplot(3,2,6), imhist(I_leq), ...
    title('Local Hist Equalization Histogram')
```

The original image's histogram is clearly bimodal, which separates the pixels of the background from the pixels that make up the coins. We have already seen how images with bimodal distribution of pixel shades do not perform well under (global) histogram equalization.

Question 10 What does the `ClipLimit` setting do in the `adapthisteq` function?

Question 11 What is the default tile size when using `adapthisteq`?

9.9 TUTORIAL 9.3: OTHER HISTOGRAM MODIFICATION TECHNIQUES

Goal

The goal of this tutorial is to learn how to perform other common histogram modification operations.

Objectives

- Learn how to adjust brightness of an image by *histogram sliding*.
- Learn how to use the `imadjust` function.
- Learn how to use the `stretchlim` function.
- Explore adjusting image contrast through *histogram stretching* (also known as *input cropping*).
- Learn how to adjust image contrast with *histogram shrinking* (also known as *output cropping*).

Procedure

Histogram sliding is the process of adding or subtracting a constant brightness value to all pixels in the image. When implementing histogram sliding, we must make sure that pixel values do not go outside the boundaries of the gray scale. Therefore, any pixels that result in values greater than 1 after adjustment will be set to 1. Likewise, any pixels resulting in values less than zero after adjustment will be set to 0.

1. Display original image and prepare subplot.

```
J = imread('pout.tif');
I = im2double(J);
clear J
figure, subplot(3,2,1), imshow(I), title('Original Image')
subplot(3,2,2), imhist(I), axis tight, ...
    title('Original Histogram')
```

2. Obtain a brighter version of the input image by adding 0.1 to each pixel.

```
const = 0.1;
I2 = I + const;
subplot(3,2,3), imshow(I2), title('Original Image + 0.1')
subplot(3,2,4), imhist(I2), axis tight, ...
    title('Original Hist + 0.1')
```

Question 1 How did the histogram change after the adjustment?

3. Produce another brighter image by adding 0.5 to original image.

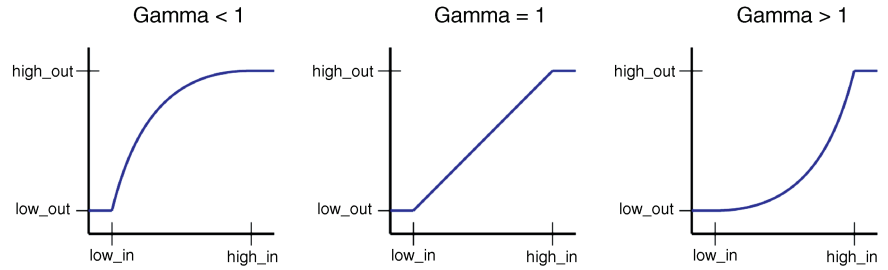

FIGURE 9.12 Gamma transformations for different values of gamma. Redrawn from [GWE04].

```
const = 0.5;
I3 = I + const;
bad_values = find(I3 > 1);
I3(bad_values) = 1;
subplot(3,2,5), imshow(I3), title('Original Image + 0.5')
subplot(3,2,6), imhist(I3), axis tight, ...
    title('Original Hist + 0.5')
```

Question 2 What does the variable `bad_values` contain?

Question 3 Why does the third plot show such an excessive number of pixels with a value of 1?

The brightness of an image can also be modified using the `imadd` function (introduced in Tutorial 6.1), which takes care of truncating and rounding off values outside the desired range in the output image.

Histogram stretching and shrinking can be achieved through use of the `imadjust` function. The syntax for the function is as follows:

```
J = imadjust(I,[low_in; high_in],[low_out; high_out], gamma)
```

Figure 9.12 illustrates what the transformation functions look like when different values of gamma are used. As we already know from Chapter 9, the value of gamma is the exponent in power law transformation.

Any values below `low_in` and above `high_in` are *clipped* or simply mapped to `low_out` and `high_out`, respectively. Only values in between these limits are affected by the curve. Gamma values less than 1 create a weighted curve toward the brighter range, and gamma values greater than 1 weight toward the darker region. The default value of gamma is 1.

Let us explore how to use `imadjust` to perform histogram stretching.

4. Close any open figures.
5. Execute the following code to see histogram stretching on the `pout` image, which is already loaded in variable `I`.

```
img_limits = stretchlim(I);
I_stretch = imadjust(I,img_limits,[]);
figure
subplot(3,2,1), imshow(I), title('Original Image')
subplot(3,2,2), imhist(I), axis tight, ...
    title('Original Histogram')
subplot(3,2,3), imshow(I_stretch), ...
    title('Stretched Image')
subplot(3,2,4), imhist(I_stretch), axis tight, ...
    title('Stretched Histogram')
```

Question 4 How did the histogram change after the adjustment?

Question 5 What is the purpose of using the `stretchlim` function?

In the previous step, we specified the `low_in`, `high_in`, `low_out`, and `high_out` parameters when calling the `imadjust` function when in fact the default operation is histogram stretching—meaning these parameters are not necessary to perform histogram stretching. Notice in the next step how just calling the function and only specifying the image as its parameter will give the same results.

6. Perform histogram stretching with `imadjust` using default parameters and confirm that the results are identical to the ones obtained before.

```
I_stretch2 = imadjust(I);
subplot(3,2,5), imshow(I_stretch2), ...
    title('Stretched Image')
subplot(3,2,6), imhist(I_stretch2), axis tight, ...
    title('Stretched Histogram')
I_stretch_diff = imabsdiff(I_stretch, I_stretch2);
figure, imshow(I_stretch_diff,[])
min(I_stretch_diff(:))
max(I_stretch_diff(:))
```

Question 6 How does the difference image look?

Question 7 What is the purpose of inspecting its maximum and minimum values?

To shrink an image histogram, we must specify the parameters explicitly.

7. Close any open figures and clear all workspace variables.
8. Execute the following code to see the result of histogram shrinking.

```
I = imread('westconcordorthophoto.png');
I_shrink = imadjust(I,stretchlim(I),[0.25 0.75]);
```

```
figure
subplot(2,2,1), imshow(I), title('Original Image')
subplot(2,2,2), imhist(I), axis tight, ...
    title('Original Histogram')
subplot(2,2,3), imshow(I_shrink), ...
    title('Shrunk Image')
subplot(2,2,4), imhist(I_shrink), axis tight, ...
    title('Shrunk Histogram')
```

When we use other techniques to adjust the histogram of an image, we have a means to view the transformation function (i.e., the `histeq` function will return the transformation function as an output parameter if requested). There is no built-in technique for viewing a transformation function when performing histogram sliding, stretching, or shrinking, but we can achieve a visual representation of the transformation function by using the `plot` function. To do so, we specify the original image as the *X* values and the adjusted image as the *Y* values.

9. Display the transformation function for the adjustment performed in the previous step.

```
X = reshape(I,1,prod(size(I)));
Y = reshape(I_shrink,1,prod(size(I_shrink)));
figure, plot(X,Y,'.')
xlim([0 255]); ylim([0 255]);
xlabel('Original Image');
ylabel('Adjusted Image');
```

Question 8 What do the above first two statements in the code do?

Question 9 What does the `xlabel` and `ylabel` functions do?

As noted earlier, gamma values other than 1 will specify the shape of the curve, toward either the bright or the dark region.

10. Close any open figures.
11. Perform histogram shrinking with a gamma value of 2.

```
I_shrink = imadjust(I,stretchlim(I),[0.25 0.75],2);
X = reshape(I,1,prod(size(I)));
Y = reshape(I_shrink,1,prod(size(I_shrink)));
figure
subplot(2,2,1), imshow(I), title('Original Image')
subplot(2,2,2), imhist(I), axis tight, ...
    title('Original Histogram')
subplot(2,2,3), imshow(I_shrink), title('Adjusted Image')
subplot(2,2,4), imhist(I_shrink), axis tight, ...
```

```
    title('Adjusted Histogram')
figure, plot(X,Y,'.'), xlim([0 255]), ylim([0 255])
```

Question 10 The transformation function plot displays a gap from 0 to 12 (on the X axis) where there are no points. Why is this so?

WHAT HAVE WE LEARNED?

- Histograms are a convenient mathematical representation of how many pixels occur at each gray level in a monochrome image.
- In MATLAB, the histogram of an image can be computed and displayed using the `imhist` function.
- Histograms provide valuable quantitative information about an image, such as minimum, maximum and average gray level, global standard deviation, and absolute contrast.
- Histograms also provide valuable qualitative information about an image, such as average brightness level and contrast.
- Histogram equalization is a mathematical technique by which we modify the histogram of the input image in such a way as to approximate a uniform distribution (flat histogram). Histogram equalization is used as a contrast enhancement technique.
- Histograms (and the images to which they correspond) can also be modified through *direct histogram specification*, whose goal is to change the histogram so as to match a desired shape.
- Other histogram modification techniques include histogram sliding (used to modify the brightness properties of an image), histogram stretching (used to increase contrast without modifying the shape of the original histogram), and histogram shrinking (used to reduce the dynamic grayscale range of an image).

LEARN MORE ABOUT IT

- Chapter 4 of [BB08] discusses histograms in detail and provides useful insights into how to use histograms to investigate image acquisition problems and image defects.
- The paper by Hummel [Hum75] provides a review of techniques related to the material in this chapter.
- Many alternative histogram modification techniques have been proposed in the literature. The paper by Stark [Sta00] is a representative recent example.
- Section 5.6 of [BB08] discusses histogram specification in more detail.
- Section 10.2 of [Pra07] provides a deeper discussion on histogram modification techniques.

- Section 2.2 of [SOS00] contains a modified version of the basic histogram expansion algorithm, in which the user can select a cutoff percentage p and a modified version of the general histogram transformation algorithm and the output histogram has the shape of a linear ramp whose slope can be specified as a parameter.
- Many image enhancement techniques based on local and global histogram statistics have been proposed. Section 3.3.4 of [GW08] and Section 8.2.3 of [Umb05] describe representative algorithms under this category.

9.10 PROBLEMS

9.1 The 7×7 image with eight gray levels is given below, where each gray level value is represented in normalized form from 0 (black pixel) to 1 (white pixel).

0	3/7	2/7	2/7	1/7	1/7	4/7
3/7	2/7	1/7	1/7	1/7	1/7	4/7
2/7	0	1	1/7	3/7	0	0
0	5/7	1/7	0	6/7	0	1/7
1/7	1/7	1/7	3/7	6/7	6/7	5/7
1/7	1/7	1/7	1/7	5/7	6/7	4/7
0	1	0	0	0	0	4/7

(a) Calculate the probabilities of each gray level and plot the image's histogram.

(b) Which pixels are predominant in the original image, dark or bright?

(c) Using the cumulative distribution function, equalize the histogram calculated in part (a) and plot the resulting (equalized) histogram.

(d) Show the resulting 7×7 image after histogram equalization.

9.2 Write a MATLAB script to show that a second (or third, fourth, etc.) consecutive application of the histogram equalization algorithm to an image will not produce any significant change in the histogram (and, consequently, in the image).

9.3 Given a 256×256 pixels image with eight gray levels, whose gray-level distribution is given in the following table.

Gray Level (r_k)	n_k	$p(r_k)$
0	2621	0.04
1/7	0	0.00
2/7	0	0.00
3/7	5243	0.08
4/7	7209	0.11
5/7	12,452	0.19
6/7	24,904	0.38
1	13,107	0.20

It is desired that the original histogram is changed to approach the histogram corresponding to the table below.

z_k	$\hat{p}(z_k)$
0	0.27
1/7	0.16
2/7	0.19
3/7	0.16
4/7	0.11
5/7	0.06
6/7	0.03
1	0.02

(a) Which pixels predominate in the original image, dark or bright? Explain.

(b) Assuming the histogram modification will be successful, what will be the probable effect of this modification on the original image?

(c) Equalize the original histogram using the function $s = T(r)$.

(d) Obtain the function $v = G(z)$ and its inverse.

(e) Plot the most relevant histograms: original, desired, equalized, and resulting.

(f) Fill out the table below with the final values for n_k and $\hat{p}(z_k)$ for the eight values of z_k, comparing with the desired values and explaining possible differences.

z_k	n_k	$\hat{p}(z_k)$
0		
1/7		
2/7		
3/7		
4/7		
5/7		
6/7		
1		

9.4 Write a MATLAB script that implements region-based histogram equalization. Your script should allow the user to interactively select (with the mouse) a region of interest (ROI) within an image to which the histogram equalization operation will be applied.

9.5

(a) Write a MATLAB function that creates an 8-bit random image with a uniform distribution of pixel values and takes two parameters: height and width. *Hint*: use MATLAB function `rand`.

(b) Write a MATLAB script that uses the function you have just written to create a 128×128 random image.

(c) Inspect the image's histogram. Does it show a uniform (i.e., flat-shaped) distribution as expected? Explain.

9.6

(a) Write a MATLAB function that creates an 8-bit random image with a Gaussian (normal) distribution of pixel values and takes four parameters: height, width, mean value (μ), and standard deviation (σ). *Hint*: use MATLAB function `randn`—which returns a pseudorandom normal distribution with $\mu = 0$ and $\sigma = 1$—and make the necessary adjustments to μ and σ.

(b) Write a MATLAB script that uses the function you have just written to create a 128×128 random image with a Gaussian (normal) distribution of pixel values, with mean value $\mu = 128$ and standard deviation $\sigma = 60$.

(c) Inspect the image's histogram. Does it show a Gaussian (i.e., bell-shaped) distribution as expected? Explain.

(d) Repeat the previous two steps for different values of μ and σ.

9.7 An 8-bit image has a minimum gray level of 140 and a maximum gray level of 195. Describe the effect on the histogram of this image after each of these operations is performed (separately):

(a) Subtraction of 130 from all pixel gray levels (histogram sliding).

(b) Histogram stretching.

(c) Histogram equalization.

9.8 Provide empirical evidence of the nonuniqueness of a histogram by writing a MATLAB script that reads a monochrome image, displays its histogram, and generates another gray-level image very different from the original, but whose histogram is identical to the original image's histogram. *Hint*: "Given an image f with a particular histogram H_f, every image that is a spatial shuffling of the gray levels of f has the same histogram H_f" [Bov00a].