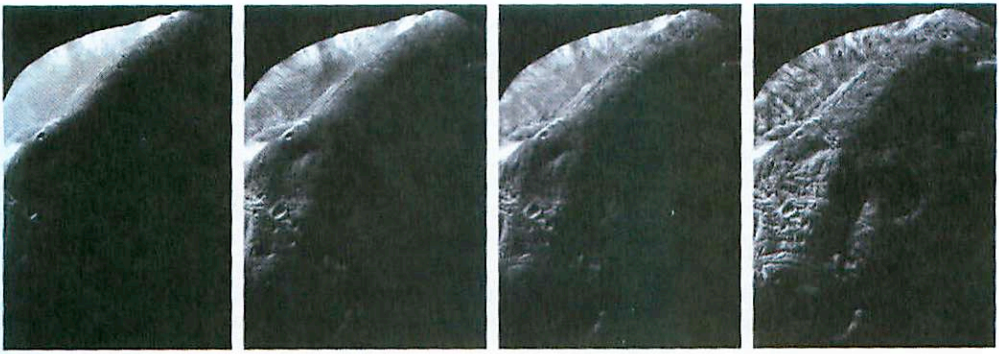

a b c d

FIGURE 3.12 (a) Same as Fig. 3.10(a). (b) Result of using function `adapthisteq` with the default values. (c) Result of using this function with parameter `NumTiles` set to `[25 25]`. Result of using this number of tiles and `ClipLimit` = `0.05`.

```
>> g3 = adapthisteq(f, 'NumTiles', [25 25], 'ClipLimit', 0.05);
```

yielded the result in Fig. 3.12(d). The enhancement in detail in this image is significant compared to the previous two results. In fact, comparing Figs. 3.12(d) and 3.11(b) provides a good example of the advantage that local enhancement can have over global enhancement methods. Generally, the price paid is additional function complexity. ■

3.4 Spatial Filtering

As mentioned in Section 3.1 and illustrated in Fig. 3.1, neighborhood processing consists of (1) selecting a center point, (x, y); (2) performing an operation that involves only the pixels in a predefined neighborhood about (x, y); (3) letting the result of that operation be the "response" of the process at *that* point; and (4) repeating the process for every point in the image. The process of moving the center point creates new neighborhoods, one for each pixel in the input image. The two principal terms used to identify this operation are *neighborhood processing* and *spatial filtering*, with the second term being more prevalent. As explained in the following section, if the computations performed on the pixels of the neighborhoods are linear, the operation is called *linear spatial filtering* (the term *spatial convolution* also used); otherwise it is called *nonlinear spatial filtering*.

3.4.1 Linear Spatial Filtering

The concept of *linear filtering* has its roots in the use of the Fourier transform for signal processing in the frequency domain, a topic discussed in detail in Chapter 4. In the present chapter, we are interested in filtering operations that

are performed directly on the pixels of an image. Use of the term *linear spatial filtering* differentiates this type of process from *frequency domain filtering*.

The linear operations of interest in this chapter consist of multiplying each pixel in the neighborhood by a corresponding coefficient and summing the results to obtain the response at each point (x, y). If the neighborhood is of size $m \times n$, mn coefficients are required. The coefficients are arranged as a matrix, called a *filter*, *mask*, *filter mask*, *kernel*, *template*, or *window*, with the first three terms being the most prevalent. For reasons that will become obvious shortly, the terms *convolution filter*, *convolution mask*, or *convolution kernel*, also are used.

Figure 3.13 illustrates the mechanics of linear spatial filtering. The process consists of moving the center of the filter mask, w, from point to point in an image, f. At each point (x, y), the response of the filter at that point is the sum of products of the filter coefficients and the corresponding neighborhood pixels in the area spanned by the filter mask. For a mask of size $m \times n$, we assume typically that $m = 2a + 1$ and $n = 2b + 1$ where a and b are nonnegative integers. All this says is that our principal focus is on masks of odd sizes, with the smallest meaningful size being 3×3. Although it certainly is not a requirement, working with odd-size masks is more intuitive because they have an unambiguous center point.

There are two closely related concepts that must be understood clearly when performing linear spatial filtering. One is *correlation*; the other is *convolution*. Correlation is the process of passing the mask w by the image array f in the manner described in Fig. 3.13. Mechanically, convolution is the same process, except that w is rotated by 180° prior to passing it by f. These two concepts are best explained by some examples.

Figure 3.14(a) shows a one-dimensional function, f, and a mask, w. The origin of f is assumed to be its leftmost point. To perform the correlation of the two functions, we move w so that its rightmost point coincides with the origin of f, as Fig. 3.14(b) shows. Note that there are points between the two functions that do not overlap. The most common way to handle this problem is to pad f with as many 0s as are necessary to guarantee that there will always be corresponding points for the full excursion of w past f. This situation is illustrated in Fig. 3.14(c).

We are now ready to perform the correlation. The first value of correlation is the sum of products of the two functions in the position shown in Fig. 3.14(c). The sum of products is 0 in this case. Next, we move w one location to the right and repeat the process [Fig. 3.14(d)]. The sum of products again is 0. After four shifts [Fig. 3.14(e)], we encounter the first nonzero value of the correlation, which is $(2)(1) = 2$. If we proceed in this manner until w moves completely past f [the ending geometry is shown in Fig. 3.14(f)] we would get the result in Fig. 3.14(g). This set of values is the correlation of w and f. If we had padded w, aligned the rightmost element of f with the leftmost element of the padded w, and performed correlation in the manner just explained, the result would have been different (rotated by 180°), so order of the functions matters in correlation.

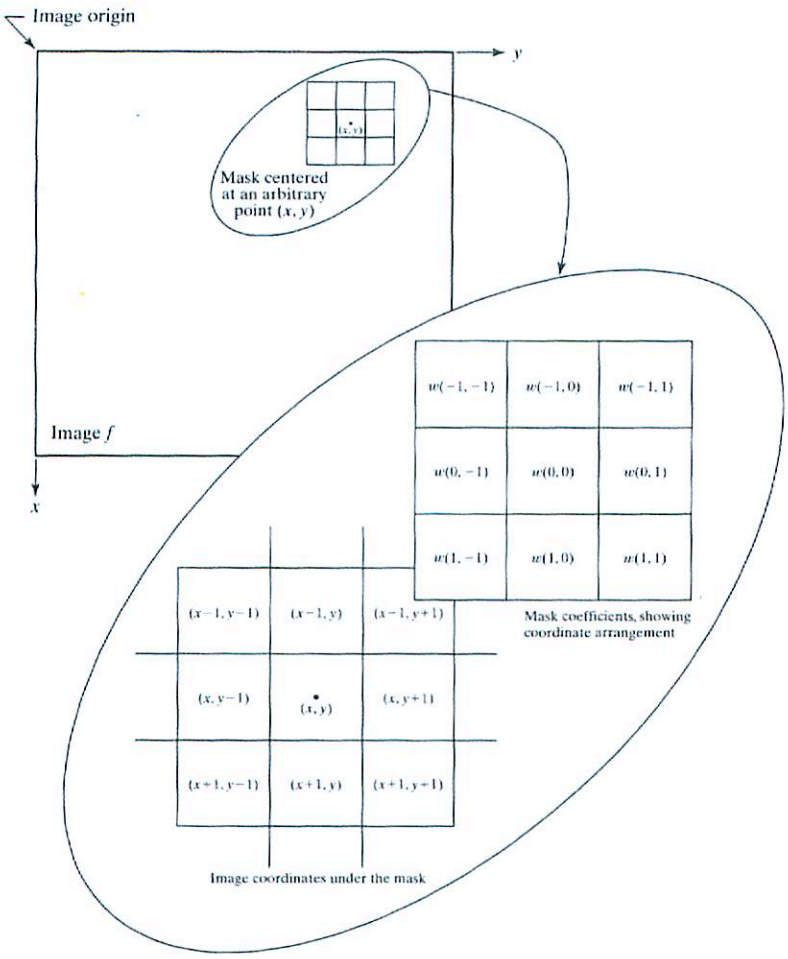

FIGURE 3.13 The mechanics of linear spatial filtering. The magnified drawing shows a 3×3 filter mask and the corresponding image neighborhood directly under it. The image neighborhood is shown displaced out from under the mask for ease of readability.

The label `'full'` in the correlation in Fig. 3.14(g) is a flag (to be discussed later) used by the toolbox to indicate correlation using a padded image and computed in the manner just described. The toolbox provides another option, denoted by `'same'` [Fig. 3.14(h)] that produces a correlation that is of the same size as f. This computation also uses zero padding, but the starting position is with the center point of the mask (the point labeled 3 in w) aligned with the origin of f. The last computation is with the center point of the mask aligned with the last point in f.

To perform convolution we rotate w by 180° and place its rightmost point at the origin of f, as Fig. 3.14(j) shows. We then repeat the sliding/computing

process employed in correlation, as illustrated in Figs. 3.14(k) through (n). The 'full' and 'same' convolution results are shown in Figs. 3.14(o) and (p), respectively.

Function f in Fig. 3.14 is a discrete unit impulse that is 1 at a point and 0 everywhere else. It is evident from the result in Figs. 3.14(o) or (p) that convolution with an impulse just "copies" w at the location of the impulse. This copying property (called *sifting*) is a fundamental concept in linear system theory, and it is the reason why one of the functions is always rotated by 180° in convolution. Note that, unlike correlation, swapping the order of the functions yields the same convolution result. If the function being shifted is symmetric, it is evident that convolution and correlation yield the same result.

The preceding concepts extend easily to images, as Fig. 3.15 illustrates. The origin is at the top, left corner of image $f(x, y)$ (see Fig. 2.1). To perform correlation, we place the bottom, rightmost point of $w(x, y)$ so that it coincides with the origin of $f(x, y)$ as in Fig. 3.15(c). Note the use of 0 padding for the

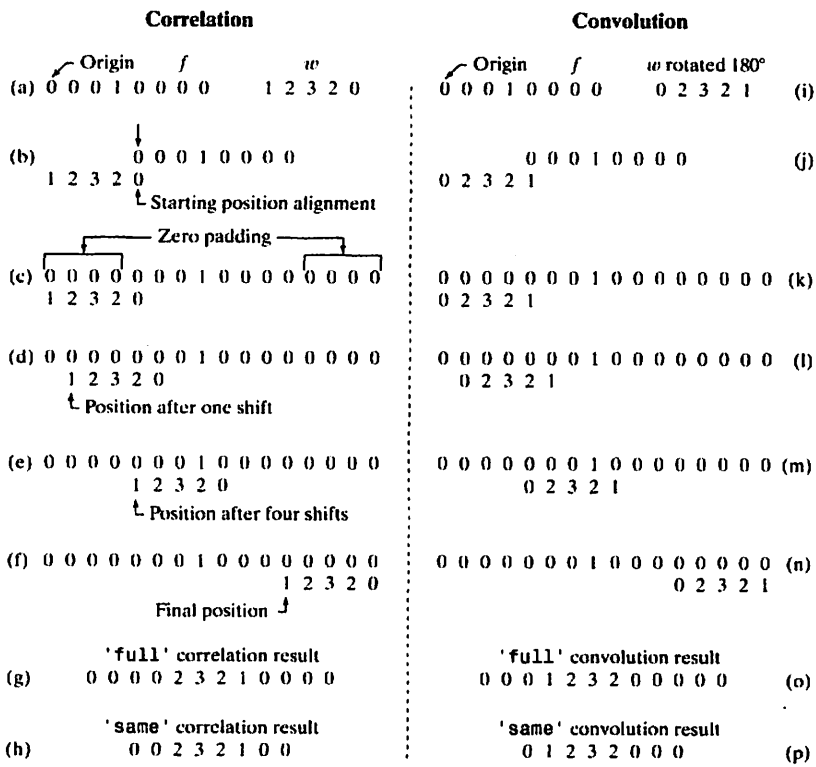

FIGURE 3.14 Illustration of one-dimensional correlation and convolution.

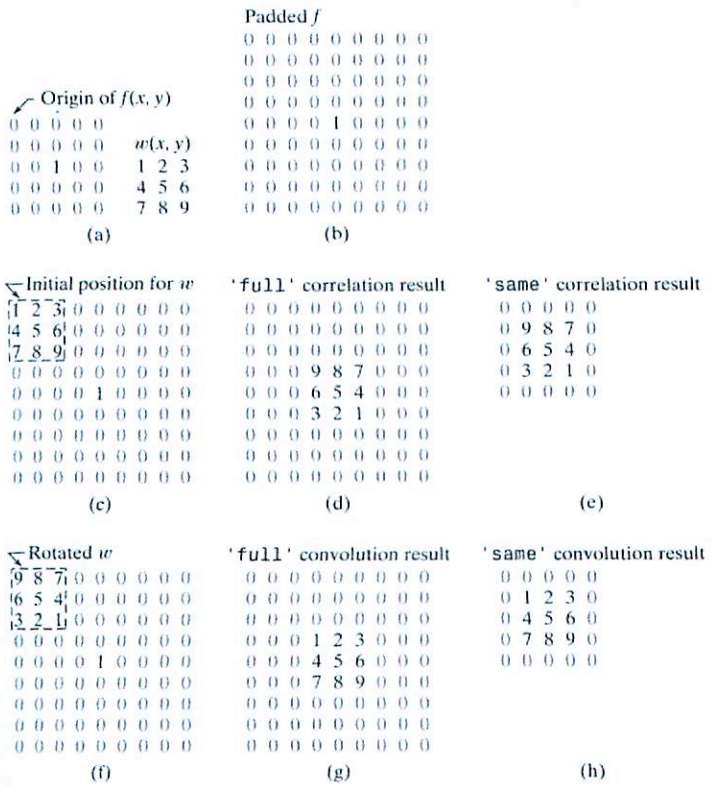

FIGURE 3.15 Illustration of two-dimensional correlation and convolution. The 0s are shown in gray to simplify viewing.

reasons mentioned in the discussion of Fig. 3.14. To perform correlation, we move $w(x, y)$ in all possible locations so that at least one of its pixels overlaps a pixel in the original image $f(x, y)$. This `'full'` correlation is shown in Fig. 3.15(d). To obtain the `'same'` correlation in Fig. 3.15(e), we require that all excursions of $w(x, y)$ be such that its center pixel overlaps the original $f(x, y)$. For convolution, we rotate $w(x, y)$ by 180° and proceed in the same manner as in correlation [see Figs. 3.15(f) through (h)]. As in the one-dimensional example discussed earlier, convolution yields the same result independently of the order of the functions. In correlation the order does matter, a fact that is made clear in the toolbox by assuming that the filter mask is always the function that undergoes translation. Note also the important fact in Figs. 3.15(e) and (h) that the results of spatial correlation and convolution are rotated by 180° with respect to each other. This, of course, is expected because convolution is nothing more than correlation with a rotated filter mask.

Summarizing the preceding discussion in equation form, we have that the correlation of a filter mask $w(x, y)$ of size $m \times n$ with a function $f(x, y)$, denoted by $w(x, y) \star f(x, y)$, is given by the expression

$$w(x, y) \star f(x, y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s, t) f(x + s, y + t)$$

This equation is evaluated for all values of the *displacement* variables x and y so that all elements of w visit every pixel in f, which we assume has been padded appropriately. Constants a and b are given by $a = (m - 1)/2$ and $b = (n - 1)/2$. For notational convenience, we assume that m and n are odd integers.

In a similar manner, the convolution of $w(x, y)$ and $f(x, y)$, denoted by $w(x, y) \star f(x, y)$, is given by the expression

$$w(x, y) \star f(x, y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s, t) f(x - s, y - t)$$

where the minus signs on the right of the equation flip f (i.e., rotate it by 180°). Rotating and shifting f instead of w is done to simplify the notation. The result is the same.[†] The terms in the summation are the same as for correlation.

The toolbox implements linear spatial filtering using function `imfilter`, which has the following syntax:


```
g = imfilter(f, w, filtering_mode, boundary_options, size_options)
```

where `f` is the input image, `w` is the filter mask, `g` is the filtered result, and the other parameters are summarized in Table 3.3. The `filtering_mode` is specified as `'corr'` for correlation (this is the default) or as `'conv'` for convolution. The `boundary_options` deal with the border-padding issue, with the size of the border being determined by the size of the filter. These options are explained further in Example 3.8. The `size_options` are either `'same'` or `'full'`, as explained in Figs. 3.14 and 3.15.

The most common syntax for `imfilter` is

```
g = imfilter(f, w, 'replicate')
```

This syntax is used when implementing standard linear spatial filters in the toolbox. These filters, which are discussed in Section 3.5.1, are prerotated by 180°, so we can use the correlation default in `imfilter` (from the discussion of Fig. 3.15, we know that performing correlation with a rotated filter is the same as performing convolution with the original filter). If the filter is symmetric about its center, then both options produce the same result.

[†] Because convolution is commutative, we have that $w(x, y) \star f(x, y) = f(x, y) \star w(x, y)$. This is not true of correlation, as you can see, for example, by reversing the order of the two functions in Fig. 3.14(a).

Options	Description
Filtering Mode	
`'corr'`	Filtering is done using correlation (see Figs. 3.14 and 3.15). This is the default.
`'conv'`	Filtering is done using convolution (see Figs. 3.14 and 3.15).
Boundary Options	
`P`	The boundaries of the input image are extended by padding with a value, `P` (written without quotes). This is the default, with value 0.
`'replicate'`	The size of the image is extended by replicating the values in its outer border.
`'symmetric'`	The size of the image is extended by mirror-reflecting it across its border.
`'circular'`	The size of the image is extended by treating the image as one period a 2-D periodic function.
Size Options	
`'full'`	The output is of the same size as the extended (padded) image (see Figs. 3.14 and 3.15).
`'same'`	The output is of the same size as the input. This is achieved by limiting the excursions of the center of the filter mask to points contained in the original image (see Figs. 3.14 and 3.15). This is the default.

TABLE 3.3 Options for function `imfilter`.

When working with filters that are neither pre-rotated nor symmetric, and we wish to perform convolution, we have two options. One is to use the syntax

```
g = imfilter(f, w, 'conv', 'replicate')
```

The other approach is to use function `rot90(w, 2)` to rotate `w` by 180°, and then use `imfilter(f, w, 'replicate')`. The two steps can be combined into one:

`rot90(w, k)` rotates `w` by `k*90` degrees, where `k` is an integer.

```
g = imfilter(f, rot90(w, 2), 'replicate')
```

The result would be an image, `g`, that is of the same size as the input (i.e., the default is the `'same'` mode discussed earlier).

Each element of the filtered image is computed using floating-point arithmetic. However, `imfilter` converts the output image to the same class of the input. Therefore, if `f` is an integer array, then output elements that exceed the range of the integer type are truncated, and fractional values are rounded. If more precision is desired in the result, then `f` should be converted to floating point using functions `im2single`, `im2double`, or `tofloat` (see Section 2.7) before using `imfilter`.