The fact that dilation and erosion are dual operations implies that erosion is identical to dilation of the background pixels. Eroding an image is identical to inverting it and then dilating it. This is illustrated by the bottom part of Figure H.35, where image $A$ is eroded into $B$, while image $C$ (the inverse of $A$) is dilated into $D$ (the inverse of $B$).

Erosion removes small-scale details from a binary image, while reducing regions of foreground pixels. A simple application of erosion is isolating the boundaries of foreground pixels in a binary image. This is done by eroding the image and then subtracting it from the original image.

Erosion and dilation do the opposite things to an image, but they are not exact inverses, which allows us to define two useful morphological operations on images, opening and closing. A morphological opening of an image is an erosion followed by a dilation with the same structuring element $H$. Closing an image is the reverse, a dilation followed by an erosion, also with the same structuring element.

The term "opening" is used because this operation can open up a gap between image regions connected by a thin bridge of pixels. Also, any image regions that have survived the erosion are restored to their original size by the dilation that follows. Opening is an idempotent operation; once an image has been opened, subsequent openings with the same structuring element do not change the image.

> Idempotence is the property of certain operations in mathematics and computer science, that can be applied multiple times without changing the result beyond the initial application.
>
> —From `wikipedia.com`.

The term "closing" is used because this operation can fill holes in the regions, while keeping the initial region sizes. Like opening, closing is idempotent. Closing and opening are dual operations, which explains the results shown in the bottom part of Figure H.35. Image $A$ is eroded with a $3 \times 3$ structuring matrix $H$ to produce image $B$. Similarly, image $C$, the inverse of $A$, is dilated with the reflected $H$ to produce image $D$, which happens to be the inverse of $B$ because closing and opening are dual.

## H.7 Convolution

The term convolution, as used in image processing, is the process of applying a filter to an image. In the general field of digital signal processing (DSP), convolution is the process of applying a filter to a signal. Many signals are one dimensional, but images are two dimensional. Thus, we will start with the one-dimensional convolution, and then extend it to two dimensions.

> Convolution
>
> 1. Something that is very complicated and difficult to understand: a twist or curve.
> 2. (In mathematics) a function derived from two given functions by integration that expresses how the shape of one is modified by the other.
> 3. An integral that expresses the amount of overlap of one function $g$ as it is shifted over another function $f$. It therefore "blends" one function with another. (From MathWorld—A Wolfram Web Resource.)
>
> —Dictionary definitions of convolution.

We start with a simple example. Given an array $a$ containing numbers, we want to smooth the differences between consecutive numbers. This can easily be done by replacing each number with the average of itself and its two immediate neighbors, an operation that can be written as

$$b[i] \leftarrow (a[i-1]+a[i]+a[i+1])/3 = a[i-1]h[-1]+a[i]h[0]+a[i+1]h[1] = \sum_{j=-1}^{1} a[i+j]h[j], \tag{H.7}$$

where index $i$ varies over all the elements of $a$. Figure H.36 illustrates two steps of this process. Array $h$, the convolution kernel (or the filter kernel), contains $1/3$ in each of its three elements. The kernel is also often referred to as the system's impulse response.

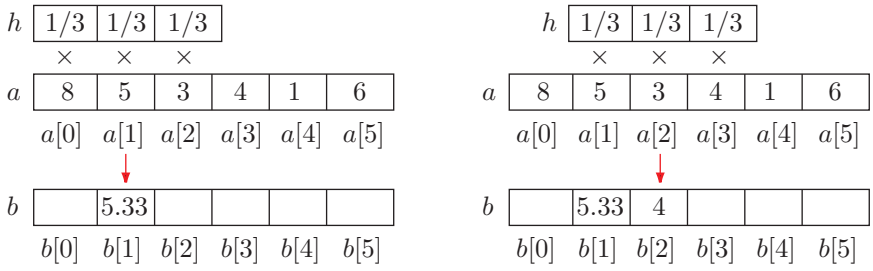


Figure H.36: Smoothing as a Convolution.

For reasons that will be explained shortly, it is customary to assign descending index values to the convolution kernel $h$ and label its elements $h[1]$, $h[0]$, and $h[-1]$, which changes Equation (H.7) to

$$b[i] \leftarrow a[i-1]h[1] + a[i]h[0] + a[i+1]h[-1] = \sum_{j=i-1}^{i+1} a[j]h[i-j]. \tag{H.8}$$

The star symbol $\star$ is used to indicate convolution, so Equation (H.8) is normally written as

$$b[i] = (a \star h)[i] = \sum_{j} a[j]h[i-j], \tag{H.9}$$

where the summation index $j$ varies over a range that includes all the nonzero elements of the kernel $h$. In technical jargon we say that the output array (or output signal) $b$ equals the input signal $a$ convolved with the impulse response $h$.

The ends (or edges) of the signal array $a$ always present a problem and may cause wrong results, because some signal values are missing. In our example, the computation of $b[0]$ requires the value of $a[-1]$ and that of $b[5]$ requires the value of $a[6]$. These values do not exist, which is why practical implementations of convolution employ solutions such as the following:

- Do not compute convolution values for $b[0]$, $b[5]$ or any other cases where input values are missing.
- Assume that any missing values are zeros.
- Copy the nearest value to cover any missing values. In our example, element $a[-1]$ would become 8 ($a[0]$) and element $a[6]$ would be 6 ($a[5]$).
- Assume that the signal array is periodic. The last element $a[5]$ of our array would be followed by $a[0]$ and the first element $a[0]$ would be preceded by $a[5]$.

Perhaps the best way to understand convolution is to work out a detailed example twice, first from the point of view of the input signal $a$ and then from the viewpoint of the output signal $b$. The former view illustrates the relation between convolution and signal processing, while the latter view clarifies the mathematical operations of convolution and explains why it may be interpreted as a weighted sum. Our example consists of the input signal $a = (-0.5, 0, -0.8, -1.2, -1.2, -2, 0.2, 1, 0)$ and the kernel $h = (0.1, 0.3, 0.5, -1)$.

The input side process illustrates how each input value (or input sample) contributes to the final output. It consists of two steps as follows:

- Multiply each of the eight input values by the four elements of the kernel, creating four numbers, shifted as illustrated in Figure H.37. Each of the first nine parts of the figure displays the impulse response contributed by one of the elements of the input signal $a$. These impulse responses are shifted and are also scaled by the kernel $h$. Notice that the last three steps create numbers with indexes 9, 10, and 11. Thus, the length of the output in our example is 12. In general the length of the output created by this variant of convolution is one less than the sum of the lengths of the input and the kernel.
- Add the eight results. Notice that each consists of 12 numbers, of which eight are zeros and four are the result of a multiplication in step 1. The final result is also shown graphically in Figure H.37.

The eight results are

$$(0.05, 0.15, 0.25, -0.5, 0, 0, 0, 0, 0, 0, 0, 0)$$
$$(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)$$
$$(0, 0, -0.08, -0.24, -0.4, 0.8, 0, 0, 0, 0, 0, 0)$$
$$(0, 0, 0, -0.12, -0.36, -0.6, 1.2, 0, 0, 0, 0, 0)$$
$$(0, 0, 0, 0, -0.12, -0.36, -0.6, 1.2, 0, 0, 0, 0)$$

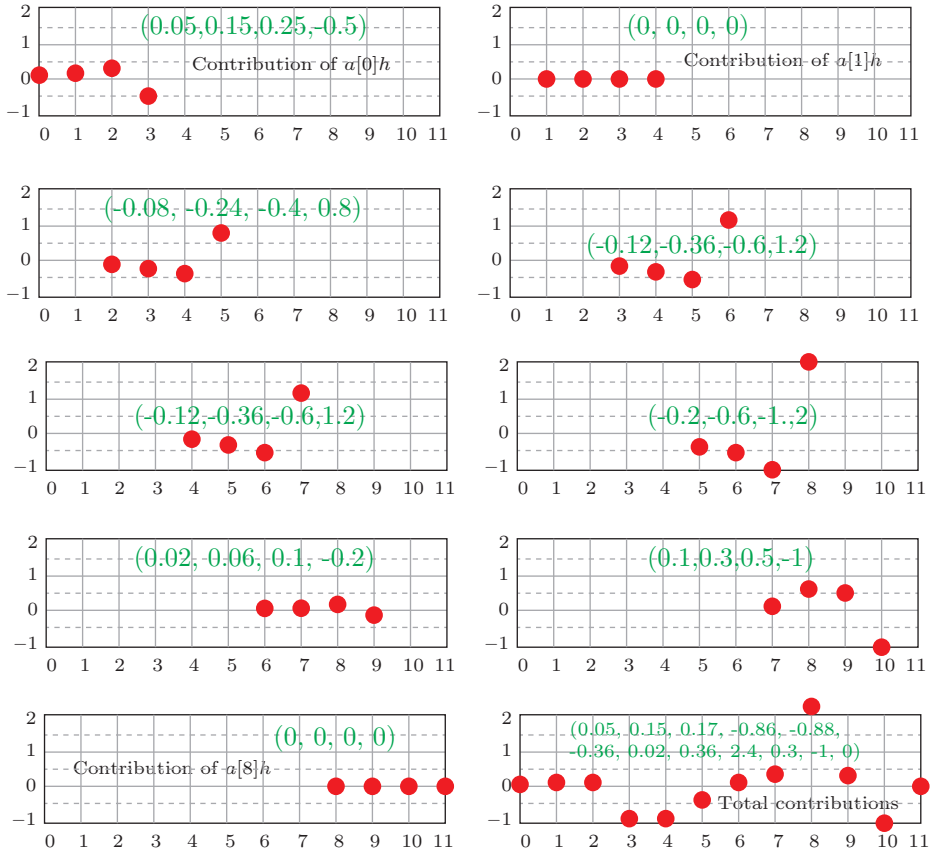


Figure H.37: Input-Side Convolution Example.

$$(0,0,0,0,0,-0.2,-0.6,-1.,2,0,0,0)$$
$$(0,0,0,0,0,0,0.02,0.06,0.1,-0.2,0,0)$$
$$(0,0,0,0,0,0,0,0.1,0.3,0.5,-1,0)$$
$$(0,0,0,0,0,0,0,0,0,0,0,0),$$

and their 12-point sum, the final output, is

$$b = (0.05, 0.15, 0.17, -0.86, -0.88, -0.36, 0.02, 0.36, 2.4, 0.3, -1, 0).$$

The two steps involve only multiplications and additions, which are commutative. Those who carefully follow the above example will agree that convolution is commutative. Exchanging the input and kernel would result in the same 12-point final output. However, even though the mathematics is simple and is commutative, there is a physical difference between the input and the kernel. The former is given by the physical problem at hand, while the latter is specified by the user, based on his experience and intuition.

Now, for the output-side viewpoint. This process examines the output values $b[]$ to figure out what input samples $a[i]$ contribute to each output value $b[j]$. This knowledge is useful because any practical software for convolution should consist of a loop where each iteration computes one output value $b[j]$ from several input samples and from all the values of the kernel. Each value $b[j]$ should be computed independently of the other output values. We use the above example to figure out how the sixth output value $b[5] = -0.36 = 0.8 - 0.6 - 0.36 - 0.2$ was obtained. This value is the sum

$$\begin{aligned} &a[2]\cdot h[3] + a[3]\cdot h[2] + a[4]\cdot h[1] + a[5]\cdot h[0] \\ &= -0.8\cdot(-1) - 1.2\cdot 0.5 - 1.2\cdot 0.3 - 2\cdot 0.1 \\ &= \sum_{i=2}^{5} a[i]h[5-i]. \end{aligned}$$

From this result we conclude that the general output value $b[j]$ is computed as the sum

$$b[j] = \sum_{i=j-3}^{j} a[i]h[j-i], \tag{H.10}$$

similar to Equation (H.8). (The value 3 in the subscript is the length of the kernel, minus 1.)

Figure H.38 is a graphic representation of this process. An element of the output $b$ is computed as the dot product of part of the input $a$ and a left-to-right flipped version of the kernel $h$. The flipped kernel (in the dashed box) is then slid to the right and the process is repeated for all the elements of the input $a$.

**Edge effects**. It is easy to implement convolution from Equation (H.10), but for extreme values of the output index $j$, the results are wrong. The sum of Equation (H.10) goes from $i = j - 3$ to $j$, so for $j = 0$, index $i$ should vary from $-3$ to 0, but input samples $a[-3]$, $a[-2]$, and $a[-1]$ do not exist, so the sum reduces to the single product $a[0]h[0]$. (The technical term for this is that the impulse response is not fully immersed in the input signal.) Similarly, for $j = 1$, index $i$ should vary from $-2$ to 1, but the actual result is only $a[0]h[1] + a[1]h[0]$. A similar situation exists for the largest values of $j$, where some input samples do not exist and the sum can be only partly computed. Several methods to handle edge effects have been discussed earlier.

The mathematical aspect of convolution can now be summarized as follows. Each input sample contributes a scaled and shifted version of the impulse response (the kernel) to the final output. Conversely, each output value is a contribution of several input samples, each multiplied by a value from the flipped impulse response.

While true, this summary does not tell us why convolution is so useful in image processing or in the more general field of digital signal processing (DSP). We can begin to grasp its importance when we again examine Equation (H.10) and Figure H.38. They both illustrate how each output value is a weighted sum of several input samples, with the elements of the kernel as the weights. The very first example in this section employs a kernel, each of whose three elements equals 1/3. This kernel generates output values each of which is the average of three input samples. Viewing convolution as a weighted sum is the key to understanding its operation and how to use it in practice.

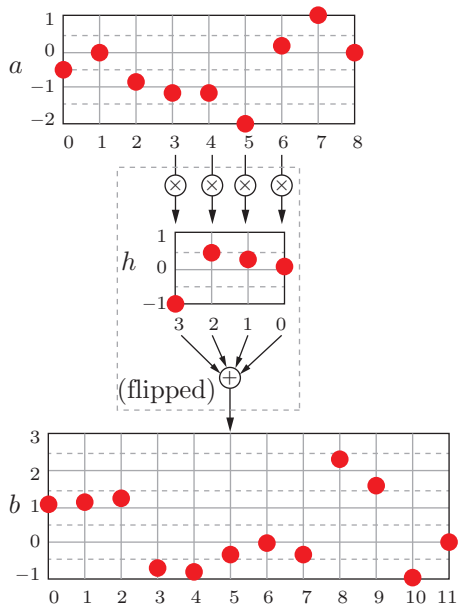


Figure H.38: Output-Side Convolution As a Dot Product.

## H.7.1 2D Convolution

Image processing is concerned with images, and a digital image is a two-dimensional array of pixels. The method of convolution can be directly extended to such arrays. All that is needed is a two-dimensional kernel. The kernel is flipped twice, vertically and horizontally, and then it is slid over the entire image row by row to create the output image. This can be summarized by the equation

$$b[m,n] = a[m,n] \star h[m,n] = \sum_j \sum_i a[i,j]h[m-i,n-j].$$

As an example, given a $3 \times 3$ kernel whose rows and columns are labeled from $-1$ to 1, this equation results in the sum

$$\begin{aligned} b[1,1] =& \sum_j \sum_i a[i,j]h[1-j,1-i] \\ =& a[0,0]h[1,1] + a[1,0]h[0,1] + a[2,0]h[-1,1] \\ &+ a[0,1]h[1,0] + a[1,1]h[0,0] + a[2,1]h[-1,0] \\ &+ a[0,2]h[1,-1] + a[1,2]h[0,-1] + a[2,2]h[-1,-1]. \end{aligned}$$

Figure H.39 is an example of five typical filters obtained by a two-dimensional convolution. The original image shows the Daigoji Temple in Kyoto (compare with Figure H.3) and the filters were achieved by the kernels

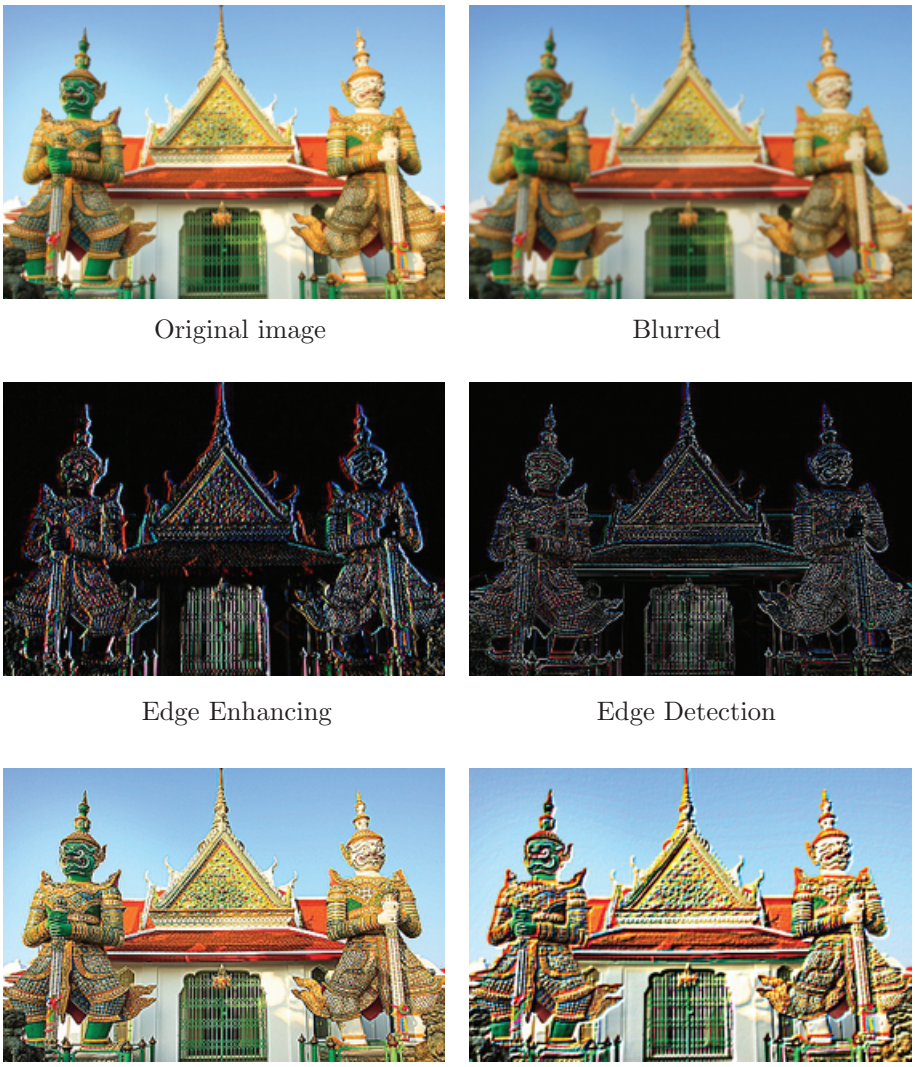


Figure H.39: Five Filters as 2D Convolutions.

| | | | | |
|---|---|---|---|---|
| | 1 | 1 | 1 | |
| | 1 | 1 | 1 | |
| | 1 | 1 | 1 | |

Blur

| | | | | |
|---|---|---|---|---|
| | 0 | 0 | 0 | |
| | -1 | 1 | 0 | |
| | 0 | 0 | 0 | |

Edge enhance

| | | | | |
|---|---|---|---|---|
| | 0 | 1 | 0 | |
| | 1 | -4 | 0 | |
| | 0 | 1 | 0 | |

Edge detect

| | | | | |
|---|---|---|---|---|
| | 0 | -1 | 0 | |
| | -1 | 5 | -1 | |
| | 0 | -1 | 0 | |

Sharp

| | | | | |
|---|---|---|---|---|
| | -2 | -1 | 0 | |
| | -1 | 1 | 1 | |
| | 0 | 1 | 2 | |

Emboss