Introduction to Convolution

Recall that a grayscale intensity image can be represented as a matrix of numbers, where the numbers are called pixel values. Computational methods in image processing use mathematical operations on these pixel values to manipulate (e.g., rotate, translate, zoom in/out, enhance, restore) images. For example, we could brighten certain areas of an image by increasing the intensity of the corresponding pixel values. In this set of notes we consider an important image processing computation that occurs in many applications: *convolution*.

1 Convolution

A dictionary defines convolution as twisting or rolling together. In image processing the term is used to mean a particular way of combining two images, one called the input image, the other called the kernel image. To be precise, each pixel of the output (combined) image is a weighted sum of neighboring pixels of the input image. The weights are defined by the kernel image. Instead of writing a mathematical formula, we provide an example that illustrates the convolution operation.

Let I denote the input image, and K denote the kernel. In particular, suppose:

$$I = \begin{array}{|c|c|c|c|c|}\hline 17 & 24 & 1 & 8 & 15 \\ \hline 23 & 5 & 7 & 14 & 16 \\ \hline 4 & 6 & 13 & 20 & 22 \\ \hline 10 & 12 & 19 & 21 & 3 \\ \hline 11 & 18 & 25 & 2 & 9 \\ \hline \end{array} \quad \text{and} \quad K = \begin{array}{|c|c|c|}\hline 8 & 1 & 6 \\ \hline 3 & 5 & 7 \\ \hline 4 & 9 & 2 \\ \hline \end{array}$$

An important part of convolution is that we need to specify the "center" (or origin) of the kernel. By convention, for odd dimension images, it is usually the middle pixel value. Thus, in our example, 5 is the center of the kernel image.

With this notation, the convolution operation is performed as:

1. Rotate the kernel, K, 180 degrees about the center to get

$$\tilde{K} = \begin{array}{|c|c|c|}\hline 2 & 9 & 4 \\ \hline 7 & 5 & 3 \\ \hline 6 & 1 & 8 \\ \hline \end{array}$$

2. Place $\tilde{K}$ on top of I, so that the center of $\tilde{K}$ lies on top of a specific pixel element of I.
3. Multiply each weight in $\tilde{K}$ by the pixel from I underneath it.
4. Sum up the individual products to get a pixel element of the output image.

For example:

- The (2,2) pixel of the output image is obtained by putting the center of $\tilde{K}$ over the (2,2) entry in I as follows:

17 ^{2}	24 ^{9}	1 ^{4}	8	15
23 ^{7}	5 ^{5}	7 ^{3}	14	16
4 ^{6}	6 ^{1}	13 ^{8}	20	22
10	12	19	21	3
11	18	25	2	9

  Then, multiplying and adding, the (2,2) pixel of the output image is:

$$17 * 2 + 24 * 9 + 1 * 4 + 23 * 7 + 5 * 5 + 7 * 3 + 4 * 6 + 6 * 1 + 13 * 8 = 595.$$

- The (2,3) pixel of the output image is obtained by putting the center of $\tilde{K}$ over the (2,3) entry in I as follows:

17	24 ^{2}	1 ^{9}	8 ^{4}	15
23	5 ^{7}	7 ^{5}	14 ^{3}	16
4	6 ^{6}	13 ^{1}	20 ^{8}	22
10	12	19	21	3
11	18	25	2	9

  Then, multiplying and adding, the (2,3) pixel of the output image is:

$$24 * 2 + 1 * 9 + 8 * 4 + 5 * 7 + 7 * 5 + 14 * 3 + 6 * 6 + 13 * 1 + 20 * 8 = 410.$$

Try to compute the (4,3) pixel of the output image by hand.

Now we must decide how to calculate the pixel values near the border of the image. For example, if we want to use this method to find the (1,1) pixel of the output image, then we must put the center of $\tilde{K}$ over the (1,1) entry in I as follows:

2 9 4

7 17 ^{5}	24 ^{3}	1	8	15
6 23 ^{1}	5 ^{8}	7	14	16
4	6	13	20	22
10	12	19	21	3
11	18	25	2	9

But in this case the kernel overlaps empty space! To overcome this inconsistency, we need to define a *boundary condition*. There are several options we could use:

- Only compute the pixel values for which the result is completely defined (i.e., **valid**). If the kernel is very small compared to the image, then only a small number of pixels near the border of the image are discarded. However, if the kernel is the same dimension as the input image, then there is only one valid pixel value in the output image.

- Other approaches usually assume particular values for the empty space. The most common is to assume the missing values are **zero**. That is:

$0^{\,2}$	$0^{\,9}$	$0^{\,4}$			
$0^{\,7}$	$17^{\,5}$	$24^{\,3}$	1	8	15
$0^{\,6}$	$23^{\,1}$	$5^{\,8}$	7	14	16
	4	6	13	20	22
	10	12	19	21	3
	11	18	25	2	9

  Multiplying and adding, the (1,1) pixel of the output image is then:

$$0 * 2 + 0 * 9 + 0 * 4 + 0 * 7 + 17 * 5 + 24 * 3 + 0 * 6 + 23 * 1 + 5 * 8 = 220.$$

- **Periodic** boundary conditions is another popular approach. In this case, the missing elements are filled in by wrapping the pixel elements of the image around to the opposite side. That is:

$9^{\,2}$	$11^{\,9}$	$18^{\,4}$			
$15^{\,7}$	$17^{\,5}$	$24^{\,3}$	1	8	15
$16^{\,6}$	$23^{\,1}$	$5^{\,8}$	7	14	16
	4	6	13	20	22
	10	12	19	21	3
	11	18	25	2	9

  Thus, multiplying and adding, the (1,1) pixel of the output image is then:

$$9 * 2 + 11 * 9 + 18 * 4 + 15 * 7 + 17 * 5 + 24 * 3 + 16 * 6 + 23 * 1 + 5 * 8 = 610.$$

  Although periodic boundary conditions may seem a bit odd, there are very good reasons for using it. The reasons are based on engineering and computational issues that are best left for more advanced courses.

- Finally, **reflexive** boundary conditions simply assume that the missing pixels are filled in by reflecting the pixel values in I. That is,

$17^{\,2}$	$17^{\,9}$	$24^{\,4}$			
$17^{\,7}$	$17^{\,5}$	$24^{\,3}$	1	8	15
$23^{\,6}$	$23^{\,1}$	$5^{\,8}$	7	14	16
	4	6	13	20	22
	10	12	19	21	3
	11	18	25	2	9

  Thus, multiplying and adding, the (1,1) pixel of the output image is then:

$$17 * 2 + 17 * 9 + 24 * 4 + 17 * 7 + 17 * 5 + 24 * 3 + 23 * 6 + 23 * 1 + 5 * 8 = 760.$$

2 MATLAB's Convolution Function

MATLAB provides a function called `conv2` which can be used for convolving images. By reading `help conv2` we see that this function may be used as follows:

- `O = conv2(I, K, 'valid');` will compute only those pixel values that are completely defined, as described above.
- `O = conv2(I, K, 'same');` will use zero boundary conditions as described above.

Though `conv2` does not allow for other boundary conditions, we will discuss later how (and why) these can be used.

3 An Application of Convolution: Edge Detection

Convolution operations are used in many applications, including reconstruction of images taken with indirect imaging techniques (as in CT, MRI and ultrasound), restoration of images degraded by defocusing, motion blur, or similar errors during image acquisition, as well as in detecting simple local structures such as edges in an image. In class we will see how convolution can be used for simple *edge detection*, and later we will consider the restoration of blurred images.

The problem of edge detection is to determine locations in an image where there is a sudden variation in the gray level. These sudden changes can sometimes be detected by convolving the image with certain kernels; in particular, the following are often used:

$$K_1 = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad K_2 = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}, \quad K_2 = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}.$$

Each of the kernels K_1, K_2 and K_3 have different properties, and are used for different purposes. To see how each of these perform in detecting edges, we will conduct some experiments in class with MATLAB.