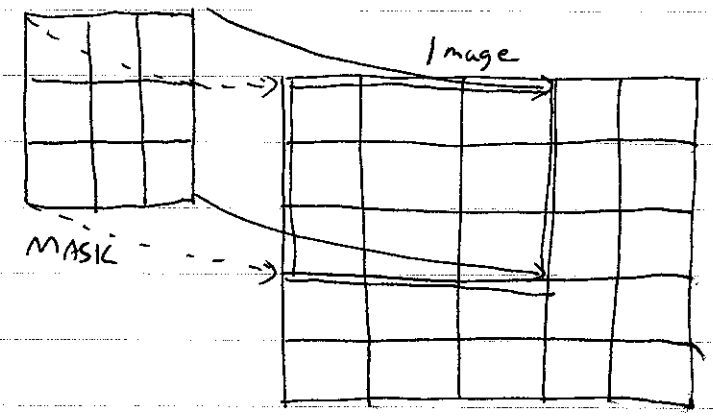
Convolution and filters

One fundamental mathematical operation on data is that of convolution, where we derive a new image by sliding a box of various properties across the original image. This box is often called a convolution mask, and different masks permit different operations on the image.

First, let's define convolution mathematically and then see how we can use it to enhance or modify images. Pictorially we can imagine convolution as follows:



We overlay the mask on the upper left corner of the image, so that it overlaps part of the original image. We then multiply the corresponding image-mask pair, and add all the products. This produces one number, which represents the value of the convolution for this particular relative location of the image and the mask.

To illustrate, let the mask and image be

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

mask

$$\begin{bmatrix} 2 & 4 & 3 & 5 \\ 5 & 3 & 4 & 2 \\ 2 & 5 & 2 & 4 \\ 3 & 3 & 4 & 2 \end{bmatrix}$$

Image

The initial value of the convolution is the sum

$$1\cdot 2 + 1\cdot 4 + 1\cdot 3 + 1\cdot 5 + 1\cdot 3 + 1\cdot 4 + 1\cdot 2 + 1\cdot 5 + 1\cdot 2 = 30$$

Next, slide the convolution mask one pixel to the right:

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \qquad \begin{bmatrix} 2 & 4 & 3 & 5 \\ 5 & 3 & 4 & 2 \\ 2 & 5 & 2 & 4 \\ 3 & 3 & 4 & 2 \end{bmatrix}$$

mask

and the sum becomes

$$1\cdot 4 + 1\cdot 3 + 1\cdot 5 + 1\cdot 3 + 1\cdot 4 + 1\cdot 2 + 1\cdot 5 + 1\cdot 2 + 1\cdot 4 = 32$$

Then we can "slide" the mask to all other possible positions over the input image, and record each convolution sum as a pixel in a new image, where each position represents a relative offset between the input and the mask. The output image is called the convolution of the input and the mask.

Convolution does not have to be two-dimensional, and in fact we often employ it in 1-D analyses. Here both the input and output images are one-dimensional:

MASK [1 1] INPUT [1 2 1 1 2 3 1 2 4 7 1 2 3]

[3 3 3 5 4 . . . .]

One thing you will notice immediately is that the input and output images or sequences are not the same length. In fact for an input sequence of length N and a mask length M, the number of output points is $N-M+1$. For example, let $N = 5$, $M = 2$, then the convolution will be of length 4:

[1 1] [1 2 3 4 5] (length 5)

MASK INPUT

[3 5 7 9] (length 4)

OUTPUT

If we want to keep image sizes constant, we have to have special rules to deal with the ends of the sequence or edges of the image. A common approach is to assume that there are enough zeroes off the ends to allow the output image to have the same size as the input.

Formal definition

The formal definition of convolution for a 1-D sequence is given by

$$g(x) = \int_{-\infty}^{\infty} f(x-x')\, m(x')\, dx'$$

where f and g are continuous input and output functions, respectively, and $m(x)$ is a mask function, also continuous. For discrete systems, with sampled data,

$$g_i = \sum_{j=-\infty}^{\infty} f_{i-j}\, m_j$$

where m_i, f_i, and g_i are the sequences of the mask,

the input, and the output. In the definition m_i is allowed to be of infinite length, but for a finite mask we can write it instead as

$$g_i = \sum_{j=1}^{M} f_{i-j}\, m_j$$

One thing we have to be careful with is to note that the j index has opposite sign when used with f and with m. What this means is that the mask function m is "flipped", or time-reversed, before the cross-multiplication takes places. This is related to generalizations of convolution that follow from filter theory, among others. For now, we will accept the reversal as a definition. (There is in fact a non-flipped cousin to convolution called correlation, which we will discuss later).

Note that if the mask is symmetrical, the flipping does not matter.

For images in two dimensions, we can define convolution as

$$g_{ij} = \sum_{i'=-\infty}^{\infty} \sum_{j'=-\infty}^{\infty} f_{i-i',\, j-j'}\; m_{i',j'}$$

Again a finite number of points implies a finite number of terms in the sums.

Application to interpolation

One immediate application of convolution is to implement the expansion of images we talked about previously. Start with our earlier example of enlarging an image by 2:

$$\begin{bmatrix} 8 & 4 & 8 \\ 4 & 8 & 4 \\ 8 & 2 & 8 \end{bmatrix}$$

original

$$\begin{bmatrix} 8 & 0 & 4 & 0 & 8 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 4 & 0 & 8 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 8 & 0 & 2 & 0 & 8 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

zero rows and columns added

Now, to implement the zero-order hold, simply convolve the larger matrix with the zero-order hold mask

$$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

mask

$$\begin{bmatrix} 8 & 0 & 4 & \cdots \\ 0 & 0 & 0 & \\ 4 & 0 & 8 & \\ \vdots & & & \end{bmatrix} \Rightarrow \begin{bmatrix} 8 & 8 & 4 & 4 & \cdots \\ 8 & 8 & 4 & 4 & \\ 4 & 4 & 8 & 8 & \\ 4 & 4 & 8 & 8 & \\ \vdots & & & & \end{bmatrix}$$

Here we have assumed a row of zeroes above and a column of zeroes to the left.

The first-order hold, which averages to obtain intermediate pixels, is obtained with the following mask

first-order hold mask:

$$\begin{bmatrix} \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ \frac{1}{2} & 1 & \frac{1}{2} \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \end{bmatrix}$$

Note on notation: Rather than write out the integral or sum definitions of convolution, many authors use the asterisk "*" to represent convolution. To avoid confusion, sometimes we use a single asterisk for 1-D convolution, and a double for 2-D:

$$g_{i,j} = \sum\sum f_{i-i',j-j'}\, m_{i',j'}$$

is equivalent to $g = f ** m$

The same * notation is used for the integral form if the functions are continuous.

Simple filters

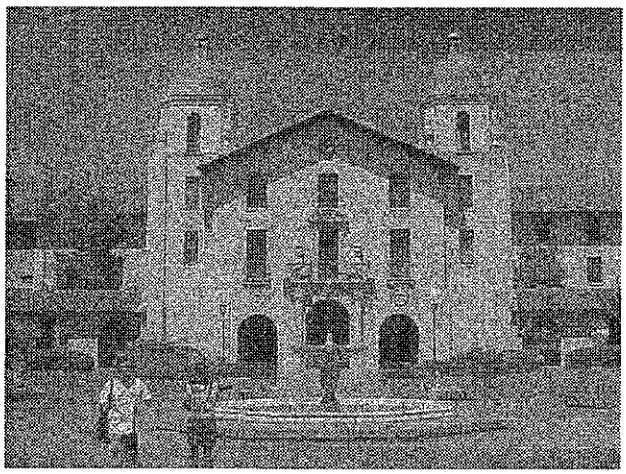
Very often images contain unwanted signals, or noise, in addition to the desired data. While we will discuss noise in greater detail later, it causes each value in an image to contain errors. In many cases these errors are sometimes positive and sometimes negative, so that on average they tend to cancel out. Simple convolutional filters allow us to clean up images by averaging nearby points.

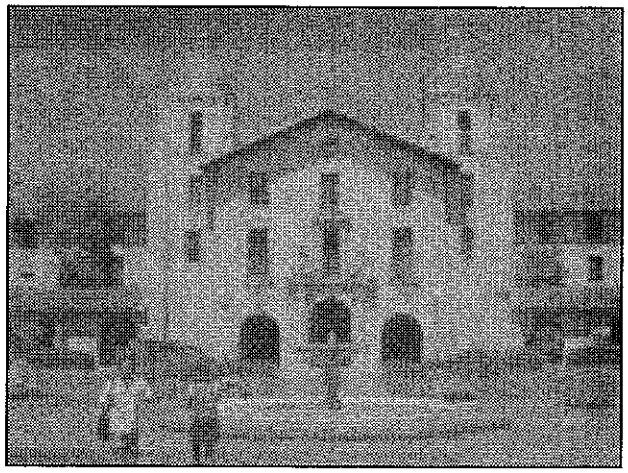
Suppose we pick as our convolution mask

$$\begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix}$$

where each point is assigned a value of one-ninth. If this

convolved with an image, each 3×3 box will be replaced by its average, retaining the image information while averaging out the noise. Thus the image will appear much "cleaner", that is less noisy.





One consequence of this operation is that the convolution tends to blur the underlying image because each single point is spread out over a 3×3 box. Thus there is always a trade-off between noise reduction and resolution of the output image. There is no hard and fast rule for choosing the "best" value, it changes for every application.

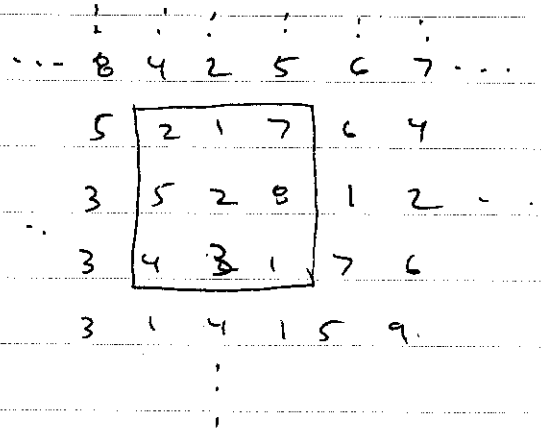
We can choose any size box as needed, and can choose unequal values for the mask pixels, if we wish. When we introduce frequency-domain techniques we will discuss how to design masks.

Median filters

Closely related to the above mean-value filters is the

median-value filter. This is not a convolutional filter, as it can not be represented by a convolution integral or sum. But it is implemented as a sliding window, similarly to the convolutional case.

For the median filter, we select a box of a subset of pixels in an input image



The resulting values are ranked in order:

[1 1 2 2 3 4 5 7 8]

↑

center, or median, value

Then the central value is used as the result of the filter. As before, the box slides from one position to another and the result at each position forms a pixel in the image output.

Median filtering is helpful when the noise at a few pixels is very large. The few pixels that have unusual values will always be far from the median and never be chosen.

Edge-detection

Another application of convolutional filters is to enhance features in an image. One such is the example of edge detection, algorithms that automatically identify sharp transitions in an image. These are often fundamental operations in computer vision programs.

The way most of these work is to recognize large changes in brightness by using a mask that calculates the difference, rather than the sum, of neighboring pixels. A simple mask to do this in the across direction is

$$\begin{bmatrix} 1 & -1 \\ 1 & -1 \end{bmatrix}$$

For the down direction, we could use $\begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix}$

Each of these finds edges in one direction only, so several must be applied to find arbitrary shapes. There is a great deal of literature on this subject and many operators that

can identify edges have been devised. Some of the more common are:

Sobel Operators: The row + column masks for Sobel are

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \text{ for rows, } \quad \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \text{ for columns}$$

In the Sobel method, the masks form two images from convolution which we can denote g_r (for rows) and g_c (for columns). Then the magnitude of the edge is given by

$$\text{magnitude} = \sqrt{g_r^2 + g_c^2} \qquad \text{(evaluated at each point)}$$

and the direction perpendicular to the edge is obtained from

$$\text{direction} = \tan^{-1}\left[\frac{g_r}{g_c}\right]$$

Prewitt operators: These are similar to the Sobel, but with masks

$$\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ +1 & +1 & +1 \end{bmatrix} \text{ for rows, } \quad \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \text{ for columns}$$

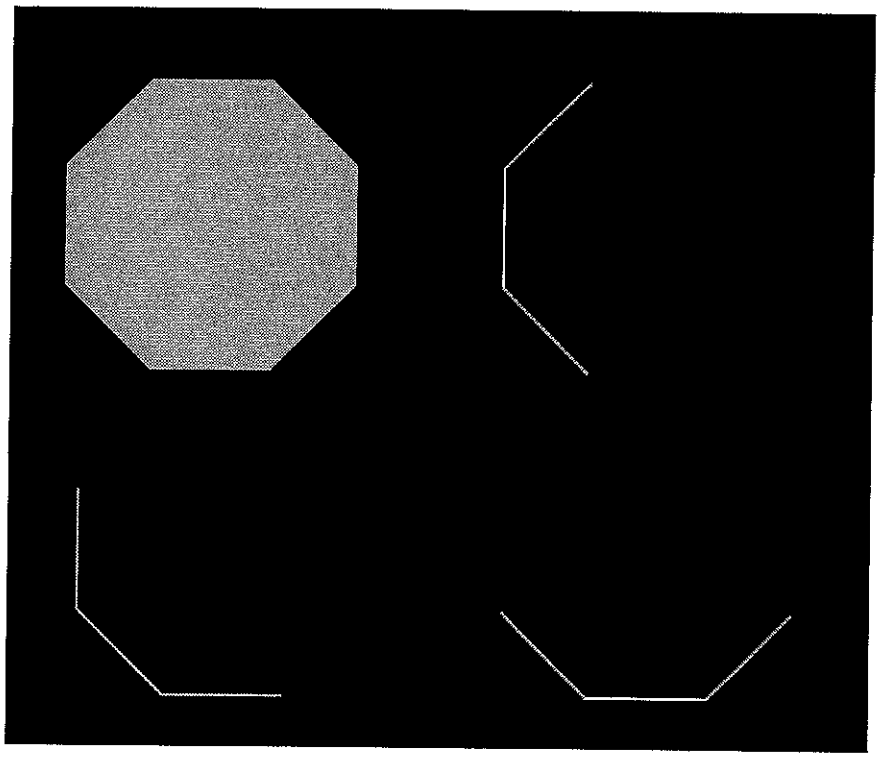
Kirsch Compass Masks: Another approach is to use masks designed to find eight possible edge orientations, and simply select the maximum convolution sum for each point to get the direction. So eight convolutions are calculated, and a single image for peak magnitudes and ~~directions~~ another for directions are created. The eight masks are rotated versions

of a single mask:

$$0: \begin{bmatrix} -3 & -3 & 5 \\ -3 & 0 & 5 \\ -3 & -3 & 5 \end{bmatrix} \quad 1: \begin{bmatrix} -3 & 5 & 5 \\ -3 & 0 & 5 \\ -3 & -3 & -3 \end{bmatrix} \quad 2: \begin{bmatrix} 5 & 5 & 5 \\ -3 & 0 & -3 \\ -3 & -3 & -3 \end{bmatrix} \quad 3: \begin{bmatrix} 5 & 5 & -3 \\ 5 & 0 & -3 \\ -3 & -3 & -3 \end{bmatrix}$$

$$4: \begin{bmatrix} 5 & -3 & -3 \\ 5 & 0 & -3 \\ 5 & -3 & -3 \end{bmatrix} \quad 5: \begin{bmatrix} -3 & -3 & -3 \\ 5 & 0 & -3 \\ 5 & 5 & -3 \end{bmatrix} \quad 6: \begin{bmatrix} -3 & -3 & -3 \\ -3 & 0 & -3 \\ 5 & 5 & 5 \end{bmatrix} \quad 7: \begin{bmatrix} -3 & -3 & -3 \\ -3 & 0 & 5 \\ -3 & 5 & 5 \end{bmatrix}$$

One can see the eight cardinal compass directions easily in the above masks. Some sample images formed by these masks are in the text.



Sample use of Robinson masks

Robinson Compass masks. Another set of compass masks was devised by Robinson:

0: $\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$ 1: $\begin{bmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & -1 & 0 \end{bmatrix}$, and so on.

Again the magnitude is defined as the convolutional image with the greatest value, with direction corresponding to the mask used.

Laplacian masks. A symmetrical type of edge-detection mask comes from a discrete form of the Laplacian operator, which calculates a vector of partial derivatives in both directions. This operator gives edge enhancement without respect to the direction of the edge:

$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$, $\begin{bmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{bmatrix}$, or $\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$

Each of these has mean zero, so that if applied to a uniform region its response is zero, as we want an edge detector to do. If we want to retain the image and still ~~enchor~~ enhance the edge, we can simply add "1" to the central value. Then the filter returns the actual grey level for a uniform area.

Enhancement Laplacians:

$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$, etc.