Neighborhood Processing
(Image Filtering)

Image filtering is done through a process called convolution. The process involves moving a kernel matrix over the source image (image to be processed – input image). Pixels are processed systematically using the contents of the kernel matrix and the source image to produce pixels of the resulting image. Image filters are used for image denoising, sharpening, blurring, edge detection and other similar effects.

Convolution
Convolution is a commonly used operation in signal (1D) and image (2D) processing. It works with two signals or images: the input and the kernel. In the one-dimensional case, if the input is **x** and the kernel is **f**, the convolution of **x** and **f** is given by:

$$(x * f)(i) = \sum_{j=1}^{m} f(j)\, x(i - j + 1) \qquad (i >= m)$$

To understand this better, let’s take a numerical example of the 1D case:
x:

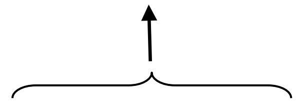
0	0	0	10	0	0	0	0	0

f:

0.25	.5	0.25

x*f (conv(**x**, **f**) in Matlab):

0	0	0	2.5	5	2.5	0	0	0	0	0

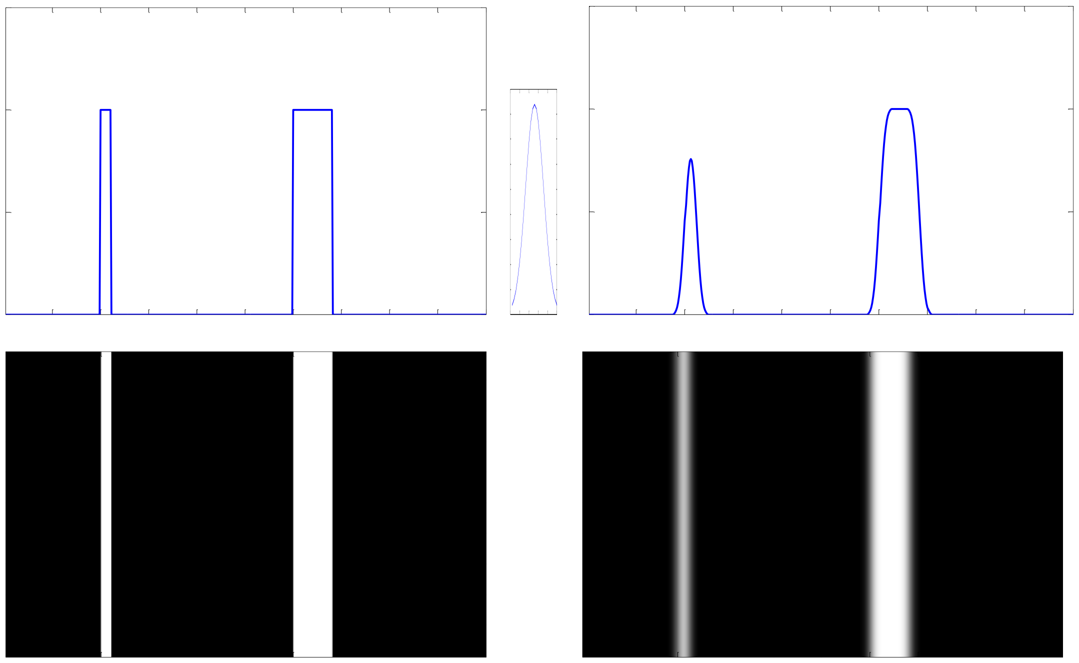


0	0	0	10	0	0	0	0	0

0.25	.5	0.25

1-D Example of Filtering
Assume we have the following information (top left graph). This shows an intensity vs. space graph of two stripes. This information can be viewed as an image by constructing a matrix in which the same intensity pattern is replicated for all rows. The corresponding image is shown below the graph. The top middle graph shows a 1-D kernel. When the kernel is applied (convolved with the top left graph) the graph on the right is produced. The smoothing effect can be seen in both the edges of the upper plot and the

corresponding image below it. This is an example of filtering in 1-D only since we assumed that there was no change in intensity in the vertical direction. Obviously, this scheme would not filter images in the vertical direction and therefore needs to be extended to 2-D for use with real images.



2-D Image Filtering

A kernel is a small NxN matrix that is applied to an image where N is usually odd and the dimensions of the image are much larger than N. The kernel is also known as the convolution mask or convolution kernel. It is multiplied elementwise by an NxN section in the image and then all elements are added to obtain the pixel value in the resultant image.

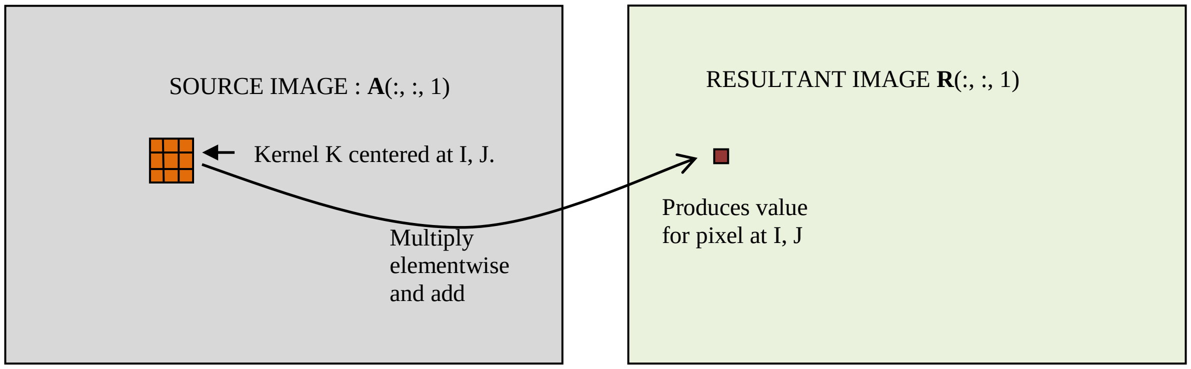
For example, given an RGB image A and a 3x3 kernel K (with N=3,) the element at row I and column J in the resultant image (R) would be determined by the following relationship for a single color channel c:

$$R(I,J,c)=\sum_{i=I-1}^{I+1}\sum_{j=J-1}^{J+1}A(i,j,c)K(i-I+2,j-J+2)$$

K being indexed from 1 (MATLAB convention). Note that you need to pay attention to the boundary conditions for I and J when constructing the loops to cover all pixels.

More formally this operation is called correlation rather than convolution which involves flipping the kernel in both dimensions. Since our kernels are symmetric or at least reversible we choose to use the simpler formulation.

The following illustration shows the operation for a single pixel for the red color channel.


SOURCE IMAGE : **A**(:, :, 1)
Kernel K centered at I, J.
Multiply elementwise and add
RESULTANT IMAGE **R**(:, :, 1)
Produces value for pixel at I, J

Note that centering the kernel means positioning the kernel such that K(2,2) corresponds to the image pixel at I,J.

This filtering process is then applied to the blue and green channels using the same kernel.

Pseudocode for Image Filtering

```
Load original image, A.
Create an empty resultant image, R, the same size as A, type uint8.

Choose the kernel size N (3, 5, or 7…)
Construct an NxN kernel, K, type double by default (see below for kernel contents)

for I in rows of A (less N/2 from the edges)
  for J in columns of A  (less N/2 from the edges)
    for color R, G, & B
      Extract NxN chunk of image centered at particular row (I), column (J) in A
      Convert the type of ‘chunk’ to double
      Multiply ‘chunk’ elementwise with kernel K and sum all elements
      Assign result to corresponding pixel in R (at I, J for color channel)
    end
  end
end

Display( A )
Display( R ) in separate figure
```

Some useful kernels

- Blur filters:

  - NxN kernel with all elements equal to $1/N^2$, e.g. for N=3:

$$\frac{1}{9} * \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

  - $\frac{1}{5} * \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$

  Small blur filters can also be used for noise reduction.

- Motion blur filter:

$$\frac{1}{5} * \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad \text{(diagonal blur filter)}$$

- Sharpen filter:

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

- Edge detection:

  - horizontal $\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$

  - vertical $\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$

- Median filter
  The median of all the numbers in the chunk. Use the **median** function.