Spatial Domain Filtering

Concept of Neighborhood

Number of pixels surrounding a given pixel in an image form its neighborhood. This neighborhood is usually a small matrix containing given pixel at its center.

The size of neighborhood matrix is usually $3 \times 3,\ 5 \times 5$ OR 7×7 i.e an odd number so that a given pixel can reside easily at the center of neighborhood matrix.

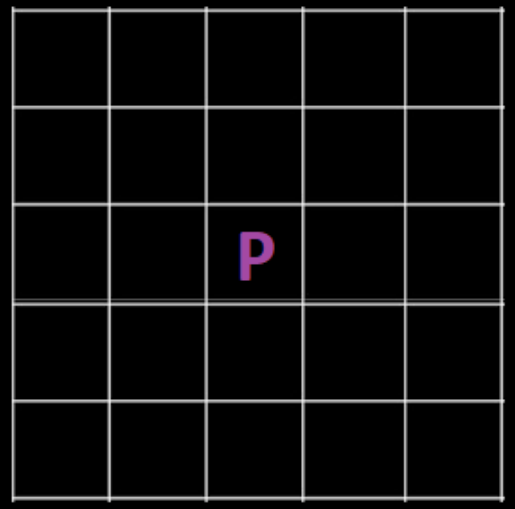
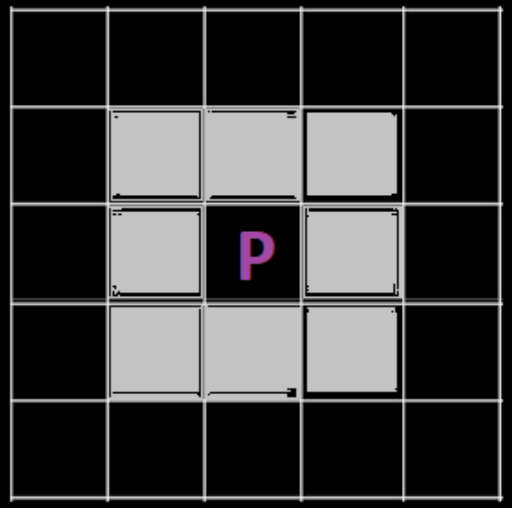
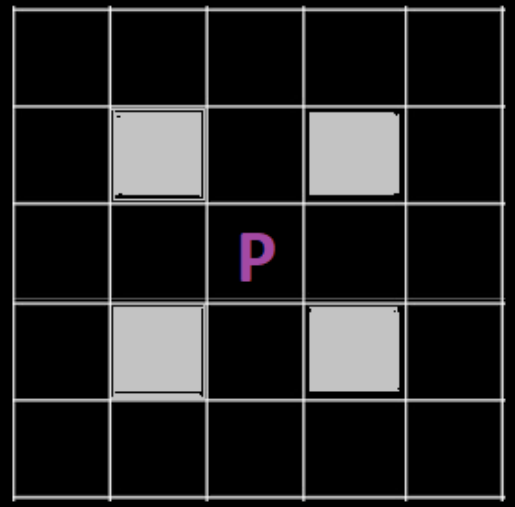


Image With Given pixel " P "



8 - Neighborhood



Diognal Neighborhood

Neighborhood Processing

Neighborhood processing involves those algorithms in image processing where the resulting value for a pixel at the reference pixel depends upon the original pixel value at that point as well as the original pixel value of some of its surrounding or neighboring points.

Steps of Neighborhood Processing

- Select a reference pixel in an input image. Lets call it $I(x_r, y_r)$.
- Perform neighborhood processing operation which involves the pixels within a neighborhood around the reference pixel in the input image.
- Apply the result of the neighborhood processing operation to the pixel of same coordinates in the output image $J(x_r, y_r)$.
- Repeat the above neighborhood processing operation for every pixel in the input image.

2D Convolution

Steps in 2D Convolution

- Flipping
- Shifting / Sliding
- Multiplication
- Addition

Numerical Example in 2D Convolution

6	3	7	9	1
7	0	3	1	3
8	1	5	3	6
3	9	4	3	5
7	8	2	3	0

Image

1	3	1
2	-1	2
3	-2	-1

Kernel

Flipping the Kernel

1	3	1
2	-1	2
3	-2	-1

Kernel

3	-2	-1
2	-1	2
1	3	1

Horizontally Flipped

-1	-2	3
2	-1	2
1	3	1

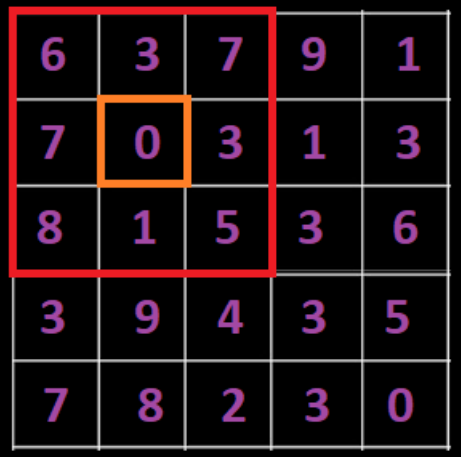
Flipped Kernel

6	3	7	9	1
7	0	3	1	3
8	1	5	3	6
3	9	4	3	5
7	8	2	3	0

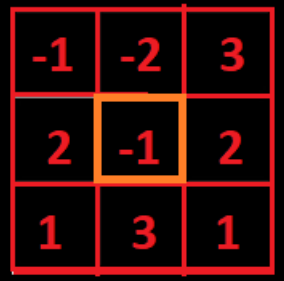
Image

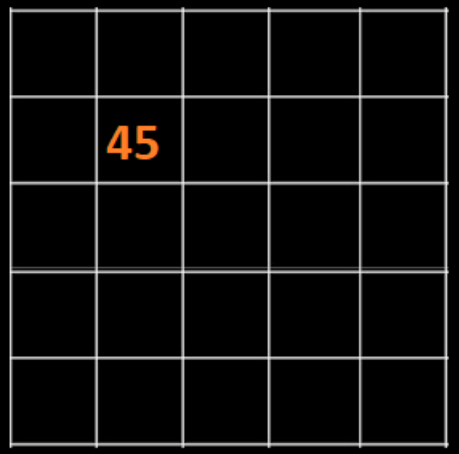
-1	-2	3
2	-1	2
1	3	1

Flipped Kernel



Image





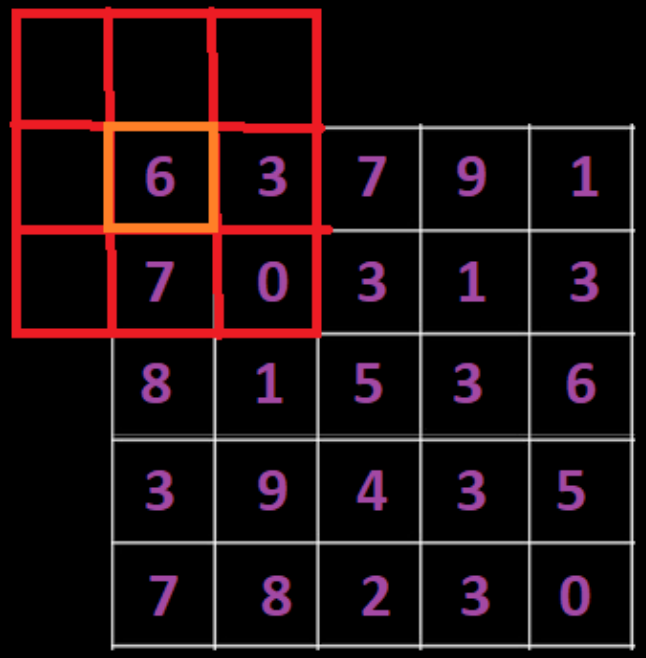
Convolution

Multiplication and Addition

(6 x -1) + (3 x -2) + (7 x 3)
(7 x 2) + (0 x -1) + (3 x 2)
(8 x 1) + (1 x 3) + (5 x 1)

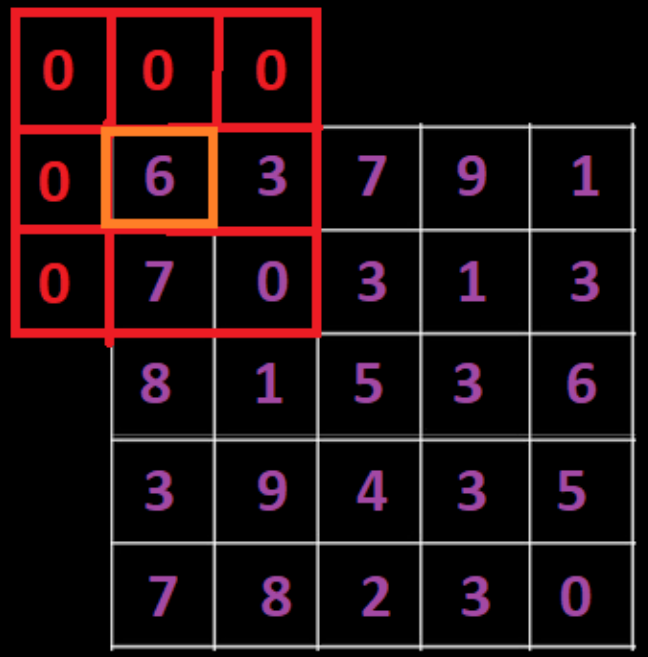
= -6 -6 +21 + 14 + 0 + 6 + 8 + 3 + 5

= 45



Image

-1	-2	3
2	-1	2
1	3	1



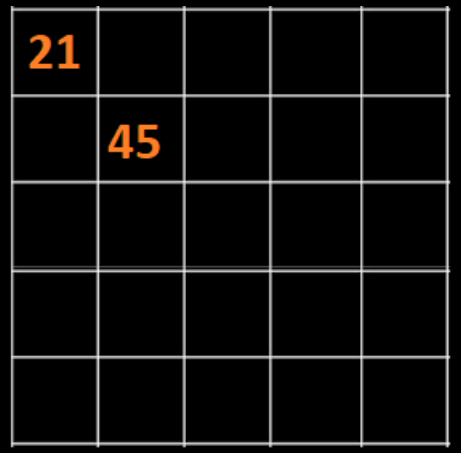
Image

-1	-2	3
2	-1	2
1	3	1

Multiplication and Addition

(6 x -1) + (3 x 2) + (7 x 3)
+ (0 x 1)

= -6 + 6 + 21 + 0
= 21



Convolution

Convolution Results

21	33	27	16	27
15	45	28	9	9
0	61	24	41	11
31	43	35	33	0
10	1	12	6	0

Convolution

Applications of 2D Convolution

Image Blurring

Image

$\circledast$

0.67	0.67	0.67
0.67	0.67	0.67
0.67	0.67	0.67

Kernel

=

Blurred Image

Image Sharpening

Image

$\circledast$

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

Kernel

=

Sharped Image

Edge Detection

Concept of Edge

An edge is a boundary between two image regions having sharp variations in their intensities.

Ideal Edge

Edge Detection

Image

$\circledast$

1	1	1
0	0	0
-1	-1	-1

Kernel

=

Edge Detected Image

Mean Filter

Mean filter is a neighborhood averaging filter which is used to smooth the image. It is used to create blurring in the image and to remove high frequency contents such as noise from the image.

This filter uses convolution with $m \times m$ kernel or mask. All the coefficients in the kernel are 1 divided by the number of elements in kernel.

A typical 3×3 kernel is given by

$$k(x,y) = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

A 5×5 kernel is given by

$$k(x,y) = \frac{1}{25}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Applications of Mean Filter

Image Blurring

$$\circledast \quad \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad =$$

Mean Filter

Image

Blurred Image

Image Blurring

Image

$$\circledast \quad \frac{1}{25}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} =$$

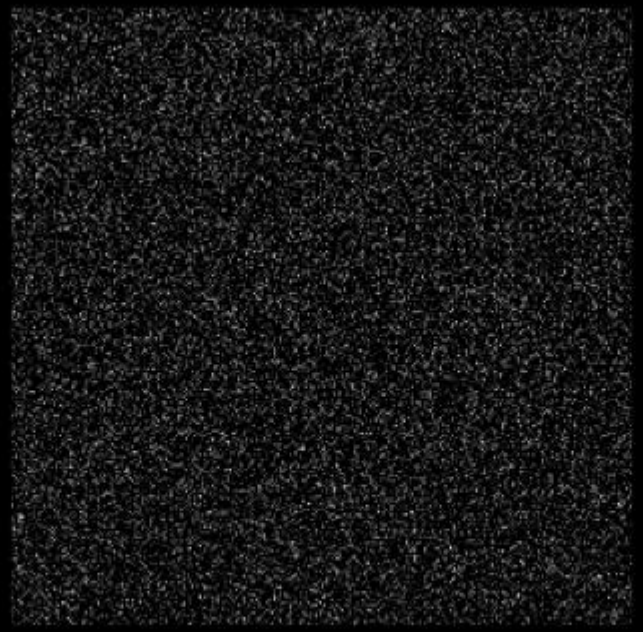
Mean Filter

Blurred Image

Image Denoising

Image

+



Noise

=

Noisy Image

Image Denoising

Noisy Image

$\circledast$

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

Mean Filter

=

Denoised Image

Image Denoising

Noisy Image

$$\circledast \quad \frac{1}{25}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Mean Filter

=

Denoised Image

Gaussian Smooth Filter

Instead if using simple mean, Gaussian smmothing filter uses a weighted mean, where neighborhood pixels that are closer to the central pixel contribute more weight to the average.

More natural smmothing or blurring is obtained with Gaussian filter as compared to Mean filter and more edges or details are preserved with Gaussian filter.

$$k(x, y) = exp[\frac{-(x^2 + y^2)}{2\sigma^2}]$$

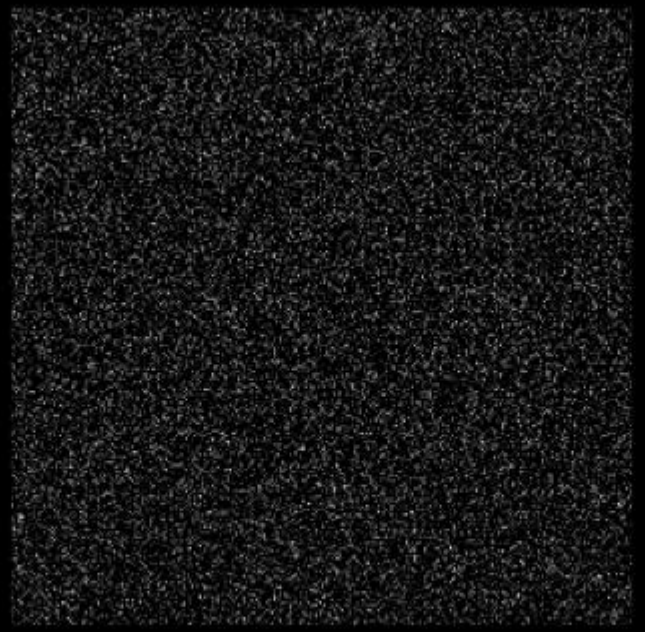
A typical 5×5 kernel is given by

$$k(x,y) = \frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

Image Denoising

Image

+



Noise

=

Noisy Image

Image Denoising

$$\circledast \quad \frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix} =$$

Image

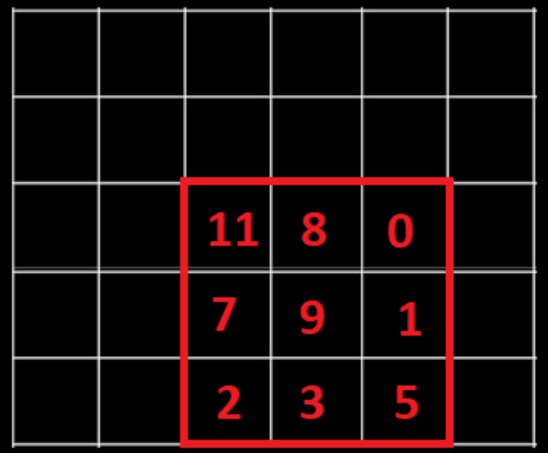
Gaussian Kernel

Denoised Image

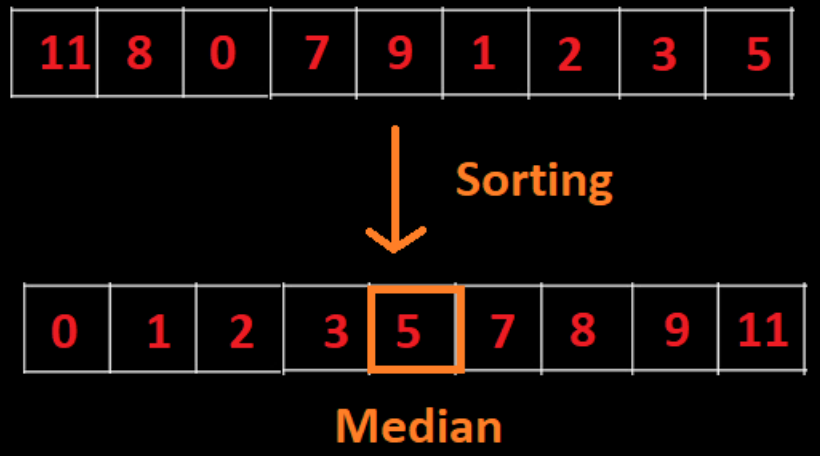
Median Filter

The median filter works by sorting the pixel values within a neighborhood, finding the median value, and replacing the original pixel value with the median of that neighborhood.

Median filter is used to remove salt and pepper noise from the image which is not well denoised by Gaussian or Mean Filter.



Image



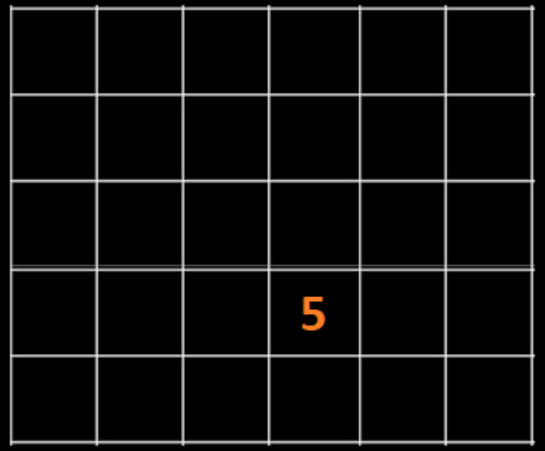
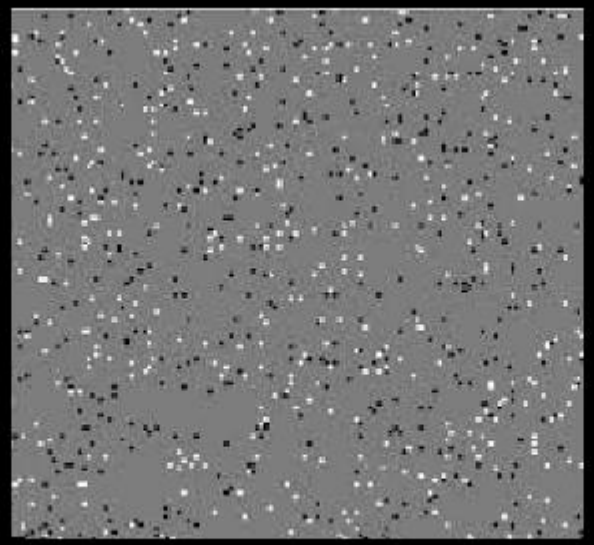


Image After Median Filtering

Image Denoising by Median Filter

Image



Salt and pepper noise

Noisy Image

Noisy Image

Denoised by Gaussian Filter

Denoised by Median Filter

The Laplacian

The Laplacian of an image $I(x, y)$ is given by

$$\nabla^2(x, y) = \frac{\partial^2(x, y)}{\partial x^2} + \frac{\partial^2(x, y)}{\partial y^2}$$

Laplacian is a second order derivative of the image pixels. It acts as a high pass filter by preserving the high frequency components such as fine details, edges, lines and also perform image sharpening.

A typical 3×3 Laplacian kernel is given by

$$k(x,y) = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

OR

$$k(x,y) = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

We can observe that signs of central element of kernel is reverse as compared to other elements.

Edge Detection by Laplacian

Image

$\circledast$

Gaussian Kernel

=

Smooth Image

Edge Detection by Laplacian

Smooth Image

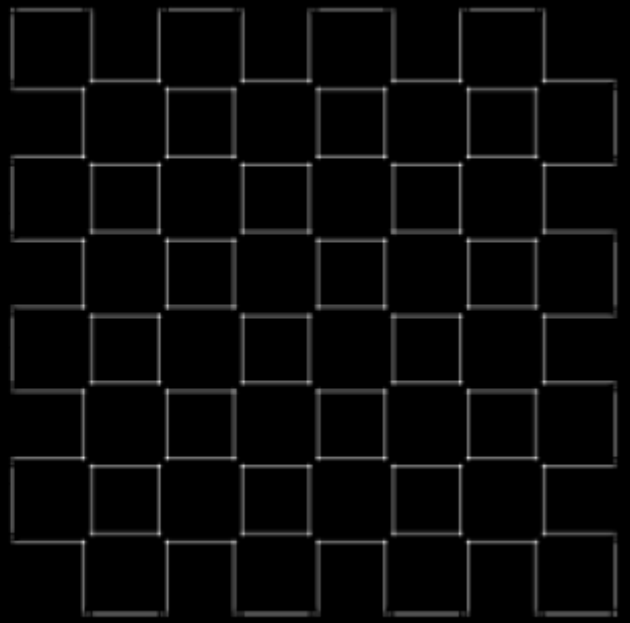
$$\circledast \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} =$$

Laplacian

Edge Detection

Edge Detection by Laplacian

Smooth Image



Edge Detection by Laplacian

High Boost Filtering

High boost filtering is a high pass filter version and is uesd to extract fine details from the image.

A typical 3×3 High boost filter is given by

$$k(x,y) = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

High Boost Filtering

Smooth Image

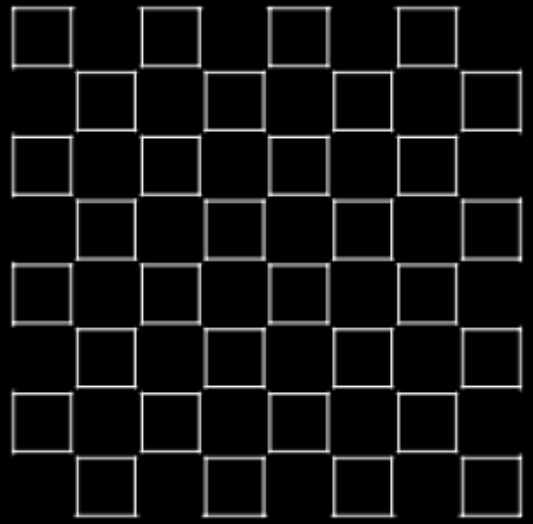
$$\circledast \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} =$$

High Boost Filter

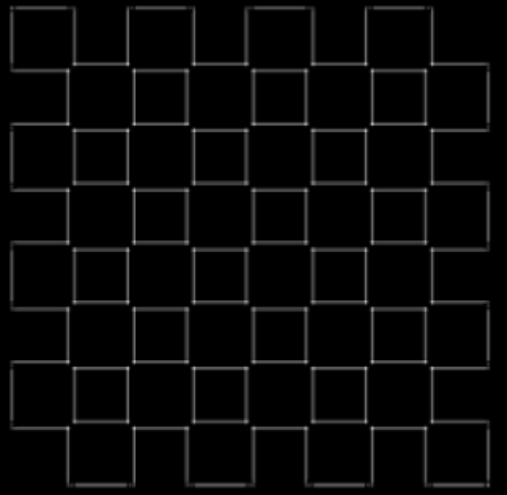
Edge Detection

High Boost Filtering

Smooth Image



Edge Detection
By High Boost Filter



Edge Detection by Laplacian

Sobel Filters

The Sobel filters are edge detectors. They detect the edges by estimating the first derivative of an image by doing a convolution between an image and the two kernels, one to detect vertical edges and one to detect horizontal edges.

Vertical edges are detected by using a horizontal gradient operator.

A typical 3×3 sobel filter for vertical edge detection is given by

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

A typical 3×3 sobel filter for horizontal edge detection is given by

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

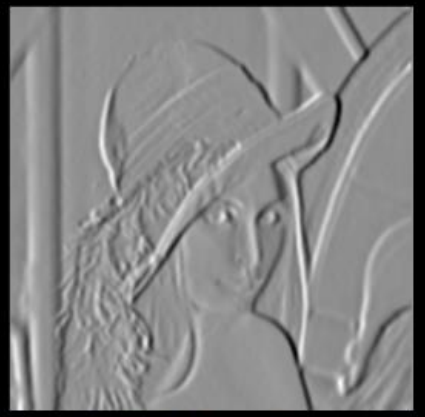
Image

$\circledast$

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

Sobel X

=



Vertical Edge Detection

$\circledast$

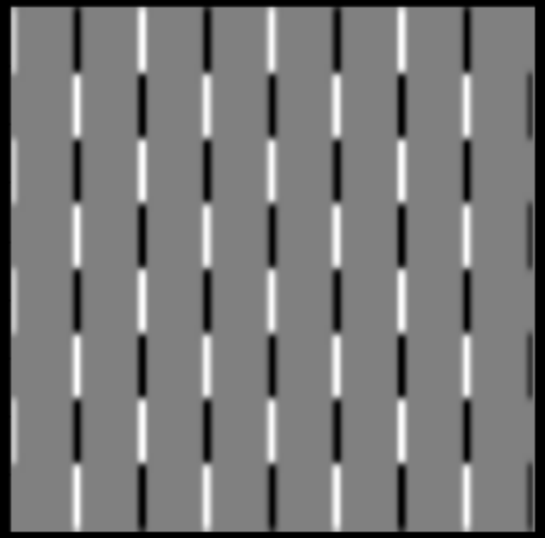
$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Sobel Y

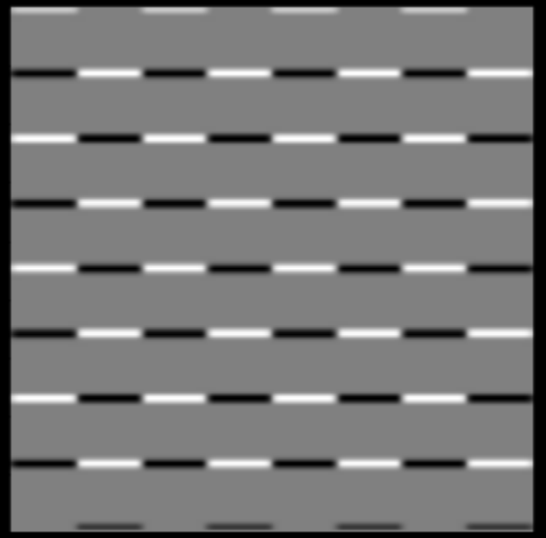
=

Horizontal Edge Detection

Image



Vertical Edge Detection By Sobel X



Horizontal Edge Detection By Sobel Y

Canny Edge Detection

Canny edge detection works in following steps

- The input image is smoothed using a Gaussian low-pass filter.
- The gradient is calculated for each pixel in the smoothed image.
- The pixels are then thresholded using two threshold values i.e low threshold and high threshold. Pixels with values greater than high threshold are considered strong edge pixels whereas the pixels with values between Low threshold and high threshold are said to be weak pixels. This process is known as hysteresis thresholding.
- Pixels having intensity between both thresholds are weak. The algorithm performs edge linking that help us identify the ones that could be considered as strong edges.

Image

Thresholds = 50, 120

Thresholds = 10, 200

Thresholds = 200, 240

Image

Thresholds = 50, 120

Thresholds = 10, 200

Thresholds = 200, 240