Computer Vision

Image Transformation

Course Instructor:
Dr. Suman Kumar Maji

Image Transformation

- In computer vision, we are developing a system whose input is an image and output is an image.

- The system can be histogram processing.

Image Transformation

- The system can be geometric transformation functions/matrix.

- Usually the black box(system) used for computer vision is an LTI system or linear time invariant system.
- By linear we mean that such a system where output is always linear , neither log nor exponent or any other.
- And by time invariant we means that a system which remains same during time.

Convolution

It can be mathematically represented in two ways:

$$g(x, y) = h(x, y) * f(x, y)$$

where h is referred to as the mask / filter / kernel, etc.

- Mask is also an image.
- It can be represented by a two dimensional matrix.
- The mask is usually of the order of 1x1, 3x3, 5x5, 7x7 . A mask should always be in odd number, because other wise you cannot find the mid of the mask.

Image courtesy: `tutorialpoints.com`

Properties

As a mathematical operation the convolution has several mathematical properties:

- **Neutral Element:** The neutral element of convolution is an image filled with zeros, but the pixel at the center equals 1.
- **Commutative Property:** The convolution is commutative:

$$g * h = h * g$$

- **Distributive Property:** The convolution is distributive with respect to addition:

$$g * (h_1 + h_2) = g * h_1 + g * h_2$$

- **Bilinearity:** The convolution is bilinear:

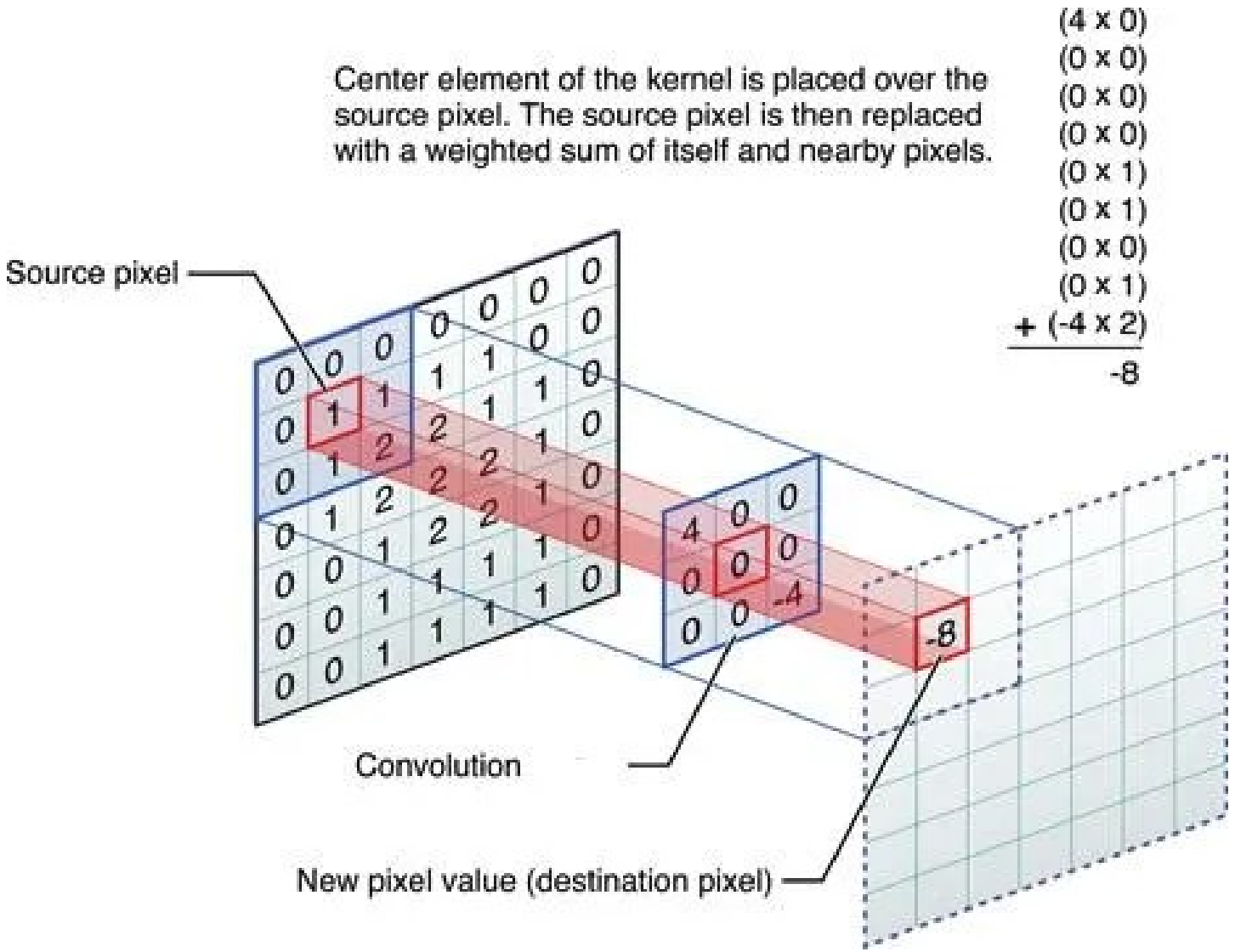
$$\alpha(g * h) = (\alpha g) * h = g * (\alpha h)$$

where, $(\alpha \in \mathbb{C})$

- **Associative Property:** The convolution is associative:

$$h_1 * (h_2 * h_3) = (h_1 * h_2) * h_3$$

The Process of Convolution:



Courtesy: `Medium.com`

The Process of Convolution: (cont...)

The Convolution Process involves these steps:

1. It places the Kernel Matrix over each pixel of the image (ensuring that the full Kernel is within the image), multiplies each value of the Kernel with the corresponding pixel it is over.
2. Then, sums the resulting multiplied values and returns the resulting value as the new value of the center pixel.
3. This process is repeated across the entire image.

As we see in the picture, a 3x3 kernel is convoluted over a 7x7 source image.

Center Element of the kernel is placed over the source pixel.

The source pixel is then replaced with a weighted sum of itself and surrounding pixels.

The output is placed in the destination pixel value.

Boundaries effects:

- The convolution formula is not defined on the boundaries of the image.
- Therefore, one has to make assumptions about the pixel values outside the image.
- In those cases, you can modify the output size by adding leading and trailing zeroes to the input.
- This is called padding, and it allows more multiplication operations to happen for the values in the borders, thus allowing their information to have a fair weight.

Boundaries effect: Padding

Input with padding

0	0	0	0	0	0
0	2	3	1	4	0
0	3	1	3	2	0
0	3	0	1	3	0
0	0	2	0	1	0
0	0	0	0	0	0

conv

kernel

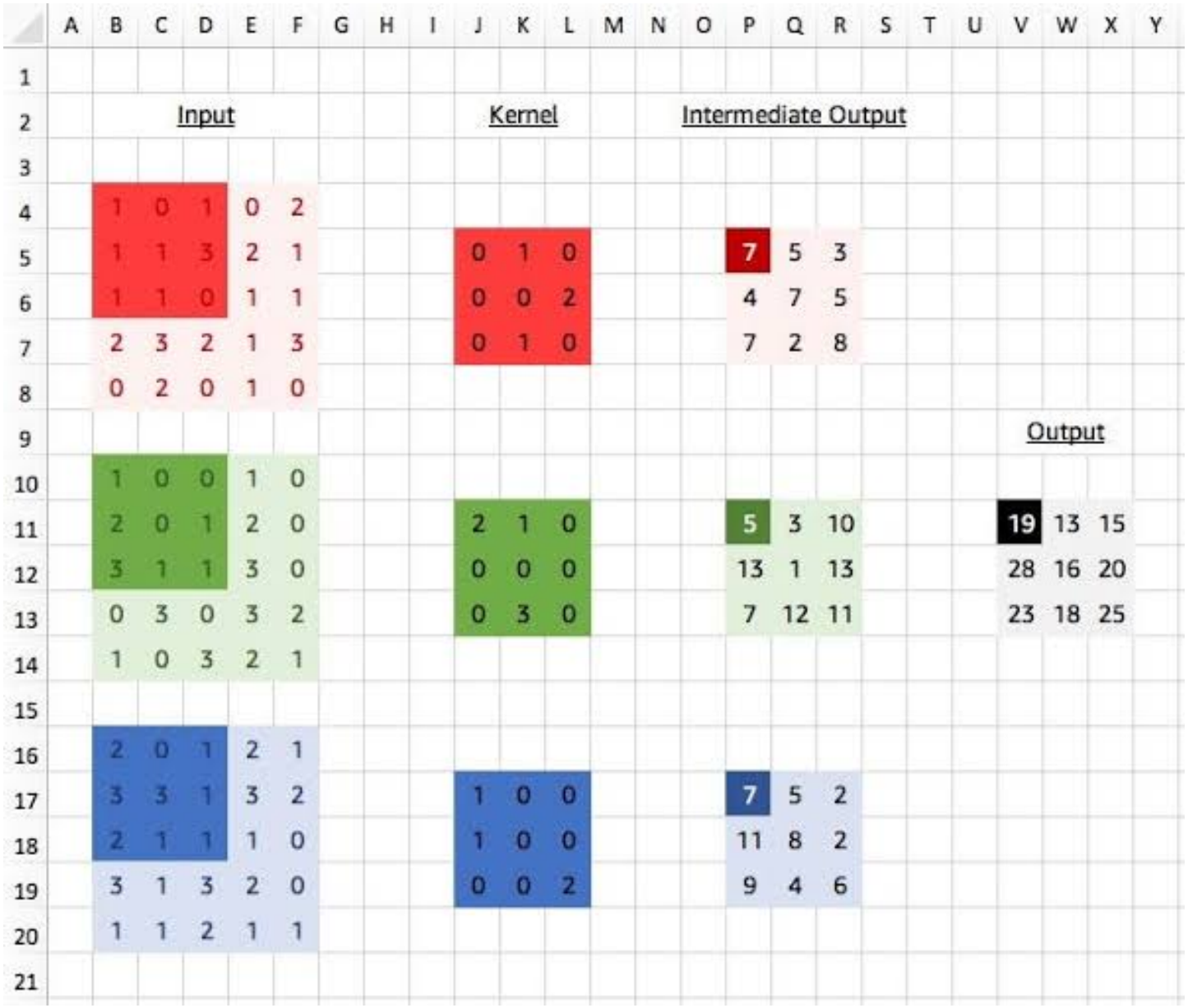
1	0	-1
1	0	-1
1	0	-1

=

Output of original size

-4	1	-2	4
-4	3	-5	5
-3	2	-3	4
-2	2	-2	1

Color channel 3D convolution



Smoothing/Blurring Filters

Smoothing Filters are used for blurring and for noise reduction. Two types of blurring filters will be discussed:

1. Smoothing Non-Linear Filters
   Ex: Mode filter, Median filter
2. Smoothing Linear Filters
   Ex: Mean filter (also called Box filter), Gaussian filter

Mode Filter

- The mode filter is one of the simplest image-smoothing algorithms.
- In this algorithm each pixel of the original image is going to be replaced by the mode value of its neighboring pixels.

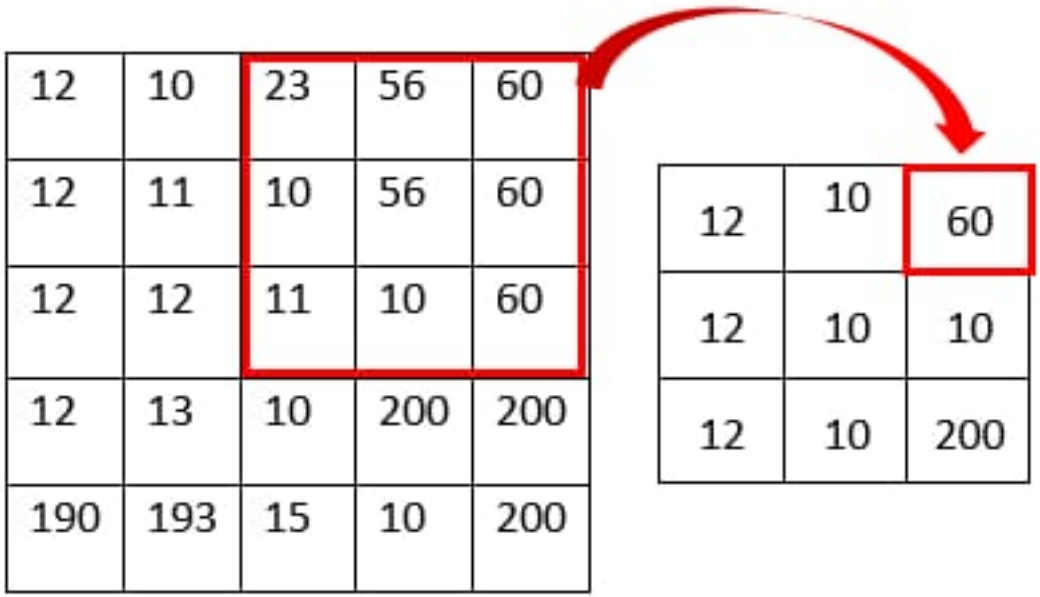


image courtesy blog.geveo.com

Mode filter (cont...)

- As shown in the above figure the pixel value is the mode value of the kernel.
- For example, the top right pixel value of the resulting image becomes 60 because the mode of the kernel is 60 (23, 56, 60, 10, 56, 60, 11, 10, 60).

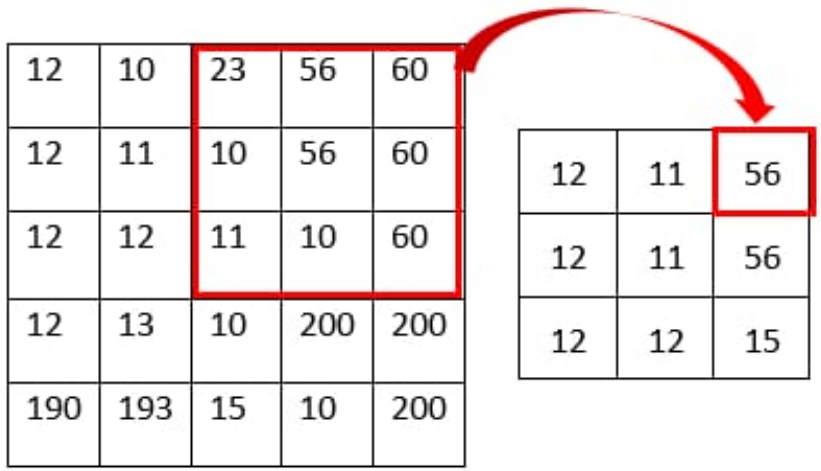
Before

After

image courtesy `blog.geveo.com`

Median Filter

- The median filter is also one of the simplest image-smoothing algorithms.
- In this algorithm, each pixel of the original image is going to be replaced by the median value of its neighboring pixels.



The top right pixel value of the resulting image becomes 56 because the median of the kernel is 56 (10, 10, 11, 23, 56, 56, 60, 60, 60).

- Sort all values in the kernel from ascending to descending, then choose the middle element.

image courtesy `blog.geveo.com`

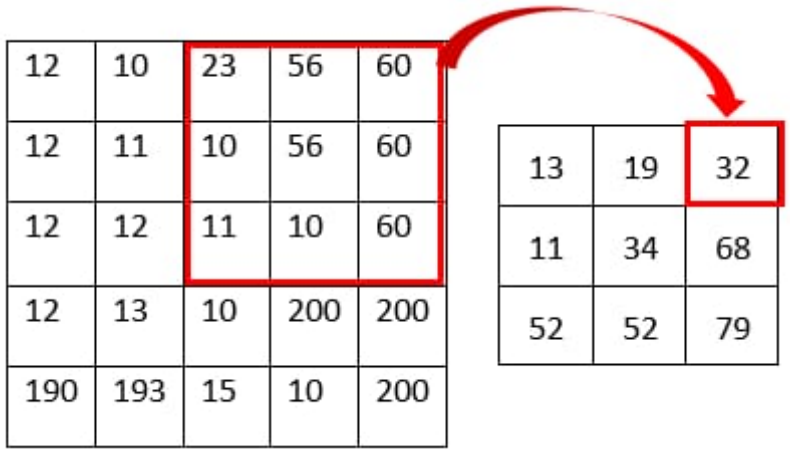
Median Filter (cont...)

image courtesy `blog.geveo.com`

Mean Filter (Box Filter)

- The mean filter is also known as averaging filter as it takes the average values of the neighboring pixels.
- In this algorithm, each pixel of the original image is going to be replaced by the mean value of its neighboring pixels.

Mean Filter (cont...)



The top right pixel value of the resulting image becomes 32 because the mean of the kernel is 32 $((10 + 10 + 11 + 23 + 56 + 56 + 60 + 60 + 60)/9 = 32)$.

image courtesy `blog.geveo.com`

Mean Filter (cont...)

Before

After

image courtesy `blog.geveo.com`

Gaussian Filter

- The Gaussian filter is almost the same as the mean filter but in the mean filter, all the neighboring pixels have the same weight regardless of the distance from the center pixel.
- In the gaussian filter the neighboring pixels are assigned a weight based on the distance from the center pixel.
- The kernel for the gaussian filter can be calculated with the help of 2D gaussian function.

Gaussian Filter (cont...)

- Following is the 2D gaussian function.

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

- In the above equation x and y are the coordinates of the pixel and sigma can be varied from 1 to any number.

image courtesy `blog.geveo.com`

Gaussian Filter (cont...)

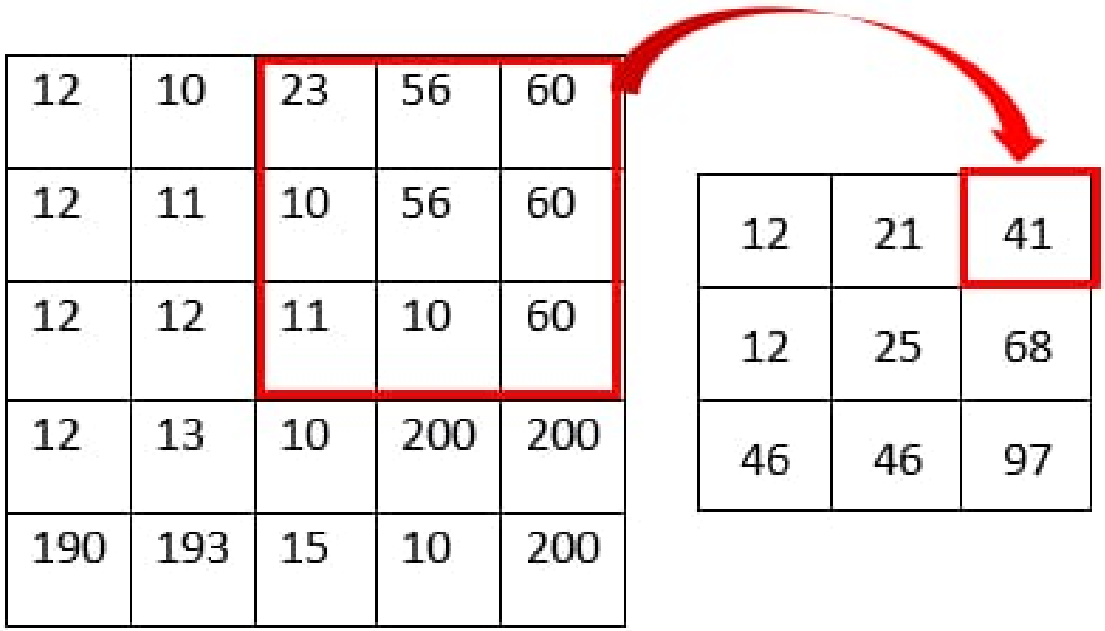
- Following is the sample 3*3 kernel for gaussian filter when sigma is 1.

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

image courtesy blog.geveo.com

Gaussian Filter (cont...)

Following image shows how to calculate the resulting image using gaussian filter.



The top right pixel value of the resulting image becomes 41 because the gaussian average of the kernel is 41 ((23 * 1 + 56 * 2 + 60 * 1 + 10 * 2 + 56 * 4 + 60 * 2 + 11 * 1 + 10 * 2 + 60 * 1)/16 = 41).

image courtesy blog.geveo.com

Gaussian Filter (cont...)

image courtesy `blog.geveo.com`

Sharpening filter

- Sharpening as name suggests is used to sharpen and highlight the edges and make the transitioning of features and details more significant.
- However sharpening doesn't take into account whether it is highlighting the original features of the image or the noise associated with it.
- It enhances both.
- We will discuss two such filters:
  1. Unsharp Masking and High Boost Filtering
  2. Laplacian Filters

Unsharp Masking and High Boost Filtering

We can sharpen an image or perform edge enhancement using a smoothing filter.

1. Blur the image. Blurring means supressing most of high frequency components.
2. Output (Mask) = Original Image - Blurred image. This output now contains now contains most of the high frequency components that were blocked by the blurring filter.
3. Adding the mask to original image will enhance the high frequency components.

Since we are using blurred image for creating our custom mask, this process is known as UNSHARP MASKING.

Thus, Unsharp Mask m(x , y) can be represented as :

$$m(x, y) = f(x, y) - f_b(x, y)$$

- f(x,y) = original image.
- $f_b(x, y)$ = blurred image.

Add this mask back to the original image resulting in enhanced high frequency components.

$$g(x, y) = f(x, y) + k * m(x, y)$$

- k = 1 represents Unsharp Masking.
- k > 1 represents High Boost Filtering because we are boosting high frequency components by assigning more weights to the mask(edge features) of the image.

cont...

Original Image

Sharpened Image

- **If the image contains noise, this method will not produce satisfactory results, like most of the other sharpening filters.**
- **Instead of subtracting the blurred image from the original, we can directly use a negative Laplacian filter to obtain the mask.**

---

image courtesy: `https://theailearner.com/2019/05/14/unsharp-masking-and-highboost-filtering/`

Laplacian Filters

- A Laplacian Filter is a second-order derivative mask.
- It tries to take out the INWARD edges and the OUTWORD edges.
- This second order derivative changes helps to find out whether the changes we are observing are due to pixel change of continous regions or from an edge.

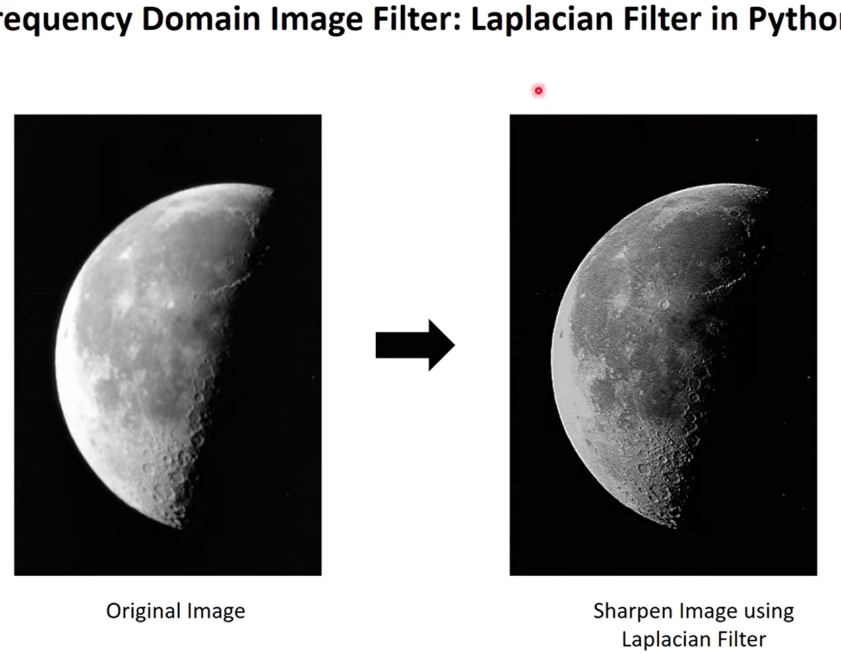


image courtesy: YouTube channel (Made Python)
https://www.youtube.com/watch?v=i-Rvo48vBKA

cont...

- A general Laplacian kernel contains a positive values at the center and negative values in cross-pattern.

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

- Now to go into the derivation of this kernel matrix we need to be familiar with partial derivatives and Laplacian operators.

image courtesy: `https://iq.opengenus.org/sharpening-filters/`

cont...

- Let us consider our image as a function of two, f(x , y).
- We will be dealing with partial derivatives along the two spatial axes.
- **Gradient operator (linear operator):**

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

- **Laplacian operator (non-linear operator):**

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

- **Discrete form of Laplacian**

$$\frac{\partial^2 f(x,y)}{\partial x^2} = f(x+1,y) + f(x-1,y) - 2f(x,y)$$

$$\frac{\partial^2 f(x,y)}{\partial y^2} = f(x,y+1) + f(x,y-1) - 2f(x,y)$$

$$\nabla^2 f = [f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1) - 4f(x,y)]$$

cont...

- **Resultant Laplacian Matrix**

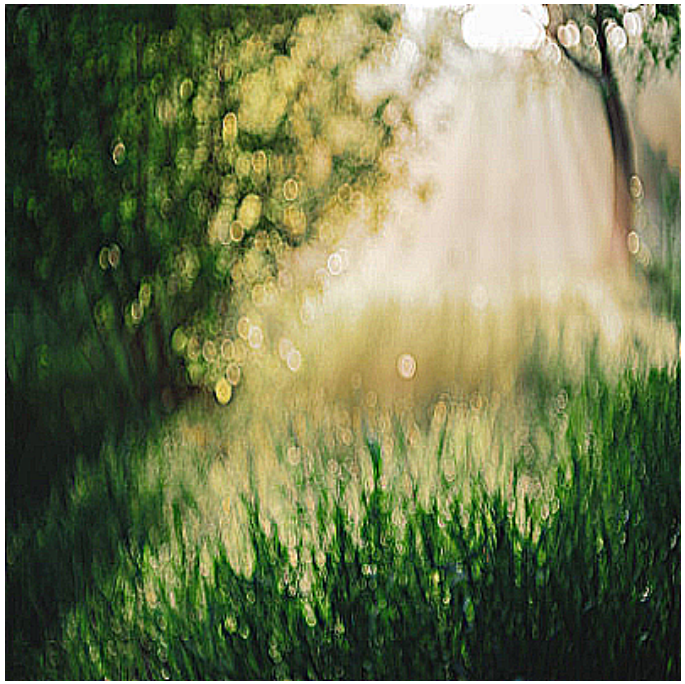
$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

image courtesy: `https://iq.opengenus.org/sharpening-filters/`

Effects of Laplacian Operators

- It highlights and enhances the gray discontinuities.
- It de-emphasizes continuous region (region without edges) i.e. with slowly varying derivatives.

Example of Laplacian sharpening filter



(left) Original v/s (right) Sharpened

Image courtesy: `https://iq.opengenus.org/sharpening-filters/`

Edge detection filters

- Mathematically, an edge is a line between two corners or surfaces.
- The key idea behind edge detection is that areas where there are extreme differences in the brightness of pixels indicate an edge.
- Therefore, edge detection is a measure of discontinuity of intensity in an image.

Types of edge detection techniques

There are various types of edge detection techniques, which include the following:

1. Sobel Edge Detection
2. Canny Edge Detection
3. Laplacian Edge Detection
4. Prewitt Edge Detection
5. Roberts Cross Edge Detection
6. Scharr edge detection

Sobel Edge Detection

- The Sobel operator uses two 3x3 convolution kernels (filters), one for detecting changes in the x-direction (horizontal edges) and one for detecting changes in the y-direction (vertical edges).
- These kernels are used to compute the gradient of the image intensity at each point, which helps in detecting the edges.
- Here are the Sobel kernels:
  1. Horizontal Kernel (G_x):
  2. Vertical Kernel(G_y):

Horizontal Kernel (G_x)

- This kernel is used to detect horizontal edges by emphasizing the gradient in the x-direction.

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

Image courtesy: `https://blog.roboflow.com/edge-detection/`

Vertical Kernel (G_y)

- This kernel is used to detect vertical edges by emphasizing the gradient in the y-direction.

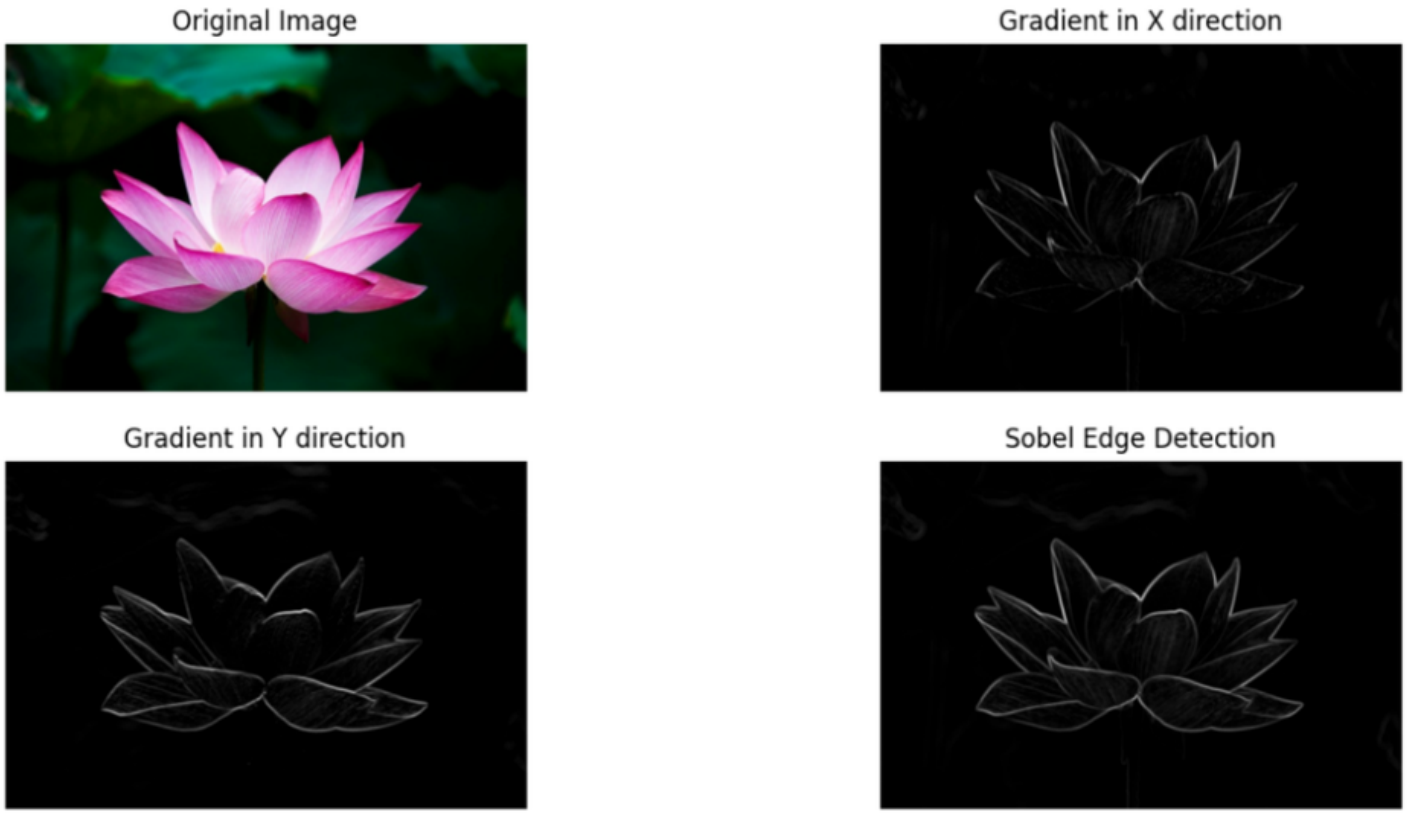
$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Image courtesy: `https://blog.roboflow.com/edge-detection/`

Step-by-step process for Sobel edge detection

1. Load and Display the Image
2. Convert to Grayscale
3. Apply Gaussian Smoothing (Optional)
4. Apply Sobel Operator
5. Calculate Gradient Magnitude
6. Normalization
7. Display the Resulting Edge Image

Example Sobel edge detection



Example of Sobel Edge Detection

Image courtesy: `https://blog.roboflow.com/edge-detection/`

Canny Edge detection

- Canny edge detection is a image processing method used to detect edges in an image while suppressing noise.
- The main steps are as follows:
  1. Grayscale Conversion
  2. Gaussian Blur
  3. Determine the Intensity Gradients
  4. Non Maximum Suppression
  5. Double Thresholding
  6. Edge Tracking by Hysteresis

Step 1. Grayscale conversion

- Convert the image to grayscale.
- In MATLAB the intensity values of the pixels are 8 bit and range from 0 to 255.

Original

Black and White

Image courtesy: `https://justin-liang.com/tutorials/canny/`

Step 2. Gaussian Blur

- Perform a Gaussian blur on the image.
- The blur removes some of the noise before further processing the image.
- A sigma of 1.4 is used in this example and was determined through trial and error.

Gaussian Blur

Image courtesy: `https://justin-liang.com/tutorials/canny/`

Step 3 - Determine the Intensity Gradients

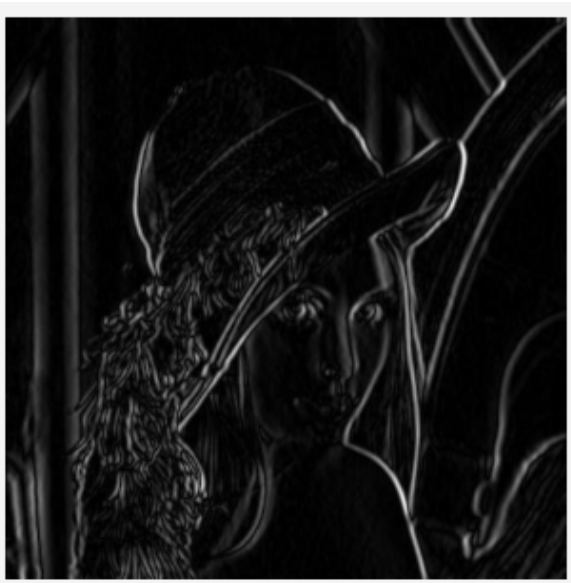
- The gradients can be determined by using a Sobel filter where A is the image.
- An edge occurs when the color of an image changes, hence the intensity of the pixel changes as well.

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} A, \quad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} A$$

Image courtesy: `https://justin-liang.com/tutorials/canny/`

Step 3 (cont...)

- Taking the derivatives will output:



Gx

Gy

- Then, calculate the magnitude and angle of the directional gradients:

$$Edge\_Gradient(G) = \sqrt{G_x^2 + G_y^2}$$

$$Angle\ (\theta) = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

Image courtesy: `https://justin-liang.com/tutorials/canny/`

Step 3. (cont...)

- The magnitude of the image results in the following output:

Gradient Magnitude

Image courtesy: `https://justin-liang.com/tutorials/canny/`

Step 4 - Non Maximum Suppression

- The image magnitude produced results in thick edges.
- Ideally, the final image should have thin edges.
- Thus, we must perform non maximum suppression to thin out the edges.

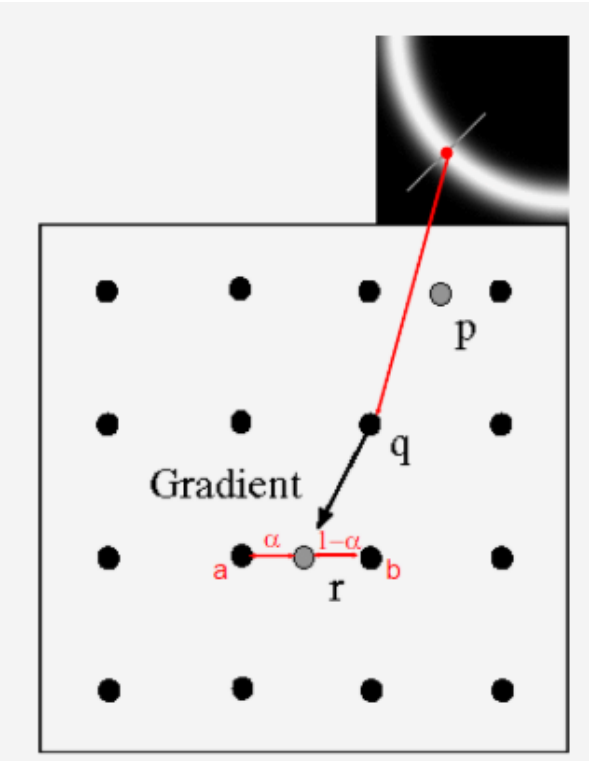


Image courtesy: `https://justin-liang.com/tutorials/canny/`

Step 4. (cont...)

- Non maximum suppression works by finding the pixel with the maximum value in an edge.
- In the above image, it occurs when pixel q has an intensity that is larger than both p and r where pixels p and r are the pixels in the gradient direction of q.
- If this condition is true, then we keep the pixel, otherwise we set the pixel to zero (make it a black pixel).

Step 4. (cont...)

- Non maximum suppression requires us to divide the 3x3 grid of pixels into 8 sections. i.e. if the gradient direction falls in between the angle -22.5 and 22.5, then we use the pixels that fall between this angle (r and q) as the value to compare with pixel p.

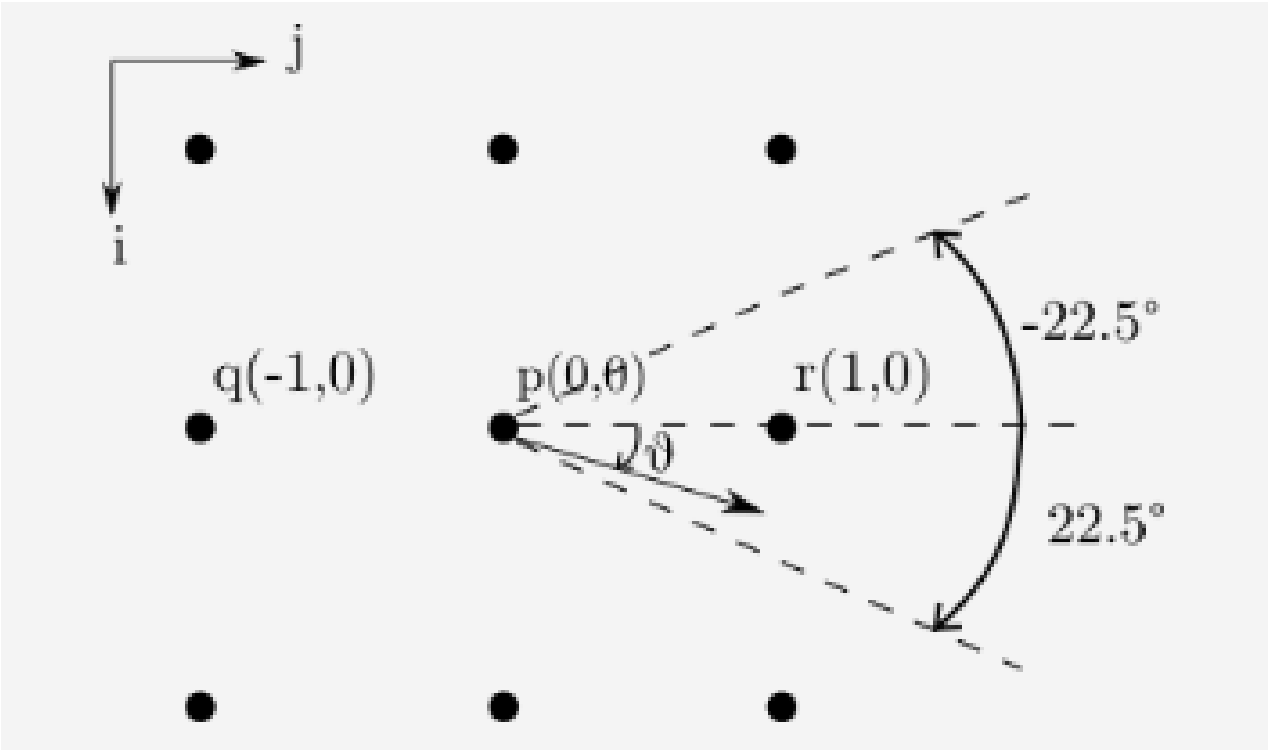


Image courtesy: `https://justin-liang.com/tutorials/canny/`

Step 4. (cont...)

- The result of this is:

Image courtesy: https://justin-liang.com/tutorials/canny/

Step 5. Double Thresholding

- We notice that the result from non maximum suppression is not perfect, some edges may not actually be edges and there is some noise in the image.
- Double thresholding takes care of this. It sets two thresholds, a high and a low threshold.
- For example, you might choose the high threshold to be 0.7, this means that all pixels with a value larger than 0.7 will be a strong edge.
- You might also choose a low threshold of 0.3, this means that all pixels less than it is not an edge and you would set it to 0.
- The values in between 0.3 and 0.7 would be weak edges, in other words, we do not know if these are actual edges or not edges at all.

Step 5. (cont...)

Image courtesy: https://justin-liang.com/tutorials/canny/

Step 6. Edge tracking by Hysteresis

- Based on the threshold results, the hysteresis consists of transforming weak pixels into strong ones, if and only if atleast one of the pixels around the one being processed is a strong one.

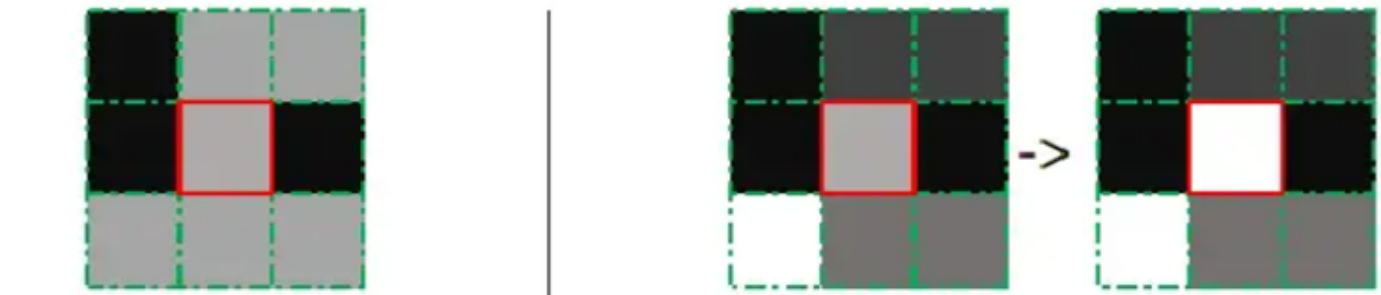


Image courtesy: scribd

Step 6. (cont...)

Edge Tracking

Image courtesy: https://justin-liang.com/tutorials/canny/

Laplacian Edge Detection

- This can be implemented using convolution with a Laplacian kernel. Common 3x3 kernels for the Laplacian operator include:

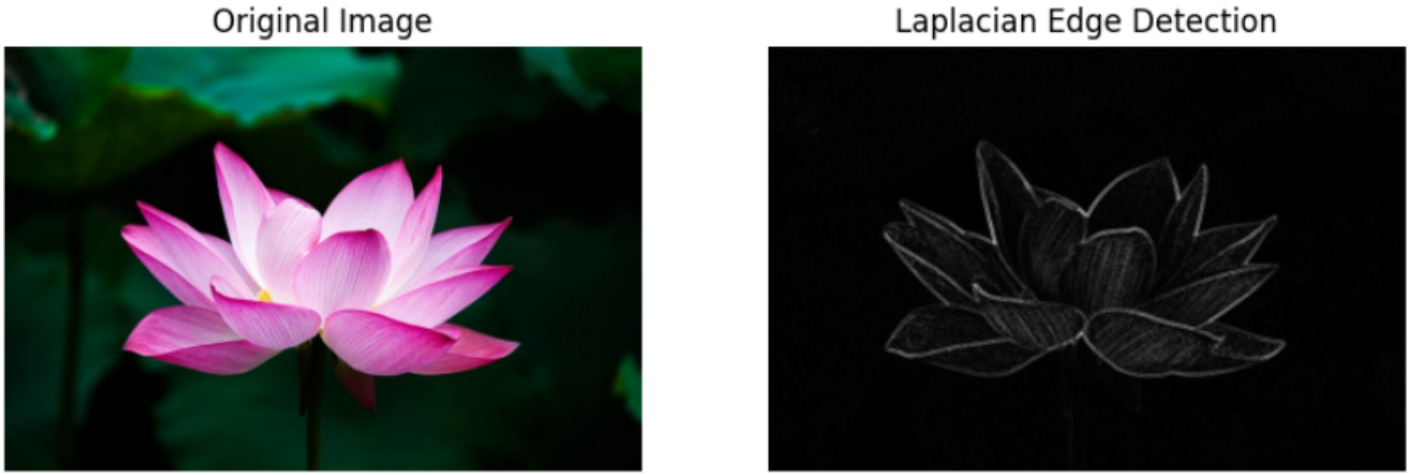
$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Image courtesy: `https://blog.roboflow.com/edge-detection/`

Steps for edge detection using Laplacian

1. Convert the Image to Grayscale
2. Apply Gaussian Blur (Optional)
3. Apply the Laplacian Operator

Example of Laplacian Edge detection



Example of Laplacian Edge Detection

Image courtesy: `https://blog.roboflow.com/edge-detection/`

Prewitt Edge Detection

- Prewitt edge detection uses two kernels, one for detecting edges in the horizontal direction and the other for the vertical direction.
- These kernels are applied to the image using convolution.

Horizontal Prewitt Kernel (Gx)

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

Image courtesy: https://blog.roboflow.com/edge-detection/

Vertical Prewitt Kernel (Gy)

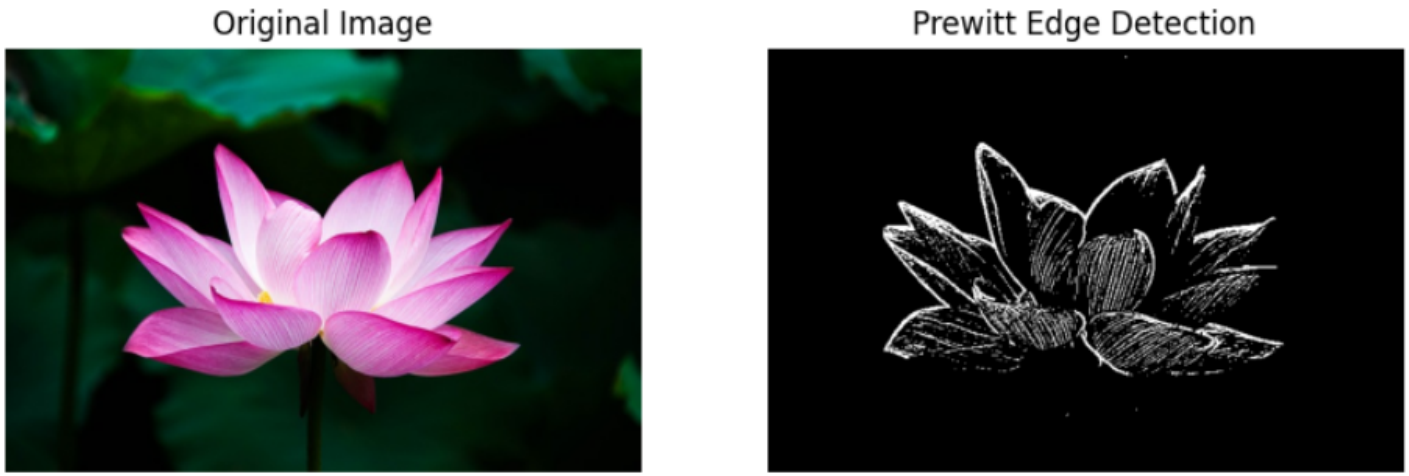
$$Gy = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

Image courtesy: `https://blog.roboflow.com/edge-detection/`

Steps to follow for Prewitt edge detection

1. Convert the Image to Grayscale
2. Apply the Horizontal and Vertical Prewitt Kernels
3. Compute Gradient Magnitude
4. Thresholding (Optional)

Example of Prewitt edge detection



Example of Prewitt Edge Detection

Image courtesy: `https://blog.roboflow.com/edge-detection/`

Roberts Cross Edge Detection

- Roberts Cross edge detection uses two kernels, one for detecting edges in the horizontal direction and the other for the vertical direction.
- These kernels are applied to the image using convolution.

Horizontal Roberts Cross Kernel (Gx)

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

Image courtesy: https://blog.roboflow.com/edge-detection/

Vertical Roberts Cross Kernel (Gy)

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

Image courtesy: https://blog.roboflow.com/edge-detection/

Steps for Roberts Cross Edge Detection

1. Convert the Image to Grayscale
2. Apply the Horizontal and Vertical Roberts Cross Kernels
3. Compute Gradient Magnitude
4. Thresholding (Optional)

Example of Roberts cross edge detection

Example of Roberts Cross Edge Detection

Image courtesy: `https://blog.roboflow.com/edge-detection/`

Scharr Edge Detection

- The Scharr operator consists of two 3x3 convolution kernels, one for approximating the horizontal gradient and the other for approximating the vertical gradient.
- The horizontal gradient kernel (Gx) is designed to approximate the rate of change of intensity in the horizontal direction, while the vertical gradient kernel (Gy) approximates the rate of change of intensity in the vertical direction.

Horizontal gradient kernel (Gx)

$$G_x = \begin{bmatrix} +3 & 0 & -3 \\ +10 & 0 & -10 \\ +3 & 0 & -3 \end{bmatrix}$$

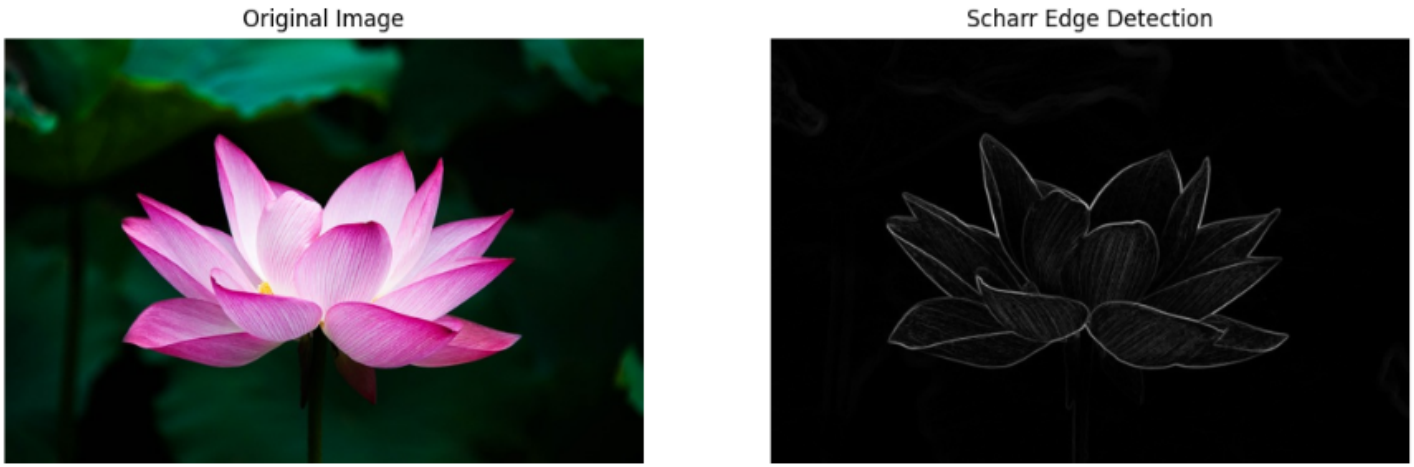
Image courtesy: `https://blog.roboflow.com/edge-detection/`

Vertical gradient kernel (Gy)

$$G_y = \begin{bmatrix} +3 & +10 & +3 \\ 0 & 0 & 0 \\ -3 & -10 & -3 \end{bmatrix}$$

Image courtesy: https://blog.roboflow.com/edge-detection/

Example of Scharr edge detection



Example of Scharr Edge Detection

Image courtesy: `https://blog.roboflow.com/edge-detection/`

Embossing Filters

- These are used to create a 3D effect by highlighting edges and providing a shadow on the other side.
- This can help in textural analysis or enhancing visual aesthetics.
- Example of an Embossing Kernel:

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

Example

(left) Original v/s (right) Embossed

Image courtesy: `Aspose.com`