আন্তর্জাতিক ইসলামী বিশ্ববিদ্যালয় চট্টগ্রাম

International Islamic University Chittagong

Lab Report

Department: Computer Science and Engineering.

Course Code: CSE-4876

Course Title: Pattern Recognition & Image Processing

Submitted By:

Name: Farida Nusrat

ID: C201242

Semester: 8th

Section: 8AF

Submitted To:

Mr. Mohammad Mahadi Hasaan

Associate Professor,

Dept. of CSE, IIUC.

Date of Submission: 26 March,2024

Lab No: 07

Lab Title:

Sharpening filtering

- High pass
- High boost

Derivative filtering

- Prewitt
- Sobel
- Robert
- Laplacian

Introduction:

In image processing, sharpening filters and derivative filtering techniques are commonly used to enhance image details and detect edges. Sharpening filters emphasize edges and fine details in an image, while derivative filters highlight changes in intensity, often indicative of edges. In this lab report, we explore various sharpening filters including high-pass and high-boost filters, as well as derivative filters such as Prewitt, Sobel, Robert, and Laplacian.

High Pass Filter:

- A 3x3 high-pass filter mask is designed to accentuate high-frequency components within the image.

-1	-1	-1
-1	8	-1
-1	-1	-1

- The mask is convolved over the image to obtain the high-frequency details, highlighting edges and fine features.

High Boost Filter:

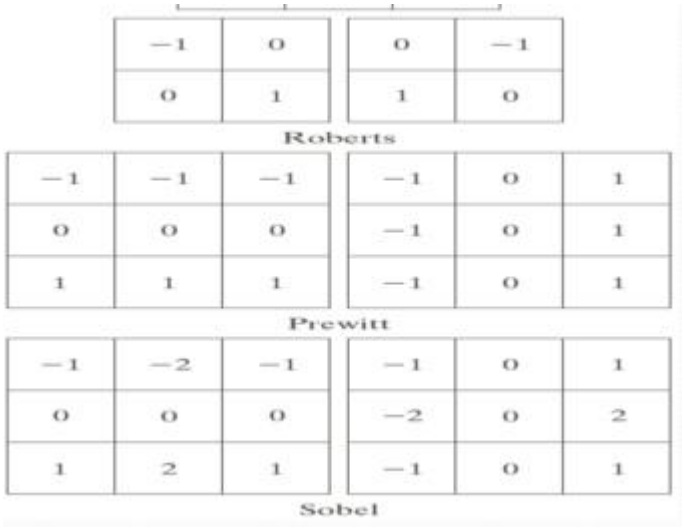
- Similar to the high-pass filter, but with a scaling factor applied to enhance high-frequency components more aggressively.

-1	-1	-1
-1	W	-1
-1	-1	-1

- W=9A-1
- The scaled high-frequency details are added back to the original image to boost sharpness while preserving overall contrast.

Derivative Filtering (Prewitt, Sobel, Robert, Laplacian):

- Specific kernels for each derivative filter (Prewitt, Sobel, Robert, Laplacian) are initialized to detect gradients and intensity changes.
- The kernels are convolved over the image to calculate horizontal and vertical derivatives, emphasizing edges and boundaries.
- For Robert and Laplacian filters, a combination of horizontal and vertical derivatives is used to enhance edge detection.



−1	0	0	−1
0	1	1	0

Roberts

−1	−1	−1	−1	0	1
0	0	0	−1	0	1
1	1	1	−1	0	1

Prewitt

−1	−2	−1	−1	0	1
0	0	0	−2	0	2
1	2	1	−1	0	1

Sobel

0	1	0	1	1	1
1	−4	1	1	−8	1
0	1	0	1	1	1

0	−1	0	−1	−1	−1
−1	4	−1	−1	8	−1
0	−1	0	−1	−1	−1

Laplacian

Methodology:

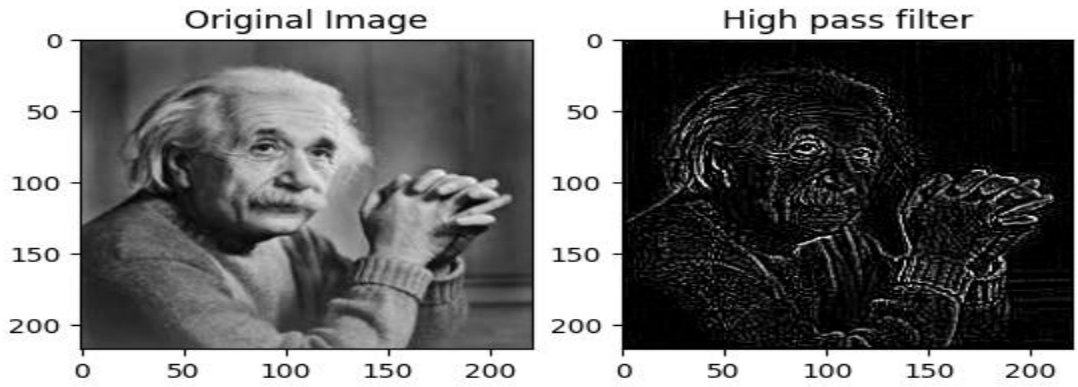
- **Sharpening filtering**

1. **High pass**
   - **Code**

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Read the image
img = cv2.imread('/content/Picture.png', 0)
# Obtain number of rows and columns of the image
m, n = img.shape
```

```
# Develop High-pass filter(3, 3) mask
mask = np.array([[-1, -1, -1],
                 [-1,  8, -1],
                 [-1, -1, -1]])
# Convolve the 3X3 mask over the image
img_high_pass = np.zeros([m, n])

for i in range(1, m-1):
    for j in range(1, n-1):
        temp = (img[i-1, j-1]*mask[0, 0] +
                img[i-1, j]*mask[0, 1] +
                img[i-1, j + 1]*mask[0, 2] +
                img[i, j-1]*mask[1, 0] +
                img[i, j]*mask[1, 1] +
                img[i, j + 1]*mask[1, 2] +
                img[i + 1, j-1]*mask[2, 0] +
                img[i + 1, j]*mask[2, 1] +
                img[i + 1, j + 1]*mask[2, 2])

        img_high_pass[i, j] = temp

# Clip the values to ensure they are within the valid range
img_high_pass = np.clip(img_high_pass, 0, 255)

plt.subplot(1, 2, 1)
plt.imshow(cv2.cvtColor(img, cv2.COLOR_RGB2BGR))
plt.title('Original Image')

plt.subplot(1, 2, 2)
plt.imshow(img_high_pass, cmap='gray')
plt.title('High pass filter')
plt.show()
```

- **Result**



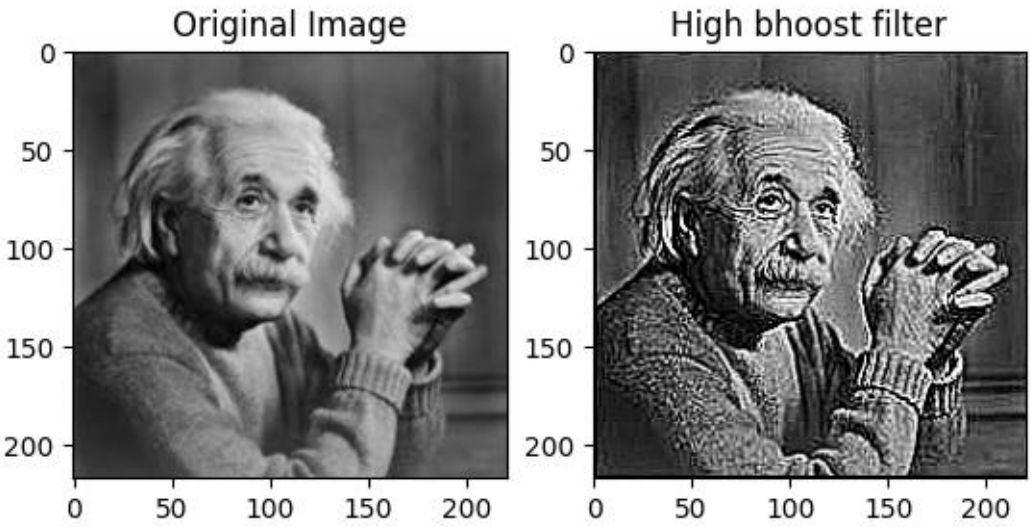
2. **High boost**

- **Code**

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Read the image
img = cv2.imread('/content/Picture.png', 0)
```

```
#img = cv2.imread('/content/img.jpg',0)
# Obtain number of rows and columns of the image
m, n = img.shape

# Develop High-pass filter(3, 3) mask
mask = np.array([[-1, -1, -1],
                 [-1,  8.9, -1],
                 [-1, -1, -1]])
# Convolve the 3X3 mask over the image
img_high_pass = np.zeros([m, n])

for i in range(1, m-1):
    for j in range(1, n-1):
        temp = (img[i-1, j-1]*mask[0, 0] +
                img[i-1, j]*mask[0, 1] +
                img[i-1, j + 1]*mask[0, 2] +
                img[i, j-1]*mask[1, 0] +
                img[i, j]*mask[1, 1] +
                img[i, j + 1]*mask[1, 2] +
                img[i + 1, j-1]*mask[2, 0] +
                img[i + 1, j]*mask[2, 1] +
                img[i + 1, j + 1]*mask[2, 2])

        img_high_pass[i, j] = temp

# Clip the values to ensure they are within the valid range
img_high_pass = np.clip(img_high_pass, 0, 255)

plt.subplot(1, 2, 1)
plt.imshow(cv2.cvtColor(img, cv2.COLOR_RGB2BGR))
plt.title('Original Image')

plt.subplot(1, 2, 2)
plt.imshow(img_high_pass, cmap='gray')
plt.title('High bhoost filter')
plt.show()
```

- **Result**

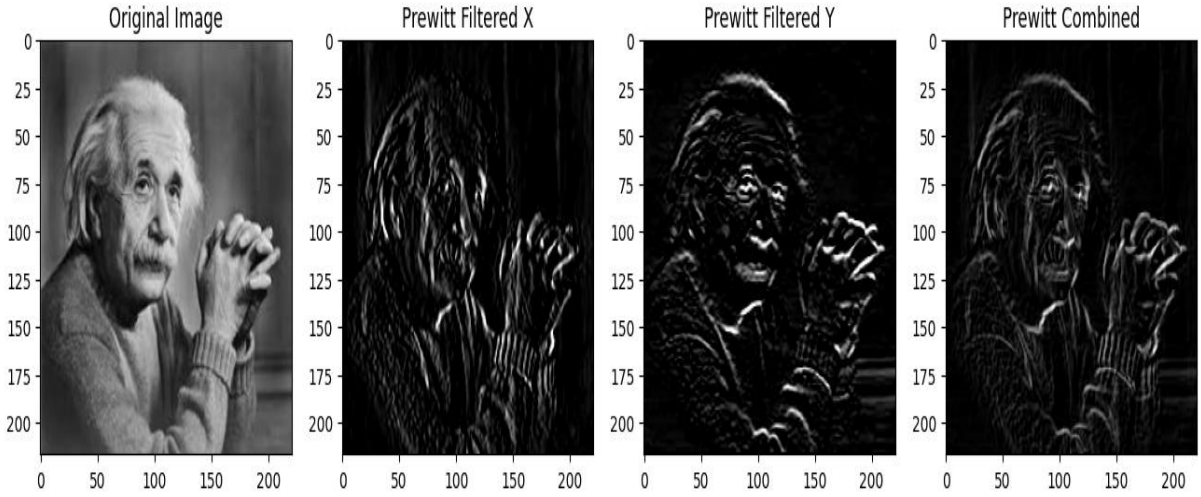


- **Derivative filtering**
  1. **Prewitt**
     - **Code**

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Read the image
img = cv2.imread('/content/Picture.png')

# Apply Prewitt filter kernels
# Initialize Prewitt kernel in x-direction
prewitt_kernel_x = np.zeros((3, 3))

# Fill Prewitt kernel values for x-direction
for i in range(3):
    for j in range(3):
        if j == 0:
            prewitt_kernel_x[i, j] = -1
        elif j == 2:
            prewitt_kernel_x[i, j] = 1


# Initialize Prewitt kernel in y-direction
prewitt_kernel_y = np.zeros((3, 3))

# Fill Prewitt kernel values for y-direction
for i in range(3):
    for j in range(3):
        if i == 0:
            prewitt_kernel_y[i, j] = -1
        elif i == 2:
            prewitt_kernel_y[i, j] = 1

# Apply Prewitt filtering
prewitt_filtered_x = cv2.filter2D(img, -1, prewitt_kernel_x)
prewitt_filtered_y = cv2.filter2D(img, -1, prewitt_kernel_y)

# Combine horizontal and vertical edge images
prewitt_combined = cv2.addWeighted(prewitt_filtered_x, 0.5, prewitt_filtered_y, 0.5, 0)

# Display the images
plt.figure(figsize=(16, 6))

plt.subplot(1, 4, 1)
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.title('Original Image')

plt.subplot(1, 4, 2)
plt.imshow(prewitt_filtered_x)
plt.title('Prewitt Filtered X')

plt.subplot(1, 4, 3)
```

```
plt.imshow(prewitt_filtered_y)
plt.title('Prewitt Filtered Y')

plt.subplot(1, 4, 4)
plt.imshow(prewitt_combined)
plt.title('Prewitt Combined')

plt.show()
```

- **Result**



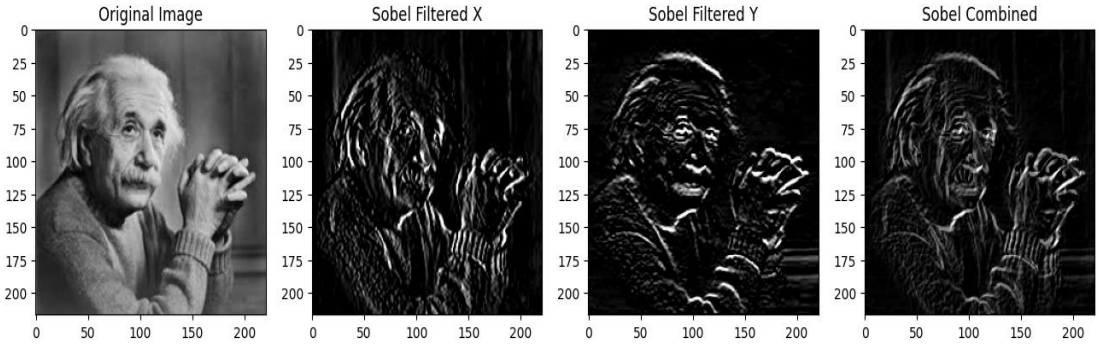
2. **Sobel**

- **Code**

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Read the image
img = cv2.imread('/content/Picture.png')

# Apply Sobel filter kernels
# Initialize Sobel kernel in x-direction
sobel_kernel_x = np.array([[-1, 0, 1],
                           [-2, 0, 2],
                           [-1, 0, 1]])

# Initialize Sobel kernel in y-direction
sobel_kernel_y = np.array([[-1, -2, -1],
                           [0, 0, 0],
                           [1, 2, 1]])

# Apply Sobel filtering
sobel_filtered_x = cv2.filter2D(img, -1, sobel_kernel_x)
sobel_filtered_y = cv2.filter2D(img, -1, sobel_kernel_y)

# Combine horizontal and vertical edge images
sobel_combined = cv2.addWeighted(sobel_filtered_x, 0.5, sobel_filtered_y, 0.5, 0)

# Display the images
plt.figure(figsize=(16, 6))
```

```
plt.subplot(1, 4, 1)
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.title('Original Image')

plt.subplot(1, 4, 2)
plt.imshow(sobel_filtered_x, cmap='gray')
plt.title('Sobel Filtered X')

plt.subplot(1, 4, 3)
plt.imshow(sobel_filtered_y, cmap='gray')
plt.title('Sobel Filtered Y')

plt.subplot(1, 4, 4)
plt.imshow(sobel_combined, cmap='gray')
plt.title('Sobel Combined')

plt.show()
```

- **Result**



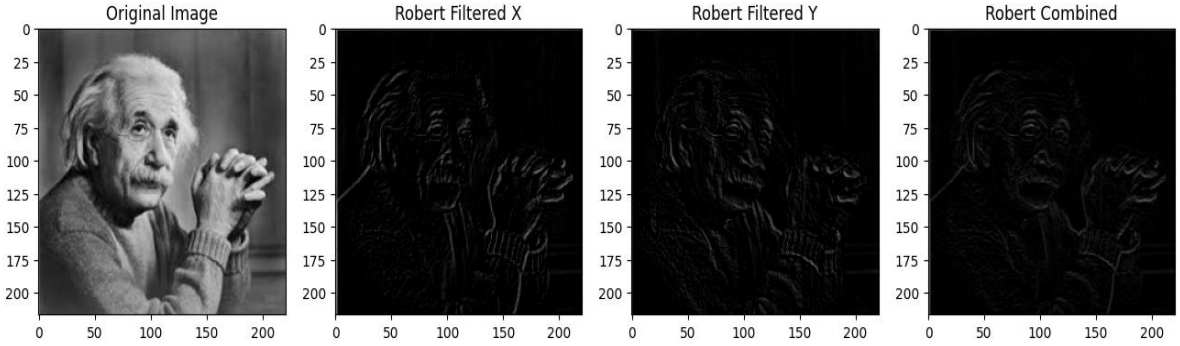
3. **Robert**

- **Code**

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Read the image
img = cv2.imread('/content/Picture.png')

# Apply Robert filter kernels
# Initialize Robert kernel in x-direction
robert_kernel_x = np.array([[1, 0],
                            [0, -1]])

# Initialize Robert kernel in y-direction
robert_kernel_y = np.array([[0, 1],
                            [-1, 0]])

# Apply Robert filtering
robert_filtered_x = cv2.filter2D(img, -1, robert_kernel_x)
robert_filtered_y = cv2.filter2D(img, -1, robert_kernel_y)

# Combine horizontal and vertical edge images
```

```
robert_combined = cv2.addWeighted(robert_filtered_x, 0.5, robert_filtered_y, 0.5, 0)

# Display the images
plt.figure(figsize=(16, 6))

plt.subplot(1, 4, 1)
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.title('Original Image')

plt.subplot(1, 4, 2)
plt.imshow(robert_filtered_x, cmap='gray')
plt.title('Robert Filtered X')

plt.subplot(1, 4, 3)
plt.imshow(robert_filtered_y, cmap='gray')
plt.title('Robert Filtered Y')

plt.subplot(1, 4, 4)
plt.imshow(robert_combined, cmap='gray')
plt.title('Robert Combined')

plt.show()
```

- **Result**



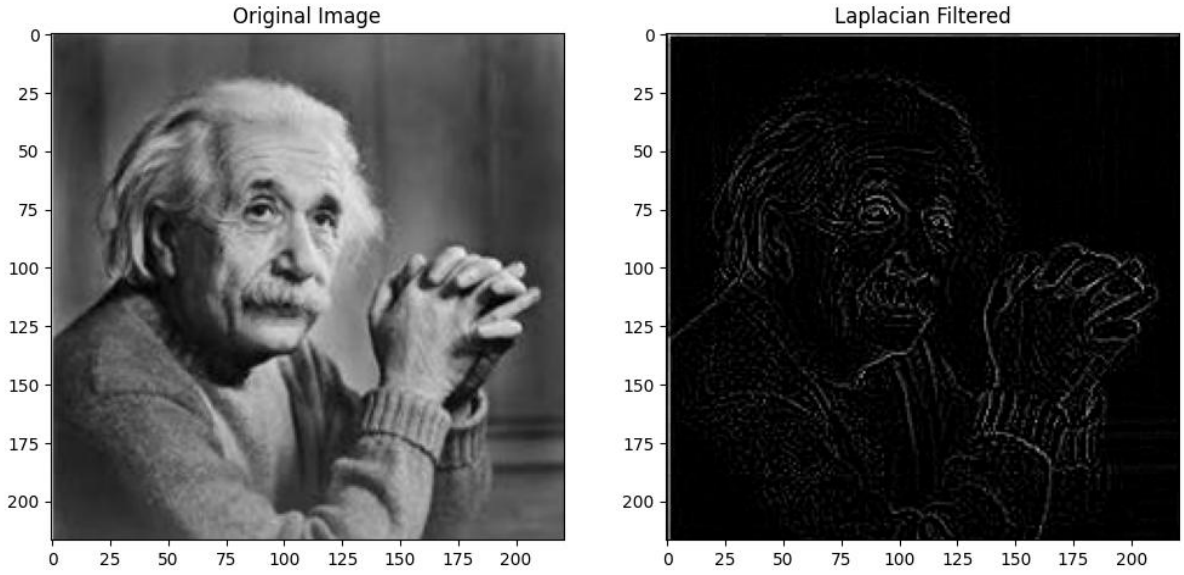
4. Laplacian

- **Code**

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Read the image
img = cv2.imread('/content/Picture.png')

# Apply Laplacian filter kernel
laplacian_kernel = np.array([[0, 1, 0],
                             [1, -4, 1],
                             [0, 1, 0]])

# Apply Laplacian filtering
laplacian_filtered = cv2.filter2D(img, -1, laplacian_kernel)

# Display the images
plt.figure(figsize=(12, 6))
```

```
plt.subplot(1, 2, 1)
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.title('Original Image')

plt.subplot(1, 2, 2)
plt.imshow(laplacian_filtered, cmap='gray')
plt.title('Laplacian Filtered')

plt.show()
```

- **Result**



Discussion:

Each sharpening filter exhibits unique characteristics and performance in enhancing image sharpness:

- High pass and high boost filters effectively emphasize high-frequency components, resulting in sharpened edges and enhanced details.
- Prewitt and Sobel filters provide gradient information, making them suitable for edge detection tasks with directional sensitivity.
- Robert filter, being a simpler form of edge detection, offers quick results but may lack accuracy compared to Sobel and Prewitt.
- Laplacian filter highlights regions of rapid intensity change, effectively enhancing edges and features.

Conclusion:

In conclusion, sharpening filters are valuable tools in image processing for enhancing image clarity and definition. The selection of a specific sharpening filter depends on factors such as the desired level of sharpening, computational complexity, and noise robustness. By understanding the characteristics and performance of each filter, practitioners can choose the most suitable method for their specific image enhancement tasks, ultimately improving visual quality and aiding in image analysis.