Heaven's Light is Our Guide

Department of Electronics & Telecommunication Engineering

Rajshahi University of Engineering & Technology

Laboratory Report

on

ETE 4226: Sessional Based on ETE 4225

Submitted by:

Md. Darul Atfal Palash

Roll No. 1804005

Submitted to:

Dr. Shah Ariful Hoque Chowdhury

Associate Professor

Dept. of ETE, RUET

Contents

List of Figures

Heaven's Light is Our Guide

Rajshahi University of Engineering & Technology

Department of Electronics & Telecommunication Engineering

ETE 4226: Sessional Based on ETE 4225

Experiment 1

Study of Geometric Transformations of Image.

Submitted by:

Md. Darul Atfal Palash

Roll: 1804005

Session: 2018-19

Submitted to:

Dr. Shah Ariful Hoque Chowdhury

Associate Professor

Dept. of ETE, RUET

Date of Experiment : 01/10/2023

Date of Submission : 25/01/2024

Report	**(Teacher's Section)**	**Viva**
☐ Excellent		☐ Excellent
☐ Very Good		☐ Very Good
☐ Good	_____	☐ Good
☐ Moderate	Signature	☐ Moderate
☐ Poor		☐ Poor

Experiment 1

Study of Geometric Transformations of Image.

Objectives

The main objectives of this experiment are:

- To understand and implement image zooming, shrinking, rotation, scaling, and histogram equalization using Python.
- To observe the effects of each image processing technique on the visual appearance of digital images.
- To analyze the computational efficiency of the implemented algorithms.

1.1 Required Apparatus/Softwares

– Python.

– A Highly Configured PC.

1.2 Program Code and Output

1.2.1 Image Zooming

Code :

```
import cv2
import matplotlib.pyplot as plt
input_image = cv2.imread('cameraman.jpg')
# Define the coordinates of the ROI (region of interest)
x, y, width, height = 100, 100, 200, 200
roi = input_image[y:y+height, x:x+width]
zoomed_in_roi = cv2.resize(roi, (width * 2, height * 2))
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.title("Original Image")
plt.imshow(cv2.cvtColor(input_image, cv2.COLOR_BGR2RGB))
```

```
plt.subplot(1, 2, 2)
plt.title("Zoomed-In Part")
plt.imshow(cv2.cvtColor(zoomed_in_roi, cv2.COLOR_BGR2RGB))
plt.show()
```

Output:

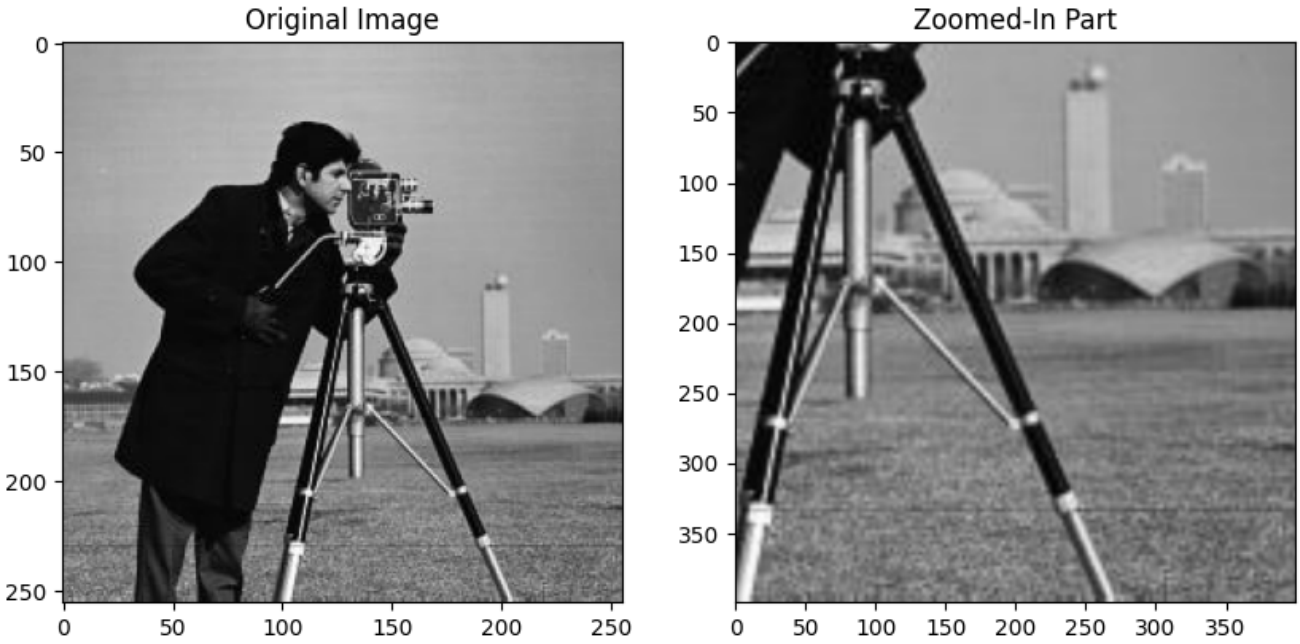

Figure 1.1: Original Image and Zoomed-in Part of the Image.

1.2.2 Image Shrinking

Code :

```
import cv2
import numpy as np
image = cv2.imread('cameraman.jpg')
scale_x = .5
scale_y = .5
translation_matrix = np.array([[scale_x, 0, 0], [0, scale_y,
    0]], dtype=np.float32)
```

```
scaled_image = cv2.warpAffine(image, translation_matrix, (int(
    image.shape[1]*scale_x), int(image.shape[0]*scale_y)))

print('Original Image')
cv2_imshow(image)
print('Shrinked Image')
cv2_imshow(scaled_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Output:

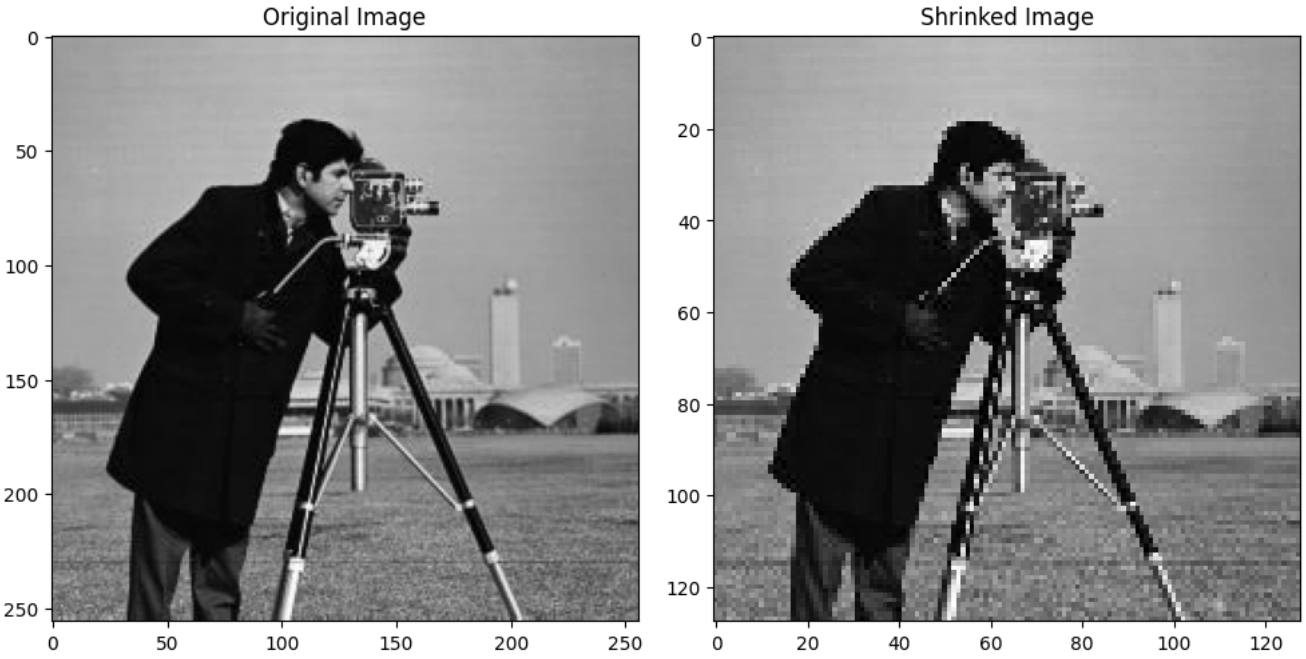

Figure 1.2: Original Image and Shrinked Image.

1.2.3 Image Rotation

Code :

```
import cv2
import numpy as np
from google.colab.patches import cv2_imshow
```

```
image = cv2.imread('/content/cameraman.jpg')
# Image rotation
h, w, _ = image.shape
angle = 90
rotation_matrix = cv2.getRotationMatrix2D((w/2, h/2), angle,
   1)
rotated_image = cv2.warpAffine(image, rotation_matrix, (w, h))
plt.subplot(1, 2, 1)
plt.title("Original Image")
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.subplot(1, 2, 2)
plt.title("Rotated Image")
plt.imshow(cv2.cvtColor(rotated_image, cv2.COLOR_BGR2RGB))
plt.show()
```

Output:

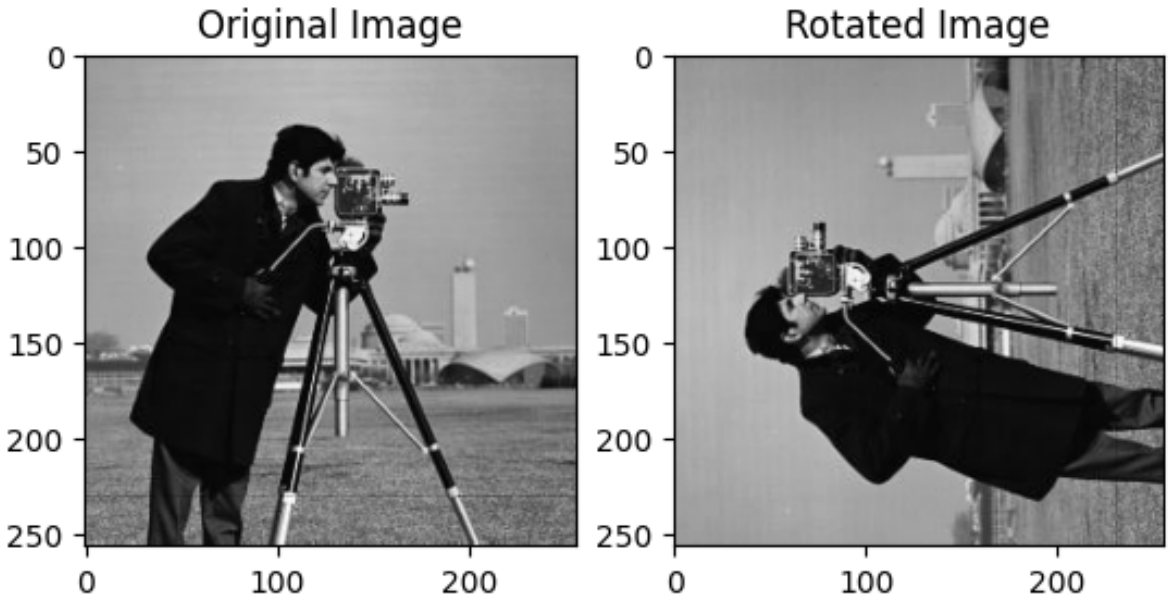

Figure 1.3: Original Image and Rotated Image.

1.2.4 Image Scalling

Code :

```
import cv2
import numpy as np
image = cv2.imread('cameraman.jpg')

# Define the scale factors for x and y directions
scale_x = 1.5
scale_y = 1.5

translation_matrix = np.array([[scale_x, 0, 0], [0, scale_y,
    0]], dtype=np.float32)

scaled_image = cv2.warpAffine(image, translation_matrix, (int(
    image.shape[1]*scale_x), int(image.shape[0]*scale_y)))

# Display the original and scaled images
plt.subplot(1, 2, 1)
plt.title("Original Image")

plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.subplot(1, 2, 2)
plt.title("Scaled Image")
plt.imshow(cv2.cvtColor(scaled_image, cv2.COLOR_BGR2RGB))

plt.show()
```

Output:

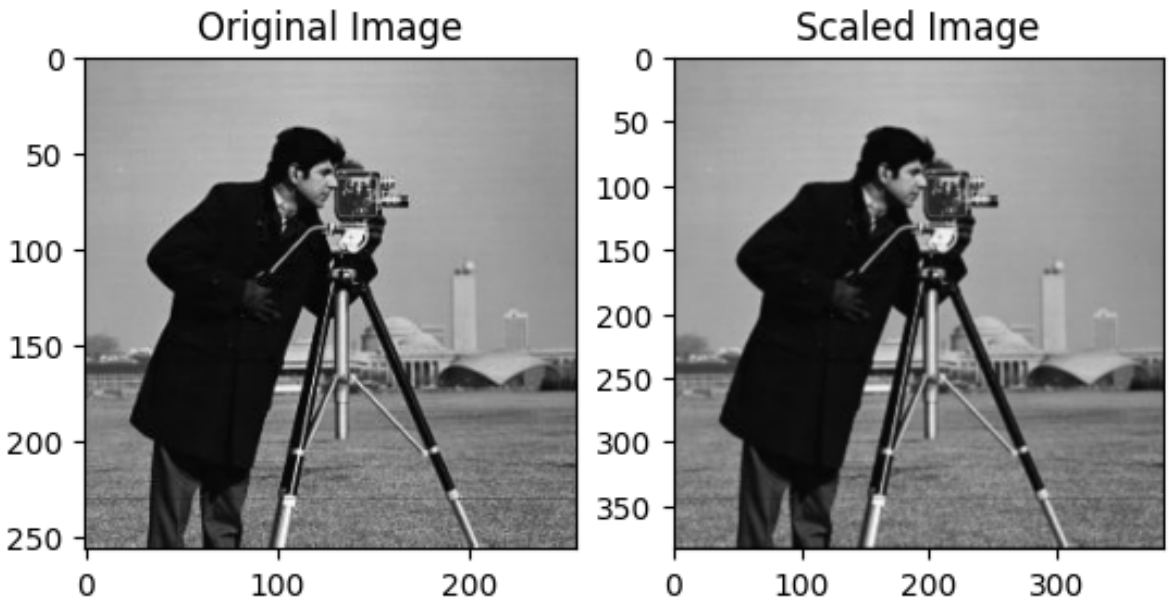

Figure 1.4: Original Image and Scaled Image.

1.2.5 Video Making with Image Rotation, Shearing, and Translation

Code :

```
import cv2
import numpy as np
image = cv2.imread('/content/cameraman.jpg')

# Get the dimensions of the image
height, width = image.shape[:2]
# Create a video writer object
fourcc = cv2.VideoWriter_fourcc(*'mp4v')
video_writer = cv2.VideoWriter('/content/transformations_video
    .mp4', fourcc, 1, (width * 2, height * 2))

# Number of frames
num_frames = 30
```

```
13
14 for i in range(num_frames):
15     # Vary the rotation angle
16     rotation_angle = i * 360 / num_frames
17     rotation_matrix = cv2.getRotationMatrix2D((width / 2,
   height / 2), rotation_angle, 1)
18     rotated_image = cv2.warpAffine(image, rotation_matrix, (
   width, height))
19
20     # Vary the shearing factor
21     shearing_factor = i / num_frames
22     shearing_matrix = np.array([[1, shearing_factor, 0], [0,
   1, 0]], dtype=np.float32)
23     sheared_image = cv2.warpAffine(rotated_image,
   shearing_matrix, (width, height))
24
25     # Vary the translation
26     translation_x = int(i * width / num_frames)
27     translation_y = int(i * height / num_frames)
28     translation_matrix = np.array([[1, 0, translation_x], [0,
   1, translation_y]], dtype=np.float32)
29     translated_image = cv2.warpAffine(sheared_image,
   translation_matrix, (width, height))
30
31     # Create a frame
32     frame = np.zeros((height * 2, width * 2, 3), dtype=np.
   uint8)
33     # Place the images in different positions
34     frame[:height, :width] = image
35     frame[:height, width:] = rotated_image
```

```
36      frame[height:, :width] = sheared_image
37      frame[height:, width:] = translated_image
38
39      cv2.putText(frame, 'Original', (10, 30), cv2.
    FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2)
40      cv2.putText(frame, 'Rotated', (width + 10, 30), cv2.
    FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2)
41      cv2.putText(frame, 'Sheared', (10, height + 30), cv2.
    FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2)
42      cv2.putText(frame, 'Translated', (width + 10, height + 30)
    , cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2)
43      video_writer.write(frame)
44  # Release the video writer
45  video_writer.release()
```

Output:

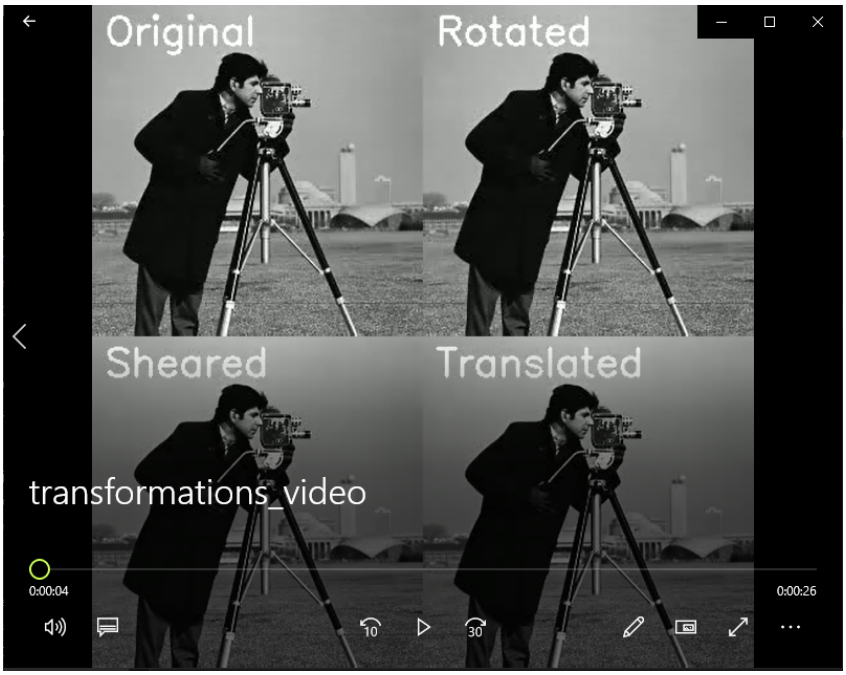

Figure 1.5: Video with with Image Rotation, Shearing, and Translation.

1.3 Conclusion

In conclusion, the study of image processing techniques, including zooming, shrinking, rotation, and scaling, was conducted using Python programming within the Google Colab platform. Each operation was applied to input images utilizing the OpenCV library, a powerful tool for computer vision tasks. Commonly used functions such as ‘imread‘ for reading images, ‘resize‘ for resizing images, ‘warpAffine‘ for geometric transformations, and ‘imshow‘ for displaying images were employed. Additionally, numpy’s ‘array‘ facilitated efficient data manipulation, while Matplotlib’s ‘imshow‘ aided in visualizing the results. The experimentation culminated in creating a video incorporating image rotation, shearing, and translation, showcasing the versatility of the OpenCV library. This study demonstrated the significance of these image processing techniques in computer vision applications and their seamless implementation using Python libraries.

Heaven's Light is Our Guide

Rajshahi University of Engineering & Technology

Department of Electronics & Telecommunication Engineering

ETE 4226: Sessional Based on ETE 4225

Experiment 2

Study of Histogram Equalization, Image Intensity Transformations, and Image Filtering in Spatial Domain.

Submitted by:

Md. Darul Atfal Palash

Roll: 1804005

Session: 2018-19

Submitted to:

Dr. Shah Ariful Hoque Chowdhury

Associate Professor

Dept. of ETE, RUET

Date of Experiment : 15/10/2024

Date of Submission : 25/01/2024

Report	**(Teacher's Section)**	**Viva**
☐ Excellent		☐ Excellent
☐ Very Good		☐ Very Good
☐ Good	_____	☐ Good
☐ Moderate	Signature	☐ Moderate
☐ Poor		☐ Poor

Experiment 2

Study of Histogram Equalization, Image Intensity Transformations, and Image Filtering in Spatial Domain.

Objectives

The primary objectives of this experiment include:

- Understanding the theoretical foundations of Histogram Equalization, Image Intensity Transformations, and Image Filtering in the Spatial Domain.
- Implementing these techniques using Python programming.
- Analyzing the effects of each technique on digital images.
- Drawing conclusions on the applicability and significance of these techniques in image processing.

2.1 Required Apparatus/Softwares

– Python.

– A Highly Configured PC.

2.2 Program Code and Output

2.2.1 Histogram Equalization for Gray Image

Code :

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
image = cv2.imread('/content/einstein.jpg', cv2.
   IMREAD_GRAYSCALE)  # Convert to grayscale for histogram
   equalization
# Apply histogram equalization
equ = cv2.equalizeHist(image)
```

```
# Display images and their histograms
plt.figure(figsize=(12, 6))
plt.subplot(2, 2, 1)
plt.imshow(image, cmap='gray')
plt.title('Original Image')
plt.subplot(2, 2, 3)
plt.hist(image.ravel(), 256, [0, 256])
plt.title('Histogram (Original)')
plt.subplot(2, 2, 2)
plt.imshow(equ, cmap='gray')
plt.title('Histogram Equalized Image')
plt.subplot(2, 2, 4)
plt.hist(equ.ravel(), 256, [0, 256])
plt.title('Histogram (Equalized)')
plt.show()
```

Output:

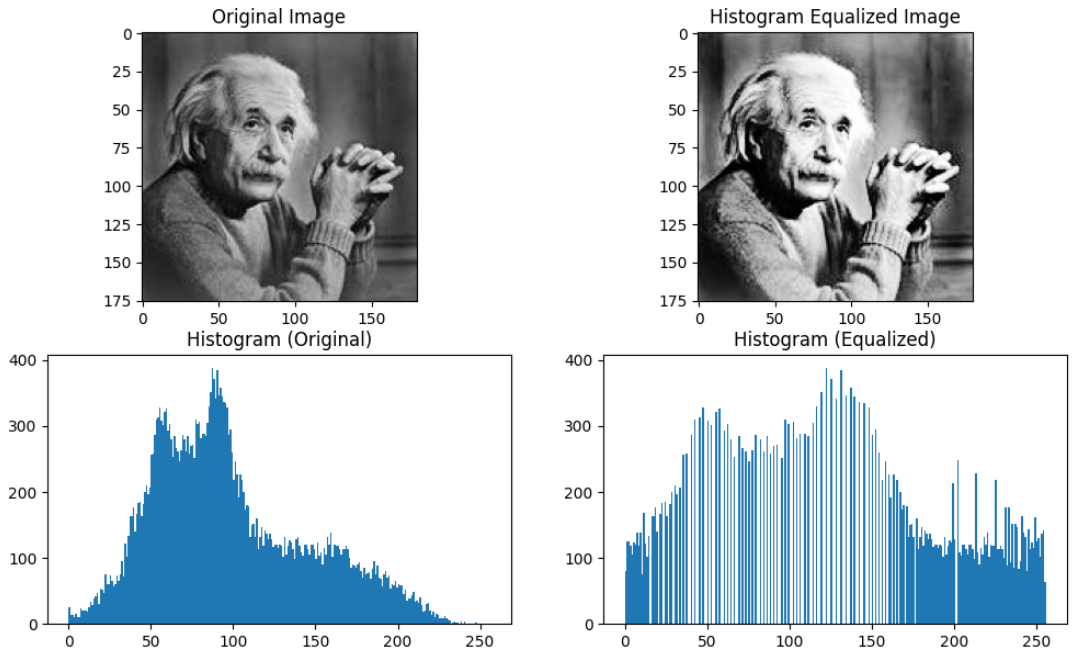

Figure 2.1: Histogram of original image and histogram equalized image.

2.2.2 Histogram Equalization for Color Image

Code :

```
import cv2
import matplotlib.pyplot as plt
image= cv2.imread("/content/einstein_img2.jpg")

ycrcb_img = cv2.cvtColor(image, cv2.COLOR_BGR2YCrCb)
ycrcb_img[:, :, 0] = cv2.equalizeHist(ycrcb_img[:, :, 0])
equalized_img = cv2.cvtColor(ycrcb_img, cv2.COLOR_YCrCb2RGB)
original_img = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

fig = plt.figure(figsize=(12,8))
fig.add_subplot(2,2,1)
plt.imshow(original_img)
plt.title("Original Image")
plt.axis(False)
fig.add_subplot(2,2,2)
plt.imshow(equalized_img)
plt.title("Histogram Equalized Image")
plt.axis(False)
fig.add_subplot(2,2,3)
color = ('b','g','r')

for channel,col in enumerate(color):
  histr = cv2.calcHist([image],[channel],None,[256],[0,256])
  plt.plot(histr,color = col)
  plt.xlim([0,256])
plt.title('Histogram of Original Color Image')
fig.add_subplot(2,2,4)
```

```
28
29 for channel,col in enumerate(color):
30   histr = cv2.calcHist([equalized_img],[channel],None
      ,[256],[0,256])
31   plt.plot(histr,color = col)
32   plt.xlim([0,256])
33 plt.title('Histogram of Equalized Color Image')
34 plt.show()
```

Output:

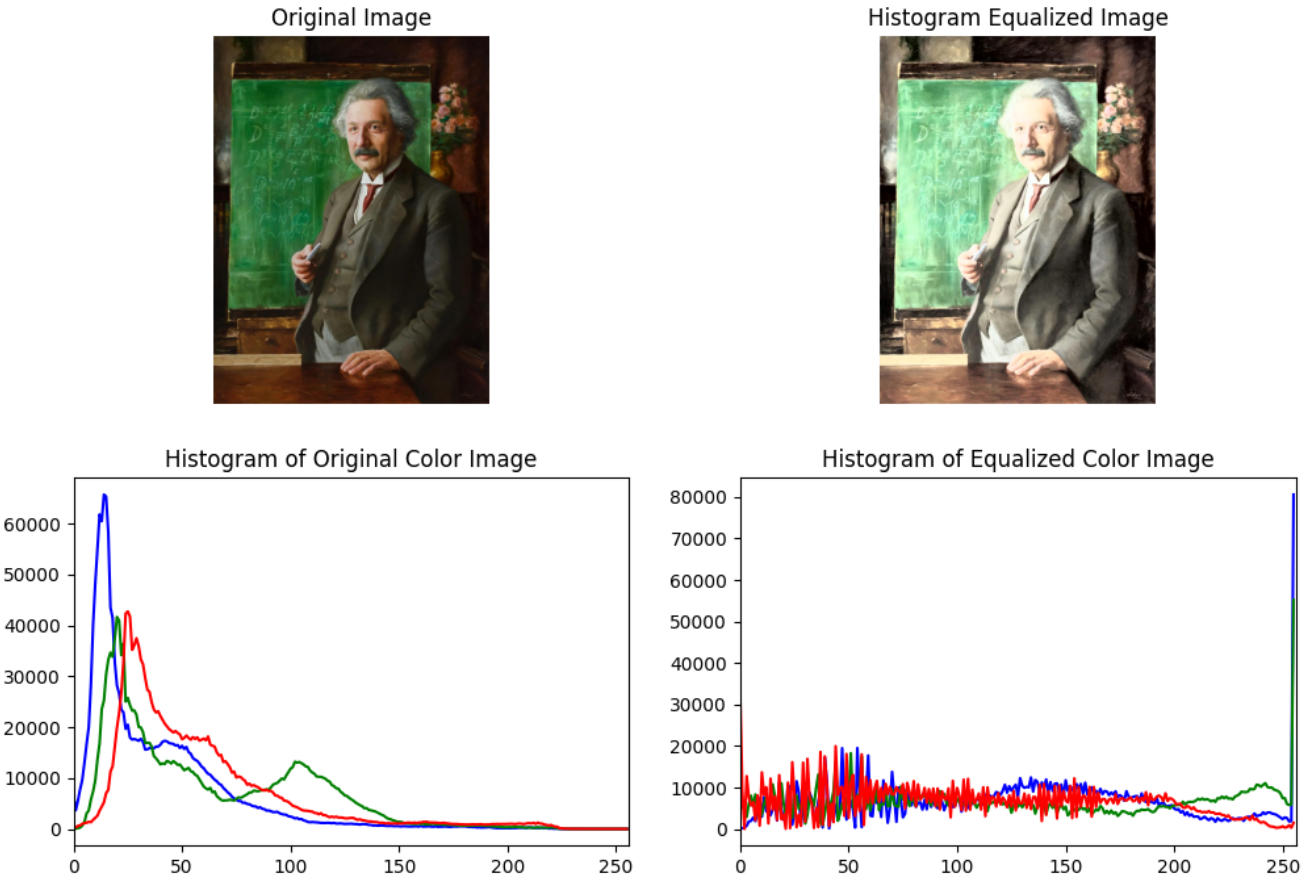

Figure 2.2: Histogram of original image and histogram equalized image.

2.2.3 Log Transform of an Image

Code :

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
image = cv2.imread("/content/einstein.jpg")
c = 255 / np.log(1 + np.max(image))
log_image = c * (np.log(image + 1))
log_image = np.array(log_image, dtype = np.uint8)
fig = plt.figure(figsize=(20,20 ))
disp_img1= cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
disp_img2= cv2.cvtColor(log_image, cv2.COLOR_BGR2RGB)
fig.add_subplot(1,2,1)
plt.imshow(disp_img1)
plt.title("Original")
fig.add_subplot(1,2,2)
plt.imshow(disp_img2); plt.title("Log Transformed")
```

Output:

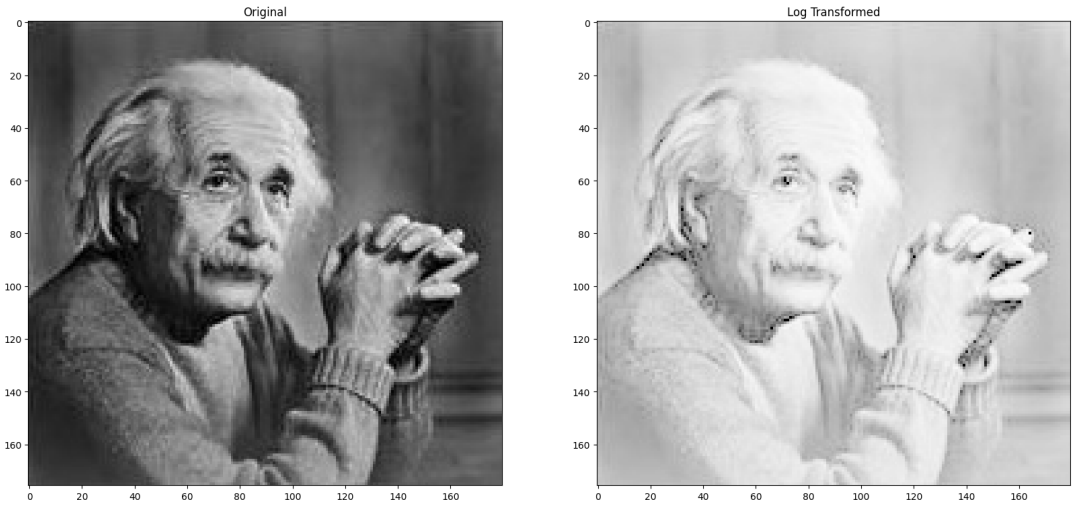

Figure 2.3: Original Image and Log Transformed Image.

2.2.4 Inverse Log Transform

Code :

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
image = cv2.imread('/content/einstein.jpg', cv2.
    IMREAD_GRAYSCALE)  # Convert to grayscale for histogram
    equalization

equ = cv2.equalizeHist(image)
c = 255 / np.log(1 + np.max(image))
log_image = c * (np.log(image + 1))
log_image = np.array(log_image, dtype=np.uint8)

# Apply inverse log transformation method
inverse_log_image = np.exp(log_image / c) - 1
inverse_log_image = np.clip(inverse_log_image, 0, 255).astype(
    np.uint8)

# Display images and their histograms
plt.figure(figsize=(12, 8))

# Input image and histogram
plt.subplot(2, 2, 1)
plt.imshow(image, cmap='gray')
plt.title('Original Image')
plt.subplot(2, 2, 3)
plt.hist(image.ravel(), 256, [0, 256])
plt.title('Histogram (Original)')
```

```
25
26 # Inverse log transformed image and histogram
27 plt.subplot(2, 2, 2)
28 plt.imshow(inverse_log_image, cmap='gray')
29 plt.title('Inverse Log Transformed Image')
30 plt.subplot(2, 2, 4)
31 plt.hist(inverse_log_image.ravel(), 256, [0, 256])
32 plt.title('Histogram (Inverse Log Transformed)')
33 plt.show()
```

Output:

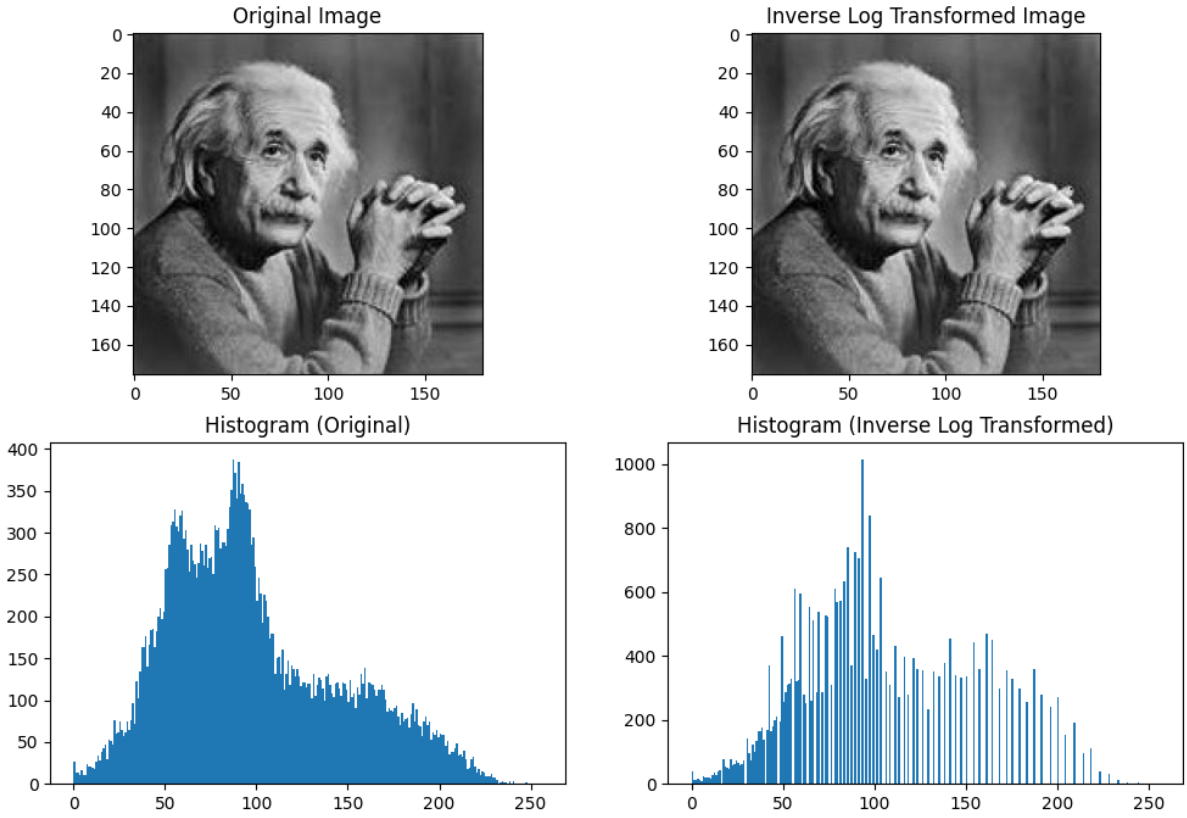

Figure 2.4: Original Image and Inverse Log Transformed Image.

2.2.5 Power Law (Gamma) Transformation

Code :

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

# Read an image
image = cv2.imread('/content/imgh.jpeg', cv2.IMREAD_GRAYSCALE)
     # Convert to grayscale for histogram equalization

# Apply histogram equalization
equ = cv2.equalizeHist(image)

# Define a range of gamma values
gamma_values = [0.2, 1.4, 4.0]

# Display images and their histograms for different gamma
   values
plt.figure(figsize=(13, 10))
rows = len(gamma_values) + 1  # Number of rows in the subplot
   grid

# Original image and histogram
plt.subplot(rows, 3, 1)
plt.imshow(image, cmap='gray')
plt.title('Original Image')
plt.subplot(rows, 3, 2)
plt.hist(image.ravel(), 256, [0, 256])
plt.title('Histogram (Original)')
```

```
25
26 # Loop through gamma values
27 for i, gamma in enumerate(gamma_values, start=1):
28     # Apply power law transformation method
29     power_log_image = np.power(image / 255.0, gamma) * 255
30
31     # Specify the data type so that
32     # float value will be converted to int
33     power_log_image = np.array(power_log_image, dtype=np.uint8
       )
34
35     # Display power law transformed image and histogram
36     plt.subplot(rows, 3, i * 3 + 1)
37     plt.imshow(power_log_image, cmap='gray')
38     plt.title(f'Power Law Transformed (Gamma={gamma})')
39
40     plt.subplot(rows, 3, i * 3 + 2)
41     plt.hist(power_log_image.ravel(), 256, [0, 256])
42     plt.title(f'Histogram (Gamma={gamma})')
43
44 # Show the plots
45 plt.tight_layout()
46 plt.show()
```

Output:

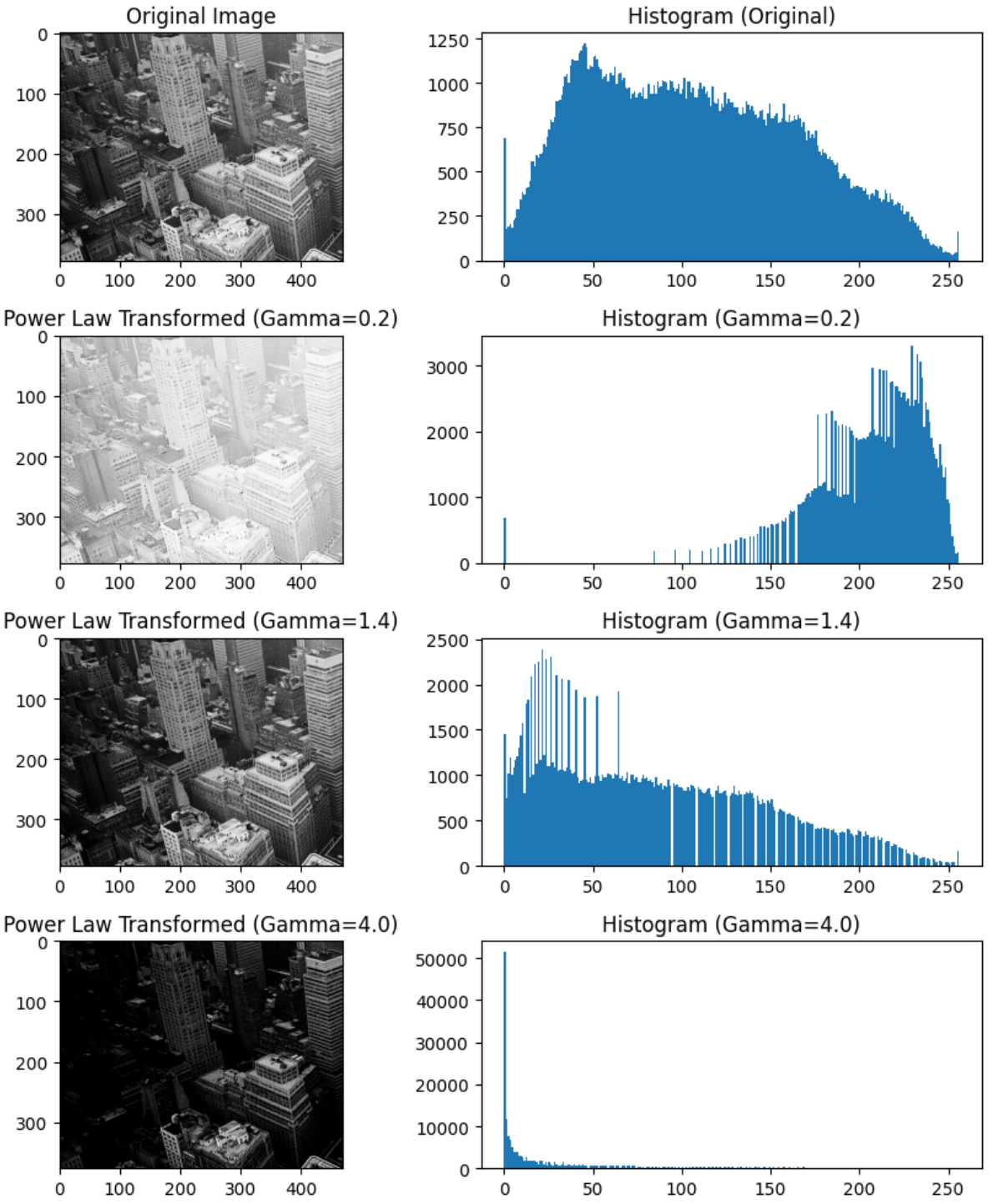

Figure 2.5: Power Law (Gamma) Transformation with Different Value of Gamma (0.2, 1.4, 4.0).

2.2.6 Image Negatives

Code :

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
image = cv2.imread('/content/einstein.jpg', cv2.
   IMREAD_GRAYSCALE)

# Compute the negative of the image
negative_image = 255 - image

# Display images and their histograms
plt.figure(figsize=(12, 8))
# Input image and histogram
plt.subplot(2, 2, 1)
plt.imshow(image, cmap='gray')
plt.title('Original Image')
plt.subplot(2, 2, 2)
plt.hist(image.ravel(), 256, [0, 256])
plt.title('Histogram (Original)')
# Negative transformed image
plt.subplot(2, 2, 3)
plt.imshow(negative_image, cmap='gray')
plt.title('Negative Transformed Image')
plt.subplot(2, 2, 4)
plt.hist(negative_image.ravel(), 256, [0, 256])
plt.title('Histogram (Negative Transformed)')

plt.show()
```

Output:

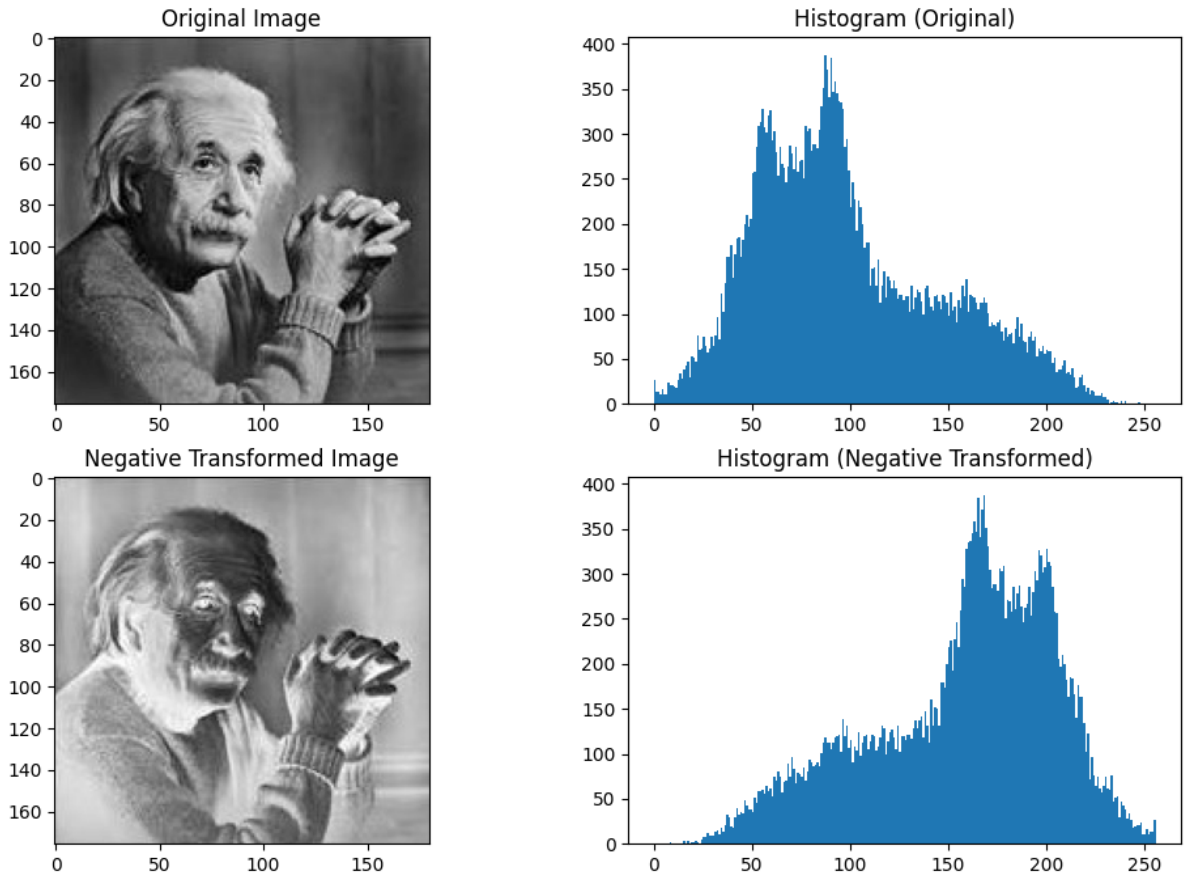

Figure 2.6: Original Image and Negative Transformed Image.

2.2.7 Some Basic Intensity Transformations

Code :

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

# Load an image from file
image = cv2.imread('/content/einstein.jpg', cv2.
    IMREAD_GRAYSCALE)  # Convert to grayscale

```

```
# Check if the image was loaded successfully
if image is None:
    print("Image not found. Please provide a valid image path.")
else:
    # Plot original and transformed images side by side with histograms
    plt.figure(figsize=(18, 16))

    transformations = [
        ("Original", image),
        ("Log Transform", np.log1p(image.astype(float)).astype(np.uint8)),
        ("Inverse Log Transform", np.expm1(np.log1p(image.astype(float))).astype(np.uint8)),
        ("Power-law Transform", (np.power(image / 255.0, 0.5) * 255.0).astype(np.uint8)),
        ("Gamma Correction (Gamma=0.4)", (np.power(image / 255.0, 0.4) * 255.0).astype(np.uint8)),
        ("Negative Image", 255 - image),
        ("Smoothing (Gaussian Blur)", cv2.GaussianBlur(image, (5, 5), 0)),
        ("Sharpening", cv2.filter2D(image, -1, np.array([[-1, -1, -1], [-1, 9, -1], [-1, -1, -1]])))
    ]

    for i in range(0, len(transformations), 2):
        title1, transformed_image1 = transformations[i]
        title2, transformed_image2 = transformations[i + 1]

```

```
30          # Display original and log-transformed images side by
    side
31          plt.subplot(4, 4, i // 2 * 4 + 1)
32          plt.imshow(transformed_image1, cmap='gray')  # Use
    cmap='gray' for grayscale images
33          plt.title(title1)
34          plt.axis('off')
35
36          plt.subplot(4, 4, i // 2 * 4 + 2)
37          plt.imshow(transformed_image2, cmap='gray')  # Use
    cmap='gray' for grayscale images
38          plt.title(title2)
39          plt.axis('off')
40
41          # Display histograms below the images
42          plt.subplot(4, 4, i // 2 * 4 + 3)
43          plt.hist(transformed_image1.ravel(), bins=256, range
    =[0, 256], color='blue', alpha=0.7)
44          plt.title(f'{title1} Histogram')
45          plt.xlabel('Pixel Value')
46          plt.ylabel('Frequency')
47
48          plt.subplot(4, 4, i // 2 * 4 + 4)
49          plt.hist(transformed_image2.ravel(), bins=256, range
    =[0, 256], color='blue', alpha=0.7)
50          plt.title(f'{title2} Histogram')
51          plt.xlabel('Pixel Value')
52          plt.ylabel('Frequency')
53      plt.tight_layout()
54      plt.show()
```

Output:

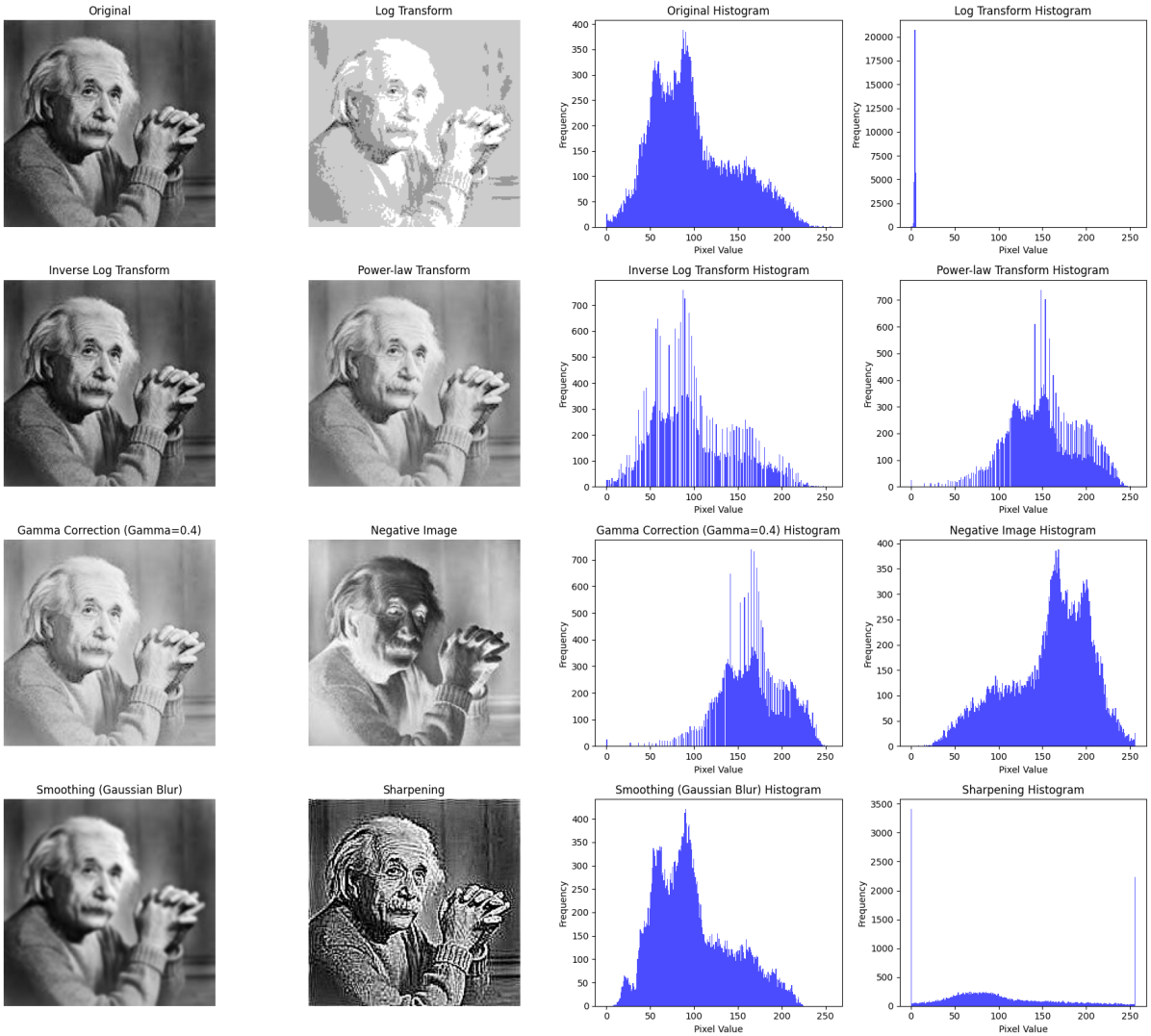

Figure 2.7: Original Image and Transformed Image with Corresponding Histogram.

2.2.8 Image Blurring with Simple, Median, Gaussian Blur

Code :

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
image = cv2.imread('/content/einstein_img3.jpg')
# Apply a simple blur
```

```
blurred = cv2.blur(image, (5, 5))
# Apply a median blur
median_blurred = cv2.medianBlur(image, 5)
# Apply a Gaussian blur
gaussian_blurred = cv2.GaussianBlur(image, (5, 5), 0)

# Plot the original and blurred images side by side
plt.figure(figsize=(8, 9))
plt.subplot(4, 2, 1), plt.imshow(cv2.cvtColor(image, cv2.
   COLOR_BGR2RGB)), plt.title('Original')
plt.subplot(4, 2, 2), plt.imshow(cv2.cvtColor(blurred, cv2.
   COLOR_BGR2RGB)), plt.title('Blurred')
plt.subplot(4, 2, 3), plt.imshow(cv2.cvtColor(median_blurred,
   cv2.COLOR_BGR2RGB)), plt.title('Median Blurred')
plt.subplot(4, 2, 4), plt.imshow(cv2.cvtColor(gaussian_blurred
   , cv2.COLOR_BGR2RGB)), plt.title('Gaussian Blurred')

# Plot histograms
plt.subplot(4, 2, 5), plt.hist(image.flatten(), 256, [0, 256],
    color='r', histtype='step'), plt.title('Histogram of
   Original Image')
plt.subplot(4, 2, 6), plt.hist(blurred.flatten(), 256, [0,
   256], color='r', histtype='step'), plt.title('Histogram of
   Blurred Image')
plt.subplot(4, 2, 7), plt.hist(median_blurred.flatten(), 256,
   [0, 256], color='r', histtype='step'), plt.title('Histogram
    of Median Blurred')
plt.subplot(4, 2, 8), plt.hist(gaussian_blurred.flatten(),
   256, [0, 256], color='r', histtype='step'), plt.title('
   Histogram of Gaussian Blurred')
```

```
24
25 plt.tight_layout()
26 plt.show()
```

Output:

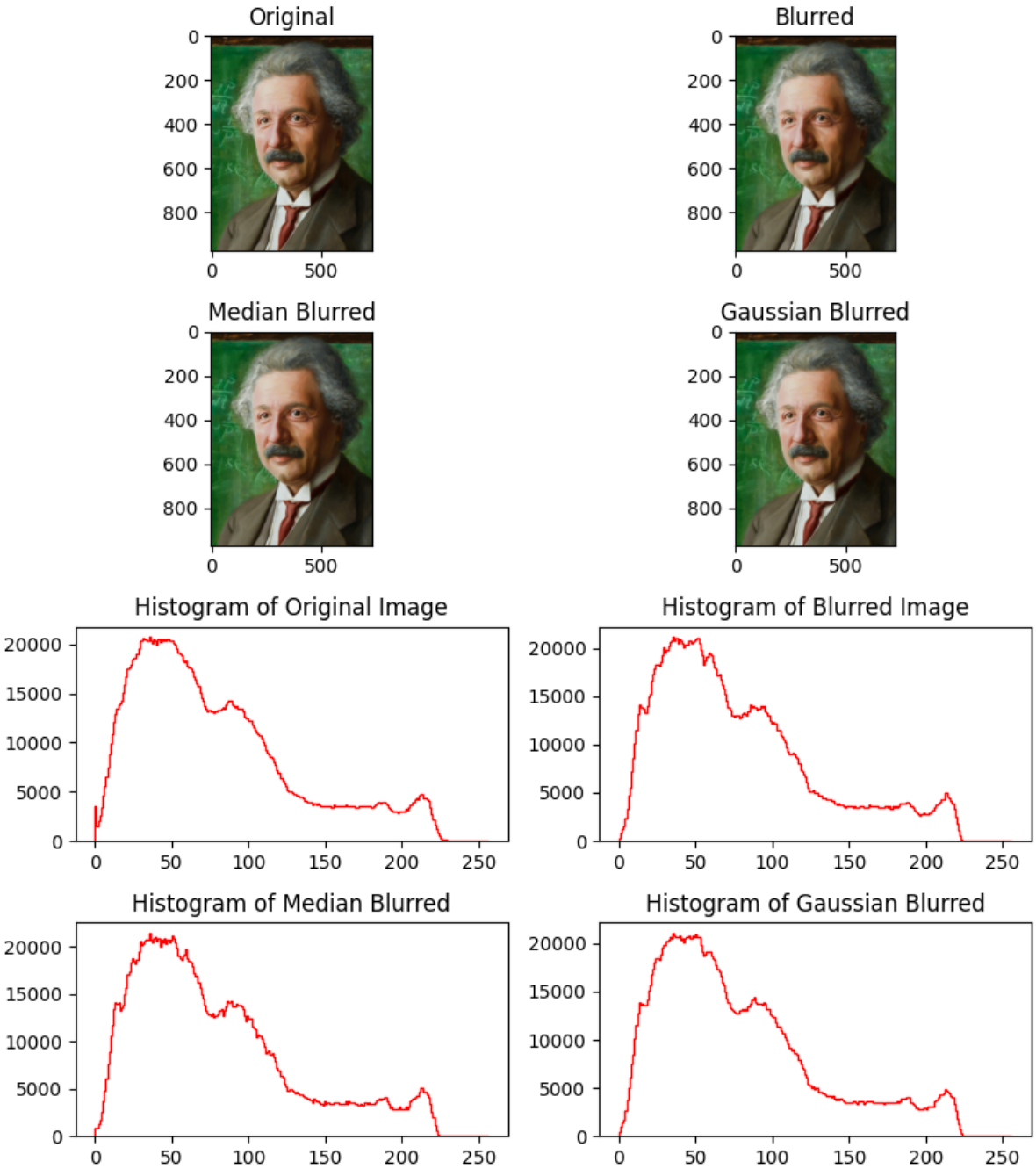

Figure 2.8: Original Image and Blurred Image with Corresponding Histogram.

2.2.9 Image Filtering with Laplacian, Sobel, Scharr Filter

Code :

```
import cv2;import numpy as np
from matplotlib import pyplot as plt
image = cv2.imread('/content/einstein.jpg', cv2.
   IMREAD_GRAYSCALE)
# Apply Laplacian filter
laplacian = cv2.Laplacian(image, cv2.CV_64F)
laplacian_sharpened = image - laplacian
# Apply Sobel filter
sobelx = cv2.Sobel(image, cv2.CV_64F, 1, 0, ksize=3)
sobely = cv2.Sobel(image, cv2.CV_64F, 0, 1, ksize=3)
sobel_sharpened = image - sobelx - sobely
# Apply Scharr filter
scharrx = cv2.Scharr(image, cv2.CV_64F, 1, 0)
scharry = cv2.Scharr(image, cv2.CV_64F, 0, 1)
scharr_sharpened = image - scharrx - scharry
plt.figure(figsize=(9, 8))
plt.subplot(4, 2, 1), plt.imshow(image, cmap='gray'), plt.
   title('Original')
plt.subplot(4, 2, 2), plt.imshow(laplacian_sharpened, cmap='
   gray'), plt.title('Laplacian Sharpened')
plt.subplot(4, 2, 3), plt.imshow(sobel_sharpened, cmap='gray')
   , plt.title('Sobel Sharpened')
plt.subplot(4, 2, 4), plt.imshow(scharr_sharpened, cmap='gray'
   ), plt.title('Scharr Sharpened')
plt.subplot(4, 2, 5), plt.hist(image.flatten(), 256, [0, 256],
    color='r', histtype='step'), plt.title('Original Histogram
   ')
```

```
21 plt.subplot(4, 2, 6), plt.hist(laplacian_sharpened.flatten(),
   256, [0, 256], color='r', histtype='step'), plt.title('
   Laplacian Sharpened Histogram')
22 plt.subplot(4, 2, 7), plt.hist(sobel_sharpened.flatten(), 256,
    [0, 256], color='r', histtype='step'), plt.title('Sobel
   Sharpened Histogram')
23 plt.subplot(4, 2, 8), plt.hist(scharr_sharpened.flatten(),
   256, [0, 256], color='r', histtype='step'), plt.title('
   Scharr Sharpened Histogram')
24 plt.tight_layout();plt.show()
```

Output:

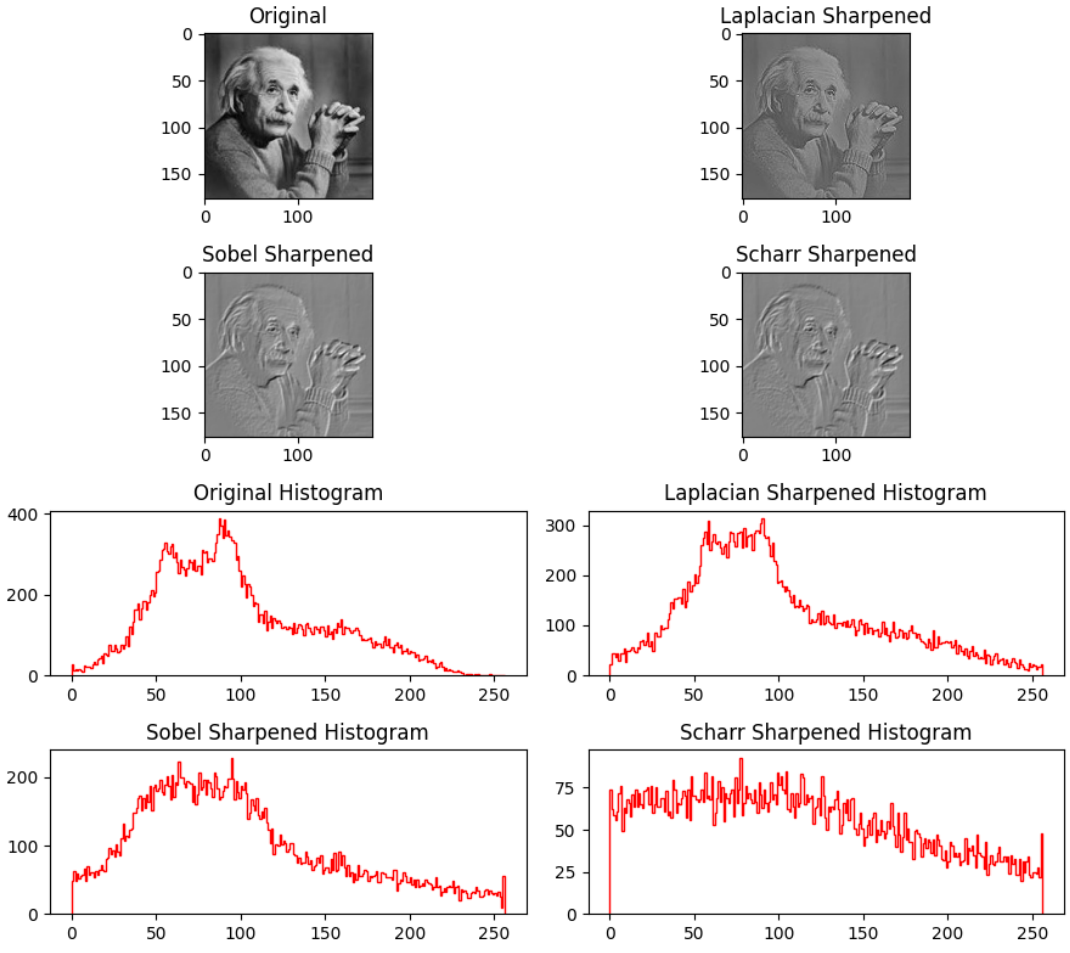

Figure 2.9: Original Image and Filtered Image with Corresponding Histogram.

2.3 Conclusion

In conclusion, the tasks of Histogram Equalization, Image Intensity Transformations, and Image Filtering in the Spatial Domain were systematically conducted using Python programming in the Google Colab environment. Images were employed as inputs, and the OpenCV library operations were performed. Crucial functions, such as ‘imread‘, ‘imshow‘, ‘warpAffine‘, ‘equalizeHist‘, ‘cvtColor‘, ‘calcHist‘, ‘filter2D‘, ‘medianBlur‘, ‘GaussianBlur‘, and various edge detection operators like ‘Sobel‘, ‘Laplacian‘, and ‘Scharr‘ from OpenCV were applied.

The OpenCV library facilitated seamless implementation, allowing for the exploration of diverse image-processing techniques. Numpy and Matplotlib complemented these operations, enhancing data manipulation and visualization. Throughout the experiments, the significance of each technique in image enhancement and analysis became evident. Utilizing these libraries in a collaborative platform like Google Colab offered a streamlined environment for comprehensive exploration. Overall, this study delved into the intricacies of image processing, providing valuable insights into the capabilities of Python and associated libraries for effective spatial domain operations.

Heaven's Light is Our Guide

Rajshahi University of Engineering & Technology

Department of Electronics & Telecommunication Engineering

ETE 4226: Sessional Based on ETE 4225

Experiment 3

Study of Frequency Domain Image Filtering.

Submitted by:

Md. Darul Atfal Palash

Roll: 1804005

Session: 2018-19

Submitted to:

Dr. Shah Ariful Hoque Chowdhury

Associate Professor

Dept. of ETE, RUET

Date of Experiment : 12/11/2023

Date of Submission : 25/01/2024

Report	**(Teacher's Section)**	**Viva**
☐ Excellent		☐ Excellent
☐ Very Good		☐ Very Good
☐ Good	_____	☐ Good
☐ Moderate	Signature	☐ Moderate
☐ Poor		☐ Poor

Experiment 3

Study of Frequency Domain Image Filtering.

Objectives

The primary objectives of this experiment include:

- Understand the theoretical foundations of frequency domain image filtering.
- Implement frequency domain filters, including Ideal, Butterworth, and Gaussian filters, using Python.
- Analyze the effects of different filter parameters on images.
- Compare the performance of the three types of filters in terms of image quality and computational efficiency.

3.1 Required Apparatus/Softwares

– Python.

– A Highly Configured PC.

3.2 Program Code and Output

3.2.1 Ideal Image Filtering

Code :

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

def ideal_filter(shape, D0, filter_type='lowpass'):
    rows, cols = shape
    center_row, center_col = rows // 2, cols // 2

    # Create a meshgrid of distances from the center
```

```
    x = np.arange(cols) - center_col
    y = np.arange(rows) - center_row
    x, y = np.meshgrid(x, y)

    # Compute the distance matrix
    distance = np.sqrt(x^2 + y^2)

    # Create the filter based on the filter type
    if filter_type == 'lowpass':
        H = (distance <= D0).astype(float)
    elif filter_type == 'highpass':
        H = (distance > D0).astype(float)
    elif filter_type == 'bandpass':
        H = ((distance >= D0[0]) & (distance <= D0[1])).astype
    (float)
    elif filter_type == 'bandreject':
        H = ((distance < D0[0]) | (distance > D0[1])).astype(
    float)
    else:
        raise ValueError("Invalid filter type")

    return H

def apply_frequency_filter(image, filter_type='lowpass', D0
    =30):
    # Convert the image to grayscale
    if len(image.shape) == 3:
        image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Apply the Fourier transform
```

```
    f_transform = np.fft.fft2(image)
    f_shift = np.fft.fftshift(f_transform)

    # Create the frequency filter
    rows, cols = image.shape
    H = ideal_filter(image.shape, D0, filter_type)

    # Apply the filter in the frequency domain
    f_filtered = f_shift * H

    # Inverse Fourier transform to get the image back to spatial domain
    img_filtered = np.abs(np.fft.ifft2(np.fft.ifftshift(f_filtered)))

    return img_filtered

# Read an example image
image = cv2.imread('/content/flw.jpg', cv2.IMREAD_GRAYSCALE)

# Apply lowpass filter
lowpass_filtered = apply_frequency_filter(image, filter_type='lowpass', D0=5)

# Apply highpass filter
highpass_filtered = apply_frequency_filter(image, filter_type='highpass', D0=2)

# Apply bandpass filter
```

```
bandpass_filtered = apply_frequency_filter(image, filter_type=
    'bandpass', D0=(5, 40))

# Apply bandreject filter
bandreject_filtered = apply_frequency_filter(image,
    filter_type='bandreject', D0=(20, 40))

# Plotting the results
plt.figure(figsize=(10, 8))

plt.subplot(2, 3, 1)
plt.imshow(image, cmap='gray')
plt.title('Original Image')

plt.subplot(2, 3, 2)
plt.imshow(lowpass_filtered, cmap='gray')
plt.title('Lowpass Filtered')

plt.subplot(2, 3, 3)
plt.imshow(highpass_filtered, cmap='gray')
plt.title('Highpass Filtered')

plt.subplot(2, 3, 4)
plt.imshow(bandpass_filtered, cmap='gray')
plt.title('Bandpass Filtered')

plt.subplot(2, 3, 5)
plt.imshow(bandreject_filtered, cmap='gray')
plt.title('Bandreject Filtered')
plt.show()
```

Output:

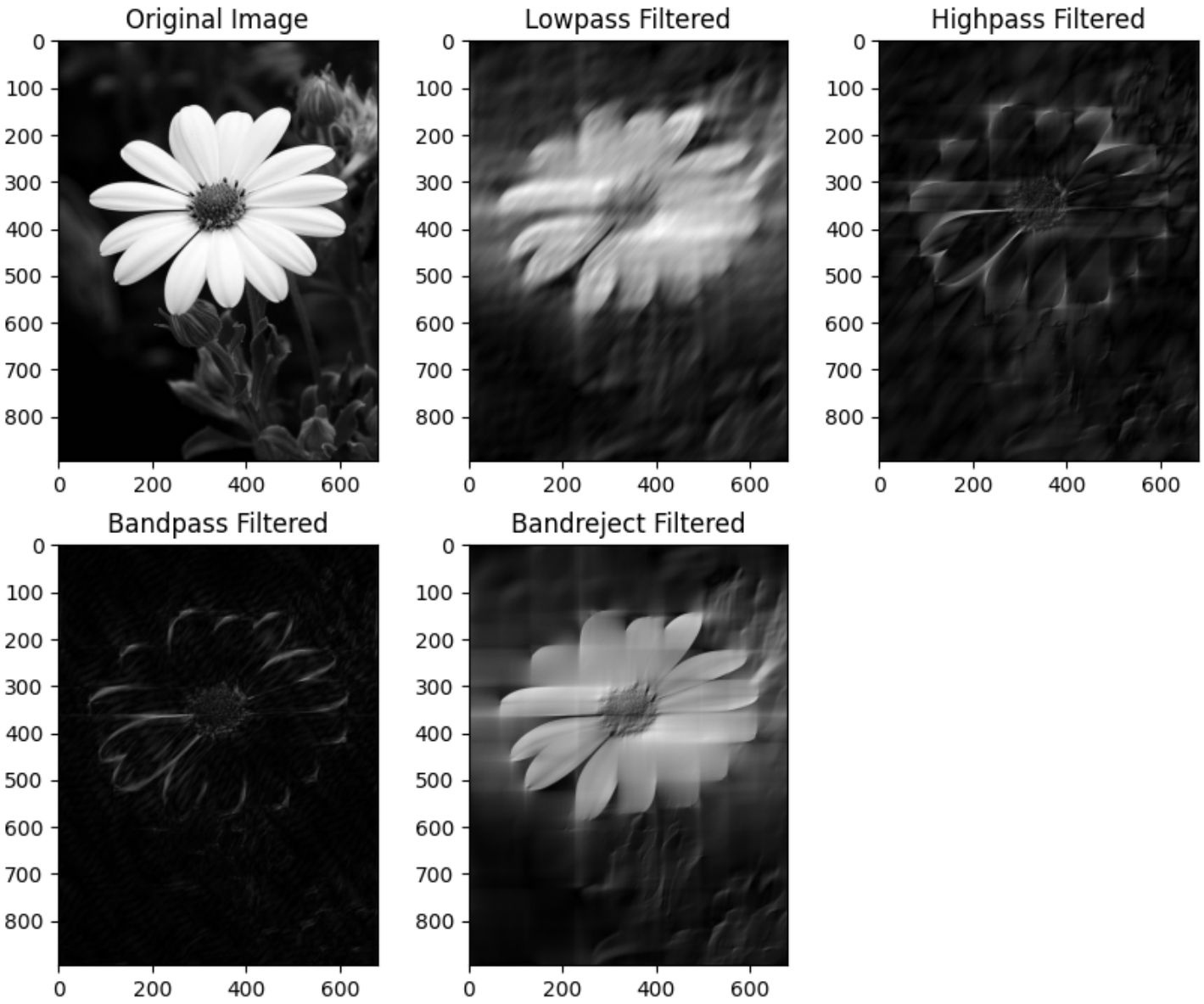

Figure 3.1: Original Image and Ideal Filtered Image (Lowpass, Highpass, Bandpass, Bandreject Filtering).

3.2.2 Butterworth Image Filtering

Code :

```
#Butterworth
import cv2
```

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

def butterworth_filter(shape, D0, n, filter_type='lowpass'):
    rows, cols = shape
    center_row, center_col = rows // 2, cols // 2

    # Create a meshgrid of distances from the center
    x = np.arange(cols) - center_col
    y = np.arange(rows) - center_row
    x, y = np.meshgrid(x, y)

    # Compute the distance matrix
    distance = np.sqrt(x**2 + y**2)

    # Create the filter based on the filter type
    if filter_type == 'lowpass':
        H = 1 / (1 + (distance / D0)**(2 * n))
    elif filter_type == 'highpass':
        H = 1 - 1 / (1 + (distance / D0)**(2 * n))
    elif filter_type == 'bandpass':
        H = 1 / (1 + ((distance**2 - D0[0]**2) / (distance * (
  D0[1]**2 - D0[0]**2)))**(2 * n))
    elif filter_type == 'bandreject':
        H = 1 - 1 / (1 + ((distance**2 - D0[0]**2) / (distance
    * (D0[1]**2 - D0[0]**2)))**(2 * n))
    else:
        raise ValueError("Invalid filter type")
    return H

```

```
def apply_butterworth_filter(image, filter_type='lowpass', D0=30, n=2):
    # Convert the image to grayscale
    if len(image.shape) == 3:
        image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Apply the Fourier transform
    f_transform = np.fft.fft2(image)
    f_shift = np.fft.fftshift(f_transform)

    # Create the frequency filter
    rows, cols = image.shape
    H = butterworth_filter(image.shape, D0, n, filter_type)

    # Apply the filter in the frequency domain
    f_filtered = f_shift * H

    # Inverse Fourier transform to get the image back to spatial domain
    img_filtered = np.abs(np.fft.ifft2(np.fft.ifftshift(f_filtered)))

    return img_filtered

# Read an example image
image = cv2.imread('/content/flw.jpg', cv2.IMREAD_GRAYSCALE)

# Define the cutoff frequency and filter order
cutoff_frequency = 30
filter_order = 2
```

```
58
59 # Apply Butterworth lowpass filter
60 butterworth_lowpass_filtered = apply_butterworth_filter(image,
       filter_type='lowpass', D0=cutoff_frequency, n=filter_order
   )
61 # Apply Butterworth highpass filter
62 butterworth_highpass_filtered = apply_butterworth_filter(image
   , filter_type='highpass', D0=cutoff_frequency, n=
   filter_order)
63 # Apply Butterworth bandpass filter
64 butterworth_bandpass_filtered = apply_butterworth_filter(image
   , filter_type='bandpass', D0=(20, 40), n=filter_order)
65 # Apply Butterworth bandreject filter
66 butterworth_bandreject_filtered = apply_butterworth_filter(
   image, filter_type='bandreject', D0=(40, 45), n=
   filter_order)
67
68 # Plotting the results
69 plt.figure(figsize=(10, 8))
70 plt.subplot(2, 3, 1)
71 plt.imshow(image, cmap='gray')
72 plt.title('Original Image')
73 plt.subplot(2, 3, 2)
74 plt.imshow(butterworth_lowpass_filtered, cmap='gray')
75 plt.title('Butterworth Lowpass Filtered')
76 plt.subplot(2, 3, 3)
77 plt.imshow(butterworth_highpass_filtered, cmap='gray')
78 plt.title('Butterworth Highpass Filtered')
79 plt.subplot(2, 3, 4)
80 plt.imshow(butterworth_bandpass_filtered, cmap='gray')
```

```
plt.title('Butterworth Bandpass Filtered')
plt.subplot(2, 3, 5)
plt.imshow(butterworth_bandreject_filtered, cmap='gray')
plt.title('Butterworth Bandreject Filtered')
plt.show()
```

Output:

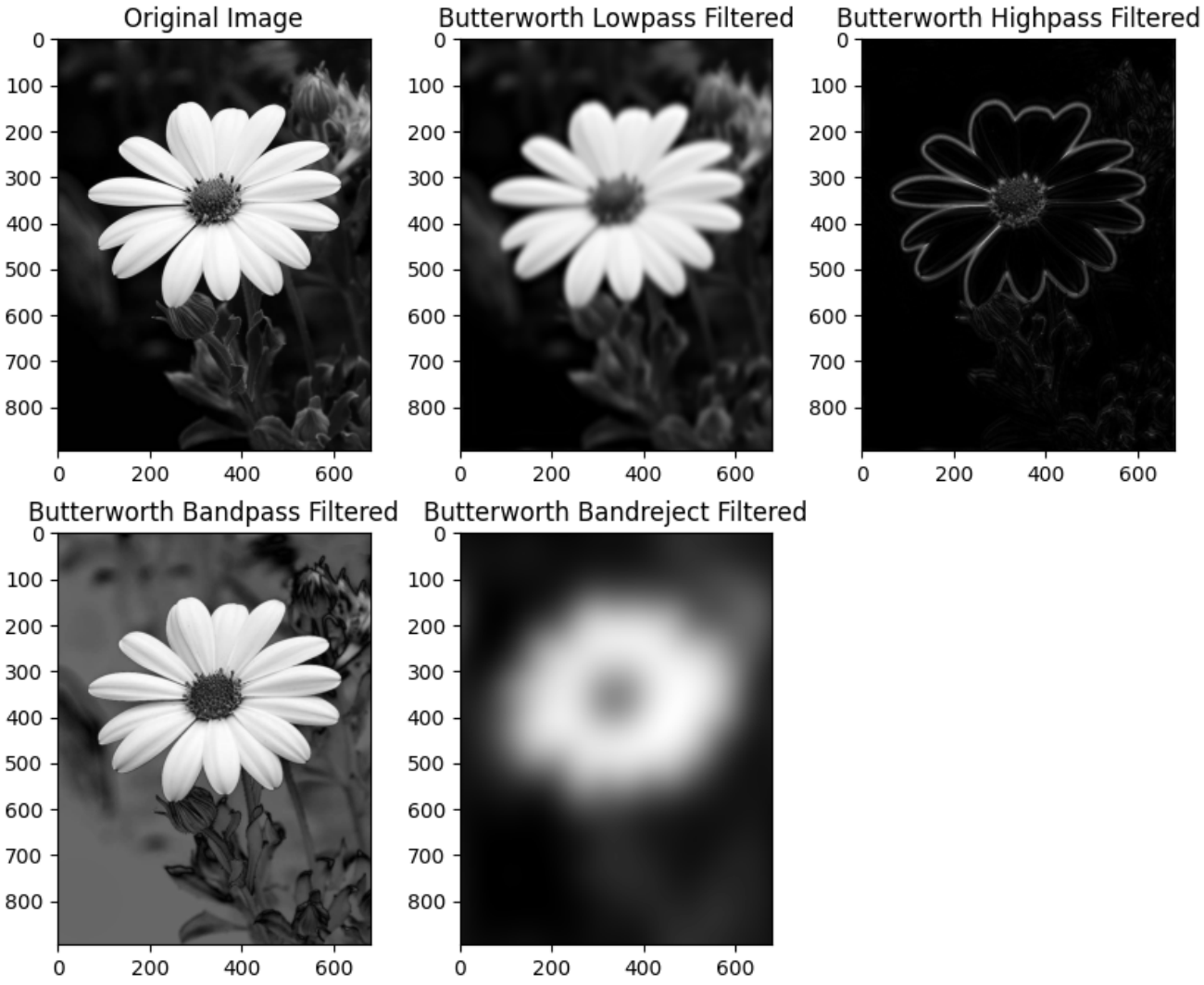

Figure 3.2: Original Image and Butterworth Filtered Image (Lowpass, Highpass, Bandpass, Band-reject Filtering).

3.2.3 Gaussian Image Filtering

Code :

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

def gaussian_filter(shape, D0, filter_type='lowpass'):
    rows, cols = shape
    center_row, center_col = rows // 2, cols // 2

    # Create a meshgrid of distances from the center
    x = np.arange(cols) - center_col
    y = np.arange(rows) - center_row
    x, y = np.meshgrid(x, y)
    # Compute the distance matrix
    distance = np.sqrt(x**2 + y**2)

    # Create the filter based on the filter type
    if filter_type == 'lowpass':
        H = np.exp(-(distance**2) / (2 * (D0**2)))
    elif filter_type == 'highpass':
        H = 1 - np.exp(-(distance**2) / (2 * (D0**2)))
    elif filter_type == 'bandpass':
        H = 1 - np.exp(-(distance**2) / (2 * (D0[1]**2))) - (
   np.exp(-(distance**2) / (2 * (D0[0]**2))))
     elif filter_type == 'bandreject':
        H = np.exp(-(distance**2) / (2 * (D0[1]**2))) + (1 -
   np.exp(-(distance**2) / (2 * (D0[0]**2))))
```

```
26      else:
27          raise ValueError("Invalid filter type")
28
29      return H
30
31  def apply_gaussian_filter(image, filter_type='lowpass', D0=30)
    :
32      # Convert the image to grayscale
33      if len(image.shape) == 3:
34          image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
35
36      # Apply the Fourier transform
37      f_transform = np.fft.fft2(image)
38      f_shift = np.fft.fftshift(f_transform)
39
40      # Create the frequency filter
41      rows, cols = image.shape
42      H = gaussian_filter(image.shape, D0, filter_type)
43
44      # Apply the filter in the frequency domain
45      f_filtered = f_shift * H
46
47      # Inverse Fourier transform to get the image back to
    spatial domain
48      img_filtered = np.abs(np.fft.ifft2(np.fft.ifftshift(
    f_filtered)))
49
50      return img_filtered
51
52  # Read an example image
```

```
image = cv2.imread('/content/flw.jpg', cv2.IMREAD_GRAYSCALE)

# Define the cutoff frequency
cutoff_frequency = 30

# Apply Gaussian lowpass filter
gaussian_lowpass_filtered = apply_gaussian_filter(image,
    filter_type='lowpass', D0=cutoff_frequency)
# Apply Gaussian highpass filter
gaussian_highpass_filtered = apply_gaussian_filter(image,
    filter_type='highpass', D0=cutoff_frequency)
# Apply Gaussian bandpass filter
gaussian_bandpass_filtered = apply_gaussian_filter(image,
    filter_type='bandpass', D0=(20, 40))
# Apply Gaussian bandreject filter
gaussian_bandreject_filtered = apply_gaussian_filter(image,
    filter_type='bandreject', D0=(20, 40))

# Plotting the results
plt.figure(figsize=(10, 8))
plt.subplot(2, 3, 1)
plt.imshow(image, cmap='gray')
plt.title('Original Image')
plt.subplot(2, 3, 2)
plt.imshow(gaussian_lowpass_filtered, cmap='gray')
plt.title('Gaussian Lowpass Filtered')
plt.subplot(2, 3, 3)
plt.imshow(gaussian_highpass_filtered, cmap='gray')
plt.title('Gaussian Highpass Filtered')
plt.subplot(2, 3, 4)
```

```
plt.imshow(gaussian_bandpass_filtered, cmap='gray')
plt.title('Gaussian Bandpass Filtered')
plt.subplot(2, 3, 5)
plt.imshow(gaussian_bandreject_filtered, cmap='gray')
plt.title('Gaussian Bandreject Filtered')
plt.show()
```

Output:

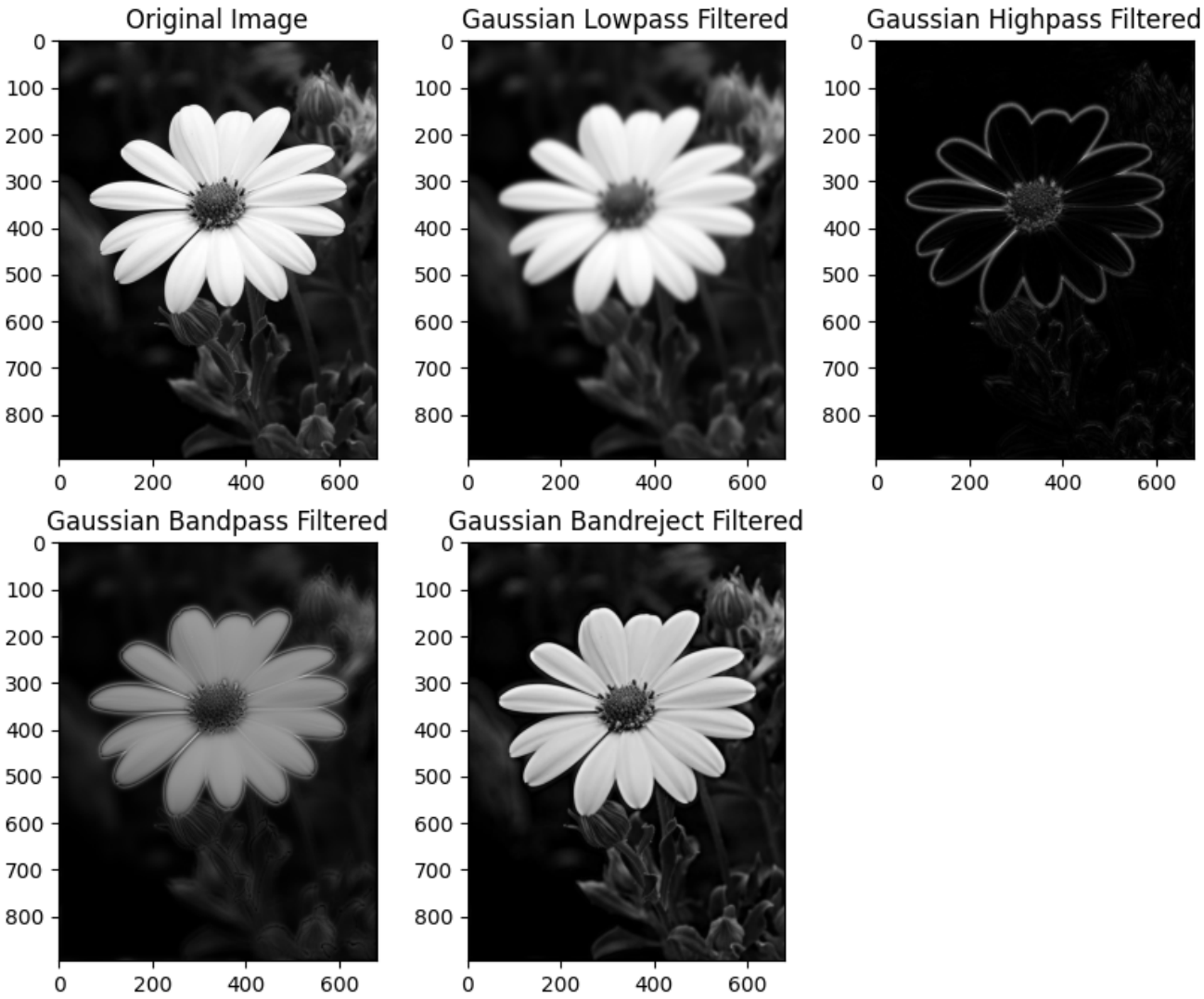

Figure 3.3: Original Image and Gaussian Filtered Image (Lowpass, Highpass, Bandpass, Bandreject Filtering).

3.3 Conclusion

In conclusion, the tasks of Histogram Equalization, Image Intensity Transformations, and Image Filtering in the Spatial Domain were systematically conducted using Python programming in the Google Colab environment. Images were employed as inputs, and the OpenCV library operations were performed. Crucial functions, such as ‘imread‘, ‘imshow‘, ‘warpAffine‘, ‘equalizeHist‘, ‘cvtColor‘, ‘calcHist‘, ‘filter2D‘, ‘medianBlur‘, ‘GaussianBlur‘, and various edge detection operators like ‘Sobel‘, ‘Laplacian‘, and ‘Scharr‘ from OpenCV were applied.

The OpenCV library facilitated seamless implementation, allowing for the exploration of diverse image-processing techniques. Numpy and Matplotlib complemented these operations, enhancing data manipulation and visualization. Throughout the experiments, the significance of each technique in image enhancement and analysis became evident. Utilizing these libraries in a collaborative platform like Google Colab offered a streamlined environment for comprehensive exploration. Overall, this study delved into the intricacies of image processing, providing valuable insights into the capabilities of Python and associated libraries for effective spatial domain operations.

Heaven's Light is Our Guide

Rajshahi University of Engineering & Technology

Department of Electronics & Telecommunication Engineering

ETE 4226: Sessional Based on ETE 4225

Experiment 4

Study of Line & Edge Detection and Image Segmentation.

Submitted by:

Md. Darul Atfal Palash

Roll: 1804005

Session: 2018-19

Submitted to:

Dr. Shah Ariful Hoque Chowdhury

Associate Professor

Dept. of ETE, RUET

Date of Experiment : 26/11/2023

Date of Submission : 25/01/2024

Report	**(Teacher's Section)**	**Viva**
☐ Excellent		☐ Excellent
☐ Very Good		☐ Very Good
☐ Good	_____	☐ Good
☐ Moderate	Signature	☐ Moderate
☐ Poor		☐ Poor

Experiment 4

Study of Line & Edge Detection and Image Segmentation.

Objectives

The primary objectives of this experiment are as follows::

- Understand the theoretical foundations of image segmentation.
- Implement and analyze Hough Transform for line and edge detection.
- Explore K-means clustering for image segmentation.
- Investigate Superpixel-based clustering methods.

4.1 Required Apparatus/Softwares

– Python.

– A Highly Configured PC.

4.2 Program Code and Output

4.2.1 Line and Edge Detection Using Hough Transformation

Code :

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

def detect_lines(image):
    # Convert the image to grayscale
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    # Apply Gaussian blur to reduce noise and help with line
   detection
    blurred = cv2.GaussianBlur(gray, (5, 5), 0)
    # Use the Canny edge detector
    edges = cv2.Canny(blurred, 50, 150)
```

```
    # Use the Hough Line Transform to detect lines
    lines = cv2.HoughLines(edges, 1, np.pi / 180, threshold=100)
    # Draw the lines on a copy of the original image
    lines_image = image.copy()
    if lines is not None:
        for line in lines:
            rho, theta = line[0]
            a = np.cos(theta)
            b = np.sin(theta)
            x0 = a * rho
            y0 = b * rho
            x1 = int(x0 + 1000 * (-b))
            y1 = int(y0 + 1000 * (a))
            x2 = int(x0 - 1000 * (-b))
            y2 = int(y0 - 1000 * (a))
            cv2.line(lines_image, (x1, y1), (x2, y2), (0, 0, 255), 2)
    return edges, lines_image
# Example usage:
image = cv2.imread('/content/logo.png')
# Perform line detection
edges, lines_image = detect_lines(image)
# Display the original, Canny edges, and lines-detected images
plt.figure(figsize=(15, 5))
plt.subplot(1, 3, 1), plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.title('Original Image'), plt.xticks([]), plt.yticks([])
plt.subplot(1, 3, 2), plt.imshow(edges, cmap='gray')
plt.title('Canny Edges'), plt.xticks([]), plt.yticks([])
```

```
plt.subplot(1, 3, 3), plt.imshow(cv2.cvtColor(lines_image, cv2
    .COLOR_BGR2RGB))
plt.title('Lines Detected'), plt.xticks([]), plt.yticks([])
plt.show()
```

Output:

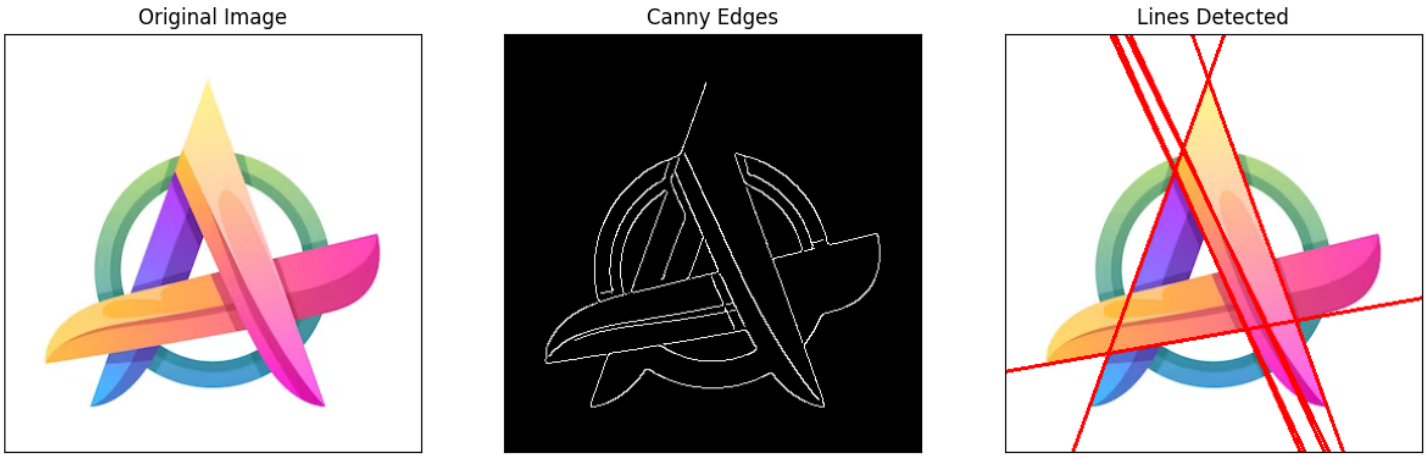

Figure 4.1: Original Image and Edges & Lines of the Image.

4.2.2 Image Segmentation using K-means Clustering

Code :

```
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image

def initialize_centroids(data, k):
    # Randomly initialize centroids
    indices = np.random.choice(len(data), k, replace=False)
    centroids = data[indices]
    return centroids

def assign_to_clusters(data, centroids):
```

```
    # Assign each data point to the nearest centroid
    distances = np.linalg.norm(data[:, np.newaxis, :] -
centroids, axis=2)
    clusters = np.argmin(distances, axis=1)
    return clusters

def update_centroids(data, clusters, k):
    # Update centroids based on the mean of the assigned data
points
    centroids = np.array([np.mean(data[clusters == i], axis=0)
    for i in range(k)])
    return centroids

def kmeans(data, k, max_iterations=100):
    centroids = initialize_centroids(data, k)
    for _ in range(max_iterations):
        clusters = assign_to_clusters(data, centroids)
        new_centroids = update_centroids(data, clusters, k)
        # Check for convergence
        if np.all(centroids == new_centroids):
            break
        centroids = new_centroids
    return centroids, clusters

def display_segmentation(image, clusters, k):
    # Create a mask based on the cluster labels
    mask = (clusters * 255 / (k - 1)).astype(np.uint8)

    # Display the original image and the mask
    plt.subplot(1, 2, 1)
```

```
    plt.imshow(image)
    plt.title('Original Image')
    plt.subplot(1, 2, 2)
    plt.imshow(mask, cmap='gray')
    plt.title('Segmentation Mask (K = 7)')

    plt.show()

def main():
    # Load image
    image_path = "/content/ishihara_5_h.jpg"
    image = np.array(Image.open(image_path))
    # Flatten the image to use RGB values as features
    data = image.reshape((-1, 3))

    # Number of clusters
    k = 7
    # Run k-means clustering
    centroids, clusters = kmeans(data, k)

    # Reshape the cluster assignments to the original image
    shape
    clusters = clusters.reshape(image.shape[:2])
    # Display the segmentation results
    display_segmentation(image, clusters, k)
if __name__ == "__main__":
    main()
```

Output:

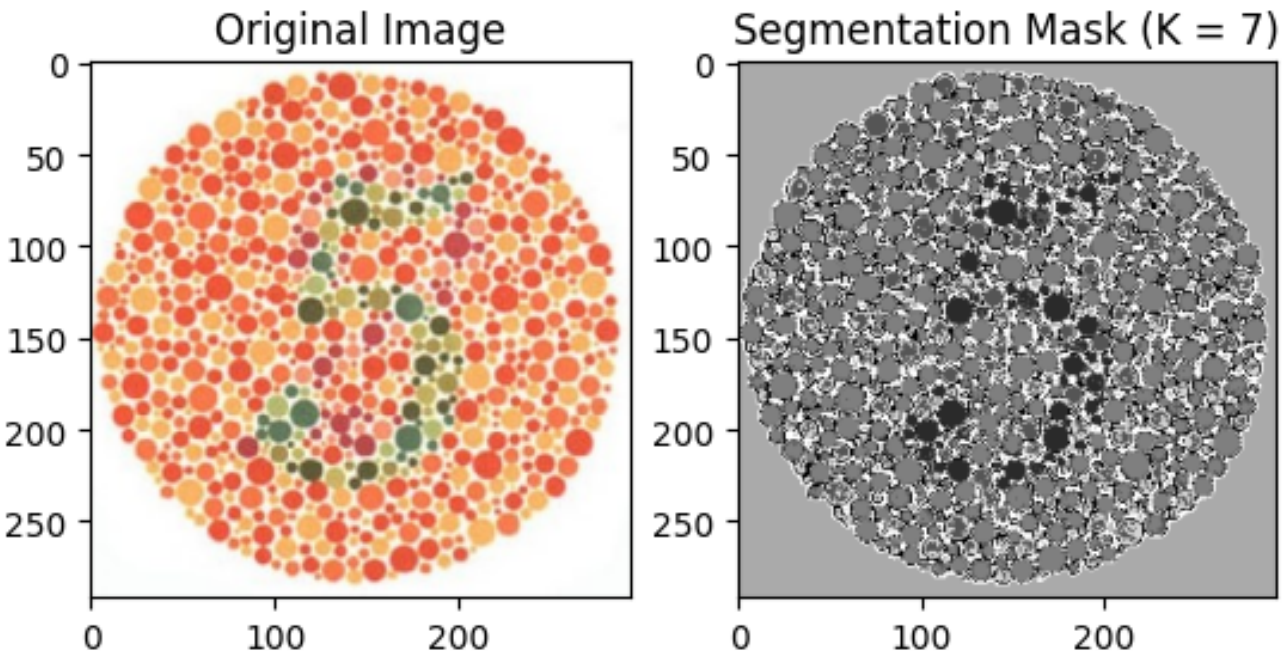

Figure 4.2: Original Image and K-Means Clustered Image.

4.2.3 Image Segmentation Using Superpixels Based Clustering

Code :

```
import numpy as np
from skimage import color, segmentation
from skimage import io
import matplotlib.pyplot as plt

def slic(image, num_segments, compactness):
    # Convert the image to Lab color space
    image_lab = color.rgb2lab(image)

    # Initialize cluster centers
    h, w, _ = image.shape
    step = int(np.sqrt((h * w) / num_segments))
```

```
    grid_y, grid_x = np.meshgrid(np.arange(0, h, step), np.arange(0, w, step))
    centers = np.vstack((grid_x.flatten(), grid_y.flatten())).T

    # Iterate until convergence
    max_iter = 10
    for _ in range(max_iter):
        # Assign pixels to clusters
        labels = segmentation.slic(image_lab, n_segments=num_segments, compactness=compactness)

        # Update cluster centers
        for i in range(num_segments):
            mask = (labels == i)
            centers[i] = np.mean(np.nonzero(mask), axis=1)

    return labels

# Example usage:
image = io.imread('/content/download.jpg')
num_segments = 5
compactness = 20
segmented_image = slic(image, num_segments, compactness)

# Display the original and segmented images
fig, ax = plt.subplots(1, 2, figsize=(12, 6))
ax[0].imshow(image)
ax[0].set_title('Original Image')
ax[1].imshow(segmented_image, cmap='nipy_spectral')
```

```
40 ax[1].set_title('Segmented Image')
41 plt.show()
```

Output:

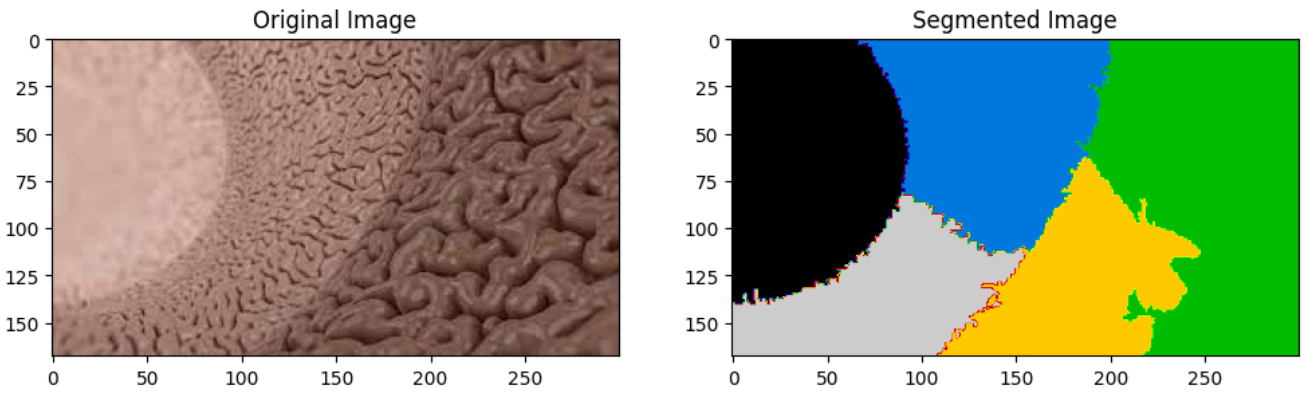

Figure 4.3: Original Image and Superpixel Based Clustered Image.

4.3 Conclusion

In this study of image segmentation conducted on the Google Colab platform using Python, a series of image processing operations were performed, encompassing line and edge detection through Hough Transformation, and image segmentation using K-means clustering as well as superpixels-based clustering. The openCV library, along with skimage, numpy, and matplotlib, played pivotal roles in the implementation of these tasks. HoughLines, slic, range, rgb2lab, vstack, mean, reshape, BGR2RGB, GaussianBlur, Canny, imread, imshow, and various other functions from these libraries were strategically employed.

The Hough Transformation facilitated the extraction of lines and edges from images, providing insights into their structural features. K-means clustering enabled efficient segmentation by grouping similar pixels, offering a means to understand and isolate distinct regions within an image. Superpixels-based clustering, specifically using the slic function, further improved segmentation precision. The employed functions collectively contributed to the analytical processes, fostering a comprehensive exploration of image segmentation techniques. This study highlights the significance of these approaches, showcasing their versatility and applicability in image analysis and computer vision applications.

Heaven's Light is Our Guide

Rajshahi University of Engineering & Technology

Department of Electronics & Telecommunication Engineering

ETE 4226: Sessional Based on ETE 4225

Experiment 5

Study of Feature Extraction and Image Pattern Classification.

Submitted by:

Md. Darul Atfal Palash

Roll: 1804005

Session: 2018-19

Submitted to:

Dr. Shah Ariful Hoque Chowdhury

Associate Professor

Dept. of ETE, RUET

Date of Experiment : 17/12/2023

Date of Submission : 25/01/2024

Report	**(Teacher's Section)**	**Viva**
☐ Excellent		☐ Excellent
☐ Very Good		☐ Very Good
☐ Good	_____	☐ Good
☐ Moderate	Signature	☐ Moderate
☐ Poor		☐ Poor

Experiment 5

Study of Feature Extraction and Image Pattern Classification.

Objectives

The primary objectives of this experiment are as follows:

- Understand the theoretical foundations of image segmentation.
- Implement and analyze Hough Transform for line and edge detection.
- Explore K-means clustering for image segmentation.
- Investigate Superpixel-based clustering methods.

5.1 Required Apparatus/Softwares

- Python.
- A Highly Configured PC.

5.2 Program Code and Output

5.2.1 Harris-Stephens (HS) Corner Detection

Code :

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
image = cv2.imread('/content/chess.jpg')
gray_image = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
float_image = np.float32(gray_image)

threshold=0.15
new_image = cv2.cornerHarris(gray_image,20,3,0.04)
new_image =cv2.dilate(new_image,None)
image[new_image > threshold * new_image.max()] = [255,0,0]

```

```
# Display the images
image2= cv2.imread('/content/chess.jpg')
plt.subplot(1, 2, 1)
plt.imshow(image2)
plt.title('Original Image')
plt.subplot(1, 2, 2)
plt.imshow(image)
plt.title('Harris Corner Detection')
plt.show()
```

Output:

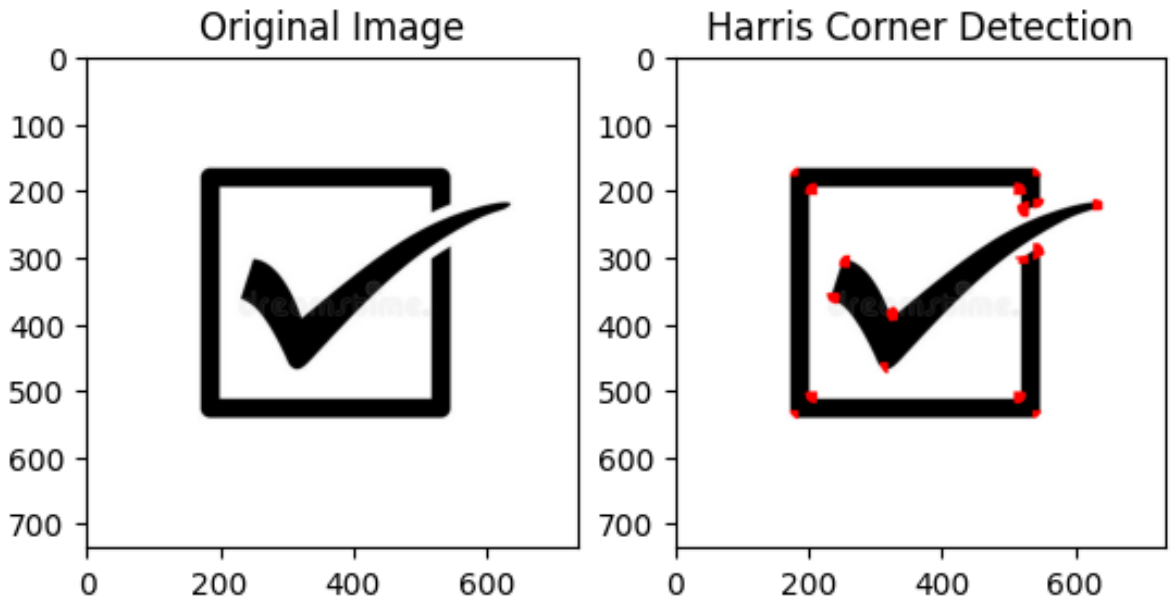

Figure 5.1: Original Image and Corner Detected Image Using Harris-Stephens (HS) Corner Detection.

5.2.2 Implement and Train a Fully Connected Neural Network

Code :

```
import numpy as np
import keras
from keras.datasets import mnist
```

```
from keras.models import Sequential
from keras.layers import Dense, Flatten
import matplotlib.pyplot as plt

# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()

batch_size = 128
num_classes = 10
epochs = 5

x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)
x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)
input_shape = (28, 28, 1)

y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255

model = Sequential()
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))

model.compile(
    optimizer='adam',
```

```
34      loss='categorical_crossentropy',
35      metrics=['accuracy']
36  )
37
38  history = model.fit(x_train, y_train, epochs=epochs,
        validation_data=(x_test, y_test))
39
40  score = model.evaluate(x_test, y_test, verbose=0)
41  print("Test loss:", score[0])
42  print("Test accuracy:", score[1])
43
44  y_pred = model.predict(x_test)
45  y_pred_classes = np.argmax(y_pred, axis=1)
46
47  plt.figure(figsize=(8, 5.5))
48  for i in range(6):
49      plt.subplot(2, 3, i + 1)
50      plt.imshow(x_test[i].reshape(28, 28), cmap='gray')
51      plt.title(f"True: {np.argmax(y_test[i])}, Predicted: {
        y_pred_classes[i]}")
52      plt.axis('off')
53  plt.show()
54  plt.figure(figsize=(8, 4))
55  # Accuracy plot
56  plt.subplot(1, 2, 1)
57  plt.plot(history.history['accuracy'], label='Training Accuracy
        ')
58  plt.plot(history.history['val_accuracy'], label='Validation
        Accuracy')
59  plt.title('Accuracy vs. Epoch')
```

```
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(False)
# Loss plot
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Loss vs. Epoch')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.grid(False)
plt.tight_layout()
plt.show()
```

Output:

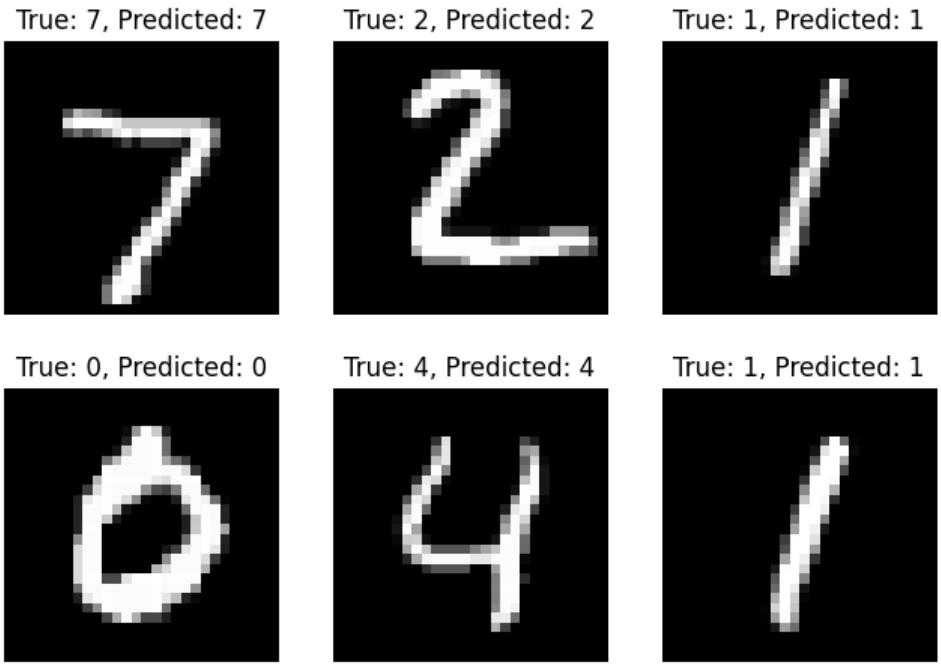

Figure 5.2: Handwritten Digits with Its True and Predicted Value.

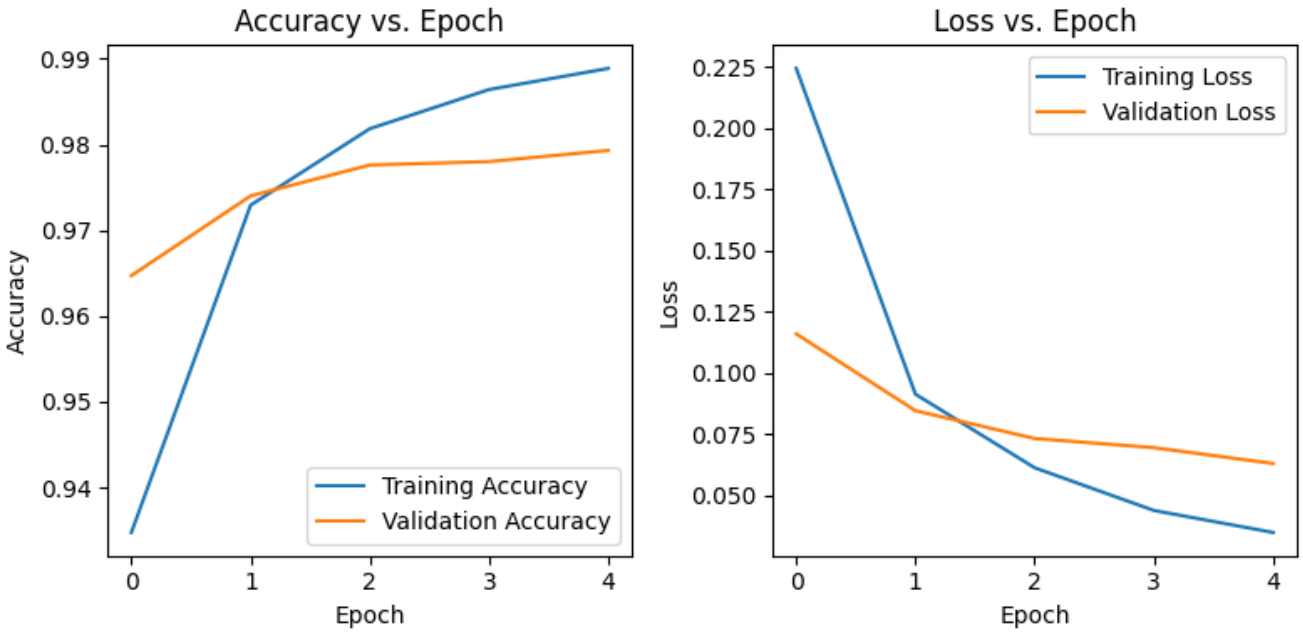

Figure 5.3: Accuracy and Loss Plot.

5.2.3 Implement and Train a Convolution Neural Network

Code :

```
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K
import matplotlib.pyplot as plt
import numpy as np
(x_train, y_train), (x_test, y_test) = mnist.load_data()
batch_size = 128
num_classes = 10
epochs = 5 # Changed the number of epochs for faster training
x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)
x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)
```

```
input_shape = (28, 28, 1)
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu',
   input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))
model.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
    )

history = model.fit(x_train, y_train, epochs=epochs,
   validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)

print("Test loss:", score[0])
print("Test accuracy:", score[1])
y_pred = model.predict(x_test)
y_pred_classes = np.argmax(y_pred, axis=1)

```

```
plt.figure(figsize=(8, 5.5))
for i in range(6):
  plt.subplot(2, 3, i + 1)
  plt.imshow(x_test[i].reshape(28, 28), cmap='gray')
  plt.title(f"True: {np.argmax(y_test[i])}, Predicted: {
   y_pred_classes[i]}")
  plt.axis('off')
plt.show()
plt.figure(figsize=(8, 4))
# Accuracy plot
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy
   ')
plt.plot(history.history['val_accuracy'], label='Validation
   Accuracy')
plt.title('Accuracy vs. Epoch')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(False)
# Loss plot
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Loss vs. Epoch')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.grid(False)
plt.tight_layout()
```

```
plt.show()
```

Output:

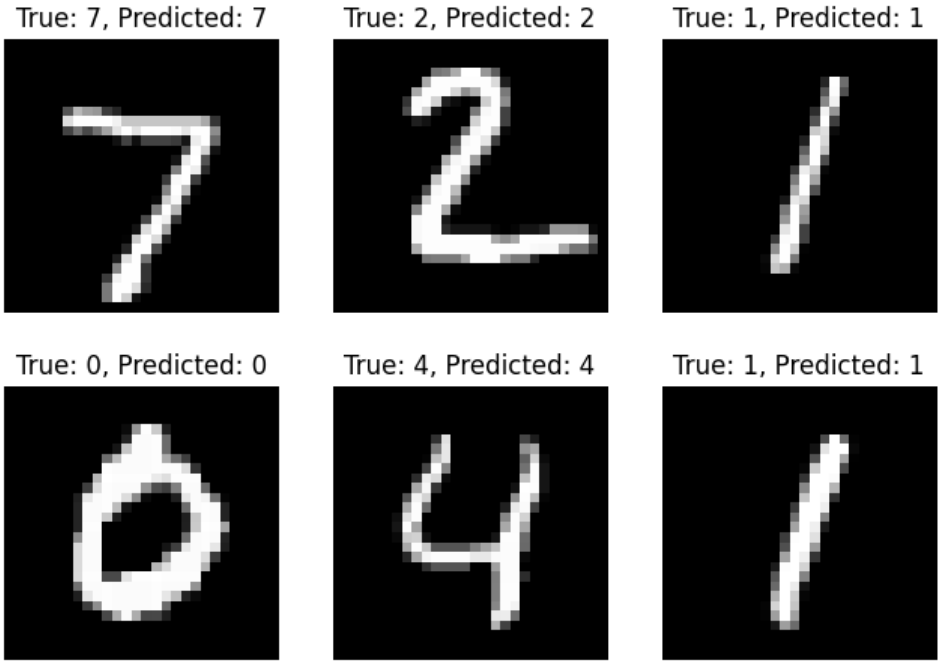

Figure 5.4: Handwritten Digits with Its True and Predicted Value.

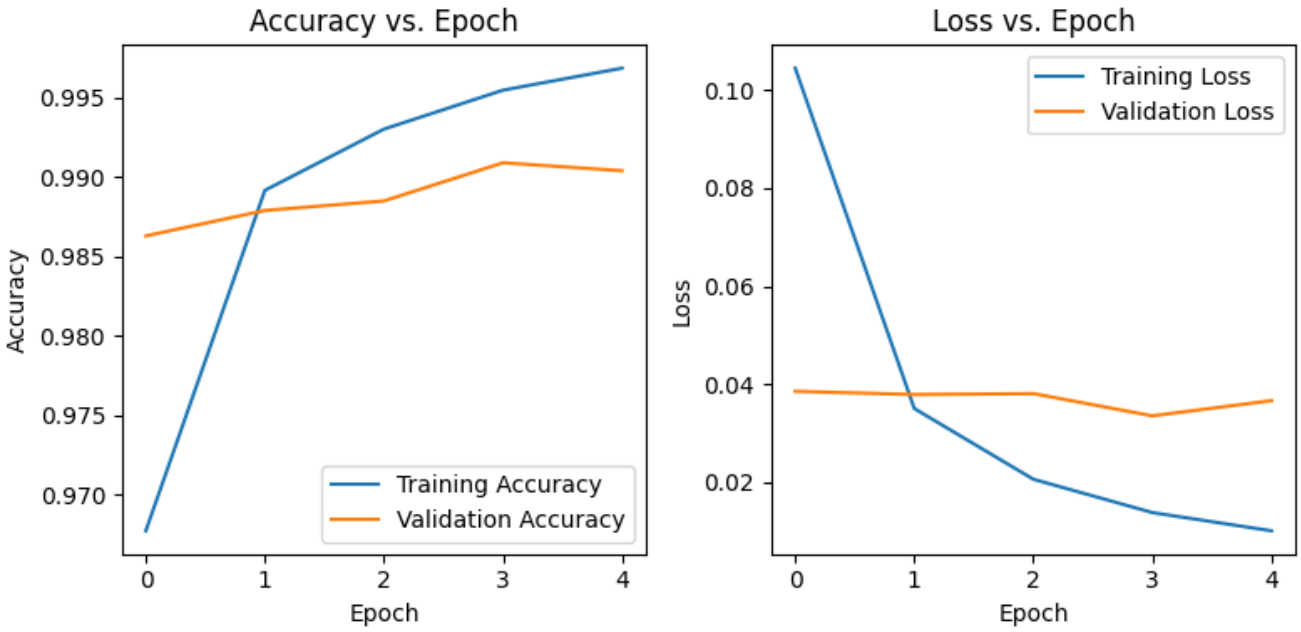

Figure 5.5: Accuracy and Loss Plot.

5.3 Conclusion

In this study, the exploration of feature extraction and image pattern classification was conducted on the Google Colab platform using Python. For corner detection, the Harris-Stephens Corner Detection algorithm was employed, utilizing OpenCV functions such as '**cornerHarris**' and '**dilate**' to identify key points and enhance corner visibility in the given image. The application of corner detection serves as a fundamental step in understanding the structure and salient features of the image.

Moving toward image pattern classification, Fully Connected Neural Network (FCNN) and Convolutional Neural Network (CNN) architectures were implemented and trained for handwritten digit recognition, leveraging the capabilities of the Keras library. Commonly used functions like '**Conv2D**', '**MaxPooling2D**', '**Flatten**', and '**Dense**' were integral in constructing and configuring the neural networks. The models were compiled with the '**adam**' optimizer and '**softmax**' activation function for accurate predictions.

The employed convolution and fully connected layers facilitated the learning of intricate patterns within the digit images. The integration of numpy and matplotlib supported data manipulation and visualization, enhancing the comprehension of feature extraction and classification processes. This study underscores the significance of these techniques in computer vision applications, showcasing their adaptability and effectiveness in analyzing and classifying visual data.

Assignment

Problem - 1: K-means Clustering without Using Built-in Functions.

Code :

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

def k_means_clustering_custom(image, k=3, max_iters=100):
    # Flatten the image to a 1D array of pixels
    pixels = image.reshape((-1, 3))
    # Randomly initialize k cluster centers
    centers = pixels[np.random.choice(pixels.shape[0], k,
replace=False)]

    for _ in range(max_iters):
        # Calculate distances from each point to each cluster
center
        distances = np.linalg.norm(pixels - centers[:, np.
newaxis], axis=2)
        # Assign each pixel to the cluster with the nearest
center
        labels = np.argmin(distances, axis=0)
        # Update cluster centers
        new_centers = np.array([pixels[labels == i].mean(axis
=0) for i in range(k)])
        # Check for convergence
        if np.all(np.abs(centers - new_centers) < 1e-4):
            break
```

```
        centers = new_centers

    # Map each pixel to its cluster center
    clustered_pixels = centers[labels]
    # Reshape the clustered pixels back to the original image
    shape
    clustered_image = clustered_pixels.reshape(image.shape)
    return clustered_image, centers, labels

# Read the image
image = cv2.imread('/content/pic_ryb_itten.jpg')
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)  # OpenCV reads
    images in BGR format

# Plot the original image
plt.figure(figsize=(8, 5))
plt.subplot(1, 2, 1)
plt.imshow(image)
plt.title('Original Image')
# Apply k-means clustering
k_clusters = 3
clustered_image, centers, labels = k_means_clustering_custom(
   image, k=k_clusters)
# Plot the clustered image
plt.subplot(1, 2, 2)
plt.imshow(clustered_image.astype(np.uint8))
plt.title(f'Clustered Image (k={k_clusters})')
# Display the plots
plt.tight_layout()
plt.show()
```

Output:

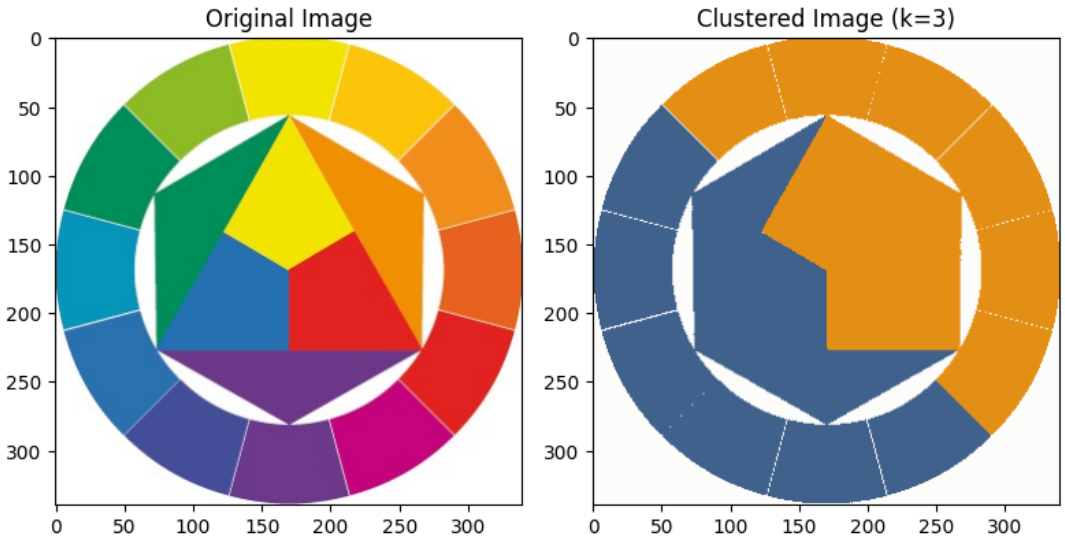

Figure 1.1: Original Image and K-Means Clustered Image.

Problem - 2: Image Segmentation Using Simple Linear Iterative Clustering (SLIC) Superpixel Algorithm (without Using Built-in Functions).

Code :

```
import numpy as np
from skimage import color, segmentation
from skimage import io
import matplotlib.pyplot as plt

def slic(image, num_segments, compactness):
    # Convert the image to Lab color space
    image_lab = color.rgb2lab(image)

    # Initialize cluster centers
    h, w, _ = image.shape
    step = int(np.sqrt((h * w) / num_segments))
    grid_y, grid_x = np.meshgrid(np.arange(0, h, step), np.
   arange(0, w, step))
```

```
    centers = np.vstack((grid_x.flatten(), grid_y.flatten())).T

    # Iterate until convergence
    max_iter = 10
    for _ in range(max_iter):
        # Assign pixels to clusters
        labels = segmentation.slic(image_lab, n_segments=num_segments, compactness=compactness)

        # Update cluster centers
        for i in range(num_segments):
            mask = (labels == i)
            centers[i] = np.mean(np.nonzero(mask), axis=1)

    return labels

# Example usage:
image = io.imread('/content/download.jpg')
num_segments = 5
compactness = 20
segmented_image = slic(image, num_segments, compactness)

# Display the original and segmented images
fig, ax = plt.subplots(1, 2, figsize=(12, 6))
ax[0].imshow(image)
ax[0].set_title('Original Image')
ax[1].imshow(segmented_image, cmap='nipy_spectral')
ax[1].set_title('Segmented Image')
plt.show()
```

Output:

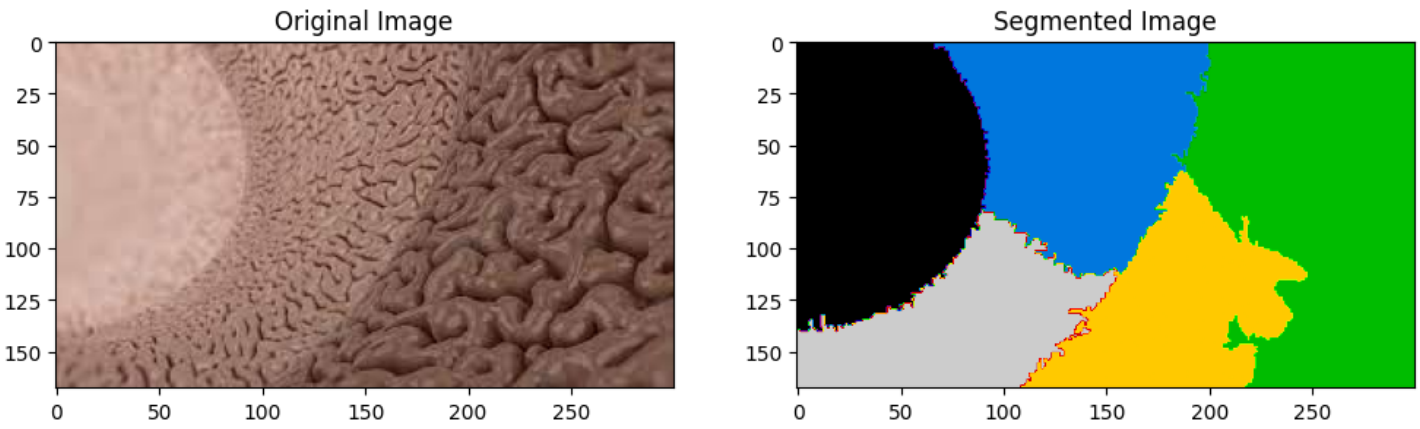

Figure 1.2: Original Image and SLIC Superpixel Clustered Image.

Problem - 3: Harris-Stephens (HS) Corner Detection without Using Built-in Functions.

Code :

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

def harris_corner_detection(image, k=0.04, threshold=0.005):
    # Convert the image to grayscale
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY).astype(np.
   float32)
    # Compute derivatives
    Ix = cv2.Sobel(gray, cv2.CV_64F, 1, 0, ksize=3)
    Iy = cv2.Sobel(gray, cv2.CV_64F, 0, 1, ksize=3)
    # Compute products of derivatives at each pixel
    Ixx = Ix * Ix
    Iyy = Iy * Iy
    Ixy = Ix * Iy
```

```
    # Compute sums of products of derivatives using a Gaussian filter
    window_size = 5
    kernel = np.ones((window_size, window_size), np.float32) / (window_size ** 2)
    Sxx = cv2.filter2D(Ixx, -1, kernel)
    Syy = cv2.filter2D(Iyy, -1, kernel)
    Sxy = cv2.filter2D(Ixy, -1, kernel)
    # Compute Harris corner response
    det_M = Sxx * Syy - Sxy**2
    trace_M = Sxx + Syy
    R = det_M - k * trace_M**2
    # Apply thresholding to detect corners
    corners = np.zeros_like(R)
    corners[R > threshold * R.max()] = 255
    return corners.astype(np.uint8)
# Read an example image
image = cv2.imread('/content/chess.jpeg')
# Perform Harris corner detection
corners_image = harris_corner_detection(image)
# Display the original and corner-detected images
plt.figure(figsize=(10, 6))
# Original Image
plt.subplot(1, 2, 1)
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.title('Original Image')
plt.axis('off')
# Corners Detected Image
color_corners_image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
```

```
color_corners_image[corners_image > 0] = [255, 0, 0]  # Mark
    corners in red
plt.subplot(1, 2, 2)
plt.imshow(color_corners_image)
plt.title('Corners Detected')
plt.axis('off')
plt.show()
```

Output:

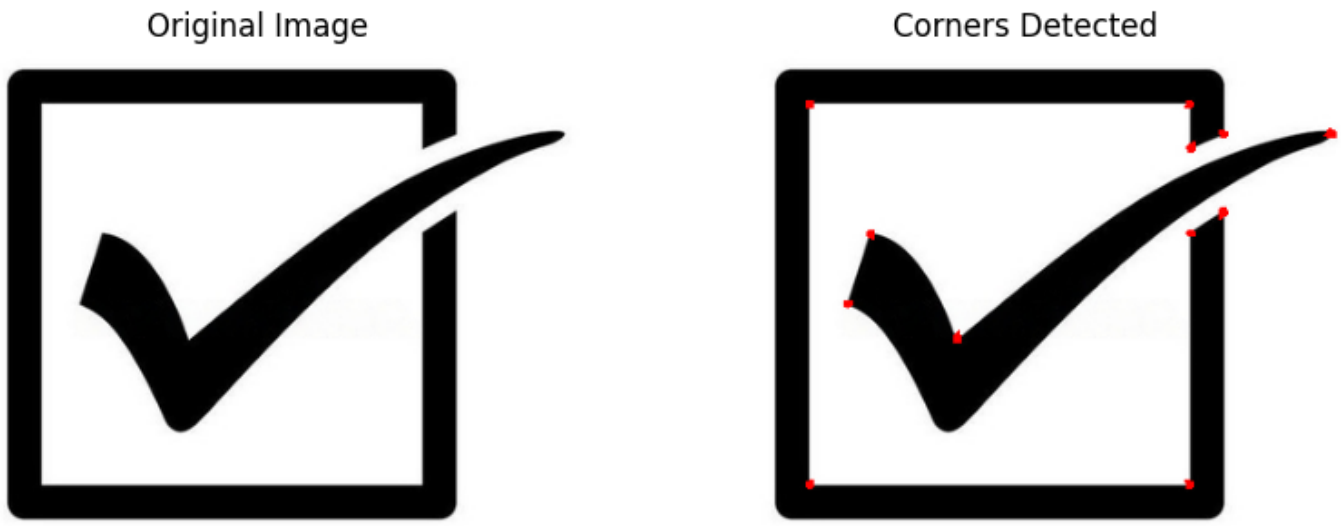

Figure 1.3: Original Image and HS Corner Detected Image.