Heaven's Light is Our Guide

Department of Electronics & Telecommunication Engineering

Rajshahi University of Engineering & Technology

Laboratory Report

on

ETE 4226: Sessional Based on Digital Image Processing

Submitted by:

Al Nahian Mugdho

Roll No. 1804021

Submitted to:

Dr. Shah Ariful Hoque Chowdhury

Associate Professor

Dept. of ETE, RUET

January, 2024

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 01

Study of Geometric Transformation of Images

Submitted by:

Al Nahian Mugdho

Roll: 1804021

Session: 2018-19

Submitted to:

Dr. Shah Ariful Hoque Chowdhury

Associate Professor

Dept. of ETE, RUET

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

Experiment 1

Study of Geometric Transformation of Images

1.1 Program Code

1.1.1 Translation, Rotation, Shearing and Scaling based on their matrices

```
### translation

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

# read the input image
loc = 'small2.jpg'
img = cv2.imread(loc)
# convert from BGR to RGB so we can plot using matplotlib
image_sized = cv2.resize(img,(500,500)) #image  # cv2.resize(
   image,(768,500))
img = image_sized
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
# disable x & y axis

# get the image shape
rows, cols, dim = img.shape
height,width = img.shape[:2]
# transformation matrix for translation
M = np.float32([[1, 0, 50],
                [0, 1, 50]])

# apply a perspective transformation to the image
```

```
translated_img = cv2.warpAffine(img, M, (cols, rows))
# transformation matrix for Shearing
M = np.float32([[1, 0.5, 0],
                [0, 1  , 0]])

# apply a perspective transformation to the image
sheared_img = cv2.warpAffine(img, M,(int(cols*1.5), int(rows*1.5)) ) #(int(cols*1.5), int(rows*1.5))

matrix_rotation  = cv2.getRotationMatrix2D((height/2,width/2),45,0.5)
rotation = cv2.warpAffine(img,matrix_rotation,(cols, rows))
matrix_scalling = np.float32([[1,0,0],[0,2,0]])
Scalling = cv2.warpAffine(img,matrix_scalling,(cols, rows))


plt.figure(figsize=(20,20 ))
plt.subplot(2, 4, 1)
plt.title("Original image",fontsize =20)
plt.axis(False)
plt.imshow(img)



plt.subplot(2, 4, 2)
plt.title("Translated Image ",fontsize =20)
plt.axis(False)
plt.imshow((translated_img))

```

```
52
53
54 plt.subplot(2, 4, 3)
55 plt.title("Vertical Shared image",fontsize =20)
56 plt.axis(False)
57 plt.imshow((sheared_img))
58
59
60 plt.subplot(2, 4, 4)
61 plt.title("45 degree Rotated image",fontsize =20)
62 plt.axis(False)
63 plt.imshow((rotation))
64
65 plt.subplot(2, 4,5)
66 plt.title("Scalled image",fontsize =20)
67 plt.axis(False)
68 plt.imshow((Scalling))
69
70
71
72
73 plt.show()
```

1.1.2 Result of Translation, Rotation, Shearing and Scaling

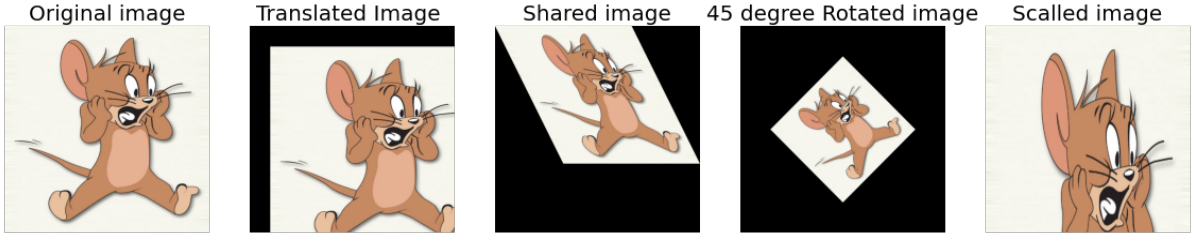

Figure 1.1: Translation, Rotation, Shearing and Scaling

1.1.3 Geometric Transformation with video

```
import cv2
import numpy
import numpy as np


image = cv2.imread('small2.jpg')


if image is None:
    print("Error: Could not open or find the image.")
else:
    #  dimensions of the image
    height, width, channels = image.shape

    print(f"Image Width: {width} pixels")
    print(f"Image Height: {height} pixels")
    print(f"Number of Channels (e.g., 3 for RGB): {channels}")



image_sized = cv2.resize(image,(500,500)) #image  # cv2.resize
   (image,(768,500))
image = image_sized

if image is None:
    print("Error: Could not open or find the image.")
else:
    #  dimensions of the image
```

```
    height, width, channels = image.shape

    print(f"Image Width: {width} pixels")
    print(f"Image Height: {height} pixels")
    print(f"Number of Channels (e.g., 3 for RGB): {channels}")
# Define the rotation angle in degrees
#angle_degrees = 5

# angle from degrees to radians
#x  = np.deg2rad(angle_degrees)


vw = width
vh = height
height,width = image.shape[:2]
source = cv2.VideoWriter_fourcc(*'XVID')

#duration_factor = 2  # Increase the duration by a factor of 2
#frame_rate = 30.0 / (10 * duration_factor)
#frame_rate = 0.30  # Set the desired frame rate
desired_duration =3
# Calculate the number of frames needed for a 20-second
   duration
num_frames = 50
frame_rate = num_frames / desired_duration
rotation_video = cv2.VideoWriter('rotation_video.avi', source,
    frame_rate, (vw,vh))
scalling_video = cv2.VideoWriter('scalling_video.avi', source,
    frame_rate, (vw,vh))
translation_video = cv2.VideoWriter('translation_video.avi',
```

```
    source, frame_rate, (vw,vh))
vertical_shear_video = cv2.VideoWriter('vertical_shear_video.
    avi', source, frame_rate, (vw,vh))
Horizontal_shear_video = cv2.VideoWriter('
    Horizontal_shear_video.avi', source, frame_rate, (vw,vh))
zoom_video = cv2.VideoWriter('zoom_video.avi', source,
    frame_rate, (vw,vh))
Shrinking_video = cv2.VideoWriter('Shrinking_video.avi',
    source, frame_rate, (vw,vh))



for a in range(0, 720,1):  # Rotation
    angle_degrees = a #* 360 / num_frames
    matrix_rotation  = cv2.getRotationMatrix2D((height/2,width
    /2),angle_degrees,0.5)
    rotation = cv2.warpAffine(image,matrix_rotation,(image.
    shape[1],image.shape[0]))
    rotation_video.write(rotation)

for s in np.arange(1, 300, 0.01): # scalling
    matrix_scalling = numpy.float32([[s,0,0],[0,s,0]])
    Scalling = cv2.warpAffine(image,matrix_scalling,(image.
    shape[1],image.shape[0]))
    scalling_video.write(Scalling)

for t in np.arange (0,300,1):#matrix_translation 300,1
    matrix_translation = numpy.float32([[1,0,t],[0,1,t]])
    translation = cv2.warpAffine(image,matrix_translation,(
    image.shape[1],image.shape[0]))
```

```
76      translation_video.write(translation)
77
78  for v in np.arange (0.0,100,0.01):#shearing vertical
79      matrix_vertical_shear = numpy.float32([[1,v,0],[0,1,0]])
80      vertical_shear= cv2.warpAffine(image,matrix_vertical_shear
        ,(image.shape[1],image.shape[0]))
81      vertical_shear_video.write(vertical_shear)
82
83  for h in np.arange (0.0,10.1,0.1):#shearing Horiz
84      matrix_Horizontal_shear = numpy.float32([[1,0,0],[h,1,0]])
85      Horizontal_shear= cv2.warpAffine(image,
        matrix_Horizontal_shear,(image.shape[1],image.shape[0]))
86      Horizontal_shear_video.write(Horizontal_shear)
87
88
89  #matrix_Identity = numpy.float32([[1,0,0],[0,1,0]])
90  #matrix_scalling = numpy.float32([[2,0,0],[0,2,0]])
91  #matrix_rotation = numpy.float32([[np.cos(x),-np.sin(x),0],[np
        .sin(x),np.cos(x),0]])
92  #matrix_translation = numpy.float32([[1,0,100],[0,1,100]])
93
94  for z in np.arange (0.0,10,0.01): #zooming
95
96      zoom_matrix = np.float32([[z, 0, 0],[0, z, 0]])
97      zoomed_image = cv2.warpAffine(image, zoom_matrix, (image.
        shape[1], image.shape[0]), flags=cv2.INTER_NEAREST)
98
99
100
101
```

```
102     zoom_video.write(zoomed_image)
103
104
105
106
107
108 for k in np.arange (5, 0, -0.01):# Shrinking (Scaling Down)
109
110     shrink_matrix = np.float32([[k, 0, 0],[0, k, 0]])
111     shrunk_image = cv2.warpAffine(image, shrink_matrix, (image
    .shape[1], image.shape[0]), flags=cv2.INTER_NEAREST)
112
113
114
115
116     Shrinking_video.write(shrunk_image)
117
118
119
120
121
122 rotation_video.release()
123 scalling_video.release()
124 translation_video.release()
125 vertical_shear_video.release()
126 Horizontal_shear_video.release()
127 zoom_video.release()
128 Shrinking_video.release()
129 # Load the individual video files
130 v1 = cv2.VideoCapture('rotation_video.avi')
```

```
v2 = cv2.VideoCapture('vertical_shear_video.avi')#cv2.
   VideoCapture('Horizontal_shear_video.avi')#cv2.VideoCapture
   ('vertical_shear_video.avi')
#Horizontal_shear_video2 = cv2.VideoCapture('
   Horizontal_shear_video.avi')
v3 = cv2.VideoCapture('scalling_video.avi')
v4 = cv2.VideoCapture('translation_video.avi')
#zoom_video2 = cv2.VideoCapture('zoom_video.avi')
#Shrinking_video2 = cv2.VideoCapture('Shrinking_video.avi')


# Check if the videos opened successfully
if not (v1.isOpened() and v2.isOpened() and v3.isOpened() and
   v4.isOpened()):
    print("One or more videos could not be opened.")
    exit()

# Get the video frame dimensions
frame_width = int(v1.get(3))
frame_height = int(v1.get(4))


# Create a video writer to save the output video
output_file = "output_video_final.avi" # avi mp4
fourcc = cv2.VideoWriter_fourcc(*'XVID')  # codec XVID mp4v
out = cv2.VideoWriter(output_file, fourcc, frame_rate, ( 2 *
   frame_width,2 * frame_height))

while True:
    ret1, frame1 = v1.read()
```

```
    ret2, frame2 = v2.read()
    ret3, frame3 = v3.read()
    ret4, frame4 = v4.read()

    if not (ret1 and ret2 and ret3 and ret4):
        break

    # Resize frames to the same dimensions
    frame1 = cv2.resize(frame1, (frame_width, frame_height))
    frame2 = cv2.resize(frame2, (frame_width, frame_height))
    frame3 = cv2.resize(frame3, (frame_width, frame_height))
    frame4 = cv2.resize(frame4, (frame_width, frame_height))


    # Create an empty frame with subplots
    empty_frame = np.zeros((2 * frame_height, 2 * frame_width,
    3), dtype=np.uint8)

    # Place each video frame in a subplot
    empty_frame[:frame_height, :frame_width] = frame1
    empty_frame[:frame_height, frame_width:] = frame2
    empty_frame[frame_height:, :frame_width] = frame3
    empty_frame[frame_height:, frame_width:] = frame4

    # Write the combined frame to the output video
    out.write(empty_frame)

# Release the video capture and writer objects
v1.release()
v2.release()
```

```
v3.release()
v4.release()
out.release()

print("Output video saved as:", output_file)

if(cv2.waitKey()==ord('q')):
    cv2.destroyAllWindows()
```

1.1.4 Result of Geometric Transformation

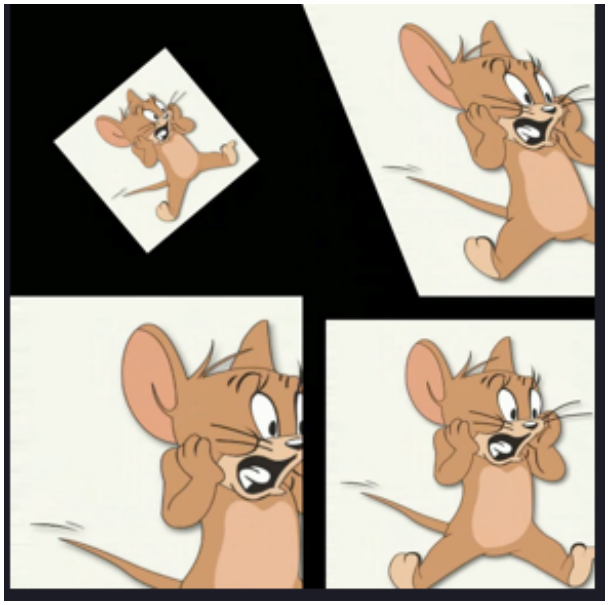

Figure 1.2: Shot from the output video of geometric transformation

Heaven's Light is Our Guide

Rajshahi University of Engineering & Technology

Department of Electronics & Telecommunication Engineering

ETE 4226: Sessional Based on ETE 4225

Experiment 02

Study of Image Intensity Transformations and Image Filtering in Spatial Domain

---

Submitted by:

Al Nahian Mugdho

Roll: 1804021

Session: 2018-19

Submitted to:

Dr. Shah Ariful Hoque Chowdhury

Associate Professor

Dept. of ETE, RUET

Report (Teacher's Section) **Viva**

Report		Viva
☐ Excellent		☐ Excellent
☐ Very Good		☐ Very Good
☐ Good	_____	☐ Good
☐ Moderate	Signature	☐ Moderate
☐ Poor		☐ Poor

Experiment 2

Study of Image Intensity Transformations and Image Filtering in Spatial Domain

2.1 Program Code

2.1.1 Histogram Equalization for Gray Image

```
#Histogram Equalization for Gray Image:


import cv2
import matplotlib.pyplot as plt
img = cv2.imread("new.jpg", cv2.IMREAD_GRAYSCALE)
equ = cv2.equalizeHist(img)
plt.figure(figsize=(20, 20))
plt.subplot(2,2,1)
plt.title("Original image",fontsize = 20)
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.xticks([])
plt.yticks([])
plt.subplot(2,2,2)
plt.title("Histogram image",fontsize = 20)
plt.imshow(cv2.cvtColor(equ, cv2.COLOR_BGR2RGB))
plt.xticks([])
plt.yticks([])
plt.subplot(2,2,3)
plt.title("Original image histogram",fontsize = 20)
plt.hist(img.ravel(),256,[0,256]);
plt.subplot(2,2,4)

```

```
plt.title("Original image histogram",fontsize = 20)
plt.hist(equ.ravel(),256,[0,256]);

plt.show()
```

2.1.2 Result of Histogram Equalization for Gray Image

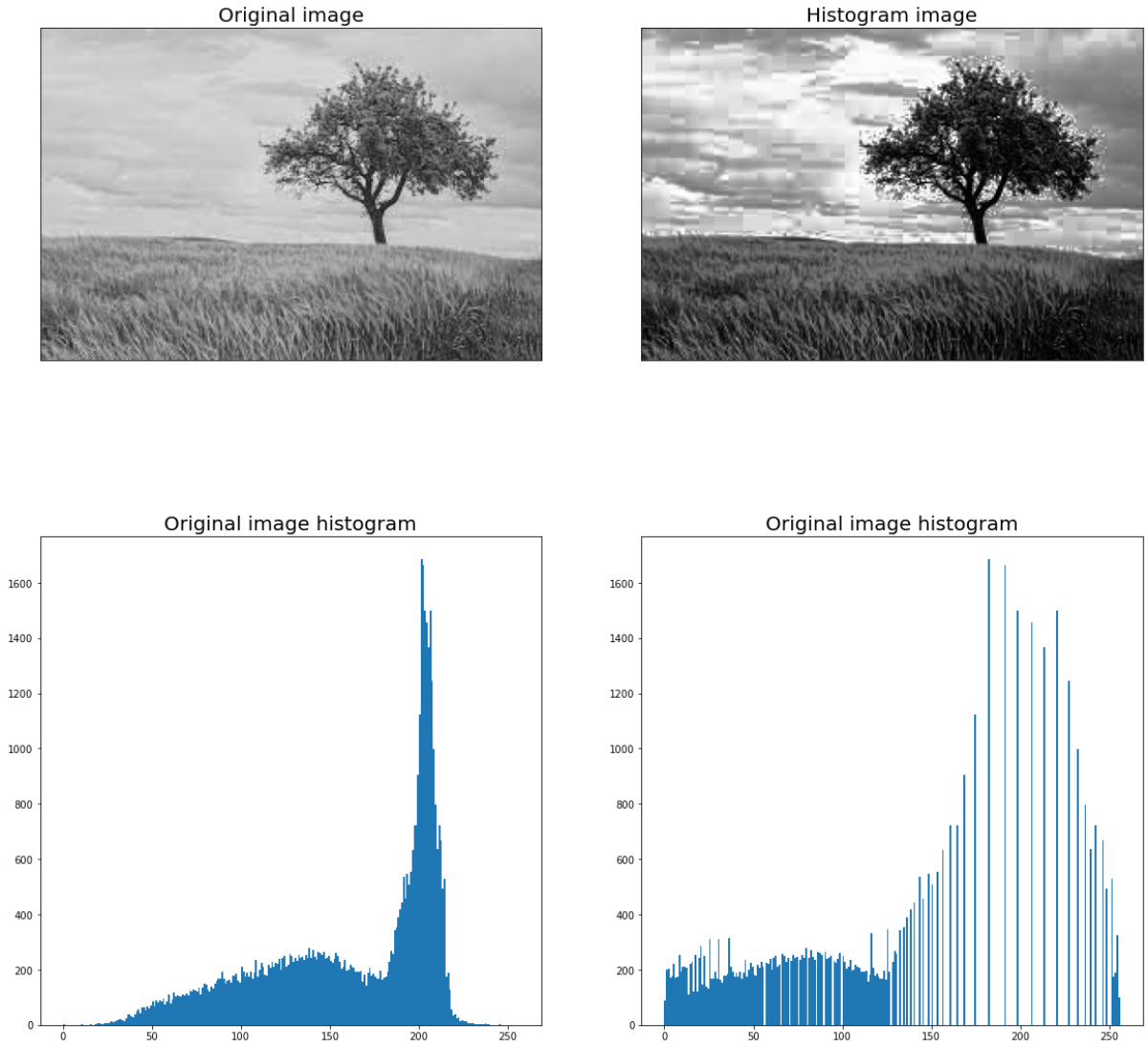

Figure 2.1: Histogram Equalization for Gray Image

2.1.3 Histogram Equalization for Color Image

```
#Histogram Equalization for Color Image
import cv2
import matplotlib.pyplot as plt

image = cv2.imread("new.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=(20, 20))
fig.add_subplot(2, 2, 1)
plt.imshow(original_img)
plt.title("Original Image",fontsize = 20)
plt.axis(False)

fig.add_subplot(2, 2, 2)
plt.imshow(equalized_img)
plt.title("Histogram Equalized Image",fontsize = 20)
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 for color scale picture',fontsize = 20)

```

```
fig.add_subplot(2, 2, 4)
for channel, col in enumerate(color):
    histr = cv2.calcHist([equalized_img], [channel], None,
   [256], [0, 256])
    plt.plot(histr, color=col)
    plt.xlim([0, 256])
plt.title('Histogram for Equalized color scale picture',
   fontsize = 20)

plt.show()
```

2.1.4 Result of Histogram Equalization for Color Image

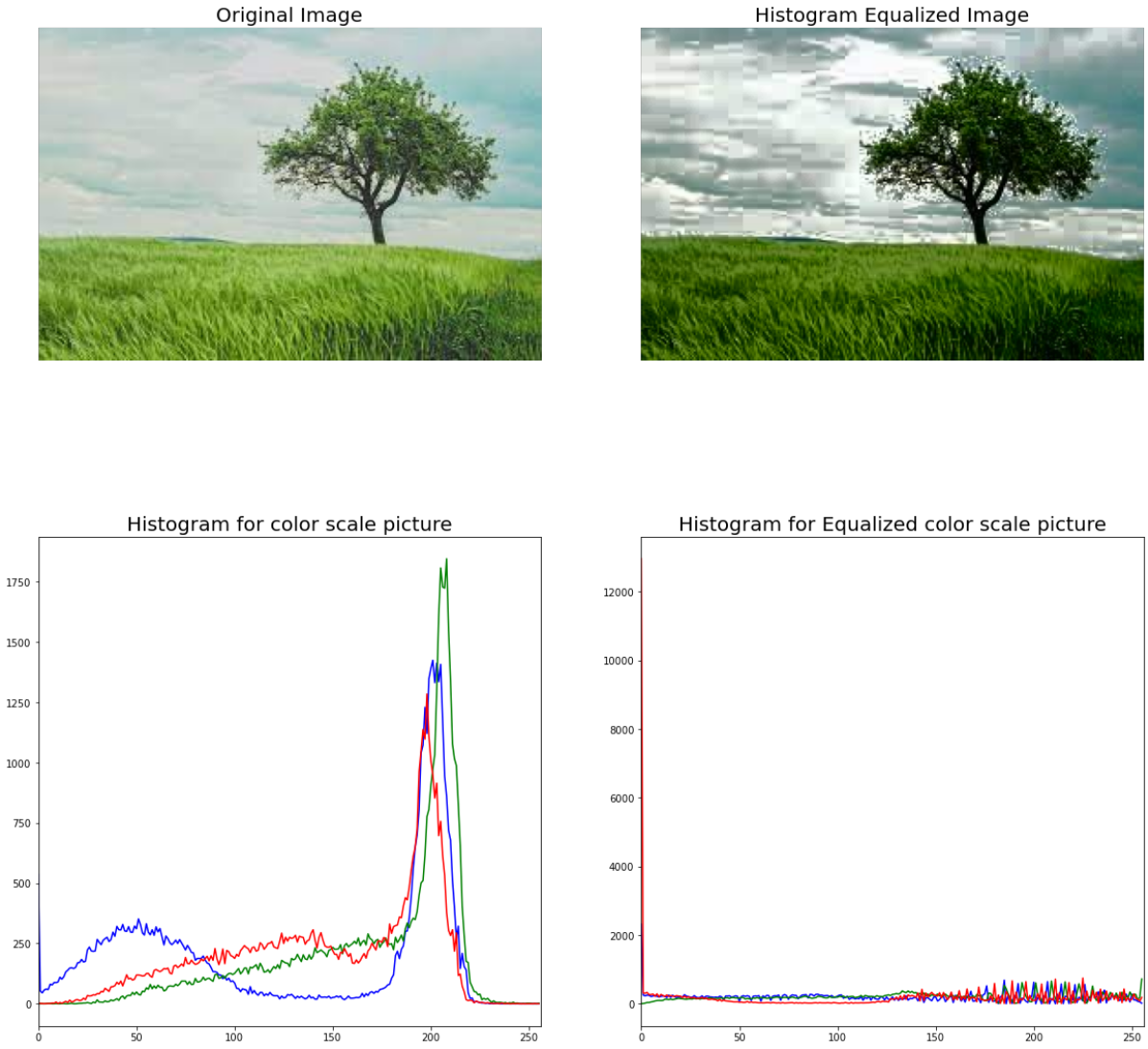

Figure 2.2: Histogram Equalization for Color Image

2.1.5 Histogram of Logarithmic Transformation

```
#Logarithimic Transformation
import cv2
import os
import numpy as np
import math
import matplotlib.pyplot as plt
img = cv2.imread("new.jpg")
```

```
img = cv2.imread("new.jpg", cv2.IMREAD_GRAYSCALE)
M, N = img.shape
g = np.zeros((M,N), dtype = np.float32)
inverse_log = np.zeros((M,N), dtype = np.float32)
c = 255 / (math.log10(1 + np.max(img)))    #see bookmark
#log transformation
for i in range(M):
    for j in range(N):
        g[i, j] = c*math.log10(1+img[i,j])

######################## Visualize
   ######################################

#inver log transformation
for i in range(M):
    for j in range(N):
        inverse_log[i, j] = 10**(img[i,j]/c) - 1




fig = plt.figure(figsize=(20, 20))
fig.add_subplot(3, 2, 1)
plt.imshow(img, cmap='gray')
plt.title("Original Image",fontsize = 20)
plt.axis(False)
```

```
37
38
39
40
41 fig.add_subplot(3, 2, 3)
42 plt.imshow(g, cmap='gray')
43 plt.title("Log Transform",fontsize = 20)
44 plt.axis(False)
45
46
47
48 fig.add_subplot(3, 2, 5)
49 plt.imshow(inverse_log, cmap='gray')
50 plt.title("Inverse Log Transform",fontsize = 20)
51 plt.axis(False)
52
53
54 fig.add_subplot(3, 2, 2)
55 plt.hist(img.ravel(),bins = 256, range= [0,256], color = 'blue
     ')
56 plt.title('Histogram for Original',fontsize = 20)
57
58 fig.add_subplot(3, 2, 4)
59 plt.hist(g.ravel(),bins = 256, range= [0,256], color = 'blue')
60 plt.title('Histogram for Log Transform picture',fontsize = 20)
61
62
63 fig.add_subplot(3, 2, 6)
64 plt.hist(inverse_log.ravel(),bins = 256, range= [0,256], color
      = 'blue')
```

```
65 plt.title('Histogram for inverse Log Transform picture',
       fontsize = 20)
66
67
68 plt.show()
```

2.1.6 Result of Logarithmic Transformation

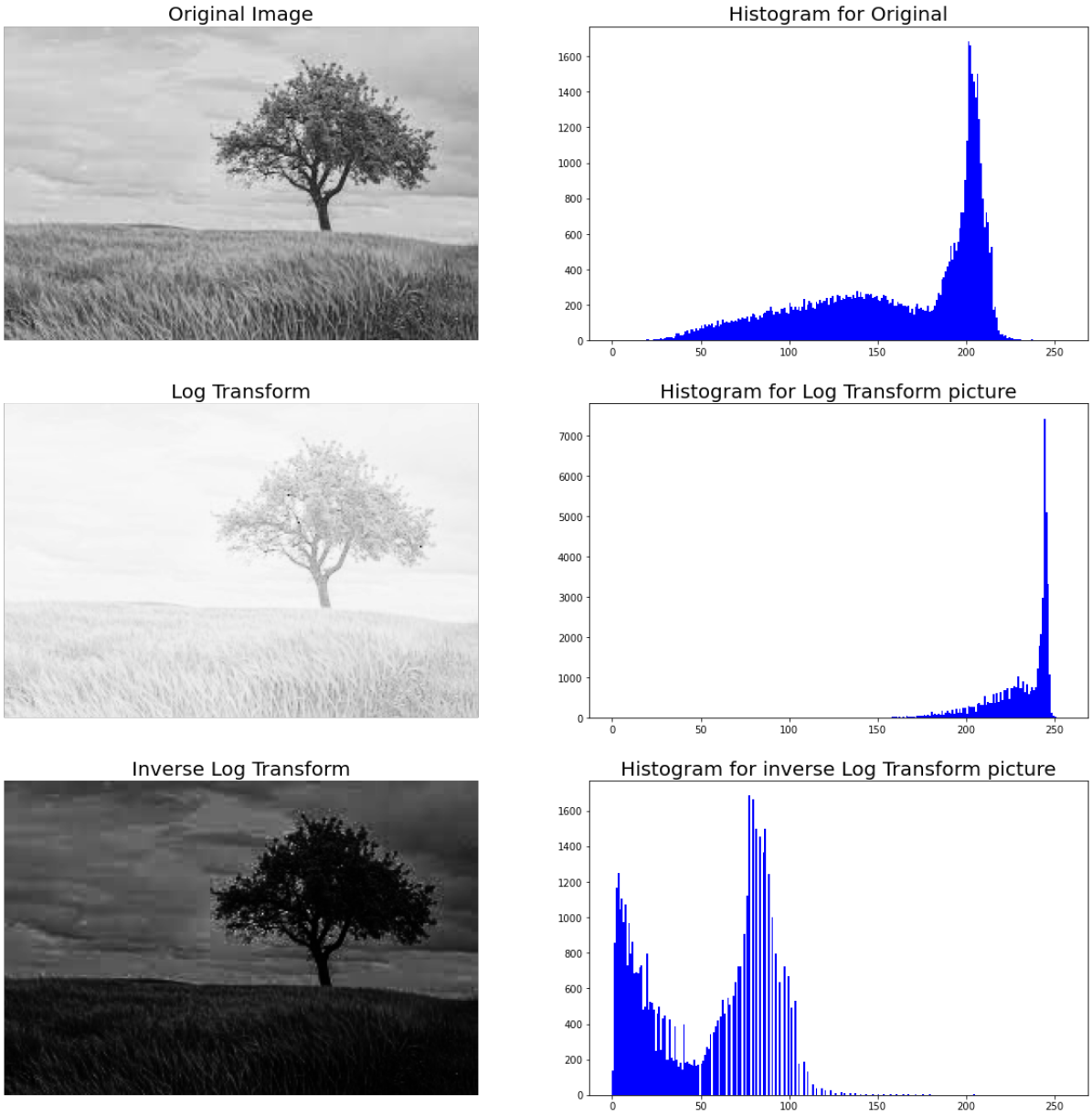

Figure 2.3: Histogram for Logarithmic Transformation

2.1.7 Histogram of Power Law Transformation

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
img = cv2.imread("new.jpg")
gamma_two_point_two = np.array(255*(img/255)**2.2,dtype='uint8')
gamma_point_four = np.array(255*(img/255)**0.4,dtype='uint8')
disp_img1= cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
disp_img2= cv2.cvtColor(gamma_two_point_two, cv2.COLOR_BGR2RGB)
disp_img3= cv2.cvtColor(gamma_point_four, cv2.COLOR_BGR2RGB)
fig = plt.figure(figsize=(20, 20))
fig.add_subplot(3, 2, 1)
plt.imshow(disp_img1, cmap='gray')
plt.title("Original Image",fontsize = 20)
plt.axis(False)



fig.add_subplot(3, 2, 3)
plt.imshow(disp_img2, cmap='gray')
plt.title("Power Law Transformed Image gamma(2.2)",fontsize = 20)
plt.axis(False)


fig.add_subplot(3, 2, 5)
```

```
26 plt.imshow(disp_img3, cmap='gray')
27 plt.title("Power Law Transformed Image  gamma(0.4)",fontsize =
       20)
28 plt.axis(False)
29
30 color = ('b', 'g', 'r')
31 fig.add_subplot(3, 2, 2)
32
33 for channel, col in enumerate(color):
34     histr = cv2.calcHist([disp_img1], [channel], None, [256],
      [0, 256])
35     plt.plot(histr, color=col)
36     plt.xlim([0, 256])
37 plt.title('Histogram for Original',fontsize = 20)
38
39 fig.add_subplot(3, 2, 4)
40
41 for channel, col in enumerate(color):
42     histr = cv2.calcHist([disp_img2], [channel], None, [256],
      [0, 256])
43     plt.plot(histr, color=col)
44     plt.xlim([0, 256])
45 plt.title('Histogram for Power Law Transform picture gamma
      (2.2)',fontsize = 20)
46
47
48
49 fig.add_subplot(3, 2, 6)
50
51 for channel, col in enumerate(color):
```

```
52      histr = cv2.calcHist([disp_img3], [channel], None, [256],
    [0, 256])
53      plt.plot(histr, color=col)
54      plt.xlim([0, 256])
55  plt.title('Histogram for Power Law Transform picture gamma
    (0.4)',fontsize = 20)
56
57
58
59  plt.show()
```

2.1.8 Result of Power Law Transformation

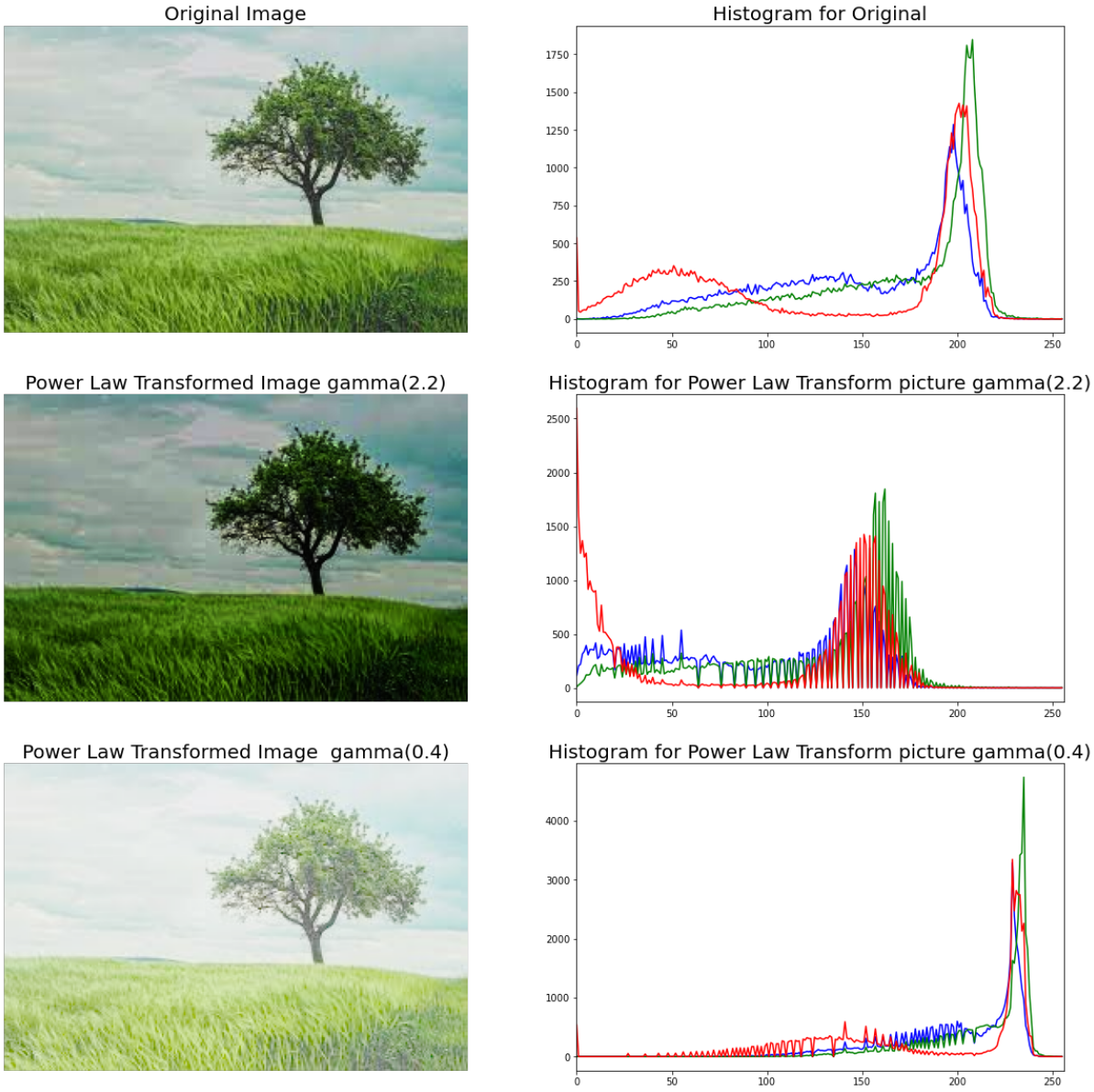

Figure 2.4: Histogram for Power Law Transformation

2.1.9 Spatial Domain Transformation

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
img = cv2.imread("new.jpg")

#form the filters
```

```
#kernel_identity = np.array([[0,0,0],[0,1,0],[0,0,0]])
kernel_3 = np.ones((3,3),dtype=np.float32)/(3*3)
#kernel_11 = np.ones((11,11),dtype=np.float32)/(11*11)
#LPF box
output = cv2.filter2D(img,-1,kernel_3)


#HPF
laplacian_kernel = np.array([
    [1, 1, 1],
    [1, -8, 1],
    [1, 1, 1]
])
output2 = cv2.filter2D(img,-1,laplacian_kernel)

output2 = img+output2

# Gaussian LPF

Gaussian=np.array([[0.3679,0.6065,0.3679],
                   [0.6065,  1, 0.6065],
                   [0.3679,0.6065,0.3679]],dtype=np.float32)
   /4.8976


output3 = cv2.filter2D(img,-1,Gaussian)


#Median PF (noise reduction)

```

```
output4 = cv2.medianBlur(img,5)

plt.figure(figsize=(20,20 ))
plt.subplot(6, 2, 1)
plt.title("Original image",fontsize =20)
plt.axis(False)
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))

plt.subplot(6, 2, 2)
plt.title('Histogram for original picture',fontsize =20)

color = ('b', 'g', 'r')
for channel, col in enumerate(color):
    histr = cv2.calcHist([img], [channel], None, [256], [0,
   256])
    plt.plot(histr, color=col)
    plt.xlim([0, 256])

plt.subplot(6, 2, 3)

plt.title("BOX filter LPF",fontsize =20)
plt.axis(False)
plt.imshow(cv2.cvtColor(output, cv2.COLOR_BGR2RGB))



plt.subplot(6, 2, 4)
for channel, col in enumerate(color):
    histr = cv2.calcHist([output], [channel], None, [256], [0,
    256])
```

```
64      plt.plot(histr, color=col)
65      plt.xlim([0, 256])
66  plt.title('Histogram for Box Filter Blurred picture',fontsize
       =20)
67
68
69  plt.subplot(6, 2, 5)
70
71  plt.title("HIGH PASS Filter",fontsize =20)
72  plt.axis(False)
73  plt.imshow(cv2.cvtColor(output2, cv2.COLOR_BGR2RGB))
74
75
76  plt.subplot(6, 2, 6)
77  for channel, col in enumerate(color):
78      histr = cv2.calcHist([output2], [channel], None, [256],
       [0, 256])
79      plt.plot(histr, color=col)
80      plt.xlim([0, 256])
81  plt.title('Histogram for High Pass Filter picture',fontsize
       =20)
82
83  plt.subplot(6, 2, 7)
84
85  plt.title("Gaussian Filter",fontsize =20)
86  plt.axis(False)
87  plt.imshow(cv2.cvtColor(output3, cv2.COLOR_BGR2RGB))
88
89
90  plt.subplot(6, 2, 8)
```

```
for channel, col in enumerate(color):
    histr = cv2.calcHist([output3], [channel], None, [256],
   [0, 256])
    plt.plot(histr, color=col)
    plt.xlim([0, 256])
plt.title('Histogram for gaussian lpf picture',fontsize =20)


plt.subplot(6, 2, 9)

plt.title("Median Filter",fontsize =20)
plt.axis(False)
plt.imshow(cv2.cvtColor(output4, cv2.COLOR_BGR2RGB))


plt.subplot(6, 2, 10)
for channel, col in enumerate(color):
    histr = cv2.calcHist([output4], [channel], None, [256],
   [0, 256])
    plt.plot(histr, color=col)
    plt.xlim([0, 256])
plt.title('Histogram for Median picture',fontsize =20)


plt.show()
```

2.1.10 Result of Spatial Domain Transformation

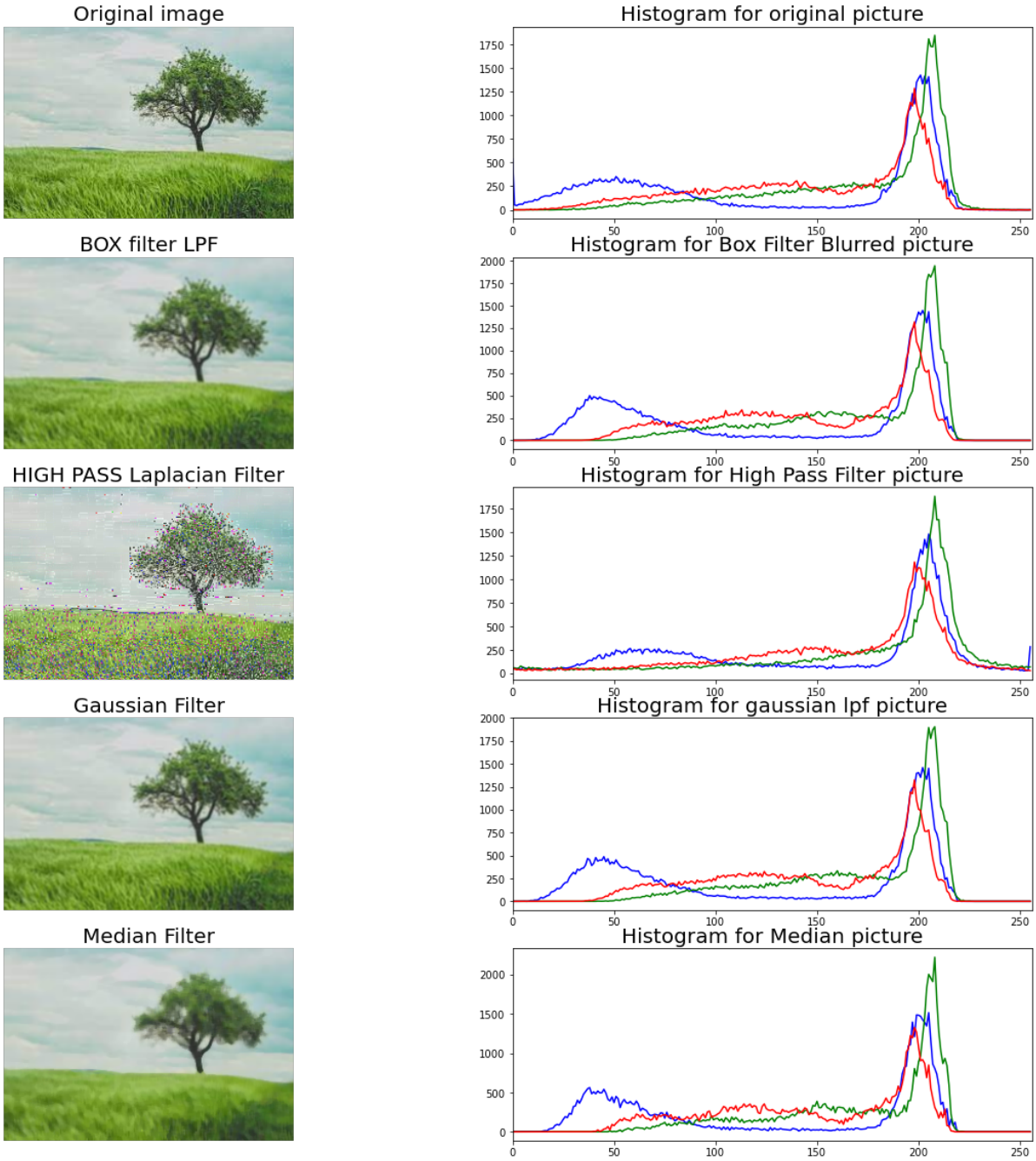

Figure 2.5: Spatial Domain Transformation

Heaven's Light is Our Guide

Rajshahi University of Engineering & Technology

Department of Electronics & Telecommunication Engineering

ETE 4226: Sessional Based on ETE 4225

Experiment 03

Study of Image Filtering in Frequency Domain

Submitted by:

Al Nahian Mugdho

Roll: 1804021

Session: 2018-19

Submitted to:

Dr. Shah Ariful Hoque Chowdhury

Associate Professor

Dept. of ETE, RUET

Report **(Teacher's Section)** **Viva**

Report		Viva
☐ Excellent		☐ Excellent
☐ Very Good		☐ Very Good
☐ Good	_____	☐ Good
☐ Moderate	Signature	☐ Moderate
☐ Poor		☐ Poor

Experiment 3

Study of Image Filtering in Frequency Domain

3.1 Program Code

3.1.1 Low Pass Filter

```
# LOW pass filter

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

def image_preprocess(image_path):
    img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    return img

def apply_filter(img_fft, filter_type, cutoff_freq, order=None
   , sigma=None):
    rows, cols = img_fft.shape
    center_row, center_col = rows // 2 , cols // 2

    # Create a filter
    filter = np.zeros((rows, cols), dtype = np.float32)

    for i in range(rows):
        for j in range(cols):
            distance = np.sqrt((i - center_row)**2 + (j -
   center_col)**2)

            if filter_type == 'ideal':
```

```
                if distance <= cutoff_freq:
                    filter[i, j] = 1
            elif filter_type == 'butterworth':
                filter[i, j] = 1 / (1 + (distance / cutoff_freq)**(2 * order))
            elif filter_type == 'gaussian':
                filter[i, j] = np.exp(-(distance**2) / (2 * (sigma**2)))

    # Apply the filter to the image in the frequency domain
    img_fft_filtered = img_fft * filter

    return img_fft_filtered

def image_postprocess(img_fft_filtered):
    img_filtered = np.fft.ifft2(np.fft.ifftshift(img_fft_filtered)).real
    #img_filtered = np.uint8(np.clip(img_filtered, 0, 255))
    return img_filtered

def main():
    image_path = 'small2.jpg'
    img = image_preprocess(image_path)

    # Fourier Transform
    img_fft = np.fft.fftshift(np.fft.fft2(img))

    # Apply low-pass filters
    cutoff_freq = 30
    order = 1
```

```
50      sigma = cutoff_freq
51
52      img_fft_ideal = apply_filter(img_fft.copy(), 'ideal',
    cutoff_freq)
53      img_fft_butterworth = apply_filter(img_fft.copy(), '
    butterworth', cutoff_freq, order)
54      img_fft_gaussian = apply_filter(img_fft.copy(), 'gaussian'
    , cutoff_freq, sigma=sigma)
55
56      # Inverse Fourier Transform and Post-process
57      img_filtered_ideal = image_postprocess(img_fft_ideal)
58      img_filtered_butterworth = image_postprocess(
    img_fft_butterworth)
59      img_filtered_gaussian = image_postprocess(img_fft_gaussian
    )
60
61      # Display the results
62      plt.figure(figsize=(20,20 ))
63
64      plt.subplot(1,4,1), plt.imshow(img, cmap='gray'), plt.
    title('Original Image',fontsize=20),plt.axis(False)
65      #plt.subplot(1,4,2), plt.imshow(np.log(1 + np.abs(img_fft)
    ), cmap='gray'), plt.title('Fourier Transform')
66
67      plt.subplot(1,4,2), plt.imshow(img_filtered_ideal, cmap='
    gray'), plt.title('Ideal LPF', fontsize=20),plt.axis(False)
68      plt.subplot(1,4,3), plt.imshow(img_filtered_butterworth,
    cmap='gray'), plt.title('Butterworth LPF', fontsize=20),plt
    .axis(False)
69      plt.subplot(1,4,4), plt.imshow(img_filtered_gaussian, cmap
```

```
    ='gray'), plt.title('Gaussian LPF', fontsize=20),plt.axis(
    False)

70

71      plt.show()

72

73  if __name__ == "__main__":
74      main()
```

3.1.2 Result of Low Pass Filter

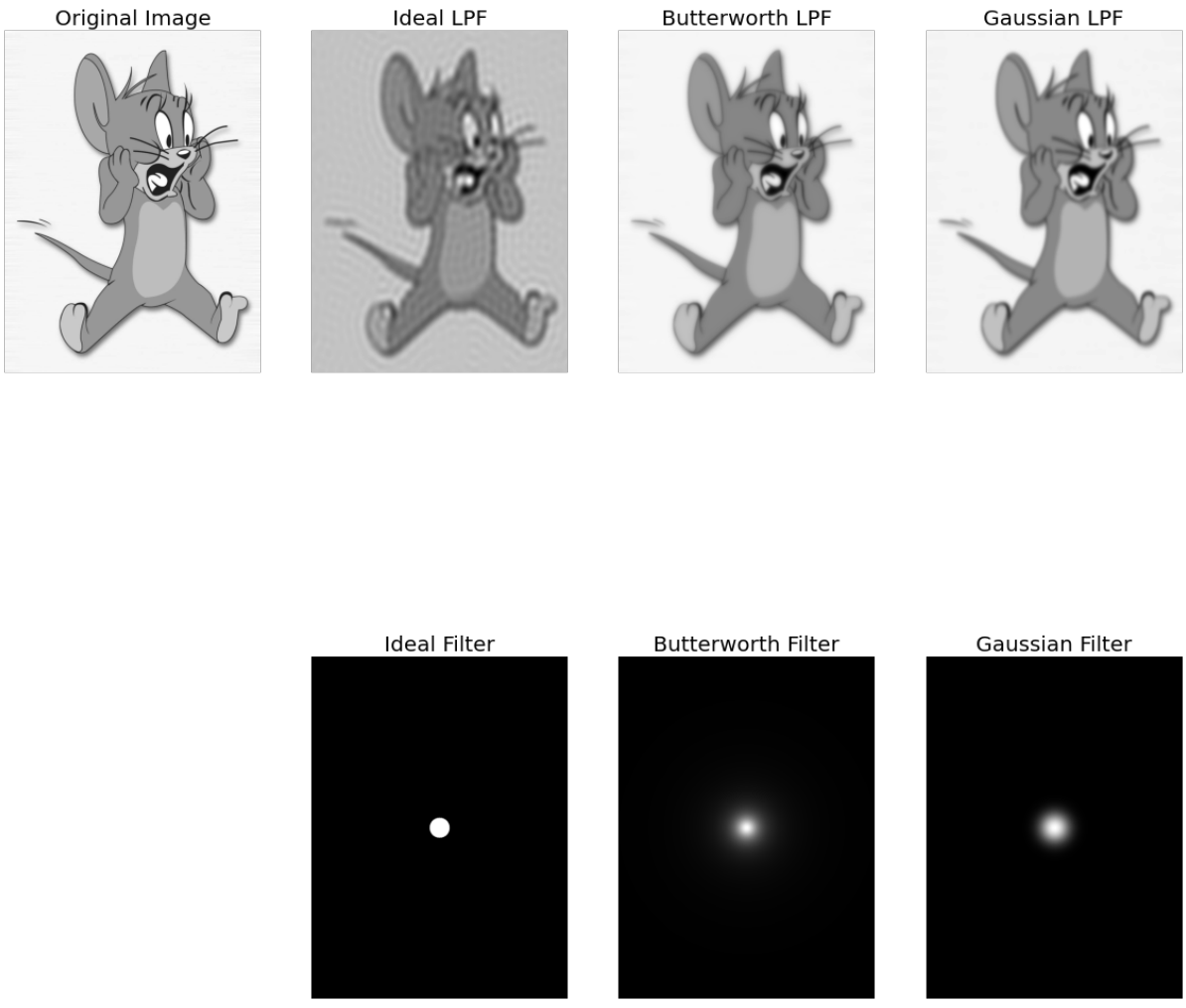

Figure 3.1: Low Pass Filtering

3.1.3 High Pass Filter

```
# High Pass filter
import cv2
import numpy as np
import matplotlib.pyplot as plt

def image_preprocess(image_path):
    img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    return img

def apply_filter(img_fft, filter_type, cutoff_freq, order=None
   , sigma=None):
    rows, cols = img_fft.shape
    center_row, center_col = rows // 2 , cols // 2

    # Create a filter
    filter = np.zeros((rows, cols), dtype = np.float32)

    for i in range(rows):
        for j in range(cols):
            distance = np.sqrt((i - center_row)**2 + (j -
   center_col)**2)

            if filter_type == 'ideal':
                if distance <= cutoff_freq:
                    filter[i, j] = 1

            elif filter_type == 'butterworth':
                filter[i, j] = 1 / (1 + (distance /
```

```
    cutoff_freq)**(2 * order))
27              elif filter_type == 'gaussian':
28                  filter[i, j] = np.exp(-(distance**2) / (2 * (
    sigma**2)))
29      filter = 1 - filter
30      # Apply the filter to the image in the frequency domain
31      img_fft_filtered = img_fft * filter
32
33      return img_fft_filtered,filter
34
35  def image_postprocess(img_fft_filtered):
36      img_filtered = np.fft.ifft2(np.fft.ifftshift(
    img_fft_filtered)).real
37      #img_filtered = np.uint8(np.clip(img_filtered, 0, 255))
38      return img_filtered
39
40  def main():
41      image_path = 'small2.jpg'
42      img = image_preprocess(image_path)
43
44      # Fourier Transform
45      img_fft = np.fft.fftshift(np.fft.fft2(img))
46
47      # Apply low-pass filters
48      cutoff_freq = 30
49      order = 2
50      sigma = cutoff_freq
51
52      img_fft_ideal,ideal = apply_filter(img_fft.copy(), 'ideal'
    , cutoff_freq)
```

```
53      img_fft_butterworth,butter = apply_filter(img_fft.copy(),
    'butterworth', cutoff_freq, order)
54      img_fft_gaussian,gauss = apply_filter(img_fft.copy(), '
    gaussian', cutoff_freq, sigma=sigma)
55
56      # Inverse Fourier Transform and Post-process
57      img_filtered_ideal = image_postprocess(img_fft_ideal)
58      img_filtered_butterworth = image_postprocess(
    img_fft_butterworth)
59      img_filtered_gaussian = image_postprocess(img_fft_gaussian
    )
60
61      # Display the results
62      plt.figure(figsize=(20,20 ))
63
64      plt.subplot(2,4,1), plt.imshow(img, cmap='gray'), plt.
    title('Original Image'),plt.axis(False)
65
66
67      plt.subplot(2,4,2), plt.imshow(img_filtered_ideal, cmap='
    gray'), plt.title('Ideal HPF', fontsize=16),plt.axis(False)
68      plt.subplot(2,4,3), plt.imshow(img_filtered_butterworth,
    cmap='gray'), plt.title('Butterworth HPF', fontsize=16),plt
    .axis(False)
69      plt.subplot(2,4,4), plt.imshow(img_filtered_gaussian, cmap
    ='gray'), plt.title('Gaussian HPF', fontsize=16),plt.axis(
    False)
70      #plt.subplot(1,4,4), plt.imshow(np.log(1 + np.abs(img_fft)
    ), cmap='gray'), plt.title('Fourier Transform')
71      plt.subplot(2,4,6), plt.imshow(ideal, cmap='gray'), plt.
```

```
    title('Ideal Filter', fontsize=20),plt.axis(False)
72      plt.subplot(2,4,7), plt.imshow(butter, cmap='gray'), plt.
    title('Butterworth Filter', fontsize=20),plt.axis(False)
73      plt.subplot(2,4,8),  plt.imshow(gauss, cmap='gray'), plt.
    title('Gaussian Filter', fontsize=20),plt.axis(False)
74      plt.show()
75
76  if __name__ == "__main__":
77      main()
```

3.1.4 Result of High Pass Filter

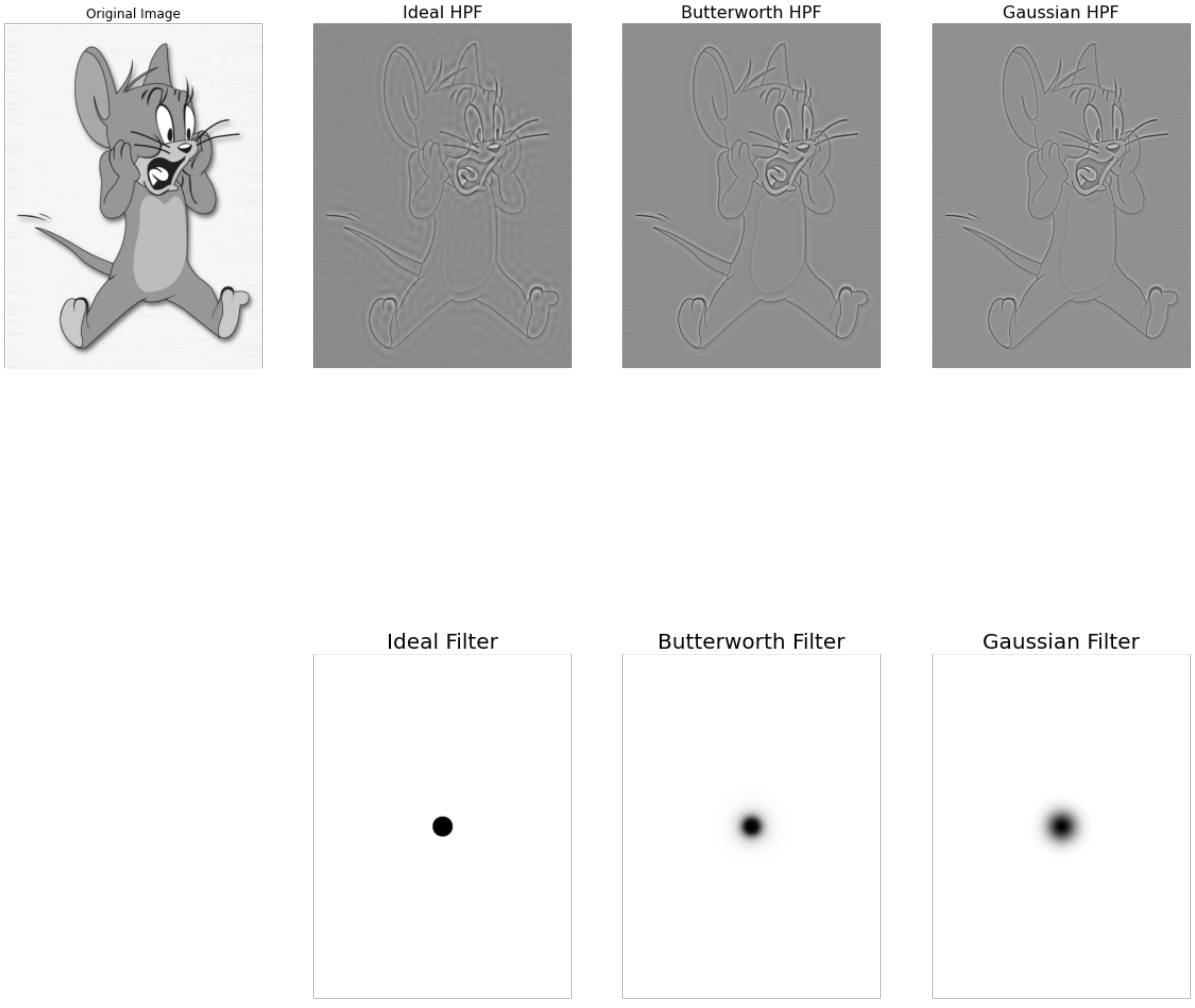

Figure 3.2: High Pass Filter

3.1.5 Band Pass Filter

```
# Band PASS Filter
import cv2
import numpy as np
import matplotlib.pyplot as plt

def image_preprocess(image_path):
    img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    return img

def apply_filter(img_fft, filter_type, cutoff_freq,order=None,
    W=None):
    rows, cols = img_fft.shape
    center_row, center_col = rows // 2 , cols // 2

    # Create a filter
    filter = np.zeros((rows, cols), dtype=np.float32)

    for i in range(rows):
        for j in range(cols):
            distance = np.sqrt((i - center_row)**2 + (j -
   center_col)**2)

            if filter_type == 'ideal':
                if  cutoff_freq - W/2 < distance < cutoff_freq
    + W/2:
                    filter[i, j] = 0
                else:
                    filter[i, j] = 1
```

```
26
27              elif filter_type == 'butterworth':
28                  filter[i, j] = (1 / (1 + (((distance * W) / ((
    distance**2) - (cutoff_freq**2)))**(2 * order))))
29              elif filter_type == 'gaussian':
30                  filter[i, j] = 1 - np.exp(-((distance**2 -
    cutoff_freq**2) / (distance * W))**(2))
31      filter = 1 - filter
32      # Apply the filter to the image in the frequency domain
33      img_fft_filtered = img_fft * filter
34
35      return img_fft_filtered,filter
36
37  def image_postprocess(img_fft_filtered):
38      img_filtered = np.fft.ifft2(np.fft.ifftshift(
    img_fft_filtered)).real
39      return img_filtered
40
41  def main():
42      image_path = 'small2.jpg'
43      img = image_preprocess(image_path)
44
45      # Fourier Transform
46      img_fft = np.fft.fftshift(np.fft.fft2(img))
47
48      # Apply band-reject filters
49      cutoff_freq = 30
50      order = 2
51      W = 50
52
```

```
53
54      # Apply ideal filter without W
55      img_fft_ideal,ideal = apply_filter(img_fft.copy(), 'ideal'
    , cutoff_freq, W=W)
56
57      # Apply other filters with W
58      img_fft_butterworth,butter = apply_filter(img_fft.copy(),
    'butterworth', cutoff_freq,order=order, W=W)
59      img_fft_gaussian,gauss = apply_filter(img_fft.copy(), '
    gaussian', cutoff_freq,order=order, W=W)
60
61      # Inverse Fourier Transform and Post-process
62      img_filtered_ideal = image_postprocess(img_fft_ideal)
63      img_filtered_butterworth = image_postprocess(
    img_fft_butterworth)
64      img_filtered_gaussian = image_postprocess(img_fft_gaussian
    )
65
66      # Display the results
67      plt.figure(figsize=(20, 20))
68
69      plt.subplot(2, 4, 1), plt.imshow(img, cmap='gray'), plt.
    title('Original Image'), plt.axis(False)
70
71      plt.subplot(2, 4, 2), plt.imshow(img_filtered_ideal, cmap=
    'gray'), plt.title('Ideal BPF', fontsize=16), plt.axis(
    False)
72      plt.subplot(2, 4, 3), plt.imshow(img_filtered_butterworth,
     cmap='gray'), plt.title('Butterworth BPF', fontsize=16),
    plt.axis(False)
```

```
73      plt.subplot(2, 4, 4), plt.imshow(img_filtered_gaussian,
    cmap='gray'), plt.title('Gaussian BPF', fontsize=16), plt.
    axis(False)
74
75      plt.subplot(2,4,6), plt.imshow(ideal, cmap='gray'), plt.
    title('Ideal Filter', fontsize=20),plt.axis(False)
76      plt.subplot(2,4,7), plt.imshow(butter, cmap='gray'), plt.
    title('Butterworth Filter', fontsize=20),plt.axis(False)
77      plt.subplot(2,4,8),  plt.imshow(gauss, cmap='gray'), plt.
    title('Gaussian Filter', fontsize=20),plt.axis(False)
78      plt.show()
79
80  if __name__ == "__main__":
81      main()
```

3.1.6 Result of Band Pass Filter

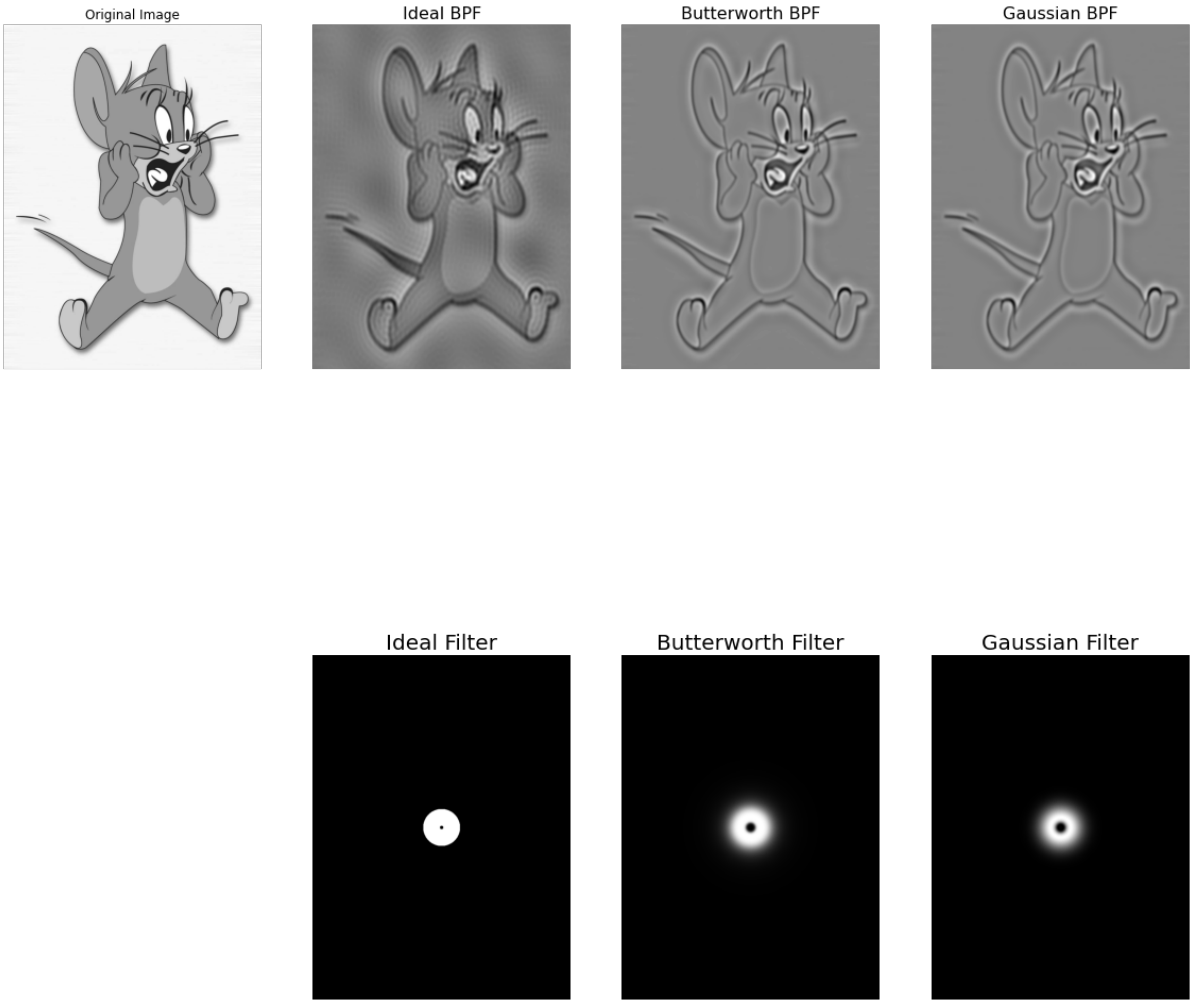

Figure 3.3: Band Pass Filter

3.1.7 Band Reject Filter

```
# Band REJECT Filter
import cv2
import numpy as np
import matplotlib.pyplot as plt

def image_preprocess(image_path):
    img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
```

```
    return img

def apply_filter(img_fft, filter_type, cutoff_freq,order=None, W=None):
    rows, cols = img_fft.shape
    center_row, center_col = rows // 2 , cols // 2

    # Create a filter
    filter = np.zeros((rows, cols), dtype=np.float32)

    for i in range(rows):
        for j in range(cols):
            distance = np.sqrt((i - center_row)**2 + (j - center_col)**2)

            if filter_type == 'ideal':
                if  cutoff_freq - W/2 < distance < cutoff_freq + W/2:
                    filter[i, j] = 0
                else:
                    filter[i, j] = 1

            elif filter_type == 'butterworth':
                filter[i, j] = (1 / (1 + (((distance * W) / ((distance**2) - (cutoff_freq**2)))**(2 * order))))
            elif filter_type == 'gaussian':
                filter[i, j] = 1 - np.exp(-((distance**2 - cutoff_freq**2) / (distance * W))**(2))

    # Apply the filter to the image in the frequency domain
```

```
33      img_fft_filtered = img_fft * filter
34
35      return img_fft_filtered,filter
36
37  def image_postprocess(img_fft_filtered):
38      img_filtered = np.fft.ifft2(np.fft.ifftshift(
    img_fft_filtered)).real
39      return img_filtered
40
41  def main():
42      image_path = 'small2.jpg'
43      img = image_preprocess(image_path)
44
45      # Fourier Transform
46      img_fft = np.fft.fftshift(np.fft.fft2(img))
47
48      # Apply band-reject filters
49      cutoff_freq = 30
50      order = 2
51      W = 20
52
53
54      # Apply ideal filter without W
55      img_fft_ideal,ideal = apply_filter(img_fft.copy(), 'ideal'
    , cutoff_freq, W=W)
56
57      # Apply other filters with W
58      img_fft_butterworth,butter = apply_filter(img_fft.copy(),
    'butterworth', cutoff_freq,order=order, W=W)
59      img_fft_gaussian,gauss = apply_filter(img_fft.copy(), '
```

```
gaussian', cutoff_freq,order=order, W=W)

 # Inverse Fourier Transform and Post-process
 img_filtered_ideal = image_postprocess(img_fft_ideal)
 img_filtered_butterworth = image_postprocess(
img_fft_butterworth)
 img_filtered_gaussian = image_postprocess(img_fft_gaussian
)

 # Display the results
 plt.figure(figsize=(20, 20))

 plt.subplot(2, 4, 1), plt.imshow(img, cmap='gray'), plt.
title('Original Image'), plt.axis(False)

 plt.subplot(2, 4, 2), plt.imshow(img_filtered_ideal, cmap=
'gray'), plt.title('Ideal BRF', fontsize=16), plt.axis(
False)
 plt.subplot(2, 4, 3), plt.imshow(img_filtered_butterworth,
 cmap='gray'), plt.title('Butterworth BRF', fontsize=16),
plt.axis(False)
 plt.subplot(2, 4, 4), plt.imshow(img_filtered_gaussian,
cmap='gray'), plt.title('Gaussian BRF', fontsize=16), plt.
axis(False)
 plt.subplot(2,4,6), plt.imshow(ideal, cmap='gray'), plt.
title('Ideal Filter', fontsize=20),plt.axis(False)
 plt.subplot(2,4,7), plt.imshow(butter, cmap='gray'), plt.
title('Butterworth Filter', fontsize=20),plt.axis(False)
 plt.subplot(2,4,8),  plt.imshow(gauss, cmap='gray'), plt.
title('Gaussian Filter', fontsize=20),plt.axis(False)
```

```
77      plt.show()
78
79  if __name__ == "__main__":
80      main()
```

3.1.8 Result of Band Reject Filter

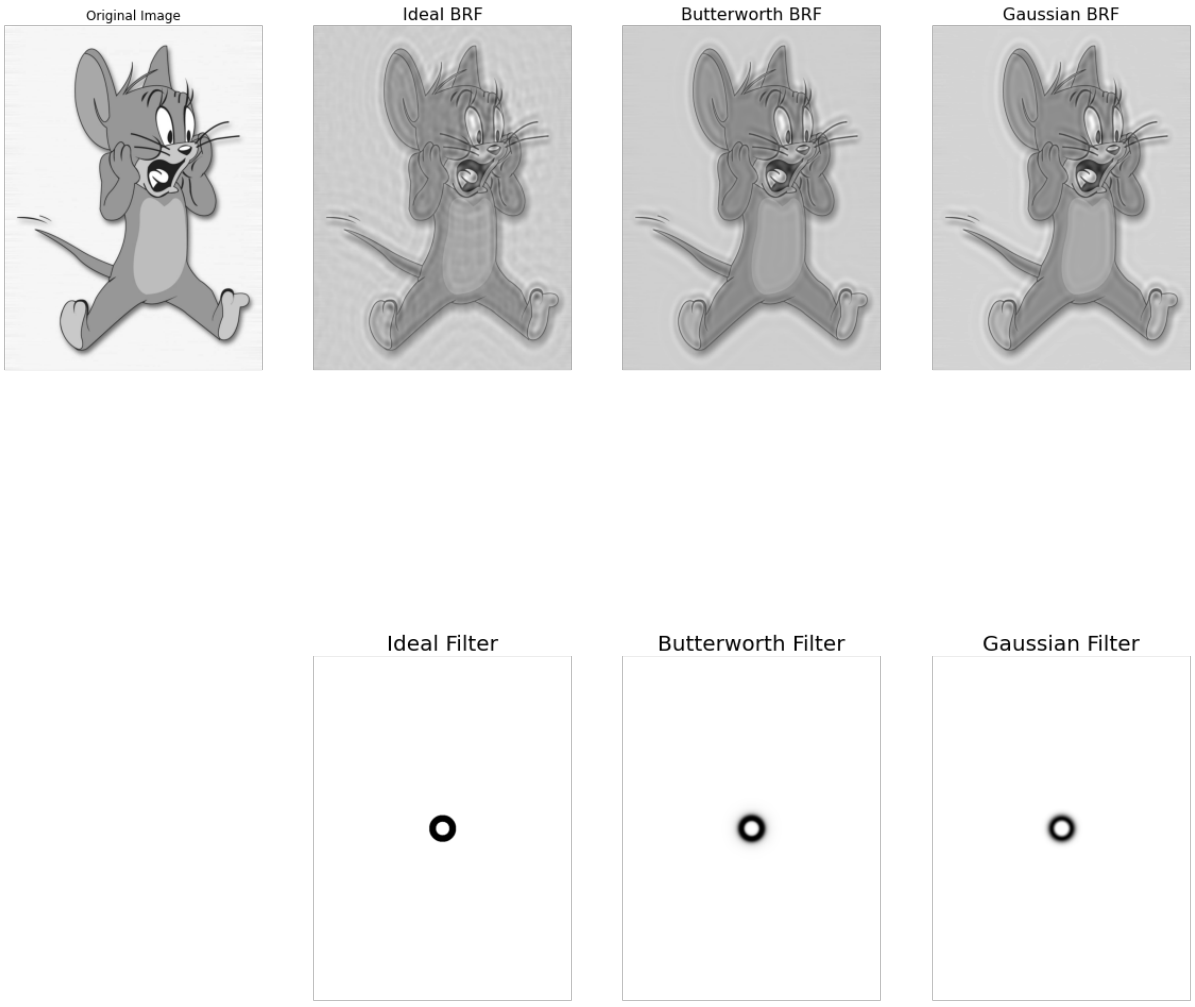

Figure 3.4: Band Reject Filter

Heaven's Light is Our Guide

Rajshahi University of Engineering & Technology

Department of Electronics & Telecommunication Engineering

ETE 4226: Sessional Based on ETE 4225

Experiment 04

Study of Line Detection, Edge Detection, and Image Segmentation.

Submitted by:

Al Nahian Mugdho

Roll: 1804021

Session: 2018-19

Submitted to:

Dr. Shah Ariful Hoque Chowdhury

Associate Professor

Dept. of ETE, RUET

Report

- ☐ Excellent
- ☐ Very Good
- ☐ Good
- ☐ Moderate
- ☐ Poor

(Teacher's Section)

_____

Signature

Viva

- ☐ Excellent
- ☐ Very Good
- ☐ Good
- ☐ Moderate
- ☐ Poor

Experiment 4

Study of Line Detection, Edge Detection, and Image Segmentation.

4.1 Program Code

4.1.1 K means Clustering

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


image = cv2.imread("new.jpg")

# convert to RGB
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# reshape the image to a 2D array of pixels and 3 color values
    (RGB)
pixel_values = image.reshape((-1, 3))
# convert to float
pixel_values = np.float32(pixel_values)

# define stopping criteria
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER
   , 200, 0.2)

# try different values of k
k_values = [1, 2, 3, 4, 5]

# display segmented images for different k values
```

```
plt.figure(figsize=(15, 8))

for i, k in enumerate(k_values, 2):
    # apply k-means clustering
    compactness, labels, centers = cv2.kmeans(pixel_values, k,
    None, criteria, 10, cv2.KMEANS_RANDOM_CENTERS)

    # convert back to 8 bit values
    centers = np.uint8(centers)

    # flatten the labels array
    labels = labels.flatten()

    # convert all pixels to the color of the centroids
    segmented_image = centers[labels]

    # reshape back to the original image dimension
    segmented_image = segmented_image.reshape(image.shape)

    # display the images
    plt.subplot(2, 3, 1), plt.imshow(image), plt.title('
   original Image')
    plt.subplot(2, 3, i), plt.imshow(segmented_image)
    plt.title(f'k = {k} Segmentation')

plt.show()
```

4.1.2 Result of K Means Clustering

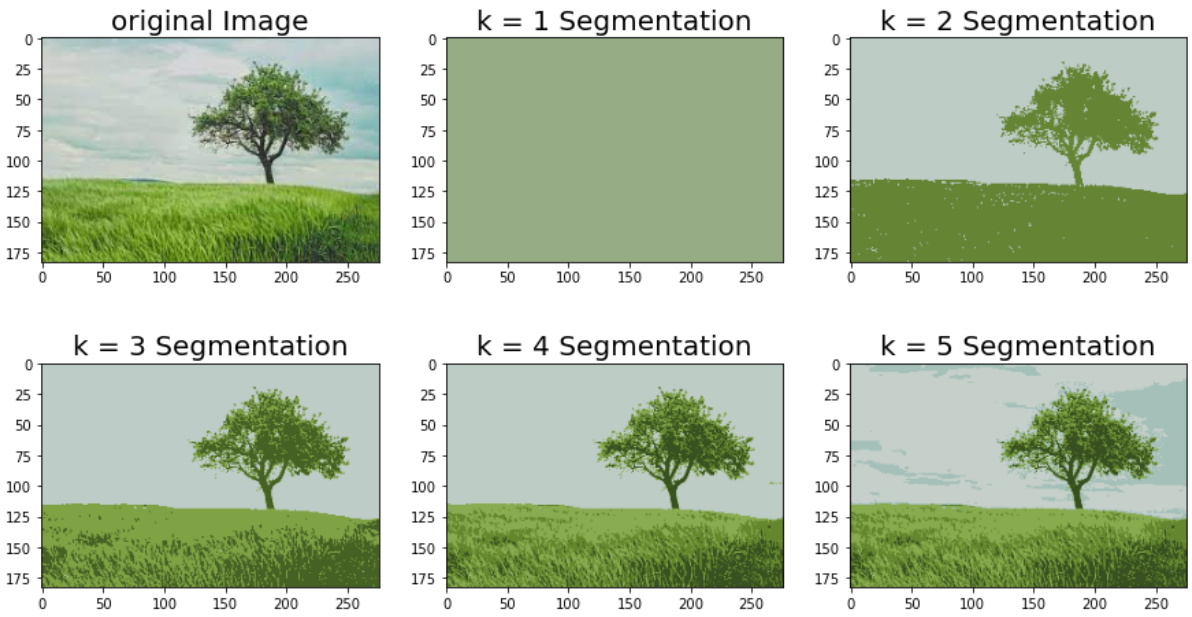

Figure 4.1: K means Clustering

4.1.3 Line Detection

```
### Line Detection
import cv2
import numpy as np
import matplotlib.pyplot as plt
def detect_lines(image_path):
    # Read the image
    image = cv2.imread(image_path)

    # Convert the image to grayscale
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Apply GaussianBlur to reduce noise and help edge
  detection
    blurred = cv2.GaussianBlur(gray, (5, 5), 0)
```

```
14
15      # Apply Canny edge detection
16      edges = cv2.Canny(blurred, 50, 150)
17
18      # Define the line kernels
19      horizontal_kernel = np.array([[-1, -1, -1],
20                                    [2, 2, 2],
21                                    [-1, -1, -1]])
22
23      minus_45_degree_kernel = np.array([[2, -1, -1],
24                                         [-1, 2, -1],
25                                         [-1, -1, 2]])
26
27      vertical_kernel = np.array([[-1, 2, -1],
28                                  [-1, 2, -1],
29                                  [-1, 2, -1]])
30
31      plus_45_degree_kernel = np.array([[-1, -1, 2],
32                                        [-1, 2, -1],
33                                        [2, -1, -1]])
34
35      # Apply the kernels to enhance lines in the image
36      horizontal_lines = cv2.filter2D(edges, -1,
    horizontal_kernel)
37      minus_45_lines = cv2.filter2D(edges, -1,
    minus_45_degree_kernel)
38      vertical_lines = cv2.filter2D(edges, -1, vertical_kernel)
39      plus_45_lines = cv2.filter2D(edges, -1,
    plus_45_degree_kernel)
40
```

```
    # Display the results using Matplotlib
     plt.figure(figsize=(20, 10))
     plt.subplot(2,3,1)
     plt.imshow(image)
     plt.title('Original')

     plt.subplot(2,3,2)
     plt.imshow(horizontal_lines, cmap='gray')
     plt.title('Horizontal Lines')

     plt.subplot(2,3,3)
     plt.imshow(minus_45_lines, cmap='gray')
     plt.title('Minus 45 Degree Lines')

     plt.subplot(2,3,4)
     plt.imshow(vertical_lines, cmap='gray')
     plt.title('Vertical Lines')

     plt.subplot(2,3,5)
     plt.imshow(plus_45_lines, cmap='gray')
     plt.title('Plus 45 Degree Lines')

     plt.show()


image_path = 'red.jpg'
detect_lines(image_path)
```

4.1.4 Result of Line Detection

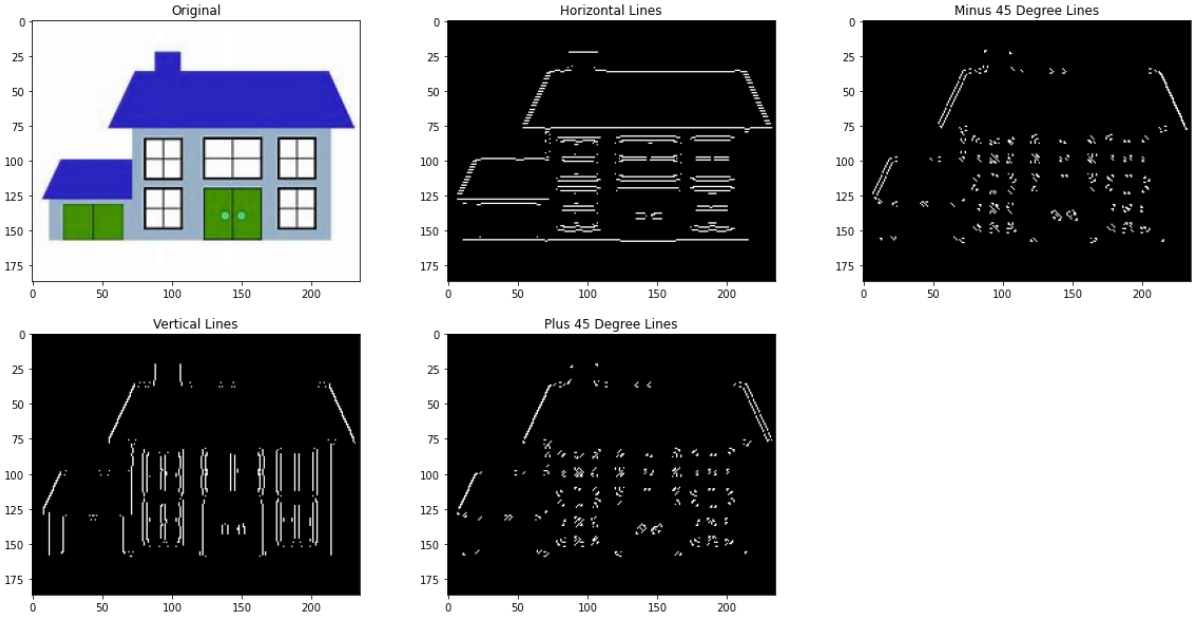

Figure 4.2: Line Detection

4.1.5 Edge Detection

```
#Edge Detection
# Laplacian 2nd order derivative
# Sobel 1st Order Derivatives
import cv2
import matplotlib.pyplot as plt

img = cv2.imread('new.jpg')
img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

blur_img = cv2.GaussianBlur(img, (3, 3), 0)

# Calculation of Sobelx
sobelx = cv2.Sobel(blur_img,cv2.CV_64F,1,0,ksize=5)

```

```
# Calculation of Sobely
sobely = cv2.Sobel(blur_img,cv2.CV_64F,0,1,ksize=5)
sobelxy = cv2.addWeighted(sobelx,0.5,sobely,0.5,0)

laplacian = cv2.Laplacian(blur_img, cv2.CV_64F, ksize=5, scale
   =1, delta=0, borderType=cv2.BORDER_DEFAULT)
canny = cv2.Canny(blur_img,80,150)
output = [img,sobelxy, laplacian,canny]
titles = ['Original', 'First Order sobel','Second Order
   Laplacian','Canny Edge Detection']
plt.figure(figsize=(20, 20))
for i in range(4):
    plt.subplot(1, 4, i + 1)
    plt.imshow(output[i], cmap='gray')
    plt.title(titles[i],fontsize=20)
    plt.xticks([])
    plt.yticks([])
plt.show()
```

4.1.6 Result of Edge Detection

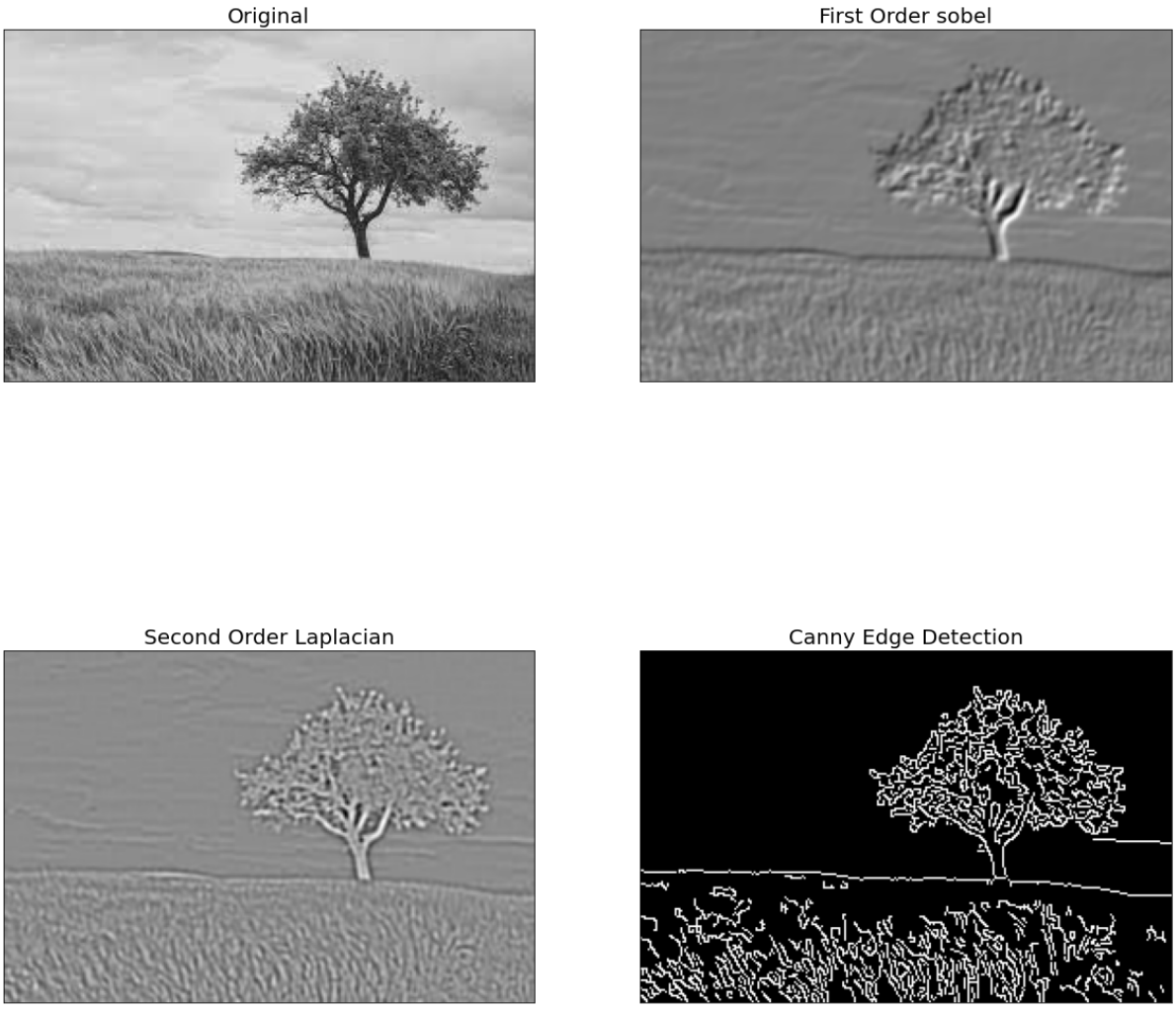

Figure 4.3: Edge detection

4.1.7 Superpixel Segmentation

```
1  ############### SLIC ############################
2
3  import cv2
4  import numpy as np
5  import matplotlib.pyplot as plt
6
7  def segment_image_slic(image_path, num_segments):
8      # Read the image
```

```
    image = cv2.imread(image_path)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    region_size = int(np.sqrt(image.shape[0] * image.shape[1]
/ num_segments))
    # Apply SLIC superpixel segmentation
    slic = cv2.ximgproc.createSuperpixelSLIC(image,
region_size=region_size, ruler=10)
    slic.iterate(num_iterations=500)

    # Get the labels and the number of superpixels
    labels = slic.getLabels()
    num_superpixels = slic.getNumberOfSuperpixels()

    # Create a mask to highlight the boundaries
    mask = slic.getLabelContourMask(thick_line=True)

    # Merge the mask with the original image
    # Merge the mask with the original image
    segmented_image = cv2.cvtColor(image, cv2.COLOR_RGB2RGBA)
    segmented_image[:, :, 3] = 255 #  alpha channel to fully
opaque
    segmented_image[mask > 0, :3] = [255, 0, 0]


    # Display the original and segmented images side by side
    plt.figure(figsize=(20, 5))

    plt.subplot(1, 2, 1)
    plt.imshow(image)
    plt.title('Original Image')
```

```
36
37      plt.subplot(1, 2, 2)
38      plt.imshow(segmented_image)
39      plt.title(f'SLIC Superpixel Segmentation for {num_segments
        } pixels')
40
41
42
43      plt.show()
44
45      return labels, num_superpixels
46
47  # Example usage
48  image_path = "new.jpg"
49  num_segments = 400  # Number of desired superpixels
50
51
52  labels, num_superpixels = segment_image_slic(image_path,
        num_segments)
53  print(f"Number of Superpixels: {num_superpixels}")
```

4.1.8 Result of Superpixel Segmentation

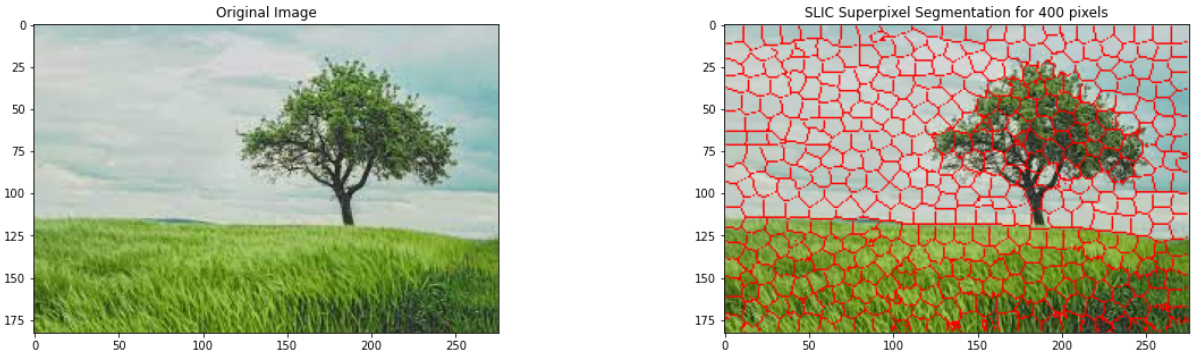

Figure 4.4: SLIC Super-pixel Segmentation

Number of Superpixels: 425

4.1.9 Hough Transform

```
1
2  ##################Hough transform#######
3  import cv2
4  import numpy as np
5  import matplotlib.pyplot as plt
6
7  # Read the image
8  image = cv2.imread('red.jpg')
9  gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
10
11 # Apply Gaussian blur to reduce noise
12 blurred_image = cv2.GaussianBlur(gray_image, (5, 5), 0)
13
14 edges = cv2.Canny(blurred_image, 50, 150)
15
16 # Perform probabilistic Hough Line Transform
```

```
lines = cv2.HoughLinesP(edges, 1, np.pi / 180, threshold=50,
    minLineLength=10, maxLineGap=5)

# Draw the detected lines on the original image
line_image = np.copy(image)
for line in lines:
    x1, y1, x2, y2 = line[0]
    cv2.line(line_image, (x1, y1), (x2, y2), (255, 0, 0), 2)

# Display the original image and the detected lines
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.title('Original Image')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(cv2.cvtColor(line_image, cv2.COLOR_BGR2RGB))
plt.title('Detected Lines')
plt.axis('off')

plt.show()
```

4.1.10 Result of Hough Transform

Figure 4.5: Hough Transform

Heaven's Light is Our Guide

Rajshahi University of Engineering & Technology

Department of Electronics & Telecommunication Engineering

ETE 4226: Sessional Based on ETE 4225

Experiment 05

Study of Implementing a Fully Connected Neural Network and Convolutional Neural Network Using Deep Learning Libraries.

Submitted by:

Al Nahian Mugdho

Roll: 1804021

Session: 2018-19

Submitted to:

Dr. Shah Ariful Hoque Chowdhury

Associate Professor

Dept. of ETE, RUET

Report **(Teacher's Section)** **Viva**

Report		Viva
☐ Excellent		☐ Excellent
☐ Very Good		☐ Very Good
☐ Good	_____	☐ Good
☐ Moderate	Signature	☐ Moderate
☐ Poor		☐ Poor

Experiment 5

Study of Implementing a Fully Connected Neural Network and Convolutional Neural Network Using Deep Learning Libraries. .

5.1 Program Code

5.1.1 Harris Detection

```
########################################## Harris Detection Assignment ##################################################
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt

def compute_harris_response(image, ksize, k):
    # Compute derivatives using Sobel operator
    sobel_x = cv.Sobel(image, cv.CV_64F, 1, 0, ksize=ksize)
    sobel_y = cv.Sobel(image, cv.CV_64F, 0, 1, ksize=ksize)

    # Compute elements of the Harris matrix
    Ixx = sobel_x**2
    Iyy = sobel_y**2
    Ixy = sobel_x * sobel_y

    # Compute the sum of squared gradients in the neighborhood
    Sxx = cv.boxFilter(Ixx, -1, (ksize, ksize))
    Syy = cv.boxFilter(Iyy, -1, (ksize, ksize))
    Sxy = cv.boxFilter(Ixy, -1, (ksize, ksize))

    # Compute the determinant and trace of the Harris matrix
    det_M = (Sxx * Syy) - (Sxy**2)
```

```
23      trace_M = Sxx + Syy
24
25      # Compute the Harris response
26      harris_response = det_M - k * (trace_M**2)
27
28      return harris_response
29
30  def detect_corners(image, threshold, ksize, k):
31      # Convert RGB image to grayscale
32      gray = cv.cvtColor(image, cv.COLOR_BGR2GRAY)
33
34      # Compute Harris response
35      harris_response = compute_harris_response(gray, ksize, k)
36
37      # Dilate the response to find local maxima
38      harris_response_dilated = cv.dilate(harris_response, None)
39
40      # Mark corners on the original image
41      image[harris_response_dilated > threshold *
     harris_response_dilated.max()] = [255, 0, 0]
42
43      return image
44
45  # Parameters
46  threshold = 0.001
47  ksize = 5  # Sobel kernel size
48  k = 0.04    # Harris sensitivity factor
49
50  # Load the image
51  img = cv.imread('red.jpg')
```

```
52 img = cv.cvtColor(img, cv.COLOR_BGR2RGB)
53 img = cv.resize(img, (600, 600))
54
55 plt.figure(figsize=(20, 10))
56 plt.subplot(1,2,1)
57 plt.imshow(img)
58 plt.title('Original')
59 plt.xticks([])
60 plt.yticks([])
61
62 # Detect corners manually
63 result_image = detect_corners(img, threshold, ksize, k)
64
65
66 plt.subplot(1,2,2)
67 plt.imshow(result_image)
68 plt.title('Harris Corner Detection')
69
70 plt.xticks([])
71 plt.yticks([])
72 plt.show()
```

5.1.2 Result of Harris Detection

Figure 5.1: Corner Detection

5.1.3 ANN

```
# Importing the libraries
import numpy as np
import pandas as pd
import tensorflow as tf

# Importing the dataset
dataset = pd.read_csv('Churn_Modelling.csv')
X = dataset.iloc[:, 3:13].values
y = dataset.iloc[:, 13].values
print(X)
print(y)

# Encoding categorical data
# Label Encoding the "Gender" column

```

```
from sklearn.preprocessing import LabelEncoder,OneHotEncoder
from sklearn.compose import ColumnTransformer
le = LabelEncoder()
X[:,1] = le.fit_transform(X[:, 1])#country
print(X)
X[:,2] = le.fit_transform(X[:, 2])#gender

ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder
    (), [1])], remainder='passthrough')# dummies for country
X = np.array(ct.fit_transform(X))
print(X)
# dummies for country

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size = 0.2, random_state = 0)

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
# Part 2 - Building the ANN

# Initializing the ANN
import keras
from keras.models import Sequential
from keras.layers import Dense

```

```
# Initializing the ANN

ann = Sequential()#?
num_columns = X.shape[1]

print("Number of columns:", num_columns)
# Adding the First hidden layer
ann.add(Dense(units=6, activation='relu'))# o/p dm = 6 as
    (11+1)/2 = 6 where input dm is 11
# Adding the second hidden layer
ann.add(Dense(units=6, activation='relu'))

# Adding the output layer
ann.add(Dense(units=1, activation='sigmoid'))

# Part 3 - Training the ANN

# Compiling the ANN
ann.compile(optimizer = 'adam', loss = 'binary_crossentropy',
    metrics = ['accuracy'])
# Training the ANN on the Training set
ann.fit(X_train, y_train, batch_size = 32, epochs = 100)

print(ann.predict(sc.transform([[1, 0, 0, 600, 1, 40, 3,
    60000, 2, 1, 1, 50000]])) > 0.5)

y_pred = ann.predict(X_test)
y_pred = (y_pred > 0.5)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.
    reshape(len(y_test),1)),1))
```

```
70
71 # Creating a DataFrame with predictions and actual labels
72 results_df = pd.DataFrame({'Predicted': y_pred.flatten(), '
      Actual': y_test})
73
74 # Exporting results to a CSV file
75 results_df.to_csv('prediction_results.csv', index=False)
76
77 # Making the Confusion Matrix
78 from sklearn.metrics import confusion_matrix, accuracy_score
79 cm = confusion_matrix(y_test, y_pred)
80 print(cm)
81 accuracy_score(y_test, y_pred)
82
83 ######### Visualization of loss and accuracy curve
      ####################
84 import matplotlib.pyplot as plt
85 # loss
86 plt.plot(hs.history['loss'], label='train loss')
87 plt.legend()
88 plt.xlabel('Epochs', fontsize=14, fontstyle='italic')
89 plt.ylabel('Loss', fontsize=14, fontstyle='italic')
90 plt.show()
91
92 # accuracies
93 plt.plot(hs.history['accuracy'], label='train acc')
94 plt.xlabel('Epochs', fontsize=14, fontstyle='italic')
95 plt.ylabel('Accuracy', fontsize=14, fontstyle='italic')
96 plt.legend()
97 plt.show()
```

5.1.4 Result of ANN

loss: 0.3342 - accuracy: 0.8627

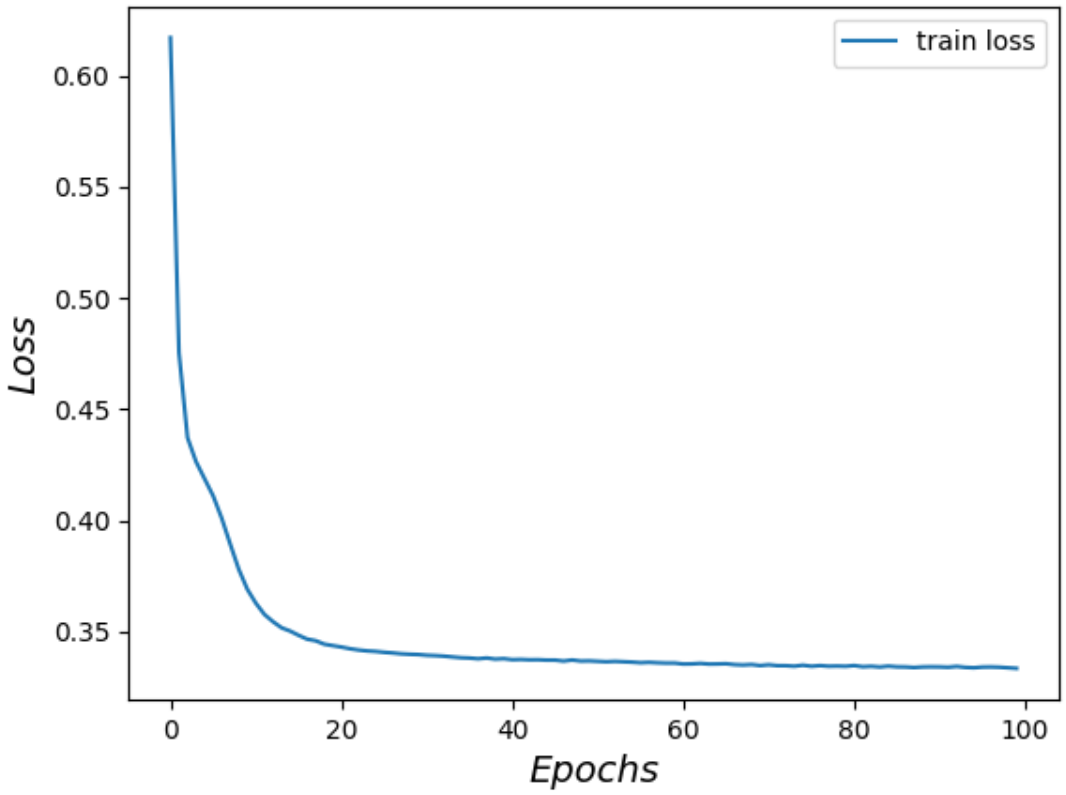

Figure 5.2: Loss Plot

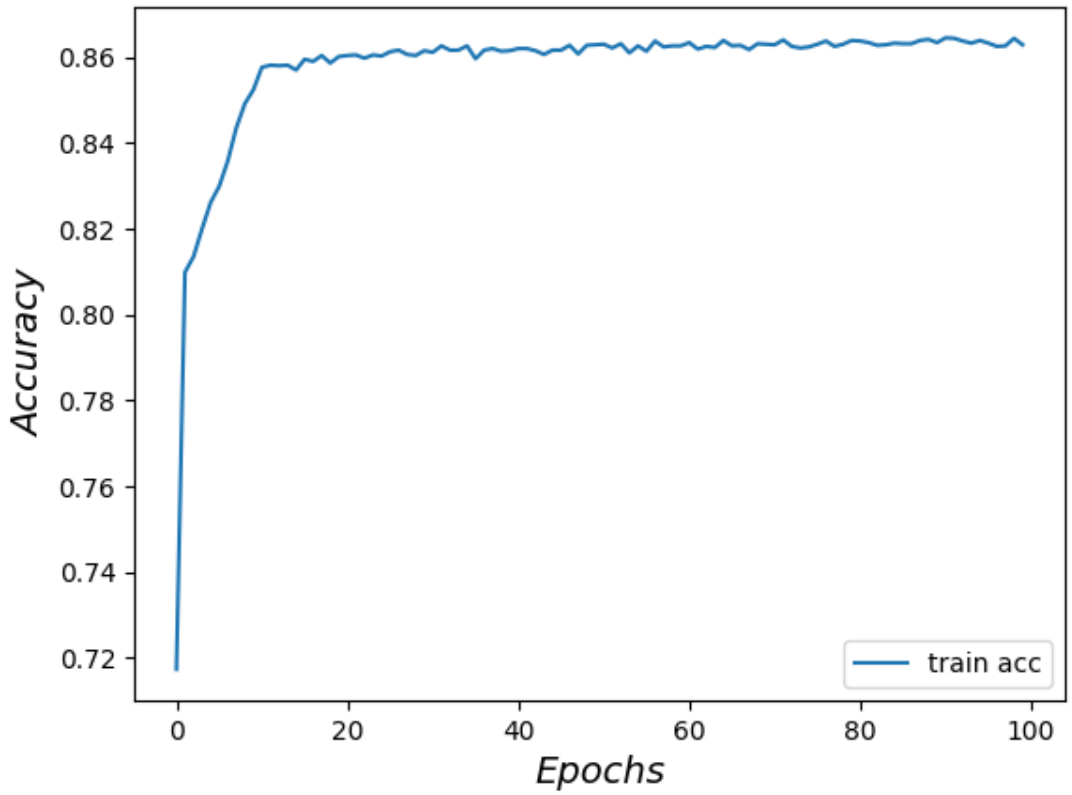

Figure 5.3: Accuracy Plot

5.1.5 CNN

```
from google.colab import drive
drive.mount('/content/drive')
import tensorflow as tf
from keras.preprocessing.image import ImageDataGenerator

#Training Set
train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True)
```

```
11 training_set = train_datagen.flow_from_directory('/content/
      drive/MyDrive/dataset/training_set',
12                                                  target_size =
       (64, 64),
13                                                  batch_size =
      32,
14                                                  class_mode =
      'binary')
15 #Test Set
16
17 test_datagen = ImageDataGenerator(rescale = 1./255)
18 test_set = test_datagen.flow_from_directory('/content/drive/
      MyDrive/dataset/test_set',
19                                             target_size = (64,
       64),
20                                             batch_size = 32,
21                                             class_mode = '
      binary')
22 cnn = tf.keras.models.Sequential()
23
24 cnn.add(tf.keras.layers.Conv2D(filters=32, kernel_size=3,
      activation='relu', input_shape=[64, 64, 3]))
25
26 cnn.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))
27
28 cnn.add(tf.keras.layers.Conv2D(filters=32, kernel_size=3,
      activation='relu'))
29 cnn.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))
30
31 # Flattening
```

```
cnn.add(tf.keras.layers.Flatten())

#full connection
# Adding the First hidden layer
from keras.layers import Dense
cnn.add(Dense(units=128, activation='relu'))

#output layer
cnn.add(Dense(units=1, activation='sigmoid'))

# Compiling the cnn
cnn.compile(optimizer = 'adam', loss = 'binary_crossentropy',
   metrics = ['accuracy'])
# Training the cnn on the Training set and  evaluating it on
   the Test set
cnn.fit(x = training_set, validation_data = test_set, epochs =
    25)


import numpy as np
from tensorflow.keras.preprocessing import image
import matplotlib.pyplot as plt

test_image1 = image.load_img('/content/drive/MyDrive/dog.jpg',
    target_size = (64, 64))
test_image = image.img_to_array(test_image1)
test_image = np.expand_dims(test_image, axis = 0)
result = cnn.predict(test_image)
training_set.class_indices
if result[0][0] == 1:
```

```
    prediction = 'dog'
else:
    prediction = 'cat'

plt.figure(figsize=(5, 5))
plt.subplot(1, 1, 1), plt.imshow(test_image1)
plt.title('Original Image'), plt.xticks([]), plt.yticks([])
plt.show()
print(prediction)
```

5.1.6 Result of CNN

accuracy: 0.8011

Figure 5.4: Input:Dog Image ; Output: Dog