**Experiment 1: Image Enhancement techniques**

# Program to mount the drive

from google.colab import drive

drive.mount('/content/drive')

Mounted at /content/drive

# Program to read and display color and gray scale image

import cv2

import numpy as np

from google.colab.patches import cv2\_imshow

im=cv2.imread('/content/Lena.jfif')

#im=cv2.imread('/content/drive/MyDrive/im\_files/rosepic12.jpg')

#new\_image = np.copy(im)

gray\_img = cv2.cvtColor(im, cv2.COLOR\_RGB2GRAY)

print(im.shape)

cv2\_imshow(im)

print(im.shape)

cv2\_imshow(gray\_img)

print(gray\_img.shape)

im1=np.copy(gray\_img)

(225, 225)



(225, 225, 3)



# Program to display image negative of given image

im2=255-im1

cv2\_imshow(im2)



#Program for gray level slicing with background

im11=np.zeros((im1.shape[0],im1.shape[1]),np.uint8)

m,n = im1.shape

min=132

max=180

for i in range(m):

for j in range(n):

if im1[i,j] >min and im1[i,j]<max:

im11[i,j]= 255

else:

im11[i,j] = im1[i,j]

cv2\_imshow(im11)



#Program for gray level slicing without background

im11=np.zeros((im1.shape[0],im1.shape[1]),np.uint8)

m,n = im1.shape

min=132

max=180

for i in range(m):

for j in range(n):

if im1[i,j] >min and im1[i,j]<max:

im11[i,j]= 0

else:

im11[i,j] = im1[i,j]

cv2\_imshow(im11)



#Program for image Tresholding

T=100

for i in range(m):

for j in range(n):

if im1[i,j] < T:

im11[i,j]= 0

else:

im11[i,j] = 255

cv2\_imshow(im11)



#Program for power law transformation(gamma Correction)

im12=np.zeros((im1.shape[0],im1.shape[1]),np.uint8)

im22=np.zeros((im1.shape[0],im1.shape[1]),np.uint8)

#m,n = im1.shape

gamma1=1.2

im12=np.power(im1,gamma1)

#cv2\_imshow(im1)

gamma2=0.8

im22=np.power(im1,gamma2)

#cv2\_imshow(im2)

hor\_stack=np.row\_stack((im12,im22))

cv2\_imshow(hor\_stack)



#Program for Log transformation

import math

import numpy as np

import cv2

L=255

d=np.zeros((225,225),np.uint8)

l1= math.log(L,10)

print(l1)

c=L/l1

print(c)

new=im1+1;

new1=np.log10(new)

d=c\*new1

cv2\_imshow(d)

2.4065401804339546

105.96124763394461



img = np.copy(im1)

lst = []

m,n = im1.shape

for i in range(m):

for j in range(n):

lst.append(np.binary\_repr(img[i][j] ,width=8)) # width = no. of bits

# We have a list of strings where each string represents binary pixel value.

#To extract bit planes we need to iterate over the strings and

#store the characters corresponding to bit planes into lists.

# Multiply with 2^(n-1) and reshape to reconstruct the bit image.

eight\_bit\_img = (np.array([int(i[0]) for i in lst],dtype = np.uint8) \* 128).reshape(img.shape[0],img.shape[1])

seven\_bit\_img = (np.array([int(i[1]) for i in lst],dtype = np.uint8) \* 64).reshape(img.shape[0],img.shape[1])

six\_bit\_img = (np.array([int(i[2]) for i in lst],dtype = np.uint8) \* 32).reshape(img.shape[0],img.shape[1])

five\_bit\_img = (np.array([int(i[3]) for i in lst],dtype = np.uint8) \* 16).reshape(img.shape[0],img.shape[1])

four\_bit\_img = (np.array([int(i[4]) for i in lst],dtype = np.uint8) \* 8).reshape(img.shape[0],img.shape[1])

three\_bit\_img = (np.array([int(i[5]) for i in lst],dtype = np.uint8) \* 4).reshape(img.shape[0],img.shape[1])

two\_bit\_img = (np.array([int(i[6]) for i in lst],dtype = np.uint8) \* 2).reshape(img.shape[0],img.shape[1])

one\_bit\_img = (np.array([int(i[7]) for i in lst],dtype = np.uint8) \* 1).reshape(img.shape[0],img.shape[1])

#Concatenate these images for ease of display using cv2.hconcat()

finalr = cv2.hconcat([eight\_bit\_img,seven\_bit\_img,six\_bit\_img,five\_bit\_img])

finalv =cv2.hconcat([four\_bit\_img,three\_bit\_img,two\_bit\_img,one\_bit\_img])

# Vertically concatenate

final = cv2.vconcat([finalr,finalv])

# Display the images

cv2\_imshow(final)



img = np.copy(im1)

lst = []

m,n = im1.shape

for i in range(m):

for j in range(n):

lst.append(np.binary\_repr(img[i][j] ,width=8)) # width = no. of bits

# We have a list of strings where each string represents binary pixel value.

#To extract bit planes we need to iterate over the strings and

#store the characters corresponding to bit planes into lists.

# Multiply with 2^(n-1) and reshape to reconstruct the bit image.

eight\_bit\_img = (np.array([int(i[0]) for i in lst],dtype = np.uint8) \* 128).reshape(img.shape[0],img.shape[1])

seven\_bit\_img = (np.array([int(i[1]) for i in lst],dtype = np.uint8) \* 64).reshape(img.shape[0],img.shape[1])

six\_bit\_img = (np.array([int(i[2]) for i in lst],dtype = np.uint8) \* 32).reshape(img.shape[0],img.shape[1])

five\_bit\_img = (np.array([int(i[3]) for i in lst],dtype = np.uint8) \* 16).reshape(img.shape[0],img.shape[1])

four\_bit\_img = (np.array([int(i[4]) for i in lst],dtype = np.uint8) \* 8).reshape(img.shape[0],img.shape[1])

three\_bit\_img = (np.array([int(i[5]) for i in lst],dtype = np.uint8) \* 4).reshape(img.shape[0],img.shape[1])

two\_bit\_img = (np.array([int(i[6]) for i in lst],dtype = np.uint8) \* 2).reshape(img.shape[0],img.shape[1])

one\_bit\_img = (np.array([int(i[7]) for i in lst],dtype = np.uint8) \* 1).reshape(img.shape[0],img.shape[1])

#Concatenate these images for ease of display using cv2.hconcat()

finalr = cv2.hconcat([eight\_bit\_img,seven\_bit\_img,six\_bit\_img,five\_bit\_img])

finalv =cv2.hconcat([four\_bit\_img,three\_bit\_img,two\_bit\_img,one\_bit\_img])

# Vertically concatenate

final = cv2.vconcat([finalr,finalv])

# Display the images

cv2\_imshow(final)

**Experiment 2: Histogram Equalization**

import cv2

import numpy as np

from google.colab.patches import cv2\_imshow

from matplotlib import pyplot as plt

im=np.zeros((200,200),np.uint8)

cv2.rectangle(im,(0,100),(200,200),(255),-1)

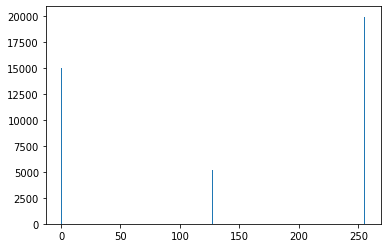
cv2.rectangle(im,(0,50),(100,100),(127),-1)

cv2\_imshow(im)

plt.hist(im.ravel(),256,[0,256])

plt.show()





import cv2

import numpy as np

from google.colab.patches import cv2\_imshow

from matplotlib import pyplot as plt

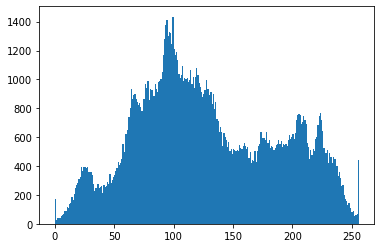
im=cv2.imread('/content/Lena.jfif')

cv2\_imshow(im)

plt.hist(im.ravel(),256,[0,256])

plt.show()





import cv2

import numpy as np

from google.colab.patches import cv2\_imshow

from matplotlib import pyplot as plt

im=cv2.imread('/content/Lena.jfif')

b,g,r=cv2.split(im)

cv2\_imshow(im)

cv2\_imshow(b)

cv2\_imshow(g)

cv2\_imshow(r)

plt.hist(im.ravel(),256,[0,256])

plt.hist(b.ravel(),256,[0,256])

plt.hist(g.ravel(),256,[0,256])

plt.hist(r.ravel(),256,[0,256])

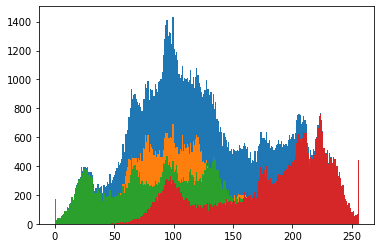
plt.show()











import cv2

import numpy as np

from matplotlib import pyplot as plt

from google.colab.patches import cv2\_imshow

#path = "/content/drive/MyDrive/content/lena.jpg"

img = cv2.imread('/content/Lena.jfif',0)

cv2\_imshow(img)

equ=np.zeros((img.shape[1],img.shape[0]),np.uint8)

equ = cv2.equalizeHist(img)

cv2\_imshow(equ)

hist,bins = np.histogram(img.flatten(),256,[0,256])

cdf = hist.cumsum()

cdf\_normalized = cdf \* float(hist.max()) / cdf.max()

plt.plot(cdf\_normalized, color = 'b')

plt.hist(img.flatten(),256,[0,256], color = 'r')

plt.xlim([0,256])

plt.legend(('cdf','histogram'), loc = 'upper left')

plt.show()

hist\_1,bins\_1 = np.histogram(equ.flatten(),256,[0,256])

cdf\_1 = hist\_1.cumsum()

#print (cdf)

cdf\_normalized\_1 = cdf\_1 \* float(hist\_1.max()) / cdf\_1.max()

#print(cdf\_normalized)

plt.plot(cdf\_normalized\_1, color = 'b')

plt.hist(equ.flatten(),256,[0,256], color = 'r')

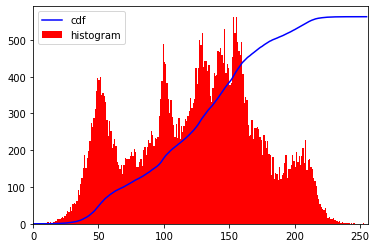
plt.xlim([0,256])

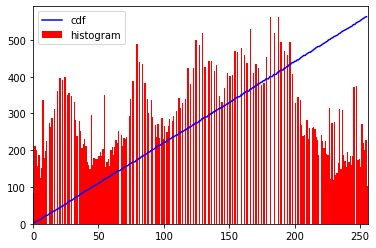
plt.legend(('cdf','histogram'), loc = 'upper left')

plt.show()









# import Opencv

import cv2

# import Numpy

import numpy as np

# read a image using imread

img = cv2.imread('/content/Lena.jfif', 0)

# creating a Histograms Equalization

# of a image using cv2.equalizeHist()

equ = cv2.equalizeHist(img)

# stacking images side-by-side

res = np.hstack((img, equ))

# show image input vs output

cv2\_imshow(res)



**Experiment 3:Spatial domain low pass and high pass filters**

**#Program for implementing Prewitt Filter**

**import cv2**

**import numpy as np**

**from google.colab.patches import cv2\_imshow**

**im=cv2.imread('/content/Lena.jfif',0)**

**p\_x=np.array([[1,1,1],[0,0,0],[-1,-1,-1]])**

**p\_y=np.array([[1,0,-1],[1,0,-1],[1,0,-1]])**

**pre\_im\_x=cv2.filter2D(im,-1,p\_x)**

**pre\_im\_y=cv2.filter2D(im,-1,p\_y)**

**#cv2\_imshow(pre\_im\_x)**

**#cv2\_imshow(pre\_im\_y)**

**#cv2\_imshow(pre\_im\_x+pre\_im\_y)**

**hor\_stack=np.column\_stack((pre\_im\_x,pre\_im\_y,pre\_im\_x+pre\_im\_y ))**

**cv2\_imshow(hor\_stack)**

****

**#Program for implementing Sobel Filter**

**import cv2**

**import numpy as np**

**from google.colab.patches import cv2\_imshow**

**im=cv2.imread('/content/Lena.jfif',0)**

**p\_x=np.array([[1, 2,1],[0,0,0],[-1,-2,-1]])**

**p\_y=np.array([[1,0,-1],[2,0,-2],[1,0,-1]])**

**pre\_im\_x=cv2.filter2D(im,-1,p\_x)**

**pre\_im\_y=cv2.filter2D(im,-1,p\_y)**

**#cv2\_imshow(pre\_im\_x)**

**#cv2\_imshow(pre\_im\_y)**

**#cv2\_imshow(pre\_im\_x+pre\_im\_y)**

**hor\_stack=np.column\_stack((pre\_im\_x,pre\_im\_y,pre\_im\_x+pre\_im\_y ))**

**cv2\_imshow(hor\_stack)**

****

**##Program for implementing Laplacian Filter**

**import cv2**

**import numpy as np**

**from google.colab.patches import cv2\_imshow**

**im=cv2.imread('/content/Lena.jfif',0)**

**p\_x=np.array([[-1,-1,-1],[-1,8,-1],[-1,-1,-1]])**

**#p\_y=np.array([[1,0,-1],[1,0,-1],[1,0,-1]])**

**pre\_im\_x=cv2.filter2D(im,-1,p\_x)**

**pre\_im\_y=cv2.filter2D(im,-1,p\_y)**

**cv2\_imshow(pre\_im\_x)**

**#cv2\_imshow(pre\_im\_y)**

**#cv2\_imshow(pre\_im\_x+pre\_im\_y)**

**#hor\_stack=np.column\_stack((pre\_im\_x,pre\_im\_y,pre\_im\_x+pre\_im\_y ))**

**#cv2\_imshow(hor\_stack)**

****

**#Program for implementing Averaging & Weighted averaging Filter**

**import cv2**

**import numpy as np**

**from google.colab.patches import cv2\_imshow**

**im=cv2.imread('/content/Lena.jfif',0)**

**print(im.shape)**

**p\_x=np.array([[1,1,1],[1,1,1],[1,1,1]])**

**pre\_im\_x=1/9\*p\_x**

**p\_y=np.array([[1,2,1],[2,4,2],[1,2,1]])**

**pre\_im\_x1=cv2.filter2D(im,-1,pre\_im\_x)**

**pre\_im\_y=1/16\*p\_y**

**pre\_im\_y1=cv2.filter2D(im,-1,pre\_im\_y)**

**cv2\_imshow(pre\_im\_y1)**

**cv2\_imshow(pre\_im\_y1)**

**hor\_stack=np.column\_stack((pre\_im\_x1,pre\_im\_y1 ))**

**cv2\_imshow(hor\_stack)**

**(225, 225)**

****

****

****

**##Program for enhancement of image implementing Laplacian Filter**

**import cv2**

**import numpy as np**

**from google.colab.patches import cv2\_imshow**

**im=cv2.imread('/content/Lena.jfif',0)**

**p\_x=np.array([[-1,-1,-1],[-1,9,-1],[-1,-1,-1]])**

**p\_y=np.array([[0,-1, 0],[-1, 5,-1],[0,-1, 0]])**

**pre\_im\_x=cv2.filter2D(im,-1,p\_x)**

**pre\_im\_y=cv2.filter2D(im,-1,p\_y)**

**cv2\_imshow(pre\_im\_x)**

**cv2\_imshow(pre\_im\_y)**

**hor\_stack=np.column\_stack((pre\_im\_x,pre\_im\_y ))**

**cv2\_imshow(hor\_stack)**

****

****

****

**Experiment 4: Image Morphing**

**# Display the given color and grayscale image**

**import cv2**

**from google.colab.patches import cv2\_imshow**

**import numpy as np**

**import matplotlib.pyplot as plt**

**im=cv2.imread('/content/Lenna\_(test\_image) (1).png')**

**lane\_image = np.copy(im)**

**gray = cv2.cvtColor(lane\_image, cv2.COLOR\_RGB2GRAY)**

**cv2\_imshow(im)**

**cv2\_imshow(gray)**

****

****

**# output image obtained after canny edge detection**

**canny\_img=cv2.Canny(gray,150,200)**

**cv2\_imshow(canny\_img)**

****

**# Output image after dilation**

**kernel=np.ones((3,3),np.uint8)**

**dilate\_img=cv2.dilate(canny\_img, kernel, iterations=1 )**

**print(kernel)**

**cv2\_imshow(dilate\_img)**

**[[1 1 1]**

**[1 1 1]**

**[1 1 1]]**

****

**# Output image after erosion**

**kernel=np.ones((3,3),np.uint8)**

**erode\_img=cv2.erode(dilate\_img, kernel, iterations=1 )**

**print(kernel)**

**cv2\_imshow(erode\_img)**

**[[1 1 1]**

**[1 1 1]**

**[1 1 1]]**

****

**# Output image after opening**

**open\_img=cv2.dilate(erode\_img,kernel,iterations=1)**

**cv2\_imshow(open\_img)**

****

**# Output image after closing**

**close\_img=cv2.erode(dilate\_img,kernel,iterations=1)**

**cv2\_imshow(close\_img)**

****

**# Output image after boudary extraction**

**boundary\_img= canny\_img - erode\_img**

**boundary\_img=boundary\_img\*255**

**cv2\_imshow(boundary\_img)**

****

**Experiment 5:Canny Edge Detection**

**#Program to run canny edge detector on a given image to find out the edges**

**import numpy as np**

**import os**

**import cv2**

**import matplotlib.pyplot as plt**

**from google.colab.patches import cv2\_imshow**

**# defining the canny detector function**

**# here weak\_th and strong\_th are thresholds for**

**# double thresholding step**

**def Canny\_detector(img, weak\_th = None, strong\_th = None):**

**# conversion of image to grayscale**

**img = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)**

**# Noise reduction step**

**img = cv2.GaussianBlur(img, (5, 5), 1.4)**

**# Calculating the gradients**

**gx = cv2.Sobel(np.float32(img), cv2.CV\_64F, 1, 0, 3)**

**gy = cv2.Sobel(np.float32(img), cv2.CV\_64F, 0, 1, 3)**

**# Conversion of Cartesian coordinates to polar**

**mag, ang = cv2.cartToPolar(gx, gy, angleInDegrees = True)**

**# setting the minimum and maximum thresholds**

**# for double thresholding**

**mag\_max = np.max(mag)**

**if not weak\_th:weak\_th = mag\_max \* 0.1**

**if not strong\_th:strong\_th = mag\_max \* 0.5**

**# getting the dimensions of the input image**

**height, width = img.shape**

**# Looping through every pixel of the grayscale**

**# image**

**for i\_x in range(width):**

**for i\_y in range(height):**

**grad\_ang = ang[i\_y, i\_x]**

**grad\_ang = abs(grad\_ang-180) if abs(grad\_ang)>180 else abs(grad\_ang)**

**# selecting the neighbours of the target pixel**

**# according to the gradient direction**

**# In the x axis direction**

**if grad\_ang<= 22.5:**

**neighb\_1\_x, neighb\_1\_y = i\_x-1, i\_y**

**neighb\_2\_x, neighb\_2\_y = i\_x + 1, i\_y**

**# top right (diagonal-1) direction**

**elif grad\_ang>22.5 and grad\_ang<=(22.5 + 45):**

**neighb\_1\_x, neighb\_1\_y = i\_x-1, i\_y-1**

**neighb\_2\_x, neighb\_2\_y = i\_x + 1, i\_y + 1**

**# In y-axis direction**

**elif grad\_ang>(22.5 + 45) and grad\_ang<=(22.5 + 90):**

**neighb\_1\_x, neighb\_1\_y = i\_x, i\_y-1**

**neighb\_2\_x, neighb\_2\_y = i\_x, i\_y + 1**

**# top left (diagonal-2) direction**

**elif grad\_ang>(22.5 + 90) and grad\_ang<=(22.5 + 135):**

**neighb\_1\_x, neighb\_1\_y = i\_x-1, i\_y + 1**

**neighb\_2\_x, neighb\_2\_y = i\_x + 1, i\_y-1**

**# Now it restarts the cycle**

**elif grad\_ang>(22.5 + 135) and grad\_ang<=(22.5 + 180):**

**neighb\_1\_x, neighb\_1\_y = i\_x-1, i\_y**

**neighb\_2\_x, neighb\_2\_y = i\_x + 1, i\_y**

**# Non-maximum suppression step**

**if width>neighb\_1\_x>= 0 and height>neighb\_1\_y>= 0:**

**if mag[i\_y, i\_x]<mag[neighb\_1\_y, neighb\_1\_x]:**

**mag[i\_y, i\_x]= 0**

**continue**

**if width>neighb\_2\_x>= 0 and height>neighb\_2\_y>= 0:**

**if mag[i\_y, i\_x]<mag[neighb\_2\_y, neighb\_2\_x]:**

**mag[i\_y, i\_x]= 0**

**weak\_ids = np.zeros\_like(img)**

**strong\_ids = np.zeros\_like(img)**

**ids = np.zeros\_like(img)**

**# double thresholding step**

**for i\_x in range(width):**

**for i\_y in range(height):**

**grad\_mag = mag[i\_y, i\_x]**

**if grad\_mag<weak\_th:**

**mag[i\_y, i\_x]= 0**

**elif strong\_th>grad\_mag>= weak\_th:**

**ids[i\_y, i\_x]= 1**

**else:**

**ids[i\_y, i\_x]= 2**

**# finally returning the magnitude of**

**# gradients of edges**

**return mag**

**frame = cv2.imread('/content/Lenna\_(test\_image) (1).png')**

**# calling the designed function for**

**# finding edges**

**canny\_img = Canny\_detector(frame)**

**# Displaying the input and output image**

**#plt.figure()**

**#f, plots = plt.subplots(2, 1)**

**#plots[0].imshow(frame)**

**#plots[1].imshow(canny\_img)**

**cv2\_imshow(frame)**

**cv2\_imshow(canny\_img)**

****

****

**Experiment 6:Fourier Spectrum of given image**

**#Program to plot the fourier spectrum of given image**

**import cv2**

**import numpy as np**

**from matplotlib import pyplot as plt**

**from google.colab.patches import cv2\_imshow**

**image = cv2.imread('/content/Lena img.jpg',0)**

**f = np.fft.fft2(image)**

**fshift = np.fft.fftshift(f)**

**magnitude\_spectrum = 20\*np.log(np.abs(fshift))**

**plt.subplot(121),plt.imshow(image, cmap = 'gray')**

**plt.title('Input Image'), plt.xticks([]), plt.yticks([])**

**plt.subplot(122),plt.imshow(magnitude\_spectrum, cmap = 'gray')**

**plt.title('Magnitude Spectrum'), plt.xticks([]), plt.yticks([])**

**plt.show()**

**#program to apply the ideal Low pass and High pass filters on the given image**

**image = cv2.resize(image,(200,200))**

**rows, cols = image.shape**

**crow,ccol = rows/2 , cols/2**

**print (image.shape)**

**dft=cv2.dft(np.float32(image),flags=cv2.DFT\_COMPLEX\_OUTPUT)**

**dft\_shift=np.fft.fftshift(dft)**

**magnitude\_spectrum=20\*np.log(cv2.magnitude(dft\_shift[:,:,0],dft\_shift[:,:,1]))**

**print(crow)**

**print(ccol)**

**r=80**

**mask = np.zeros((rows, cols,2), dtype=np.float32)**

**mask1 = np.ones((rows, cols,2), dtype=np.float32)**

**center=[crow,ccol]**

**x,y=np.ogrid[:rows,:cols]**

**mask\_area=(x-center[0])\*\*2+(y-center[1])\*\*2<=r\*r**

**mask[mask\_area]=1**

**mask1[mask\_area]=0;**

**fshift = dft\_shift \* mask**

**fshift1=dft\_shift\*mask1**

**fshift\_mask\_mag=20\*np.log(cv2.magnitude(fshift[:,:,0],fshift[:,:,1]))**

**f\_shift=np.fft.ifftshift(fshift)**

**img\_back=cv2.idft(f\_shift)**

**img\_back=cv2.magnitude(img\_back[:,:,0],img\_back[:,:,1])**

**fshift\_mask\_mag1=20\*np.log(cv2.magnitude(fshift1[:,:,0],fshift1[:,:,1]))**

**f\_shift1=np.fft.ifftshift(fshift1)**

**img\_back1=cv2.idft(f\_shift1)**

**img\_back1=cv2.magnitude(img\_back1[:,:,0],img\_back1[:,:,1])**

**plt.subplot(441),plt.imshow(image, cmap = 'gray')**

**plt.title('Input Image'), plt.xticks([]), plt.yticks([])**

**plt.subplot(442),plt.imshow(magnitude\_spectrum, cmap = 'gray')**

**plt.title('Image after HPF'), plt.xticks([]), plt.yticks([])**

**plt.subplot(443),plt.imshow(fshift\_mask\_mag, cmap = 'gray')**

**plt.title('Result after filtering'), plt.xticks([]), plt.yticks([])**

**plt.subplot(444),plt.imshow(img\_back, cmap = 'gray')**

**plt.subplot(445),plt.imshow(image, cmap = 'gray')**

**plt.title('Input Image'), plt.xticks([]), plt.yticks([])**

**plt.subplot(446),plt.imshow(magnitude\_spectrum, cmap = 'gray')**

**plt.title('Image after LPF'), plt.xticks([]), plt.yticks([])**

**plt.subplot(447),plt.imshow(fshift\_mask\_mag1, cmap = 'gray')**

**plt.title('Result after filtering'), plt.xticks([]), plt.yticks([])**

**plt.subplot(448),plt.imshow(img\_back1, cmap = 'gray')**

**#program to apply Butterworth Low pass filter on the given image**

**image = cv2.resize(image,(200,200))**

**rows, cols = image.shape**

**crow,ccol = rows/2 , cols/2**

**print (image.shape)**

**dft=cv2.dft(np.float32(image),flags=cv2.DFT\_COMPLEX\_OUTPUT)**

**dft\_shift=np.fft.fftshift(dft)**

**magnitude\_spectrum=20\*np.log(cv2.magnitude(dft\_shift[:,:,0],dft\_shift[:,:,1]))**

**print(crow)**

**print(ccol)**

**r=40**

**hh\_mask = np.zeros((rows, cols,2), dtype=np.float32)**

**center=[crow,ccol]**

**x,y=np.ogrid[:rows,:cols]**

**for i in range(image.shape[0]):**

**for j in range(image.shape[1]):**

**mask\_area=(i-center[0])\*\*2+(j-center[1])\*\*2**

**a=mask\_area/r**

**a1=pow(a,2)**

**hh\_mask[i,j] =1/(1+a1)**

**fshift = dft\_shift \* hh\_mask**

**fshift\_mask\_mag=20\*np.log(cv2.magnitude(fshift[:,:,0],fshift[:,:,1]))**

**f\_shift=np.fft.ifftshift(fshift)**

**img\_back=cv2.idft(f\_shift)**

**img\_back=cv2.magnitude(img\_back[:,:,0],img\_back[:,:,1])**

**hh\_mask1=1-hh\_mask**

**plt.subplot(221),plt.imshow(image, cmap = 'gray')**

**plt.title('Input Image'), plt.xticks([]), plt.yticks([])**

**plt.subplot(222),plt.imshow(magnitude\_spectrum, cmap = 'gray')**

**plt.title('Image after LPF'), plt.xticks([]), plt.yticks([])**

**plt.subplot(223),plt.imshow(fshift\_mask\_mag, cmap = 'gray')**

**plt.title('Result after filtering'), plt.xticks([]), plt.yticks([])**

**plt.subplot(224),plt.imshow(img\_back, cmap = 'gray')**

**#program to apply Butterworth High pass filters on the given image**

**fshift1 = dft\_shift \* hh\_mask1**

**fshift\_mask\_mag1=20\*np.log(cv2.magnitude(fshift1[:,:,0],fshift1[:,:,1]))**

**f\_shift1=np.fft.ifftshift(fshift1)**

**img\_back1=cv2.idft(f\_shift1)**

**img\_back1=cv2.magnitude(img\_back1[:,:,0],img\_back1[:,:,1])**

**hh\_mask1=1-hh\_mask**

**plt.subplot(221),plt.imshow(image, cmap = 'gray')**

**plt.title('Input Image'), plt.xticks([]), plt.yticks([])**

**plt.subplot(222),plt.imshow(magnitude\_spectrum, cmap = 'gray')**

**plt.title('Image after HPF'), plt.xticks([]), plt.yticks([])**

**plt.subplot(223),plt.imshow(fshift\_mask\_mag1, cmap = 'gray')**

**plt.title('Result after filtering'), plt.xticks([]), plt.yticks([])**

**plt.subplot(224),plt.imshow(img\_back1, cmap = 'gray')**

**#program to apply Gaussian Low pass filter on the given image**

**import math**

**image = cv2.resize(image,(200,200))**

**rows, cols = image.shape**

**crow,ccol = rows/2 , cols/2**

**print (image.shape)**

**dft=cv2.dft(np.float32(image),flags=cv2.DFT\_COMPLEX\_OUTPUT)**

**dft\_shift=np.fft.fftshift(dft)**

**magnitude\_spectrum=20\*np.log(cv2.magnitude(dft\_shift[:,:,0],dft\_shift[:,:,1]))**

**print(crow)**

**print(ccol)**

**r=60**

**gg\_mask = np.zeros((rows, cols,2), dtype=np.float32)**

**gg\_mask1 = np.zeros((rows, cols,2), dtype=np.float32)**

**center=[crow,ccol]**

**x,y=np.ogrid[:rows,:cols]**

**for i in range(image.shape[0]):**

**for j in range(image.shape[1]):**

**mask\_area=(i-center[0])\*\*2+(j-center[1])\*\*2**

**a=2\*r\*r**

**a2=mask\_area/a;**

**a1=math.exp(a2)**

**gg\_mask[i,j] =a1**

**fshift\_g = dft\_shift \* gg\_mask**

**fshift\_mask\_mag\_g=20\*np.log(cv2.magnitude(fshift\_g[:,:,0],fshift\_g[:,:,1]))**

**f\_shift\_g=np.fft.ifftshift(fshift\_g)**

**img\_back\_g=cv2.idft(f\_shift\_g)**

**img\_back\_g=cv2.magnitude(img\_back\_g[:,:,0],img\_back\_g[:,:,1])**

**plt.subplot(221),plt.imshow(image, cmap = 'gray')**

**plt.title('Input Image'), plt.xticks([]), plt.yticks([])**

**plt.subplot(222),plt.imshow(magnitude\_spectrum, cmap = 'gray')**

**plt.title('Image after LPF'), plt.xticks([]), plt.yticks([])**

**plt.subplot(223),plt.imshow(fshift\_mask\_mag\_g, cmap = 'gray')**

**plt.title('Result after filtering'), plt.xticks([]), plt.yticks([])**

**plt.subplot(224),plt.imshow(img\_back\_g, cmap = 'gray')**

**#program to apply Gaussian High pass filter on the given image**

**gg\_mask1=1-gg\_mask**

**fshift\_g1 = dft\_shift \* gg\_mask1**

**fshift\_mask\_mag\_g1=20\*np.log(cv2.magnitude(fshift\_g1[:,:,0],fshift\_g1[:,:,1]))**

**f\_shift\_g1=np.fft.ifftshift(fshift\_g1)**

**img\_back\_g1=cv2.idft(f\_shift\_g1)**

**img\_back\_g1=cv2.magnitude(img\_back\_g1[:,:,0],img\_back\_g1[:,:,1])**

**plt.subplot(221),plt.imshow(image, cmap = 'gray')**

**plt.title('Input Image'), plt.xticks([]), plt.yticks([])**

**plt.subplot(222),plt.imshow(magnitude\_spectrum, cmap = 'gray')**

**plt.title('Image after HPF'), plt.xticks([]), plt.yticks([])**

**plt.subplot(223),plt.imshow(fshift\_mask\_mag\_g1, cmap = 'gray')**

**plt.title('Result in Filtering'), plt.xticks([]), plt.yticks([])**

**plt.subplot(224),plt.imshow(img\_back\_g1, cmap = 'gray')**

****

**(200, 200)**

**100.0**

**100.0**

**/usr/local/lib/python3.7/dist-packages/ipykernel\_launcher.py:44: RuntimeWarning: divide by zero encountered in log**

**/usr/local/lib/python3.7/dist-packages/ipykernel\_launcher.py:49: RuntimeWarning: divide by zero encountered in log**

**(200, 200)**

**100.0**

**100.0**

**(200, 200)**

**100.0**

**100.0**

**/usr/local/lib/python3.7/dist-packages/ipykernel\_launcher.py:112: RuntimeWarning: divide by zero encountered in log**

**/usr/local/lib/python3.7/dist-packages/ipykernel\_launcher.py:118: MatplotlibDeprecationWarning: Adding an axes using the same arguments as a previous axes currently reuses the earlier instance. In a future version, a new instance will always be created and returned. Meanwhile, this warning can be suppressed, and the future behavior ensured, by passing a unique label to each axes instance.**

**/usr/local/lib/python3.7/dist-packages/ipykernel\_launcher.py:120: MatplotlibDeprecationWarning: Adding an axes using the same arguments as a previous axes currently reuses the earlier instance. In a future version, a new instance will always be created and returned. Meanwhile, this warning can be suppressed, and the future behavior ensured, by passing a unique label to each axes instance.**

**/usr/local/lib/python3.7/dist-packages/ipykernel\_launcher.py:122: MatplotlibDeprecationWarning: Adding an axes using the same arguments as a previous axes currently reuses the earlier instance. In a future version, a new instance will always be created and returned. Meanwhile, this warning can be suppressed, and the future behavior ensured, by passing a unique label to each axes instance.**

**/usr/local/lib/python3.7/dist-packages/ipykernel\_launcher.py:124: MatplotlibDeprecationWarning: Adding an axes using the same arguments as a previous axes currently reuses the earlier instance. In a future version, a new instance will always be created and returned. Meanwhile, this warning can be suppressed, and the future behavior ensured, by passing a unique label to each axes instance.**

**/usr/local/lib/python3.7/dist-packages/ipykernel\_launcher.py:159: MatplotlibDeprecationWarning: Adding an axes using the same arguments as a previous axes currently reuses the earlier instance. In a future version, a new instance will always be created and returned. Meanwhile, this warning can be suppressed, and the future behavior ensured, by passing a unique label to each axes instance.**

**/usr/local/lib/python3.7/dist-packages/ipykernel\_launcher.py:161: MatplotlibDeprecationWarning: Adding an axes using the same arguments as a previous axes currently reuses the earlier instance. In a future version, a new instance will always be created and returned. Meanwhile, this warning can be suppressed, and the future behavior ensured, by passing a unique label to each axes instance.**

**/usr/local/lib/python3.7/dist-packages/ipykernel\_launcher.py:163: MatplotlibDeprecationWarning: Adding an axes using the same arguments as a previous axes currently reuses the earlier instance. In a future version, a new instance will always be created and returned. Meanwhile, this warning can be suppressed, and the future behavior ensured, by passing a unique label to each axes instance.**

**/usr/local/lib/python3.7/dist-packages/ipykernel\_launcher.py:165: MatplotlibDeprecationWarning: Adding an axes using the same arguments as a previous axes currently reuses the earlier instance. In a future version, a new instance will always be created and returned. Meanwhile, this warning can be suppressed, and the future behavior ensured, by passing a unique label to each axes instance.**

**/usr/local/lib/python3.7/dist-packages/ipykernel\_launcher.py:171: RuntimeWarning: divide by zero encountered in log**

**/usr/local/lib/python3.7/dist-packages/ipykernel\_launcher.py:176: MatplotlibDeprecationWarning: Adding an axes using the same arguments as a previous axes currently reuses the earlier instance. In a future version, a new instance will always be created and returned. Meanwhile, this warning can be suppressed, and the future behavior ensured, by passing a unique label to each axes instance.**

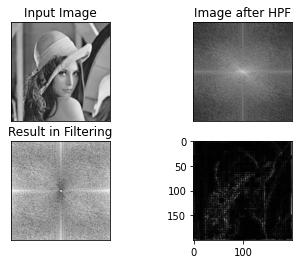
**/usr/local/lib/python3.7/dist-packages/ipykernel\_launcher.py:178: MatplotlibDeprecationWarning: Adding an axes using the same arguments as a previous axes currently reuses the earlier instance. In a future version, a new instance will always be created and returned. Meanwhile, this warning can be suppressed, and the future behavior ensured, by passing a unique label to each axes instance.**

**/usr/local/lib/python3.7/dist-packages/ipykernel\_launcher.py:180: MatplotlibDeprecationWarning: Adding an axes using the same arguments as a previous axes currently reuses the earlier instance. In a future version, a new instance will always be created and returned. Meanwhile, this warning can be suppressed, and the future behavior ensured, by passing a unique label to each axes instance.**

**/usr/local/lib/python3.7/dist-packages/ipykernel\_launcher.py:182: MatplotlibDeprecationWarning: Adding an axes using the same arguments as a previous axes currently reuses the earlier instance. In a future version, a new instance will always be created and returned. Meanwhile, this warning can be suppressed, and the future behavior ensured, by passing a unique label to each axes instance.**

**(<matplotlib.axes.\_subplots.AxesSubplot at 0x7fe08d676410>,**

**<matplotlib.image.AxesImage at 0x7fe08d5e5a10>)**

****