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])
```

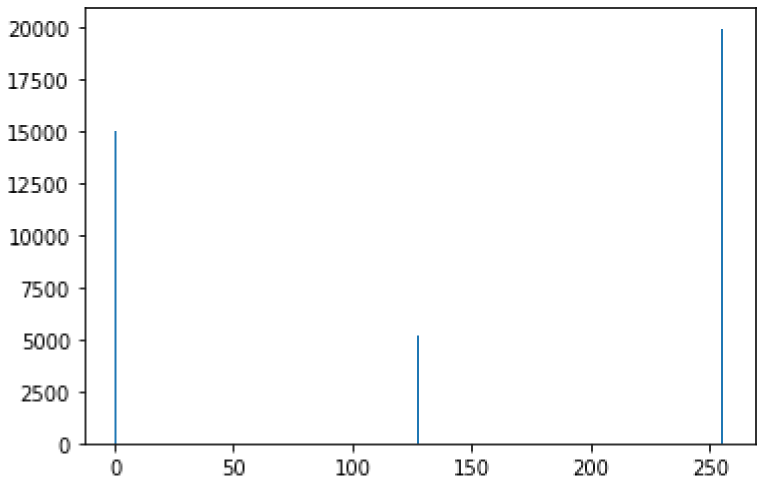
```
two_bit_img = (np.array([int(i[6]) for i in lst],dtype = np.uint8) *
2).reshape(img.shape[0],img.shape[1])
one_bit_img = (np.array([int(i[7]) for i in lst],dtype = np.uint8) *
1).reshape(img.shape[0],img.shape[1])

#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
```

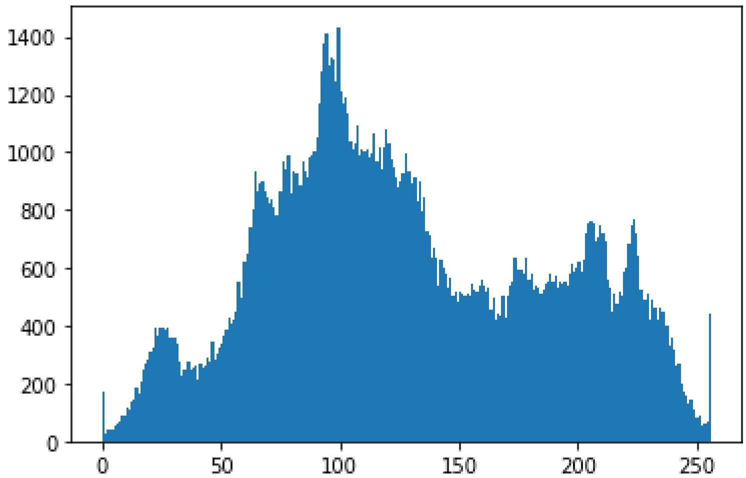
```
# We have a list of strings where each string represents binary
pixel value.
#To extract bit planes we need to iterate over the strings and
#store the characters corresponding to bit planes into lists.
# Multiply with 2^(n-1) and reshape to reconstruct the bit image.
eight_bit_img = (np.array([int(i[0]) 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

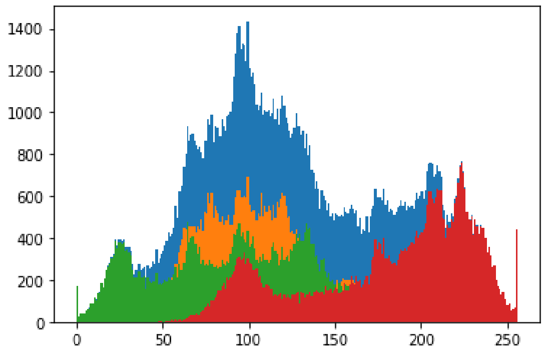
```
import cv2
import numpy as np
from google.colab.patches import cv2_imshow
from matplotlib import pyplot as plt
im=np.zeros((200,200),np.uint8)
cv2.rectangle(im,(0,100),(200,200),(255),-1)
cv2.rectangle(im,(0,50),(100,100),(127),-1)
cv2_imshow(im)
plt.hist(im.ravel(),256,[0,256])
plt.show()
```



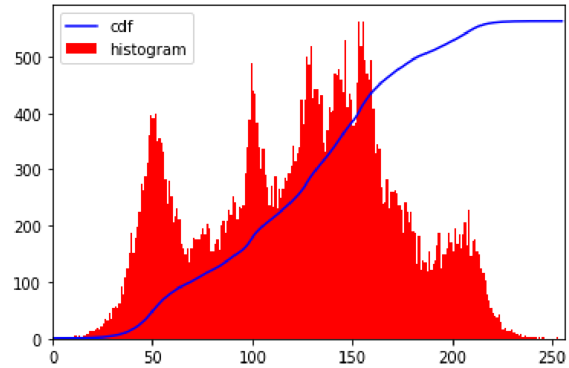
```
import cv2
import numpy as np
from google.colab.patches import cv2_imshow
from matplotlib import pyplot as plt
im=cv2.imread('/content/Lena.jfif')
cv2_imshow(im)
plt.hist(im.ravel(),256,[0,256])
plt.show()
```

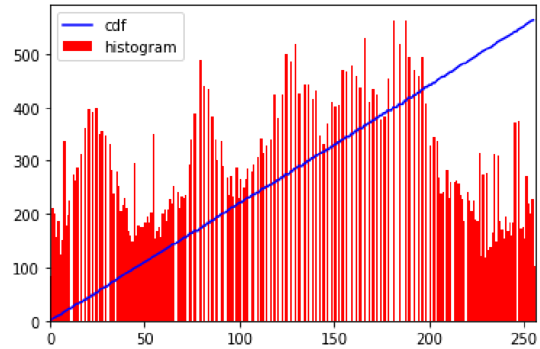


```
import cv2
import numpy as np
from google.colab.patches import cv2_imshow
from matplotlib import pyplot as plt
im=cv2.imread('/content/Lena.jfif')
b,g,r=cv2.split(im)
cv2_imshow(im)
cv2_imshow(b)
cv2_imshow(g)
cv2_imshow(r)
plt.hist(im.ravel(),256,[0,256])
plt.hist(b.ravel(),256,[0,256])
plt.hist(g.ravel(),256,[0,256])
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()
```


cdf
histogram
0
100
200
300
400
500
0
50
100
150
200
250


cdf
histogram
0
100
200
300
400
500
0
50
100
150
200
250

```
# 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)

```python
##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
```

```python
            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)
```

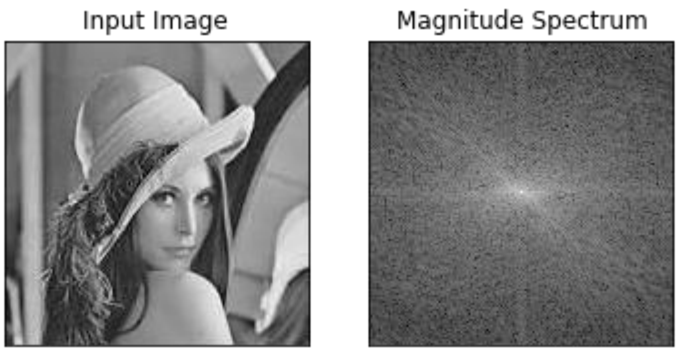
Experiment 6:Fourier Spectrum of given image

```
#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_shif
t[:,:,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_shif
t[:,:,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_shif
t[:,:,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_g
1[:,:,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>)