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]))
```

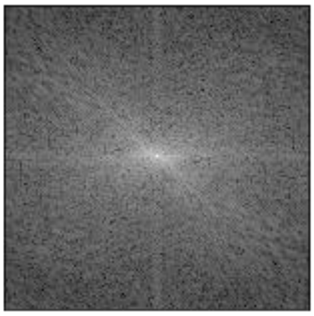
```
f_shift_g=np.fft.ifftshift(fshift_g)
img_back_g=cv2.idft(f_shift_g)
img_back_g=cv2.magnitude(img_back_g[:,:,0],img_back_g[:,:,1])

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>)