International Institute of Information Technology
Bangalore

DIGITAL SIGNAL PROCESSING

PROJECT REPORT

Achintya Harsha

Akash Perla

Yogesh Goyal

May 23, 2023

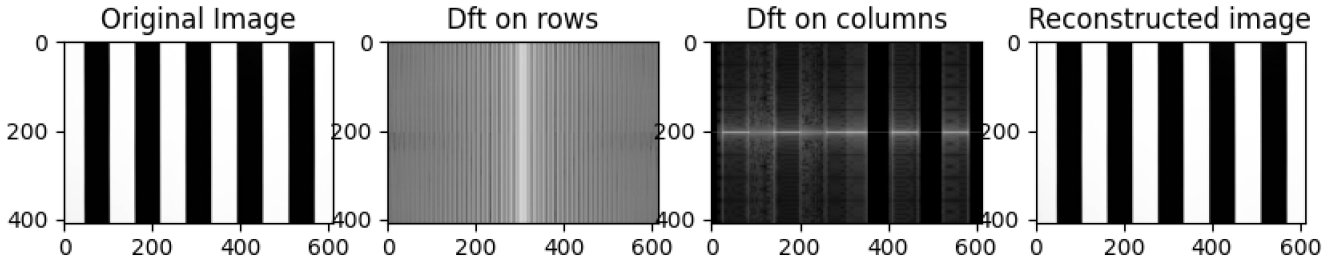
Question 1

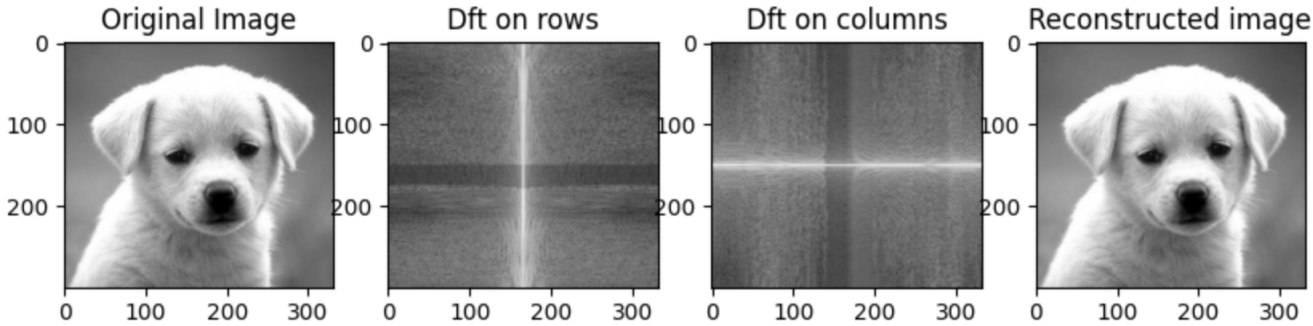
. 1-D DFT on an image.

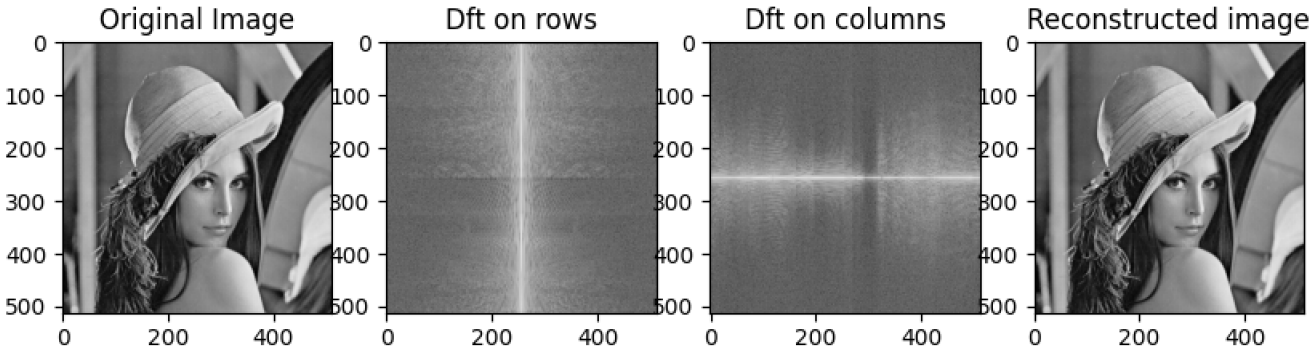
We have to seperately perform DFT on rows and columns.

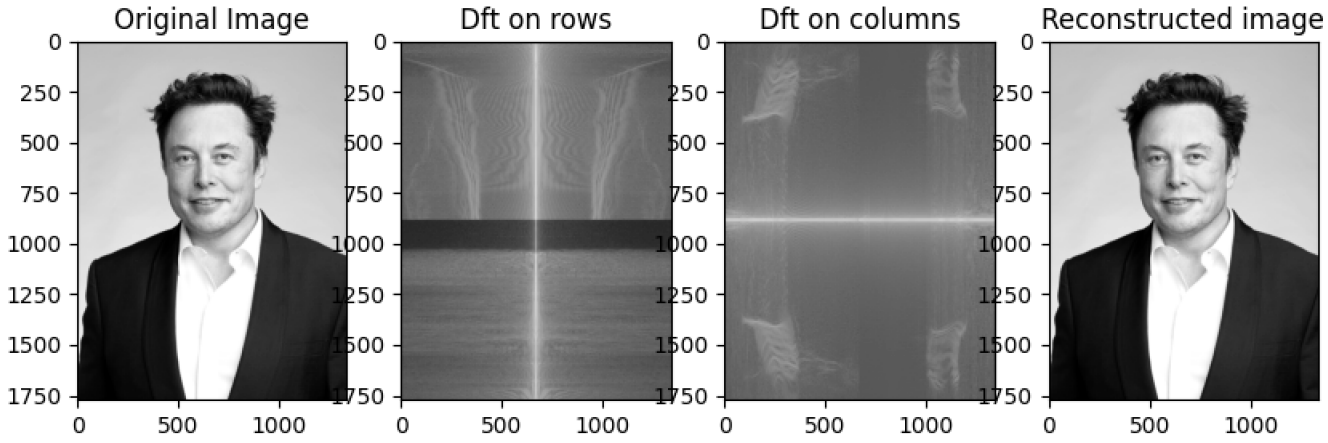
Observations/Graphs :-

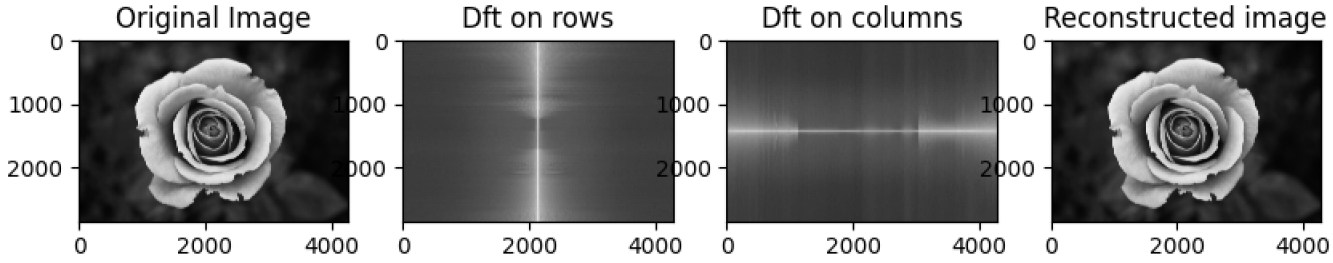
Below are the obtained results.











Python code:-

The below mentioned code is just for one image.

```
1
2  import numpy as np
3  import cv2
4  from matplotlib import pyplot as plt
5
6  # Read the image and convert it to grayscale
7  img1 = cv2.imread('Elon.jpg')
8  img1_gray = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
9
10 # Compute 1D DFT of rows
11 img1_dft_rows = np.fft.fft(img1_gray, axis=1)
12 img1_dft_rows_shift = np.fft.fftshift(img1_dft_rows)
13
14 # Compute 1D DFT of columns
15 img1_dft_cols = np.fft.fft(img1_gray, axis=0)
16 img1_dft_cols_shift = np.fft.fftshift(img1_dft_cols)
17
18 fig, axs = plt.subplots(1, 4, figsize=(10, 5))
19 axs[0].imshow(img1_gray, cmap='gray')
```

```
20 axs[0].set_title('Original Image')
21
22 # Visualize the magnitude spectrum of the DFT of a row
23 axs[1].imshow(np.log10(1+np.abs(img1_dft_rows_shift)), cmap='gray')
24 axs[1].set_title('Dft on rows')
25
26 # Visualize the magnitude spectrum of the DFT of a column
27 axs[2].imshow(np.log10(1+np.abs(img1_dft_cols_shift)), cmap='gray')
28 axs[2].set_title('Dft on columns')
29
30
31 img1_idft_rows=np.fft.ifft(img1_dft_rows,axis=1)
32 img1_idft_cols=np.fft.ifft(img1_dft_cols,axis=0)
33
34 axs[3].imshow(np.real(img1_idft_rows+img1_idft_cols), cmap='gray')
35 axs[3].set_title('Reconstructed image')
36 plt.show()
```

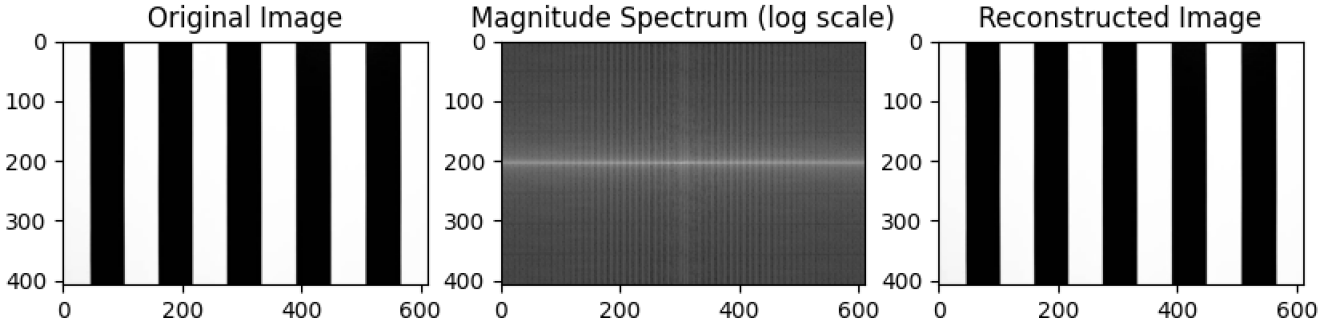
Question 2

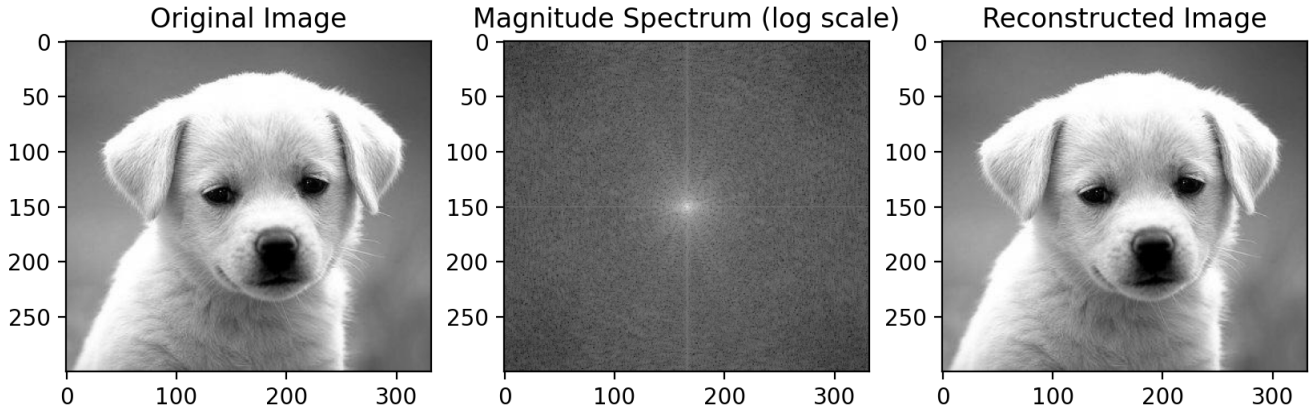
. 2-D DFT on an image

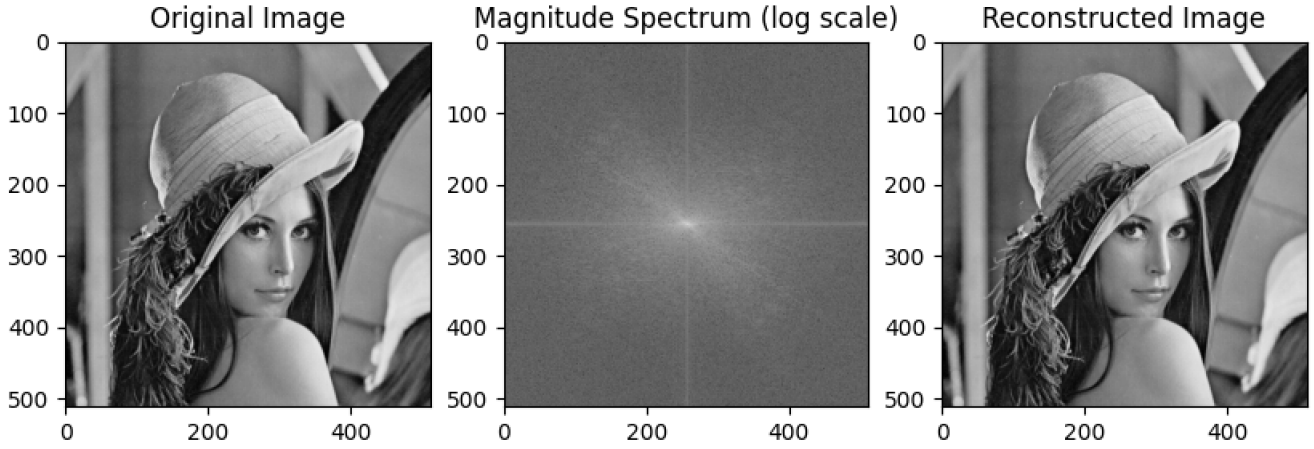
Observations/Graphs :-

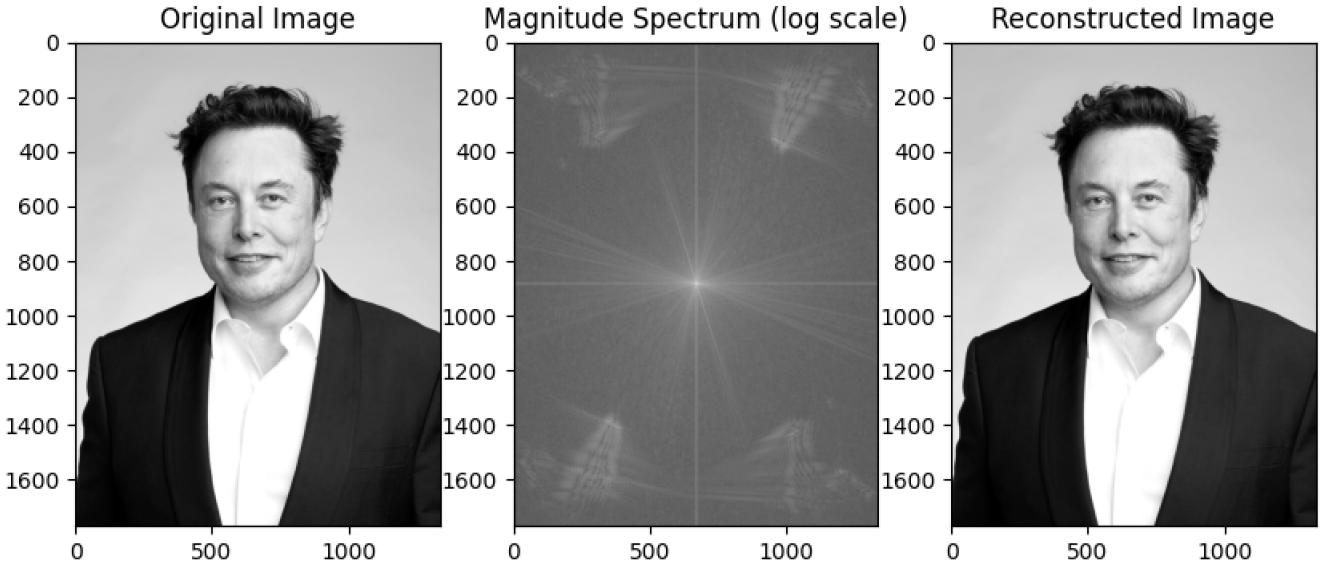
Below are the obtained results.

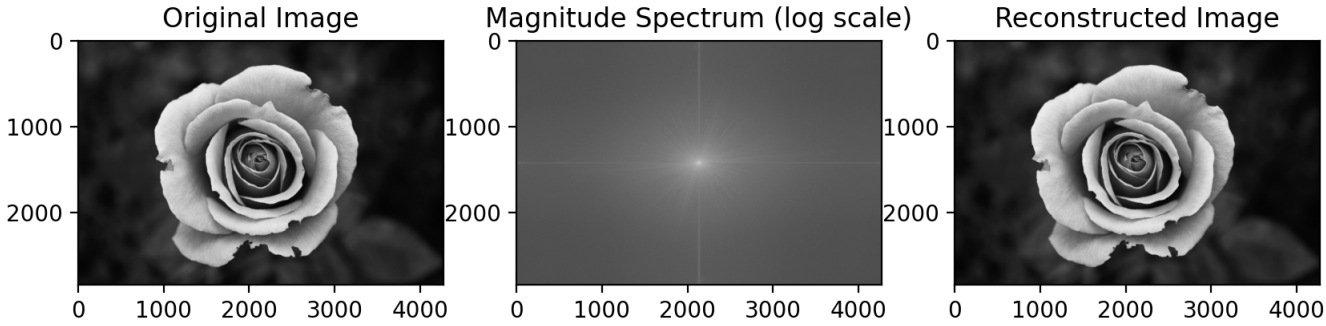
The x-axis represents the horizontal frequencies or spatial frequencies along the columns of the image.
The y-axis represents the vertical frequencies or spatial frequencies along the rows of the image.






Original Image
Magnitude Spectrum (log scale)
Reconstructed Image


Original Image
Magnitude Spectrum (log scale)
Reconstructed Image


Original Image
Magnitude Spectrum (log scale)
Reconstructed Image

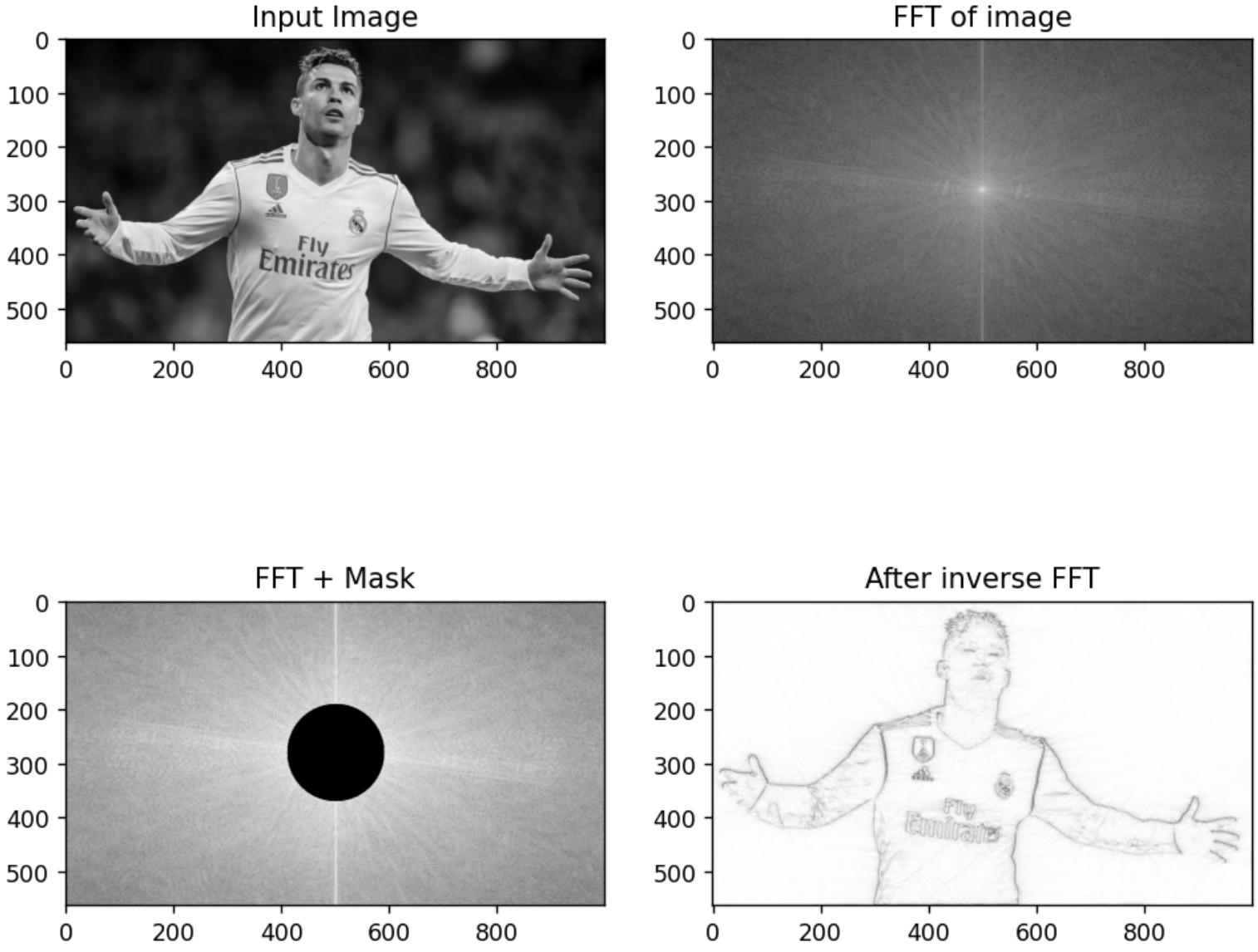
Python code:-

The below mentioned code is just for one image.

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

img1 = cv2.imread('Elon.jpg')
img1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)

# Apply 2D DFT
img1_dft = np.fft.fft2(img1)
img1_dft_shift = np.fft.fftshift(img1_dft)

# Calculate the magnitude spectrum
img1_mag = np.abs(img1_dft_shift)

# Display the original image and the magnitude spectrum
fig, axs = plt.subplots(1, 3, figsize=(10, 5))
axs[0].imshow(img1, cmap='gray')
axs[0].set_title('Original Image')

axs[1].imshow(np.log10(1+img1_mag), cmap='gray')
axs[1].set_title('Magnitude Spectrum (log scale)')
# plt.show()

# Apply inverse 2D DFT
img1_idft= np.fft.ifft2(img1_dft)

# Take the real part of the reconstructed image
img1_idft= np.real(img1_idft)

# Display the original image and the reconstructed image
axs[2].imshow((img1_idft), cmap='gray')
axs[2].set_title('Reconstructed Image')
plt.show()
```

Note:

- The low frequency regions in the FFT correspond to the monotonous areas in the picture.
- The high frequency regions in the FFT attributes the edges in the picture.
- So we can do edge detection of an image before removing the low frequency component of the image by passing the FFT through a high pass filter.



Edge Detection by removing low frequency components

Python code:-

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


img = cv2.imread('ron.jpg', 0)

dft = cv2.dft(np.float32(img), flags=cv2.DFT_COMPLEX_OUTPUT)
dft_shift = np.fft.fftshift(dft)

magnitude_spectrum = 20 * np.log(cv2.magnitude(
                                  dft_shift[:, :, 0],
                                  dft_shift[:, :, 1]) + 1)

rows, cols = img.shape
crow, ccol = int(rows / 2), int(cols / 2)

mask = np.ones((rows, cols, 2), np.uint8)
r = 90
center = [crow, ccol]
x, y = np.ogrid[:rows, :cols]
mask_area = (x - center[0]) ** 2 + (y - center[1]) ** 2 <= r*r
mask[mask_area] = 0


fshift = dft_shift * mask

fshift_mask_mag = 2000 * np.log(cv2.magnitude(fshift[:, :, 0],
                                  fshift[:, :, 1]) + 1)

f_ishift = np.fft.ifftshift(fshift)
img_back = cv2.idft(f_ishift)
img_back = cv2.magnitude(img_back[:, :, 0], img_back[:, :, 1])

m = -1
for i in range(len(img_back)):
    x = img_back[i]
    for j in  range(len(x)):
        m = max(img_back[i][j], m)

img_back = m - img_back


fig = plt.figure(figsize=(12, 12))
ax1 = fig.add_subplot(2,2,1)
ax1.imshow(img, cmap='gray')
ax1.title.set_text('Input Image')
ax2 = fig.add_subplot(2,2,2)
```

```
49 ax2.imshow(magnitude_spectrum, cmap='gray')
50 ax2.title.set_text('FFT of image')
51 ax3 = fig.add_subplot(2,2,3)
52 ax3.imshow(fshift_mask_mag, cmap='gray')
53 ax3.title.set_text('FFT + Mask')
54 ax4 = fig.add_subplot(2,2,4)
55 ax4.imshow(img_back, cmap='gray')
56 ax4.title.set_text('After inverse FFT')
57 plt.show()
```

Question 3:

Gaussian distribution :-

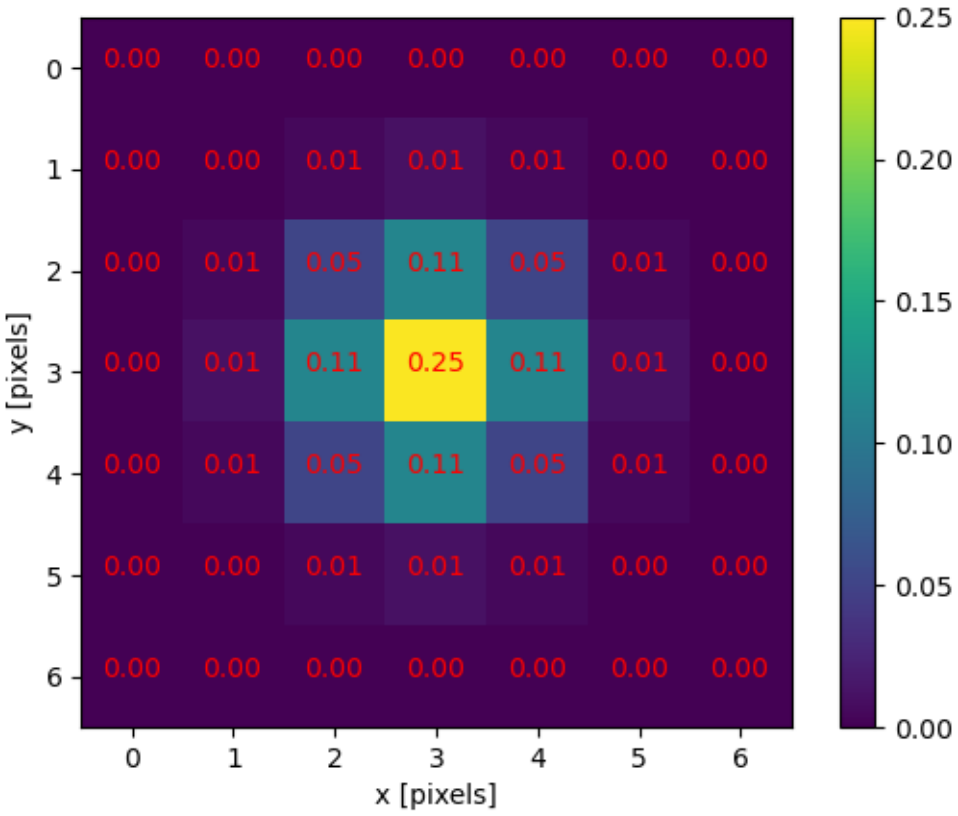
The Gaussian distribution (also known as the normal distribution) is a bell-shaped curve with an equal number of observations above and below the mean value.
1-D Gaussian Distribution function,

$$f(x) = \frac{1}{\sqrt{2\pi\sigma}} \cdot e^{\frac{-x^2}{2\sigma^2}}$$

2-D Gaussian Distribution function,

$$f(x,y) = \frac{1}{2\pi\sigma_X\sigma_Y\sqrt{1-\rho^2}} \exp\left(-\frac{1}{2(1-\rho^2)}\left[\left(\frac{x-\mu_X}{\sigma_X}\right)^2 - 2\rho\left(\frac{x-\mu_X}{\sigma_X}\right)\left(\frac{y-\mu_Y}{\sigma_Y}\right) + \left(\frac{y-\mu_Y}{\sigma_Y}\right)^2\right]\right)$$

where ρ is the correlation between X and Y



2-D Gauss Kernel

Now this kernel is applied on an image.

The actual Image

.

Image when Gaussian kernel is applied.

Explanation:-

The Gaussian distribution (Normal) has the highest or the peak value as the mean which is the center of the picture int his case.
As the Gaussian kernel is multiplied with the image pixel values, and the kernel has low values towards the sides, the image near the sides becomes darker and the maximum brightness is retained at the center.
The radius of the visible region can be controlled by changing the standard deviation of the Gaussian distribution and the centring can be controlled be changing the mean of the distribution.

Python code:-

```
import cv2
import numpy as np

img = cv2.imread('norway.jpg')
rows, cols = img.shape[:2]

kernel_x = cv2.getGaussianKernel(cols,200)
kernel_y = cv2.getGaussianKernel(rows,200)

kernel = kernel_y * kernel_x.T

kernel = kernel/np.linalg.norm(kernel)

mask = 255 * kernel
output = np.copy(img)

for i in range(3):
        output[:,:,i] = output[:,:,i] * mask
cv2.imshow('Original', img)
cv2.imshow('Gaussian', output)
cv2.waitKey(0)
```

References :

- https://www.youtube.com/@DigitalSreeni
- https://www.youtube.com/@Aryanverma2infoaryan