# <u>Project Report</u>

Bijeet Basak(IMT2022510)

Rohan Rajesh(IMT2022575)

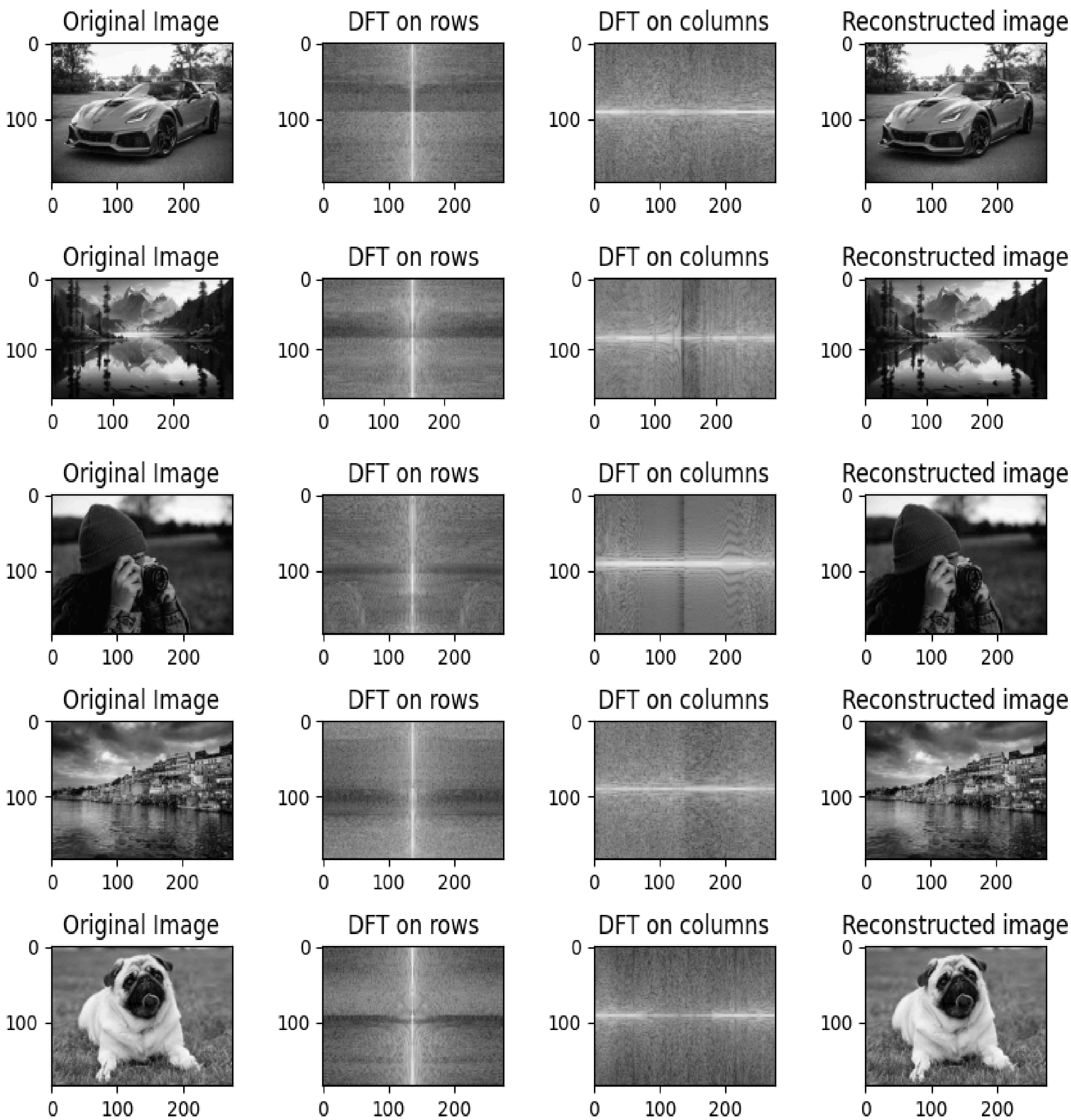
Teerth Bhalgat(IMT2022586)

B) <u>Compare the original image with its one-dimensional DFT and two-dimensional image and comment on the images obtained. (Obtain the 1D and 2D images by taking the absolute of the DFT.)</u>

## Observations/Graphs for 1D DFT-

- Peaks in the magnitude spectrum indicate dominant frequencies in each row/column.
- White spaces represent high magnitude while black spaces represent low magnitude.

## The results of 1D DFT:



**Code -**

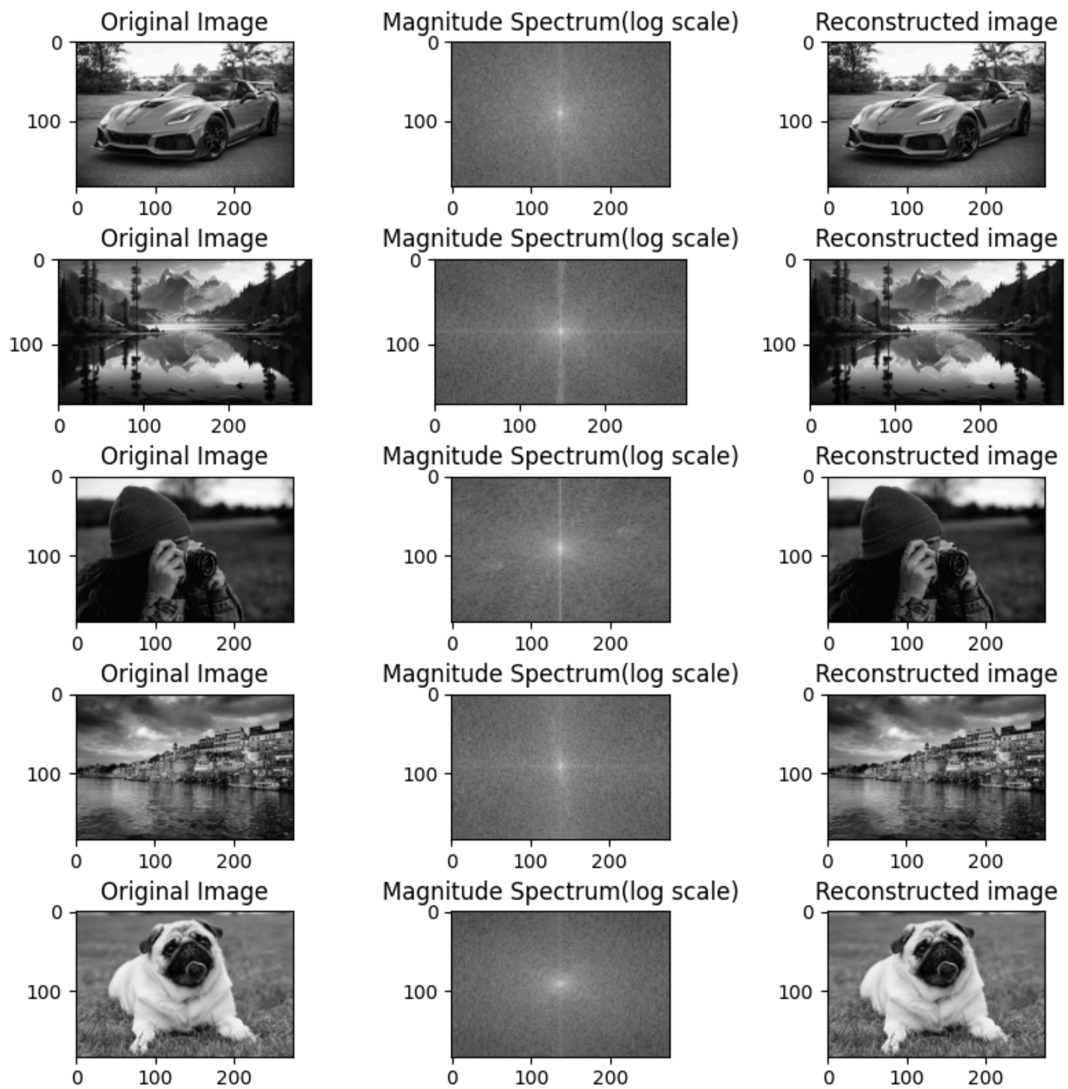
```
import numpy as np
import cv2
import matplotlib.pyplot as plt

images=[cv2.imread('image1.jpg'),
        cv2.imread('image2.jpg'),
        cv2.imread('image3.jpg'),
        cv2.imread('image4.jpg'),
        cv2.imread('image5.jpg')]

# converting to gray scale
images_gray = [cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) for img in images]

# implementing fft on rows
images_dft_rows = [np.fft.fft(img_gray,axis=1) for img_gray in images_gray]
images_dft_rows_shift = [np.fft.fftshift(img) for img in images_dft_rows]
# implementing fft on columns
images_dft_columns = [np.fft.fft(img_gray,axis=0) for img_gray in images_gray]
images_dft_columns_shift = [np.fft.fftshift(img) for img in images_dft_columns]

# implementing inverse fft on rows
images_idft_rows = [np.fft.ifft(image,axis=1) for image in images_dft_rows]
```

```python
# implementing inverse fft on columns
images_idft_columns = [np.fft.ifft(image,axis=0) for image in images_dft_columns]

# reconstructing the original image
reconstructed_images = [(np.real(image_rows+image_columns)) for
image_rows,image_columns in zip(images_idft_rows,images_idft_columns)]

fig,axes = plt.subplots(5,4,figsize=(10,20))

for i in range(len(images)):
axs[i][0].imshow(images_gray[i],cmap='gray')
axs[i][0].set_title('Original Image')
axs[i][1].imshow(np.log(1+np.abs(images_dft_rows_shift[i])),cmap='gray')
axs[i][1].set_title('DFT on rows')
axs[i][2].imshow(np.log(1+np.abs(images_dft_columns_shift[i])),cmap='gray')
axs[i][2].set_title('DFT on columns')
axs[i][3].imshow(reconstructed_images[i],cmap='gray')
axs[i][3].set_title('Reconstructed image')
plt.subplots_adjust(wspace=0.5)
plt.show()
```

## Observations/Graphs for 2D DFT-

- The x-axis represents the horizontal frequencies and y-axis represents the vertical frequencies.
- The 2D DFT result is typically shifted so that the low frequencies are centered in the image. This makes it easier to visualize the low-frequency components, which are usually more important for image reconstruction.

**The results of 2D DFT:**



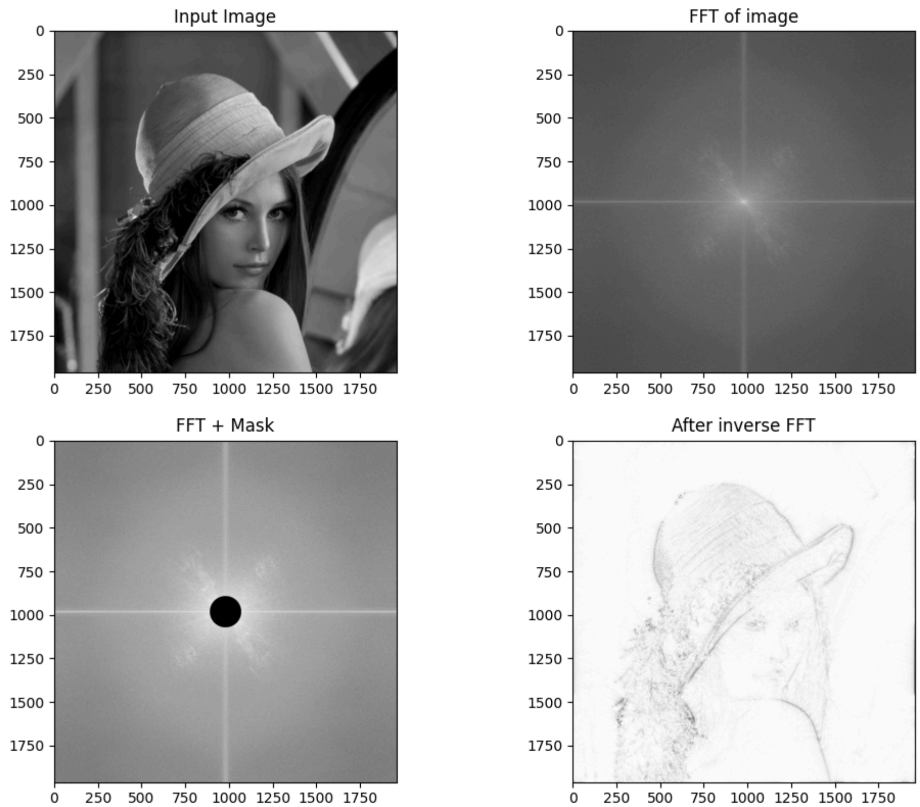
**Code -**

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

Images = [cv2.imread('image1.jpg'),
          cv2.imread('image2.jpg'),
          cv2.imread('image3.jpg'),
          cv2.imread('image4.jpg'),
          cv2.imread('image5.jpg')]
# converting to grayscale
images_gray = [cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) for img in images]

# implementing fft
images_dft = [np.fft.fft2(img_gray) for img_gray in images_gray]
images_dft_shif t= [np.fft.fftshift(img) for img in images_dft]

# magnitude spectrum
```

```
magnitude_spectrum = [np.log(1+np.abs(img)) for img in images_dft_shift]

# implementing inverse fft
images_idft = [np.fft.ifft2(image) for image in images_dft]

# reconstructing the original image
reconstructed_images = [(np.real(img)) for img in images_idft]
fig,axes=plt.subplots(5,3,figsize=(10,20))
for i in range(len(images)):
axs[i][0].imshow(images_gray[i],cmap='gray')
axs[i][0].set_title('Original Image')
axs[i][1].imshow(magnitude_spectrum[i],cmap='gray')
axs[i][1].set_title('Magnitude Spectrum(log scale)')
axs[i][2].imshow(reconstructed_images[i],cmap='gray')
axs[i][2].set_title('Reconstructed image')
plt.subplots_adjust(hspace=0.5)
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

Code:

```
import cv2

import numpy as np

import matplotlib.pyplot as plt
```

```
# Load the image in grayscale
img = cv2.imread('image2.jpg', cv2.IMREAD_GRAYSCALE)

# Compute the Discrete Fourier Transform
dft = cv2.dft(np.float32(img), flags=cv2.DFT_COMPLEX_OUTPUT)
dft_shift = np.fft.fftshift(dft)

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

# Create a mask for high-pass filtering
rows, cols = img.shape
crow, ccol = rows // 2, 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 ** 2
mask[mask_area] = 0

# Apply the mask and compute the inverse DFT
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])

# Adjust the brightness of the image
```

```
max_val = np.max(img_back)
img_back = max_val - img_back

# Display the images
plt.figure(figsize=(12, 12))
plt.subplot(2, 2, 1)
plt.imshow(img, cmap='gray')
plt.title('Input Image')

plt.subplot(2, 2, 2)
plt.imshow(magnitude_spectrum, cmap='gray')
plt.title('Magnitude Spectrum')

plt.subplot(2, 2, 3)
plt.imshow(fshift_mask_mag, cmap='gray')
plt.title('FFT + Mask')

plt.subplot(2, 2, 4)
plt.imshow(img_back, cmap='gray')
plt.title('After Inverse FFT')
plt.show()
```

<u>C) What happens if the 2D-DFT image is multiplied by a 2D Gaussian symmetric window? What does the original image look like after inverse DFT?</u>

## Gaussian distribution :-

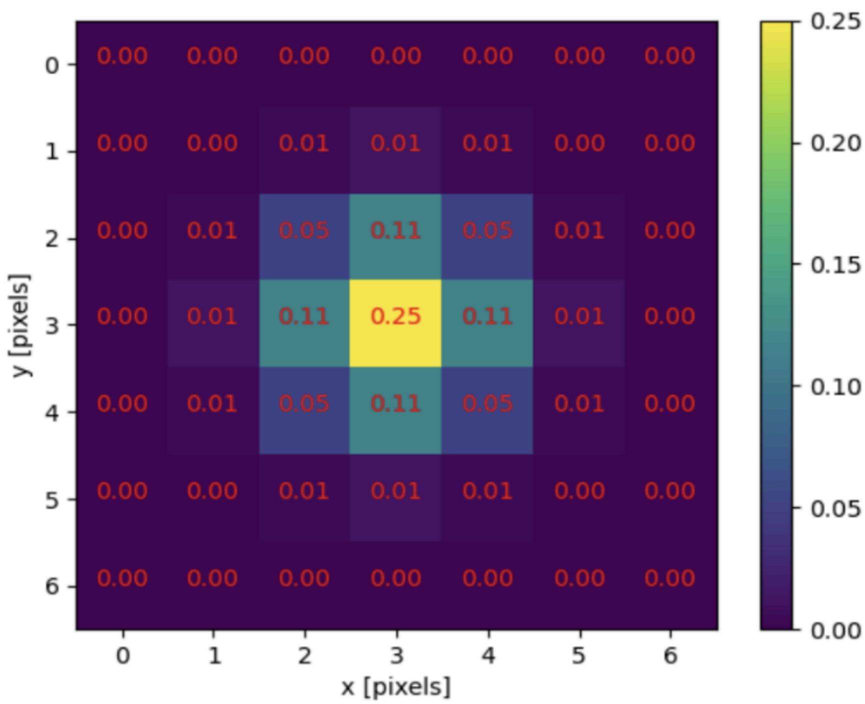
The Gaussian distribution (also known as the normal distribution) is a bell-shaped curve with an equal number of observations above and below the mean value.
1-D Gaussian Distribution function,

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

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 $\rho$ is the correlation between X and Y



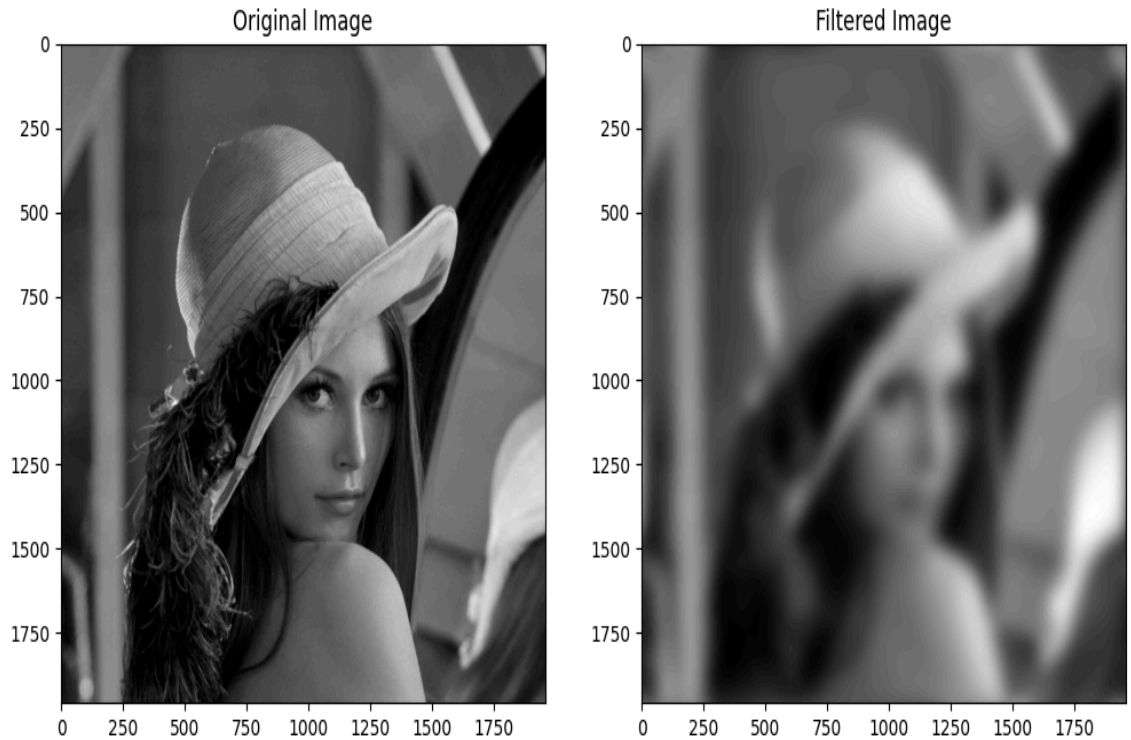

2-D Gauss Kernel

## Code:

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

def gaussian_filter(shape, sigma):
   m, n = [(ss-1.)/2. for ss in shape]
   y, x = np.ogrid[-m:m+1,-n:n+1]
   h = np.exp(-(x*x + y*y) / (2.*sigma*sigma))
   h /= h.sum()
   return h

# Load the image
img = cv2.imread('image1.jpg', cv2.IMREAD_GRAYSCALE)

# Apply 2D DFT
f = np.fft.fft2(img)
fshift = np.fft.fftshift(f)

# Create Gaussian filter
rows, cols = img.shape
gaussian = gaussian_filter((rows, cols), sigma=10)

# Apply the Gaussian filter in the frequency domain
fshift_filtered = fshift * gaussian

# Apply inverse DFT to reconstruct the image
f_ishift_filtered = np.fft.ifftshift(fshift_filtered)
img_filtered = np.fft.ifft2(f_ishift_filtered)
img_filtered = np.abs(img_filtered)
```

```
# Display the original and filtered images
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.imshow(img, cmap='gray')
plt.title('Original Image')
plt.subplot(1, 2, 2)
plt.imshow(img_filtered, cmap='gray')
plt.title('Filtered Image')
plt.show()
```

**Output:**




## Conclusion:

- Multiplying the 2D-DFT image by a 2D Gaussian symmetric window in the frequency domain effectively applies a Gaussian low-pass filter to the image. This process attenuates higher frequencies while preserving lower frequencies, resulting in a smoother version of the image. This is because the Gaussian filter acts as a smoothing function, reducing the contribution of high-frequency components.

- After applying the inverse DFT to the filtered image, you will obtain a version of the original image where high-frequency details are suppressed, leading to a smoother appearance. This is because the Gaussian filter effectively blurs the image by attenuating high-frequency components.