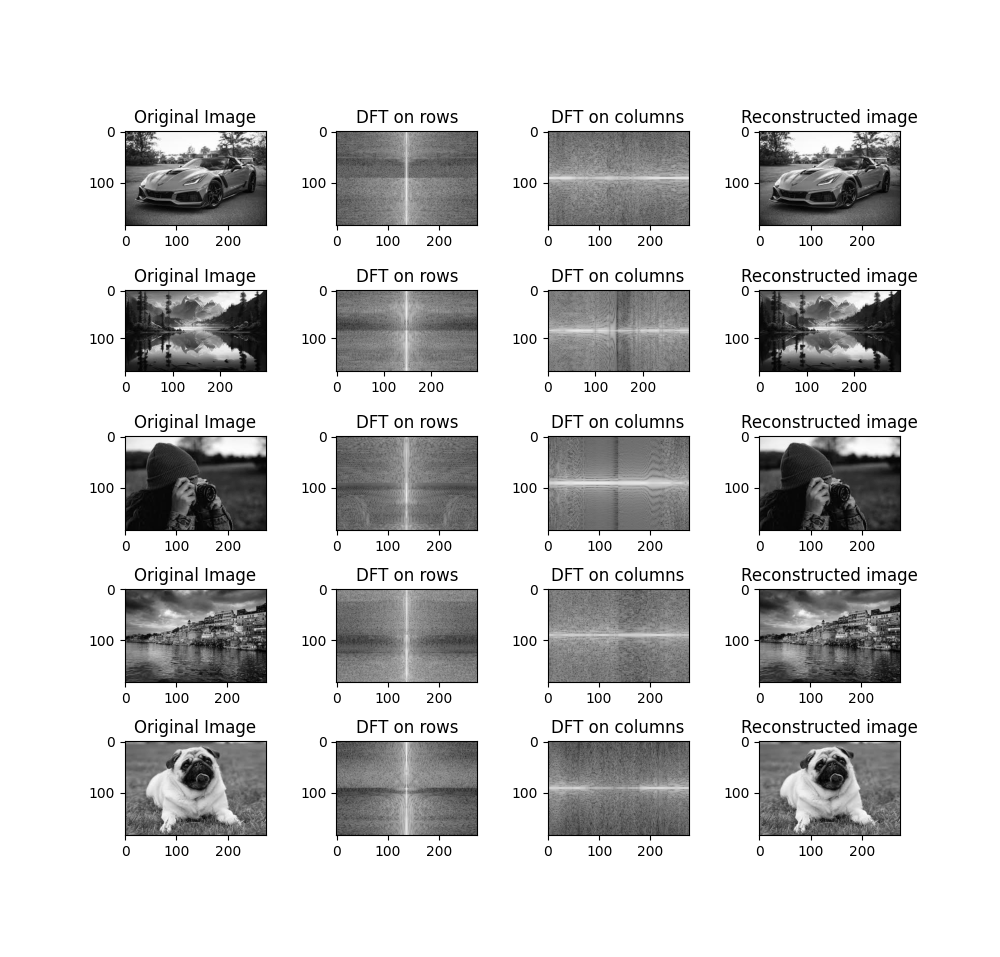
B) Compare the original image with its one-dimensional DFT and two-dimensional image and comment on the images obtained. (Obtain the 1D and 2D images by taking the absolute of the DFT.)

**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]

# *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')

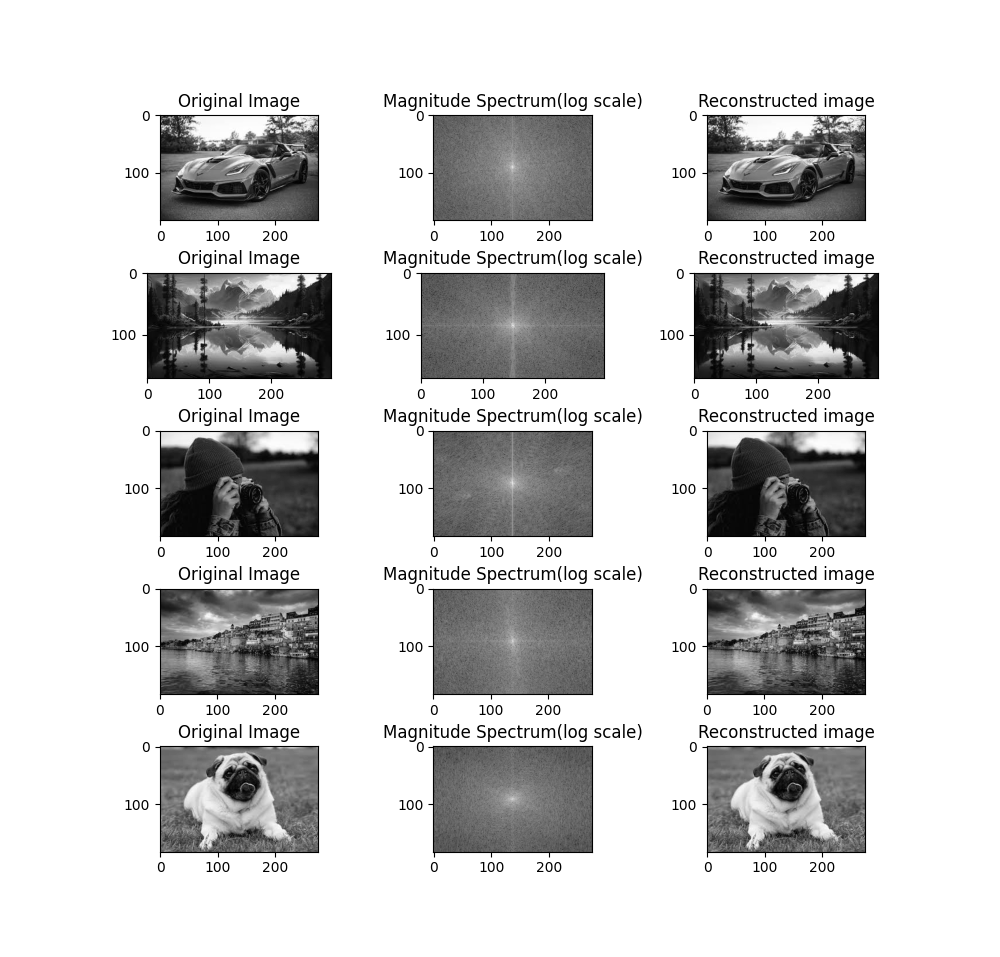
plt.subplots\_adjust(wspace=0.5)

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()

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?

**Code:**

**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')**

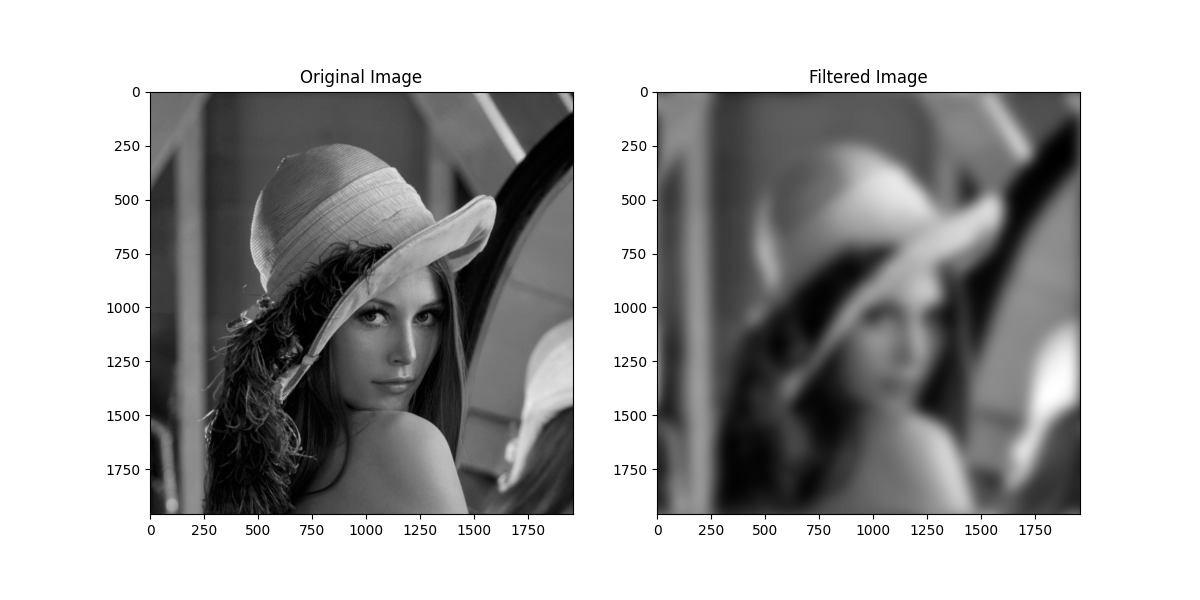
**plt.subplot(1, 2, 2)**

**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.