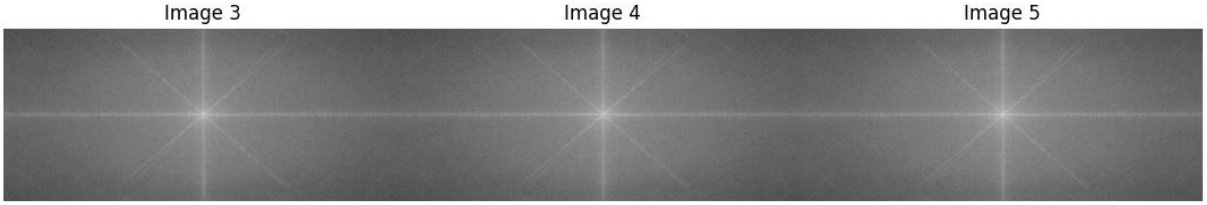
```python
import numpy as np
import pandas as pd
import cv2
from matplotlib import pyplot as plt
import matplotlib.cm as cm
import matplotlib.gridspec as gridspec
from pylab import imread
from skimage.color import rgb2gray

def ShowImage(ImageList, nRows = 1, nCols = 2, WidthSpace = 0.00, HeightSpace = 0.00):
    from matplotlib import pyplot as plt
    import matplotlib.gridspec as gridspec

    gs = gridspec.GridSpec(nRows, nCols)
    gs.update(wspace=WidthSpace, hspace=HeightSpace) # set the spacing between axes.
    plt.figure(figsize=(20,10))
    for i in range(len(ImageList)):
        ax1 = plt.subplot(gs[i])
        ax1.set_xticklabels([])
        ax1.set_yticklabels([])
        ax1.set_aspect('equal')

        plt.subplot(nRows, nCols,i+1)

        image = ImageList[i].copy()
        if (len(image.shape) < 3):
            plt.imshow(image, plt.cm.gray)
        else:
            plt.imshow(image)
        plt.title("Image " + str(i))
        plt.axis('off')

    plt.show()
# Read Image
image_color = imread("Sample04/house.jpg")
# Convert Image into Gray
image_gray = cv2.cvtColor(image_color, cv2.COLOR_RGB2GRAY)

# Display Image
ShowImage([image_color, image_gray], 1, 2)
```

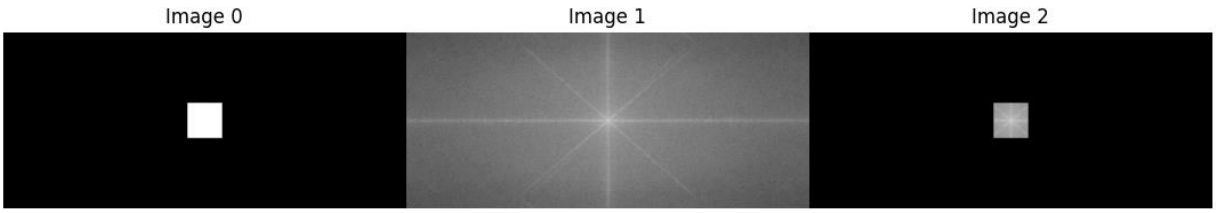
```python
def Image3Dto2D(image):
    if(len(image.shape) >= 3):
        image_2D = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
    else:
        image_2D = image.copy()
    return image_2D

# https://www.bogotobogo.com/python/OpenCV_Python/python_opencv3_Image_Fourier_Transform_FFT_DFT.php
def DFT_Transformation(image):
    img = Image3Dto2D(image)

    img_float32 = np.float32(img)
    dft = cv2.dft(img_float32, flags = cv2.DFT_COMPLEX_OUTPUT)
    dft_shift = np.fft.fftshift(dft)
    magnitude_spectrum = 20*np.log(cv2.magnitude(dft_shift[:,:,0],dft_shift[:,:,1]))

    return magnitude_spectrum, dft_shift

image_dft_frequency, dft_shift = DFT_Transformation(image_gray)
image_dft_frequency1, dft_shift1 = DFT_Transformation(image_color[:,:,0])
image_dft_frequency2, dft_shift2 = DFT_Transformation(image_color[:,:,1])
image_dft_frequency3, dft_shift3 = DFT_Transformation(image_color[:,:,2])
ShowImage([image_color, image_gray,  image_dft_frequency, image_dft_frequency1, image_dft_frequency2, image_dft_frequ
```

Image 0 Image 1 Image 2

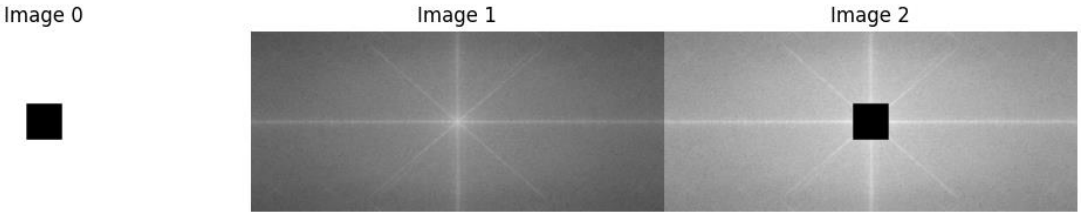
Image 3 Image 4 Image 5



```
    # create a mask first, center square is 1, remaining all zeros
    mask = np.zeros((rows, cols, 2), np.uint8)
    size = 50
    mask[crow-size:crow+size, ccol-size:ccol+size] = 1
    image_dft_frequency_crop = image_dft_frequency* mask[:,:,0]

    # apply mask and inverse DFT
    fshift = dft_shift*mask
    f_ishift = np.fft.ifftshift(fshift)
    img_inverse = cv2.idft(f_ishift)
    image_inverse = cv2.magnitude(img_inverse[:,:,0],img_inverse[:,:,1])

    ShowImage([mask[:,:,0], image_dft_frequency, image_dft_frequency_crop], 1, 3)
    ShowImage([image_gray, image_inverse], 1, 2)
```



```
    # create a mask first, center square is 1, remaining all zeros
    mask = np.zeros((rows, cols, 2), np.uint8)
    size = 50
    mask[crow-size:crow+size, ccol-size:ccol+size] = 1
    mask = 1 - mask
    image_dft_frequency_crop = image_dft_frequency* mask[:,:,0]

    # apply mask and inverse DFT
    fshift = dft_shift*mask
    f_ishift = np.fft.ifftshift(fshift)
    img_inverse = cv2.idft(f_ishift)
    image_inverse = cv2.magnitude(img_inverse[:,:,0],img_inverse[:,:,1])

    ShowImage([mask[:,:,0], image_dft_frequency, image_dft_frequency_crop], 1, 3)
    ShowImage([image_gray, image_inverse], 1, 2)
```



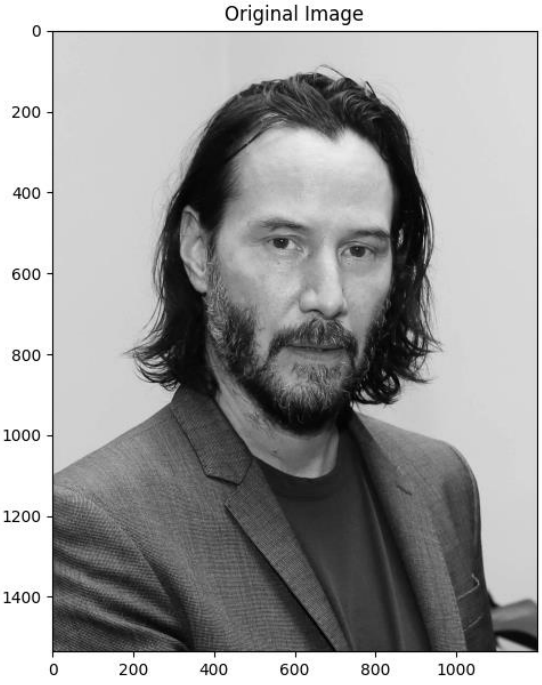
```
    # Read Image
    image_color = imread("Sample04/keanu.jpg")
    # Convert Image into Gray
    image_gray = cv2.cvtColor(image_color, cv2.COLOR_RGB2GRAY)

    # Display Image
    ShowImage([image_color, image_gray], 1, 2)
```

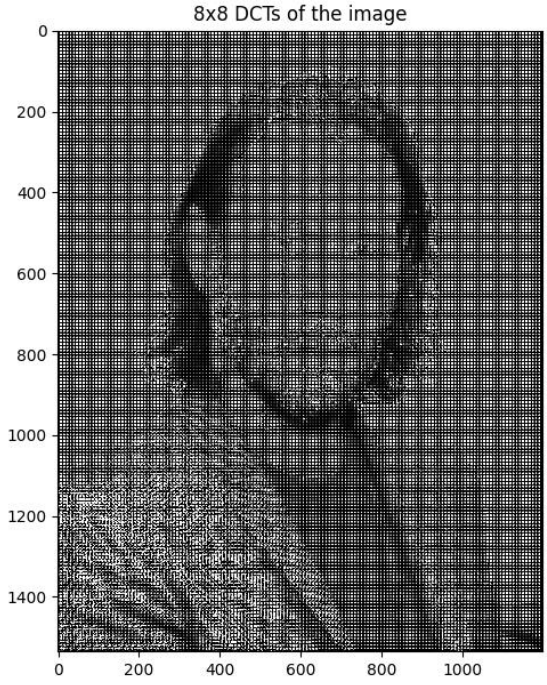
Image 0

Image 1

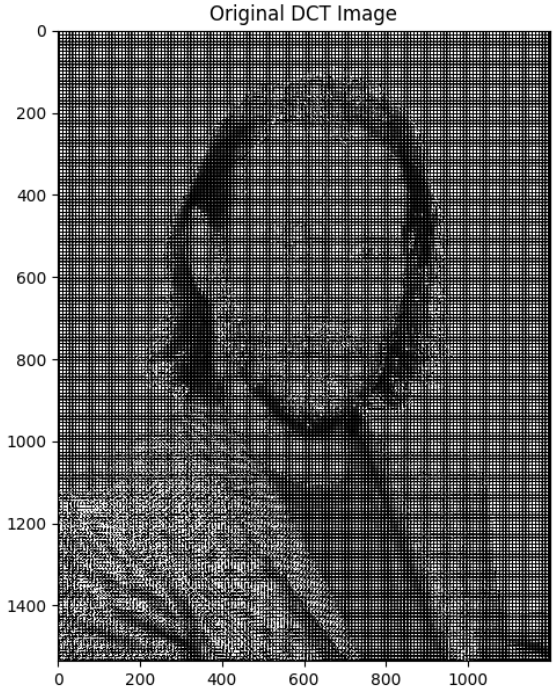
```
def dct2(a):
    import scipy.fftpack
    return scipy.fftpack.dct(scipy.fftpack.dct( a, axis=0, norm='ortho' ), axis=1, norm='ortho' )

def idct2(a):
    import scipy.fftpack
    return scipy.fftpack.idct(scipy.fftpack.idct( a, axis=0 , norm='ortho'), axis=1 , norm='ortho')

im = image_gray
imsize = im.shape
dct = np.zeros(imsize)

# Do 8x8 DCT on image (in-place)
for i in np.r_[:imsize[0]:8]:
    for j in np.r_[:imsize[1]:8]:
        dct[i:(i+8),j:(j+8)] = dct2( im[i:(i+8),j:(j+8)] )
```

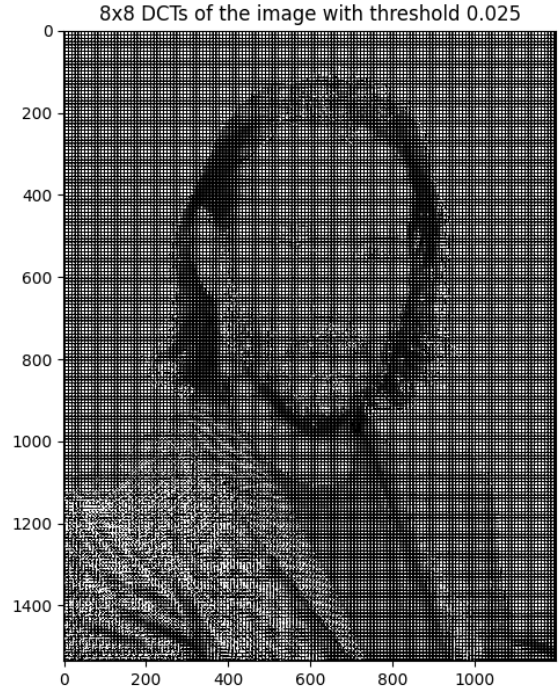
```
    # Display entire DCT
    plt.figure(figsize=(20,10))
    plt.subplot(1,2,1)
    plt.imshow(image_gray, cmap = 'gray')
    plt.title( "Original Image")
    plt.subplot(1,2,2)
    plt.imshow(dct,cmap='gray',vmax = np.max(dct)*0.01,vmin = 0)
    plt.title( "8x8 DCTs of the image")
    plt.show()
```






```
188     # Threshold
189     thresh = 0.025
190     dct_thresh = dct * (abs(dct) > (thresh*np.max(dct)))
191
192     plt.figure(figsize=(20,10))
193     plt.subplot(1,2,1)
194     plt.imshow(dct,cmap='gray',vmax = np.max(dct)*0.01,vmin = 0)
195     plt.title( "Original DCT Image")
196     plt.subplot(1,2,2)
197     plt.imshow(dct_thresh,cmap='gray',vmax = np.max(dct)*0.01,vmin = 0)
198     plt.title( "8x8 DCTs of the image with threshold " + str(thresh))
199     plt.show()
```





```
    percent_nonzeros = np.sum( dct_thresh != 0.0 ) / (imsize[0]*imsize[1]*1.0)
    print("Keeping only %f%% of the DCT coefficients" % (percent_nonzeros*100.0))

    im_dct = np.zeros(imsize)
    for i in np.r_[:imsize[0]:8]:
        for j in np.r_[:imsize[1]:8]:
            im_dct[i:(i+8),j:(j+8)] = idct2( dct_thresh[i:(i+8),j:(j+8)] )

    print("Comparison between original and DCT compressed images" )
    ShowImage([im, im_dct])
```

```
Keeping only 4.925564% of the DCT coefficients
Comparison between original and DCT compressed images
```

Image 0

Image 1