REPORT: # IMAGE COMPRESSION AND QUALITY ESTIMATION

Atanda Abdullahi Adewale

INTRODUCTION:

This exercise applies threshold to the magnitude spectrum to filter out high-frequency components, and then reconstructs the image using the inverse Fourier Transform and Inverse Cosine Transform seperately.
It visualizes the frequency coefficients of the image using a logarithmic scale and displays the magnitude spectrum of the filtered image.

Peak Signal to noise ratio (PSNR) and Compression ratio are both calculated between the Original and Reconstructed image at different Values of Threshold T

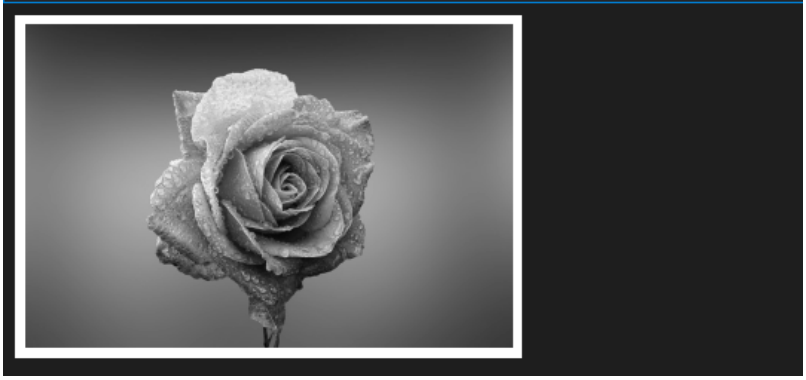
This in essence is to apply Fourier Transform and DCT for image compression

METHOD:

- Load the image into the program and convert it to Gray-scale if it's in color.

```
img = cv2.imread('flower.jpg', cv2.IMREAD_GRAYSCALE)

plt.figure()
plt.imshow(img, cmap='gray')
plt.axis('off')
plt.show()
```

✓ 0.1s



- Take the 2D FFT of the image using a function such as np.fft.fft2() in NumPy.
- Shift the zero frequency component (DC component) to the center of the spectrum using np.fft.fftshift().
- Take the absolute value of the complex result to get the amplitude spectrum.
- Take the logarithm of the result to make it easier to visualize.

```
f = np.fft.fft2(img)
fshift = np.fft.fftshift(f)
magnitude_spectrum = 20 * np.log(np.abs(fshift))
```

Threshold

- Discarding low coefficients, or thresholding, involves setting any coefficients in the amplitude spectrum below a certain threshold to zero:

- Determine the threshold value (e.g. 30% of the maximum amplitude)

```
#threshold = 0.3 * np.max(magnitude_spectrum)
threshold = 100
```

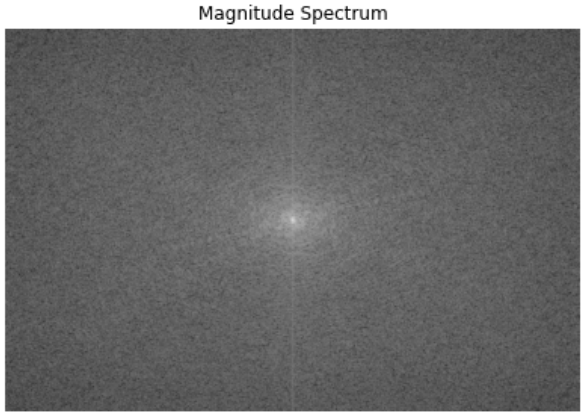
- Create a mask with the same size as the amplitude spectrum, where all values below the threshold are set to zero and all values above are set to one
- Obtain a filtered magnitude spectrum: Multiply the amplitude spectrum by the mask to set the low coefficients to zero.

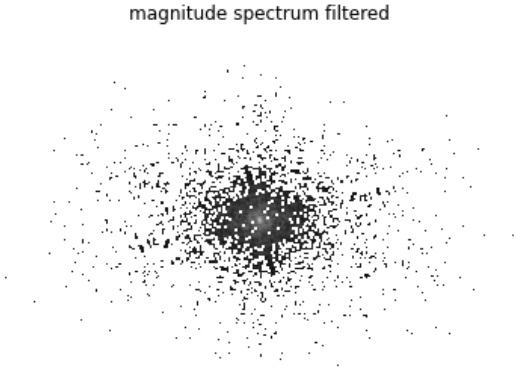
```
#Apply a high-pass filter to the Fourier transform
N,M = img.shape
print('image shape:',N,M)
mask = magnitude_spectrum < threshold
fshift[mask]=0

magnitude_spectrum_filtered = 20 * np.log(np.abs(fshift))
```

Reconstructing the Frequency and Creating the Reconstructed Image:

Shift the magnitude spectrum back to its original position using numpy.fft.ifftshift(). Multiply the phase spectrum (angle of the complex result from the original FFT) by the modified magnitude spectrum. Take the inverse Fourier transform of the modified complex spectrum using a function like numpy.fft.ifft2() to get the reconstructed image

```
f_ishift = np.fft.ifftshift(fshift)
reconstructed_img = np.fft.ifft2(f_ishift)
reconstructed_img = np.abs(reconstructed_img)
```





normalized magnitude_spectrum array to range 0-1, and then scaling them to the range 0-255 so that they can be displayed as an image.

magnitude_spectrum = (magnitude_spectrum -np.min(magnitude_spectrum))/(np.max(magnitude_spectrum)-np.min(magnitude_spectrum))*255

Magnitude_Spectrum_filtered is obtained at Threshold T = 100

```
plt.imshow(reconstructed_img, cmap='gray')
plt.axis('off')
plt.title('Reconstructed Image')
plt.show()
```

✓ 0.1s

Reconstructed Image

Computing PSNR:

Load the original image and the reconstructed image.
Calculate the Mean Squared Error (MSE) between the two images using a function like skimage.metrics.mean_squared_error().
Compute the maximum pixel value of the image (e.g., 255 for an 8-bit image).
Calculate the PSNR using the formula: PSNR = 20 * log10(max_pixel_value / sqrt(MSE)).

```
def PSNR(original, reconstructed):
    mse = np.mean((original - reconstructed) ** 2)
    if(mse == 0):  # MSE is zero means no noise signal
        return 100
    max_pixel = 255.0
    psnr = 20 * math.log10(max_pixel / math.sqrt(mse))
    return psnr
```

Computing Compression Ratio:

Ratio of the number of frequency coefficients that are below a specified threshold to the total number of coefficients in the magnitude spectrum of an image

```
# count True and False values
n_false = np.count_nonzero(mask == False)
n_true = np.count_nonzero(mask == True)
discarded_ratio = np.count_nonzero(mask == True)/(N*M) #OR
discarded_ratio = np.sum(magnitude_spectrum < threshold)/(N*M)

# print results
print("Number of False values:", n_false)
print("Number of True values:", n_true)
print("Discarded Ratio:", discarded_ratio, "%")
```

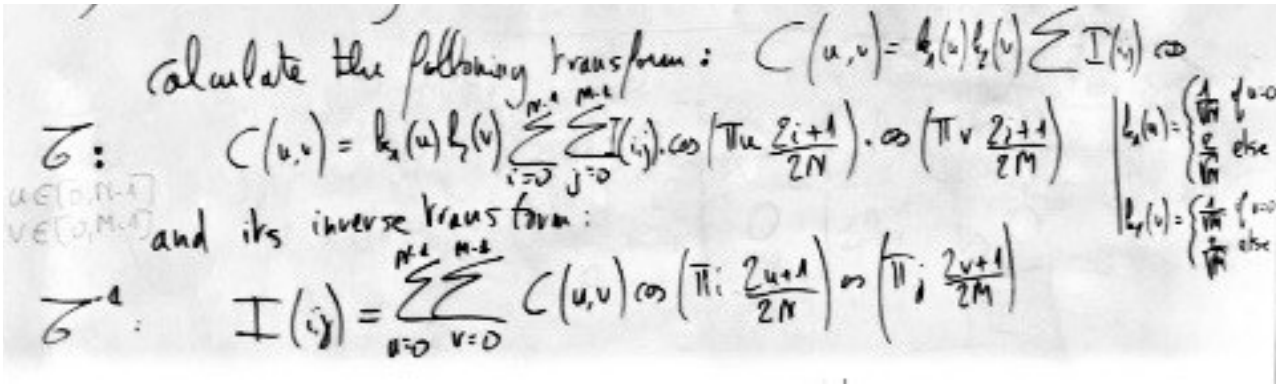
Changing the Threshold level for optimal compression

Threshold	PSNR	Discarded Ratio
50	78.63	0.0044
100	34.40	0.59
150	23.59	0.98
200	19.09	0.99

PART 2

Following the same method in the image reconstruction, we replace FFT and inverse iFFT with the Transform below and obtain the PSNR and Discarded Ratio with varying Thresholds:

Discrete cosine transform (DCT) and its inverse-transform is given by Z and Z^{-1}:

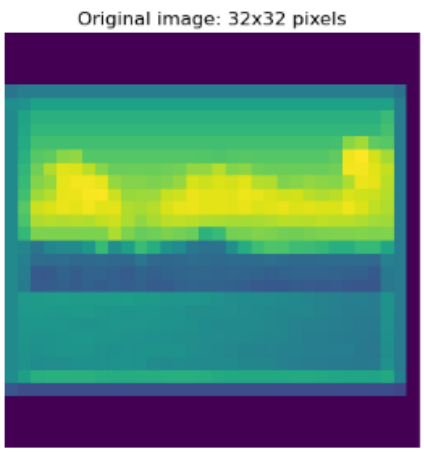


computed by:

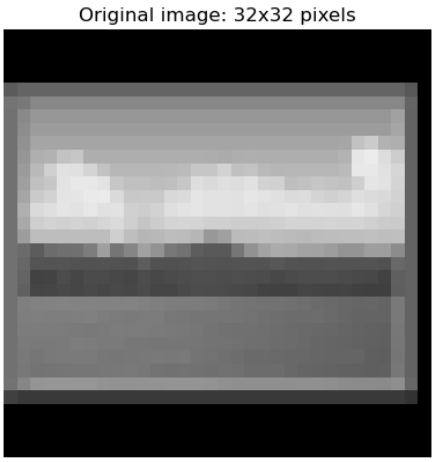
```
N,M = img.shape
C=np.zeros((N,M))
I_recover=np.zeros((N,M))

Threshold = 20

for u in range(N):
    for v in range(M):
        k1=2/np.sqrt(N)
        k2=2/np.sqrt(M)
        if u==0:
            k1=1/np.sqrt(N)
        if v==0:
            k2=1/np.sqrt(M)
        for i in range (N):
            for j in range(M):
                C[u,v]+=k1*k2*img[i,j]*np.cos(np.pi*u*(2*i+1)/(2*N))*np.cos(np.pi*v*(2*j+1)/(2*M))
```

To visualize the filtered magnitude spectrum from threshold:

```
Threshold = 20
magnitude_spectrum =  10*np.log(np.abs(C))
magnitude_spectrum = (magnitude_spectrum -np.min(magnitude_spectrum))/(np.max(magnitude_spectrum)-np.min(magnitude_spectrum))*255
mask= magnitude_spectrum<Threshold
C[mask]=0

magnitude_spectrum_filtered = 20 * np.log(np.abs(C))
```

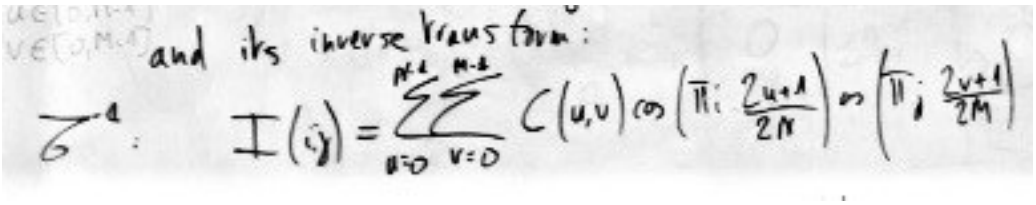


convert to Grayscale:

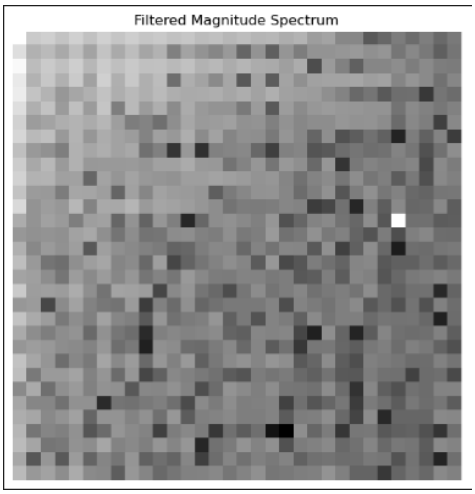


Original image is a 32×32 pixels image: Higher resolution images takes time to compute, GPU or Multi-threading computation may help

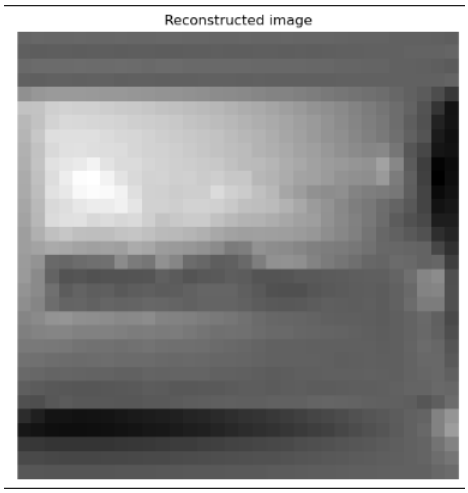
the Inverse DCT computation:




```python
#Reconstruct Image from the transform
for i in range(N):
    for j in range(M):
        for u in range(N):
            for v in range(M):
                I_recover[i,j]+=C[u,v]*np.cos(np.pi*i*(2*u+1)/(2*N))*np.cos(np.pi*j*(2*v+1)/(2*M))

I_recover = ((I_recover - np.min(I_recover)) / (np.max(I_recover) - np.min(I_recover))) * 255 # normalize 0-255
I_recover = cv2.convertScaleAbs(I_recover)  # convert 8-bit integer (uint8)
```



Filtered magnitude spectrum



Reconstructed Image

Varying Threshold level for optimal compression:

Threshold	**PSNR**	**Discarded Ratio**
50	27.88	0.0022
100	27.90	0.05
150	27.89	0.56
200	28.01	0.96

CONCLUSION:
For Fourier transform, PSNR at threshold 50 shows good quality in the compressed image, threshold 100 produces better compression with good discarded ratio of pixel for compression.

For Discrete cosine transform (DCT), threshold at 150 produces optimal compression per discarded ratio. Reconstructed image becomes blurry at Threshold > 150.