Sixth report - Image Compression using FFT and DCT

Kimberly Grace Sevillano C.

***Subject*— Comparing FFT, DCT compression techniques using PSNR for image quality.**

I. Objective

The objective of this report is to compare the performance of two image compression techniques, Fast Fourier Transform (FFT) and Discrete Cosine Transform (DCT), by applying various threshold values and calculating the corresponding Peak Signal-to-Noise Ratio (PSNR) values.

A. Problem Description

Image compression is crucial for reducing the storage space and transmission bandwidth required for images. The two techniques under consideration, FFT and DCT, are widely used for transforming images into frequency domain representations, which can then be compressed by discarding low-frequency coefficients.

II. Methods

The following methods were used in this study:

1) Fast Fourier Transform (FFT): A fast algorithm for computing the discrete Fourier transform of an image.
2) Discrete Cosine Transform (DCT): A widely-used transformation technique that converts an image into its frequency domain representation, focusing on cosine functions.

III. Methodology

The methodology for each technique (FFT and DCT) is explained below:

A. Fast Fourier Transform (FFT)

The FFT is an efficient algorithm to compute the Discrete Fourier Transform (DFT) of a sequence, including images. The DFT of an image is given by the following equation:

$$F(u,v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) \cdot e^{-j2\pi\left(\frac{ux}{M} + \frac{vy}{N}\right)} \tag{1}$$

where $F(u,v)$ is the transformed image, $f(x,y)$ is the original image, and M and N are the dimensions of the image.

The image was first transformed using the FFT function (fft2), and the magnitude spectrum was calculated using the absolute values of the FFT coefficients. Then, the threshold was applied, setting the coefficients below the threshold to zero. The inverse FFT (ifft2) was used to reconstruct the image, and the PSNR was calculated for the original and reconstructed images.

B. Discrete Cosine Transform (DCT)

The DCT is another widely-used technique to transform images into the frequency domain, focusing on cosine functions. The 2D DCT of an image is given by the following equation:

$$C(u,v) = k_{1(u)} k_{2(v)} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) \cos\left[\frac{\pi(2x+1)u}{2M}\right] \cos\left[\frac{\pi(2y+1)v}{2N}\right] \tag{2}$$

where $C(u,v)$ is the transformed image, $f(x,y)$ is the original image, M and N are the dimensions of the image, and $k_{1(u)}$ and $k_{2(v)}$ are normalization factors:

$$k_{1(u)} = \begin{cases} \frac{1}{\sqrt{M}} & \text{if } u = 0 \\ \frac{2}{\sqrt{M}} & \text{else} \end{cases}$$

$$k_{2(v)} = \begin{cases} \frac{1}{\sqrt{N}} & \text{if } v = 0 \\ \frac{2}{\sqrt{N}} & \text{else} \end{cases}$$

The image was first transformed using a custom DCT function, then the threshold was applied, setting the coefficients below the threshold to zero. The inverse DCT function was used to reconstruct the image, and the PSNR was calculated for the original and reconstructed images.

IV. Results

The results of the FFT and DCT methods with different threshold values are presented in the following tables:

TABLE I
FFT RESULTS

Threshold	PSNR
1222.22	18.57
2444.44	15.45
3666.67	13.52
4888.89	10.10
6111.11	7.33
7333.33	7.43
8555.56	5.40
9777.78	5.10
11000.0	4.30

The threshold and PSNR are inversely related in the context of image reconstruction using the Fourier Transform. As the threshold increases, we remove more frequency components, which decreases the quality of the reconstructed image and lowers the PSNR. By decreasing the threshold, we preserve more frequency components, improving the quality of the reconstructed image and increasing the PSNR.

Fig. 1. Comparison between real image and image reconstruction with 3 different thresholds.

TABLE II
DCT RESULTS

Threshold	PSNR
0.104	26.56
0.208	22.57
0.312	20.95
0.416	19.67
0.520	18.86
0.625	18.17
0.729	17.60
0.833	17.26
0.937	16.87

With the DCT method, as the threshold (T) increases, more high-frequency information is discarded, leading to higher compression and a lower quality reconstructed image. Consequently, the PSNR value decreases with increasing threshold. Conversely, lower thresholds retain more coefficients, resulting in a higher PSNR value and a more similar reconstructed image to the original. However, lower thresholds also mean less compression and larger file sizes.

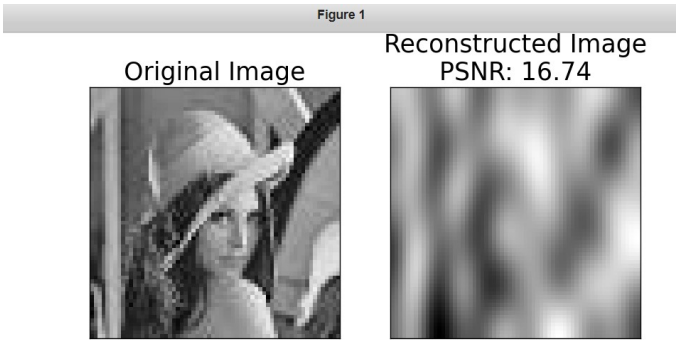

Fig. 2. Comparison between real 64-bit image and image reconstruction with threshold 0.95 and PSNR 16.74.

V. CONCLUSIONS

Based on the results, it can be concluded that both FFT and DCT are effective methods for image compression. However, DCT appears to perform better than FFT in terms of maintaining a higher PSNR value for the same threshold values. This suggests that DCT may be more suitable for image compression applications where the quality of the reconstructed image is a priority.

REFERENCES

[1] Cooley, J. W., Tukey, J. W. (1965). An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation*, 19(90), 297-301.
[2] Oppenheim, A. V., Schafer, R. W., Buck, J. R. (1999). *Discrete-time signal processing* (2nd ed.). Prentice Hall.
[3] Ahmed, N., Natarajan, T., Rao, K. R. (1974). Discrete cosine transform. *IEEE Transactions on Computers*, C-23(1), 90-93.
[4] Rao, K. R., Yip, P. (1990). *Discrete cosine transform: algorithms, advantages, applications*. Academic Press.

Report Sixth

March 06, 2023

[1]:
```
import matplotlib.pyplot as plt
import numpy as np
from skimage import io
from skimage.metrics import peak_signal_noise_ratio
from scipy.fft import fft2, ifft2
import ipywidgets as widgets
from tqdm.notebook import tqdm
import pandas as pd
```

1. Apply the FFT and discard low coefficients (Threshold T) reconstruct the image real and compute the PSNR (image [0,255])

[2]:
```
%matplotlib notebook
# Load the image (grayscale)
img = io.imread('lena.png', as_gray=True)

# Apply the FFT
img_fft = fft2(img)

# Get the magnitude spectrum
img_mag = np.abs(img_fft)

def update_image(threshold):
    if threshold == 0:
        img_recon = img
        psnr = "N/A (original image)"
    else:
        img_mag_thresh = img_mag.copy()
        img_mag_thresh[img_mag_thresh < threshold] = 0
        img_recon = np.real(ifft2(img_mag_thresh))
        img_recon = (img_recon - img_recon.min()) * 255 / (img_recon.max() -␣
 ↪img_recon.min())
        img_recon = img_recon.astype(np.uint8)
        psnr = peak_signal_noise_ratio(img, img_recon, data_range=255)

    print("PSNR:", psnr)
    ax[1].imshow(img_recon, cmap='gray')
    ax[1].set_title('reconstruction', fontsize=20)
```

```
    fig.canvas.draw_idle()

# Set up the slider widget
slider = widgets.IntSlider(value=0, min=0, max=11000, step=1,␣
↪description='Threshold:')

# Create the figure and axes
fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(8, 3))

# Display the original image
ax[0].imshow(img, cmap='gray')
ax[0].set_title('image', fontsize=20)

# Display the reconstructed image
ax[1].imshow(img, cmap='gray')
ax[1].set_title('reconstruction', fontsize=20)

# Add the slider to the plot and set the callback function
widgets.interact_manual(update_image, threshold=slider)

# Show the plot
plt.show()
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
interactive(children=(IntSlider(value=0, description='Threshold:', max=11000),␣
↪Button(description='Run Interac...
```

[3]:
```
# Load the image (grayscale)
img = io.imread('lena.png', as_gray=True)

# Apply the FFT
img_fft = fft2(img)

# Get the magnitude spectrum
img_mag = np.abs(img_fft)

def calculate_psnr(threshold):
    if threshold == 0:
        img_recon = img
        psnr = "N/A (original image)"
    else:
        img_mag_thresh = img_mag.copy()
        img_mag_thresh[img_mag_thresh < threshold] = 0
        img_recon = np.real(ifft2(img_mag_thresh))
```

```
        img_recon = (img_recon - img_recon.min()) * 255 / (img_recon.max() -␣
↪img_recon.min())
        img_recon = img_recon.astype(np.uint8)
        psnr = peak_signal_noise_ratio(img, img_recon, data_range=255)

    return psnr

# Calculate PSNR for 10 threshold values in ascending order
threshold_values = np.linspace(0, 11000, num=10)
threshold_values.sort()
psnr_values = [calculate_psnr(threshold) for threshold in threshold_values]

# Create a table using pandas
data = {'Threshold': threshold_values, 'PSNR': psnr_values}
df = pd.DataFrame(data)
print(df)
```

```
      Threshold                  PSNR
0      0.000000  N/A (original image)
1   1222.222222             18.572932
2   2444.444444             15.449383
3   3666.666667             13.517774
4   4888.888889              10.09876
5   6111.111111              7.330563
6   7333.333333              7.429627
7   8555.555556              5.399063
8   9777.777778              5.099381
9  11000.000000              4.303033
```

2. using DCT (Discrete Cosine Transform)

[7]:
```
%matplotlib notebook
# Load the image (grayscale)
img = io.imread('lena64.jpg', as_gray=True)

# Set the threshold value
T = 0.95

# Implement DCT function
def custom_dct2(a):
    N, M = a.shape
    C = np.zeros((N, M))
    for u in range(N):
        for v in range(M):
            k1 = 1 / np.sqrt(N) if u == 0 else np.sqrt(2 / N)
            k2 = 1 / np.sqrt(M) if v == 0 else np.sqrt(2 / M)
            C[u, v] = k1 * k2 * np.sum([a[i, j] * np.cos(np.pi * u * (2 * i + 1)␣
↪/ (2 * N))
```

```
                                        * np.cos(np.pi * v * (2 * j + 1) / (2 *␣
↪M)) for i in range(N) for j in range(M)])
    return C

# Implement inverse DCT function
def custom_idct2(C):
    N, M = C.shape
    a = np.zeros((N, M))
    for i in range(N):
        for j in range(M):
            a[i, j] = np.sum([C[u, v] * (1 / np.sqrt(N) if u == 0 else np.sqrt(2␣
↪/ N))
                              * (1 / np.sqrt(M) if v == 0 else np.sqrt(2 / M))
                              * np.cos(np.pi * u * (2 * i + 1) / (2 * N))
                              * np.cos(np.pi * v * (2 * j + 1) / (2 * M)) for u␣
↪in range(N) for v in range(M)])
    return a

# Perform custom DCT
dct_img = custom_dct2(img)

# Apply the threshold
dct_img_thresh = dct_img.copy()
dct_img_thresh[np.abs(dct_img_thresh) < T] = 0

# Perform custom inverse DCT
idct_img = custom_idct2(dct_img_thresh)

# Compute the PSNR
psnr = peak_signal_noise_ratio(img, idct_img, data_range=img.max() - img.min())
print("PSNR:", psnr)

# Create the figure and axes
fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(8, 4))

# Display the original image
ax[0].imshow(img, cmap='gray')
ax[0].set_title('Original Image', fontsize=20)

# Display the reconstructed image
ax[1].imshow(idct_img, cmap='gray')
ax[1].set_title(f'Reconstructed Image\nPSNR: {psnr:.2f}', fontsize=20)

# Remove axis ticks
for a in ax:
    a.set_xticks([])
    a.set_yticks([])
```

```
# Show the plot
plt.show()
```

```
PSNR: 16.742427021510043
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

[6]:
```
# Load the image (grayscale)
img = io.imread('lena64.jpg', as_gray=True)

# Define threshold values to test
threshold_values = np.linspace(0,img.max(), num=10)

# Implement DCT function
def custom_dct2(a):
    N, M = a.shape
    C = np.zeros((N, M))
    for u in range(N):
        for v in range(M):
            k1 = 1 / np.sqrt(N) if u == 0 else np.sqrt(2 / N)
            k2 = 1 / np.sqrt(M) if v == 0 else np.sqrt(2 / M)
            C[u, v] = k1 * k2 * np.sum([a[i, j] * np.cos(np.pi * u * (2 * i + 1)␣
 ↪/ (2 * N))
                                        * np.cos(np.pi * v * (2 * j + 1) / (2 *␣
 ↪M)) for i in range(N) for j in range(M)])
    return C

# Implement inverse DCT function
def custom_idct2(C):
    N, M = C.shape
    a = np.zeros((N, M))
    for i in range(N):
        for j in range(M):
            a[i, j] = np.sum([C[u, v] * (1 / np.sqrt(N) if u == 0 else np.sqrt(2␣
 ↪/ N)) * (1 / np.sqrt(M) if v == 0 else np.sqrt(2 / M)) * np.cos(np.pi * u * (2␣
 ↪* i + 1) / (2 * N)) * np.cos(np.pi * v * (2 * j + 1) / (2 * M)) for u in␣
 ↪range(N) for v in range(M)])
    return a

# Initialize lists to store threshold and PSNR values
threshold_list = []
psnr_list = []

# Loop through the threshold values and calculate PSNR for each value
for T in threshold_values:
```

```
    # Perform custom DCT
    dct_img = custom_dct2(img)

    # Apply the threshold
    dct_img_thresh = dct_img.copy()
    dct_img_thresh[np.abs(dct_img_thresh) < T] = 0

    # Perform custom inverse DCT
    idct_img = custom_idct2(dct_img_thresh)

    # Compute the PSNR
    psnr = peak_signal_noise_ratio(img, idct_img, data_range=img.max() - img.
 ↪min())

    # Append the threshold and PSNR values to the lists
    threshold_list.append(T)
    psnr_list.append(psnr)

# Create a table using pandas
data = {'Threshold': threshold_list, 'PSNR': psnr_list}
df = pd.DataFrame(data)
print(df)
```

```
   Threshold        PSNR
0   0.000000  290.562041
1   0.104139   26.559582
2   0.208279   22.567492
3   0.312418   20.947842
4   0.416558   19.674507
5   0.520697   18.858531
6   0.624837   18.165151
7   0.728976   17.595277
8   0.833115   17.264851
9   0.937255   16.866715
```