DEPARTMENT OF ELECTRONIC AND TELECOMMUNICATION
UNIVERSITY OF MORATUWA

EN3551 Digital Signal Processing

This is offered as a "EN3551 Digital Signal Processing" module's partial completion.

Assignment 02 : Application of 2D-DCT for Image Compression

200686J : Vishagar A.

7^{th} of October, 2023

Abstract

This report deals with the explanation of the solutions for the given questions in the assignment 02 of the EN3551 module. The solutions are explained in a way that it is easy to understand and follow. The solutions are explained with the help of the code snippets and the results. We are mainly focussed on some real world problems like "Application of 2D-DCT for Image Compression". ***And I have used <u>python</u> as the programming language to find the solutions for the given problems.***

Contents

1 Introduction

In this assignmet, we are mainly focussed on a real world problem, **Image Compression** and we need to apply the 2D-DCT for the given image. And we need to find the compression ratio, percentage of zero and the PSNR value for the given images.

2 Dataset

The provided dataset contains **.mat** files of a image. We need to extract the necessary image data from the .mat file and use it for the further processing. I have assigned to use 3 images from the provided dataset and one image as per my wish.

3 Methadology

3.1 Extracting and Preparing the Data

As I mentioned earlier, we need to import the required .mat file and extract the image data from the .mat file using the relevant key name. Then if we wish we can visualize and see the imported image.

Then after importing the necessary image data, image array was divided in to blocks in a size of 8x8. And meanwhile as we are dealing with negative integer values also, I converted the default image data format from **np.uint8** to **np.int16**.

3.2 Implementation of DCT

Then after converting into blocks for better results the image data range was converted from **0 to 255** to **-128 to 127** , by subtracting 128 from each entry of the image matrix.

Then for each blocks which we have divided earlier, we need to apply the 2D-DCT. For that I have used the **dctn() function** in the **numpy** library.

And then the DCT coefficients were quantized using the given quantization matrix. Standard quantization matrix was given for a quality factor of 50 % and for the other factors the same matrix was multiplied by a scaling factor where the scaling factor can be derived using the given formula.

From each image block each individual element from the block was dived in to relevant element in the quantization matrix. And the derived matrix is the quantized matrix (named as S here).Then the quantized coefficients were encoded using the **zigzag scan** and the **run length encoding** methods.

3.3 Decompression and Visualization

After coding for visualization coded matrix will be again multiplied ny the quantization matrix (Each element in each block will be multiplied with relevent entry in the quantization marix in bitwise.) Then we apply inverse DCT using **idctn() function**. Then afterwards already deducted DC value will be compensated by adding the same DC value again. And by combining the small reconstructed blocks the image will be reconstructed. Then we can visualize the reconstructed image and compare it with the original image.

4 Implementation and Results

4.1 Image 01 : camera256

- Quality level 80

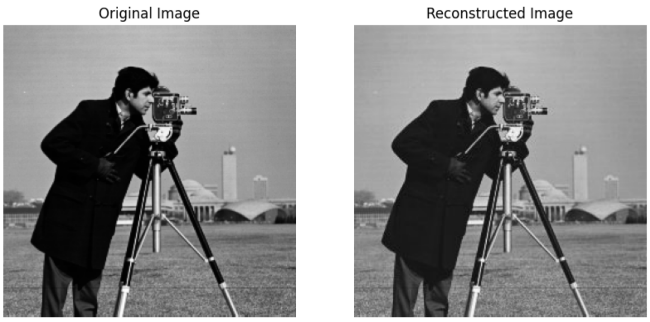

Figure 1: Provided sample file

  - Percentage of zeroes : 74.942 %
  - Peak SNR (in dB) : 35.7245 dB

- Quality level 35

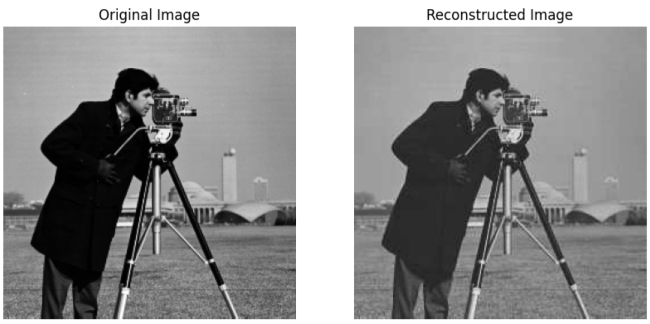

Figure 2: Provided sample file

  - Percentage of zeroes : 88.275 %
  - Peak SNR (in dB) : 30.3056 dB

- Quality level 15

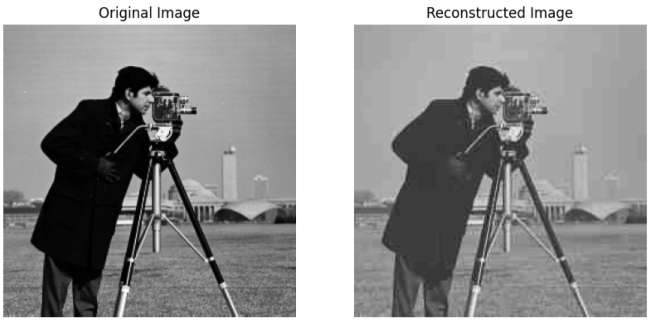

Figure 3: Provided sample file

  - Percentage of zeroes : 93.348 %
  - Peak SNR (in dB) : 27.536 dB

4.2 Image 02 : boat512

- Quality level 80

Figure 4: Provided sample file

  - Percentage of zeroes : 78.343 %
  - Peak SNR (in dB) : 38.0537 dB

- Quality level 35

Figure 5: Provided sample file

  - Percentage of zeroes : 89.334 %
  - Peak SNR (in dB) : 29.9607 dB

- Quality level 15

Figure 6: Provided sample file

  - Percentage of zeroes : 93.820 %
  - Peak SNR (in dB) : 60.1594 dB

4.3 Image 03 : peppers512

- Quality level 80

Figure 7: Provided sample file

  - Percentage of zeroes : 79.283 %
  - Peak SNR (in dB) : 36.9139 dB

- Quality level 35

Figure 8: Provided sample file

  - Percentage of zeroes : 91.356 %
  - Peak SNR (in dB) : 33.9170 dB

- Quality level 15

Figure 9: Provided sample file

  - Percentage of zeroes : 94.886 %
  - Peak SNR (in dB) : 31.5075 dB

4.4 Image 01 : Camera256

- Quality level 80

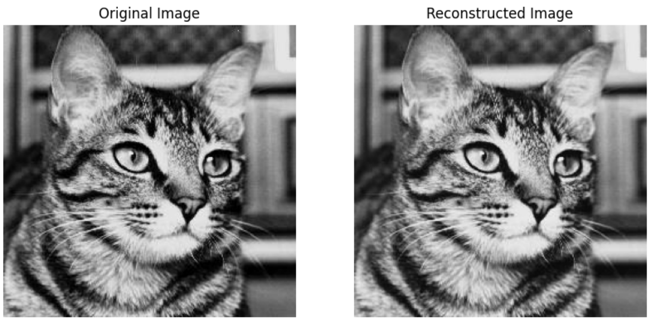

Figure 10: Provided sample file

  - Percentage of zeroes : 67.567 %
  - Peak SNR (in dB) : 40.7102 dB

- Quality level 35

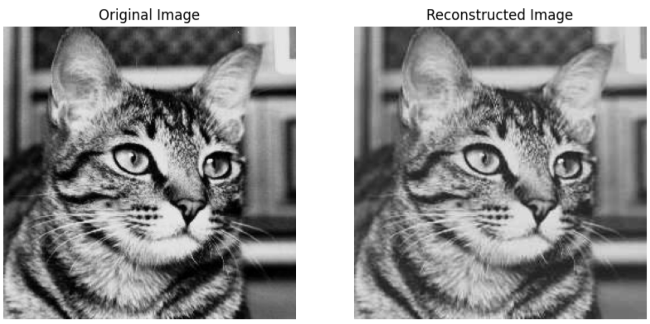

Figure 11: Provided sample file

  - Percentage of zeroes : 82.170 %
  - Peak SNR (in dB) : 28.4592 dB

- Quality level 15

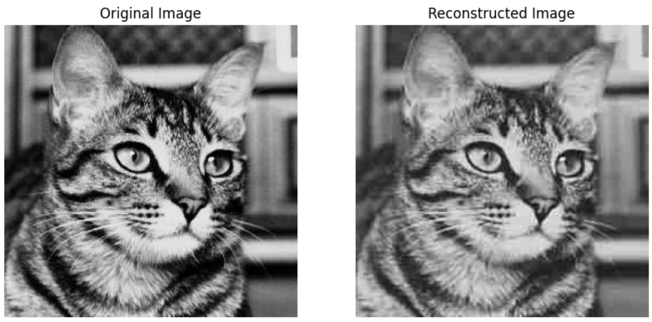

Figure 12: Provided sample file

  - Percentage of zeroes : 89.554 %
  - Peak SNR (in dB) : 25.7649 dB

5 Difficulty in Compression

6 References

- Numpy Documentation
- Preferred Text Book

7 Github Repository

Following is the link to my Github repository for this assignment.

Github/EN3551_Assignment_02

8 Appendix

8.1 Extracting and Preparing the Data

```
import numpy as np
import matplotlib.pyplot as plt
import scipy
import sympy

# Load the .mat file into a dictionary
mat_data = scipy.io.loadmat('SampleImages/camera256.mat')

# Extract the image data from the dictionary
image_data = mat_data['camera256']

image_data = image_data.astype(np.int16)

# Display the image using Matplotlib
plt.imshow(image_data, cmap='gray')
plt.axis('on')
plt.show()

# Separate image into 8x8 blocks
block_size = 8
blocks = []
for i in range(0, image_data.shape[0], block_size):
    for j in range(0, image_data.shape[1], block_size):
        blocks.append(image_data[i:i+block_size, j:j+block_size])

# Convert blocks to numpy array
blocks = np.array(blocks)
blocks = blocks - 128
```

8.2 DCT and Quantization

```
#Apply 2 dimensional DCT to each block

dct_blocks = []
for i in range(blocks.shape[0]):
    dct_blocks.append(scipy.fftpack.dctn(blocks[i], norm='ortho'))
dct_blocks = np.array(dct_blocks)

# Quantize the DCT coefficients

quantization_matrix = np.array([[16, 11, 10, 16, 24, 40, 51, 61],[12, 12, 14, 19, 26, 58, 60, 55],[14, 13, 16, 24, 40, 57, 69, 56],[14, 17, 22, 29, 51, 87, 80, 62],[18, 22, 37, 56, 68, 109, 103, 77],[24, 35, 55, 64, 81, 104, 113, 92],[49, 64, 78, 87, 103, 121, 120, 101],[72, 92, 95, 98, 112, 100, 103, 99]])

quality_factor = 5

if quality_factor > 50:
    scale = (100 - quality_factor) / 50
    quantization_matrix = quantization_matrix * scale
else:
    scale = quality_factor / 50
    quantization_matrix = quantization_matrix * scale

quantized_dct_blocks = []


for x in dct_blocks:
    s = np.zeros((8, 8))
    for i in range(8):
        for j in range(8):
            s[i][j] = round(x[i][j] / quantization_matrix[i][j])
    quantized_dct_blocks.append(s)

quantized_dct_blocks = np.array(quantized_dct_blocks)
```

8.3 Coding Schemes

```
# Coding the DC coefficients

dc_coefficients = []
for x in quantized_dct_blocks:
    dc_coefficients.append(x[0][0])

# differential pulse code modulation for DC coefficients

dpcm_dc_coefficients = []
dpcm_dc_coefficients.append(dc_coefficients[0])
for i in range(1, len(dc_coefficients)):
    dpcm_dc_coefficients.append(dc_coefficients[i] - dc_coefficients[i-1])

# run,level modulation for AC pairs

run_level_pairs_blocks = []

for x in quantized_dct_blocks:
    run = 0
    level = 0
    run_level_pairs = []
    for i in range(1, 8):
        for j in range(1, 8):
            if x[i][j] != 0:
                run_level_pairs.append((run, level))
                run = 0
                level = x[i][j]
            else:
                run += 1

    run_level_pairs_blocks.append(run_level_pairs)

    # Entropy coding for run-level pairs

entropy_coded_blocks = []

for x in run_level_pairs_blocks:
    entropy_coded_block = []
    for i in range(len(x)):
        if i == 0:
            entropy_coded_block.append((x[i
```

```
][0], x[i][1]))
        else:
            if x[i][0] == 0 and x[i][1] == 0:
                entropy_coded_block.append((15, 0))
            else:
                entropy_coded_block.append((x[i][0], x[i][1]))
    entropy_coded_blocks.append(entropy_coded_block)
```

8.4 Inverse DCT and re-construction

```
# inverse quantization
inverse_quantized_dct_blocks = []

for x in quantized_dct_blocks:
    s = np.zeros((8, 8))
    for i in range(8):
        for j in range(8):
            s[i][j] = x[i][j] * quantization_matrix[i][j]
    inverse_quantized_dct_blocks.append(s)

inverse_quantized_dct_blocks = np.array(inverse_quantized_dct_blocks)

# perform inverse DCT

inverse_dct_blocks = []

for x in inverse_quantized_dct_blocks:
    inverse_dct_blocks.append(scipy.fftpack.idctn(x, norm='ortho'))
inverse_dct_blocks = np.array(inverse_dct_blocks)

inverse_dct_blocks = inverse_dct_blocks + 128
inverse_dct_blocks = inverse_dct_blocks.astype(int)

num_blocks = int(np.sqrt(inverse_dct_blocks.shape[0]))
block_size = inverse_dct_blocks.shape[1]

# Reshape the blocks array
blocks_reshaped = inverse_dct_blocks.reshape((num_blocks, num_blocks, block_size, block_size))

# Initialize an empty image
reconstructed_image = np.zeros((num_blocks * block_size, num_blocks * block_size))

# Reconstruct the image from blocks
for i in range(num_blocks):
    for j in range(num_blocks):
        reconstructed_image[i*block_size:(i+1)*block_size, j*block_size:(j+1)*block_size] = blocks_reshaped[i, j, :, :]

# Visualize the images
fig,ax = plt.subplots(1,2)
ax[0].imshow(image_data, cmap='gray')
ax[0].set_title('Original Image')
ax[0].axis('off')
ax[1].imshow(reconstructed_image, cmap='gray')
ax[1].set_title('Reconstructed Image')
ax[1].axis('off')
plt.show()
```

8.5 Evaluation Metrics

```
# find percentage of zeroes
num_zeroes = 0
for x in quantized_dct_blocks:
    for i in range(8):
        for j in range(8):
            if x[i][j] == 0:
                num_zeroes += 1

percentage_zeroes = num_zeroes / (quantized_dct_blocks.shape[0] * 64) * 100
print("Percentage of Zeroes : ", percentage_zeroes)

# two dimensional mean square error
error_matrix = image_data - reconstructed_image
mse = np.mean(np.square(error_matrix))
print("Mean Square Error : ",mse)

# calculate the peak signal to noise ratio
psnr = 20 * np.log10((255) / mse)
print("Peak Signal to Noise Ratio (PSNR) : ",psnr)
```