

Signals Project Report
Computer Department - FECU

Muhammad Amr Hassan 9220728
Youssef Tarek Mehany 9220990

Presented To
Dr. Micheal Melek
Eng. Sayed Kamel

May 2024

Contents

1	Used Libraries For Project	2
1.1	Numpy	2
1.2	Matplotlib	2
1.3	Scipy	2
1.4	CV2	2
1.5	OS	2
2	Second Requirement	3
2.1	Reading Image File	3
3	Second Requirement	5
3.1	Compressing Image Using Discrete Cosines Transform	5
4	Third Requirement	6
4.1	Comparing the sizes of all images	6
5	Fourth Requirement	7
5.1	Finding Output Using Convolution	7
6	Fifth Requirement	10
6.1	Computing PSNR	10
7	Sixth Requirement	11
7.1	Analysing the relation between PSNR and Quality	11
A	Appendices	12
A.1	Project's Code	12
B	References	14

List of Figures

2.1	Red component of Image.	3
2.2	Green component of Image.	4
2.3	Blue component of Image.	4
5.1	Decompressed image using $m = 1$	8
5.2	Decompressed image using $m = 2$	8
5.3	Decompressed image using $m = 3$	9
5.4	Decompressed image using $m = 4$	9
7.1	The Relation between m and PSNR.	11

Chapter 1

Used Libraries For Project

1.1 Numpy

Famous Library in python that deals with mathematical operations on vectors and scalars.

1.2 Matplotlib

Famous Library in python for plotting data.

1.3 Scipy

Famous Library in python that deals with operations on signals like DCT and IDCT.

1.4 CV2

Famous Library in python that deals with images and processing it.

1.5 OS

Famous Library in python that deals with operations in operating system like creating directories and files.

```
1 ##### All used Libraries #####
2
3 import numpy as np
4 import matplotlib.pyplot as plt
5 import cv2
6 import sys
7 import os
8 from scipy.fft import dct,idct
9 from math import log10
10
11 #####
```

Listing 1.1: Importing the used libraries in project.

Chapter 2

Second Requirement

2.1 Reading Image File

From Reading The file, we can get the image data as a 3D array. The first dimension is the row of the pixel, The second one is the column of the pixel and the third is the intensity of the color RGB which varies from 0 to 255.

```
1 def getImageComponents(inputImage):
2     os.makedirs("Image Components", exist_ok=True)
3     colors = ["Reds", "Greens", "Blues"]
4     for i in range(3):
5         plt.figure(figsize=(14, 6))
6         plt.imshow(inputImage[:, :, 2-i], cmap=colors[i])
7         plt.axis("off")
8         plt.colorbar()
9         plt.savefig(f"Image Components/a_{colors[i]}[:-1].lower().png")
10 inputImageArray = cv2.imread("image1.png")
11 getImageComponents(inputImageArray)
```

Listing 2.1: Getting Image Data.

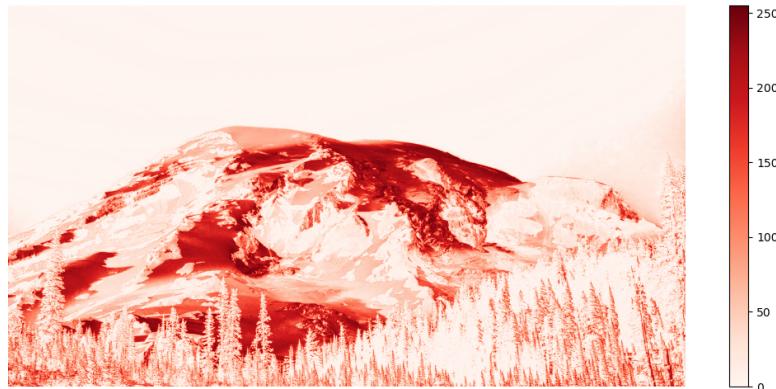


Figure 2.1: Red component of Image.

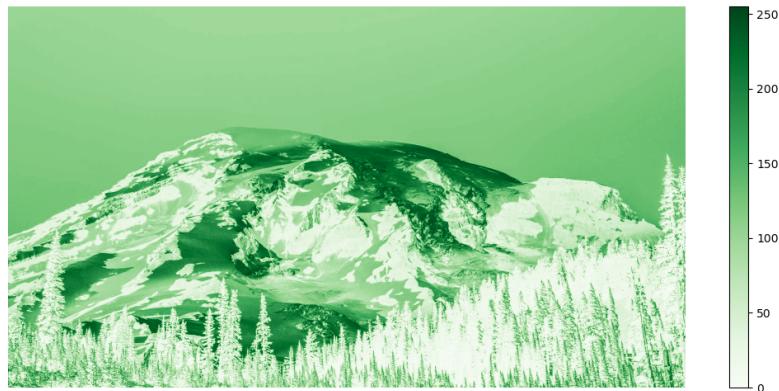


Figure 2.2: Green component of Image.

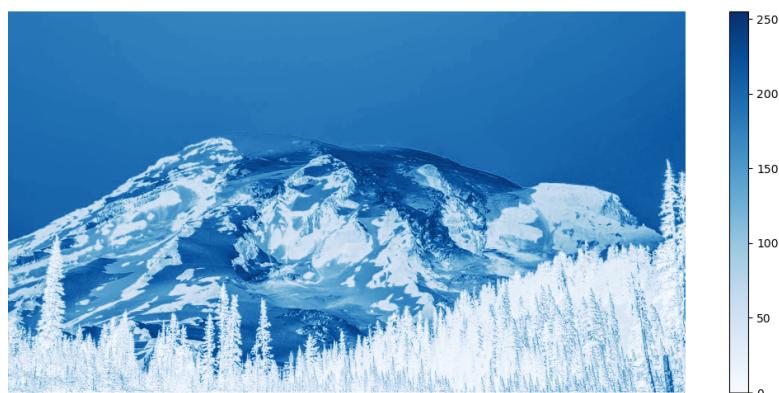


Figure 2.3: Blue component of Image.

Chapter 3

Second Requirement

3.1 Compressing Image Using Discrete Cosines Transform

As stated in the project document , We deal with each color channel of image matrix as group of blocks. Each block is 8x8 pixels.We apply to each block the DCT such that we get the lower frequencies that has the higher impact on eye and then we can neglect the other frequencies and just store the important ones.

Discrete Cosines Transform Equation

$$X [p, q] = \alpha_p \alpha_q \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} x[m, n] \cos \frac{\pi (2m + 1)p}{2M} \cos \frac{\pi (2n + 1)q}{2N}$$
$$\alpha_p = \begin{cases} \frac{1}{\sqrt{M}} & p = 0 \\ \sqrt{\frac{2}{M}} & 1 \leq p \leq M - 1 \end{cases} \quad \alpha_q = \begin{cases} \frac{1}{\sqrt{N}} & q = 0 \\ \sqrt{\frac{2}{N}} & 1 \leq q \leq N - 1 \end{cases}$$

```
1 def compressImage(inputImage,m):
2     blocksShape = (inputImage.shape[0]//8,inputImage.shape[1]//8)
3     compressedImage = np.zeros((m*blocksShape[0],m*blocksShape[1],3))
4     for channel in range(3):
5         for row in range(blocksShape[0]):
6             for col in range(blocksShape[1]):
7                 processedBlock = inputImage[8*row:8*(row+1),8*col:8*(col+1),channel]
8                 dctBlock = DCT(processedBlock)
9                 dctRetainedBlock = dctBlock[:m,:m]
10                compressedImage[m*row:m*(row+1),m*col:m*(col+1),channel] =
11                    dctRetainedBlock
12    return compressedImage
```

Listing 3.1: Code For Compressing Image using DCT.

Chapter 4

Third Requirement

4.1 Comparing the sizes of all images

Image	Size (MB)
Original	5.93
Compressed with $m = 1$	0.19
Compressed with $m = 2$	0.74
Compressed with $m = 3$	1.67
Compressed with $m = 4$	2.97

Chapter 5

Fourth Requirement

5.1 Finding Output Using Convolution

To get the original image we decompress the compressed image using IDCT but on the retained $m \times m$ block with ignoring the other values by setting them to zeros.

Inverse Discrete Cosines Transform

$$x[m, n] = \sum_{p=0}^{M-1} \sum_{q=0}^{N-1} \alpha_p \alpha_q X[p, q] \cos \frac{\pi(2m+1)p}{2M} \cos \frac{\pi(2n+1)q}{2N}$$
$$\alpha_p = \begin{cases} \frac{1}{\sqrt{M}} & p = 0 \\ \sqrt{\frac{2}{M}} & 1 \leq p \leq M-1 \end{cases} \quad \alpha_q = \begin{cases} \frac{1}{\sqrt{N}} & q = 0 \\ \sqrt{\frac{2}{N}} & 1 \leq q \leq N-1 \end{cases}$$

```
1 def decompressImage(compressedImage, m):
2     blocksShape = (compressedImage.shape[0]//m, compressedImage.shape[1]//m)
3     decompressedImage = np.zeros((8*blocksShape[0], 8*blocksShape[1], 3))
4     for channel in range(3):
5         for row in range(blocksShape[0]):
6             for col in range(blocksShape[1]):
7                 processedBlock = np.zeros((8, 8))
8                 processedBlock[:m, :m] = compressedImage[m*row:m*(row+1), m*col:m*(col+1), channel]
9                 idctBlock = IDCT(processedBlock)
10                decompressedImage[8*row:8*(row+1), 8*col:8*(col+1), channel] =
11                    idctBlock
12    return decompressedImage
```

Listing 5.1: Code For Decompressing The Compressed Image.

We will show below how increasing m can effectively change the quality of the image.



Figure 5.1: Decompressed image using $m = 1$.



Figure 5.2: Decompressed image using $m = 2$.



Figure 5.3: Decompressed image using $m = 3$.



Figure 5.4: Decompressed image using $m = 4$.

Chapter 6

Fifth Requirement

6.1 Computing PSNR

The quality of the decompressed image is measured using the Peak Signal-to-Noise Ratio (PSNR), which is defined by

$$\text{PSNR} = 10 \log_{10} \left(\frac{\text{peak}^2}{\text{MSE}} \right)$$

Where peak is the max value of pixel data type and MSE is defined by

$$\text{MSE} = \frac{1}{3MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} \sum_{c=0}^2 (x[m, n, c] - \hat{x}[m, n, c])^2$$

Where \hat{x} is decompressed image matrix and x is the original image matrix.

m	PSNR (dB)
1	31.51
2	32.08
3	32.57
4	33.21

```
1 def calculatePSNR(inputImage, decompressedImage):
2     SE = (inputImage-decompressedImage)**2
3     MSE = np.mean(SE)
4     peak = 255
5     PSNR = 10*log10(peak**2/MSE)
6     return PSNR
```

Listing 6.1: Code For Calculating PSNR.

Chapter 7

Sixth Requirement

7.1 Analysing the relation between PSNR and Quality

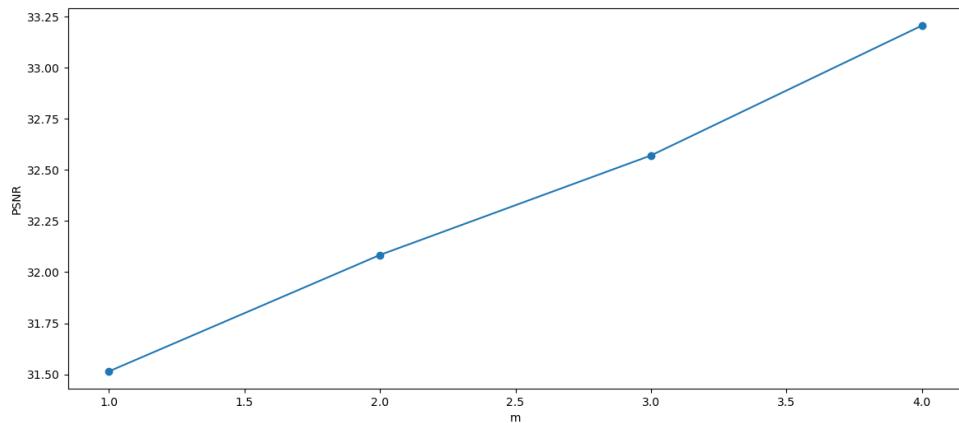


Figure 7.1: The Relation between m and PSNR.

It's obvious that by increasing m we take more effective frequencies, thus we enhance its quality more and more and increasing the PSNR also. Thus the PSNR is directly proportional with m .

Appendix A

Appendices

A.1 Project's Code

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 import cv2
4 import sys
5 import os
6 from scipy.fft import dct,idct
7 from math import log10
8
9 def getImageComponents(inputImage):
10     os.makedirs("Image Components", exist_ok=True)
11     colors = ["Reds","Greens","Blues"]
12     for i in range(3):
13         plt.figure(figsize=(14, 6))
14         plt.imshow(inputImage[:, :, 2-i], cmap=colors[i])
15         plt.axis("off")
16         plt.colorbar()
17         plt.savefig(f"Image Components/a_{colors[i]}[:-1].lower().png")
18 def DCT(processedBlock):
19     return dct(dct(processedBlock.T, norm='ortho').T, norm='ortho')
20 def IDCT(processedBlock):
21     return idct(idct(processedBlock.T, norm='ortho').T, norm='ortho')
22 def compressImage(inputImage,m):
23     blocksShape = (inputImage.shape[0]//8,inputImage.shape[1]//8)
24     compressedImage = np.zeros((m*blocksShape[0],m*blocksShape[1],3))
25     for channel in range(3):
26         for row in range(blocksShape[0]):
27             for col in range(blocksShape[1]):
28                 processedBlock = inputImage[8*row:8*(row+1),8*col:8*(col+1),channel]
29                 dctBlock = DCT(processedBlock)
30                 dctRetainedBlock = dctBlock[:m,:m]
31                 compressedImage[m*row:m*(row+1),m*col:m*(col+1),channel] =
32                     dctRetainedBlock
33     return compressedImage
34 def decompressImage(compressedImage ,m):
35     blocksShape = (compressedImage .shape[0]//m,compressedImage .shape[1]//m)
36     decompressedImage = np.zeros((8*blocksShape [0],8*blocksShape [1],3))
37     for channel in range(3):
38         for row in range(blocksShape [0]):
39             for col in range(blocksShape [1]):
40                 processedBlock = np.zeros((8,8))
41                 processedBlock[:m,:m] = compressedImage [m*row:m*(row+1),m*col:m*(col
42                     +1),channel]
43                 idctBlock = IDCT(processedBlock)
44                 decompressedImage [8*row:8*(row+1),8*col:8*(col+1),channel] =
45                     idctBlock
46     return decompressedImage
47 def calculatePSNR(inputImage ,decompressedImage):
48     SE = (inputImage-decompressedImage)**2
49     MSE = np.mean(SE)
50     peak = 255
51     PSNR = 10*log10(peak**2/MSE)
52     return PSNR
53 def writeImageInfo(file, sizeInBytes, inputImage ,decompressedImage ,m):
54     file.write(f"\n-----For m = {m}
```

```

52 }-----\n\n")
53     file.write(f"\t\t\t\t\t\t\tCompressed Image Size : {sizeInBytes/(1024)**2:.2f} MB\n"
54     ")
55 PSNR = calculatePSNR(inputImage,decompressedImage)
56     file.write(f"\t\t\t\t\t\t\tPSNR : {PSNR:.2f} dB\n\n")
57     file.write("\n\n")
58 return PSNR
59 def plotPSNRS(allMs,PSNRS):
60     plt.figure(figsize=(14, 6))
61     plt.plot(allMs,PSNRS,marker='o')
62     plt.xlabel("m")
63     plt.ylabel("PSNR")
64     plt.savefig("PSNR_plot.png")
65 def Project():
66     with open('sizes.txt', 'w') as file:
67         # Write to the file
68         inputImageArray = cv2.imread("image1.png")
69         getImageComponents(inputImageArray)
70         os.makedirs("Decompressed Images", exist_ok=True)
71         os.makedirs("Compressed Images", exist_ok=True)
72         file.write(f"Original Image Size : {sys.getsizeof(inputImageArray)/(1024**2):.2f} MB\n")
73         PSNRS = np.zeros(4)
74         for m in range(1,5):
75             compressedImage = compressImage(inputImageArray,m)
76             np.save(f"Compressed Images/compressedImage_m{m}",compressedImage)
77             decompressedImage = decompressImage(compressedImage,m)
78             PSNR = writeImageInfo(file,sys.getsizeof(decompressedImage.astype(np.
79 float16)),inputImageArray,decompressedImage.astype(np.uint8),m)
80             PSNRS[m-1]=PSNR
81             cv2.imwrite(f"Decompressed Images/decompressedImage_m{m}.png",
82             decompressedImage)
83             allMs = np.linspace(1,4,4,dtype=np.uint8)
84             plotPSNRS(allMs,PSNRS)
85     Project()

```

Listing A.1: Code For Compressing Image using DCT.

Appendix B

References

1. [Matlab DCT explaination](#)
2. [Wikipedia DCT explaination](#)