PROJECT REPORT – PART 1

Information Theory - CIE 425

BY:
Alaa Hesham 201500638
Anhar Hassan 201601171
Rahma Hassan 201501344
Sara El besomy 201601849

Zewail city of science & technology

Results of DCT Compression:

- Original Image:

The size of the image is 576x576 pixels.

- Compressed Image After 8x8 quantization table 1:

- Compressed Image after 8x8 quantization table 2:

- Compressed Image after 16x16 quantization table 1:

- Compressed Image after 16x16 table 2:

We notice that the compression of the quantization table 2 is much higher than the compression of table 1 due to the large values in the table 2 that reduces more pixels to zeroes and cause a large loss in the floating-point numbers that result from DCT matrix.

In the following table, we will compare the four versions of compressed image in respect to 3 different criteria.

	Amount of compression	Number of floating-point operations	Quality of the image (MSE)
Image 1 (8x8 – table 1)	1.13	515.584744×10^6	197.964656876929
Image 2 (8x8 – table 2)	1.92	515.584744×10^6	1080.002212336034
Image 3 (16x16 – table 1)	0.9	1.025976864×10^9	42.71117862654321
Image 4 (16x16 – table 2)	1.68	1.025976864×10^9	894.5382245852624

It is obvious that the compressed images from 16x16 tables are higher in quality than 8x8 tables images but this high quality comes with a huge number of computations and this is a noticeable disadvantage. We also note that Image three was extended not compressed after DCT. We think that the values in the quantization table might be too low to do compression.

How we measured the complexity...?

- **Compression ratio:**
  We used the built-in function of Huffman in MATLAB and then get the ratio between the number of bits of the original image and the number of bits of compressed image from Huffman.

- **Number of floating-point operations:**
  We calculate them manually.

- **Mean Square Error quality metric:**
  We used this function:

```
# calculate the MSE:
import numpy as np
def Calc_MSE(I_original,I_compressed):

  difference = I_original - I_compressed
  square_diff = difference**2
  square_diff = np.matrix(square_diff)
  summer = square_diff.sum()
  MSE = summer/(576*576)

  return MSE
```

Results of DWT Compression:

First stage DWT:

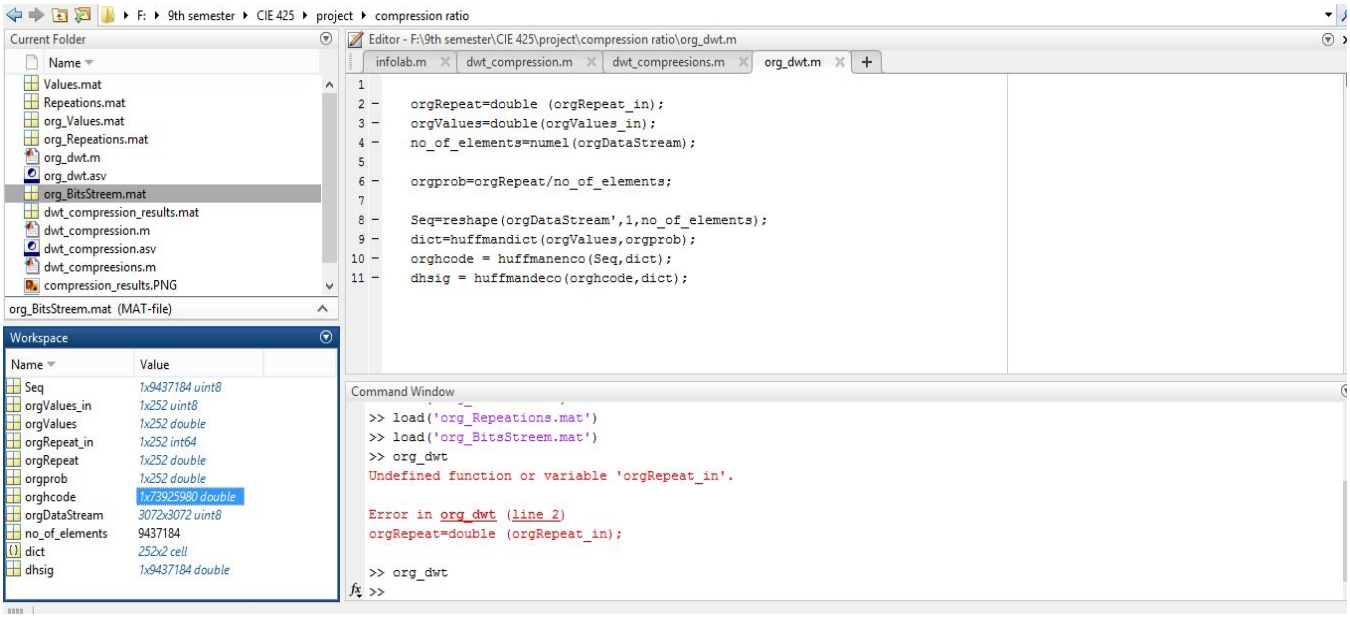
- Original Image:
  The size of the image is (3072, 4096) pixels. We have then cropped it to make it square image (3072,3072) to make calculations easier .

- Compressed Image after performing one stage discrete wavelet transform with the quantization scheme: Divide LL image by scalar (4) while ignoring other images HL, LH, and HH. It is a kind of severe quantization as if dividing HL, LH, and HH by infinity so the result is zero adding zeros to LL image will not change the results.

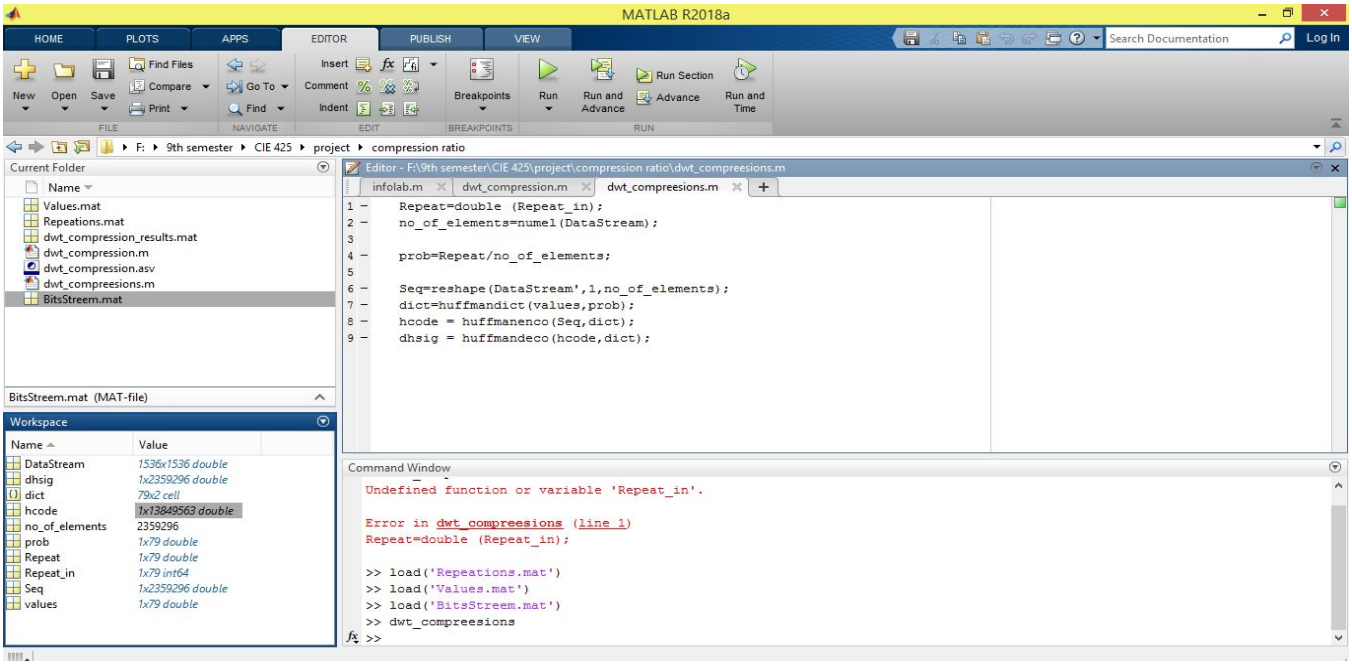
Compressed Image after performing one stage discrete wavelet transform

- Results:

Original bit stream was 73,925,980 bit using Huffman encoder. The compressed stream is 13,849,563 bit using Huffman encoder. ***The compression ratio is 5.337 less times.***



Original bit stream



Compressed bit stream

First stage DWT with variations in quantization:

- Compressed Image after performing one stage discrete wavelet transform

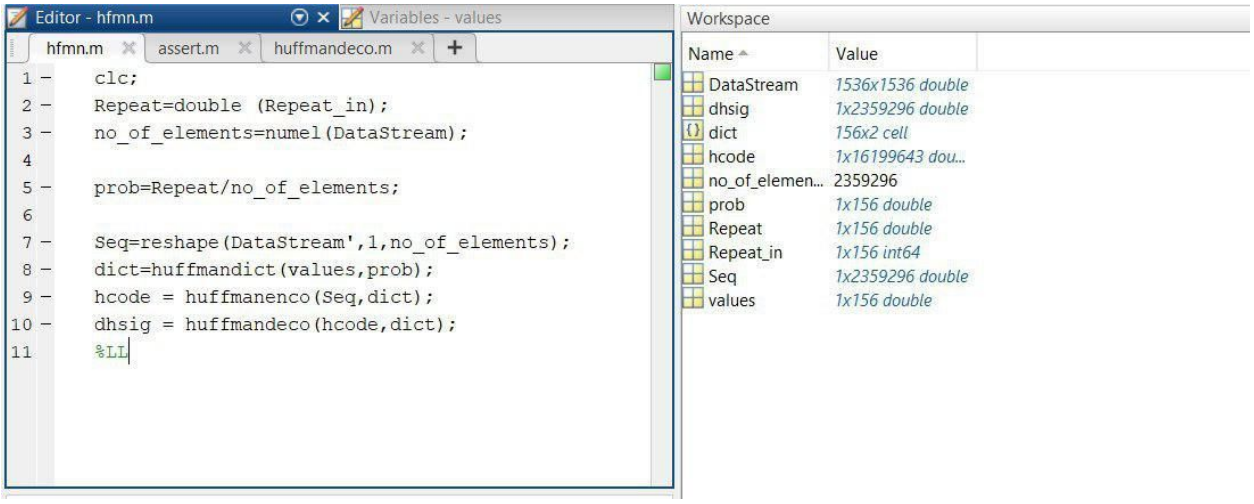
With the following quantization scheme: Divide LL image by scalar (2). While Dividing other images LH by scalar of (8) and HH by scalar of (16).

I have neglected HL to achieve better compression ratio especially that its effect is negligible to the image besides trying new things.

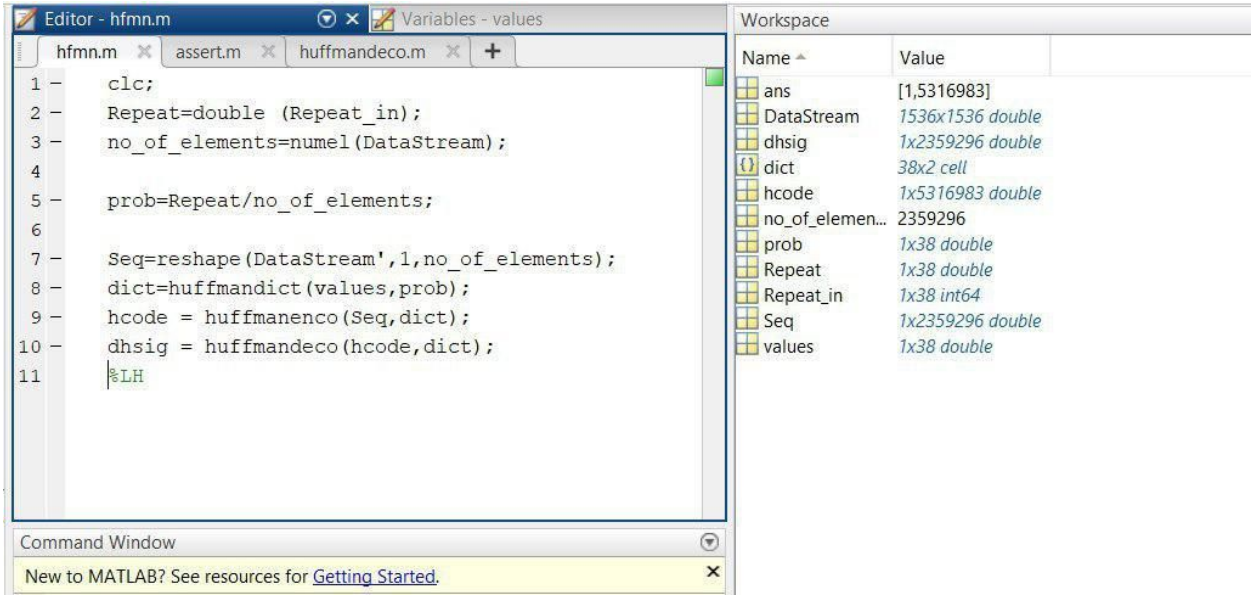
Compressed Image after performing one stage discrete wavelet transform

- Results :

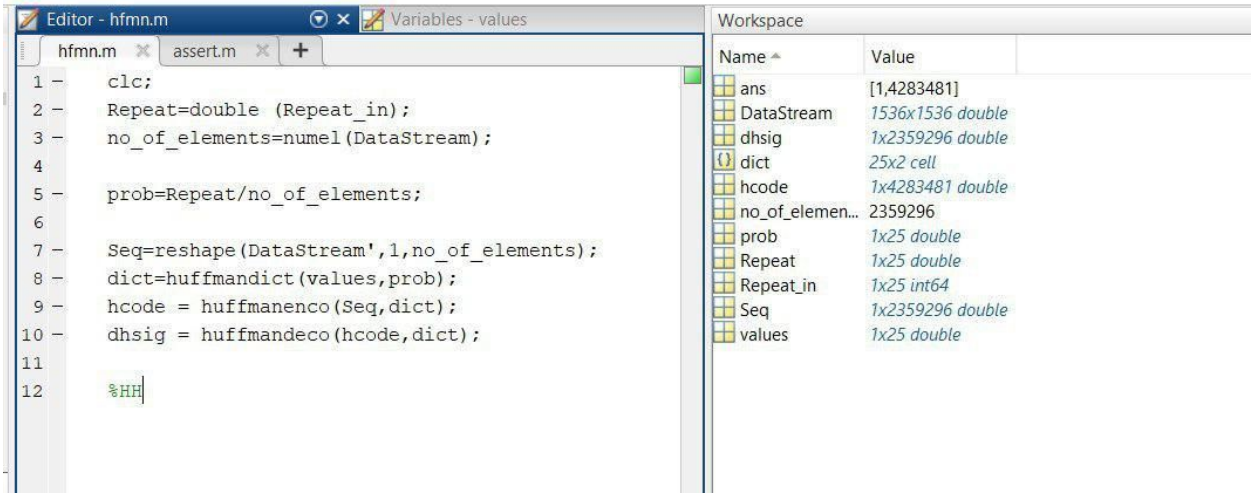
The first stage discrete wavelet transform LL, LH, HH have the following bit stream size 16,199,643 – 5,316,983----4,283,481---respectively .So the total bit stream is 25,800,107 .By comparing it with the original bit stream which is 73,925,980 bit. ***The compression ratio is 2.8653.***



LL



LH


Editor - hfmn.m
Variables - values
hfmn.m
assert.m
1 clc;
2 Repeat=double (Repeat_in);
3 no_of_elements=numel(DataStream);
4
5 prob=Repeat/no_of_elements;
6
7 Seq=reshape(DataStream',1,no_of_elements);
8 dict=huffmandict(values,prob);
9 hcode = huffmanenco(Seq,dict);
10 dhsig = huffmandeco(hcode,dict);
11
12 %HH
Workspace
Name
Value
ans [1,4283481]
DataStream 1536x1536 double
dhsig 1x2359296 double
dict 25x2 cell
hcode 1x4283481 double
no_of_elemen... 2359296
prob 1x25 double
Repeat 1x25 double
Repeat_in 1x25 int64
Seq 1x2359296 double
values 1x25 double

HH

Second Stage DWT:

- Compressed Image after performing two stage discrete wavelet transform

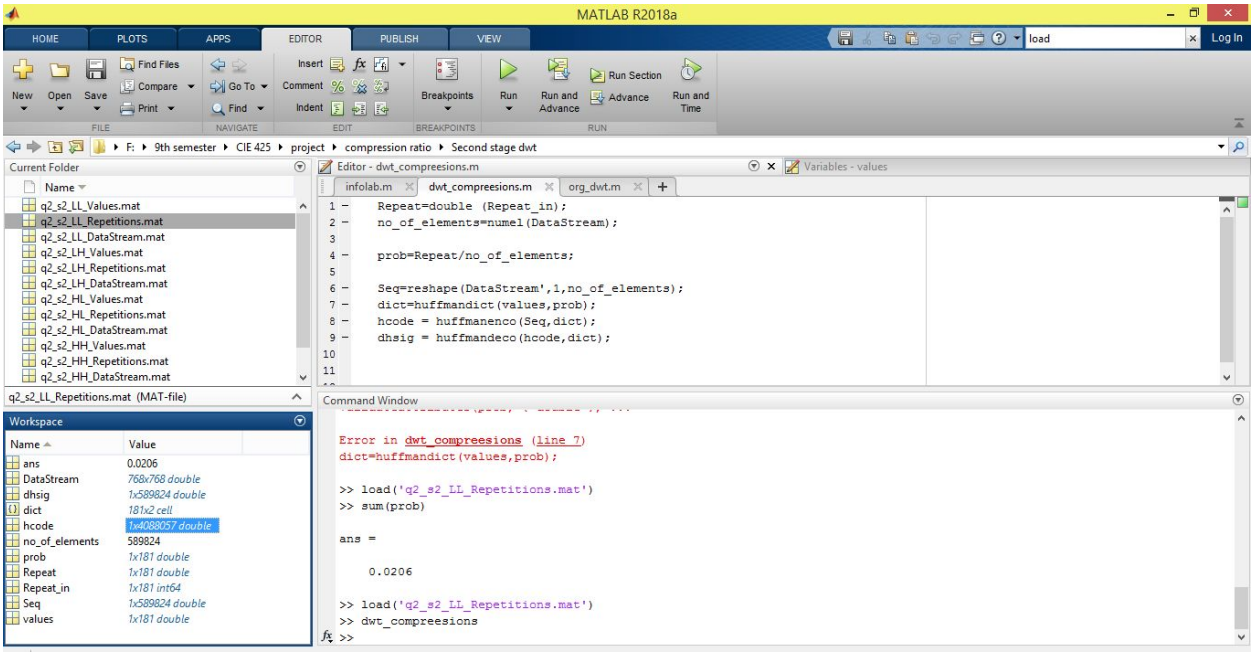
With the following quantization scheme: Divide LL image by scalar (2). While Dividing other images HL, LH by scalar of (4) and HH by scalar of (8).

Note that: The input of 2nd stage is the output of first stage which was quantized by dividing by scalar (4)

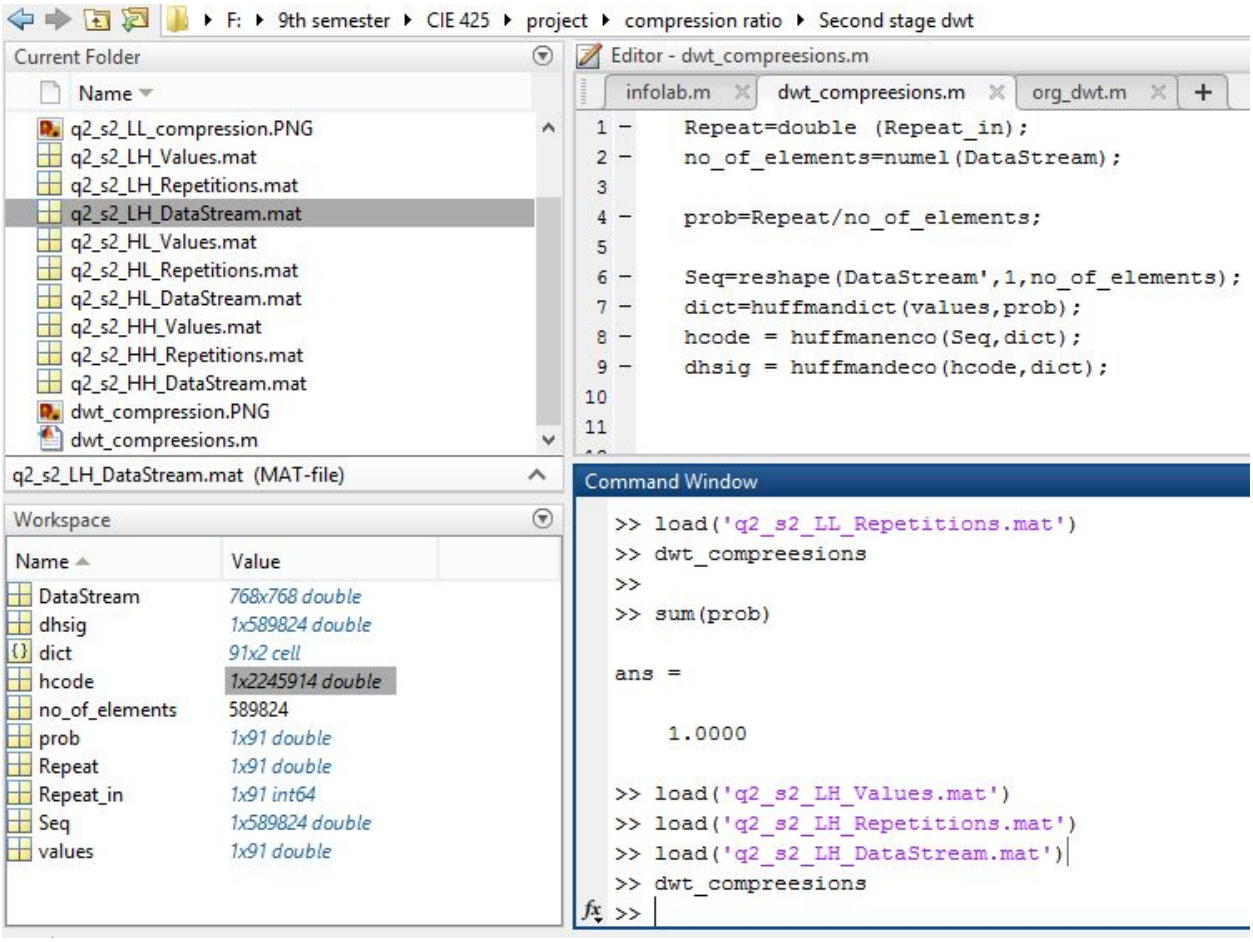
Compressed Image after performing two stage discrete wavelet transform

Results:

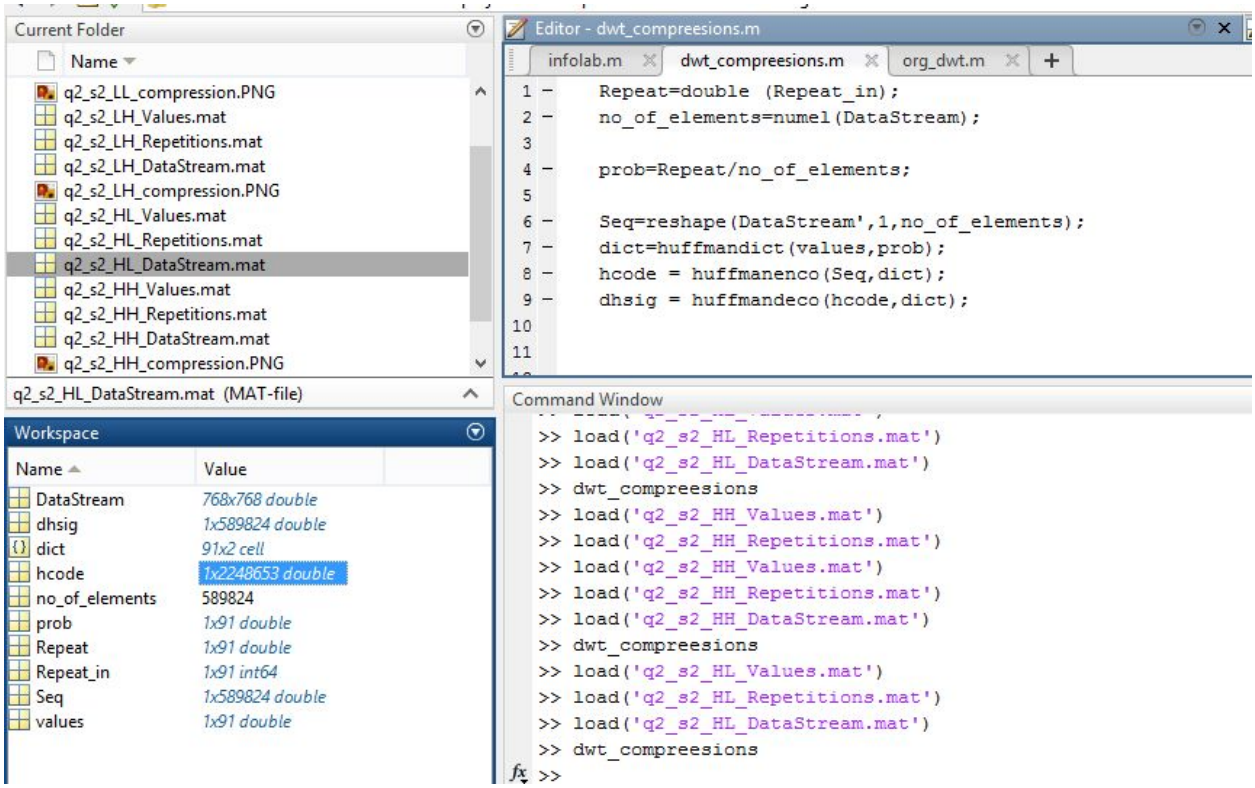
The second stage discrete wavelet transform LL, LH, HL, HH have the following bit stream size 4,088,057 – 2,245,914-2,248,653-2,369,401 respectively .So the total bit stream is 10,952,025 .By comparing it with the original bit stream which is 73,925,980 bit. ***The compression ratio is 6.75***



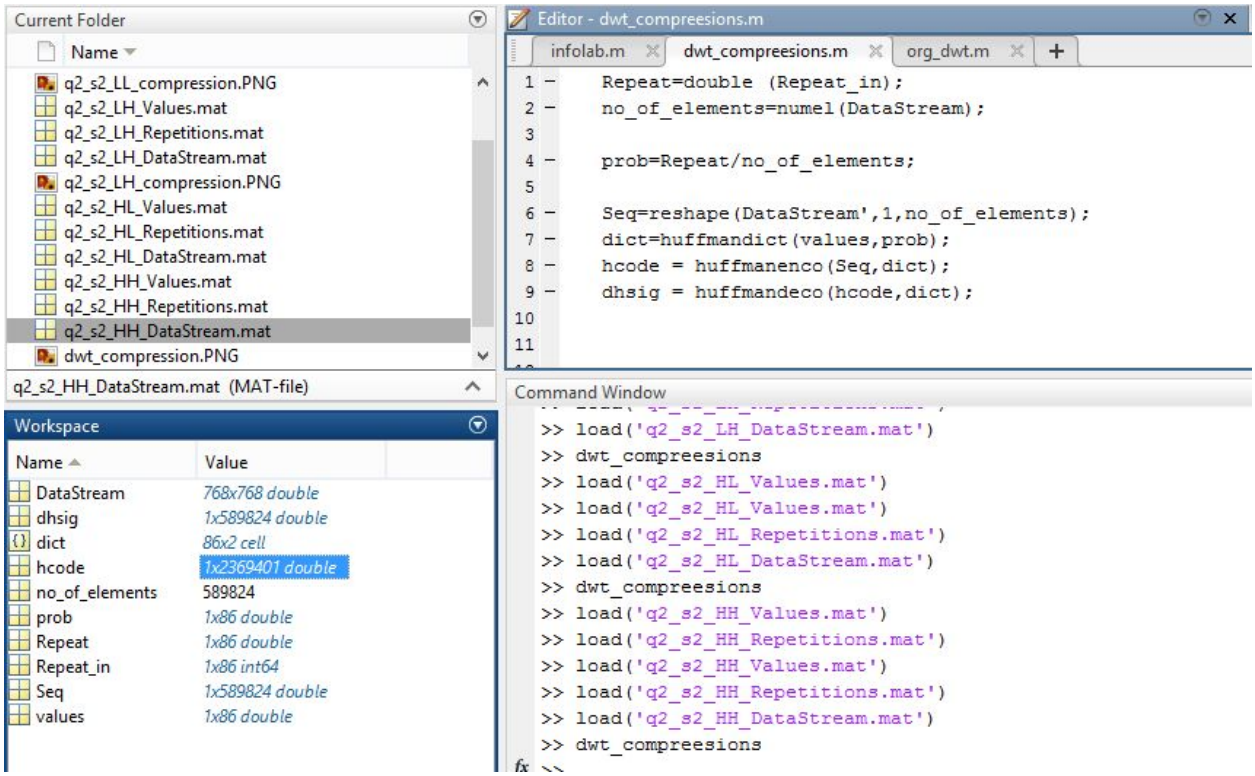
2nd stage dwt LL



2nd stage dwt LH



2nd stage dwt HL



2nd stage dwt HH

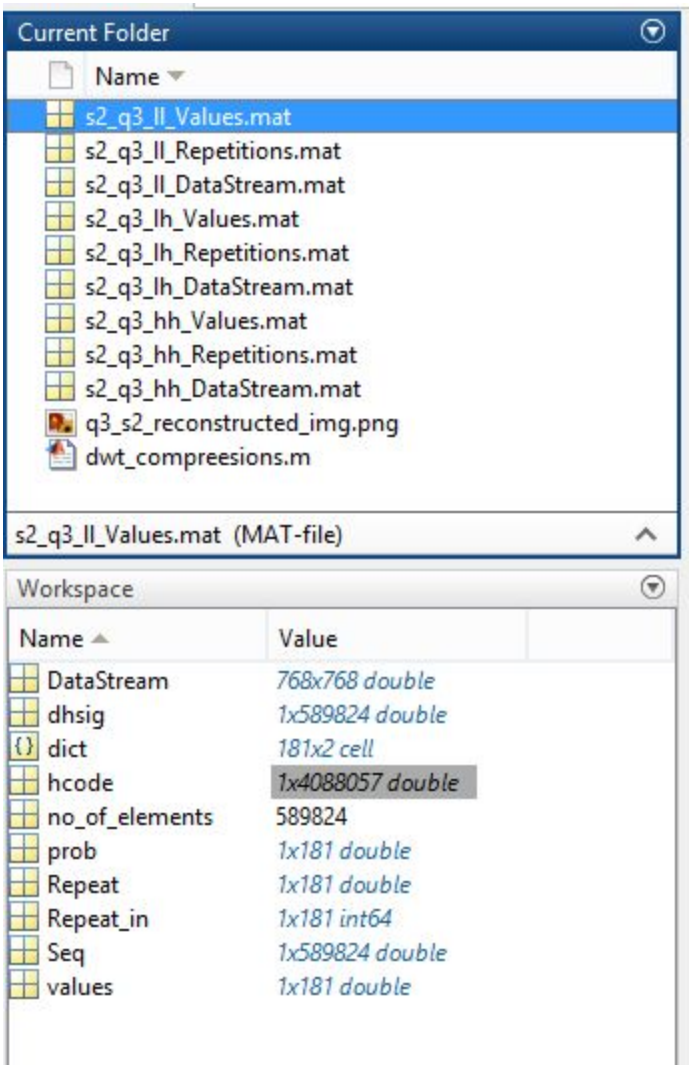
Second stage DWT with Third quantization scheme:

- Compressed image after performing two stage discrete wavelet transform .The main idea was to use LL image from first stage as input to the second stage .LL image's quantization has been performed through division by scalar (2).Quantization in the second stage has been performed by dividing LL_2 image by scalar of (2),LH_2 by scalar of (8), and HH_2 by scalar of (16).

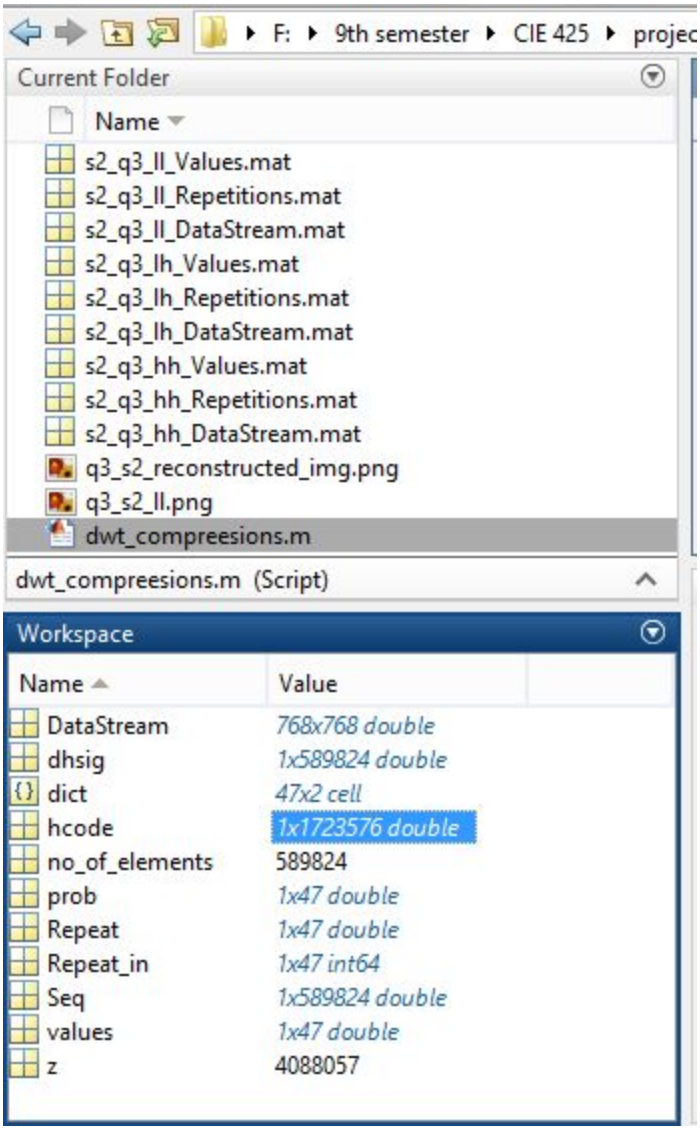
Compressed Image after performing two stage discrete wavelet transform with third quantization scheme

Results:

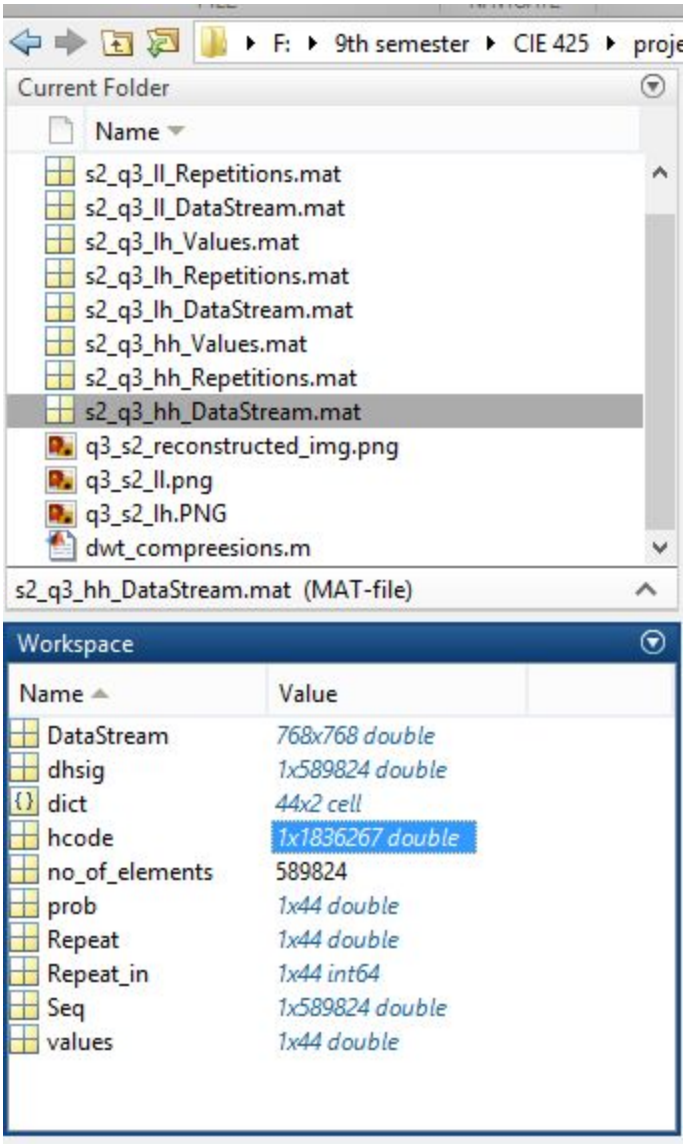
The second stage discrete wavelet transform LL, LH, HH have the following bit stream size 4,088,057 – 1,723,576-1,836,267 respectively .So the total bit stream is 7,647,900 .By comparing it with the original bit stream which is 73,925,980 bit. ***The compression ratio is 9.662.***



LL image


F: ▸ 9th semester ▸ CIE 425 ▸ projec
Current Folder
Name
s2_q3_ll_Values.mat
s2_q3_ll_Repetitions.mat
s2_q3_ll_DataStream.mat
s2_q3_lh_Values.mat
s2_q3_lh_Repetitions.mat
s2_q3_lh_DataStream.mat
s2_q3_hh_Values.mat
s2_q3_hh_Repetitions.mat
s2_q3_hh_DataStream.mat
q3_s2_reconstructed_img.png
q3_s2_ll.png
dwt_compreesions.m
dwt_compreesions.m (Script)
Workspace
Name
Value
DataStream 768x768 double
dhsig 1x589824 double
dict 47x2 cell
hcode 1x1723576 double
no_of_elements 589824
prob 1x47 double
Repeat 1x47 double
Repeat_in 1x47 int64
Seq 1x589824 double
values 1x47 double
z 4088057

LH


F: ▸ 9th semester ▸ CIE 425 ▸ proje
Current Folder
Name
s2_q3_ll_Repetitions.mat
s2_q3_ll_DataStream.mat
s2_q3_lh_Values.mat
s2_q3_lh_Repetitions.mat
s2_q3_lh_DataStream.mat
s2_q3_hh_Values.mat
s2_q3_hh_Repetitions.mat
s2_q3_hh_DataStream.mat
q3_s2_reconstructed_img.png
q3_s2_ll.png
q3_s2_lh.PNG
dwt_compreesions.m
s2_q3_hh_DataStream.mat (MAT-file)
Workspace
Name
Value
DataStream 768x768 double
dhsig 1x589824 double
dict 44x2 cell
hcode 1x1836267 double
no_of_elements 589824
prob 1x44 double
Repeat 1x44 double
Repeat_in 1x44 int64
Seq 1x589824 double
values 1x44 double

HH

Summary:

Three quantization schemes have been used:

- First one was to divide LL image by scalar of 4 while neglecting LH, HL, HH images as if they become zeros, division by infinity.
- Second one was to divide LL image by scalar of 2, divide LH, HL by scalar of 4, and divide HH by scalar of 8.
- Third one was to divide LL image by scalar of 2, LH by scalar of 8, HH image by scalar of 16.

Variations	**Number of stages**	**1ST stage Quantization**	**2nd Stage Quantization**	**Compression ratio**
1st constructed image	1	1	-	**5.3337**
2nd Constructed image	1	3	-	**2.865**
3rd Constructed image	2	1	2	**6.75**
4th Constructed image	2	3	3	**9.662**

Team Members and their roles:

Member Name	Role
Alaa Hesham	Implementing DWT,IDWT algorithm , report rewriting from page 5:23
Anhar Hassan	Implementing run length code its integration with DCT , code to calculate compression ratio with Huffman (Integration between python and Matlab) .
Rahma Hassan	Huffman implementation with python (working on it) .
Sara Elbasomy	Implementing DCT,IDCT algorithm , report rewriting from page 1:5

Notes on submission:

We have written both DCT, IDCT, DWT, IDWT using python. To be able to work in a parallel way without waiting for Huffman code to be finished, we developed a code to take outputs from python notebooks to be processed on Huffman built in function in Matlab to be able to compute compression ratios. Especially that Huffman code with python is still in progress.

Notes on Late submission:

All the submitted project except DWT and its related parts algorithm was totally finished by the deadline Tuesday 5/11.

For DWT algorithm and IDWT, the algorithm was totally implemented and the first constructed image (first variation) has been attained on Sunday morning 3/11. I, Alaa, have sent to my colleagues over internet and show it to Dr Mahmoud on Sunday lecture so I can easily prove that.

On Sunday evening, the internet connection was really bad so I could not work on Colab(online python editor) , and since the deadline has been extended , I choose to complete the variations in DWT , and quantization tables on Monday .

Due to the accident of telephone cables robbery, the internet connection was totally down in Ferdous area for more than *24 hours* .
The remaining day was Tuesday, I was reassured that I can small part to finish and will be able to meet deadline yet the Colab (the most common online python editor) went down for me so I could not finish the remaining part .

To overcome this technical issue , I have used *Jupyter* (another python editor) , so I had to refine my code to work on it , install libraries such as OpenCV, and other technicalities due to poor image plotting capabilities on Jupyter unlike Colab (such as the absence of cv2_imshow) . This operation was extremely time consuming, it has taken about *40 hours* of continuous work without sleep nor attending lectures). Finally Jupyter is now working right, and I managed to finish implementing variations on DWT.