Program to encode images lena.raw, baboon.raw and pepper.raw using Discrete Cosine Transform.

Author: Siffi Singh
Student ID: 0660819
Dated: 03/11/2017

The file main_part1.m, main_part2.m and main_part3.m contains the MATLAB code for encoding the images using Discrete Cosine Transform.

Here in this report, I will be explaining:

I. Steps to run this file
II. Explanation of my code logic
III. Inferences from result
IV. Conclusion

I. Steps to run the file:

The Process:

a. The image is broken into 8*8 blocks of pixels.
b. Working from left to right, top to bottom, the DCT is applied to each block.
c. Each block is compressed through quantization.
d. The array of compressed blocks that constitute blocks that constitute the image is sorted in a drastically reduced amount of space.
e. When desired, the image is reconstructed through decompression, a process that uses the Inverse Discrete Cosine Transform.

1. There are eight MATLAB files in the mail:
    a. Energy_compaction.m
    b. Main_part1.m
    c. Zonal.m
    d. Main_part2.m
    e. Threshold.m
    f. Main_part3.m
    g. Zigzag.m
    h. Invzigzag.m

2. I will explain step by step what each file does.

2.1. Energy_compaction.m

Energy Compaction, i.e. part-1, has to be run using **main_part1.m** file, to run this file, you need to have the **energy_compaction.m** file in the same folder and the three images file **lena.raw**, **baboon.raw**, and **pepper.raw** file in the same folder as well. The file contains a function that takes in two parameters, i.e. the **filename** and the no of **coefficients**. You can enter the name of whichever file you want to encode in the filename portion and run it.

2.2. Main_part1.m

As explained above main_part1.m is associated with the energy_compaction.m, because that is the file that calls the function in energy_compaction() in energy_compaction.m file.The file passes two parameters to the energy_compaction.m, the parameters are the **filename** and the no of **coefficients**.

2.3. Zonal.m

Zonal Coding, i.e. part-2, has to be run using **main_part2.m** file, to run this file, you need to have the **zonal.m** file in the same folder and the three images file **lena.raw**, **baboon.raw**, and **pepper.raw** file in the same folder as well. The file contains a function that takes in three parameters, i.e. the **filename, the stepsize** and the no of **coefficients**. You can enter the name of whichever file you want to encode in the filename portion and run it.

2.4 main_part2.m

As explained above main_part2.m is associated with the zonal.m, because that is the file that calls the function in zonal() in zonal.m file. The file passes three parameters to the zonal.m, the parameters are the **filename, stepsize** and the no of **coefficients**.

2.5 Threshold.m

Threshold Coding, i.e. part-3, has to be run using **main_part2.m** file, to run this file, you need to have the **zonal.m** file in the same folder and the three images file **lena.raw**, **baboon.raw**, and **pepper.raw** file in the same folder as well. The file contains a function that takes in three parameters, i.e. the **filename, the stepsize** and the no of **coefficients**. You can enter the name of whichever file you want to encode in the filename portion and run it.

2.6 main_part3.m

As explained above main_part2.m is associated with the zonal.m, because that is the file that calls the function in zonal() in zonal.m file.The file passes three parameters to the threshold.m, the parameters are the **filename, stepsize** and the no of **coefficients**.

2.7 zigzag.m

This file contains the **function** for the **zigzag traversal of the array** as demonstrated in the question file. The file takes one parameter, i.e. the array to be traversed in zigzag order as the input, and **returns a single dimensional array** after traversing the array in zigzagged order.

2.8 invzigzag.m

This file contains the **function** for the **inverse zigzag traversal of the array** as demonstrated in the question file. The file takes three parameter, i.e. the zigzag traversed array as the input, the size of row, and the size of column and **returns a two dimensional array** after inverse zigzag traversal of the array.

II. Explanation of code logic step by step:

---

3. **ENERGY COMPACTION:**

As per the question, we need to compute the MSE after discrete cosine transform and selection of coefficients and for each value of coefficient i.e. 1, 3, 5 and 10.

4. There is no quantization, and we need to compute MSE for each picture.

5. So, we take in the image first, then we divide it into blocks of 8*8.

6. Then we perform discrete cosine transform on each block and we select the coefficients as given by the user. We also clip the values in the range of -2048 to 2048.

7. There is no quantization, so we traverse the array in the zigzag order as mentioned and store the non-zero values in a 1-D array. This is the final compressed vector.

8. Then on this vector we do, inverse cosine transform, and we traverse the image in inverse zigzag format.

Therefore, running this MATLAB code is pretty smooth, all the user needs to do is to enter the name of the file that it wants to use, and the coefficients asked for.

9. The final image has been stored in happy.bmp and the corresponding values from this happy.bmp and our original image has been used to calculate MSE.

ZONAL ENCODING:

As per the question, we need to compute the MSE and the total entropy after discrete cosine transform and selection of coefficients and quantization and for each value of step_sizest i.e. 8, 16 and 32.

4. Since, we need to compute MSE and the total entropy for each picture.

5. So, we take in the image first, then we divide it into blocks of 8*8.

6. Then we perform discrete cosine transform on each block and we select the coefficients as given by the user. We also clip the values in the range of -2048 to 2048.

7. We traverse the array in the zigzag order as mentioned and store the non-zero values in a 1-D array. We now do quantization on it using the given step_size by the user and we get our image in the form of compressed vector.

8. Then on this vector we do, inverse cosine transform, and we traverse the image in inverse zigzag format.We perform inverse quantization on it and compute the reconstructed image.

Therefore, running this MATLAB code is pretty smooth, all the user needs to do is to enter the name of the file that it wants to use, and the coefficients asked for.

9. The final image has been stored in happy.bmp and the corresponding values from this happy.bmp and our original image has been used to calculate MSE.

3.**THRESHOLD:**
As per the question, we need to compute the MSE after discrete cosine transform and selection of coefficients and for each value of coefficient, all coefficients and for step_sizes 8,16 and 32.

4. We need to compute MSE for each picture.

5. So, we take in the image first, then we divide it into blocks of 8*8.

6. Then we perform discrete cosine transform on each block and we select the coefficients as given by the user. We also clip the values in the range of -2048 to 2048.

7. We traverse the array in the zigzag order as mentioned and store the non-zero values in a 1-D array. We now do quantization on it using the given step_size by the user and we get our image in the form of compressed vector.

8. Then on this vector we do, inverse cosine transform, and we traverse the image in inverse zigzag format.We perform inverse quantization on it and compute the reconstructed image.
Therefore, running this MATLAB code is pretty smooth, all the user needs to do is to enter the name of the file that it wants to use, and the coefficients asked for.

9. The final image has been stored in happy.bmp and the corresponding values from this happy.bmp and our original image has been used to calculate MSE.

III. Inferences from result:

---

1. From the results we can see that as we increase the number of coefficients the MSE value decreases which makes sense because more is the information, better is the reconstructed image.
2. As we increase the step size the MSE increases, as the step size increases the MSE increases, as we quantize more, our image is more lossy.
3. Discrete cosine transform is an efficient algorithm to compress images for smaller step sizes and optimal number of coefficients.
4. final result for all three files and all three parts are in the excel file **HW02_SC.xlsx**. It significantly shows how the compression algorithm has reduced the way of representation for encoding and decoding.
The **Example from the sheet:**

Energy Compaction (MSE)

Coefficients	Lena	Baboon	Pepper
1	282.339062	696.453365	337.143478
3	122.246517	547.081654	135.370853
5	92.910618	461.613411	99.638725
10	40.980045	338.390316	46.455929
30	8.556271	119.717319	12.577877

IV. Conclusion:

- DCT is surprisingly good for compression when it comes to natural images and even very few coefficients, you could construct the image quite well.(thanks to its energy compaction properties)
- Since the basis functions are orthogonal(for common type, it is actually orthonormal as well), the coefficients are uncorrelated to each other and missing one coefficient does not affect the remaining ones. This provides a reliable reconstruction even if one or more coefficients are missing. In transformative coding or other coding schemas that coefficients are somehow related, the error in the communication channel may propagate to other pixels and reconstructed image and original image difference would result in quite big error even if only a few coefficients get corrupted.

- In the receiver, one needs to only take inverse dct transform as the basis functions are same in both transmitter and receiver, transmitter does not have to send the basis functions to the receiver. This provides a common, universal coding schema where if your computer has inverse dct transform capability, you just need to download few coefficients to reconstruct the image.
- DCT is actually applied block by block to the images and this may not be a best way to approach compression as blocks do not take the edges and texture into consideration when DCT is applied to the image.
- In general, DCT throws aways very high frequency components so it may be better to use some other compression schema if your image has high frequency components(rich texture, lots of transitions and sharp edges)
- DFT produces symetric signals in frequency domain for real signals. Therefore, part of the computation becomes a waste as you could reproduce the half of the coefficients from the rest of the coefficients. DCT not only produces more compact representation than DFT but also does not have this disadvantage that DFT has.