JPEG compression using discrete cosine transform

EE-305 Course Project
Muhammed Roshan (B17094)

***Abstract*—Compressing and storing images in more and more efficient ways is very important when we need to transfer images between various devices. There are different methods existing to convert an image to a format with a smaller size. Among these, one of the wildly used methods is JPEG compression of images using Discrete cosine transform. This project describes how to convert a BMP image into a compressed JPEG bitstream which can be later used to generate actual JPEG files.**

***Index Terms*—JPEG, Discrete cosine transform, DC and AC components, Quantization Matrix, Differential Pulse Code Modulation, Run-length encoding, Huffman coding.**

I. Introduction

The objective of JPEG compression is to reduce the storage required to store an image by removing the redundant data in the image. Compression methods can be classified into two on the basis of the quality of reconstructed images. These are,

- Lossy Compression:- In Lossy compression algorithms, the reconstructed image from the compressed file format will not contain as much information as it was in the original image.
- Lossless Compression :- In lossless compression algorithms, we can reconstruct the actual image weithout causing any loss ti image quality.

JPEG (Joint Photographic Experts Group) is a lossy compression algorithm for images which means the compressed image data can not be decompressed using 100% originality. The degree of compression can be adjusted allowing a trade-off between storage size and quality. A large majority of useful image data changes relatively slowly across images. It implies, in the frequency domain, low-frequency components contain much more useful data than the high-frequency counterpart. And our vision is not particularly sensitive towards these high-frequency components. Therefore, even if remove the data corresponding high-frequency region in the image, it will not suffer much loss in our perception. We utilize this fact to compress an image (in BMP format) into bitstream which can be used to generate the actual JPEG file. Bitstream generated will be decompressed to see the quality of the compressed image.

II. Compression steps

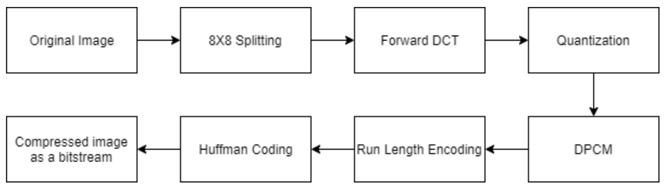


Fig. 1. Flow chart of steps

A. Splitting Image

Instead of doing the operations on the entire image, we are doing all of the operations of each 8X8 block of the image. This because performing DCT over the entire image which is basically a huge 2D array is computationally expensive and takes a lot of time. Splitting image into 8X8 block can give the optimal result without taking too much computation time but at the same time contains enough information to carry out further.

B. Forward DCT

Later we have to perform DCT to change the image from the pixel domain to the frequency domain. Here DCT is used because of its excellent energy compaction properties. Energy compaction means that a large portion of total energy is concentrated on a handful of DCT coefficients. And also unlike other transformations, DCT has only real terms. The forward DCT equation is as follows,

$$G_{u,v} = \frac{\alpha(u)\alpha(v)\Sigma\Sigma\ g_{x,y}cos(\frac{(2x+1)u\pi}{16})cos(\frac{(2y+1)v\pi}{16})}{4}$$

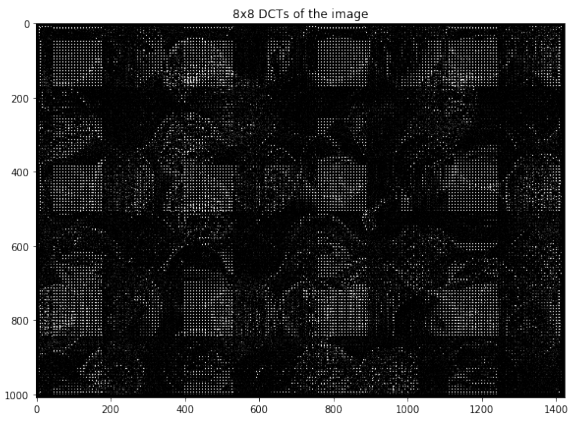


Fig. 2. Flow chart of steps

In the above figure, white parts represent frequency components with non-zero magnitude and black parts represent frequency components with zero magnitudes. The first component in each of the DCT blocks is called the DC component. For this component, the value of u and v are 0 therefor DCT of these is just an algebraic sum of all the elements in the input matrix.

All the other elements are called AC components since they all have an AC component associated with them.

C. Quantization

Next step is quantization. This is done to remove those high frequency components which we treat as redundant data. The resultant matrix will only have low frequency components up to a certain point (which is determined by the quality of output we want). Quantization step is the major step where we lose some image data. The result obtained after performing Block wise DCT over input will undergo element wise division with a matrix called Q matrix (which determines the degree of compression) and are rounded off to the nearest integer. Q matrix have larger values towards the lower right corner in order to introduce more loss on high frequency components. This results in matrix which is mostly sparse essentially decreasing the memory required to represent it. The values in Q matrix is obtained through psychological experiments in order to maximize compression and minimize loss.

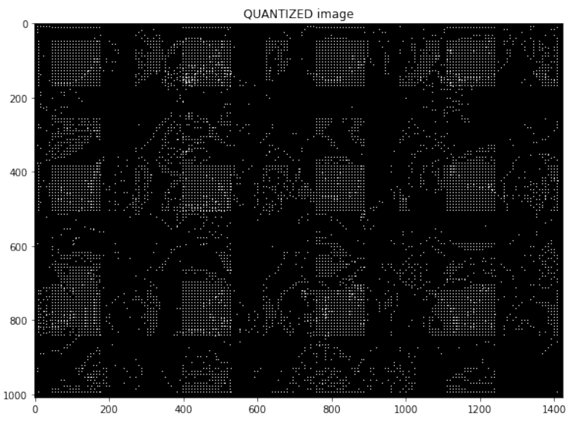


Fig. 3. DCT of image after quantization

D. DPCM and Run length encoding

Matrices obtained after quantization will be scanned in a zig-zag manner to convert them into 1D vectors with the DC component as the first element. DPCM (differential pulse code modulation) is done on DC components and Run-length encoding is for AC component.

In DPCM, the first element of the first vector is kept as it is, but the second element is stored as the difference between first and second. This is to utilize the fact that the DC components are integers closer to each other, and by taking differences, we get even smaller numbers which eventually results in lesser storage space.

Among AC components, many are zeros, therefore to decrease size, we can express each of these numbers as *(run length/size)value*, where run length represents the number of trailing zeros, value represents the non zero AC coefficient, and size is the number of bits required to store value. And if the end part vector is just zeros, we represent it as *(0,0)*. For example, if our array was like

[2, 4, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0],

we represent it as

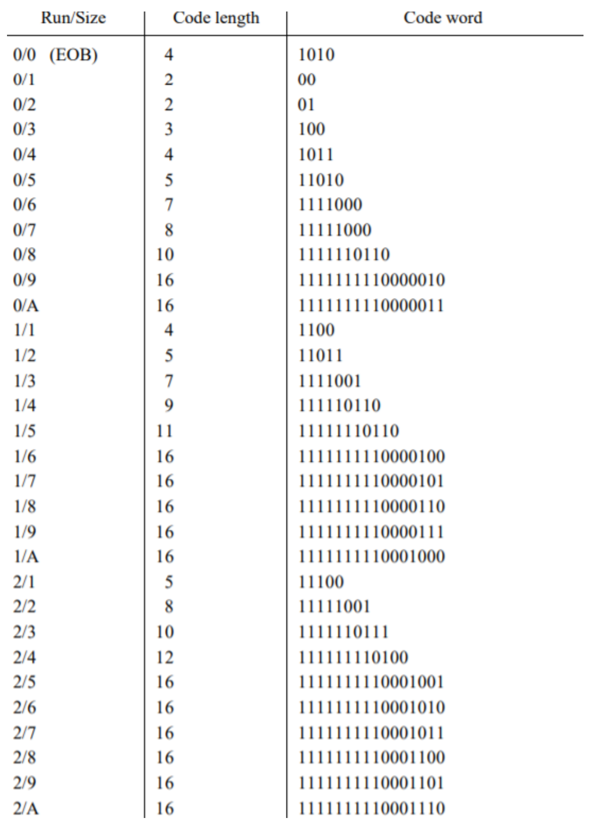
(0,2)/2, (0,4)/3, (3,1)/1, (0,0).

E. Huffman Encoding

DC and AC components finally need to be represented as a stream of bits. Though there are many coding algorithms in existence, here we use a particular method called Huffman coding.

In Huffman coding, the length of each codeword is a function of its relative frequency of occurring. Here, the code (bit sequences) are assigned in such a way that the code assigned to one character is not the prefix of code assigned to any other character. This is how Huffman Coding makes sure that there is no ambiguity when decoding the generated bitstream.

Determining Huffman tables optimized for a particular image requires going over the input data twice, first to collect the frequencies of input symbols to determine optimal Huffman codes and second to do the actual encoding using the codes found in the first step. But The JPEG standard contains Huffman tables which have been found to work well with input derived from regular continuous-tone images. These pre-computed tables can also be used for compression.



Run/Size	Code length	Code word
0/0 (EOB)	4	1010
0/1	2	00
0/2	2	01
0/3	3	100
0/4	4	1011
0/5	5	11010
0/6	7	1111000
0/7	8	11111000
0/8	10	1111110110
0/9	16	1111111110000010
0/A	16	1111111110000011
1/1	4	1100
1/2	5	11011
1/3	7	1111001
1/4	9	111110110
1/5	11	11111110110
1/6	16	1111111110000100
1/7	16	1111111110000101
1/8	16	1111111110000110
1/9	16	1111111110000111
1/A	16	1111111110001000
2/1	5	11100
2/2	8	11111001
2/3	10	1111110111
2/4	12	111111110100
2/5	16	1111111110001001
2/6	16	1111111110001010
2/7	16	1111111110001011
2/8	16	1111111110001100
2/9	16	1111111110001101
2/A	16	1111111110001110

Fig. 4. Sample Huffman Table

After Huffman coding, we get a stream of bits which represnts our input image. This bitstream can later be transformed into an actual JPEG file.

III. Result

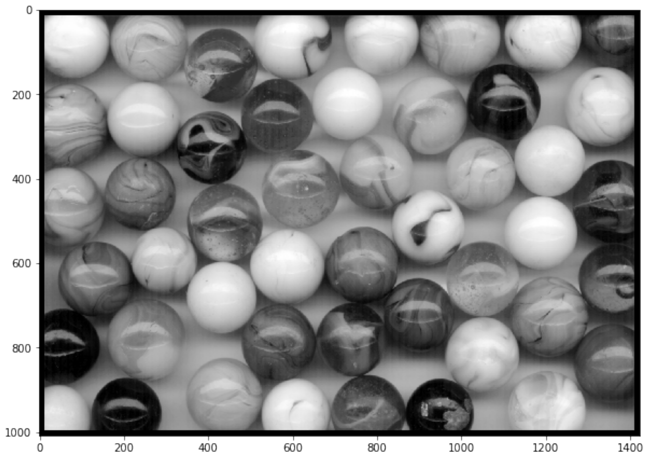


Fig. 5. Original Image

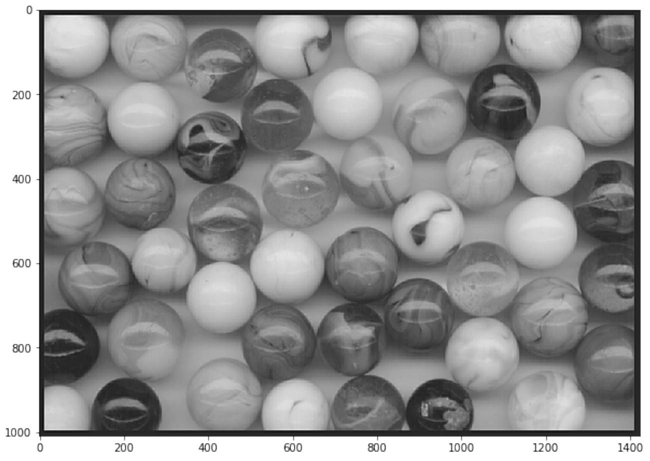


Fig. 6. Reconstructed image by retaining 90% of information

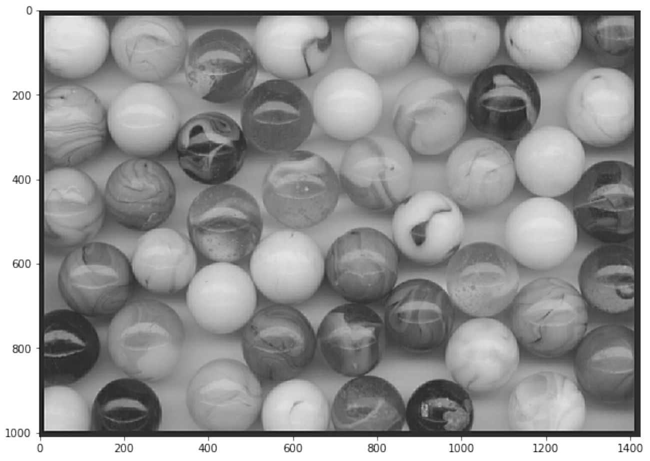


Fig. 7. Reconstructed image by retaining 50% of information

References

[1] https://www.circuitlab.com/
[2] https://www.ijg.org/files/Wallace.JPEG.pdf
[3] https://www.w3.org/Graphics/JPEG/itu-t81.pdf