Matthew Lonis

Adeel Bhutta

CSCI-B 456

27 April 2018

**JPEG Image Compression**

**Abstract**

Full description of the JPEG Compression Algorithm and how I implemented it. Comparisons to other projects and my results. A description of attempted improvements and a conclusion with what I wish I could have done with this algorithm.

**Introduction**

For my final project I chose to implement JPEG image compression. As a child, I was always interested in different file types for images and audio, often wondering why there were so many different kinds and what each of them meant. JPEG, often associated with .jpg or .jpeg, is one of the most common still-image file extensions. Not only is JPEG used for basic Internet images but also in applications such as satellites or the medical field where efficiency in transportation of the image in question is of high priority. To compress an image into a JPEG format, an introduction into image compression and JPEG is necessary.

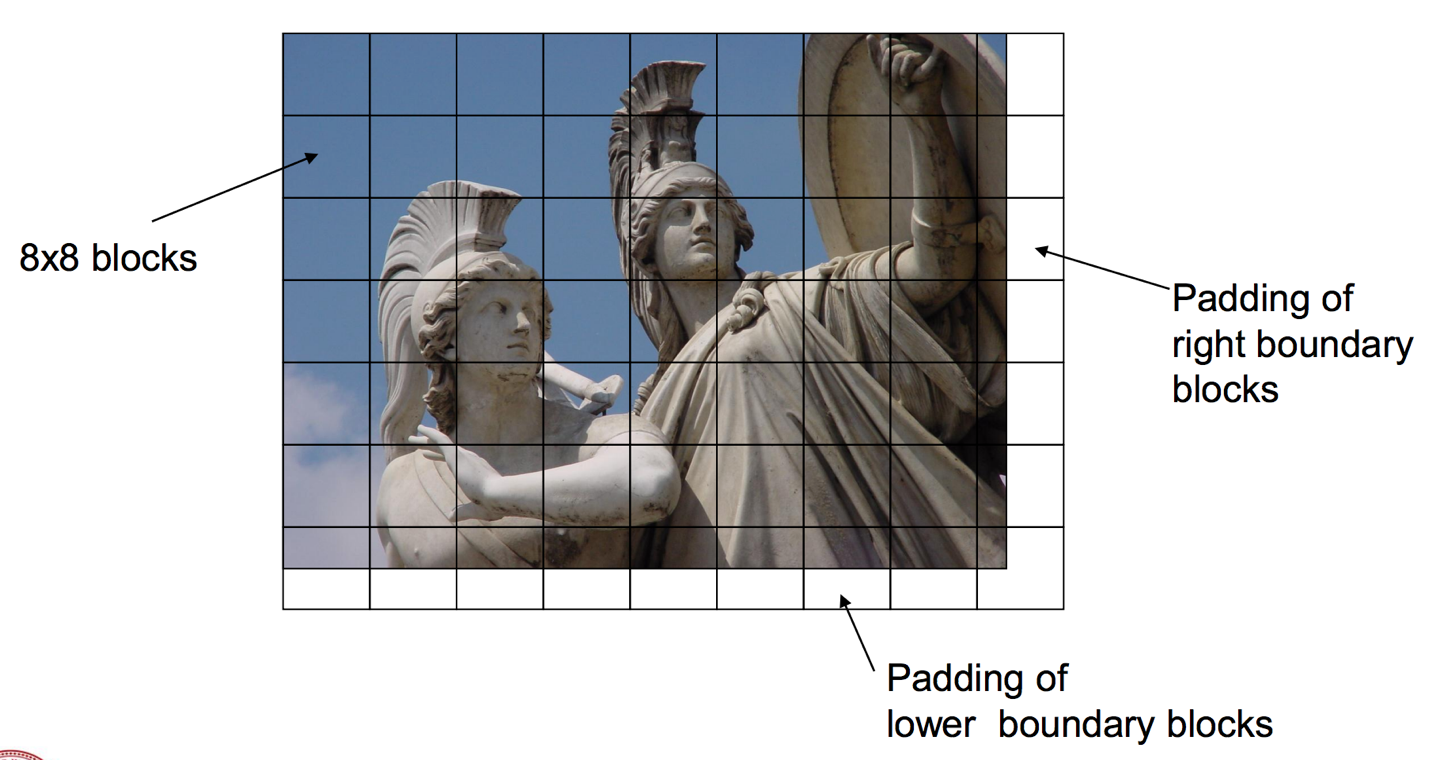
For starters, image compression, and compression in general, is a means of reducing the overall size of the file by implementing some sort of algorithm. The goal of compression is always to reduce the file size because compression cannot improve a file or filter an image. Compression can be either lossless or lossy each offering their own different perks. Lossless compression is a type of compression in which the original data is compressed in such a way that it can be decompressed back to the original data. Lossy compression on the other hand is a type of compression in which the original data is compressed down and a certain portion of the original data is lost. The decompressed lossy data cannot be reconstructed back to the original.

Image compression takes many forms, but the most common and widespread compression technique is JPEG. The JPEG standard was created by the Joint Photographic Experts Group committee; hence the name JPEG ("About JPEG"). The JPEG compression technique is a lossy image compression algorithm. This means that the image reconstructed from the compressed JPEG data is in no way the original image or data but is however extremely similar. The high-level overview of the JPEG compression algorithm is as follows:

1. **Gridding:** The image is divided into 8x8 blocks of pixels. If the image cannot be evenly divided into 8x8 blocks, the image is zero-padded to be evenly divided into 8x8 blocks.
2. **Color Space Transformation:** The gridded image is transformed from the RGB color space into the YCbCr color space. The YCbCr image is then down sampled on the Cb and Cr channels.
3. **Discrete Cosine Transformation:** The discrete cosine transformation (DCT) is applied to each 8x8 block.
4. **Quantization:** Each channel is multiplied by a specified quantization table according to the JPEG standard.
5. **Vectoring:** Each 8x8 block is zigzag scanned into a 1x64 vector.
6. **Differential Pulse Code Modulation (DPCM):** DPCM is applied to the DC components in each 1x64 vector.
7. **Rune Length Encoding (RLE):** RLE is applied to the AC components in each 1x64 vector
8. **Huffman Coding:** The vectors are combined into one giant vector and then compressed using Huffman Coding.

**Gridding**

The first step in the JPEG compression algorithm is to divide the image into 8x8 blocks of non-overlapping pixels. If the image cannot be evenly divided into non-overlapping blocks, the image is zero-padded to achieve an even division.



*Image Obtained from Slide #2 (Girod, Bernd)*

For the purposes of this project, because I wrote my algorithm in MATLAB, I did not truly divide the image into 8x8 blocks and store them in some data structure. Instead, I merely processed the image 1 block at a time.

**Color Space Transformation**

The next step is to convert the original image from the RGB color space into the YCbCr color space. The YCbCr color space used Y for luminance and Cb/Cr for chrominance blue and chrominance red. The reason for using the YCbCr color space is used because the human eye is more sensitive to Luminance (Y) as opposed to color (Cb/Cr). Therefore, by working on the image in the YCbCr color space, we are able to process color separately from luminance (Roman10, 1).

Once we have the image in the YCbCr color space, we can down sample the chrominance channels to reduce the number of bits representing color while having minimal impact to the overall visual quality of the image.

**Discrete Cosine Transformation**

The next step in the algorithm is to perform the discrete cosine transformation on each 8x8 block of pixels. The DCT converts the current spatial domain values in each block into frequency domain values. The human visual system is less sensitive to high frequency data so by converting the image to the frequency domain, we can find data that we can remove (Maan, Anmol Jyot, 2).

Although the DCT has a mathematical function, I found it quicker and easier to process the DCT using a DCT matrix. This is because MATLAB processes matrix operations faster than it processes summation (loops). For this to work we need an 8x8 DCT matrix shown in the table below.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0.3536 | 0.3536 | 0.3536 | 0.3536 | 0.3536 | 0.3536 | 0.3536 | 0.3536 |
| 0.4904 | 0.4157 | 0.2778 | 0.0975 | -0.0975 | -0.2778 | -0.4157 | -0.4904 |
| 0.4619 | 0.1913 | -0.1913 | -0.4619 | -0.4619 | -0.1913 | 0.1913 | 0.4619 |
| 0.4157 | -0.0975 | -0.4904 | -0.2778 | 0.2778 | 0.4904 | 0.0975 | -0.4157 |
| 0.3536 | -0.3536 | -0.3536 | 0.3536 | 0.3536 | -0.3536 | -0.3536 | 0.3536 |
| 0.2778 | -0.4904 | 0.0975 | 0.4157 | -0.4157 | -0.0975 | 0.4904 | -0.2778 |
| 0.1913 | -0.4619 | 0.4619 | -0.1913 | -0.1913 | 0.4619 | -0.4619 | 0.1913 |
| 0.0975 | -0.2778 | 0.4157 | -0.4904 | 0.4904 | -0.4157 | 0.2778 | -0.0975 |

To compute the DCT, we perform the following function on each 8x8 block:

where is the 8x8 DCT matrix shown above and is the transposed DCT matrix.

After performing the DCT on each block, some interesting features and terminology arise. The DCT formula places the mean of all 64 values in the upper leftmost pixel. The other 63 values in the block are pixels with a frequency value. Due to this, the top leftmost value is referred to as the DC component of the block. The rest of the values in the block are referred to as AC components.

**Quantization**

Since the human visual system struggles to see high frequencies, we can remove the high frequency data in each block. This is referred to as quantization and is the most significant reason why JPEG is a lossy compression technique. The quantization process defines an 8x8 quantization matrix that will be multiplied with each 8x8 block in the following way:

Where represents the final block after quantization, represents the 8x8 quantization matrix, and F represents the original block of pixels. For the Y channel, the quantization matrix is as follows:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **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** |

The Cb/Cr channels us the following matrix:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **17** | **18** | **24** | **47** | **99** | **99** | **99** | **99** |
| **18** | **21** | **26** | **66** | **99** | **99** | **99** | **99** |
| **24** | **26** | **56** | **99** | **99** | **99** | **99** | **99** |
| **47** | **66** | **99** | **99** | **99** | **99** | **99** | **99** |
| **99** | **99** | **99** | **99** | **99** | **99** | **99** | **99** |
| **99** | **99** | **99** | **99** | **99** | **99** | **99** | **99** |
| **99** | **99** | **99** | **99** | **99** | **99** | **99** | **99** |
| **99** | **99** | **99** | **99** | **99** | **99** | **99** | **99** |

These separate matrices allow for us to increase the amount of high frequency data removed from the Cb/Cr channels. Furthermore, the values tend to increase as you move to the bottom right of the matrix to introduce higher loss since each block has higher frequencies in the bottom right corners. We round to the nearest integer to speed up our Huffman coding while having a negligible impact to the overall quality of the picture.

**Vectoring**

After quantization, the next step is to zig-zag scan each block to reduce the 8x8 block into a 1x64 vector. The zig-zag scan scans through the blocks in a zig-zag path as shown in the picture below.



The purpose of using a zig-zag scan is to group the higher frequency data, zeroed out by quantization, together to aid in our run length encoding.

**Differential Pulse Code Modulation (DPCM)**

Once we have each block in vector form, the next step is to perform differential pulse code modulation on the DC components in each vector. To do this, we merely need to subtract the value in the first cell by the value in the previous vector’s first cell and store the result in the vector’s first index. This isn’t applied to the first vector since there would be no previous vector to subtract from.

The reason for performing this step is because, as stated earlier, the DC component is a mean of all the AC components in the blocks. This means that each DC component will be large and relatively similar to every other DC component. By performing the differential pulse code modulation, we are able to store the DC components with fewer number of bits.

**Run Length Encoding (RLE)**

From here, run length encoding needs to be performed. The remaining 63 AC components will contain a lot of zeros. Run length encoding will reduce the size of each vector by encoding the zeros as (value, skip) pairs. If there were 10 zeros in a row, then the coding would be (0, 10). An example of this is shown below:

Original Vector

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **7** | **0** | **0** | **0** | **0** | **9** |

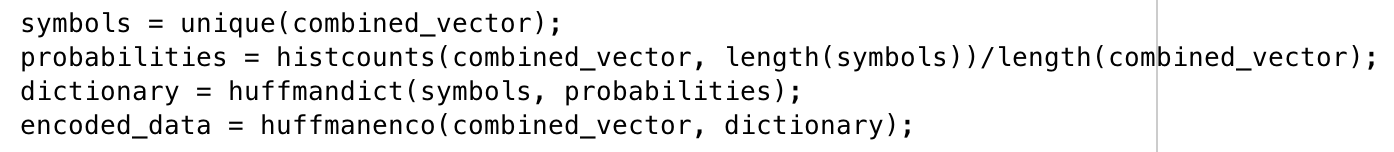
RLE Vector

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | **0** | **17** | **7** | **0** | **4** | **9** | **0** | **0** |

As shown in the example above, the length of the original vector is decreased from a size of 24 to a size of 9.

**Huffman Coding**

The final step is to combine all the vectors into one massive 1D vector. From here, Huffman coding needs to be performed on the combined vector to compress the number of bits. Huffman coding is used because it provides significant levels of compression by replacing the most commonly occurring values with smaller code words. The scope of how to implement Huffman coding was out of reach for this project. It was far easier to perform the Huffman coding using built in Huffman tools in MATLAB’s Communications System Toolbox.



In the code above, I obtain the unique values and use those for my symbols of the combined vector. Next, I obtain the probability of each “symbol” and finally I create my Huffman dictionary using a built-in MATLAB function and encode the vector. I found this to be the easiest and quickest way to get Huffman coding working for my project. In order to be as transparent as possible, I gained inspiration and technical know-how on how to use these built-in functions from <http://www.math.cornell.edu/~web6140/TopTenAlgorithms/JPEG.html>.

**Next Steps**

Once the image is compressed, it’s easy to decompress the data by using MATLAB’s built in Huffman tools. Then the RLE, DPCM, and Vectoring all need to be reversed. From here we need to perform the Inverse DCT function. This is easily replicated with the DCT matrix by following the formula below:

Next, we need to reverse our quantization. Since we can’t perfectly reverse quantization, due to the zeroing out of data, we can simply approximate by multiplying the quantization matrix to each block (Maan, Anmol Jyot, 3). After all these steps, the final image is an approximation of the original image.

**Attempted Improvements**

Since the JPEG compression algorithm is a defined standard, there isn’t much by way of improving this algorithm. The standard even has parameters for quality of compression by changing the quantization matrices and down-sampling amount so there are no improvements to be made there (Maan, Anmol Jyot, 1-2). So to try to improve my project, I choose to implement DCT using matrices since MATLAB performs faster on matrix operations than loop operations. This made my DCT step nearly instant, even for large images. Furthermore, by using MATLAB’s built-in Huffman tools, I was able to achieve compression extremely fast and even better than another project to which I compared to my project.

**My Results**

I found a project online at <http://www.math.cornell.edu/~web6140/TopTenAlgorithms/JPEG.html> which helped inspire my use of MATLAB’s built in Huffman tools. Ironically, my algorithm yields a higher level of compression. The project I found tests the JPEG compression algorithm on MATLAB’s built-in project images; saturn.png and peppers.png. I ran my algorithm on these two images and recorded the results.

*Online Project Results*

peppers.png

Final Compression = 0.147449705335829

saturn.png

Final Compression = 0.031958287037037

*My Project Results*

peppers.png

Final Compression = 0.136

saturn.png

Final Compression = 0.035

Unfortunately, my algorithm performed slightly less on saturn.png than I would have liked, however, it performed far better on peppers.png which is good news because the project on the website states that Saturn performs better for them (probably because it has so much black which are zeros). Based on this, my algorithm seemingly performs better on images with lots of color than the algorithm on the website. This is extremely useful because many images have a lot of color in them. I cannot determine why my algorithm performs less on Saturn than the one on the website, however, I think it is because the algorithm on the website removes all the zeros before compressing the data which would result in a much smaller encoded data stream.

**Conclusion**

JPEG is an extremely useful file format, allowing us to transfer images in a smaller size while still retaining their overall clarity. This is extremely useful for mobile phones, the Internet, and any application that does not need perfect quality images. I learned a lot by doing this project and feel as though I now intimately understand JPEG compression. If I had more time I would like to dive into the JPEG standard more and find out a way to implement a quality parameter to allow my algorithm to be dynamic and change the quality of the JPEG image by passing in a value to the function. Furthermore, it would be interesting to learn how to encode my encoded data into a data stream and write it to a file. I attempted to consider this for my project but found it would be too difficult in MATLAB given my current experience. Furthermore, the JPEG file format was complicated to me at first glance because there are many different versions of JPEG file formats and I felt it would impede my ability to successfully complete the project if I attempted to implement it.

**Works Cited**

"About JPEG." JPEG - About JPEG. N.p., n.d. Web. <<https://jpeg.org/about.html>>.

Girod, Bernd. "JPEG Standard." *Transform Coding - Overview*. N.p., n.d. Web. <<https://web.stanford.edu/class/ee398a/handouts/lectures/08-JPEG.pdf>>.

Maan, Anmol Jyot. "An Introduction to JPEG Image Compression Algorithm." Microsoft Word - EECEDEL-1711-229. University College of Engineering, Punjabi University, Patiala, Dec. 2013. Web. <<http://www.iraj.in/journal/journal_file/journal_pdf/1-10-139036859144-46.pdf>>.

Roman10. "YCbCr Color Space–An Intro and Its Applications." Roman10. N.p., n.d. Web. <<http://www.roman10.net/2011/08/18/ycbcr-color-spacean-intro-and-its-applications/>>.