Felipe Lamarca & Guilherme Melo

JPEG Compressor

Rio de Janeiro
2021

Felipe Lamarca & Guilherme Melo

JPEG Compressor

Project for Linear Algebra
Professor: Yuri Fahham Saporito

School of Applied Mathematics
FGV EMAp

Rio de Janeiro
2021

1 Introduction

This report is intended to analyze the JPEG compression process. This task brings Linear Algebra into practice as it deals with a series of steps related to applications with matrices, such as linear transformations, the use of orthogonal basis, and linear combinations, which are central in the compression process. It will be discussed how the theory behind JPEG compression takes not only Mathematics into account, but also some aspects related to human vision and its low capacity to perceive variations of color and brightness. This is fundamental when it comes to choosing which components must be stored and which can be discarded, and then making it possible to save images with equivalent quality — to some extent, of course — using less space.

The code used to develop the analysis was written in Python and all the information gathered to guide this report, which goes beyond the basics of Linear Algebra and tries to venture into a more hands-on scope of the discipline, will be duly referenced at the end of the text.

2 JPEG

Even though JPEG is not a file format, it is almost always referred as one. JPEG is rather an encoding algorithm to compress digital images effectively, which are then stored in JFIF files (the actual format). It is true that the algorithm is so effective that it is one of the most popular image extensions worldwide. It joins lossy and lossless compression methods to significantly reduce image size and facilitate its sharing and storing. It means that, in opposition to other methods like PNG, it does not guarantee that the output image is going to be just the same as the one input. However, the algorithm can make sure that both of them will be as alike as possible.

JPEG algorithm also takes advantage of human anatomy to drastically (according to one's interest) reduce file size without sacrificing a lot of image quality: as the human vision does not perceive colors or high-frequency changes very well, it is possible to get rid of some information of that nature and still maintain image quality to human eyes.

2.1 Color Space Conversion

The first step at image compression is Color Space Conversion. By the way, a *color space* stands for how the colors of an image are represented. For the most part input images are in the **RGB** color space and should be converted to a space-based on Luminance (brightness in greyscale) and Chrominance to attend the next steps of the algorithm. For this project, **YCbCr** was chosen as the color space. The conversion can be made by a simple linear combination stated as follows:

$$Y = 0,299R + 0,587G + 0,114B$$

$$Cb = 0,564B - 0,564Y$$

$$Cr = 0,713R - 0,713Y$$

The **Cb** component corresponds to the blueness of the image, while de **Cr** corresponds to its redness. And as humans do not see Chrominance as well as Luminance, the former can be down-sampled without generating much perceptible difference.

2.2 Chroma Subsampling

Now that we prepared our image in the last step, we will now do what is called Chroma Subsampling. As said before, since the human eye has a low acuity at perceiving color change, we can safely reduce the space taken by the color channels **(CbCr)**. And once we are keeping our luminosity channel in the same resolution, this will not cause a significant loss of image quality.

The subsampling is accomplished by deleting rows (and sometimes columns) to decrease image size. By deleting every other row, we can already get an image 33% smaller than the original, which is called a 4:2:2 Subsampling since we now have a luminosity matrix that is twice the size of both the chromatic matrices. That is the Subsampling scheme chosen for our compressor. However, it is not unusual to compress it further on this process by also deleting every other column, getting as result a 4:1:1 Subsampling and reducing the matrix to 50% of the original size. After the subsampling we apply the inverse transformation to get the image back to **RGB** Color Space.

2.3 Discrete Cosine Transform

As we know, the cosine function varies between 1 and -1 on the y-axis. The purpose of the Discrete Cosine Transform is to represent the image data in terms of cosine curves. More precisely: what happens when it comes to JPEG is that we divide each image into groups of 8 by 8 pixels, and each of these groups is encoded by its Discrete Cosine Transform. Furthermore, what the DCT application actually does is to change from the canonical basis to an orthonormal one, where each one of the 64 entries (8×8) is given by a different cosine curve for a certain frequency. Notably, it is a basis for any 8 by 8 pixels image space and, in fact, it is the linear combinations of these series of distinct cosine curves that allow the production of a variety of pictures.

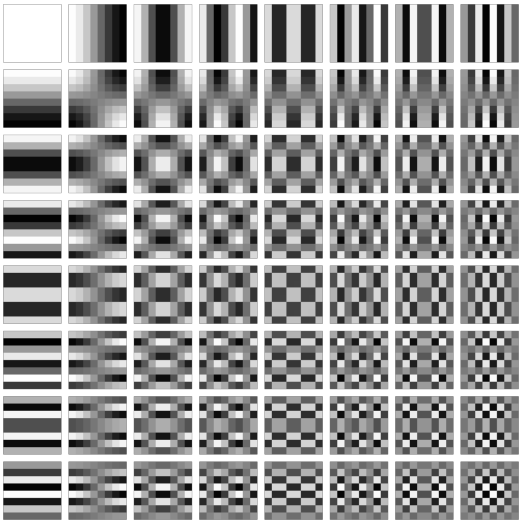

Figure 1: Basis for any 8×8 pixels image space

To put it another way, and that should be clear now, every image in 8 by 8 pixels can be reproduced as a combination of the "blocks" from Figure 1. In practice, it will be possible to assemble any image we would like by just weighing the contribution — or the *coefficient*, on the DCT language — of each one of those cosine curves to the whole. This calculation is exactly what Discrete Cosine Transform does throughout the formula:

$$\text{DCT(i, j)} = \frac{1}{\sqrt{2N}}\text{C(i) C(j)} \sum_{\text{x=0}}^{N-1} \sum_{\text{y=0}}^{N-1} \text{pixel(x, y)} \cos[\frac{(2\text{x}+1)\text{i}\pi}{2N}] \cos[\frac{(2\text{y}+1)\text{j}\pi}{2N}]$$

$$C(x) = \frac{1}{\sqrt{2}} \text{ if x is 0, else 1 if x} > 0$$

Note that the N in this particular case will always be 8 as we are dealing with 8 by 8 matrices. In short, the summation corresponds to the weighted sum of the basis images, while the multiplication of the cosines stands for the product of the vertical and horizontal cosine curves according to their frequencies.

An image — or a part of it — is given by a matrix in which the entries determine, in values between 0 and 255 (from the RGB system), how intense the pixel is. Suppose, for instance, the matrix below, which represents a slice of an image:

$$\begin{bmatrix} 89 & 97 & 101 & 101 & 113 & 206 & 231 & 235 \\ 85 & 85 & 89 & 89 & 109 & 231 & 235 & 235 \\ 85 & 81 & 85 & 89 & 105 & 215 & 239 & 239 \\ 105 & 101 & 101 & 101 & 113 & 239 & 243 & 243 \\ 178 & 186 & 190 & 186 & 194 & 231 & 239 & 243 \\ 235 & 239 & 239 & 243 & 243 & 239 & 243 & 243 \\ 235 & 239 & 235 & 235 & 231 & 239 & 231 & 239 \\ 231 & 231 & 231 & 231 & 227 & 227 & 227 & 227 \end{bmatrix}$$

The first step is to get this matrix ready for the Discrete Cosine Transform process by centering the data around 0 (after all, it deals with cosine curves) instead of 128, as it would be on RGB. To do so, we should subtract 128 units from each matrix entry. The result is the same image, but now the matrix that represents it is centered around zero.

Once that step is completed, applying the DCT II method (typically the one used for JPEG) will calculate the coefficients of each cosine curve for the "sum" that produces the exact image we desire. The result will be the one that follows:

$$\begin{bmatrix} 1500.2 & -263.7 & 91.0 & 26.8 & -61.0 & 16.1 & 28.3 & -32.3 \\ -311.9 & -211.7 & 71.8 & 25.3 & -47.2 & 15.4 & 20.8 & -20.3 \\ 16.9 & 38.3 & -13.0 & -8.6 & 9.5 & -2.8 & -5.7 & 4.4 \\ 103.8 & 62.4 & -27.1 & -17.6 & 15.7 & -8.7 & -11.4 & 10.2 \\ -25.7 & -8.3 & 5.9 & -3.2 & -5.0 & 1.2 & 0.7 & -0.2 \\ -24.8 & -16.4 & 13.4 & 1.6 & -4.0 & 2.4 & 2.6 & -6.3 \\ 10.8 & 11.6 & -4.5 & -8.5 & 9.3 & 3.7 & -10.2 & 8.0 \\ 2.5 & 10.6 & -2.7 & -11.0 & 6.0 & -2.4 & -5.2 & 5.9 \end{bmatrix}$$

An important fact to be noticed is that, in the JPEG case, the low-frequency blocks have much more effect on the final result than the high-frequency ones — note how the coefficients are much bigger on the top left-hand side when compared to the coefficients on the bottom right-hand side. This information is fundamental for the Quantization step, once it points out which components can be ignored — saving space on the process — without losing too much of the image quality.

2.4 Quantization

The Quantization process is responsible for removing the high-frequency data to save space and avoid too much of quality loss on the image. Note that there are many quantization tables and their usages depend on the needs and goals of each software. Moreover, they are directly related to the quality of the compressed image as it can be more or less strict in terms of which components should or should not be taken into account. The quantization table used on this project is the same as the JPEG standard and guarantees a compression of 33%:

$$\begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

The first step is to divide each one of the coefficients pointed out by the DCT by its correspondent on the quantization table, and then round the result to the nearest integer. It must be observed that, along this process, most of the lower coefficients get divided by huge numbers, which implies that they will not rarely be rounded to zero. What happens in practice is that most of the coefficients have their contributions on the final image set from some small value to zero:

$$\begin{bmatrix} 94 & -24 & 9 & 2 & -3 & 0 & 1 & -1 \\ -26 & -18 & 5 & 1 & -2 & 0 & 0 & 0 \\ 1 & 3 & -1 & 0 & 0 & 0 & 0 & 0 \\ 7 & 4 & -1 & -1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The result, as you can see, is that only a few non-zero numbers will be left on the table. They represent the values that are able to get back not exactly the original image, but one very alike — that is, it looks almost the same to human vision.

2.5 Run Length Encoding

With the quantized matrix we are now to cash in on our previous transformations and compress it with Run Length Encoding. This encoding mechanism will take advantage of the abundant amount of 0 entries quantization has given us, collapsing them together and referring to the amount of time it repeats. To get the most optimal result out of this method though, we shall first set a reading pattern that maximizes the number of repetitions; reading it either row by row, or column by column wouldn't be as effective as isolating the top left values with a Zig-Zag pattern. By reading the matrix this way we get our high-frequency values on the first reads, and then compress the most 0's with run length. By following this pattern we make sure we get the most of our lossless compression: once we serialized the data in a long line and compressed the zeros, this is the information that goes into the JPEG.

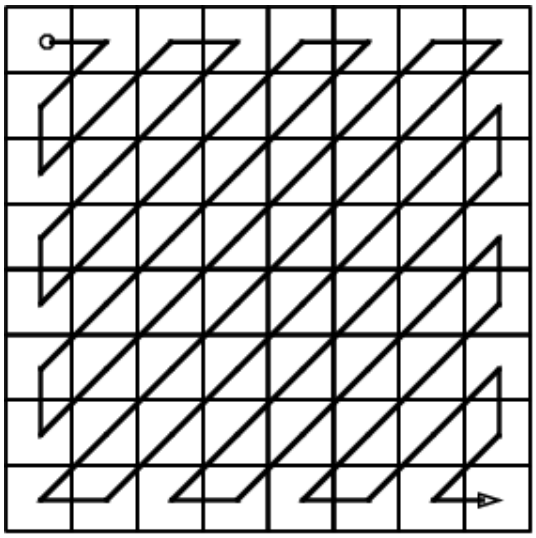

Figure 2: Zigzag Coding

2.6 The Decompression Process

As for the process of decompressing the image, the decoder just follows the inverse path of the compression steps. It takes the quantized table and multiplies each entry by its correspondent on the quantization table. Of course, many of the entries on the quantized table are zeros. The result of this multiplication is the coefficients, in such a way that many entries are also going to be zero, what means in DCT language that a lot of cosine curves will not contribute to the output image anymore. Applying the Inverse Discrete Cosine Transform (DCT III) gives the matrix centered around zero, which corresponds to a part of the image. Adding 128 units to each entry finally produces the output. Input and output shall be slightly different in terms of the intensity level of the pixels, but these changes are not perceptible to human eyes. On the other hand, the size of the image should be lower than when it was input.

3 Results

3.1 Images

Figure 3: Berries default

Figure 4: Berries compressed

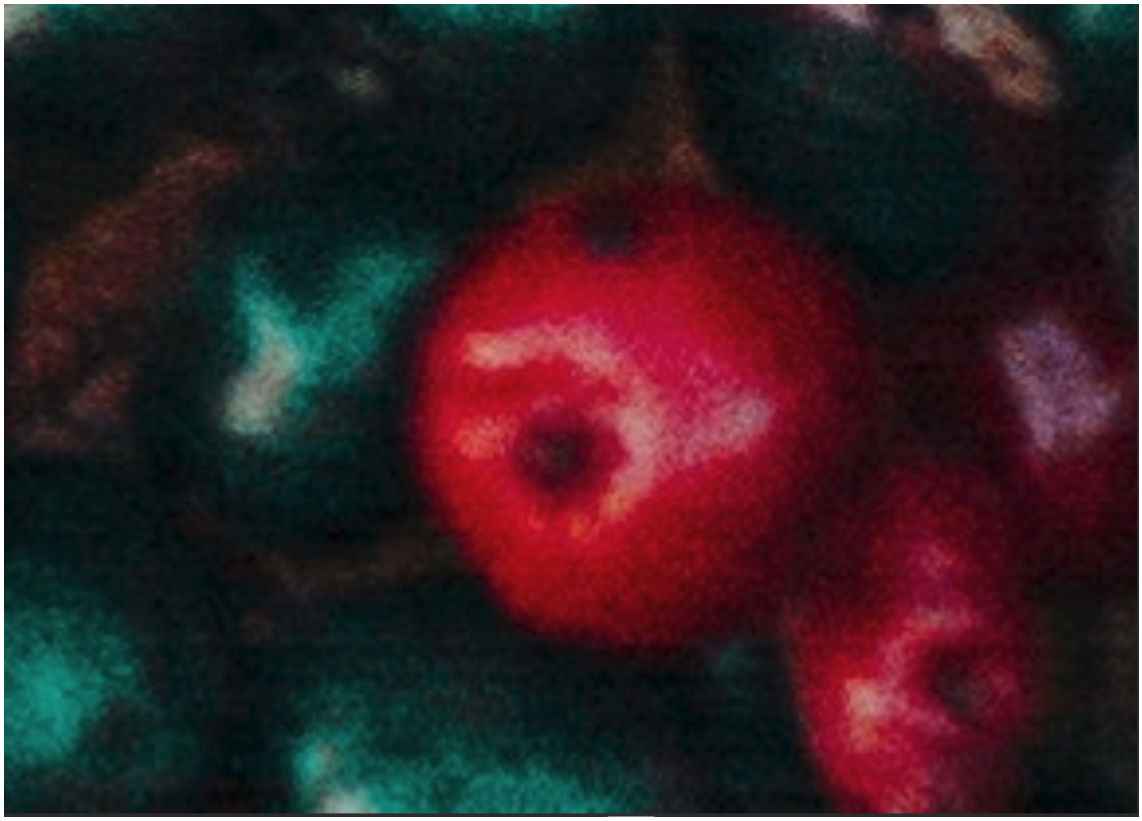

Figure 5: Close up on berries

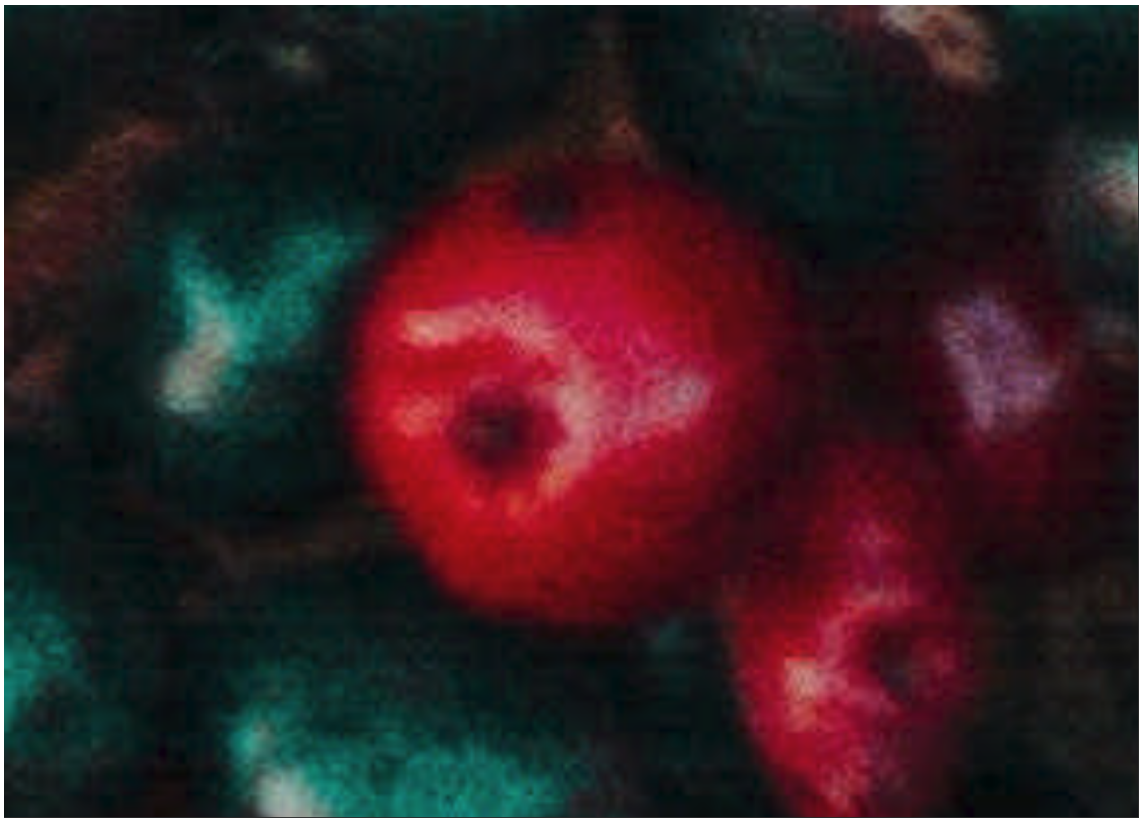

Figure 6: Close up on berries compressed

Figure 7: Cat default

Figure 8: Cat compressed

Figure 9: Cat's eye close up

Figure 10: Cat's eye close up compressed

3.2 Final Considerations

It is easy to observe in the results presented some aspects of jpeg compression. Its lossy nature becomes clear in the close-ups. Due to our quantization process, we get those squared artifacts that were not there in the initial image. In fact, in lower resolution images, we find it more easily: we are able to see it even with no zoom in. Besides that, we can verify that the compression does a good job in preserving the image integrity. In zoom out the higher resolution images look the same when comparing the pre and post-compression states. In summary, the result analysis confirm our initial assumptions.

It is also clear that the comprehension of Linear Algebra and its methods is an essential part of the process. As it was widely reinforced during this analysis as a whole, the concept of linear combination and basis is fundamental to most steps. Moreover, in order to define which components can or cannot be discarded, the facts that human beings do not perceive neither color nor high-frequency changes very well were also taken into account, what is an elegant way of saving storage space while retaining image quality.

During this report, our primary goal was to deeply, but also in an accessible way, explain how the process of JPEG compression works in practice. The Discrete Cosine Transform is certainly the most important step of the whole process. Even though it is not a trivial method, the idea behind it becomes pretty much intuitive once the reader has some knowledge of the basic concepts of Linear Algebra.

References

- Github project
- IsChen916's Github
- Professor's Miguel Frasson (USP) lesson on Discrete Cosine Transform and JPEG compression
- Colourspaces (JPEG Pt0) - Computerphile
- JPEG 'files' Colour (JPEG Pt1) - Computerphile
- JPEG DCT, Discrete Cosine Transform (JPEG Pt2) - Computerphile
- The Two-Dimensional Discrete Cosine Transform
- Discrete Cosine Transform — Stanford
- Stackoverflow on: RGB to YUV conversion