JPEG vs JPEG 2000
CS 166 Final Project

Kevin Do

June 5, 2024

1. Introduction

In my final project, I was able to implement a basic form of JPEG 2000 and JPEG. From it, I gained a deeper understanding of the differences between the image techniques, and the pros and cons of the two compared to various other forms of image formats. Through the implementations, I was surprised to see how well images could be compressed despite their initial huge size.

The algorithm for JPEG and JPEG 2000 seem quite simple, however verifying the correctness of the implementation was difficult.

JPEG Compression

The main steps for JPEG compression are as follows:

1. Convert the image into the YCbCr channels.
2. Partition the image into 8x8 chunks. When the image size is not divisible by 8x8, pad the image so that it can be split into the chunks.
3. Transform the image using the discrete cosine transform (DCT).
4. Discard less relevant features (high frequency components) (Quantization step)
5. Implement entropy coding, such as Dynamic Huffman Coding, and run-length encoding to better compress the data.

JPEG 2000 Compression

The steps for JPEG 2000 compression are:

1. Compute the Wavelet transform to get the wavelet coefficients.
2. Quantize the wavelet coefficients.
3. Repeat this process for a variable amount of times.
4. Utilize entropy coding, just like JPEG. Instead of run-length encoding, we can use more sophisticated algorithms as instead of the numbers going to zero, a lot of the numbers are repeated.

What about RAW and PNG files?

The primary advantage of raw files is that they contain the full dynamic range (typically 12- or 14-bit) data as read out from each of the camera's image sensor pixels. As we know from our assignments, PNG and JPEG images are typically only 8 bit. Therefore, we cannot encode the whole dynamic range as effectively as RAW files, losing quality.
PNG vs JPEG vs JPEG 2000 vs RAW So what are the main differences between the techniques?

1. RAW uses minimal image compression techniques, and takes up a lot of storage.

2. PNG implements the deflate format, the same way that zip files are created. This is done with huffman codes as well as LZ77, a variant of the run length encoding algorithm. This
3. JPEG 2000 implements a discrete wavelet transform, while JPEG implements the discrete cosine transform
4. JPEG is faster at rendering compared to JPEG 2000
5. In terms of image quality RAW > PNG > JPEG 2000 > JPEG
6. JPEG files typically take up 2 to 6 times less space compared to RAW files.

2. Methods

Quantization

JPEG quantization works by simply dividing the 8x8 block by a quantization matrix. The Quantization matrix for a q factor of 50 I found comes as follows:

$$Q_{50} = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

Figure 1: Standard quantization matrix

However, comparing my implementation of JPEG and a standard library won't generate the same results, since the quantization steps are completely dependent on the quantization matrix. There is a huge variety of quantization matrices, and is an active field of research, which many companies like Adobe keep secret. Even for JPEG 2000, there's a lot of variation, because the step sizes may differ. Another difference between my implementation and other JPEG implementation is that other implementations may use subsampling of the YCbCr channels. This is because when we convert to YCbCr, the Y channel contains more of the information to reconstruct the image compared to Cb and Cr.
JPEG 2000 uses a step quantization, essentially groups values within a range. The formula therefore looks like

$$q = sign(x) \left\lfloor \frac{|x|}{\Delta} \right\rfloor$$

where Delta is the step size. Note that smaller deltas result in less groupings of values. After the quantization, we can use standard entropy encoding to reduce the size of the files, given that we know there are only so many possible values of q. Check out the histogram on the next page.

Transformations and Quantization

We know that python is an interpreted language, and therefore takes a lot of time running all the code. I found a few optimizations that sped up and made the compression and decompression of JPEG a lot easier.

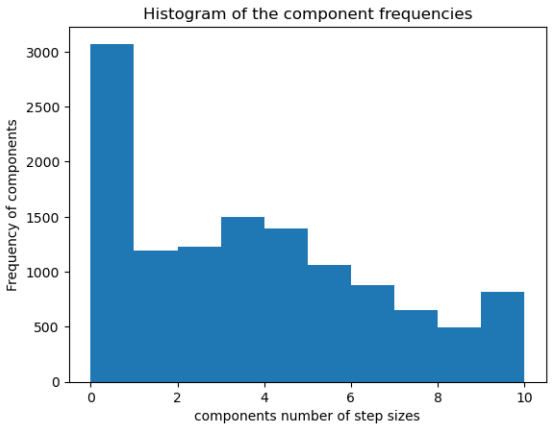

Figure 2: Histogram of quantized coefficients of the blur block of the "Dog.png" image. Note that we don't have any negative values here because the approximation coefficients are always averaged (like the bright cat in Fig.5)

Transformation Matrices

For both implementations, we have to implement the transformation matrices. The discrete Cosine transformation matrix can be defined as

$$T_{pq} = \begin{cases} \frac{1}{\sqrt{M}} & \text{if } p = 0 \text{ and } 0 \leq q \leq M-1 \\ \sqrt{\frac{2}{M}} \cos \frac{\pi(2q+1)p}{2M} & \text{if } 1 \leq p \leq M-1 \text{ and } 0 \leq q \leq M-1 \end{cases}$$

For extra optimization with JPEG, since we know JPEG computes the DCT in 8x8 blocks, I hardcoded the 8x8 JPEG's DCT matrix with this formula. However, this loses a fair bit of image quality due to roundings.
For JPEG 2000, we know that the approximation coefficients are given by the output of the lowpass, and the detail coefficients are given by the output of the high pass stuff. Therefore, we can say that the wavelet transform for the Haar wavelet is

$$H = \begin{bmatrix} 1 & 1 & 0 & \dots & 0 & 0 \\ 1 & -1 & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & 1 & \dots & 0 \\ 0 & 0 & 1 & -1 & \dots & 0 \\ \vdots & \vdots & & & & \\ 0 & 0 & \dots & 0 & 1 & 1 \\ 0 & 0 & \dots & 0 & 1 & -1 \end{bmatrix}$$

To get the respective detail and approximation coefficients from the Wavelet transform, we can just permute the output so that the approximation coefficients correspond to the numbers getting summed together, and the detail coefficients are the numbers getting subtracted.
Note that the 2d wavelet transformation is different, as we need to compute the high frequency components and the low frequency components of each the detail and approximation coefficients

JPEG Dimples

Here, since my quantization implementation utilizes the floor operator, we can see some artifacts from the JPEG compression. We therefore should see some JPEG dimples, or a darker spot every 8x8 blocks.

After tinkering with it a while, I realized that if I were to create a prominence map, it would take a really long time. This is because for every pixel, we need to do a convolution operation over the whole 32x32 template, over the whole picture.

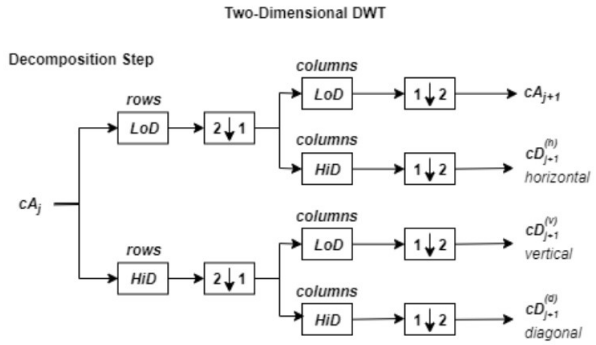

Figure 3: Matlab's implementation of a single level 2-D Discrete Wavelet Transform

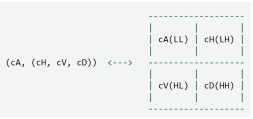

Figure 4: Composition of image

However, by digging through the documentation, I realized that I can use scipy's FFTconvolve to do the convolution much faster. This works because convolution in the time domain is multiplication in the frequency domain. In the end, I was able to see my JPEG dimples, even with minimal post processing. In the results section, you can see the JPEG dimples and I included the prominence map for the given image of the cat ear.

3. Results

I decided to steal the cat image from PS4 to run my tests. I also tested the algorithm with a few other images, but they're not included here as my report is running long.

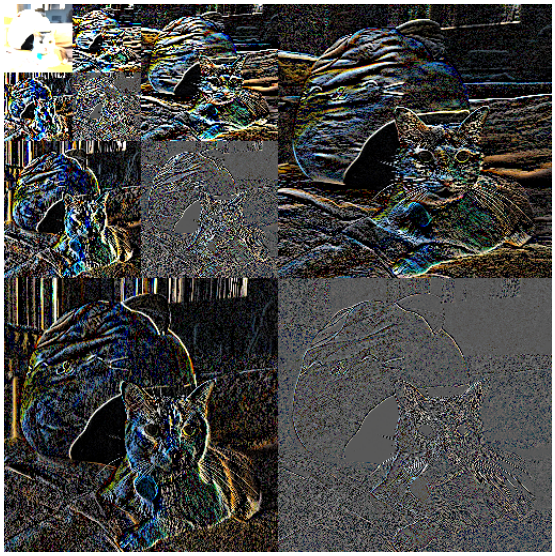

Figure 5: Visualization of wavelet domain from the composition. after 3 iterations of the wavelet transform. Image is slightly tweaked to see details more clearly

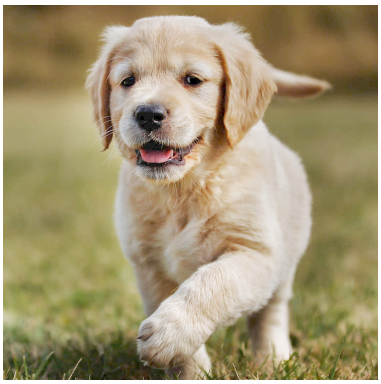

Figure 6: Dog compressed with OpenJPEG 2000

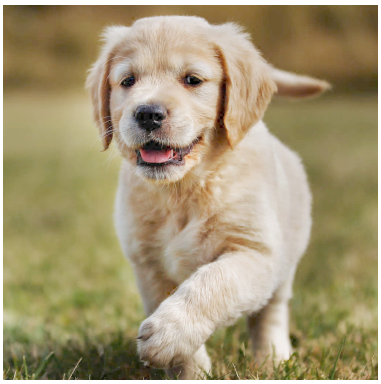

Figure 7: Dog compressed with my implementation of JPEG 2000

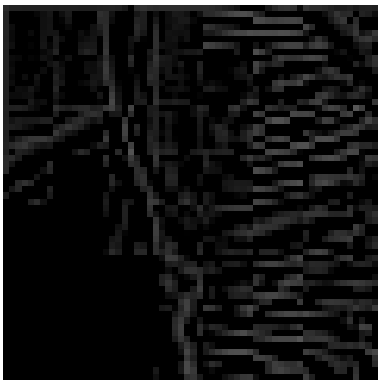

Figure 8: Wiener filtered Cat Ear. Notice the blocky shapes and the typical lighter pixel in the top left corner

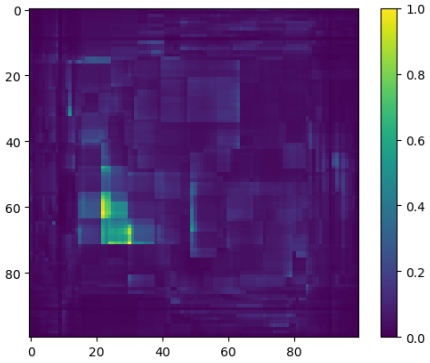

Figure 9: Prominence map of the cat ear. Higher levels signify more correlation

Figure 10: My JPEG 2000 implementation

Figure 11: OpenJPEG2000 implementation

Figure 12: PIL's implementation of JPEG, with Q factor=50

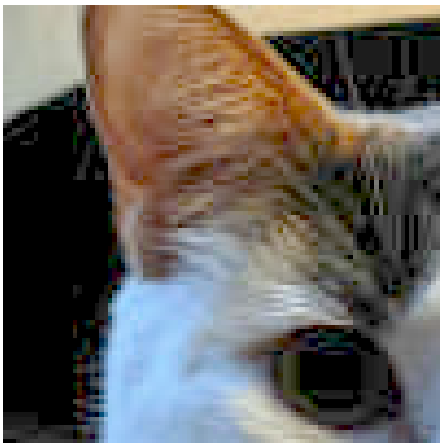

Figure 13: My JPEG implementation artifacts

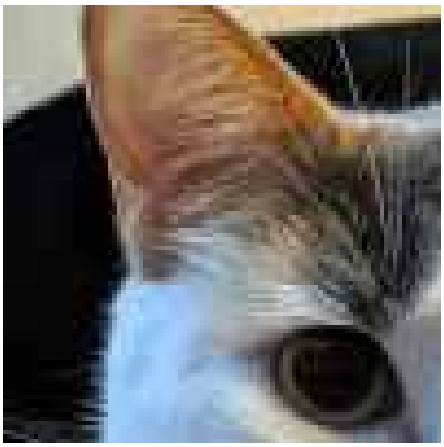

Figure 14: PIL's implementation of JPEG artifacts

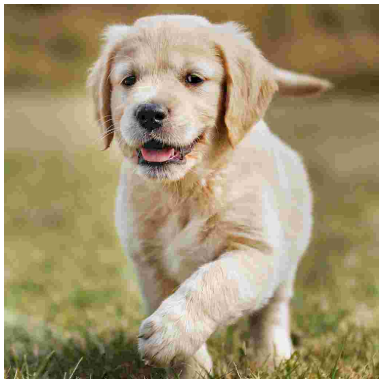

Figure 15: Dog Compressed with Q-factor of 10

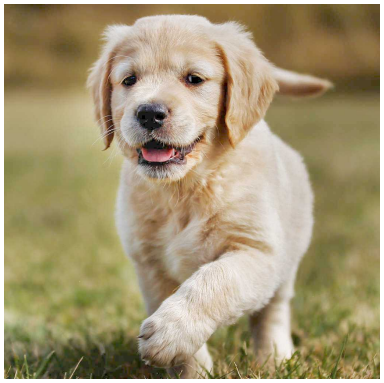

Figure 16: Dog Compressed with Q-factor of 50

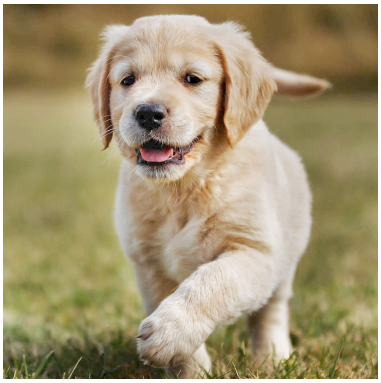

Figure 17: Dog Compressed with Q-factor of 90

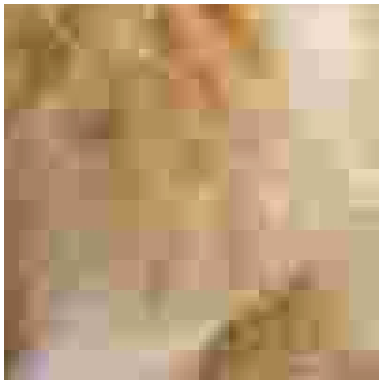

Figure 18: Dog Compressed with Q-factor of 10

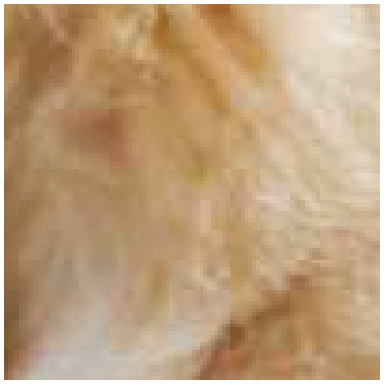

Figure 19: Dog Compressed with Q-factor of 50

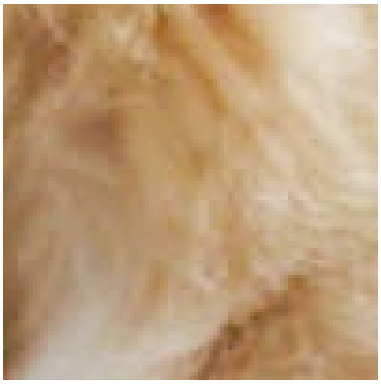

Figure 20: Dog Compressed with Q-factor of 90