Comparison of Methods Used in Lossy Digital Image Compression

by

Joseph Henry

Submitted to the Department of Mathematics/Computer Science
School of Natural Sciences
in partial fulfillment of the requirements
for the degree of Bachelor of Arts

Purchase College
State University of New York

May 2020

Sponsor: Dr.Knarik Tunyan

Second Reader:Dr.Irina Shablinsky

Abstract

This project explores the need for lossy image compression methods to analyze what information is lost when compressing images. For digital images, compression is particularly important because depending on how you change or alter an image can impact its quality. Digital images in their raw form require an enormous amount of storage capacity. Considering the important role played by digital imaging, it is necessary to develop a system that produces a high degree of compression while preserving critical image information. There are various lossy transformation techniques used for data compression. The Discrete Cosine Transform (DCT) and Discrete Wavelet Transform (DWT) are the two most commonly used transformations. In this research project, Google Colab and MATLAB were used to examine and compare the differences between the JPEG and JPEG 2000 compression algorithm. This project also examines which transforms compact the most energy(or information) in an image.

Table of contents

Chapter 1: Introduction

The rapid rise of smartphones has also come with the rapid rise of social media platforms like Instagram and Snapchat. These platforms have made it possible for individuals to share photos from their everyday lives with people from across the world. In an uncompressed form, these photos would take up an excessive amount of disk space and overburden networks with the vast amount of data being transferred. While numerous compression methods and standards have emerged over the years, the most widely used standard by far has been JPEG. The JPEG standard was established in 1992 and stands for “Joint Photographic Experts Group”. It became one of the most widely used standard because JPEG is a lossy digital image format which means that some information is lost during the compression process. This lost visual information tends to be imperceptible to the human eye for natural images like photos of people or landscapes but can be noticeable in images of documents and medical images because the information in the image is dependent on the consistency between pixels. This allowed pixel consistency loss makes lossy compression methods particularly useful for social media platforms because the photos people upload tend to be natural images that are still acceptable with the higher compression ratios with lossy compression methods.

Chapter 2: Literature review

2.1 What is an image

What are the different components required to represent digital images? A digital image is represented as a matrix of numerical values called pixels. Each pixel represents an 8-bit range of values from the interval [0,255] that corresponds to different values such as brightness or color. This range is called a color channel and defines the color information for one of the primary color components of a color model. The number of pixel values being used to display an image determines the image's spatial resolution. This number is independent of the image's width and height.

A color model is an abstract mathematical model that describes the range of colors as tuples of numbers. This color model is different from a color space which is a specific organization of colors that a device can represent. RGB is an additive color model that uses red, green, and blue to represent an image. There are several different RGB models, but the most commonly used ones are Adobe RGB and sRGB. The YCbCr is also an additive color model but instead of red, green, and blue it uses the luminance y, and chrominances cb and cr. Grayscale images use a single-color channel and 8-bits per pixel value to represent 256 different shades of gray in an image. In a similar way to grayscale images, 24-bit true-color images use the combined values of three different 8-bit color channels where each channel represents 256

shades of each color value. It follows, then, that one pixel can represent 256^3 or 16,777,216 different colors. This range is typical of the many different color spaces used in image processing.

2.2 Compression methods

Image compression is the process of reducing digital images without degrading the quality image to an acceptable level. The reduction in file size allows for more images to be stored in a limited amount of disk space. It also reduces the time required to send or receive an image. There are three types of data redundancies that all compression methods try to address: psycho visual, inter-pixel, and coding.

A psycho visual redundancy is information that can be reduced because it doesn't have a noticeable impact on the visual information perceived by humans. This reduction is possible because the human eye doesn't respond with equal sensitivity to all visual information. This means that when compressing an image certain information can be quantized or transformed is relatively less important than the rest of it.

An Inter-pixel redundancy refers to the correlation of neighboring pixels values in an image. This means that the value of any given pixel can be predicted from the value of its neighbors. Inter-pixel redundancies are reduced by narrowing the difference between related pixel values.

A coding redundancy is the reduction of the number of bits needed to represent each symbol in a set of data. This is usually done by analyzing the frequency of each symbol and creating a shorter representation of the more frequent symbols.

Digital image compression is classified into two different types of compression, lossless and lossy. Lossless data compression refers to compression algorithms that can compress and decompress an image without losing any information. These methods fall into two categories: statistical methods and dictionary-based methods. For lossy image compression, these methods are usually applied at the end to further compress the data.

2.3 Entropy encoding

Run-length encoding (RLE) is a simple statistical lossless data compression technique that by itself is used for faxes and black and white scanned images. Run-length encoding works by counting the number of consecutive pixels in an image and encodes it by the color of the pixel and number of times it appears consecutively. Here is an example:

A	A	A	B	B	C	C

Using run-length encoding on this text gives you :

3	A	2	B	2	C

Huffman coding is also a statistical lossless compression algorithm used for compressing several different types of data by reducing 8 bits typically used to represent a number and

creating a tree that is used to assign codewords of a lower bit value. The first step in Huffman coding is analyzing the frequency that a character or number appears in a set of data and each character then becomes a one node binary tree where the parent node is the frequency the character appeared and where the leaf node is the character and the frequency it appeared. Then the two trees with the lowest frequencies are then merged into one tree with their parent node now equaling the sum of their frequency. This is repeated until all the trees form a single tree where each edge represents a single bit where the left bit is zero and the right bit is one.

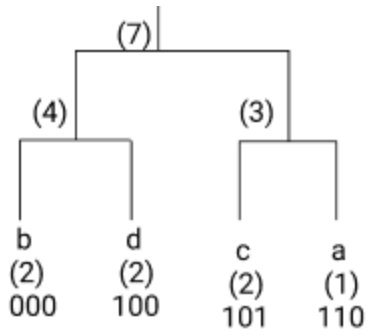

Figure 2.1: Huffman Tree example

The Huffman code is a prefix code which means that the binary code of any symbol is not the start of a codeword of any other symbol(Pujar, Jagadish 2010).

2.4 Lossy compression.

Lossy image compression is only an approximation of what the original image looked like. Lossy compression methods are designed to reduce inter-pixel and psycho-visual redundancies. The lossy compressed image can look similar to the original uncompressed image but there is some information lost which humans can't see. Pixels in an image tend to be highly

correlated and it can make compressing an image difficult. Transform coding methods are aimed to simplify an image's data by decorrelating inter-pixel information and concentrating its information into a small number of coefficients. Common lossy compression schemes such as jpeg and jpeg2000 use both data transforms and quantization methods to achieve a higher compression than lossless techniques.

2.4.1 Discrete Cosine Transform

The main function behind the JPEG compression scheme is the Discrete Cosine Transform (DCT), an orthogonal transform that transforms an image from the spatial domain to the frequency domain. The DCT represents an image as the sum of sinusoidal forms of varying frequencies and magnitudes and attempts to decorrelate the data in an image and focus most of the information in as few coefficients as possible. This is referred to as "energy compaction" and the DCT has this property for most images where the data in an image is correlated from one pixel to the next. Before applying the DCT to an image block, its value range is shifted from its initial range of 0 to 255 to a range of -128 to 127. This is to center the range of values around zero because Cosine returns values between 1 and -1 which is also centered around zero. In the JPEG compression scheme the DCT is typically performed on 8 x 8 blocks but can be performed on matrices of varying sizes. The 2-D DCT is given by (Anamitra Bardhan 2012):

$$D(i,j) = \frac{2}{\sqrt{2N}} c(i)\, c(j) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} p(x,y) \cos\left(\frac{(2x+1)\, i\pi}{2}\right) \cos\left(\frac{(2y+1)\, j\pi}{2N}\right)$$

Where p(x,y) is an input matrix NxN, x and y are the coordinate of the input matrix and i and j are the coordinates of DCT coefficients, and the value of c(u) is given by:

$$c(u) = \begin{cases} \frac{1}{\sqrt{2}} & if\ u = 0 \\ 1 & if\ u > 0 \end{cases}$$

After performing the DCT on an 8x8 image block a majority of its low-frequency visual information is concentrated in the top-left matrix entry known as the DC coefficient. The rest of the coefficients in a matrix are the AC coefficients and they contain more high-frequency information. The next step in compressing the data is to quantize the data by dividing each entry in the image block by its corresponding entry in the quantization matrix Figure 2.2.

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

Figure 2.2: JPEG quantization table Data Compression: The Complete Reference. p. 345.

The quantization matrix's values can be increased to have greater compression or decreased to have lesser compression depending on the quality level. To reconstruct the original matrix, you

multiply each entry by its corresponding matrix entry quantization matrix. You then compute the Inverse Discrete Cosine Transform or IDCT which is given by:

$$P(x,y) = \frac{2}{\sqrt{2N}} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} c(i)\, c(j)\, D(i,j)\ \cos\left(\frac{(2x+1)\, i\pi}{2}\right) \cos\left(\frac{(2y+1)\, j\pi}{2N}\right)$$

After the IDCT 128 is added to every matrix entry to bring its range back to 0 to 255. Let's test this process by using a random input 8 × 8 data matrix representing a portion of an image as shown in Figure 2.3 :

Input matrix

$$\begin{bmatrix} 55 & 51 & 60 & 70 & 70 & 71 & 78 & 86 \\ 58 & 61 & 70 & 77 & 82 & 82 & 91 & 94 \\ 57 & 65 & 75 & 78 & 97 & 90 & 88 & 95 \\ 67 & 78 & 83 & 88 & 90 & 93 & 94 & 90 \\ 75 & 80 & 89 & 93 & 100 & 102 & 98 & 96 \\ 79 & 80 & 92 & 96 & 100 & 104 & 100 & 98 \\ 94 & 90 & 97 & 104 & 100 & 104 & 105 & 108 \\ 94 & 90 & 97 & 104 & 100 & 104 & 105 & 108 \end{bmatrix}$$

Matrix after applying the DCT

$$\begin{bmatrix} -327 & -67 & -19 & -5 & 4 & 1 & 3 & 1 \\ -84 & -18 & 0 & -3 & 2 & -4 & -3 & 4 \\ -10 & 0 & 13 & -3 & 8 & 6 & 1 & -2 \\ -8 & 1 & 0 & 1 & 0 & 4 & 2 & -1 \\ -6 & 6 & 1 & 0 & -1 & 2 & 0 & -3 \\ -4 & 4 & 1 & -1 & 0 & 0 & 0 & -1 \\ -8 & -3 & -4 & 2 & 1 & -4 & 2 & 2 \\ 2 & -2 & 0 & 1 & 3 & -3 & 1 & 2 \end{bmatrix}$$

Matrix after quantization:

$$\begin{bmatrix} -84 & -6 & -1 & 0 & 0 & 0 & 0 & 0 \\ -7 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Matrix dequantization and IDCT:

$$\begin{bmatrix} 56 & 60 & 67 & 74 & 80 & 83 & 84 & 84 \\ 58 & 63 & 69 & 76 & 82 & 85 & 86 & 86 \\ 63 & 67 & 74 & 80 & 86 & 89 & 89 & 90 \\ 70 & 74 & 80 & 86 & 91 & 93 & 94 & 94 \\ 77 & 80 & 86 & 92 & 96 & 98 & 99 & 98 \\ 83 & 87 & 92 & 98 & 101 & 103 & 103 & 103 \\ 88 & 91 & 97 & 102 & 105 & 107 & 107 & 106 \\ 91 & 94 & 99 & 104 & 108 & 109 & 108 & 108 \end{bmatrix}$$

Figure 2.3: example of JPEG compression on 8x8 block

When the image is decompressed there are two types of distortion introduced into the image: one is referred to as blocking artifacts the other is false contouring. Blocking artifacts are distortions that appear as large pixel blocks that form the boundaries of each 8x8 block and are very apparent. False contouring is the creation of false edges or outlines where the original image had none. Figure 2.4(c) demonstrates the distortions created from the original image and showing up in the two reconstructed images at different quality levels, the distortions that are introduced into the image go unnoticed at higher quality levels but are very visible at lower ones.

(a) (b) (c)

Figure 2.4 : Illustration of compression using Discrete Cosine Transform:(a) original image, (b) Quality level at 50, (c) quality level at 10 percent. *Dataset of Standard 512x512 Grayscale Test Images*

2.4.2 Discrete Wavelet Transform

The main function behind the JPEG 2000 compression scheme is the Discrete Wavelet Transform (DWT) which represents images as the sum of wavelet functions known as wavelets, at a different location and scale. The lossy Discrete Wavelet Transform used for JPEG 2000 is the biorthogonal Daubechies 9-tap/7-tap filter. The data is calculated by putting the data through a set of high pass and low pass filters. This first pass is applied to the rows of the image and creates two separate image subbands, one with all the high-frequency information and the other with all the low-frequency information. The low-pass subband contains a smaller low-resolution version of the image and the high pass subband contains a residual version of the image and edges. Every row in each subband is then subsampled by a factor of two. Subsampling by a factor of two means that, out of every two pixels only one of them is kept while the other is discarded. This process is then repeated on the columns of both subbands and results in four different subbands shown in Figure.

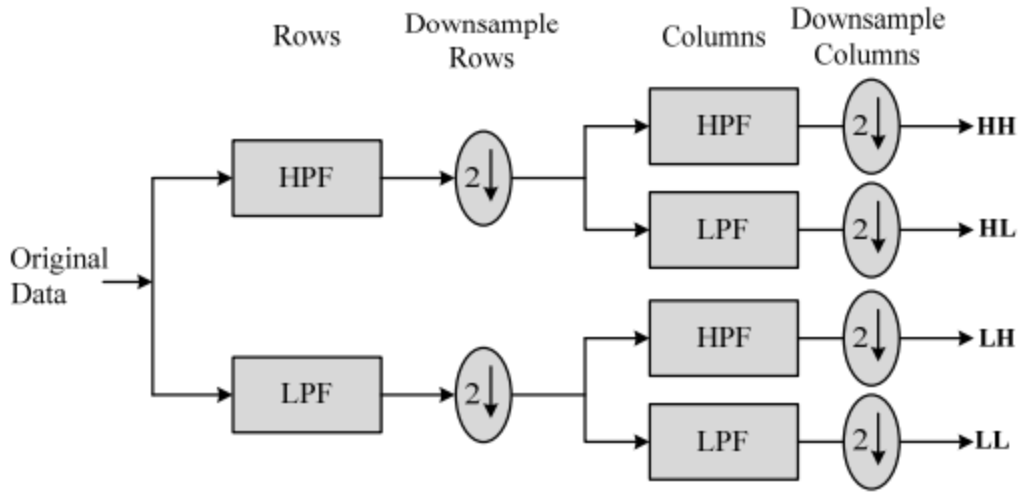

Figure 2.5: Forward DWT

The four subbands obtained from the DWT are the LL, HL, LH, HH. Where the first letter represents the transform in a row and the second letter represents the transform column. L refers to the low pass filter subband and H refers to the High pass filter subband.The LH signal is a low pass subband row and a high pass subband column. Hence, the LH subband contains horizontal elements. Similarly, HL and HH contain vertical and diagonal subbands, respectively. This process can be repeated multiple times on the LL subband to further compress the data.

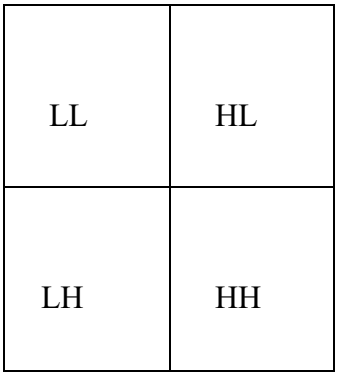

Figure 2.6: dyadic decomposition

In order to reconstruct the original image, the compressed data is up-sampled by a factor of 2. The signal is further passed through the same set of high pass and low pass filters in both rows and columns.

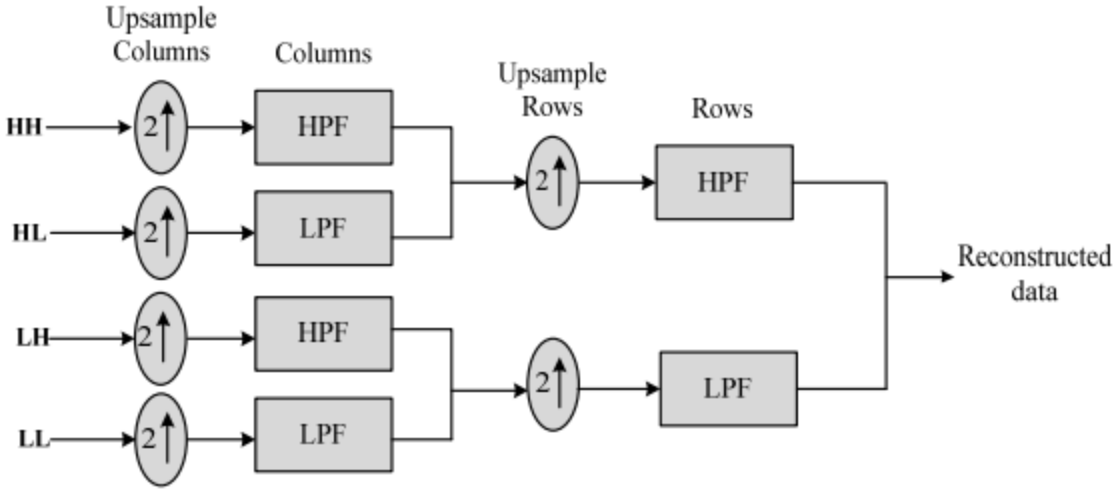

Figure 2.7: backward DWT

Chapter 3:Methods

Most of this project is accomplished by using Python and MATLAB for the implementation and comparison of different compression methods. For my work in Python I will be using four libraries: Numpy, Scipy, Opencv(image reading library), MATLAB plot library, and MATLAB for image comparison. Numpy is a math library that supports large multi-dimensional matrices and many trigonometric functions. This library is being used for the implementation of the Transforms. The Scipy library has its own implementation of the transforms and other compression methods used in this project. Pillow is an open-source image processing library that I will be using to separate the different colors in an image. Matplotlib will be used to demonstrate the differences in image data such as artifacts and noise. MATLAB is a matrix based programming language designed for engineers and scientist and is used to compare the different compression algorithms. To compare the effects of compression on images I used

these three image metrics: the compression ratio (CR), structural similarity index (SSIM), and peak signal to noise ratio (PNSR).

3.1 Compression Ratio

In image compression, it's important to know how much information is reduced when compressing an image. The compression ratio (CR) measures this reduction in information by dividing the file size of the compressed data by the file size of the original data.

$$CR = \frac{Compressed\ Data}{Original\ Data}$$

3.2 PSNR

Peak signal to noise ratio (PSNR) is a measure of the peak error between two signals. It is the ratio between the maximum possible value of a signal(in this case 255) and the power of the distorting noise that affects the quality of its representation usually expressed as the MSE. Because many signals have a very wide range of values the PSNR is usually expressed in terms of the decibel scale. The PNSR is given by this equation (Alain hore 2010) :

$$PSNR = 10\log_{10} \frac{255}{MSE}$$

Mean square error (MSE) is an additional method for comparing compression techniques. The higher the PSNR the higher the image quality. This is because as the MSE approaches zero the

PSNR goes to infinity. It compares the original data and the reconstructed data and returns the level of distortion between the images. The MSE is given by this equation (Alain hore 2010) :

$$MSE = \frac{1}{MN} \sum_{y=1}^{M-1} \sum_{x=1}^{N-1} \left[I(x,y) - I'(x,y) \right]^2$$

Where I(x,y) is the original image, and I'(x,y) is the reconstructed image, and M and N are the dimensions of the image. The lower the MSE the lower the error and the better the image quality.(Alain hore 2010)

3.3 Structural similarity index (SSIM)

SSIM is based on the assumption that the neighboring pixels of the image signals are highly correlated. This correlation carries information about the structure of the image in the visual perception. Therefore, SSIM provides a good approximation to the perceived image quality. The SSIM is given by this equation (Alain hore 2010) :

$$SSIM(x, y) = \frac{(2\mu_x \mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}$$

Where x and y are two images to be compared, μ_x and μ_y are the mean intensity of image x and y respectively, σ_x and σ_y are the standard deviation of image x and y respectively, σ_{xy} is the covariance of image x and y, and c_1 and c_2 are small constants to avoid the denominator being zero equal to zero. The values of the SSIM index are in the range from [0,1]. A value of 0

means no correlation between images, and 1 means that images are equal meaning no data was lost.

Chapter 4: Discussion

To accurately determine the difference in each compression algorithm, a comparison between the PSNR and SSIM at different compression ratios on grayscale images will are used. Using MATLAB to analyze a set of five images at compression ratios of 2:1, 5:1, and 10:1 will compare the average difference of each algorithm at each compression ratio. To demonstrate how each algorithm compacts information histograms of a test image are made using python.

4.1 JPEG VS JPEG2000

The JPEG and JPEG 2000 algorithms both compress the size of an image in different ways. For the JPEG compression, the main factor that affects the size of an image is the quality level of the quantization table. For the JPEG2000 algorithm, the main factor that affects the size of an image is the level of DWT decomposition and irrational values being rounded or converted to zero. The most important measurements of an image's quality are the PSNR and SSIM.

Compression Ratio: 2:1										
Images	Airplane		Baboon		Barbra		Goldengate		Goldhill	
Metrics	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM
JPEG	45.17	0.986	42.39	0.9908	43.83	0.9871	44.93	0.983	43.10	0.984
JPEG2000	54.65	0.9978	45.33	0.9942	52.49	0.998	55.31	0.997	51.91	0.997

Figure 4.1: Compression Ratio: 2:1

Compression Ratio: 5:1										
Images	Airplane		Baboon		Barbra		Goldengate		Goldhill	
Metrics	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM
JPEG	38.59	0.961	31.3401	0.930	35.78	0.955	38.16	0.944	35.71	0.932
JPEG2000	44.63	0.981	32.243	0.912	40.72	0.970	45.377	0.982	39.33	0.9613

Figure 4.2: Compression Ratio: 5:1

Compression Ratio: 10:1										
Images	Airplane		Baboon		Barbra		Goldengate		Goldhill	
Metrics	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM
JPEG	36.11	0.942	28.22	0.881	32.53	0.927	36.47	0.922	33.57	0.895
JPEG2000	39.54	0.957	27.39	0.809	34.73	0.931	40.33	0.953	34.91	0.906

Figure 4.3: Compression Ratio: 10:1

Compression ratio 2:1		
	PSNR	SSIM
JPEG	43.89	0.9864
JPEG2000	51.94	0.9971
Percent Difference	16.80%	1.078%

Figure 4.4 : Average values Compression Ratio: 2:1

Compression ratio 5:1		
	PSNR	SSIM
JPEG	35.91	0.9450
JPEG2000	40.46	0.9615
Percent Difference	11.91%	1.730%

Figure 4.5: Average values Compression Ratio: 5:1

Compression ratio 10:1		
	PSNR	SSIM
JPEG	33.38	0.9140
JPEG2000	35.38	0.9148
Percent Difference	5.817%	0.087%

Figure 4.6: Average values Compression Ratio: 10:1

As the compression ratio increases the average PSNR and SSIM for both compression algorithms decrease. The PSNR and SSIM for the images compressed using theJPEG2000 algorithm were in general better than the JPEG algorithm. However, as the compression ratio increases the percent difference in the average PSNR and SSIM starts to decrease. We note that while JPEG2000 out performs JPEG at all compression ratios the difference between the two shrinks.

4.2 Energy compaction.

The goal of any image compression algorithm is to concentrate the information of an image into as few coefficients as possible. The JPEG and JPEG2000 algorithm compresses images at the cost of information. After the DCT is applied to an image most of the coefficients center around zero and after the data is quantized the majority of the coefficients are concentrated at zero.

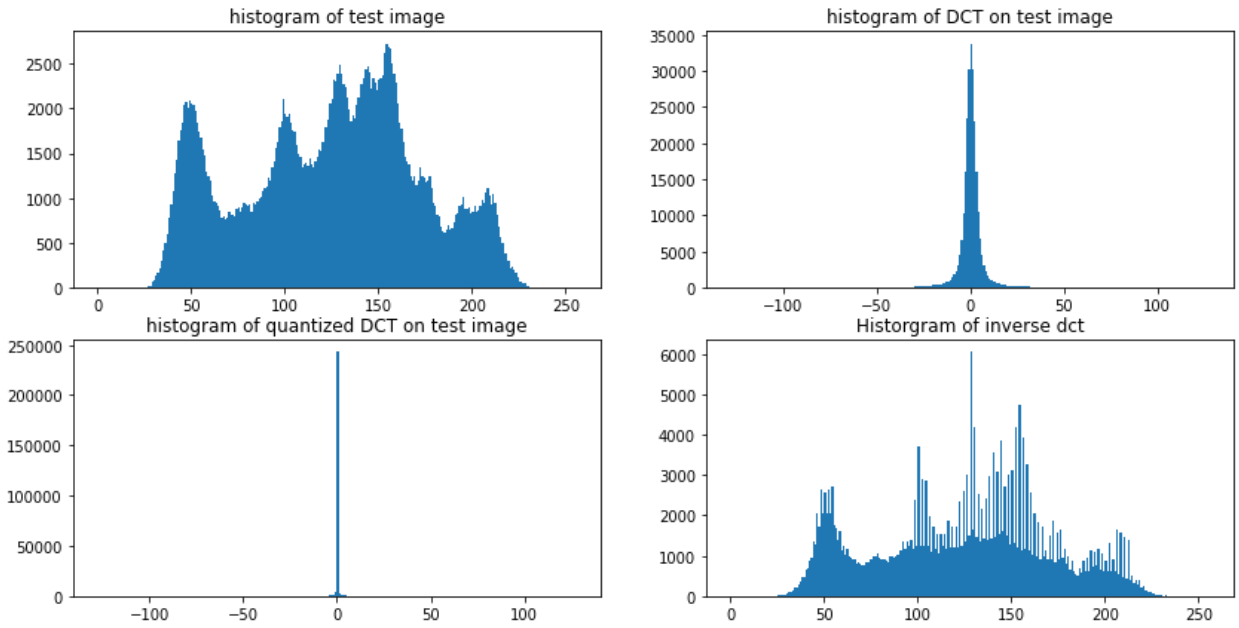

Figure 4.7: JPEG histograms of an image at each stage

When the image data is dequantized and the IDCT is applied you can see how information is lost. Most of the high frequency information is now concentrated in a few pixel values. This happens because most of the high frequency data is discarded during transformation.

Chapter 5: Conclusion

This project presented with many challenges and complex information to research. In order to fully understand the results, it is important to consider these limitations and how they could distort my results. The use of both Python and MATLAB allowed for exploring how compression algorithms work and how information is lost during compression. Learning Google Colab and MATLAB to examine images led to a greater understanding of image processingand compression. This also led to a greater understanding of the strengths and weaknesses of JPEG and JPEG2000 compression algorithms. The JPEG2000 algorithm outperforms the JPEG

algorithm in terms of noise introduced into the image and the structural similarity compared to the uncompressed image.

In future research I would like to better understand the compression process in Python or MATLAB. I would like to create my own compression algorithm by examining and using other DWT and DCT transforms together. Going forward I would like to compare other Lossy Compression algorithms to the JPEG and JPEG2000 compression algorithms.

Appendix:

MATLAB Code:

imgcomparison.m

```
imagefiles = dir('*.tiff');
nfiles = length(imagefiles);
compratio = '10';

jpg_psnr_average = 0;
j2k_psnr_average = 0;
jpg_SSIM_average = 0;
j2k_SSIM_average = 0;
for ii=1:nfiles
   currentfilename = imagefiles(ii).name;
   currentimage = imread(currentfilename);
   [jpgpsnr,j2kpsnr,jpgssim,j2kssim] = comparison(currentimage,
currentfilename,compratio);

   jpg_psnr_average = jpg_psnr_average + jpgpsnr;
   j2k_psnr_average = j2k_psnr_average + j2kpsnr;
   jpg_SSIM_average = jpg_SSIM_average+ jpgssim;
   j2k_SSIM_average = j2k_SSIM_average + j2kssim;

   images{ii} = currentimage;
end
   jpg_psnr_average = jpg_psnr_average/5;
   j2k_psnr_average = j2k_psnr_average/5;
   jpg_SSIM_average = jpg_SSIM_average/5;
   j2k_SSIM_average = j2k_SSIM_average/5;
fprintf('\n The average Peak-SNR value of the jpeg algorithm at
compression ratio of %s  is %0.4f' ,compratio, jpg_psnr_average);
fprintf('\n The average Peak-SNR value of the jpeg2000 algorithm at
compression ratio of %s  is %0.4f' ,compratio, j2k_psnr_average);
fprintf('\n The average SSIM value of the jpeg algorithm at
compression ratio %s  is %0.4f' ,compratio, jpg_SSIM_average);
fprintf('\n The average SSIM value value of the jpeg2000 algorithm at
compression ratio of %s  is %0.4f' ,compratio, j2k_SSIM_average);
```

Comparison.m

```
function [jpgpsnr,j2kpsnr,jpgssim,j2kssim]
=comparison(img,filename,compratio)

    temp = strrep(filename,'.tiff','CR');
    temp = temp + string(compratio);
    jpegfile = temp + '.jpg';
    jpeg2000file = temp + '.j2k';

    jpg = imread(jpegfile);
    j2k = imread(jpeg2000file);

    [ jpgpsnr, snr] = psnr(img,jpg);
    fprintf('\n The Peak-SNR value of %s at a compression ratio of %s
is %0.4f', jpegfile ,compratio, jpgpsnr);

    [ j2kpsnr, snr] = psnr(img,j2k);

    fprintf('\n The Peak-SNR value of %s at a compression ratio of %s
is %0.4f', jpeg2000file ,compratio, j2kpsnr);

    [jpgssim,ssimmap] = ssim(img,jpg);
    fprintf('\n The SSIM  valueof %s at a compression ratio of %s  is
%0.4f', jpegfile ,compratio,jpgssim);

    [j2kssim,ssimmap] = ssim(img,j2k);
    fprintf('\n The SSIM value of %s at a compression ratio of %s  is
%0.4f', jpeg2000file ,compratio,j2kssim);
end
```

imgcompression.m

```
function imgcompress(img,filename,compratio,quality)

    temp = strrep(filename,'.tiff','CR');
    temp = temp + string(compratio);
    jpegfile = temp + '.jpg';
    jpeg2000file = temp + '.j2k';
    imwrite(img,jpegfile,'quality',quality);

imwrite(img,jpeg2000file,'CompressionRatio',compratio,'ReductionLevel
s',5,'TileSize',[128 128]);

end
```

PYTHON CODE:

```
# -*- coding: utf-8 -*-
"""project

Automatically generated by Colaboratory.

Original file is located at
   https://colab.research.google.com/drive/1HlSgE0cJnBGQlh_kJkbR1cebO0PXyc1j
"""

import numpy as np
import os
import math
from matplotlib import pyplot as plt
from matplotlib import pylab
import scipy
import pywt
import cv2
from pywt import dwt2, idwt2
import PIL
from PIL import Image

from numpy import zeros
from numpy import r_
from scipy import fftpack
from scipy import signal
```

```
from scipy import misc

pylab.rcParams['figure.figsize'] = (14.0, 7.0)

def dct2d(matrix):
 """
 calculates the Forward dct of an 8x8 image block
 """
 dct_matrix = np.zeros((8,8)).astype('int16')
 for i in range(0,8):
   for j in range(0,8):
     matrix[i][j] -=128


 for u in range(0, 8):
   for v in range(0, 8):
     temp = 0
     for x in range(0, 8):
         for y in range(0, 8):
           temp += matrix[x][y] * math.cos((math.pi*(2 * x + 1) * u)/(16)) *
math.cos((math.pi*(2 * y + 1) * v)/(16))

     if (u==0):
       Cu = 1/math.sqrt(2)
     else:
      Cu = 1

     if v == 0:
      Cv = 1/math.sqrt(2)
     else:
       Cv = 1
     dct_matrix[u][v] =  (1/4)*Cu*Cv*temp

 return dct_matrix

def idct2d(matrix):


 """
 calculates the inverseForward dct of an 8x8 image block
 """



 temp = 0
 Cu = 1/math.sqrt(2)
 Cv = 1/math.sqrt(2)
 dct_matrix = np.zeros((8,8)).astype(float)
 for x in range(0, 8):
     for y in range(0, 8):
       temp = 0
```

```
       for u in range(0, 8):
         for v in range(0, 8):
           if (u==0):
             Cu = 1/math.sqrt(2)
           else:
             Cu = 1

           if v == 0:
             Cv = 1/math.sqrt(2)
           else:
             Cv = 1
           temp += matrix[u][v] * Cu*Cv*math.cos((math.pi*(2 * x + 1) * u)/(16)) *
math.cos((math.pi*(2 * y + 1) * v)/(16))

       dct_matrix[x][y] =  (1/4)*temp
 for i in range(0,8):
   for j in range(0,8):
     matrix[i][j] +=128
 return dct_matrix

def fdct(in_matrix):
 """
 calculates the Forward dct of an 8x8 image block.
 """
 x = np.copy(in_matrix)

 for i in range(0,8):
   for j in range(0,8):
     x[i][j] -=128


 return  np.around( scipy.fftpack.dct( scipy.fftpack.dct( x, axis=0, norm='ortho' ), axis=1,
norm='ortho' ),decimals=0)

def ifdct(matrix):
 """
 calculates the inverse dct of an 8x8 image block.
 """
 x = np.copy(matrix).astype('int16')


 x = scipy.fftpack.idct( scipy.fftpack.idct( x, axis=0 , norm='ortho'), axis=1 ,
norm='ortho')
 for i in range(0,8):
   for j in range(0,8):
     x[i][j] +=128
 return np.around(x, decimals=0)

def mse(original, reconstructed):
  """
```

```
 Calculates the mean square error (MSE) between two images.
 """
  error = np.sum((original.astype(float) - reconstructed.astype(float)) ** 2)
  error /= float(original.shape[0] * original.shape[1])


  return error

def Psnr(original, reconstructed):
 """
 Calculates the peak signal-to-noise ratio between two images
 """
 error = mse(original, reconstructed)
 if(error == 0):
   return 100

 psnr = 20 * log10(255 / math.sqrt(mse))
 return psnr


def quantize(in_matrix,qr):
 """
 quantizes the information in an 8x8 block
 """

 matrix = np.zeros((8,8)).astype(int)

 quantization = np.array([
   [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]])

 if(qr > 50):
   qr = (100-qr)/50
   for i in range(0,8):
     for j in range(0,8):
       if(quantization[i][j]* qr > 255):
         quantization[i][j] = 255
       else:
         quantization[i][j] = ( quantization[i][j]* qr)
 elif(qr < 50):
   qr = 50/qr
   for i in range(0,8):
    for j in range(0,8):
     if(quantization[i][j]* qr > 255):
```

```
      quantization[i][j] = 255
    else:
      quantization[i][j] = ( quantization[i][j]* qr)

 for i in range(0,8):
   for j in range(0,8):
     matrix[i][j] = in_matrix[i][j]/quantization[i][j]
 return np.around(matrix, decimals=0)

def unquantize(in_matrix,qr):
 """
 unquantizes information in an 8x8 block
 """

 matrix = np.zeros((8,8)).astype(int)
 quantization = np.array([
   [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]])
 if(qr > 50):
   qr = (100-qr)/50

   for i in range(0,8):
     for j in range(0,8):
       if(quantization[i][j]* qr > 255):
         quantization[i][j] = 255
       else:
         quantization[i][j] = ( quantization[i][j]* qr)
 elif(qr < 50):
   qr = 50/qr

   for i in range(0,8):
    for j in range(0,8):
     if(quantization[i][j]* qr > 255):
       quantization[i][j] = 255
     else:
       quantization[i][j] = ( quantization[i][j]* qr)

 return np.multiply(in_matrix,quantization)


def scalar_quantization(im,qr):
 """
 quantizes the information of an image block
 """
```

```
 (h, w) = im.shape
 matrix = np.zeros((h,w)).astype(float)

 for i in range(0,w):
   for j in range(0,h):

     if im[i][j] >= 0:
       sign = 1
     else:
       sign = -1

     matrix[i][j] = sign * math.floor(abs(im[i][j]) / qr)
 return matrix

def inverse_scalar_quantization(im,qr):

 """
inverse quantizes the information of an image block
 """

 (h, w) = im.shape
 matrix = np.zeros((h,w)).astype(float)

 for i in range(0, w):
   for j in range(0, h):

      matrix[i][j] = im[i][j] * qr



 return matrix

def maketuple(inputmatrix):
 """
 Seperates image into tuple
 """
 (h, w) = inputmatrix.shape


 cA = inputmatrix[0:int(h/2),0:int(w/2)]
 cV = inputmatrix[int(h/2):(h),0: int(w/2)]
 cH = inputmatrix[0:int(h/2),int(w/2):(w)]

 cD = inputmatrix[int(h/2):(h),int(w/2):(w)]



 return (cA, (cH, cV, cD))

im = cv2.imread('/content/sample_data/lena512.bmp',0).astype(int)
```

```
plt.figure()
plt.imshow(im,cmap='gray',vmax= 255,vmin = 0)

"""DCT In place example"""

imrange = im.shape
dct_img = np.zeros(im.shape)


#DCT  8x8 Test
for i in r_[:imrange[0]:8]:
   for j in r_[:imrange[1]:8]:
       dct_img[i:(i+8),j:(j+8)] = fdct( im[i:(i+8),j:(j+8)] )

plt.subplot(121),plt.imshow(im, cmap='gray',vmax= 255,vmin = 0)
plt.title( 'Test Image'), plt.xticks([]),plt.yticks([])




plt.subplot(122),plt.imshow(dct_img, cmap='gray',vmax= 255,vmin = 0)
plt.title( "DCT preformed on 8x8 blocks of the image"), plt.xticks([]),plt.yticks([])
plt.suptitle("test")

#quantization of image

q = np.zeros(imrange)
for i in r_[:imrange[0]:8]:
   for j in r_[:imrange[1]:8]:
       q[i:(i+8),j:(j+8)] = quantize( dct_img[i:(i+8),j:(j+8)],50)

#unquantization of image

de = np.zeros(imrange)
for i in r_[:imrange[0]:8]:
   for j in r_[:imrange[1]:8]:
       de[i:(i+8),j:(j+8)] = unquantize( q[i:(i+8),j:(j+8)],50)
idct_img = np.zeros(imrange)
#inverse discrete cosine
for i in r_[:imrange[0]:8]:
   for j in r_[:imrange[1]:8]:
       idct_img[i:(i+8),j:(j+8)] = ifdct( de[i:(i+8),j:(j+8)] )

plt.subplot(121),plt.imshow(im, cmap='gray',vmax= 255,vmin = 0)
plt.title( 'Test Image'), plt.xticks([]),plt.yticks([])
```

```
plt.subplot(122),plt.imshow(idct_img, cmap='gray')
plt.title( "Inverse DCT"), plt.xticks([]),plt.yticks([])

imghist,bins = np.histogram(im,256,[0,256])
plt.subplot(221),plt.hist(im.ravel(),256,[0,256])
plt.title( "histogram of test image")


imghist,bins = np.histogram(dct_img,256,[0,256])

plt.subplot(222),plt.hist(dct_img.ravel(),256,[-128,128])
plt.title( "histogram of DCT on test image")


imghist,bins = np.histogram(q,256,[0,256])

plt.subplot(223),plt.hist(q.ravel(),256,[-128,128])
plt.title( "histogram of quantized DCT on test image")

imghist,bins = np.histogram(idct_img,256,[0,256])

plt.subplot(224),plt.hist(idct_img.ravel(),256,[0,256])
plt.title( "Historgram of inverse dct")

"""DCT example"""

DCTex = np.copy(im[i:(i+8),j:(j+8)])

print(DCTex)
print(fdct(DCTex))
print(quantize(fdct(DCTex),50))
print(unquantize(quantize(fdct(im[i:(i+8),j:(j+8)]),50),50))
print(ifdct(unquantize(quantize(fdct(im[i:(i+8),j:(j+8)]),50),50)))

plt.subplot(121),plt.hist(im.ravel(),256,[0,256])
plt.title( 'Test Image')




plt.subplot(122),plt.hist(dct_img.ravel(),256,[-128,128])
plt.title( "DCT preformed 8x8 blocks of an image")

pos = 0

# grab 8x8 data from image
plt.figure()
```

```
plt.imshow(im[pos:pos+8,pos:pos+8],cmap='gray',vmax= 255,vmin = 0)

plt.title( "An 8x8 Image block")

# Display the dct of that block
plt.figure()

plt.imshow(dct_img[pos:pos+8,pos:pos+8],cmap='gray',vmax= 255,vmin = 0 )
plt.title( "An 8x8 DCT block")

plt.figure()

plt.imshow(idct_img,cmap='gray',vmax= 255,vmin = 0  )
plt.title( "DCTs of the image")

coeffs = pywt.dwt2(im, 'bior4.4',mode = 'per')

fullimg = cv2.hconcat([cv2.vconcat([coeffs[0],coeffs[1][0]]),
cv2.vconcat([coeffs[1][1],coeffs[1][2]])])
plt.imshow(fullimg, cmap='gray',)

dwt_img = np.zeros(imrange)
idwt_img = np.zeros(imrange)

# Do 128x128 dwt on image (in-place)
for i in r_[:imrange[0]:128]:
   for j in r_[:imrange[1]:128]:


     temp = pywt.dwt2(im[i:(i+128),j:(j+128)], 'bior4.4',mode = 'per')

     dwt_img[i:(i+128),j:(j+128)] =
cv2.hconcat([cv2.vconcat([scalar_quantization(temp[0],30),scalar_quantization(temp[1][0],30)
]), cv2.vconcat([scalar_quantization(temp[1][1],30),scalar_quantization(temp[1][2],30)])] )


# Do 128x128 idwt on image (in-place)
for i in r_[:imrange[0]:128]:
   for j in r_[:imrange[1]:128]:

     tempBlock = inverse_scalar_quantization(dwt_img[i:(i+128),j:(j+128)],30)



     temp = pywt.idwt2(maketuple(dwt_img[i:(i+128),j:(j+128)]), 'bior4.4',mode = 'per')
     idwt_img[i:(i+128),j:(j+128)] += temp

plt.imshow(idwt_img, cmap='gray',)
```

```
test1 = PIL.Image.open('/content/sample_data/airplane.tiff')
test1.show()
test1.save('test1.jpg', "JPEG",quality=50)

test2 = PIL.Image.open('//content/sample_data/airplane.tiff')
test2.show()
test2.save('test2.jpg', "JPEG",quality=10)

testjpeg1 = PIL.Image.open('/content/test1.jpg')
testjpeg2 = PIL.Image.open('/content/test2.jpg')

plt.subplot(131),plt.imshow(test1,cmap ='gray')
plt.title(''), plt.xticks([]),plt.yticks([])


plt.subplot(132),plt.imshow(testjpeg1,cmap ='gray')
plt.title( ""), plt.xticks([]),plt.yticks([])

plt.subplot(133),plt.imshow(testjpeg2,cmap ='gray')
plt.title( ""), plt.xticks([]),plt.yticks([])
```

Images used:

Images used in PSNR and SSIM test:

Dataset of Standard 512x512 Grayscale Test Images

Image used in histogram:

Dataset of Standard 512x512 Grayscale Test Images

Bibliography

Arora, Sunny, and Gaurav Kumar. “Review of Image Compression Techniques.” *International Journal of Recent Research Aspects*,
vol. 5, no. 1, Mar. 2018, pp. 185–188. *EBSCOhost*,

"Dataset Of Standard 512X512 Grayscale Test Images." *Dataset of Standard 512x512 Grayscale Test Images*. University of Granada, 23 July 03. Web.

Garg, Mamta. "Performance Analysis of Chrominance Red & Chrominance Blue in JPEG." *World Academy of Science, Engineering and Technology, WASET* 43 (2008): 110-113.

Hore, Alain, and Djemel Ziou. "Image Quality Metrics: PSNR vs. SSIM." *2010 20th International Conference on Pattern Recognition* (2010): 2366-369. Print.

Jeevitha. “A Survey on Lossless and Lossy Image Compression Techniques.” *International Journal of Advanced Research in Computer Science*,
vol. 9, no. 1, Feb2018 Special 2018, pp. 135–137.

N. Ahmed, T. Natarajan and K. R. Rao, "Discrete Cosine Transform," in IEEE Transactions on Computers, vol. C-23, no. 1, pp. 90-93, Jan. 1974, doi: 10.1109/T-C.1974.223784.

Pappas M., Pitas I., “Digital Color Restoration of Old Paintings,” IEEE Transactions on Image Processing, vol. 9, no. 2, pp. 291-294, 2000

PUJAR, JAGADISH H, and LOHIT M KADLASKAR. “A New Lossless Method Of Image Compression and Decompression Using Huffman Coding Techniques .” *Journal of Theoretical and Applied Information Technology*, vol. 15, no. 1, 2010, doi:jatit.org.

Rathee, Monika. "Image Compression Using Discrete Haar Wavelet Transforms." *International Journal of Engineering and Innovative Technology (IJEIT)* Volume 3.Issue 12 (2014): 47-51. Print.

Roy, Anamitra Bardhan, et al. "Comparison of FFT, DCT, DWT, WHT compression techniques on electrocardiogram and photoplethysmography signals." *IJCA Special Issue on International Conference on Computing, Communication and Sensor Network CCSN*. 2012.

Salomon, David. "Image Compression." *Data Compression: The Complete Reference*. N.p.: Springer, 2007. 298-378. Print.

Singh, Hari Kumar, et al. "Discrete cosine transform and discrete Fourier transform of RGB image." *International Journal of Computer Applications* (2012).