Image Compression Using Discrete Wavelet Transforms

Capstone Project Final Report

Salma Taoufiq

Supervised by

Dr. Naeem N. Sheikh

School of Science and Engineering

Al Akhawayn University

Morocco

Spring 2016

Image Compression Using Discrete Wavelet Transforms

Capstone Project Final Report

Approved by the supervisor:

Naeem Nisar Sheikh

1 Acknowledgments

This is an opportunity to show some of my sincere gratitude to my parents for everything they have done and given up for me. I am also thankful to my sister and brother: my mentors in spite of the distance separating us.

I am extremely grateful to Dr. Sheikh, without whom this project would have not seen the light of day. Thank you for the long hours of brainstorming and guidance. Thank you for believing in me.

I would also like to thank my friends, family, professors, and AUI staff.

These fours years were one memorable ride.

Contents

2 Abstract

Image processing is one of the areas that rely on mathematics. Throughout this project, focus will be on a specific topic in that field: image compression. Our aim is to examine how discrete wavelet transforms in general, and the Haar wavelet in particular, apply to image compression, and how linear algebra can be employed to make this compressor more powerful while yielding compressed but visually acceptable results.

First, this project analyzes the details of the processes of averaging and differencing used in the Haar transform along with their different possible implementations. Then, we investigate each variant of that procedure to decide on which one is the most conclusive. Hence, extensive testing needs to be performed based on a set of criteria such as the compression ratio, and the visual quality of the resulting images.

However, the Haar wavelet transform can only process $2^n \times 2^m$ matrices. We will present a novel method whose purpose is to overcome this obstacle to some extent. This new variant uses reduction by 3; that is averaging 3 values instead of only 2 to have more freedom in terms of the size of the image. Finally, we look at the discrete cosine transform (DCT) which is quite different from the wavelet-based image compression techniques. This transform is actually the key to the JPEG standard baseline compression process.

3 Introduction

Mathematics is everywhere: from forensics to entomology. Mathematical sciences, thus, seem like a really good start in attempting to satiate one's curiosity about the world around us. Isn't the answer to the ultimate question of life, the universe, and everything, after all, a number: 42, as stated in Douglas Adams' sci-fi book The Hitchhiker's Guide to the Galaxy?

It is, then, not surprising that mathematics are used in data compression. Now, more than ever, we have been trying to immortalize every moment -even the most mundane ones- by taking hundreds of images and videos. Multimedia content is ruling our world. This growing demand is concurrent to problems of insufficient network bandwidth and storage capacity of memory devices. Consequently, the field of data compression has been gaining increasing attention as it has become an essential part of computer technology due to the increasing need for fast data transmission over networks.

In computer science and information theory, data compression is the process of encoding information using fewer bits than the original version; making this a useful technique that contributes to reducing the consumption of many costly resources. For the past few years, the demand for powerful compression has risen side by side with the demand for aesthetically appealing, and engaging user platforms that have a short enough response time that a user is not left waiting.

Image compression particularly is an important field of image processing which can be performed using discrete transforms, namely, the Haar transform. An image compressor is a key technology that can substantially help with file size and bandwidth usage reduction with the assumption that loss of precision is okay. As long as the user is willing to compromise some of the images precision details, we can afford to implement lossy compression and get to significantly reduce the size of an image.

In order to implement the image compressor, we will follow a specific methodology whose most essential step is heavy consultation of mathematical literature as it is necessary to have a solid understanding of the wavelet transforms to implement them. Working on this tool is an opportunity to learn about and use different mathematical concepts, data structures, and algorithms. Testing the image compressor will be based on a few criteria such as the compression ratio.

4 Steeple Analysis

Discrete transforms are an efficient way for compressing images. Image compression using those transforms helps substantially with file size and bandwidth usage reduction, when we can afford losing precision to some extent. On the other hand, with no compression, there is a need for bigger drives, and usage of more resources; thereby having negative side-effects on the economic as well as the environmental levels.

Large uncompressed images may be loaded by some electronic devices very slowly. CD drives for instance can only process data at a given specific rate and cannot display large images in real time. Hard drives also find loading uncompressed images quickly problematic.

Compressing those image files enables making loading data fast even for “slow” devices. Moreover, for some web hosts that have a low data transfer rate, compressed images are essential to having a fully operational website. Therefore, we can conclude that, technologically speaking, image compression presents an extremely significant advantage: file size reduction.

It can allow regular users to send many compressed images by mail, etc. Webmasters can also create sites that are rich in images without consuming much bandwidth or using up much storage space.

5 Methodology

First of all, in building the image compressor, we need to understand the math behind it. Choosing the discrete wavelet transforms that are to be exploited in making our lossy image compressor is important as well. The main transform we will focus on is the Haar transform. A prerequisite to using it in the image compression program is to understand its intricacies. There are many variants of the Haar wavelet transform that we will go through to find the most performant one. The differences between the variants are mainly in the method adopted in the process of averaging and differencing which is the very essence of the Haar wavelet.

The design phases main concern would be to analyze the process of using those transforms, come up with the algorithms -that will be presented later on- for each one of them, and implement them using GNU Octave, a high-level interpreted language. Octave is a free software alternative to the commercial Matlab. After implementation, testing will primarily be concerned with two parameters: the resulting compression ratios, and the nature of images for which the loss of precision in the compressor is bearable from a human eye point of view.

The desired outcome would be to find the best alternative among those Haar variants. After analyzing the performance of the different implementations of the Haar wavelet transform, we will check the possibility of reduction by 3; that is averaging 3 values instead of only 2 to have more freedom in terms of the size of the image. Indeed, Haar only works with $2^n \times 2^m$ matrices. Additionally, we will look at other discrete wavelet transforms such as Daubechies' and how they compare to the one developed by Alfred Haar. Finally, we wi look at the discrete cosine transform (DCT) which is quite different from the wavelet-based image compression techniques. The DCT is actually the key to the JPEG standard baseline compression process.

6 Wavelet Transforms

In everyday life, when one hears "wavelet", they understand a "small water wave". They do not wander off to things like image compression for instance. However, in mathematics, wavelets refer to short wavelike functions that can be scaled and translated. They are called wavelets due to their characteristic of integrating to 0, "waving" up and down the $x - axis$. In fact, wavelet transforms can take any signal and express it based on those scaled and translated wavelets. The result of this procedure is a representation of the given signal at different scales [1].

Wavelet transforms are definitely very important computational tools. A transform is a familiar concept to mathematicians. It is a widely used standard mathematical tool that helps with solving problems in multiple areas. The fundamental idea of transforms is changing a mathematical quantity (it could be a number, a vector, a function, etc.) to another form where it may be unrecognizable, but would present useful features. This transformed quantity is, thus, used to solve the problem on hand, or to perform some helpful calculation. The result can then be transformed back to the original form [2, p.104].

Initially, wavelets were solely in mathematics. Now, the extent of their usage has reached areas such as seismology, image processing, quantum mechanics, signal processing, non-stationary signals in particular, and data compression. Among their applications in real-life, we can give two examples:

- $*$ Because of the large number and size of fingerprint images, the FBI developed a fingerprint compression specification called Wavelet Scalar Quantization which is based on wavelet compression [3].

- $*$ In order to determine accurately the redshifts of galaxies, it is necessary to identify the key lines of their spectra. This identification problem needs to be tackled in an automated and reliable manner because it requires large sky surveys which produce a huge volume of data. Consequently, a wavelet-based method, called the Darth Fader algorithm, has been established for estimating redshifts of galaxy spectra [4].

Wavelet transforms are employed profusely in image processing and compression. Actually, they enable computers to store images in many scales of resolution. They decompose a given image into a number of details and approximations. Since many of the compression processes are quite similar to each other, investigating any one algorithm is enough to get a lot of insight into the field of image compression as a whole. Therefore, through this capstone project, focus will be on the Haar wavelet transform, its usage in image compression, as well as the performance of its different variants.

7 Haar Transform

The Haar transform is one of the simplest discrete wavelet transforms. It is based on the idea of decomposing a signal into two components: one is the average (approximation), and the other is the difference (detail). The Haar wavelet transform represents the first discrete wavelet transform. It was invented by the Hungarian mathematician Alfred Haar [6, p.2]. It works on an input of size 2^n.

In order to show how the Haar wavelet transform works, we will take the following one-dimensional array r:

$$r = \begin{pmatrix} 156 & 159 & 158 & 155 & 158 & 156 & 159 & 158 \end{pmatrix}$$

1.
   - Group the columns in pairs: [156 159], [158 155], [158 156], [159 158]
   - Average each of these pairs, then replace the four first columns of r with those values and compute $\frac{1}{2}$ the difference of each pair, then replace the last four columns of r with those. Our array r becomes:

     $$r' = \begin{pmatrix} 157.5 & 156.5 & 157 & 158.5 & -1.5 & 1.5 & 1 & 0.5 \end{pmatrix}$$

     Terminology: The first four entries are called *approximation coefficients*. The last four ones are called *detail coefficients*.

2.
   - Group the first four columns of r' in pairs: [157.5 156.5], [157 158.5]
   - Average each pair and replace the first two columns of r' with the resulting values. Replace the other two columns with $\frac{1}{2}$ the difference of each pair. Denote the

result by r'' .

$$r'' = \begin{pmatrix} 157 & 157.75 & 0.5 & -0.75 & -1.5 & 1.5 & 1 & 0.5 \end{pmatrix}$$

3. • Group the first two columns of r'' in a pair: [157 157.75]

   • Average the pair and replace the first column of r'' with the result. Replace the second column with $\frac{1}{2}$ the difference of the pair. Denote the result by r''' :

$$r''' = \begin{pmatrix} 157.375 & -0.375 & 0.5 & -0.75 & -1.5 & 1.5 & 1 & 0.5 \end{pmatrix}$$

As we can see, the resulting array has only one entry that is big; the leftmost one. All the other entries, the detail coefficients, are close to 0. The beauty of what we have done so far lies in the fact that it is completely reversible.

We have been able to reconstruct the original array. We call this type of compression *lossless*. For greater reduction, we can implement *lossy* compression by using a threshold value. Pick a number ϵ such that $\epsilon > 0$. We will now set all the entries in r''' whose absolute value is at most ϵ to 0. This new array will, consequently, have more 0 entries This array is even more compressed.

Let $\epsilon = 0.5$. So, r''' becomes:

$$r''' = \begin{pmatrix} 157.375 & 0 & 0 & -0.75 & -1.5 & 1.5 & 1 & 0 \end{pmatrix}$$

If we reverse the process to get the original, we get:

$$\begin{pmatrix} 155.88 & 158.88 & 158.88 & 155.88 & 157.62 & 155.62 & 158.12 & 158.12 \end{pmatrix}$$

which is really close to our original array r. We have lost some precision in order to gain in compression.

At this point one might wonder how this can be applied to pixels and images. Luckily, images can be represented in the form of matrices. In this project, we will be working principally with two types of images: grayscale or colored.

7.1 Image Matrix Representation

If we are to process a grayscale image, each entry of the matrix determines the intensity of the pixel it represents. This intensity is expressed by an integer ranging from 0 to 255; with 0 indicating black which is the color with minimal intensity, and 255 indicating white which the color with maximal intensity. Thus, we have 256 different levels and intensities of gray. 256 levels are more than enough for the majority of common images; however, more might be needed for applications in fields like medicine.

As for color images, assuming that we are following the RGB color system, we can represent them using three matrices; one for each: red, green, and blue (RGB). The entries of each one of the three matrices are integers between 0 and 255 again, determining the intensity of the pixel with respect to the color defined by the matrix. Therefore, we can see that in the RGB system, we can represent up to $256^3 = 2^{24} = 16,777,216$ different colors.

Using Octave, one line of code is enough to generate the matrix representation of the image on hand:

```
image = imread(  image.png  )
```

7.2 Image Compression Using the Haar Wavelet Transform

Actually, the array r that we have compressed earlier is the first row of the upper left 8×8 corner of a well-known grayscale image. It is the standard cameraman test image. Courtesy of the Massachusetts Institute of Technology.

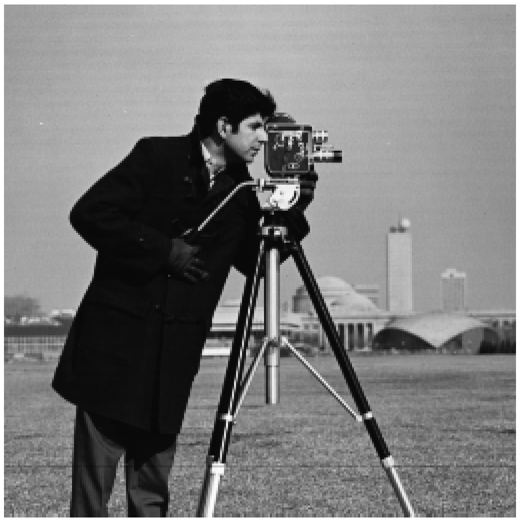


Figure 1: 'cameraman.tif' famous test image

The 8×8 matrix I representing that corner is::

$$\begin{pmatrix} 156 & 159 & 158 & 155 & 158 & 156 & 159 & 158 \\ 160 & 154 & 157 & 158 & 157 & 159 & 158 & 158 \\ 156 & 159 & 158 & 155 & 158 & 156 & 159 & 158 \\ 160 & 154 & 157 & 158 & 157 & 159 & 158 & 158 \\ 156 & 153 & 155 & 159 & 159 & 155 & 156 & 155 \\ 155 & 155 & 155 & 157 & 156 & 159 & 152 & 158 \\ 156 & 153 & 157 & 156 & 153 & 155 & 154 & 155 \\ 159 & 159 & 156 & 158 & 156 & 159 & 157 & 161 \end{pmatrix}$$

Compressing this matrix or the whole image follows the same algorithm that we applied to the array r. We just need to repeat the procedure for the rest of the matrix's rows. Then do the same for its columns. The resulting matrix looks like:

$$\begin{pmatrix} 156.86 & -0.2031 & -0.1562 & -0.0625 & 0.75 & -0.1875 & -0.25 & -0.5 \\ 0.6406 & -0.1718 & 0.2812 & -0.3125 & 0 & 0.6875 & 0.25 & 0.75 \\ \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} \\ -0.2812 & -0.2812 & -0.4375 & 0.75 & 0 & -0.625 & 0.75 & \boxed{0} \\ -0.125 & \boxed{0} & 0.375 & -0.375 & -2.25 & 1 & 1 & 0.25 \\ -0.125 & \boxed{0} & 0.375 & -0.375 & -2.25 & 1 & 1 & 0.25 \\ 0.06250 & 0.0625 & -0.375 & -0.25 & 0.75 & -0.5 & 1.75 & 1.75 \\ -1.625 & 0.375 & -1 & 0.25 & 0.75 & 0.75 & 0.25 & 0.75 \end{pmatrix}$$

As we can see, the resulting matrix has multiple 0 entries, and most of the other entries are actually close to 0. This result can be attributed to the usage of differencing, as well as to the fact that usually, an images adjacent pixels do not differ by much.

Our process can be implemented using loops in order to go over all the rows and columns of the image, which is a bit tedious. However, we can simplify this and make it faster using linear algebra. It may seem daunting to use linear algebra but the process is pretty straightforward. Matrix multiplication is the key here. We will use what is called the *Haar wavelet transformation matrix* [8, p.3]:

$$H_1 = \begin{pmatrix} \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{2} & 0 & 0 & 0 & -\frac{1}{2} & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 & 0 & -\frac{1}{2} & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & -\frac{1}{2} & 0 \\ 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} \\ 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & -\frac{1}{2} \end{pmatrix}$$

As we can see $I \cdot H_1$ gives the following:

$$I \cdot H_1 = \begin{pmatrix} 157.5 & 156.5 & 157 & 158.5 & -1.5 & 1.5 & 1 & 0.5 \\ 157 & 157.5 & 158 & 158 & 3 & -0.5 & -1 & 0 \\ 157.5 & 156.5 & 157 & 158.5 & -1.5 & 1.5 & 1 & 0.5 \\ 157 & 157.5 & 158 & 158 & 3 & -0.5 & -1 & 0 \\ 154.5 & 157 & 157 & 155.5 & 1.5 & -2 & 2 & 0.5 \\ 155 & 156 & 157.5 & 155 & 0 & -1 & -1.5 & -3 \\ 154.5 & 156.5 & 154 & 154.5 & 1.5 & 0.5 & -1 & -0.5 \\ 159 & 157 & 157.5 & 159 & 0 & -1 & -1.5 & -2 \end{pmatrix}$$

This matrix represents Step 1 of our procedure.

Now, let H_2 be the following:

$$H_2 = \begin{pmatrix} \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & 0 & -\frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & -\frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

We, thus, get that $I \cdot H_1 \cdot H_2$ is the following:

$$I \cdot H_1 \cdot H_2 = \begin{pmatrix} 157 & 157.75 & 0.5 & -0.75 & -1.5 & 1.5 & 1 & 0.5 \\ 157.25 & 158 & -0.25 & 0 & 3 & -0.5 & -1 & 0 \\ 157 & 157.75 & 0.5 & -0.75 & -1.5 & 1.5 & 1 & 0.5 \\ 157.25 & 158 & -0.25 & 0 & 3 & -0.5 & -1 & 0 \\ 155.75 & 156.25 & -1.25 & 0.75 & 1.5 & -2 & 2 & 0.5 \\ 155.5 & 156.25 & -0.5 & 1.25 & 0 & -1 & -1.5 & -3 \\ 155.5 & 154.25 & -1 & -0.25 & 1.5 & 0.5 & -1 & -0.5 \\ 158 & 158.25 & 1 & -0.75 & 0 & -1 & -1.5 & -2 \end{pmatrix}$$

$I \cdot H_1 \cdot H_2$ is basically equivalent to applying of steps 1 and 2 to the rows of matrix I.

Finally, let's consider H_3

$$H_3 = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & -\frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

So all three steps applied to the rows of I give the following: $I \cdot H_1 \cdot H_2 \cdot H_3$:

$$I \cdot H_1 \cdot H_2 \cdot H_3 = \begin{pmatrix} 157.375 & -0.375 & 0.5 & -0.75 & -1.5 & 1.5 & 1 & 0.5 \\ 157.625 & -0.375 & -0.25 & 0 & 3 & -0.5 & -1 & 0 \\ 157.275 & -0.375 & 0.5 & -0.75 & -1.5 & 1.5 & 1 & 0.5 \\ 157.625 & -0.375 & -0.25 & 0 & 3 & -0.5 & -1 & 0 \\ 156 & -0.25 & -1.25 & 0.75 & 1.5 & -2 & 2 & 0.5 \\ 155.875 & -0.375 & -0.5 & 1.25 & 0 & -1 & -1.5 & -3 \\ 155.875 & 0.625 & -1 & -0.25 & 1.5 & 0.5 & -1 & -0.5 \\ 158.125 & -0.125 & 1 & -0.75 & 0 & -1 & -1.5 & -2 \end{pmatrix}$$

Let H be: $H_1 \cdot H_2 \cdot H_3$

$$H = H_1 \cdot H_2 \cdot H_3 \begin{pmatrix} \frac{1}{8} & \frac{1}{8} & \frac{1}{4} & 0 & \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{8} & \frac{1}{8} & \frac{1}{4} & 0 & -\frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{8} & \frac{1}{8} & -\frac{1}{4} & 0 & 0 & \frac{1}{2} & 0 & 0 \\ \frac{1}{8} & \frac{1}{8} & -\frac{1}{4} & 0 & 0 & -\frac{1}{2} & 0 & 0 \\ \frac{1}{8} & -\frac{1}{8} & 0 & \frac{1}{4} & 0 & 0 & \frac{1}{2} & 0 \\ \frac{1}{8} & -\frac{1}{8} & 0 & -\frac{1}{4} & 0 & 0 & -\frac{1}{2} & 0 \\ \frac{1}{8} & -\frac{1}{8} & 0 & -\frac{1}{4} & 0 & 0 & 0 & \frac{1}{2} \\ \frac{1}{8} & -\frac{1}{8} & 0 & -\frac{1}{4} & 0 & 0 & 0 & -\frac{1}{2} \end{pmatrix}$$

In order to apply that procedure to the columns of I as well; in order to complete the compression, we need to multiply by the transpose of H:

The resulting matrix is $H^T \cdot I \cdot H$:

$$\begin{pmatrix}
156.86 & -0.2031 & -0.1562 & -0.0625 & 0.75 & -0.1875 & -0.25 & -0.5 \\
0.6406 & -0.1718 & 0.2812 & -0.3125 & 0 & 0.6875 & 0.25 & 0.75 \\
\boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} & \boxed{0} \\
-0.2812 & -0.2812 & -0.4375 & 0.75 & 0 & -0.625 & 0.75 & \boxed{0} \\
-0.125 & \boxed{0} & 0.375 & -0.375 & -2.25 & 1 & 1 & 0.25 \\
-0.125 & \boxed{0} & 0.375 & -0.375 & -2.25 & 1 & 1 & 0.25 \\
0.06250 & 0.0625 & -0.375 & -0.25 & 0.75 & -0.5 & 1.75 & 1.75 \\
-1.625 & 0.375 & -1 & 0.25 & 0.75 & 0.75 & 0.25 & 0.75
\end{pmatrix}$$

As we can see, the two matrices are the same. This confirms our results. We can use matrix multiplication from now on, as it makes things faster and easier to process.

Important note:

H is invertible; thereby making this whole procedure reversible. We can remake the original picture. The fact that H is invertible makes this compression lossless. In this project, however, we are more concerned with lossy compression. One crucial parameter for us to keep an eye on is the compression ratio: the ratio of the non-zero elements in the original to the non-zero elements in the compressed image [8, p.5]. Let I be our 8×8 image matrix, and let $H^T \cdot I \cdot H$ be our Haar wavelet compressed image of I. $H^T \cdot I \cdot H$ has, as we have seen, several detail coefficients that are close to 0. We will pick our threshold value ϵ, as discussed earlier. Consequently, all the entries in $H^T \cdot I \cdot H$, whose absolute value is at most ϵ to 0, will be set to 0. The new matrix will have more 0 entries: it is the representation of an even more compressed image. Decompressing the latter image (using the inverse Haar wavelet transform matrix H^{-1}) will result in an image that is close to the original but not exactly it; we have lost some precision details. This is lossy compression.

7.2.1 More Linear Algebra for More Power

More power can be given to the Haar wavelet using a variant of the previous averaging and differencing process. Indeed, we can make the transform even stronger by using a common alternative: using $\sqrt{2}$ instead of 2. When this method is used for the wavelet transform, it is referred to as the **normalized wavelet transform**, which is a variant of the Haar wavelet transform.

By using $\sqrt{2}$, we change the nature of our Haar matrices. Indeed, now, they have columns of length 1, that are orthogonal to each other $\Rightarrow$ They are orthogonal matrices.

Definition:

Let A be an $n \times n$ orthogonal matrix. This means that $A \cdot A^T = I$. This basically implies that the inverse of the orthogonal matrix is nothing but its transpose.

Clearly, when it comes to an orthogonal matrix, its inverse is available for no effort. Basically, its inverse is gotten from exchanging indices since it's its transpose. Hence, they are easier to use in computations because the transpose operation is simpler than the inverse one [9]. This property helps increase the speed of performing our calculations.

Another useful property is the fact that orthogonal transformations preserve the dot product. That is:

$$A\mathbf{x} \cdot A\mathbf{y} = \mathbf{x} \cdot \mathbf{y}$$

They also preserve the length of vectors and the angles between them [10]. Let $\mathbf{x}$ and $\mathbf{y}$ be two vectors :

$$\|A\mathbf{x}\| = \|\mathbf{x}\|$$

and

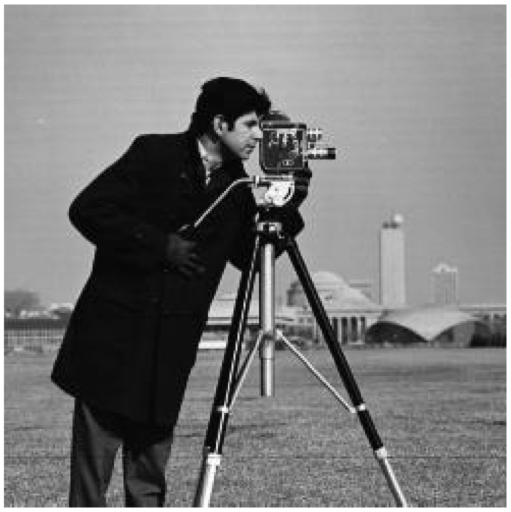
$$cos(\theta) = \frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\| \cdot \|\mathbf{y}\|}$$

We can conclude that the usage of orthogonal matrices will enable us to decrease the level of distortion in our final resulting image. Using the normalized Haar wavelet, which is orthogonal, will enable us to make compressed images that are more visually acceptable, and nicer to the human eye.

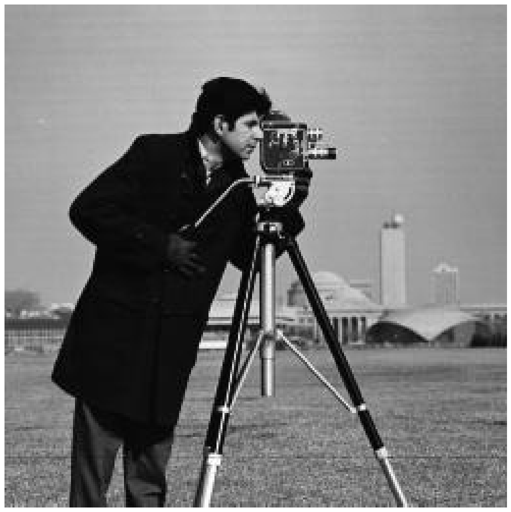
Moreover, multiplication by orthogonal matrices is known in computer algorithms to be extremely advantageous as it does not magnify error [11, p.4]. This is due to the fact that those matrices have a condition number of 1, which is the minimum. Indeed, they are used ubiquitously in the field of Numerical Analysis[12, p.1]. When building an algorithm on multiplications with orthogonal matrices, the errors that are in the original problem will not be amplified by that algorithm.

All in all, the normalized Haar wavelet offers advantages that result in better compression quality. These advantages are noticeable in the coming examples.

First of all, in Figure 2, we can see on the left, the original 'cameraman' image. On the right is its lossless version. On its right is a lossless version, which makes it essentially the same as the original. Since no detail coefficient has been thrown out, the lossless version is actually about 95% dense.



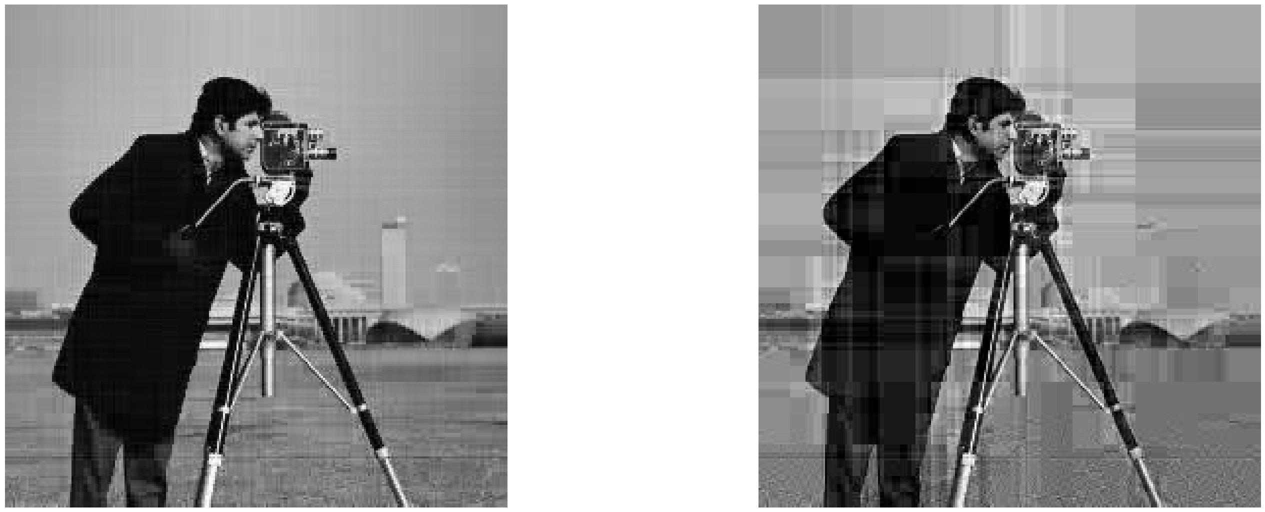
(a) Original image



(b) Lossless compression

Figure 2: The original cameraman image and its lossless version

With a compression ratio of 11.1 to 1, about 85.78% of the detail coefficients are removed. The compressed matrix contains only 5,907 entries out of the $256^2 = 65,536$. For that same ratio, we get the two images of Figure 3.



(a) Normalized wavelet **(b)** Standard wavelet

Figure 3: Both images compressed to a ratio of about 11.1 to 1

In order to have look at the sparse transform matrix, we can establish its bitmap, but making the nonzero entries be shown in black and the zero ones in white. When this is applied to the original image (which does not have any zero entry in the case of the cameraman.tif), we get a black square. However, for the transform matrix for the 11.1:1 compression ratio, we get the following which has a lot of white areas proving that it truly has a big number of 0s. The bitmap is shown in Figure 4:

Figure 4: The bitmap representation of the transform matrix

By now, the advantages of the normalized process can be seen quite clearly.

We can take it even further. With a compression ratio of 74.6 to 1. We can see from Figure 5 that the normalized wavelet maintains a visually acceptable output. With the standard Haar wavelet, we are barely able to recognize the image with that compression ratio. These images have 64,659 zero entries compared to the original. This means that they only have $256^2 - 64,659 = 65,536 - 64,659 = 877$ non-zero entries. It is incredible that we can still recognize the cameraman.

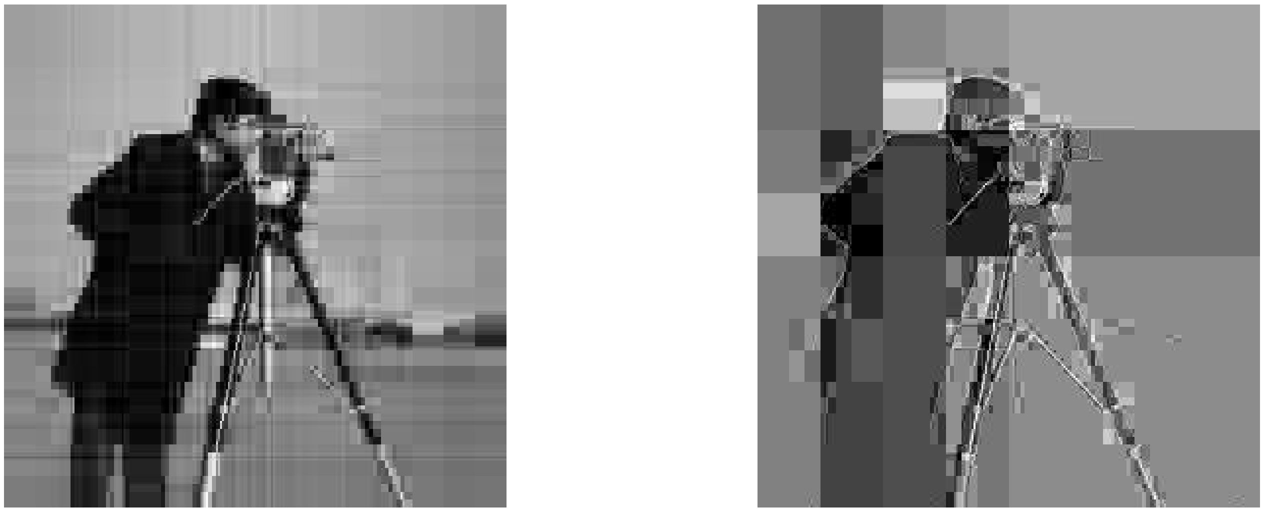

(a) Normalized wavelet **(b)** Standard wavelet

Figure 5: Both images compressed to a ratio of about 74.6 to 1

We won't use the compressed image gotten from the normalized wavelet with this high compression ratio because the image has gotten blurry, even though it is still way better than what we got with the standard Haar transform for the same ratio. However, we can use that image in thumbnail size. It looks almost as good as how any higher resolution image would look when resized to thumbnail proportions as we can see from Figure 6.

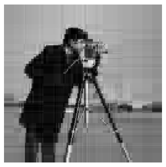
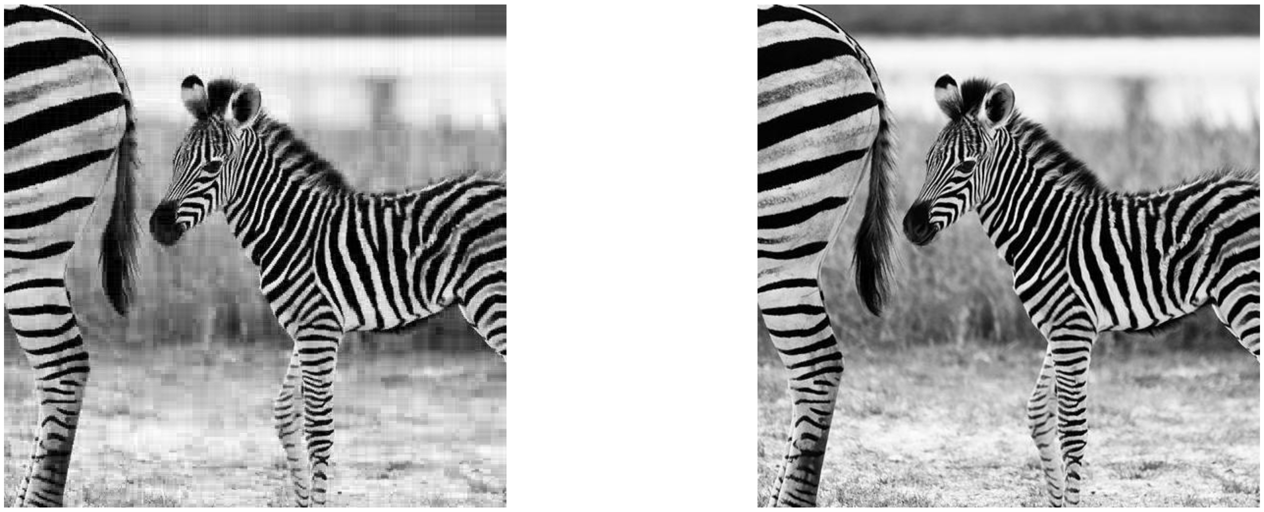


Figure 6: Thumbnail size version of Figure 4(a)

Actually, when using the normalized Haar wavelet, the resulting approximations of the original data are visually superior to those produced using the more intuitive regular method. With normalization, even more detail coefficients can be thrown out, achieving sparser data. Thus, this results in far greater compression while keeping a satisfactory output.

In Figure 7, we use a 512×512 square from a National Geographic image of a zebra and foal in Botswana. The superiority of normalization can be seen from the fact that we compressed the image using the normalized wavelet to a ratio of 11.3 to 1, and got a visually finer result than with the standard wavelet with a ratio of only 7.6 to 1.



(a) Normalized wavelet with compression ratio 11.3:1

(b) Standard wavelet with compression ratio 7.6:1

Figure 7: Better looking output with greater compression through normalization

At this point, we can clearly see that the normalized Haar wavelet is superior to the standard one. Hence, for the rest of our analysis, we will focus on this normalized variant.

At this point, we know that building the image compressor based on the normalized Haar wavelet transform is the best alternative as it is the one that yields the most visually-pleasing, and best-looking results. We have only seen grayscale images so far. To work with color ones, we process each matrix that represents one of the RGB colors, and then reconstruct the final image from those three transformed matrices. The code snippet that implements this is:

```
%Get the dimensions of the image (RGB) in an array
dimensions = size(image);
%Get the different RGB layers of the image
for x=1:dimensions(1)
        for y=1:dimensions(2)
                red(x,y) = image(x,y,1);
                green(x,y) = image(x,y,2);
                blue(x,y) = image(x,y,3);
        end
end
```

After that, we compress each one of the layers separately:

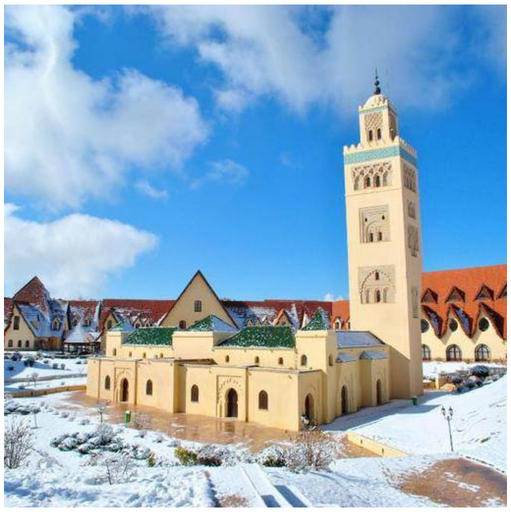
```
 %Now, matrix multiplication using
 %the generated Haar transformation matrix to compress
 compred = H*red*transpose(H);
 compgreen = H*green*transpose(H);
```

```
compblue = H*blue*transpose(H);
```

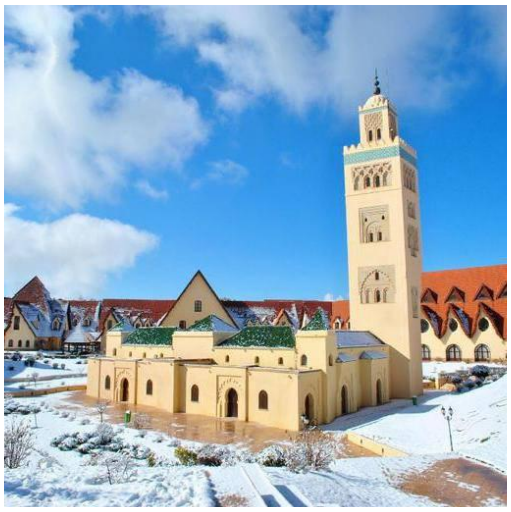
Finally, we can reconstruct the final result by reversing the process, and merging the RGB layers:

```
reconstruct_red = transpose(H)*compred*H;
reconstruct_green = transpose(H)*compgreen*H;
reconstruct_blue = transpose(H)*compblue*H;
%Let's merge them in "result" which is the final image's matrix
result(1:rowNum,1:colNum,1) = decompred(1:rowNum,1:colNum);
result(1:rowNum,1:colNum,2) = decompgreen(1:rowNum,1:colNum);
result(1:rowNum,1:colNum,3) = decompblue(1:rowNum,1:colNum);
```

Figure 8 shows an example of a color image and its lossless compression:



(a) Original image



(b) Lossless compression

Figure 8: The original image and its lossless version

<u>**Note:**</u> The original image in Figure 8 which represents Al Akhawayn University's mosque was originally 1011×680 and was resized to 512×512 (since the Haar transform only works with images that are $2^n \times 2^m$) using Ayman Bentourki's resizing using seam carving code.

Figure 9 shows that same image compressed to a ratio of 10.9:1.

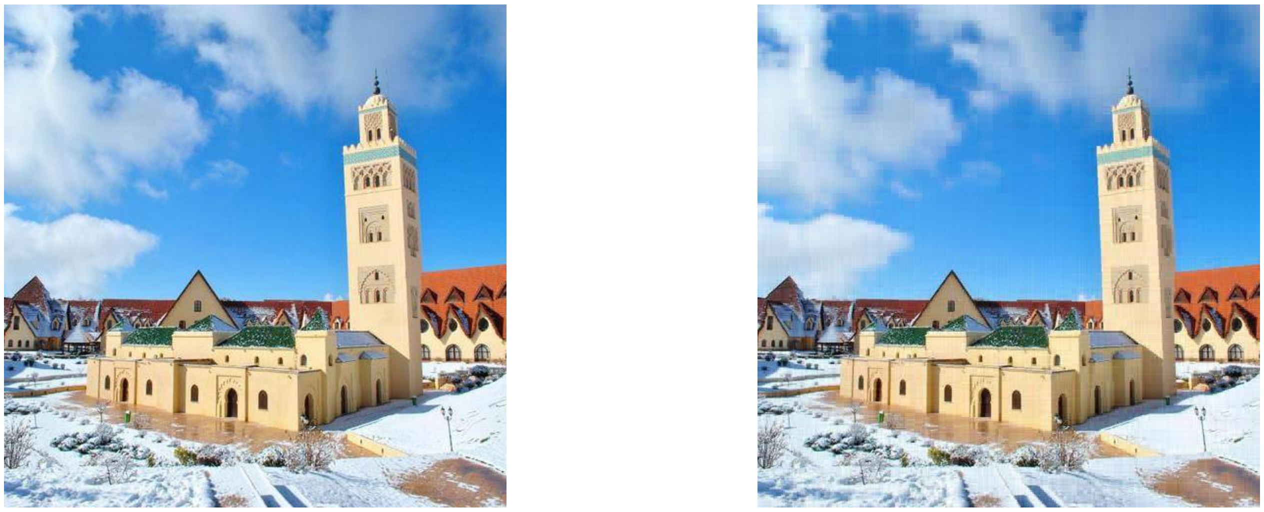

(a) Original image **(b)** Compressed image

Figure 9: The image was compressed to a ratio of 10.9:1

7.2.2 Algorithm Complexity

Throughout this project, various codes have been developed in order to compare between the performance of the different variants of the Haar wavelet transform in image compression, mainly:

1. The standard Haar wavelet using loops
2. The normalized Haar wavelet variant using loops
3. The normalized Haar wavelet variant using matrix multiplication

Since we have seen that the normalized wavelet is actually way better than the standard one, we will focus on that one. Two variants of normalization have been implemented. We will look at each one in detail to establish their complexities. Both of these implementations follow similar patterns. For instance, after reading the image, we need to check whether the rows and columns are powers of 2. This is important because the Haar wavelet transform only works with powers of 2. This is due to the fact that averaging and differencing are applied to two values by two. If the image is not of size $2^p \times 2^q$ with p, and $q \in \mathbb{N}$, an error message is displayed.

Normalized Haar Wavelet with Loops Consider that the image I is $m \times n$ (with m and n being powers of 2, and m and n are not necessarily different). The averaging and differencing process is encapsulated in the *OneStep* function. It is completely reversible, the function *OneStepReverse* is used for that purpose. In *OneStep*, we go through each row, and take the entries 2 by 2 to average and difference them. At the end of this process, we have approximation coefficients concentrated in the first column and the other entries are detail coefficients. Then, we do the same thing by going through the columns. We end up having the transform matrix which has only one approximation coefficient at the upper leftmost corner and all other entries are details. As we can see, this is $\mathcal{O}(N^2)$. At this point, we can do the thresholding to implement the make our compression lossy. This implies having to go through all the entries to throw out any detail coefficient whose absolute value is below our threshold: $\mathcal{O}(N^2)$. As for reconstructing the compressed image from the transform, we go through *OneStepReverse* which is based on the following:

$$\begin{aligned} average &= \tfrac{x+y}{\sqrt{2}} \\ difference &= \tfrac{x-y}{\sqrt{2}} \end{aligned} \tag{1}$$

$$\Leftrightarrow \begin{aligned} x &= \frac{average+difference}{\sqrt{2}} \\ y &= \frac{average-difference}{\sqrt{2}} \end{aligned} \tag{2}$$

By doing this to the whole matrix, we reach a complexity of $\mathcal{O}(N^2)$. All in all, we can draw the conclusion that this variant of the algorithm is quadratic: $\mathcal{O}(N^2)$.

Normalized Haar Wavelet with Matrix Multiplication The $m \times n$ image I is to be compressed. For that, we need to generate the square orthogonal Haar matrices that will enable us to do so. The function that does this is a straightforward implementation of the Haar functions $h_k(x)$ developed by the Hungarian mathematician Alfred Haar. These functions are defined for $x \in [0,1]$ and for $k = 0, 1, ..., N-1$, where $N = 2^n$:

$$h_0(x) = h_{00}(x) = \frac{1}{\sqrt{N}} \text{ for } x \in [0,1]$$

and

$$h_k(x) = h_{pq}(x) = \frac{1}{\sqrt{N}} \begin{cases} 2^{\frac{p}{2}}, & \frac{q-1}{2^p} \le x < \frac{q-\frac{1}{2}}{2^p} \\ -2^{\frac{p}{2}}, & \frac{q-\frac{1}{2}}{2^p} \le x < \frac{q}{2^p} \\ 0, & \text{otherwise for } x \in [0,1] \end{cases}$$

Any integer k can be expressed as the sum: $k = 2^p + q - 1$, where $0 \le p \le n-1, q = 0$ or 1 for $p = 0$, and $1 \le q \le 2^p$ for $p \ne 0$. For instance, $2 = 2^1 + 1 - 1$.

The Haar transform matrix H_N of order $N \times N$ can be constructed in the following way: an element i,j of this matrix is actually the basis Haar function $h_i(j)$, where $i = 0, 1, ..., N-1$ and $j = \frac{0}{N}, \frac{1}{N}, ..., \frac{(N-1)}{N}$ [2, p.109]. For instance:

$$H_4 = \begin{pmatrix} h_0(0/4) & h_0(1/4) & h_0(2/4) & h_0(3/4) \\ h_1(0/4) & h_1(1/4) & h_1(2/4) & h_1(3/4) \\ h_2(0/4) & h_2(1/4) & h_2(2/4) & h_2(3/4) \\ h_3(0/4) & h_3(1/4) & h_3(2/4) & h_3(3/4) \end{pmatrix} = \frac{1}{\sqrt{4}} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ \sqrt{2} & -\sqrt{2} & 0 & 0 \\ 0 & 0 & \sqrt{2} & -\sqrt{2} \end{pmatrix}$$

Therefore, we generate the two orthogonal transformation matrices using the implementation of the Haar functions: $m \times m$ H_c to process the columns and $n \times n$ H_r to process the rows. We get the transform as follows:

$$transform = H_c \cdot I \cdot \text{transpose}(H_r) \text{ (1)}$$

Note: $H_r = transpose(H_c)$ when the image I is a square matrix $(m = n)$.

In order to achieve lossy compression, we choose a threshold value ϵ and throw out any detail coefficient whose absolute value is at most ϵ. This has a complexity of $\mathcal{O}(N^2)$ since we have to go through each entry and check whether its absolute value is below the chosen threshold, and zero it out if it is the case. After that, we just need to reconstruct our resulting compressed image by reversing the process by using inverses which are nothing but the transposes of the $\underline{\text{orthogonal}}$ Haar matrices used earlier:

$$result = transpose(H) \cdot compression \cdot H \text{ (2)}$$

Finally, for testing purposes, we compute the compression ratio. This operation evaluates to a complexity of $\mathcal{O}(N^2)$ because we need to loop through all the entries to get the number of non-zero ones to be able to compute the ratio.

By now, it may seem that this algorithm is $\mathcal{O}(N^2)$. It is actually $\mathcal{O}(N^3)$ because of the matrix multiplication operations ((1) and (2)). It is known that matrix multiplication involves three fundamental operations:

Operation	Number of Times Performed
Multiplication	N^3
Addition	$(2N^3 + N^2 + N)$
Assignment	$(N^3 + 2N^2 + N + 1)$

Thus culminating in $\mathcal{O}(N^3)$. Consequently, this variant of the Haar process has a complexity of $\mathcal{O}(N^3)$.

Conclusion Even though Octave is optimized for matrix operations, multiplication presents $\mathcal{O}(N^3)$. A Haar transformation matrix has quite a few zero entries since it is sparse, so a lot of effort goes wasted when taking the time to perform multiplication, addition, and assignment that make up the matrix multiplication process. Therefore, in this case, the quadratic variant that implements loops in averaging and differencing happens to be the winner.

7.3 Reduction by 3

As we have seen earlier, the Haar wavelet transform can only process $2^n \times 2^m$ size image matrices. This is an obstacle as the majority of images do not fulfill that requirement. An attempt to overcoming some of this problem would be to create a code based on the same fundamental idea than the Haar transform: averaging and differencing. However, we will now reduce by 3. This will enable us to work with images that are $3^n \times 3^m$, and by extension $2^n \times 3^m$ and $3^n \times 2^m$ images as well by merging the two techniques.

When averaging and differencing, we follow this method:

Consider an array of size 3^2:

$$\begin{pmatrix} a & b & c & d & e & f & g & h & i \end{pmatrix}$$

The first iteration would yield:

$$\begin{pmatrix} \frac{a+b+c}{\sqrt{3}} & \frac{d+e+f}{\sqrt{3}} & \frac{g+h+i}{\sqrt{3}} & \frac{a-b}{\sqrt{3}} & \frac{c-b}{\sqrt{3}} & \frac{d-e}{\sqrt{3}} & \frac{f-e}{\sqrt{3}} & \frac{g-h}{\sqrt{3}} & \frac{i-h}{\sqrt{3}} \end{pmatrix}$$

Let:

$$A = \frac{a+b+c}{\sqrt{3}}$$

$$B = \frac{d+e+f}{\sqrt{3}}$$

$$C = \frac{g+h+i}{\sqrt{3}}$$

So, the next step gives:

$$\begin{pmatrix} \frac{A+B+C}{\sqrt{3}} & \frac{A-B}{\sqrt{3}} & \frac{C-B}{\sqrt{3}} & \frac{a-b}{\sqrt{3}} & \frac{c-b}{2} & \frac{d-e}{\sqrt{3}} & \frac{f-e}{\sqrt{3}} & \frac{g-h}{\sqrt{3}} & \frac{i-h}{\sqrt{3}} \end{pmatrix}$$

In order to reverse this process, let's consider three elements a, b, c. Their average is referred to as Avg, the first difference D_1, and the second one D_2 :

$$Avg = \frac{a+b+c}{\sqrt{3}}$$

$$D_1 = \frac{a-b}{\sqrt{3}}$$

$$D_2 = \frac{c-b}{\sqrt{3}}$$

Hence, we get that:

$$a = \frac{Avg+2D_1-D_2}{\sqrt{3}}$$

$$b = \frac{Avg-D_1-D_2}{\sqrt{3}}$$

$$c = \frac{Avg-D_1+2D_2}{\sqrt{3}}$$

Unfortunately, the implemented procedure does not involve orthogonal matrices but is working. Taking a 729×729 portion of an image from the National Geographic website called "Cold Light of Day" which was their photo of the day on the 1st of January 2016, we can show the performance of our reduction by 3 code.

First, we show the lossless compression:

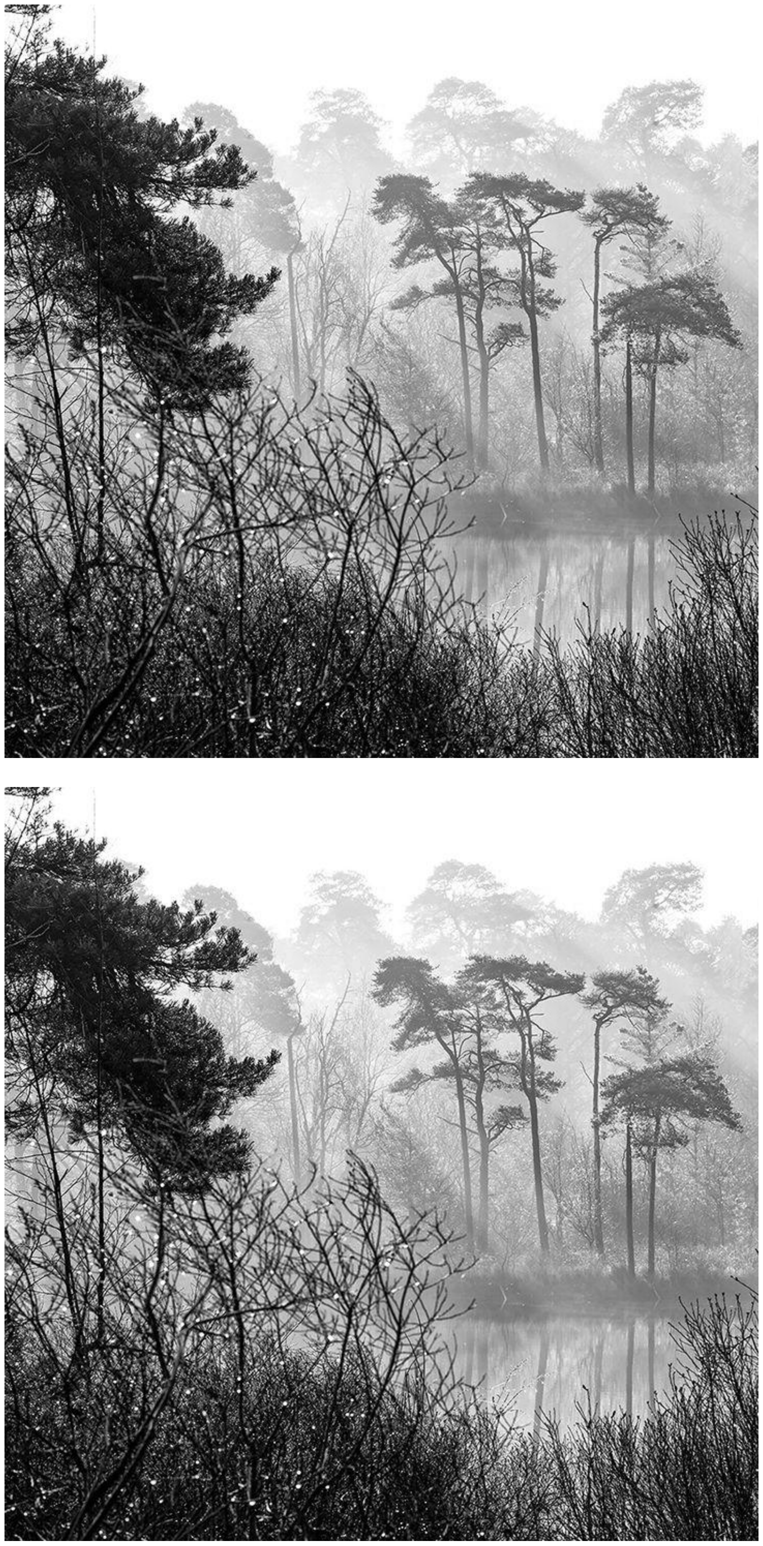


Figure 10: The original image (top) and its lossless version (bottom)

As for lossy compression, Figure 11 shows the image compressed to a ratio of approximately 7:1.

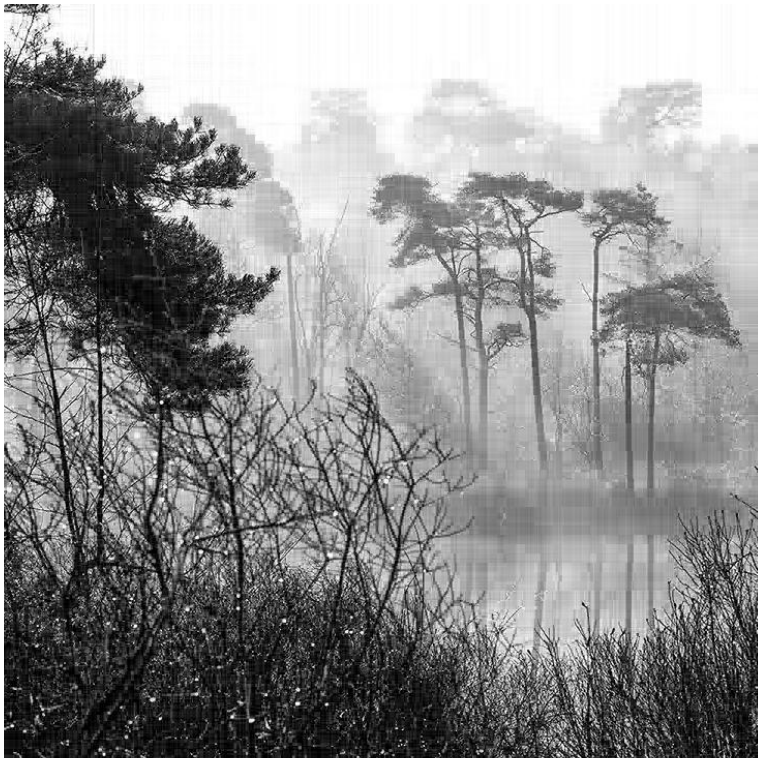


Figure 11: The image compressed to 7:1

As we have seen by now, orthogonality is needed to give better results with greater compression. Preliminary testing was not conclusive. This type of averaging and differencing does not yield an orthogonal transform which would explain the resulting distortion that are concurrent to a growing ratio. Further investigation and analysis is required to find a similar average/difference that would offer orthogonality.

8 The Discrete Cosine Transform

The discrete cosine transform (DCT) presents similarities to the discrete Fourier transform (DFT). The DCT was developed by N. Ahmed, T. Natarajan, and K. R. Rao and shared with the world through their landmark publication "Discrete Cosine Transform", IEEE Transactions on Computers, 9093, Jan 1974 [7, p.1]. This transform is quite well-known as it is efficient and used by popular data compression methods such as JPEG and MPEG. There is a variety of known fast algorithms for DCT calculation which makes the application and use of this technique even more attractive [1 p. 111-112]. In fact, the JPEG committee chose to use the DCT because of its good performance, and because there are methods to speed it up [2, p.146]. We are mainly concerned with the 2-dimensional DCT since we are with images and their pixels are correlated in two dimensions. This is given by:

$$G_{i,j} = \frac{1}{\sqrt{2n}} C_i C_j \sum_{x=0}^{n-1} \sum_{y=0}^{n-1} p_{xy} \cos(\frac{(2y+1)j\pi}{2n}) \cos(\frac{(2x+1)i\pi}{2n}) \tag{3}$$

for $0 \geq i,\ j \geq n-1$. As a result, the image gets broken up into blocks of size $n \times n$ pixels p_{xy}. Equation 3 is actually used to produce a block of $n \times n$ DCT coefficients $G_{i,j}$ for each block of pixels. In order to achieve lossy compression, the coefficients would need to be quantized.

In order to reconstruct a block of data values, we just need to compute the inverse discrete cosine transform (IDCT) [2, p.117]. For $n = 8$, for instance, we would use:

$$p_{xy} = \frac{1}{4} \sum_{i=0}^{7} \sum_{j=0}^{7} p_{xy} C_i C_j G_{i,j} \cos(\frac{(2x+1)i\pi}{16}) \cos(\frac{(2y+1)j\pi}{16}) \tag{4}$$

where $C_f = \begin{cases} \frac{1}{\sqrt{2}}, & f = 0 \\ 1, & f > 0 \end{cases}$

The consulted literature about the DCT states that time-wise, this transform is much faster than the wavelet-based image compression techniques. However, the DCT is worse error-wise. This raises the following question: why aren't discrete wavelet transforms more commonly used than they seem to be?

The literature does not give convincing reasons as to why the DCT is so widely used. In fact, studies of the Haar wavelet transform are constant and ongoing. Some sources state that the DCT has orthogonality and speed as undeniable advantages. Haar has an orthogonal variant as we have seen, and is not too slow in terms of performance neither.

9 Results and Conclusions

The normalized version of the Haar wavelet offers greater compression, and yields better looking results compared to the standard one. This is due to the properties of orthogonal matrices. The variant that implements loops to perform the normalization in the Haar wavelet transformation process is better in terms of algorithm complexity compared to the variant that generates the required Haar matrices and performs matrix multiplication.

Throughout this project, we focused on the Haar wavelet transform as a window to better understanding the different compression processes since they boil down to the same essence. Thus, this research and implementation have been useful in terms of gaining a lot of insight into the field of image compression and its application of mathematical concepts.

10 Future Work

In building an image compressor, attention must be given to storing the sparse matrices that hold the information about the image: approximation and detail coefficients so that we can rebuild the image, compressed this time, using those matrices. There are numerous formats for storing sparse matrices. By definition, they are arrays of data that contain a lot of zero entries. Then, when one needs to optimize the storage of this type of matrices, one can focus on keeping track of the non-zero entries and their locations. There are multiple methods for doing that: dictionary of keys, list of lists, coordinate list, compressed sparse row/column, etc. Analysis of their performances and implementation of the best one in an appropriate file type could be a boost to the compressor.

It would also be very interesting to look further into other wavelet-based image compression methods such as Daubechies' wavelets.

References

[1] Wavelet Analysis. Wolfram Documentation Center. 2016. Web. 1 Apr. 2016.
`https://reference.wolfram.com/language/guide/Wavelets.html`

[2] Salomon, David. *A Guide to Data Compression Methods*. Springer Professional Computing. New York: Springer-Verlag New York, 1 Feb. 2002. Print.

[3] Remigius, Onyshczak and Abdou Youssef. "Fingerprint Image Compression and the Wavelet Scalar Quantization Specification", a book chapter in *Fingerprint Imaging Technologies*. Springer-Verlag, 2004, pp. 385-413.
`https://www.seas.gwu.edu/ ayoussef/papers/FingerPrintWSQ-chapter.pdf`

[4] Machado, D. P., et al. Darth Fader: Using Wavelets to Obtain Accurate Redshifts of Spectra at Very Low Signal-to-Noise. Astronomy and Astrophysics 560. (16 Dec. 2013): 20. Web. 27 Feb. 2016.
`http://www.aanda.org/articles/aa/pdf/2013/12/aa19857-12.pdf`

[5] Tamboli, S. S., and V. R. Udupi. Image Compression Using Haar Wavelet Transform. International Journal of Advanced Research in Computer and Communication Engineering 2.8 (2013). Web. 20 Mar. 2016.
`http://www.ijarcce.com/upload/2013/august/43-h-shabanam-image%20compression%20using%20haar%20wavelet.pdf`

[6] Porwik, Piotr, and Agnieszka Lisowska. “The HaarWavelet Transform in Digital Image Processing: Its Status and Achievements.” Machine GRAPHICS and VISION 13. (2004): 7998. Web. 27 Feb. 2016.
`http://zsk.tech.us.edu.pl/publikacje/Art44!.pdf`

[7] Watson, Andrew B. “Image Compression Using the Discrete Cosine Transform.” Mathematica Journal 4.1 (1994): 8188. Web. 28 Feb. 2016.
`http://vision.arc.nasa.gov/publications/mathjournal94.pdf`

[8] Husen, Bill. “Haar Wavelet Image Compression”. 3 June 2010. Web. 7 Feb. 2016.
`https://people.math.osu.edu/husen.1/teaching/572/image_comp.pdf`

[9] Weisstein, Eric W. “Orthogonal Matrix”. Wolfram Research, 21 Apr. 2002. Web. 1 Apr. 2016.
`http://mathworld.wolfram.com/OrthogonalMatrix.html`

[10] Knill, Oliver. Orthogonal Matrices. Math.Harvard.edu. 2008. Web. 2 Apr. 2016.
`http://www.math.harvard.edu/archive/21b_spring_08/handouts/orthomatrix.pdf`

[11] Wright, Will. Linear Algebra and the Haar Wavelet Transform: A Glimpse into Data Compression. College of the Redwoods. 2011. Web. 20 Mar. 2016.
`http://msemac.redwoods.edu:16080/ wagner/-`
`math45projects/f11/Wright/WBW_Math45ProjectPaper.pdf`

[12] Alfeld, Peter. The Significance of Orthogonal Matrices. The University of Utah. 2013. Web. 20 Mar. 2016.

`http://www.math.utah.edu/ pa/6610/20130916.pdf`