Using Orthogonal Wavelet Expansions for Image Compression

Greyson Hall, Zane Perry, Indiana Kretzschmar

Abstract

The purpose of this paper is analyzing the usage of wavelet transforms and Singular Value Decomposition in signal processing, particularly its usage in image compression and processing. By utilizing knowledge of Haar wavelets, SVD, and MATLAB coding to decompose an example image to different levels and examining the efficiency and accuracy of storage at that level. Resulting in a large percentage of the compressed matrices equaling 0, especially when compared the SVD methodology, Wavelet compressions are shown to be a primary option because of how much storage it saves in overall decompression and re-compression.

1. Attribution

Greyson performed research on the experiment, analyzed the results, and wrote the abstract and conclusion; Indiana performed research on the background information, applications, and the overall experiment, analyzed results, and wrote the introduction; Zane wrote the code, analyzed the code, wrote up the mathematical formulation, and provided the examples and numerical results."

2. Introduction

For our project, we are proposing to investigate the use of wavelet transforms and Singular Value Decomposition in the application of signal processing, specifically in relation to image compression and processing. We will be applying many ideas and techniques from matrix methods, such as matrix multiplication, inverses, orthogonality, and more.

What is image compression? As stated by Robert Sheldon, "it is a process applied to a graphics file to minimize its size in bytes without degrading image quality below an acceptable threshold." For example, JPEG uses "lossy compression," which means that less information is preserved in the compressed file than in the original[2]. We can afford to ignore the highest-frequency components of our modified image

because human eyes are better at detecting lower-frequency components. The more consistent an image is, the more data we may discard without sacrificing the quality of the image. While it is still possible to compress more complex images, the effects of excessive compression are more obvious. One good example of how modern society is using image compression can be seen in the way the FBI stores fingerprints. The FBI has over 30 million fingerprints of criminals, and it is crucial for all of these to be compressed and stored as efficiently as possible[1]. They do this by the Joint Photographic Experts Group (JPEG), which used appy a Fourier based algorithm, but shifted to wavelet algorithms in recent years. In principle, wavelets are better for images, and Fourier is better for music. Images have sharp edges; music is sinusoidal. The jth Fourier coefficient of a step function is of the order $1/j$. The wavelet coefficients (mostly zero) are multiples of $2^{(-j/2)}$. The L^2 error drops exponentially, not polynomially[1], when N terms are kept. Wavelets are also a relatively new discovery. There are many examples of how wavelets can be used now, and there will most likely be more applications in the future.

One way image compression can be achieved is by using a specific wavelet called the Haar Wavelet. The Haar wavelet in mathematics is a collection of downscaled "square-shaped" functions that collectively make up a wavelet family or basis[1], in that it enables the representation of a target function over an interval in terms of an orthonormal basis. Wavelet analysis is analogous to Fourier analysis, but differ in some consequential ways that make is superior to Fourier analysis in the field of image analysis. The Haar sequence is frequently used as a teaching example and is widely acknowledged as the first wavelet basis. Moreover the wavelet transformation can be converted to a matrix multiplicatio,n as all linear transformations can be. What makes wavelet transforms particularly unique is the fact that they form an orthogonal basis as defined by the inner product space of $L^2[0,1]$ (the inner product of <f(x),g(x)> = f(x)g(x)dx) which provides it any discretely sampled function the ability to uniquely decompose into a linear combination of the wavelet basis. This is convenient for compression of an image as there exists only a single transform of an associated image, and for storage purposes, the basis itself does not need to be stored, only the coefficient matrix. To compress the actual matrix, an algorithm will begin by converting the chosen image to a grayscale format and then extracting a matrix representation of the image. It will then take pairs of rows and columns and execute something called “averaging and differencing.” This is where a pair of values is first averaged, and then the average is subtracted from the original. This will be replicated throughout all of the pairs in the rows and columns of the matrix. These averages and differences are then what creates the compression.

Another concept that comes up in this paper that one should be familiar with is Singular Value Decomposition (SVD). This is a factorization of a matrix into three matrices that takes the form of $A = P\Sigma Q^T$, where Σ is an nxn diagonal matrix of the singular values which are the square roots of the eigenvalues of the matrix. Q^T is the transpose of an nxn matrix containing the orthonormal eigenvectors of A^TA, and P is mxn matrix of the orthonormal eigenvectors of A^TA.

In the upcoming pages, we will use our knowledge of matrices, Haar wavelets,

SVD, and a little bit of coding to decompose an image. The following are some steps we will take to accomplish this goal: Obtain the analysis of an image into its component RGB matrices, apply the wavelet transform to them, use a threshold function to reduce the number of stored coefficients, and transform the image back into full form. We will also use SVD to decompose the image and compare the two techniques in multiple ways such as storage, efficiency, and more.

3. Mathematical Formulation

3.1. The Haar Wavelet Function

A wavelet function is used for the localization of small signals. One wavelet in particular that has gained popularity recently is the Haar Wavelet, illustrated below[1]:

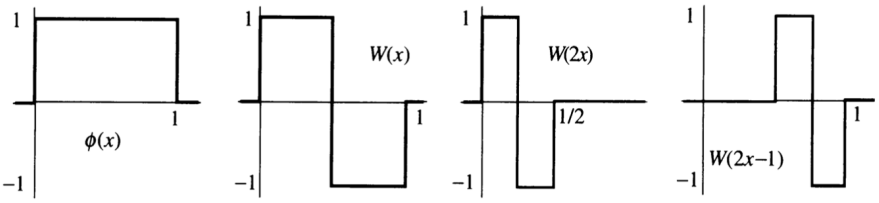

Figure 1. The scaling $\phi(x)$ function and the wavelet $W(x)$ function

Also shown in Figure 1 is the idea of *translation* and *dilation*. Translation is represented by $W(2x)$ where the wavelet function is compressed, and dilation is shown by $W(2x-1)$ where that translation is then shifted to the right. This translation and dilation can be extended to any size in order to approximate any function. One thing to note about these functions is that they are all orthogonal to each other under the inner product given by[3]:

$$< f(x), g(x) >= \int_0^1 f(x)g(x)dx$$

Another significant detail about the Haar Wavelet is that it forms a basis for $L^2[0,1]$, meaning that it is an orthogonal basis overall[2], which becomes important later. A common problem with using the Haar basis to approximate a function is that the discrete piecewise nature of the functions makes it very hard to approximate smooth curves without unnecessarily many dilations and translations, for which other transformations such as Fourier and Sine that use smooth trigonometric functions for approximations are well suited. However, in the field of image compression and manipulation, the Haar wavelet is exceptionally well suited due to the discrete nature of adjacent pixels[4]. That is, differences between intensities of pixels are not continuous, but jump between different values.
By sampling each function in a Haar family on the left side of every interval, a matrix representation of the basis can be created. For example, the functions shown in Figure 1 sampled at 0, 1/4, 1/2, and 3/4 generates the following matrix[5]:

$$\begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & -1 & 0 \\ 1 & -1 & 0 & 1 \\ 1 & -1 & 0 & -1 \end{bmatrix}$$

Note that it can be easily shown that the columns of the above matrix are both linearly independent and orthogonal. This matrix provides a simple way to multiply any sampled signal and determine the corresponding coefficients of the basis. By treating the values of pixel intensities of rows and columns in an image as a sampled signal, an image can then be decomposed into a Haar Wavelet Transform. But conceptually how does this relate to image compression?

3.2. Image Compression Through Averaging and Differencing

Every image can be represented in a matrix form as a value ranging from 0 to 255 for each pixel. When using a single matrix it represents a grayscale image where 0 represents black and 255 represents white. To extend this to a full-color image, three matrices can be used to represent the red, green, and blue intensity values stored in a pixel. Take for example the following 4×4 matrix to represent a small grayscale image (arbitrary numbers):

$$\begin{bmatrix} 24 & 36 & 47 & 57 \\ 44 & 44 & 153 & 175 \\ 16 & 22 & 149 & 145 \\ 4 & 2 & 196 & 200 \end{bmatrix}$$

A rather simplistic way to begin compressing this matrix would be to take the average of adjacent pairs. The problem with that is that there is no way to invert the transformation from pure averages. The solution to this is to also store the averaged difference between the two adjacent points as well[4]. While this would result in a matrix of the same size, you can see that a lot of the values are much smaller than they were previously. This is shown in the matrix below where the first two columns hold the averages and the last two hold the averaged differences:

$$\begin{bmatrix} 30 & 52 & -6 & -5 \\ 44 & 164 & 0 & -11 \\ 19 & 147 & -3 & -2 \\ 3 & 198 & 1 & -2 \end{bmatrix}$$

By repeating this process instead with entries that are vertically adjacent instead you can compress the image vertically:

$$\begin{bmatrix} 37 & 108 & -3 & -8 \\ 11 & 172.5 & -1 & -6.5 \\ -7 & -56 & -3 & 3 \\ 8 & -25.5 & -2 & 0 \end{bmatrix}$$

When looking at this matrix it is important to note that all of the averages are stored in the top left quadrant, which contains numbers that are much larger than the rest of the entries in general magnitude. This is significant because the most important information is stored in a 2×2 section of the matrix only a quarter size of the original. Therefore this strategy can be repeated to further compress the data. This and subsequent operations can be formulated as a matrix multiplication as well, as shown for an 8×8 matrix below[4]:

$$\begin{bmatrix} 1/2 & 0 & 0 & 0 & 1/2 & 0 & 0 & 0 \\ 1/2 & 0 & 0 & 0 & -1/2 & 0 & 0 & 0 \\ 0 & 1/2 & 0 & 0 & 0 & 1/2 & 0 & 0 \\ 0 & 1/2 & 0 & 0 & 0 & -1/2 & 0 & 0 \\ 0 & 0 & 1/2 & 0 & 0 & 0 & 1/2 & 0 \\ 0 & 0 & 1/2 & 0 & 0 & 0 & -1/2 & 0 \\ 0 & 0 & 0 & 1/2 & 0 & 0 & 0 & 1/2 \\ 0 & 0 & 0 & 1/2 & 0 & 0 & 0 & -1/2 \end{bmatrix}$$

$$\begin{bmatrix} 1/2 & 0 & 1/2 & 0 & 0 & 0 & 0 & 0 \\ 1/2 & 0 & -1/2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/2 & 0 & 1/2 & 0 & 0 & 0 & 0 \\ 0 & 1/2 & 0 & -1/2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1/2 & 1/2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1/2 & -1/2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

By multiplying these matrices on the right side of the original image matrix in succession, all of the row transformations can be converged into a single matrix operation (the same applies for column transformations but would be the transpose multiplied on the left):

$$\begin{bmatrix} 1/8 & 1/8 & 1/4 & 0 & 1/2 & 0 & 0 & 0 \\ 1/8 & 1/8 & 1/4 & 0 & -1/2 & 0 & 0 & 0 \\ 1/8 & 1/8 & -1/4 & 0 & 0 & 1/2 & 0 & 0 \\ 1/8 & 1/8 & -1/4 & 0 & 0 & -1/2 & 0 & 0 \\ 1/8 & -1/8 & 0 & 1/4 & 0 & 0 & 1/2 & 0 \\ 1/8 & -1/8 & 0 & 1/4 & 0 & 0 & -1/2 & 0 \\ 1/8 & -1/8 & 0 & -1/4 & 0 & 0 & 0 & 1/2 \\ 1/8 & -1/8 & 0 & -1/4 & 0 & 0 & 0 & -1/2 \end{bmatrix}$$

This resulting matrix is strikingly similar to the Haar Wavelet Basis in pattern. In fact, the columns of this matrix are a scaled basis for the wavelet function. Because we know that the Haar Wavelet forms an orthogonal basis, if we normalize each column, we create an orthogonal matrix. The orthogonality of the basis also explains how the transpose can be multiplied on the other side of the matrix to independently apply the transformation to the columns as well.

3.3. The Haar Wavelet Transform

In this way, by normalizing a Haar Wavelet Matrix, W, any matrix A that represents an image can be transformed into a new matrix where the entry at row 1 and column 1 represents an average of all pixels in the image, and all other entries are detail coefficients for the wavelet expansion[3]:

$$M = W^T A W$$

As demonstrated above, many of those detail coefficients are very small in magnitude and therefore can be regarded as insignificant. This also relates to the sparse nature of the Haar basis. A threshold can then be decided on in order to eliminate these small entries. The sparse nature of the resulting matrix M is significant because it is magnitudes more efficient for a computer to store sparse matrices by simply storing where the non-zero entries are rather than storing an entire matrix. Additionally, computations, especially multiplication, with sparse matrices are simplified by the presence of so many zeros.[1].

In reversing (or decompressing) the matrix, additional benefits of the Haar Wavelet Basis become clear. Most importantly, the orthogonal nature of the normalized matrix W means that inverting the transformation is just a matter of multiplying by the transpose:

$$A' = W M W^T$$

Where A' signifies a slightly altered version of the original image matrix due to the elimination of insignificant values. Another benefit to the Haar Wavelet is the standardization of the basis. Because the basis is not dependent on the image itself, the only thing that has to be stored is the compressed, sparse matrix M, a large percentage of which will just be 0.

4. Examples and Numerical Results

For the purposes of demonstrating these techniques for various forms of image compression, the following image was used:

This image was loaded into MATLAB and re-scaled so that its size was 1024 pixels by 1024 pixels. This is because using an square matrix with dimensions that are a power of 2 simplify the resulting manipulations. MATLAB then stores the image in a matrix where each entry is a value between 0 and 255 which represents the intensity of the color at that picture.

(a) (b)

Figure 2. A color and grayscale image of Wonder Woman used for reference

4.1. Averaging and Differencing

As described above, the basis of wavelet compression is derived from finding the averages of two adjacent points and storing them on one half of the matrix, and finding the averaged difference between them to store in the other half of the matrix. By applying this to the horizontal rows of the image and then applying the same algorithm to the vertical columns of the resulting matrix, the most prominent details of the image get compressed into a matrix one quarter the size of the original. This process can then be repeated multiple times to a desired level of compression:

(a) (b)

Figure 3. The averaging algorithm applied to the figure shown in 2(b) once (a) and twice (b)

What is notable about the result of these compressions is how much information is saved in the top left quadrant despite halving the dimensions of the original image. The other three quadrants of the matrix still store information about variation between pixels, though they are so unimportant that they can barely be seen without

any enhancement. For reference, the following image shows the same compressions scaled by a factor of 10 to display the information stored in the other quadrants:

Figure 4. The same compressions shown in Figure 3, but scaled by a factor of 10 to illustrate the variations stored in the bottom right quadrants

4.2. Converging to the Haar Wavelet

Repeatedly applying the averaging algorithm described above to the image until only a single averaged value is stored in the entry at row 1 and column 1, the rest being detail coefficients, converges onto a scaled version of the Discrete Haar Wavelet Transform (see above for an explanation). For that reason, the entire process can be simplified by multiplying by a matrix whose columns form a basis for the Haar Wavelet. By normalizing those columns, the matrix becomes orthogonal, which makes the inverse transform just its transpose.
The algorithm (shown in the appendix) to build a Haar Transform matrix of any size actually uses that strategy of recursively multiplying the Averaged-Differenced Matrix of different levels until the identity matrix is reached. The resulting matrix after the transform then stores a number of coefficients which are ultimately not crucial to the image itself, as demonstrated above in Figures 3 and 4. Therefore, a threshold can be applied to the absolute values of the Transformed Matrix in order to remove unnecessary information and resulting in a very sparse matrix that is much easier for a computer to store. This process can be applied separately to each value of an RGB image in order to compress full color images.
The figure below portays the re-expansion of the reference image (in color and grayscale) using different thresholds after compression from the Haar Wavelet:

While the resulting compression matrix is still the same size as the original matrix, the fact that the matrix is so sparse makes it much easier for a computer to store. The original reference image has a total of 1,048,576 pixels stored in its matrix, and 3,145,728 values for stored for the full RGB image. However, using a threshold of 10 (row a) on the compressed matrix, which yields an image almost indistinguishable from the original, has a total of 930,071 and 2,785,278 entries respectively that are equal to zero. That averages out to around 89% of the entries

Figure 5. Re-expanded images after applying a Haar Wavelet Transform and using a threshold of (a) 10 (b) 100 (c) 1000

in the matrix being zero. The compression with a threshold of 100 (row b) has a total of 1,040,711 and 3,122,445 entries respectively that are equal to zero, which is about 99.3% of all entries stored in the matrix, despite an impressive amount of similarity to the original image. While the compressions with a threshold of 1000 (row c) are not very clear, the general shape of the figure can still be made out, and what makes this notable is that the compressed matrices only have 301 and 149 entries respectively that are a non-zero value.

4.3. Comparison to SVD Conditioning

The methodology described above of using a threshold to eliminate unnecessary values is very similar to the technique used to discard ill-conditioned singular values of a Singular Value Decomposition[6]. This strategy can also be applied to image compression and serves as a useful comparison to the Wavelet Transform method. Though the exact threshold cannot be compared between the two strategies as they

represent different concepts, an arbitrary threshold can be chosen based on a rough comparison of the resulting compressed images:

(a) (b)

Figure 6. Re-expansion of an SVD Compression using a threshold of 1500

The figures above were chosen as a rough comparison of the threshold chosen in Figure 5(b). Firstly, it is notable that using an SVD factorization to store an image requires 3 matrices[6] instead of the single coefficient matrix used in Wavelet Compression. This means that just to store a grayscale image using the SVD method you need the same amount of matrices that would be used to store a full color image using Wavelet Compression. Secondly, the number of values equal to zero across all 3 matrices in the above matrix decomposition (after the threshold) is 1,048,526 and 3,145,579 for the gray and color imaged respectively. That is only about 33% of the total entries stored in the matrix. So despite using 3 times as many matrices to store roughly the same information, SVD compression has only 1/3 the amount of zero entries as Wavelet Compression, making it around a factor of 9 times worse for about the same quality of information.

5. Discussion and Conclusions

Wavelet transforms are not currently used often as a technique for image compression and decompression. Through the process detailed above, we discovered that it can be a useful technique and can be applied in a multitude of situations. In principle, wavelets are better for images, and while Fourier transforms may be the right choice for audio signals, Images have sharp edges; music is sinusoidal. As our understanding of how to utilize these wavelet transforms increases over time, it is likely that more situations will arise where they will prove useful and effective. Already, wavelets appear to have a distinct advantage over Fourier transforms when looking at fingerprints, as a study headed by the FBI entails[1]. To improve this first wavelet, we are led to dilation equations and their unusual solutions. Higher-order

wavelets are constructed, and it is surprisingly quick to compute with them — always indirectly and recursively. Fourier methods generally use real transforms (cosines), which, while effective, may eventually fall out of style due to high computational power and financial restraints. The largest advantage that Wavelet compressions have over their competitors is their sparse construction and standard construction that makes storage magnitudes more simple for computers. Wavelets will likely continue to take over, and will begin to be applied to more and more scenarios.

References

[1] Strang, Gilbert. "Wavelet Transforms versus Fourier Transforms." Bulletin of the American Mathematical Society, vol. 28, no. 2, Apr. 1993, pp. 288–305., https://doi.org/10.1090/s0273-0979-1993-00390-2.

[2] Shakiban, Chehrzad. "Wavelets." Applied Linear Algebra, 2nd ed., Springer, Cham, Switzerland, 2019.

[3] S. Ruikar and D. D. Doye, "Image denoising using wavelet transform," 2010 International Conference on Mechanical and Electrical Technology, 2010, pp. 509-515, doi: 10.1109/ICMET.2010.5598411.

[4] Mulcahy, Colm. "Image Compression Using the Haar Wavelet Transform." Spelman College, Spelman Science and Math Journal, 1996, pp. 22–31.

[5] Zhang, D. (2019). Wavelet Transform. In: Fundamentals of Image Data Mining. Texts in Computer Science. Springer, Cham. https://doi.org/10.1007/978-3-030-17989-2_3

[6] P, Gowri, et al. "Image Compression Using Singular Value Decomposition." International Journal of Mathematics Trends and Technology, vol. 67, no. 8, 2019, pp. 74–81., https://doi.org/10.14445/22315373/ijmtt-v65i8p507.

6. Appendix: MATLAB Code

Table of Contents

Initialization

```
clear;
RGB = imread('WonderWoman.jpg'); %Reads in an image
Photo = double(imresize(RGB, [1024, 1024])); %Standardizes the size of the
 image
Gray = double(rgb2gray(uint8(Photo))); %Returns a grayscale version of the
 image
```

Function Calls

```
% Variables used to store the results of image manipulations
TransformedGray = compressToLevel(Gray, 2, 1024);
[CompressedGray, numberOfGrayZeros] = waveletCompress(Gray, 1000);
[CompressedFull, numberOfRGBZeros] = waveletCompressFull(Photo, 1000);
[CompressedSVD, numberOfSVDZeros] = CompressSVD(Gray, 1500);
[CompressedSVDFull, numberOfSVDRGBZeros] = CompressSVDFull(Photo,1500);
```

Image Display

```
%Functions used to display the manipulated images
%{
imshow(uint8(uint8(Gray)));
imshow(uint8(TransformedGray));
imshow(uint8(CompressedGray));
imshow(uint8(CompressedFull));
imshow(uint8(CompressedSVD));
imshow(uint8(CompressedSVDFull));
%}
```

Average/Differencing Function

```
% Function that approximates a single level of a Haar wavelet transform
function Transform = HaarTransformApprox(M)
```

```
    sz = size(M);

    Intermediate = zeros(sz); %Stores the horizontal wavelet transform
    for i = 1:sz(1,1) %Loop that calculates the averages that make up wavelet
 coefficients
        for j = 1:2:sz(1,2)
            Intermediate(i, (j + 1) / 2) = (M(i,j) + M(i,j+1)) / 2;
        end
    end

    for i = 1:sz(1,1) %Loop that calculates the differences that make up
 detail coefficients
        for j = 1:2:sz(1,2)
            Intermediate(i, (sz(1,2) / 2) + ((j + 1) / 2)) = (M(i,j) - M(i,j
+1)) / 2;
        end
    end

    Transform = zeros(sz); %Stores the full 2D transform (horizontal then
 vertical)


    for i = 1:2:sz(1,1)%Loop that calculates the averages that make up wavelet
 coefficients
        for j = 1:sz(1,2)
            Transform((i + 1) / 2, j) = (Intermediate(i,j) + Intermediate(i
+1,j)) / 2;
        end
    end

    for i = 1:2:sz(1,1) %Loop that calculates the differences that make up
 detail coefficients
        for j = 1:sz(1,2)
            Transform((sz(1,1) /2) + ((i + 1) / 2), j) = (Intermediate(i,j) -
 Intermediate(i+1,j)) / 2;
        end
    end
end
```

Generating Haar Function

```
% Function that creates a Haar wavelet basis matrix based on the
% matrix multiplication described in the Mathematical Formulation Section
function Hr = generateHaar(N, origSize)

    if(N == 1)
        Hr = eye(origSize);
    else
        A = zeros(origSize);
        index = 1;
        for i = 1:(N / 2)
            A(index,i) = 1;
            A((index + 1),i) = 1;
```

```
            index = index + 2;
        end
        index = 1;
        for i = ((N / 2) + 1): N
            A(index, i) = 1;
            A((index + 1), i) = -1;
            index = index + 2;
        end
        for i = (N + 1):origSize
            A(i,i) = 1;
        end
        Hr = A * generateHaar((N / 2), origSize);
    end

end
```

Function to generate an Average/Difference Matrix

```
% Function that generates a matrix of the given level as described in the
% mathematical formulation section
function W = generateAveDiff(N, fullSize)

    W = zeros(fullSize);
        index = 1;
        for i = 1:(N / 2)
            W(index,i) = 1/2;
            W((index + 1),i) = 1/2;
            index = index + 2;
        end
        index = 1;
        for i = ((N / 2) + 1): N
            W(index, i) = 1/2;
            W((index + 1), i) = -1/2;
            index = index + 2;
        end
        for i = (N + 1):fullSize
            W(i,i) = 1;
        end

end
```

Compression Function

```
%Function that compresses an image to a specified level based on repeated
%Average/Difference matrices
function compressed = compressToLevel(A, N, fullSize)

    compressed = A;

    i = 1;
```

```
    while(i <= N)
        avgDiff = generateAveDiff((fullSize / i), fullSize);
        compressed = transpose(avgDiff) * compressed * avgDiff;
        i = i * 2;
    end

end
```

Wavelet Compression Functions

```
% Function that compresses an image using the discrete Haar wavelet
% transform matrix and deletes entries based on a specified threshold. Also
% returns the number of entries in the compressed matrix that have a value
% of zero
function [expanded, numberOfZeros] = waveletCompress(A, threshold)

    waveletTransform = normc(generateHaar(1024, 1024));
    compressedImage = transpose(waveletTransform) * A * waveletTransform;
    compressedImage(abs(compressedImage) < threshold) = 0;
    expanded = waveletTransform * compressedImage *
 transpose(waveletTransform);

    numberOfZeros = sum(sum(compressedImage == 0));


end

% Function that can take a full RGB matrix as an input and returns the
% wavelet compression of each component and recombines them. Also returns
% the number of entries across all 3 compressed matrices that have a value
% of zero
function [expanded, numberOfZeros] = waveletCompressFull(A, threshold)

    [Red, RedZeros] = waveletCompress(A(:,:,1), threshold);
    [Green, GreenZeros] = waveletCompress(A(:,:,2), threshold);
    [Blue, BlueZeros] = waveletCompress(A(:,:,3), threshold);

    expanded = cat(3,Red,Green,Blue);
    numberOfZeros = RedZeros + GreenZeros + BlueZeros;
end
```

SVD Compression Functions

```
%Function that computes the SVD decomposition of an image and deletes
%singular values based on a specified threshold. Also returns the number of
%elements across the full factorization that have a value of zero
function [expanded, numberOfZeros] = CompressSVD(A, threshold)

    [P,S,Q] = svd(A);
    S(S < threshold) = 0;
    expanded = P * S * transpose(Q);
```

```
    numberOfZeros = sum(sum(P == 0)) + sum(sum(S == 0)) + sum(sum(Q == 0));

end

%Function that computes the SVD decomposition of a full RGB image and
%recombines the components. Also returns the number of entries across all 3
%factorizations that have a value of zero
function [expanded, numberOfZeros] = CompressSVDFull(A, threshold)

    [Red, RedZeros] = CompressSVD(A(:,:,1), threshold);
    [Green, GreenZeros] = CompressSVD(A(:,:,2), threshold);
    [Blue, BlueZeros] = CompressSVD(A(:,:,3), threshold);

    expanded = cat(3, Red, Green, Blue);
    numberOfZeros = RedZeros + GreenZeros + BlueZeros;

end
```

Published with MATLAB® R2022a