Knox College Mathematics Department

Image Compression using the Haar-Wavelet Transform

Robert Schick and Phongkiet Sisaikeo
3/17/2013

Abstract

Discovered by Alfred Haar in the early 20th century, wavelets are a relatively recent development in the fields of mathematics and computer science, but they have applications in a wide range of fields, most notably signal processing. The Haar wavelet transform (HWT) can be used for both lossless and lossy image compression. This paper briefly introduces the subject of image compression, explains the basic idea behind the HWT, and also investigates how one can apply linear algebra to make this process simpler, faster, and more efficient.

Introduction

Compression is a term that many are probably familiar with on at least some level: the idea of representing something in "broad strokes" or making a "long story short" is a fairly accessible concept. This idea forms the foundation of image compression. When retrieving a digital file from the internet, it usually takes a considerably amount of time, especially if your internet speed was quite slow. To get a sense of how much of a problem this can be, let's first define how images are represented digitally.

A pixel (abbreviation of "picture element") is the unit building block for a digital picture. In the grayscale color space, it is simply a scalar value between 0 and 1, with 0 corresponding to black and 1 corresponding to white. When talking about color images, each pixel is a 3-tuple with each color channel (red, green, blue) represented in the same way as a grayscale image. So it follows that an image is simply a two-dimensional array of pixels: a matrix.

Consider an image with dimension 1024x1024 pixels will be stored in a 1024 by 1024 matrix with entries of whole numbers representing its color intensity. This image of 1024x1024x32 bits of space will require about 4 MB of space, which will take approximately 10 minutes to transfer (given an internet speed of 56,000 bits per second). Since digital pictures need a large storage space and require some amount of time to transfer, image compression will be a nice tool that can save us a lot of time.

Let's outline some goals for what our "compressed" image should look like. First of all, we need to decide whether we want to be able to reconstruct the original image perfectly or if we can afford to sacrifice some detail for the sake of saving space. The first process, where no detail is lost, is called "lossless" compression, while the second is called "lossy." We also (obviously) want the compressed image to take up less space than the original image.

In practice, there are many algorithms that compress images. However, we will focus on the HWT and slight modifications of this transform throughout the remainder of this paper.

Vector transformation using the HWT

Averaging and differencing

The main idea behind the HWT is an iterative process of "averaging and differencing." Consecutive pixels are paired up, averaged, and the value of the omitted pixel is subtracted and stored.

To make this process easier to understand, consider the vector u defined as such:

$$u1 = [420 \quad 680 \quad 448 \quad 708 \quad 1260 \quad 1420 \quad 1600 \quad 1600]$$

This vector, which is essentially a 1x8 matrix with 8 entries, will take three steps to complete the averaging and differencing process. In general, if a data string has length equal to *k*, $\log_2 k$ determines how many iterations are required to fully compress the data.

Let's treat each value in the vector as a pixel, with value of 0 corresponding to black and 1600 corresponding to white. Figure 1 is the image representation of $u1$:

Figure 1

Step 1

First, group the entries into a set of four pairs. Then, take the average of each pair. We place the four average values in the first four positions of the new vector. These values are called "**approximation coefficients**." Next, we compute the differences between the first element of each pair and the average obtained in the first step. These values will be placed in the last four entries of the new vector. These values are called "**detail coefficients.**"

Back to the example:

- We have 4 pairs:

  {(420,680), (448,708), (1260, 1420), (1600,1600)}

- Take the average of each pair:

  $\left(\frac{\mathbf{420}+680}{2}, \frac{\mathbf{448}+708}{2}, \frac{\mathbf{1260}+1420}{2}, \frac{\mathbf{1600}+1600}{2}\right)$ = (550, 578, 1340, 1600)

- Compute the difference between the first element in each pair and the average:

$$(420 - 550{,}448 - 578{,}1260 - 1340{,}1600 - 1600) = (-130, -130, -80{,}0)$$

- The final vector $u2$ is formed by appending the detail coefficients to the approximation coefficients:

$$u2 = [\underbrace{550 \quad 578 \quad 1340 \quad 1600}_{approximation} \quad \underbrace{-130 \quad -130 \quad -80 \quad 0}_{detail}]$$

Step 2

Repeat Step 1 to the approximation coefficients of $u2$ to form the second vector:

$$u3 = [564 \quad 1470 \quad -14 \quad -130 \quad -130 \quad -130 \quad -80 \quad 0]$$

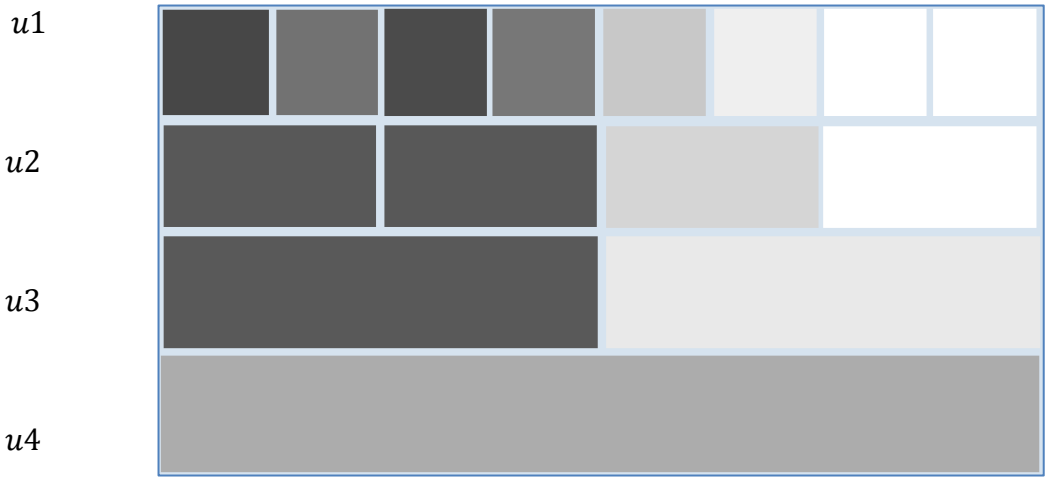
Step 3

And again, we apply the same method to the approximation coefficients of $u3$. Then, we will have:

$$u4 = [1017 \quad -453 \quad -14 \quad -130 \quad -130 \quad -130 \quad -80 \quad 0]$$

Figure 2 shows the consecutive iterations of the HWT applied to u1. Only the approximation coefficients are shown; showing the detail coefficients isn't particularly enlightening in this example. The first row is $u1$. The fourth row is the approximation coefficient of $u4$, which is simply the average of all the values of $u1$, our original vector.

Figure 2



Sparse Matrices

At first glance, it might be difficult to see how this process has compressed the original vector; after all, $u4$ has the same number of entries as $u1$. However, there is an important observation that can be made: since the last two entries of $u1$ were the same, the difference between the omitted entry and the average is 0. In general, areas in pictures that have little to no variance between consecutive pixels will, after the HWT is applied, produce matrices that have areas that are mostly either 0's or numbers that are very close to 0.

Matrices that have mostly 0 entries are called "sparse," and are much easier to store and transmit than their "dense" counterparts. Consider the identity matrix, a classic example of a sparse matrix. When describing it, you can simply say "place 1s along the main diagonal and zeros everywhere else." Although this is a rather crude example, this is somewhat analogous to the way that sparse matrices are actually represented. One format that is commonly used is called the Coordinate List system, which stores each non-zero value as a 3-tuple which holds the row, column, and value. Since you are replacing each value with a set of three-values, the matrix needs to have at least three times the amount of zero entries as non-zero entries to realize actual space savings.

So, after we apply the HWT to an image, unless every pixel differs significantly from its neighbor, we will end up with a somewhat sparse matrix. To make this matrix even more sparse, we can choose some non-negative threshold number ϵ, and set all numbers that have a magnitude less than ϵ to zero. Depending on how we pick our ϵ, we can create varying levels of "sparseness" and varying qualities of approximation to our original image.

Matrix representation of the HWT

For the sake of simplicity, we will work with the same example we used to demonstrate the concept of averaging and differencing process:

Step 1

As in the example above, we have to take the average for each pairs and take the differences. We can easily do the same thing by multiplying the following matrix (i.e., W_1) on the right hand side of a vector,

$$\overrightarrow{u_1} \cdot W_1 = (420\ 680\ 448\ 708\ 1260\ 1420\ 1600\ 1600) \cdot \begin{pmatrix} 1/2 & 0 & 0 & 0 & 1/2 & 0 & 0 & 0 \\ 1/2 & 0 & 0 & 0 & -1/2 & 0 & 0 & 0 \\ 0 & 1/2 & 0 & 0 & 0 & 1/2 & 0 & 0 \\ 0 & 1/2 & 0 & 0 & 0 & -1/2 & 0 & 0 \\ 0 & 0 & 1/2 & 0 & 0 & 0 & 1/2 & 0 \\ 0 & 0 & 1/2 & 0 & 0 & 0 & -1/2 & 0 \\ 0 & 0 & 0 & 1/2 & 0 & 0 & 0 & 1/2 \\ 0 & 0 & 0 & 1/2 & 0 & 0 & 0 & -1/2 \end{pmatrix}$$

```
{{550, 578, 1340, 1600, -130, -130, -80, 0}}
```

Step 2

Similarly, the first two pairs of entries of $\overrightarrow{u_2}$ can be averaged and differenced easily by multiplying the following matrix (i.e., W_2) to the new vector obtained from the first step.

$$\overrightarrow{u_2} \cdot W_2 = (550\ 578\ 1340\ 1600\ -130\ -130\ -80\ 0) \cdot \begin{pmatrix} 1/2 & 0 & -1/2 & 0 & 0 & 0 & 0 & 0 \\ 1/2 & 0 & -1/2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/2 & 0 & 1/2 & 0 & 0 & 0 & 0 \\ 0 & 1/2 & 0 & -1/2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

```
{{564, 1470, -564, -130, -130, -130, -80, 0}}
```

Step 3

The final product can again be obtained simply by multiplying this following matrix (i.e., W_3) to the right hand side of the vector obtained from the second step.

$$\overrightarrow{u_3} \cdot W_3 = (564\ 1470\ -564\ -130\ -130\ -130\ -80\ 0) \cdot \begin{pmatrix} 1/2 & 1/2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1/2 & -1/2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

```
{{1017, -453, -564, -130, -130, -130, -80, 0}}
```

To make this process even simpler, we can multiply these three matrices (W_1, W_2, W_3) to obtain a single transform matrix. Then, these three steps of averaging and differencing can be done in just one step:

$$\overrightarrow{u_1} \cdot (W_1 \cdot W_2 \cdot W_3) = (420\ 680\ 448\ 708\ 1260\ 1420\ 1600\ 1600) \cdot \begin{pmatrix} \frac{1}{8} & \frac{1}{8} & -\frac{1}{4} & 0 & \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{8} & \frac{1}{8} & -\frac{1}{4} & 0 & -\frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{8} & \frac{1}{8} & -\frac{1}{4} & 0 & 0 & \frac{1}{2} & 0 & 0 \\ \frac{1}{8} & \frac{1}{8} & -\frac{1}{4} & 0 & 0 & -\frac{1}{2} & 0 & 0 \\ \frac{1}{8} & -\frac{1}{8} & 0 & \frac{1}{4} & 0 & 0 & \frac{1}{2} & 0 \\ \frac{1}{8} & -\frac{1}{8} & 0 & \frac{1}{4} & 0 & 0 & -\frac{1}{2} & 0 \\ \frac{1}{8} & -\frac{1}{8} & 0 & -\frac{1}{4} & 0 & 0 & 0 & \frac{1}{2} \\ \frac{1}{8} & -\frac{1}{8} & 0 & -\frac{1}{4} & 0 & 0 & 0 & -\frac{1}{2} \end{pmatrix}$$

```
{{1017, -453, -564, -130, -130, -130, -80, 0}}
```

Note: The columns of matrices W_1, W_2, and W_3 each form an orthogonal subset for $\mathbb{R}^8$; thus, they also form a basis for $\mathbb{R}^8$. Since they are non-singular matrices, they are invertible. In addition, the product of invertible matrices ($W = W_1.W_2.W_3$) is also invertible. As a consequence, we can retrieve our original vector, $\overrightarrow{u_1}$, as follows,

$$\overrightarrow{u_1} = W^{-1}.\overrightarrow{u_4}$$

Examples

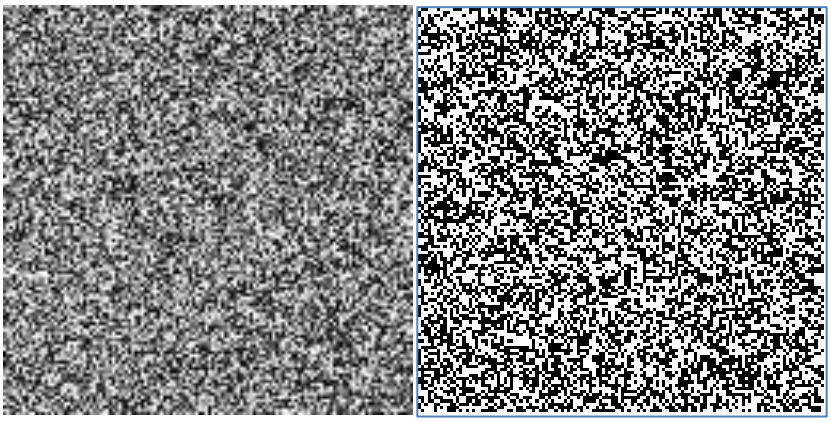
To test how well this algorithm works, we need to establish a metric to measure how much space is saved by compression. This can be easily defined: just take the ratio of non-zero entries in the original image and divide by the number of non-zero entries in the compressed image.

$$Compression\ Ratio = \frac{Non - Zero\ Entries_{original}}{Non - Zero\ Entries_{compressed}}$$

Earlier, we discussed the Coordinate List system of representing sparse matrices, which, if applied to the first example, actually takes up more space than the original vector. In that case, the compression ratio doesn't actually help us determine the amount of space saved, but as the dimension of the image grow larger, this ratio approaches the amount of space saved by compression.

Example A: Noise

Figure 3: Randomly generated Images



The pictures above show two randomly generated 128x128 images, the first being random real numbers between 0 and 1, the second being randomly generated 0s and 1s. Although they may appear to be virtually identical, the image on the left can be compressed much more than the image on the right. The reason behind this is that the image on the left has much less "detail" than the image on the right, which allows the compression algorithm to find areas that can be represented with a more sparse matrix.

Example B: Lena

Realistically, neither of these images are particularly good images to test this algorithm with. They do not adequately represent the "average case" of an image you would be likely to encounter in the real world. There is actually a standard image that has been used for image-processing since the 1970's, called "Lena."

Figure 4: Lena

This image has been used since June 1973 to test various image-manipulation algorithms, but its history is more seedy than you might expect. Jamie Hutchison recounts the story behind the Lena image:

"Alexander Sawchuk estimates that it was in June or July of 1973 when he, then an assistant professor of electrical engineering at the USC Signal and Image Processing Institute (SIPI), along with a graduate student and the SIPI lab manager, was hurriedly searching the lab for a good image to scan for a colleague's conference paper. They had tired of their stock of usual test images, dull stuff dating back to television standards work in the early 1960s. They wanted something glossy to ensure good output dynamic range, and they wanted a human face. Just then, somebody happened to walk in with a recent issue of *Playboy*."

Obviously, this is the cropped version. You can see why it is a good test image: there are areas of high and low detail, and it is much more representative of an "average" image that you might expect to be compressing.

Let's take a look at how the HWT compresses the image with different threshold values and compression ratios. For these tests, we converted the original image to a 128x128 grayscale version.

Figure 5: Lena Compressed

This series of pictures from left to right, is the original, uncompressed version, followed by the compression with a threshold of .01, .02, and .05 respectively. Although the first two images may appear to be nearly identical, there are certain "artifacts" that appear in areas of low detail: if you look closely at her shoulder and areas in the background, you can see large rectangles where the algorithm has made approximations. The compression ratios for these images, from left to right, are 1:1 (uncompressed), ~2.05:1, ~2.93:1, ~5.76:1. This means that the second image, although it looks almost identical to the first, requires ½ the space to store and transmit than the original.

Improvements: Normalizing the HWT

Figure 6: Non-normalized vs. Normalized

To improve the results of the transformation, we must go back to the matrix representation of the HWT. Although the columns of this transformation are orthogonal, they are not *orthonormal*. This means that the transformed vector will not have the same magnitude as the original vector. This will manifest itself in the form of distortions and artifacts, as shown in Figure 6. Both images have approximately the same compression ratio of 2.93:1, but the right image is significantly closer to the original image than the left.

References

[1] Application to image compression. (n.d.). University of Ottawa Press. Retrieved February, 27, 2013, from http://aix1.uottawa.ca/~jkhoury/haar.htm

[2] Morton, P., & Peterson, A. (1997, December 19). Image Compression Using the Haar Wavelet Transform. College of the Redwoods Press. Retrieved March, 2, 2013, from http://online.redwoods.edu/instruct/darnold/laproj/Fall97/PMorton/pmorton.pdf

[3] Haar Wavelet Compression. (n.d.). Ohio State University Press.Retrieved Web. 2 Mar. 2013. from http://www.math.osu.edu/~husen.1/teaching/572/image_comp.pdf

[4] Image Compression. [Powerpoint Slides] Retrieved February, 27, 2013, from cpsc.ualr.edu/milanova/image.../Image%20compression_part1.doc

[5] Energy (signal Processing). (2013, February 28) Wikimedia Foundation. Retrieved March, 17, 2013, from http://en.wikipedia.org/wiki/Energy_(signal_processing)

[6] Sparse Matrix. (2013, February 03) Wikimedia Foundation. Retrieved March, 17, 2013, from http://en.wikipedia.org/wiki/Sparse_matrix

[7] Haar Wavelet. (2013, February 26) Wikimedia Foundation Retrieved March, 17, 2013, from http://en.wikipedia.org/wiki/Haar_wavelet