Normalized Wavelet Transform For Image Compression Using Orthogonality

Shubham Sahu(MIT2020051)
Indian Institute Of Information Technology Allahabad

***Abstract*— Digital images require large amounts of memory to store and, when retrieved from the internet, can take a considerable amount of time to download. The Haar wavelet transform provides a method of compressing image data so that it takes up less memory.Another way to make these transformation matrices more powerful is to use normalized wavelet transform. Having an orthogonal transformation matrix has two main benefits. First, because the inverse of an orthogonal matrix is equal to its transpose, this greatly increases the speed at which we can perform the calculations that reconstitute our image matrix.Second, as normalized transformations tend be closer to the original image.This is due to some powerful properties of orthogonal matrices.**

INDEX TERMS- Orthogonality, Haar Wavelet Transform, Image Compression, Normalised wavelet Transform.

I. INTRODUCTION

In recent years there has been an astronomical increase in the usage of computers for a variety of tasks. With the advent of digital cameras, one of the most common uses has been the storage, manipulation, and transfer of digital images. The files that comprise these images, however, can be quite large and can quickly take up precious memory space on the computer's hard drive. A gray scale image that is 256 x 256 pixels has 65, 536 elements to store, and a a typical 640 x 480 color image has nearly a million! The size of these files can also make downloading from the internet a lengthy process. One of the important factors for image storage or transmission over any communication media is the image compression.

Compression makes it possible for creating file sizes of manageable, storable and transmittable dimensions. A 4 MB image will take more than a minute to download using a 64kbps channel, whereas, if the image is compressed with a ratio of 10:1, it will have a size of 400KB and will take about 6 seconds to download.

Image Compression techniques fall under 2 categories, namely, Lossless and Lossy.

In Lossless techniques the image can be reconstructed after compression, without any loss of data in the entire process.

Lossy techniques, on the other hand, are irreversible, because, they involve performing quantization, which results in loss of data. Some of the commonly used techniques are Transform coding, (Discrete Cosine Transform, Wavelet Transform, Gabor Transform), Vector Quantization, Segmentation and Approximation methods, Spline approximation methods (Bilinear Interpolation/Regularisation), Fractal coding etc..

II. RELATED WORK ABOUT TOPIC

Before we go into details of the method, we present some background topics of image compression which include the principles of image compression, the classification of compression methods and the framework of a general image coder and wavelets for image compression.

A. How Images are Stored

Before we can understand how to manipulate an image, it is important to understand exactly how the computer stores the image. For a 256 x 256 pixel gray scale image, the image is stored as a 256 x 256 matrix, with each element of the matrix being a number ranging from zero (for black) to some positive whole number (for white). We can use this matrix, and some linear algebra to maximize compression while maintaining a suitable level of detail.

B. How the Wavelet Transform Works

In order to transform a matrix representing an image using the Haar wavelet transform, we will first discuss the method of transforming vectors called averaging and differencing. First, we start out with an arbitrary vector representing one row of an 8 x 8 image matrix:

y = (448 768 704 640 1280 1408 1600 1600)

Because the data string has length

$$8 = (2^3)$$

there will be three steps to the transform process. If the string were

$$(2^k)$$

long, there would be k steps in the process. For the first step our data string becomes

y1 = (608 672 1344 1600 160 32 64 0)

We get this be first thinking of our original data as being four pairs of numbers (448 768, 704 640, etc...). We average each of these pairs, and the results become t

first four entries of our modified string y1. These numbers are known as approximation coefficients. Next we subtract these averages from the first member of each pair. These answers become the last four entries of y1 and are known as detail coefficients, which are shown in bold. The detail coefficients are repeated in each subsequent transformation of this data string. The visual interpretation of approximation and detail coefficients will be discussed in Section 5. For now we will proceed with the second step which changes

our data string to: y2 =(640 1472 32 128 160 32 64 0)

We get this by treating approximation coefficients from the first step as pairs (608 672, 1344 1600). Once again we average the pairs and the results become the first two entries of y2, and are our new approximation coefficients. We then subtract the averages from the first element of each pair. The results become the third and fourth elements of y2(These are also detail coefficients and are repeated in the next step). The last four elements in y2 are identical to y1. For the last step our data string becomes: y3 =(1056 416 32 128 160 32 64 0) This time we obtain the first entry of y3 by averaging the two approximation coefficients of y2 (640 1472). We obtain the second element by subtracting the average from the first element of the pair. This is the final detail coefficient and is followed by the detail coefficients from y2.

C. Haar Wavelet Transformation Matrix

The Haar wavelet does this transformation to each row of the image matrix, and then again to every column in the matrix. The resulting matrix is known as the Haar wavelet transform of the original matrix. It is important to note at this point that this process is entirely reversible. It is this fact that makes it possible to retrieve the original image from the Haar wavelet transform of the matrix

The averaging and differencing method that we just discussed is very effective, but, as you can imagine, the calculations can quickly become quite tedious. We can, however, use matrix multiplication to do the grunt work for us. In our previous example, we can describe the transformation of y to y1 as: y1 = y A1

III. METHODOLOGY

Image compression using the Haar wavelet transform can be summed up in a few simple steps.

1. Convert the image into a matrix format(I)

2. Calculate the row-and-column transformed matrix (T) using Equation 1. The transformed matrix should be relatively sparse.

$$T = ((I\,W)^T\,W)^T = W^T\,I\,W \qquad (1)$$

3. Select a threshold value , and replace any entry in T whose magnitude is less than with a zero. This will result in a sparse matrix denoted as S.

$$I = ((T^T\,W^{-1})^T\,W^{-1} = (W^{-1})^T\,T\,W^{-1} \qquad (2)$$

4. To get our reconstructed matrix R from matrix S, we use an equation similar to equation 2, but, because the inverse of an orthogonal matrix is equal to its transpose, we modify the equation as follows:

R = W S W1 (3)

If you choose equal to zero, then S = T and therefore R = I. This is what we referred to previously as lossless compression. Because is equal to zero, no element of T is changed, and therefore no data is lost.

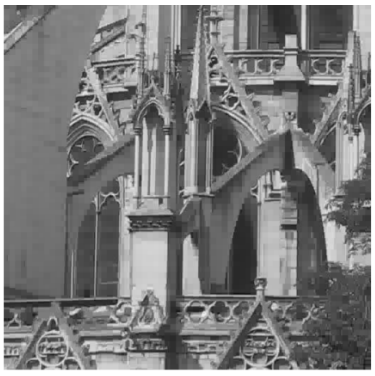
If you choose greater than 0, then elements of T are reset to zero, and therefore some original data is lost. The reconstituted image will contain distortions. This is what we referred to as lossy compression. The secret of compression is to choose your value so that compression is maximized while distortions in the reconstituted image are minimized.

We measure the level of compression by the compression ratio. The compression ratio is the ratio of nonzero entries in the transformed matrix T to the number of nonzero entries in the compressed matrix S. According to Dr. Colm Mulcahy of Spelman University, a compression ratio of 10 : 1 or greater means that S is sparse enough to have a significant savings in terms of storage and transmission time

IV. EXPERIMENT RESULT

The two images below represent a 10:1 compression using the standard and normalized Haar wavelet transform. Notice that the rounded and angled sections in the normalized image are of a higher quality.

Below is a 512 × 512 pixel grayscale image of the flying buttresses of the Notre Dame Cathedral in Paris:



(g) Standard, 10:1 Compression

(h) Normalized, 10:1 Compression

V. Conclusion and future work

A picture can say more than a thousand words. However, storing an image can cost more than a million words. This is not always a problem because now computers are capable enough to handle large amounts of data. However, it is often desirable to use the limited resources more efficiently. For instance, digital cameras often have a totally unsatisfactory amount of memory and the internet can be very slow. In these cases, the importance of the compression of image is greatly felt. The rapid increase in the range and use of electronic imaging justifies attention for systematic design of an image compression system and for providing the image quality needed in different applications. Wavelet can be effectively used for this purpose. A low complex 2D image compression method using Haar wavelets as the basis functions along with the quality measurement of the compressed images have been presented here. As for the further work, the tradeoff between the value of the threshold and the image quality can be studied and also fixing the correct threshold value is also of great interest.

I think that it is clear to see that the wavelets have their advantages.They greatly increase the speed at which image files are transferred electronically and drastically decrease the amount of valuable memory needed to store them. According to Dr. Mulcahy, the Haar wavelet transform is the simplest, crudest wavelet transform[4]. It does, however, illustrate the power and potential of wavelet transforms. They have broad applications in fields such as signal processing, computer graphics, medical imaging, seismology, as well as the imaging applications previously discussed. After witnessing the power Of the Haar wavelet transform I can only image what is possible with a more powerful transform.

References are important to the reader; therefore, each citation must be complete and correct. If at all possible, references should be commonly available publications.

References

[1] Strang, Gilbert Introduction to Linear Algebra

[2] Mulcahy, Colm Image Compression Using the Haar Wavelet Transform

[3] Ames, Rebecca For instruction in Adobe Photoshop

[4] Arnold, Dave College of the Redwoods

[5] Talukder, K.H. and Harada, K., A Scheme of Wavelet Based Compression of 2D Image, Proc. IMECS, Hong Kong, pp. 531-536, June 2006.

[6] Ahmed, N., Natarajan, T., and Rao, K. R., Discrete Cosine Transform, IEEE Trans. Computers, vol. C-23, Jan. 1974, pp. 90- 93.

[7] Pennebaker, W. B. and Mitchell, J. L. JPEG, Still Image Data Compression Standards, Van Nostrand Reinhold, 1993.

[8] Rao, K. R. and Yip, P., Discrete Cosine Transforms - Algorithms, Advantages, Applications, Academic Press, 1990.

[9] Wallace, G. K., The JPEG Still Picture Compression Standard, Comm. ACM, vol. 34, no. 4, April 1991, pp. 30-44.

APPENDIX

```
from __future__ import print_function
import numpy as np
import mahotas
from mahotas.thresholding import soft_threshold
from matplotlib import pyplot as plt
from os import path

# Let us load the data first:
luispedro_image = path.join(
                path.dirname(path.abspath(__file__)),
                'data',
                'luispedro.jpg')

f = mahotas.imread(luispedro_image, as_grey=True)
f = f[:256,:256]
plt.gray()
# Show the data:
plt.imshow(f)
print("Fraction of zeros in original image:", np.mean(f==0))

# A baseline compression method: save every other pixel and only high-order bits:
direct = f[::2,::2].copy()
direct /= 8
direct = direct.astype(np.uint8)
print("Fraction of zeros in original image (after division by 8):", np.mean(direct==0))
plt.imshow(direct)
```

```
# Transform using D8 Wavelet to obtain transformed image t:
t = mahotas.daubechies(f,'D8')
plt.imshow(t)

# Discard low-order bits:
t /= 8
t = t.astype(np.int8)
print("Fraction of zeros in transform (after division by 8):", np.mean(t==0))
plt.imshow(t)

# Let us look at what this looks like
r = mahotas.idaubechies(t, 'D8')
plt.imshow(r)

# Go further, discard small values in the transformed space:
tt = soft_threshold(t, 12)
print("Fraction of zeros in transform (after division by 8 & soft thresholding):", np.mean(tt==0))

# Let us look again at what we have:
rt = mahotas.idaubechies(tt, 'D8')
plt.imshow(rt)
```