Indian Institute of Technology, Mandi

Course : Digital Image Processing - EE608

Haar Wavelet Compression Report

Group Members	*Student ID*
Kena Patel	B18016
Sunil Kumar Singh	B18026
Ashish Anand	B18106

Contents

Abstract

Since data growth in this electronics era is huge, so we need to store data in the limited space. We need compression techniques for reducing the size of data. Data can be a recorded speech signal, image or a video file. In this project we are doing image compression using Haar Wavelet compression. Image compression allows transmission and storage of data efficiently without losing the main data but reducing the redundancy and noise present in it. In this project, we have implemented Haar transform for lossy and lossless compression of images. Lossy compression can be used and effective because when we see an image, we do not focus on finer variation in intensities in image rather than focus on larger variation in intensities. So such smaller variation can be removed to reduce the size of the image. Comparison of Haar transform with other wavelet transform is also shown in this report.

1 Introduction

Wavelet-Transform is a very popular and widely used Image compression method. As the name suggests, it uses wavelets that are oscillations with wave-like shape, i.e. amplitude of the signal starts with zero, increases and then again starts to decreases until it attains the value zero again. From the corresponding wavelet transform, we can get useful information about the signal including its frequency spectrum and temporal extent.

The specific kind of Wavelet compression method we have used in our project is called "Haar Wavelet Image Compression" method. This method comprises of two important steps- Approximation and Detailing. These are done by averaging and differentiating the pixel values respectively. Through averaging and differentiating of the given image matrix (both colored and gray-scale), we generate a more sparse image matrix, i.e. an image matrix that has more zero pixel value. This image matrix is the compressed image that is much smaller in size as compared to the given input image. This compressed image can then be decompressed to get the final image that is similar to the input image.

Note that, there are two implementations of the Haar Wavelet image compression- *Lossy* and *Lossless*. Both the implementations have been performed in this project. As the name suggests, the lossless implementation can be used where the final image after the reconstruction is exactly the same as the input image, but in the lossy implementation the final image after reconstruction loses some details. A more detailed explanation is given in the next section.

2 Haar Wavelet Compression

This sections covers detailed implementation of the Haar Wavelet compression for both gray-scale as well as colored images for both lossy and lossless implementations.

2.1 Step-wise Mathematical Explanation

After reading the image and converting into corresponding image matrix, we apply the Haar Wavelet compression in blocks of size 8x8. Let's say the first block in our image matrix looks like this:

```
[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;]
```

In every such block, we form four pairs of the columns to perform the transformation like the following:

```
   pair 1        pair2         pair 3        pair 4
[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]
```

And in this pairwise, we do the transformation in three steps. Note that since the transformations done in all the rows are same, we are just explaining the first row.

```
   pair 1        pair2         pair 3        pair 4
[156   159]   [158   155]   [158   156]   [159   158]
```

In the first step, we do averaging and differentiating, by replacing the first four pixel values with the average of the 4 pairs and replacing the rest 4 pixel values with half of the difference of the two values in each pair. After such transformation, the first row looks like this:

```
[157.5  156.5]   [157   158.5]   [-1.5   1.5]   [1   0.5]
```

In the second transformation, we do averaging in the first column and differentiating in the second pair, leaving the last two pairs as it is. After the second transformation, the first row looks like this:

```
[157  157.75]   [0.5   -0.75]   [-1.5   1.5]   [1   0.5]
```

In the third that is the last transformation, we perform replace the first pixel value with the average of the first pair and the second pixel value with the half other difference of the pixel values in the first pair. After this transformation, the first row looks like this:

```
[157.375  -0.375]   [0.5   -0.75]   [-1.5   1.5]   [1   0.5]
```

Similarly, we perform these transformation on the same row of the image matrix. We see the the values that we received after differentiating have become very smaller. In some cases, this value becomes zero as well that shows the adjacent pixel values don't change drastically in intensities. The lesser the number of non-zero pixel values in this matrix, the smaller will be the size of the compressed image.

This 8x8 matrix the iterates over the whole image matrix which can be of any size to transform all the values in the image matrix to finally give us the matrix that represents the compressed state of our image owing to the increased number of zero pixels.

2.2 Optimization of Mathematical Equations

One can use for loops, and in this case it will be many and much complicated, doing 3 transformations row wise and then doing the same column wise will add more time complexity to the code. So to simplify this, we consider step one, that is dividing into 4 pairs and first 4 columns will be average and next 4 columns will be differences divided by 2. To do this, we can define a matrix H_1 which can do this to all rows. And

let A be the 8x8 matrix block of the image. So, we can have our new matrix say $M_1 = A * H_1$. Here matrix H_1 will be

```
[0.5    0      0      0      0.5      0       0       0;
 0.5    0      0      0     -0.5      0       0       0;
 0      0.5    0      0      0        0.5     0       0;
 0      0.5    0      0      0       -0.5     0       0;
 0      0      0.5    0      0        0       0.5     0;
 0      0      0.5    0      0        0      -0.5     0;
 0      0      0      0.5    0        0       0       0.5;
 0      0      0      0.5    0        0       0      -0.5;]
```

Similarly, for transform 2, i.e. step 2 we did in the previous section, we need to apply similar to 4x4 i.e. left upper half of the updated matrix and remaining part will be untouched. For this, we use the matrix H_2 that multiples to the above matrix M. Heren H_2 will be

```
[0.5    0      0.5     0       0       0       0       0;
 0.5    0     -0.5     0       0       0       0       0;
 0      0.5    0       0.5     0       0       0       0;
 0      0.5    0      -0.5     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;]
```

Similarly, for step 3, H_3 matrix will be

```
[0.5    0.5    0       0       0       0       0       0;
 0.5   -0.5    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;]
```

So to perform row wise all the steps we have, updated matrix M as

$$M = A * H_1 * H_2 * H_3$$

Now, we need to update this matrix further but applying the same method column wise. To do this, we can take transpose of this M and then multiply the three matrices then again take the transpose so rows are converted back to columns. Let, $H = H_1 * H_2 * H_3$.

Therefore, the compressed image will be $(M^T * H)^T$. Simplifying this, we get

$$compressed\_img = ((AH)^T H)^T = H^T((AH)^T)^T = H^T * A * H$$

Hence, we get the simplified version of compressed image.

2.3 Implementation of Compression

The above method discussed is used in our implementation of the Haar Wavelet Image compression. So we have Matrix H i.e. $H_1H_2H_3$ as follows:

```
[0.1250    0.1250    0.2500         0    0.5000         0         0         0;
 0.1250    0.1250    0.2500         0   -0.5000         0         0         0;
 0.1250    0.1250   -0.2500         0         0    0.5000         0         0;
```

```
 0.1250    0.1250   -0.2500         0         0   -0.5000         0         0;
 0.1250   -0.1250         0    0.2500         0         0    0.5000         0;
 0.1250   -0.1250         0    0.2500         0         0   -0.5000         0;
 0.1250   -0.1250         0   -0.2500         0         0         0    0.5000;
 0.1250   -0.1250         0   -0.2500         0         0         0   -0.5000;]
```

So the compressed image A_2 will be as follows:-

$$A_2 = H^T * A * H$$

Here, H^T represents transpose of matrix H, A represents an 8x8 block of our image matrix, and H represents our transformation matrix. The resultant A_2 is the compressed equivalent of the 8x8 image matrix block. Performing the same operation on all the 8x8 blocks of the image matrix results in the matrix that represents the compressed form of the given image.

2.4 Implementation of Decompression

The decompression of the compressed form of the image is fairly simple and can be done by yet another simple matrix multiplication as follows: From $A_2 = H^T * A * H$, we need A back, that will be out decompressed image. So multiply by inverses of left and right side of A_2 will get

$$A = (H^T)^{-1} * A_2 * H^{-1}$$

Here, H^{-1} represents inverse of matrix H. Performing this same operation on all the 8x8 blocks of the image matrix decompresses and results in the image that is exactly the same as the image matrix.

These implementations of compression as well as decompression differ a bit depending upon what kind of image are we compressing and whether we are performing lossy or lossless compression.

2.5 Implementations for Different Types of Images

2.5.1 For Grayscale Images

Since a grayscale image has only one channel and its image matrix if of shape MxN only. We can perform the compression using the same matrix multiplication way described above. But the above implementation does lossless compression.

Lossless compression compresses image such that after the decompression the final image reconstructed is same as the given input image. But since, the number of zeroes in the compressed state is very less, the compression ratio is fairly small. **Compression ratio** is defined as the ratio of the number of non-zero pixels in the input image with the number of the non-zero pixels in the final compressed image. This also makes size of the compressed image small but not very small. To further reduce the size of the compressed image, we can perform lossy compression.

In lossy compression, after compression, we set a threshold named **delta** with very small value, say 1. And we replace all the pixel values in the matrix (that represents the compressed state) whose absolute value is less that *delta* with zero, This way, the number of zeroes in the compressed form increases and thus, the compression ratio increases as the number of non-zero pixels decreases in the compressed form. This makes the final compressed image much smaller. To further decrease the size, we can increase the value of *delta* and thus increase count of pixels with value zero.

2.5.2 For Colored Images

Since a colored image has three channels, thus we need to implement Haar Wavelet Image compression on the three channels separately and then reconstruct the final colored image from the three compressed forms of different channels.

Both the lossless and the lossy Haar Wavelet compression implementations can be used on the individual channels just as we did with the grayscale.

3 Results

3.1 For Grayscale Images

Output:

```
Enter file name with extension (Example: cameraman.tif) :
mandi.tif
File size of input image (in bytes): 576958.000000
```

Lossy Grayscale Image:

```
File size of decompressed image (in bytes): 72621.000000
No. of non-zeroes entries in Original Image: 6120164
No. of non-zeroes entries in Compressed Image: 254729
No. of non-zeroes entries in Decompressed Image: 6120527
Compression Ratio: 2.402618e+01
PSNR Value: -1.337623e+01
```

Lossless Grayscale Image:

```
File size of decompressed image (in bytes): 81023.000000
No. of non-zeroes entries in Original Image: 6120164
No. of non-zeroes entries in Compressed Image: 5815910
No. of non-zeroes entries in Decompressed Image: 6120164
Compression Ratio: 1.052314e+00
PSNR Value: Inf
```

Fig. 1. Gray Scale Original Image

Fig. 2. Gray Scale lossy Image Compression

Fig. 3. Gray Scale lossless Image Compression

As seen in the results obtained above, 576KB image is compressed and is now reduced to 72KB and 81KB for lossy and lossless respectively. As clearly seen number of zeros in compressed images increases hence it takes lesser file size. The delta is 5 in lossy compression algorithm. From comparison between Fig. 1 and Fig. 2, we can see that despite of high value of delta, reconstructed image from lossy compression looks same as original image. Despite of data loss there is not any noticeable difference between the two. Also in case of lossless Grayscale, PSNR is inf, indicating the original and decompressed images are same and no data is lost.

3.2 For Colored Images

Output:

```
Enter file name with extension:
peppers.png
File size of input image (in bytes): 23509.000000
```

Lossy Color Image:

```
File size of lossy color decompressed image (in bytes): 7136.000000
No. of non-zeroes entries in Original Image: 579071
No. of non-zeroes entries in Compressed Image: 32215
No. of non-zeroes entries in Decompressed Image: 578026
Compression Ratio: 1.797520e+01
PSNR Value: -1.394894e+01
```

Lossless Color Image:

```
File size of lossless color decompressed image (in bytes): 10153.000000
No. of non-zeroes entries in Original Image: 579071
No. of non-zeroes entries in Compressed Image: 538243
No. of non-zeroes entries in Decompressed Image: 579071
Compression Ratio: 1.075854e+00
PSNR Value: Inf
```

Fig. 4. Color Original Image

Fig. 5. Color lossy Image Compression

Fig. 6. Color lossless Image Compression

As seen in the results obtained above, 23KB image is compressed and is now reduced to 7KB and 10KB for lossy and lossless respectively. As clearly seen number of zeros in compressed images increases hence it takes lesser file size. The delta is 5 in lossy compression algorithm. Despite such high value of delta for lossy case, recontructed image from lossy case still gives all the important details of the image. From Fig. 5, one can see that image is slightly blurred, low strength edges on violet background is removed in the process.

4 Comparison with Other Compression Methods

We also compared the results of our Haar Wavelet Image Compression implementation with other inbuilt compression methods. The following PSNR results are obtained upon compressing the grayscale image *cameraman.tif* image with our compression method, spatial orientation tree wavelet compression method (STW), and the set partitioning in hierarchical trees compression method (SPIHT):

```
PSNR Value for our Lossless Haar: Inf
PSNR Value for our Lossy Haar: -2.772571e+00
PSNR Value for STW: -2.178014e+01
PSNR Value fof SPIHT: -4.062358e+01
```

Here, we see that as expected in the lossless Haar implementation the PSNR value is tending to infinity which is because the reconstructed image is exactly same as the input image (i.e. lossless) and thus the mean squared error is zero that makes the PSNR value infinitely big.

Upon comparing the PSNR values of our lossy Haar Wavelet implementation with other Wavelet implementations like STW and SPIHT, we observed that all the PSNR values are negative. This is because the PSNR is logarithm of square of peak signal divided by the mean squared error. Negative value shows that the peak signal is less than the mean squared error, due to noise. The magnitude of all the PSNR values except for the Lossless case, is very small. But even the lossy implementation of the Haar wavelet has smaller PSNR magnitude as compared to SPIHT compression method showing that it has less square of peak signal to mean squared error ratio. We also see that the magnitude of PSNR of STW compression is lowest suggesting smaller mean squared error as compared to the other two.

5 Conclusion

We successfully implemented the Haar Wavelet Image Compression in MATLAB that were able to compress and decompress both gray-scale image and colored image. Further, after implementing both lossless and lossy implementations of Haar Wavelet Compression, we concluded that although the lossless implementation gives

a compressed form of image that decompressed to get the exact image, but the compressed file can be further reduced in size by using lossy implementation. We also concluded that there is trade-off between size of file and correctness of reconstructed image after decompressing. The bigger the threshold value *delta* the smaller the size. After comparing our implementation with other in-built compressing methods, we concluded that lossless Haar Wavelet compression performs best, and even the lossy implementation of Haar wavelet can perform better than the SPIHT wavelet method.

References

[1] https://people.math.osu.edu/husen.1/teaching/572/image_comp.pdf

[2] S. S. Tamboli and V. R. Udupi, "Image compression using Haar Wavelet Transform", International Journal of Advanced Research in Computer and Communication Engineering, Vol. 2, Issue 8, August 2013.

[3] https://www.mathworks.com/help/wavelet/ref/wcompress.html