Danmarks Tekniske Universitet

Final Report - Fingerprint Image Compression Based On Wavelet Transform

AUTHOR

XUAN LUO - s192270

SUPERVISOR

Søren Forchhammer

May 19, 2020

Contents

List of Figures **I**

List of Tables **II**

1 Introduction **2**

2 Theory Description and Block Implementation **4**
2.1 Discrete Wavelet Transform and Inverse Transform 4
2.1.1 Filter Choice 4
2.1.2 Decomposition and Reconstruction 5
2.2 Quantization and Dequantization 8
2.3 Huffman Coding and Decoding 9

3 Test Result and Analysis **12**
3.1 Quality Assessment 12
3.1.1 Objective Assessment 12
3.1.2 Subjective Assessment 13
3.2 Rate Distortion 15
3.3 Compression Ratio 16
3.4 Code length and Entropy Estimation 17

4 Conclusion **18**
4.1 Future Work 18

References **19**

5 Appendix **20**
5.1 The Extra Test Results 20
5.2 Matlab Code of related blocks 21

List of Figures

1 An overview design of the project . . . 3
2 Two types of filters 'db4' and 'bior3.5' . . . 4
3 2-D DWT decomposition[1] . . . 5
4 The subbands structure for one-level and four-level, 2-D wavelet transform[1] . . . 5
5 One-level 2-D DWT decomposition with bior 3.5 filter . . . 6
6 Three-level 2-D DWT decomposition . . . 7
7 The decomposition and reconstruction process[2] . . . 7
8 Midtread quantizer . . . 9
9 Huffman encoding procedure[1] . . . 10
10 PSNR between two filters with differnt quantization level . . . 13
11 Quality comparison of original and reconstructed fingerprints . . . 14
12 Rate distortion . . . 15
13 Compression Ratio . . . 16
14 Quality comparison of original and reconstructed "Lenna" picture . . . 20
15 Rate distortion(10 test images) . . . 21

List of Tables

1 Huffman encode for the five-letter case 10
2 Huffman encoder tested with quantizated fingerprint(Quantizted level=30) 11
3 Adaptive and actual codelength 17

Abstract

The report describes a fingerprint image compression project based on Discrete Wavelet Transform(DWT). The project mainly consists of four steps: the first step is wavelet transform decomposition; the second step is the scalar quantization of the wavelet coefficients; the third step is entropy encoding and the final step is the decoding, dequantization and reconstruction. We further measure and analyse the PSNR values, rate-distortion, compression ratio and codelength. In the experiment, a suitable PSNR value is 30-40 dB, and the compression ratio of those are around 3.6-6.4.

1 Introduction

Data compression is a process which encodes the information with fewer bits than the original representation. There are two basic ideas of data compression namely the lossless and lossy compression. The lossless compression reduces bits by identifying and eliminating statistical redundancy[3]. Although lossless compression preserves all the information perfectly, the compression rate would rather low. Conversely, lossy compression gives a much higher compression rate by removing unnecessary or less important information. In most cases people cannot observe the removing details and using lossy compression can save much storing resource. While it's not suitable for some data such as HiFi music and high quality pictures, because the loss will still influence the user experience in such cases. Therefore, there's a quality-space trade-off in data compression scheme.

Image compression is necessary for medical or biometrical images, for instance different kinds of CT images or fingerprints because the data sets of those are usually very large. In order to save the storing resource and remain as much detail as possible, a new standard named JPEG 2000 has developed. Comparing to existed standard JPEG, JPEG 2000 offers a superior performance at low bit rates (e.g., below 0.25 b/p for highly detailed gray-scale images)[4]. Besides, for the fingerprint image compression, JPEG 2000 uses DWT which preserves more details at the edges and has a better quality in high frequency comparing to another popular transform method Discrete cosine transform(DCT)[5].

This project is somehow like a simple version of JPEG2000, containing the basic blocks and implementing the basic functions. Figure 1 shows an overview design of the project, which mainly consists of three procedures: wavelet transform decomposition/reconstruction, scalar quantization/dequantization and entropy encoding/decoding. In the project, the source fingerprint image is decomposed into subbands by DWT and then passed to a scalar quantizer. Next the outputs from the quantization encoder are encoded by the entropy encoder which uses the Huffman coding method. The compressed binary data will go through an inverse process and recover to the reconstructed image. All the inverse processes are similar to the encoding processes and will be also illustrated in the block implementation section. Finally, we will analyse the test results like quality estimation, trade off between rate and distortion, codelength and entropy estimation.

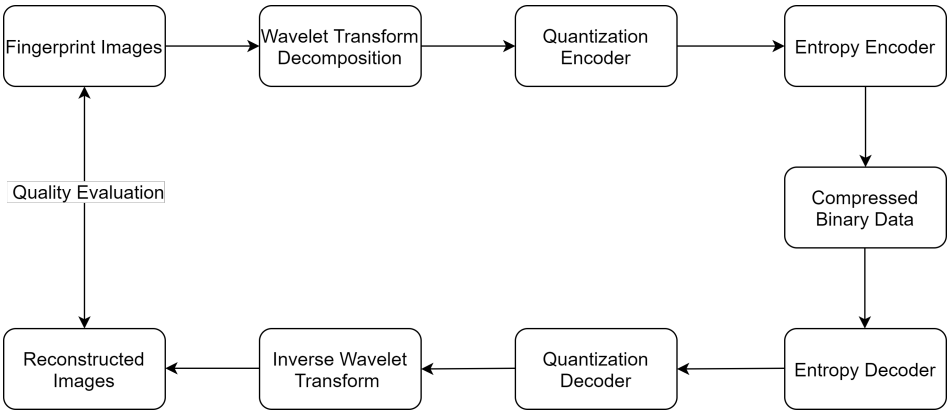


Figure 1: An overview design of the project

The goal of the project[6] is to:

- Implement an encoder and decoder.
- Calculate and evaluate the compression factor and the image quality objectively and subjectively.
- Analyse and explain the code length in relation to an entropy estimate.
- Explain decorrelation of data for compression, quantization of a data representation and entropy coding.

2 Theory Description and Block Implementation

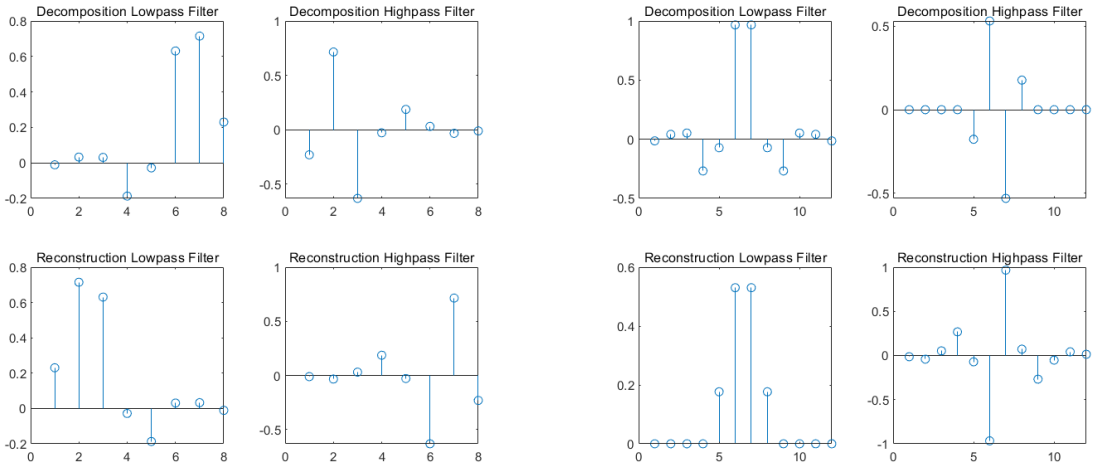
This section introduces the theory and block implementation, including the Discrete Wavelet Transform/Inverse Transform, Scalar Quantization/Dequantization and Huffman coding/decoding. Also the test results for each block will be shown in the corresponding subsection.

2.1 Discrete Wavelet Transform and Inverse Transform

Discrete Wavelet Transform (DWT) is a transform method which has been widely used in image compression, such as JPEG 2000 standard. It is capable of providing the time and frequency information simultaneously, hence giving a time-frequency representation of the signals[7]. This characteristic provides the higher spatial resolution, and lower frequency resolution.

2.1.1 Filter Choice

Before decomposing the source image with the Discrete Wavelet Transform(DWT), it's important to choose an appropriate filter first. There are mainly two kinds of filter families we can choose namely the Orthogonal Wavelet Daubechies Filter(db) and Biorthogonal Filter(bior). Figure 2 shows this two kind of filters, 'Daubechies Filter 4' and 'Biorthogonal 3.5' respectively. The biggest difference is that "db" is least asymmetric wavelet, while "bior" is symmetric wavelet and linear phase. The bior has the advantage that the use of symmetric extensions and linear phase wavelet filters can solve the problem of border effects in the wavelet transform. Therefore, theoretically using the Biorthogonal filter will have a better performance in PSNR comparing to Daubechies Filter and in section 3 we will verify the PSNR performance by testing the actual fingerprint images.



(a) db4 (b) bior3.5

Figure 2: Two types of filters 'db4' and 'bior3.5'

2.1.2 Decomposition and Reconstruction

The DWT of a signal x is calculated by passing through a series of filters. First the samples are passed through a low pass filter H_1 and a high pass filter H_0 simultaneously with impulse response resulting in a decomposition of the two parts. Next is repeating the decomposition from two dimensions to further increase the frequency resolution and get an approximation coefficient matrix, as shown in figure 3. The source image can be decomposed to different subbands depending on the decomposition level, as shown in figure 4.

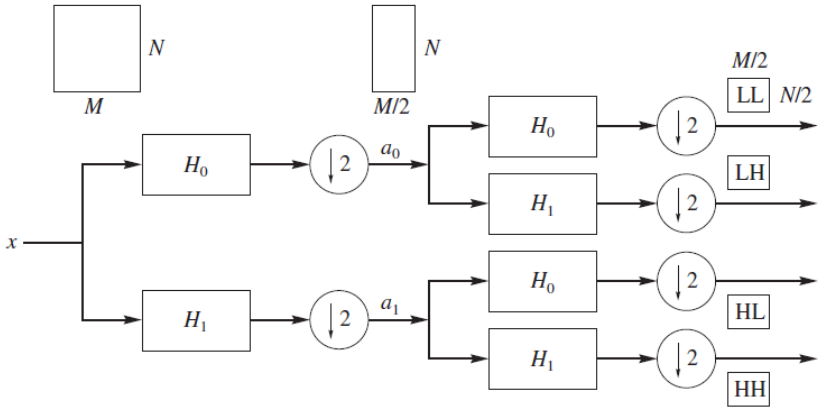
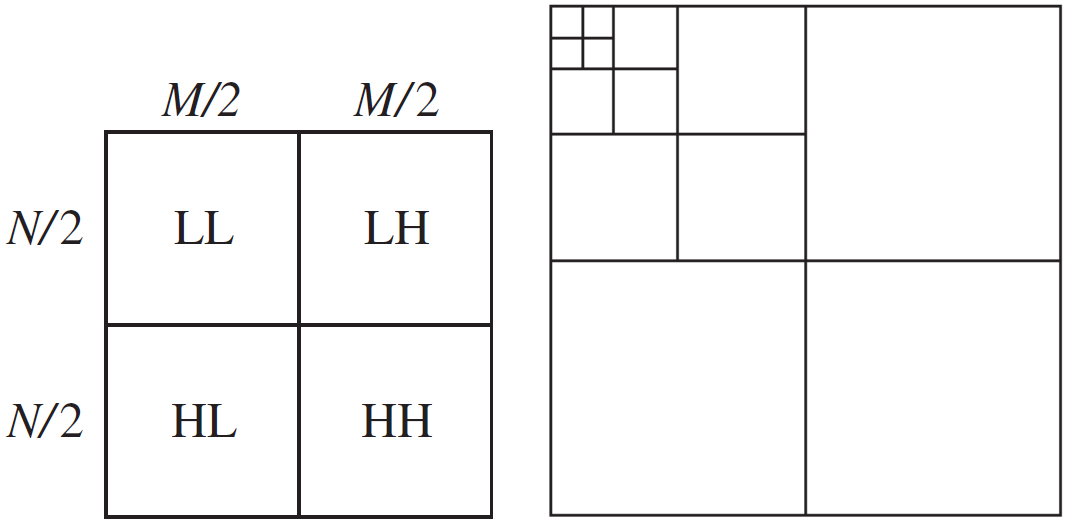


Figure 3: 2-D DWT decomposition[1]



(a) One-level, 2-D DWT Decomposition (b) Four-level, 2-D DWT Decomposition

Figure 4: The subbands structure for one-level and four-level, 2-D wavelet transform[1]

We can summarize the 2-D DWT decomposition as equation 1-4. Here the 2-D dimension is denoted as $[m, n]$ and the decomposition level can be increased further by repeating the equations below:

$$x_{1,LL}[m,n] = \sum_{k=0}^{K-1} v_{1,L}[2m-k,n]H_0[k] \tag{1}$$

$$x_{1,HL}[m,n] = \sum_{k=0}^{K-1} v_{1,L}[2m-k,n]H_1[k] \tag{2}$$

$$x_{1,LH}[m,n] = \sum_{k=0}^{K-1} v_{1,H}[2m-k,n]H_0[k] \tag{3}$$

$$x_{1,HH}[m,n] = \sum_{k=0}^{K-1} v_{1,H}[2m-k,n]H_1[k] \tag{4}$$

We implement the decomposition block in Matlab and test with a fingerprint example. The one-level and three-level decomposition are namely shown in figure 5 and 6. Note that in figure 6 the high-frequency bands have been enhanced to show details. In such decomposition cases, we observe that the coefficients of LL_K subband are relatively large, which means the most energy in the DWT preserves in the LL_K subband. While the energy preserved in the high frequency bands is small(in three level case the coefficient values are almost zero or negative).

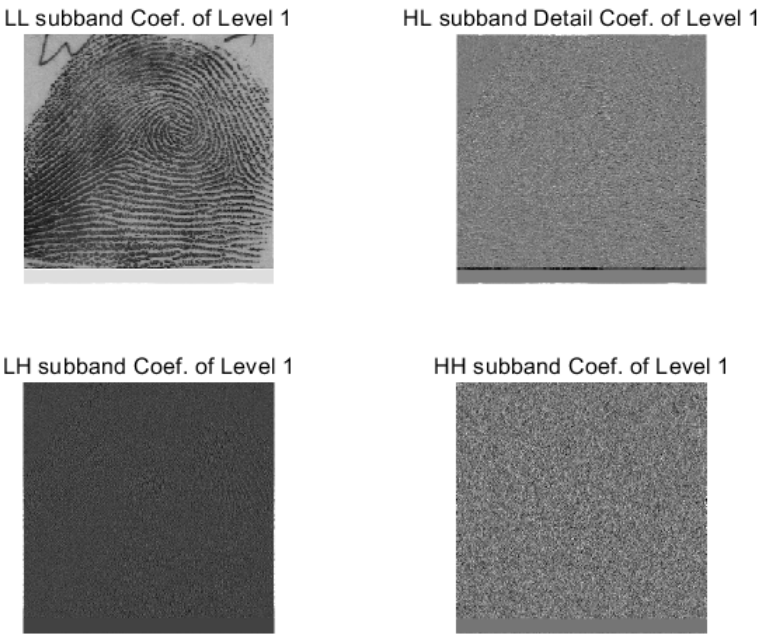


Figure 5: One-level 2-D DWT decomposition with bior 3.5 filter

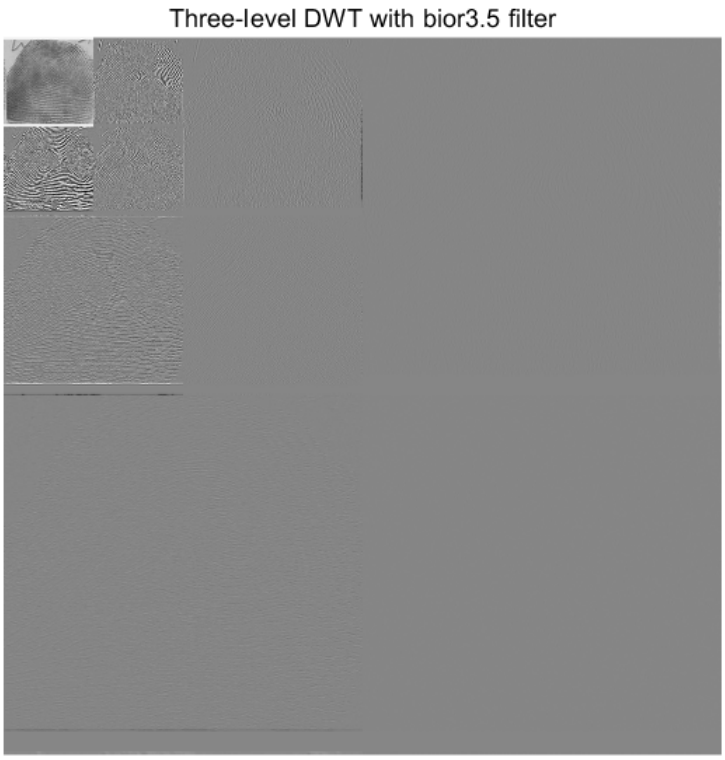


Figure 6: Three-level 2-D DWT decomposition

After DWT, the source image is decomposed into subbands of floating-point coefficients. The coefficient matrix is then passed through a uniform scalar quantizer and quantized to an integer coefficient matrix, which will be illustrated in section 2.3.

The reconstruction is an inverse transform of DWT decomposition straightly by upsampling the decomposed coefficients, as the right side of figure 7. In Matlab, we can use the "idwt" to implement the function.

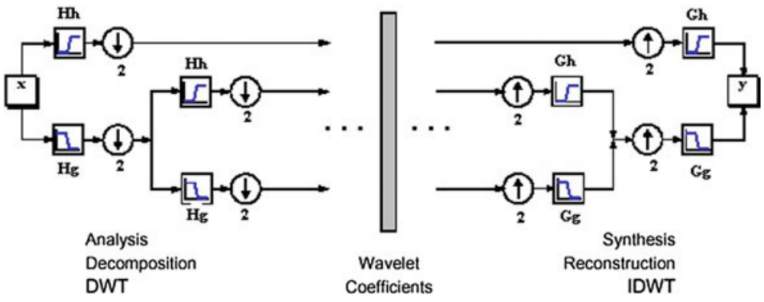


Figure 7: The decomposition and reconstruction process[2]

2.2 Quantization and Dequantization

Quantization is one of the simplest and most common ideas in lossy compression. One idea of quantization is to quantize the float-point coeffients to integer. By losing some less important information, it compresses the image coefficient data and reduces the storage size. In this subsection, a scalar quantization& dequantization method and implementation is introduced.

Quantization consists of Scalar Quantization and Vector Quantization. In Scalar Quantization, each input coefficient is treated separately and gets a quantized output. While in Vector Quantization, the inputs are placed in groups as vectors, then are quantized as outputs.

In the project, we implement a uniform scalar quantizer, as shown in figure 8. In this quantizer, which is also called midtread quantizer, all intervals are in the same size, except two outer intervals. In other words, all the decision boundaries and the corresponding values are spaced evenly. In addition, as the midtread quantizer has zero as one of its output levels, it is useful in this project, because the zero value can be represented. The implementation of the midtread quantizer can be expressed by the equation below:

$$Q = round(\mathrm{D}/L) * L \tag{5}$$

Here Q denotes the integer coefficient matrix, D denotes the float-point coefficient matrix and L denotes the quantization level.

Dequantization is the inverse process of quantization, which maps the quantized coefficients Q back to the reconstructed dequantization values. The specific implementation is similar to the equation 5.

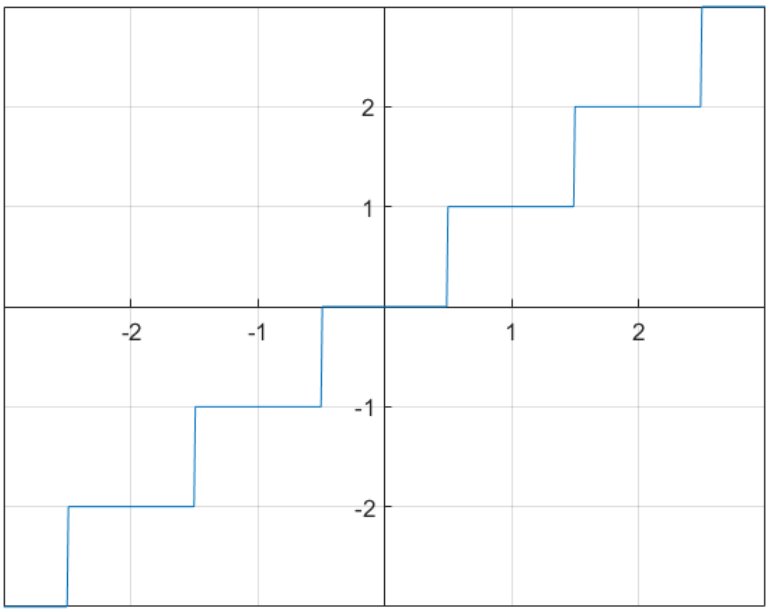


Figure 8: Midtread quantizer

2.3 Huffman Coding and Decoding

Huffman coding is a lossless entropy coding scheme which is widely used for different kind of data compression. The Huffman coder consists of the encoder and decoder and we mainly discuss the encoder part because the decoder is simply the inverse process. The encoded values from Huffman's encoder are a list of a variable-length codes. There are four steps for the Huffman encoding scheme:

- First we arrange the coefficient matrix into 1-D array and calculate the probability of each symbol in the array. The codewords can be assigned as $c\left(a_n\right)$ according to the symbol.
- Second we sort the symbol in a descending probability order. The two symbols with the lowest probability are denoted as a_{n-1} and a_n. The codeword of them can be assigned as:

$$\begin{aligned} c\left(a_n\right) &= \alpha_1 * 0 \\ c\left(a_{n-1}\right) &= \alpha_1 * 1 \end{aligned} \tag{6}$$

  We concatenate them as a new letter a'_n. The probability of a'_n will be the sum probability of a_{n-1} and a_n.
- Third we repeat step 1 and 2 until there are only two symbols in the reduced alphabet.
- When there are only two symbols, the codeword can be simply assigned 0 and 1.

Here we design a simplified case[1] to illustrate the steps concretely. Suppose the input letters are $\{a_1, a_2, a_3, a_4, a_5\}$ with the probabilities $P(a_1) = P(a_3) = 0.2, P(a_2) = 0.4$, and $P(a_4) = P(a_5) = 0.1$. By following the step 1-4 we can draw a Huffman encoding procedure:

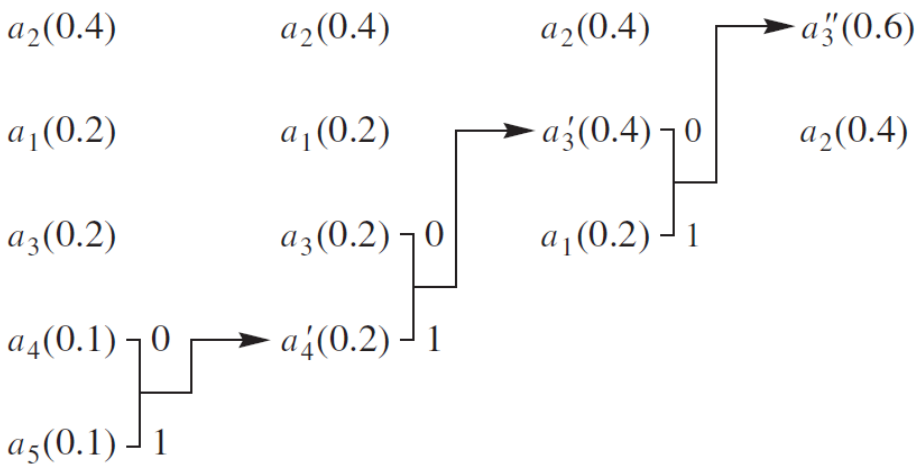


Figure 9: Huffman encoding procedure[1]

We further list the result as a Huffman variable-length code table:

Table 1: Huffman encode for the five-letter case

Letter	Probability	Codeword
a_2	0.4	1
a_1	0.2	01
a_3	0.2	000
a_4	0.1	0010
a_5	0.1	0011

The average length for this code is

$$l = 0.4 \times 1 + 0.2 \times 2 + 0.2 \times 3 + 0.1 \times 4 + 0.1 \times 4 = 2.2 \text{ bits/symbol} \quad (7)$$

In the project, all subbands are encoded as one data set using the same Huffman table. In other words, all the quantizated subband coefficients of the image are arranged to a 1-D array, then encoded by one Huffman encoder. The Huffman coding process is also implemented by Matlab: we use function "huffmanenco" and "huffmandeco" to encode and decode the array. Table 2 shows an encoded fingerprint result through a Huffman encoder:

Table 2: Huffman encoder tested with quantizated fingerprint(Quantizted level=30)

Coefficient value	Probability	Codeword
0	0.8051	0
-21	0.0496	101
...	...	...
84	0.0074	111100
...	...	...
-798	3.8147e-06	10011000001000111

In this example the image is quantized with a three-level quantization, most coefficients are 0 and consequently there is a very high probability of the value 0, which the Huffman code assigns 1 bit to code. Here one way to improve the coding can be using an Embedded Zero-trees of Wavelet(EZW) algorithm to avoid allocating any bits to encode 0 values. The detail of EZW will be illustrated in section 4.1.

3 Test Result and Analysis

We have introduced the implementation and the test within blocks in section 2. In this section, we will test ten fingerprint images, show and analyse the overall experimental results including the quality assessment, rate-distortion, compression ratio and code length in relation to the entropy estimation.

3.1 Quality Assessment

We evaluate the quality of compressed fingerprints objectively and subjectively, specifically by calculating the Peak Signal-to-Noise Ratio(PSNR) and evaluating the quality visually.

3.1.1 Objective Assessment

PSNR is a common method to assess the quality of compressed image objectively, which calculates the size of the error relative to the peak value of the signal:

$$PSNR = 10 \cdot \log_{10} \left(\frac{MAX_I^2}{MSE} \right) \tag{8}$$

The MAX_I is the maximum possible pixel value of the image. In this project, all test fingerprints are 8-bit images so the MAX_I here equals to $2^8 - 1 = 255$. MSE refers to Mean Square Error between original image array X_n and reconstructed image array Y_n.

$$MSE = E \left\{ \frac{1}{N} \sum_{n=1}^{N} (X_n - Y_n)^2 \right\} = \frac{1}{N} \sum_{n=1}^{N} E \left[(X_n - Y_n)^2 \right] \tag{9}$$

In our lossy compression scenario, all the loss comes from the quantization. In order to find the range that balances the quality(PSNR) and compression level(quantization level), we draw a PSNR plot varying from quantization level 1 to 91(each interval is 10), as shown in figure 10. Note that here the PSNR is a mean value of ten compressed fingerprints.

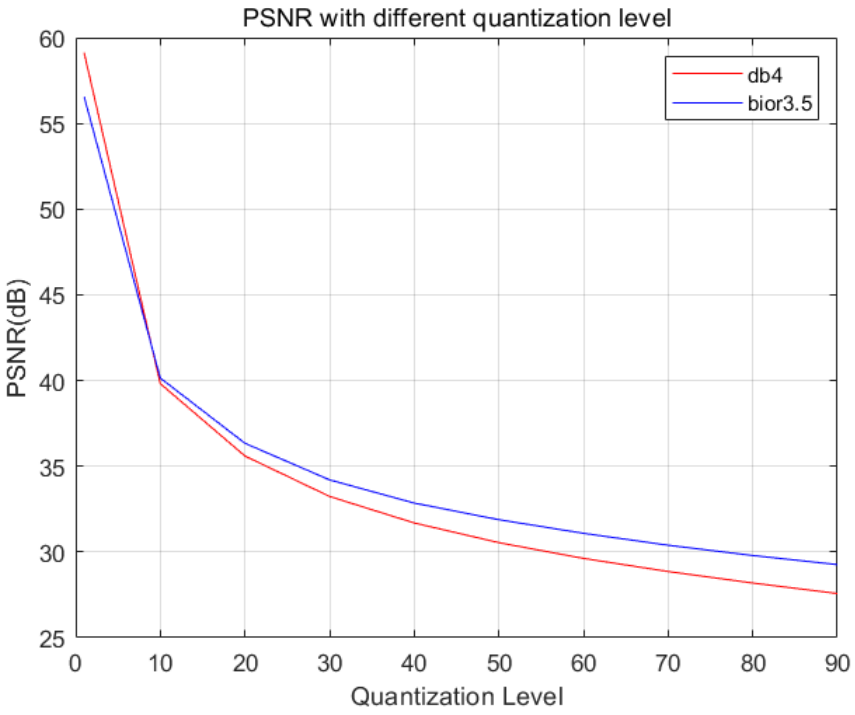


Figure 10: PSNR between two filters with differnt quantization level

Within our expectation, the PSNR performance drops when the quantization level(QL) increases. In addition, the PSNR drops substantially in the first 10 QL. The reason is there are some details in high frequency subbands that being removed in the quantization. Afterwards the decreasing trend of the curve becomes slower is because most of the signal energy is concentrated in the low frequency subband, which is less sensitive to the QL. In figure 10 we can also find out an optimal range of QL. According to [8], the typical values for the PSNR in lossy image compression are between 30 and 50 dB, provided the bit depth is 8 bits, and the acceptable PSNR values for wireless transmission are between 20 and 25 dB. Therefore considering a quality-space trade-off, a reasonable QL would be 10 to 80, where the PSNR values are ranging from 30 to 40dB.

Moreover, we notice that in the test the PSNR performance using "bior3.5" filter is better than that with "db4", except when QL is rather small(less than 10), which verifies the theoretical inference. Due to the better performance in most quantization level, we set "bior 3.5" as our default wavelet filter in the following tests.

3.1.2 Subjective Assessment

In 3.1.1 we know that a good QL range is from 10 to 80. Now we evaluate the image quality visually. Here we pick up four fingerprint images, respectively the original, three reconstructed images with QL=10, 80 and 150. The PSNR values for those three reconstructed images are namely 40.6dB, 29.3dB and 26.2dB.

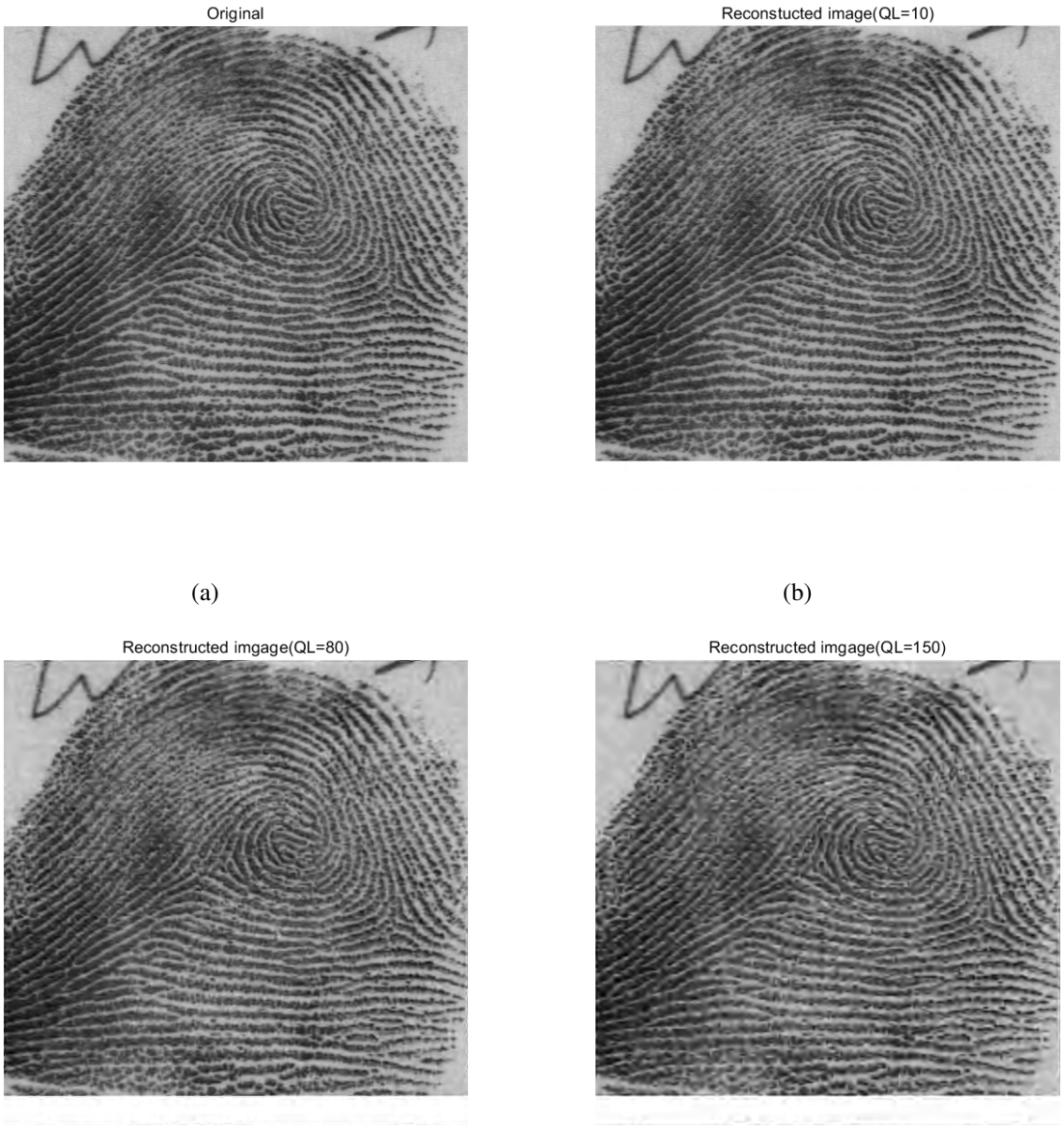


Figure 11: Quality comparison of original and reconstructed fingerprints

In figure 14, we cannot recognize the difference between the original and the reconstructed image when QL=10, because human eye is not sensitive to the high frequency components. When QL = 80, although the lossy compression slightly obscures fingerprint vein, the image quality is still acceptable. While when QL increases to 150, the image is quite unclear: it is hard to identify the fingerprint detail such as vein and border. Hence, in this visual assessment we prove that an acceptable PSNR value is no less than 30dB, otherwise the image quality will be rather poor.

Notice that the subjective quality of experience may differ because of different type of images. Appendix also shows the reconstructed "lenna" image with different QL. We find that it is easier to observe the distortion of the image so the acceptable QL will be lower to preserve a better detail (such subjectively experience may also happen in some images focusing on the quality such as portrait, art and so on)

3.2 Rate Distortion

In lossy compression scheme, we always need to consider the trade-offs between rate and distortion. Rate refers to the average number of bits used to represent each sample value, which is calculated as bit per pixel(bpp). The distortion refers to the PSNR value. Here we plot the rate-distortion among five different fingerprint images, as shown in figure 12. Note that a rate distortion plot among ten fingerprint images has been put into the Appendix.

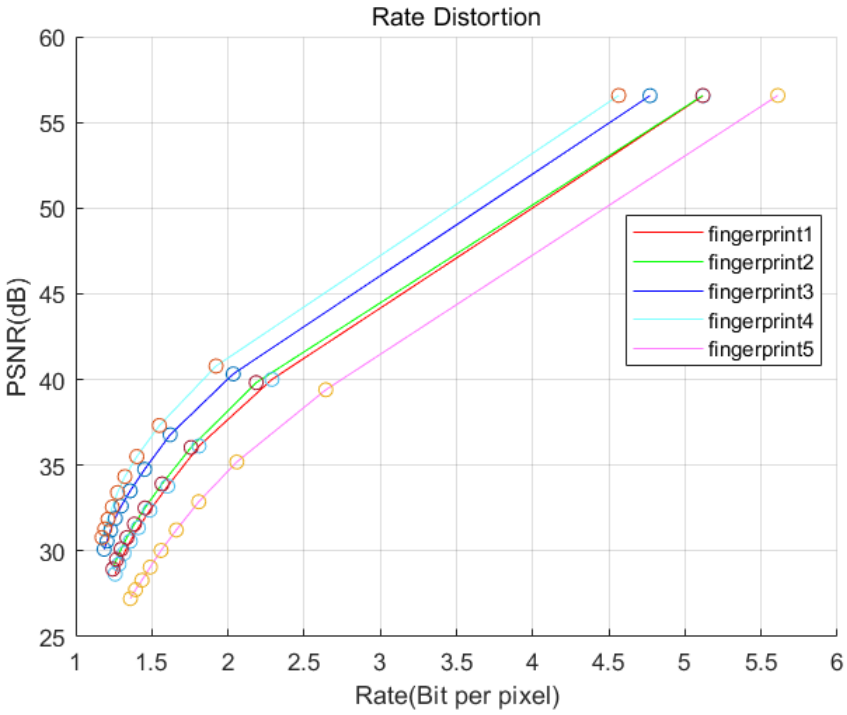


Figure 12: Rate distortion

As expected, the rate-distortion curves are similar to each other. When the PSNR is ranging from 30 to 40 dB, an average bit rate will be around 1.25bpp to 2.25bpp, which is closed to the performance in [4] (It is reasonable that the specific value is slight different due to the different source images and implementation).

3.3 Compression Ratio

Compression Ratio(CR) is a way to measure how well the compression scheme performs in the space dimension. The main idea is to look at the ratio of the number of bits required to represent the data before compression to the number of bits required to represent the data after compression.

$$\mathrm{CR} = \mathrm{S_{uncomp}} / \mathrm{S_{comp}} \tag{10}$$

Here S_{uncomp} and S_{comp} denote as the size of source image and compression image. Nevertheless, compressed images are usually transformed into different representations, so we can evaluate the compression ratio by taking the following equation instead :

$$\mathrm{Bpp_{comp}} = \mathrm{S_{comp}} / \mathrm{N_{Pixels}} \tag{11}$$

As the number of pixels (N_{Pixels}) remains unchanged and the source fingerprint images are all 8-bit images, the compression ratio can be evaluated as follows:

$$\mathrm{CR} = 8 / \mathrm{Bpp_{comp}} \tag{12}$$

We calculate the average bit rate in ten fingerprints with different QL and the result is shown in figure 15. From section 3.2 we know that a reasonable QL is 10-80, which indicates the compression ratio is around 3.6 to 6.4.

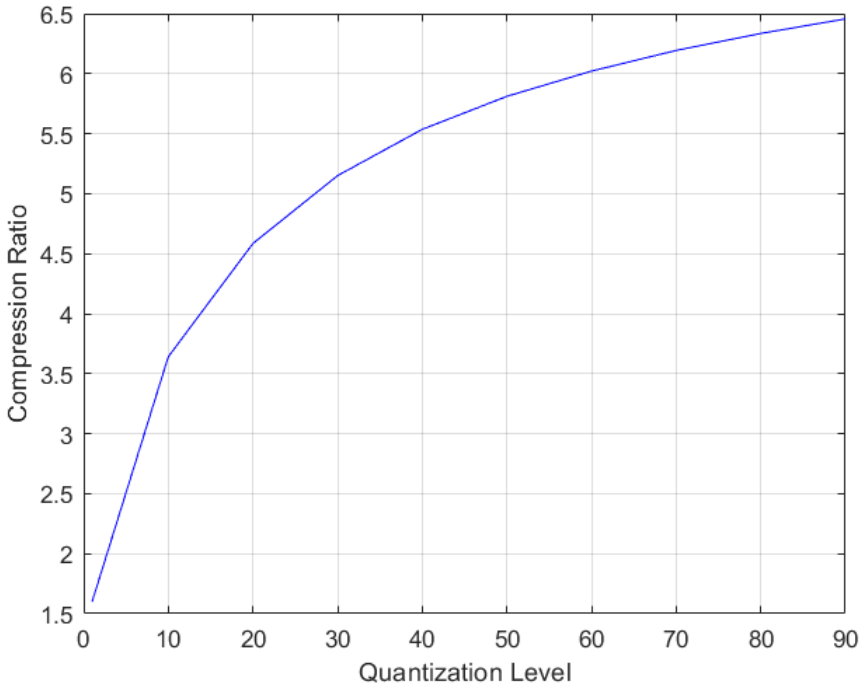


Figure 13: Compression Ratio

3.4 Code length and Entropy Estimation

In this subsection, we measure the code length from the Huffman encoder and analyse it in relation to the entropy estimation.

It is almost impossible get a "true" entropy in the real data compression process. Nevertheless we can use an adaptive code length providing an upper bound on the entropy. In a finite state source of order, the expected value of upper will converge to the entropy.

The ideal adaptive codelength for x^n is defined as

$$L\left(x^n\right) = \sum_{k=0}^{n} -\log \hat{p}\left(x_k|c_k\right) \tag{13}$$

x^n denotes as the coefficient array of the image and $\hat{p}\left(x_k|c_k\right)$ denotes the conditional probability regularized by previous causal context x^{k-1}. The estimate probability is

$$\hat{p}\left(x_k = i|c_k\right) = \frac{n_i + \delta}{N + |A|\delta}, \quad 1 \leq i \leq |A|, \quad \sum_i n_i = N \tag{14}$$

n_i denotes all the different length of symbol in the Huffman code. N denotes the sum of n_i. A denotes the number of symbols and δ equals to 1[9]. Now we calculate the ideal adaptive codelength and the actual Huffman codelength with a different QL and different fingerprint images. We use equation (13) and (14) to calculate the adaptive codelength with a conditional probability $\hat{p}$. The adaptive codelength and actual codelength(not includes the required codelength of Huffman table) of three fingerprint images are shown in table 3.

Table 3: Adaptive and actual codelength

Image index	Quantization Level	Adaptive Codelength	Actual codelength	Codelength of Huffman table
Fingerprint1	1	37476	12645	38965
Fingerprint1	10	4478	764	4285
Fingerprint2	1	38804	13042	40383
Fingerprint2	10	4230	723	4008
Fingerprint3	1	30779	9671	31994
Fingerprint3	10	4201	622	3922

We find that there is a relative large difference between adaptive codelength and Huffman encoded codelength. The reason is that the performance of the adaptive codelength on all symbols relies on the statistics and contexts for all symbols. In some cases, the adaptive codelength can be possibly lower than the actual codelength because when coding a high probability event (p>0.5), the Huffman coding requires 1 bit but for the adaptive code length, it only needs less than 1 bit to to code($-\log \hat{p}\left(x_k|c_k\right)$<1). A possible way to avoid the case is to combine more than one symbol in a Huffman code.

4 Conclusion

In this project, a lossy fingerprint image compression scheme based on wavelet transform has been implemented in Matlab, including the decomposition/reconstruction, quantization/dequantization, encoding and decoding blocks. In the experiment part, the image quality has been assessed objectively and subjectively. Moreover, rate distortion and compression ratio have been measured and analysed. The PSNR value of a good compressed fingerprint image is around 30-40 dB, with a 10-80 quantization level and the compression ratio ranges from 3.6 to 6.4. In addition, the ideal adaptive codelength has been calculated and compared with the actual codelength.

4.1 Future Work

We can improve the project in the future development by implementing an Embedded Zero-trees of Wavelet transforms algorithm[10]. It is a quantization and coding method that considers the transformed coefficients in the wavelet decomposition part. The coefficients are regarded as a tree and the lowest frequency coefficients are at the root node. As there is less energy in the high frequency subband, it is prone to consist entirely zero coefficients when the tree extends to next higher frequency subband, as called zero-tree. If it is a zero-tree, which means the coefficients are insignificant, then we don't allocate any bits to encode them. Concretely, We can set a threshold to choose those significant coefficients. There are many advantages of using EZW algorithm. For example, EZW provides a better performance in preserving the image detail. Also it can stop and provide the reconstructed image in any steps so it is suitable for image progressive transmission.

References

[1] K. Sayood, *Introduction to data compression*. Morgan Kaufmann, 2017.

[2] "Real-time wavelet decomposition and reconstruction for ecg feature extraction." https://dsp.stackexchange.com/questions/45718/real-time-wavelet-decomposition-and-reconstruction-for-ecg-feature-ext

[3] Wikipedia contributors, "Data compression — Wikipedia, the free encyclopedia." https://en.wikipedia.org/w/index.php?title=Data_compression&oldid=951261400, 2020. [Online; accessed 9-May-2020].

[4] A. Skodras, C. Christopoulos, and T. Ebrahimi, "The jpeg 2000 still image compression standard," *IEEE Signal processing magazine*, vol. 18, no. 5, pp. 36–58, 2001.

[5] J. N. Bradley, C. M. Brislawn, and T. Hopper, "Fbi wavelet/scalar quantization standard for gray-scale fingerprint image compression," in *Visual Information Processing II*, vol. 1961, pp. 293–304, International Society for Optics and Photonics, 1993.

[6] "Project guidelines and requirements." https://cn.inside.dtu.dk/cnnet/filesharing/download/7c7c738a-b089-4efc-8952-ad17a877192al.

[7] "Wavelet tutorial - part 1." http://users.rowan.edu/~polikar/WTpart1.html.

[8] Wikipedia contributors, "Peak signal-to-noise ratio — Wikipedia, the free encyclopedia." https://en.wikipedia.org/w/index.php?title=Peak_signal-to-noise_ratio&oldid=950746371, 2020. [Online; accessed 15-May-2020].

[9] "Lecture 7." https://cn.inside.dtu.dk/cnnet/filesharing/download/7c7c738a-b089-4efc-8952-ad17a877192a.

[10] J. M. Shapiro, "Embedded image coding using zerotrees of wavelet coefficients," *IEEE Transactions on signal processing*, vol. 41, no. 12, pp. 3445–3462, 1993.

5 Appendix

5.1 The Extra Test Results

Figure 14: Quality comparison of original and reconstructed "Lenna" picture

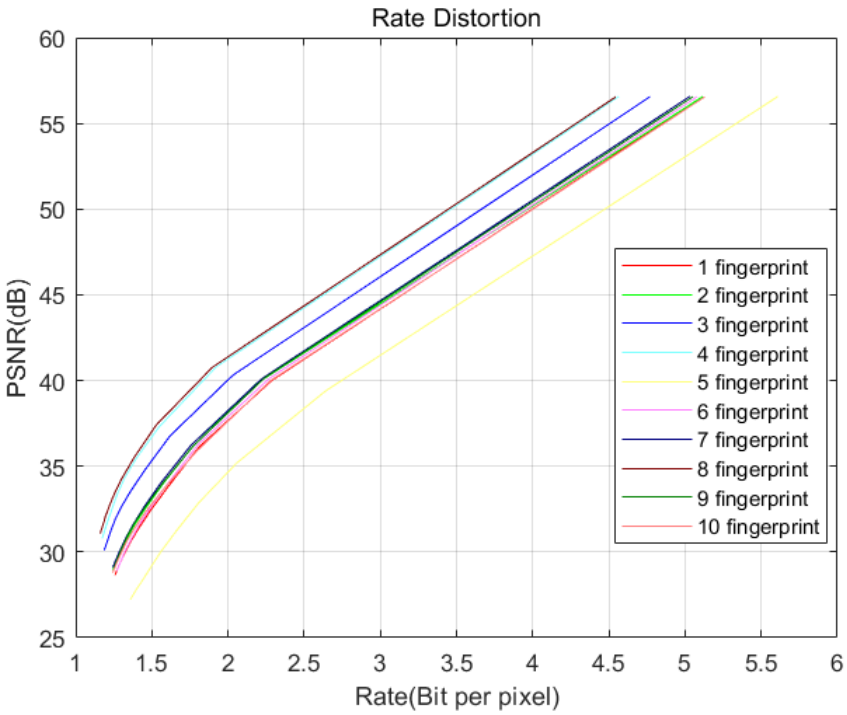


Figure 15: Rate distortion(10 test images)

5.2 Matlab Code of related blocks

```
dwtmode('per');
filter = 'bior3.5';
[lowD,highD] = wfilters(filter);
[lowR,highR] = wfilters(filter);
fingerprint = double(imread('2.png'));
im = fingerprint;
output = [];
f=[]
U=[];
H=[]
Filesize=[]
BPP=[]
Codelength=[]
AL=[]
Ie=[]
%three-level decomposition
 for iter = 1:3

     [c,s] = wavedec2(fingerprint,iter,lowD,highD);
     clear A1 H1 V1 D1
```

```
     [H1,V1,D1] = detcoef2('all',c,s,iter);
     A1 = appcoef2(c,s,'bior3.5',iter);
     im = A1;
     output = [H1(:)' V1(:)' D1(:)' output];
 end

for iter = iter:-1:1
    right = size(fingerprint,2)/2^iter;
    bot = size(fingerprint,1)/2^iter;
    im(bot+1:bot*2,1:right) = reshape(output(1:(bot*right)),bot,right);
    output(1:(bot*right)) = [];
    im(1:bot,right+1:right*2) = reshape(output(1:(bot*right)),bot,right);
    output(1:(bot*right)) = [];
    im(bot+1:bot*2,right+1:right*2) = reshape(output(1:(bot*right)),bot,
        right);
    output(1:(bot*right)) = [];
end
%Quantization
Q = 1:10:100;
k=1
for k = 1:10
    imq(:,:,k) = round(im/Q(k))*Q(k);
%Dequantization
    imd = round(imq/Q(k))*Q(k);
    fig=imd(:,:,k)



% sum all the frequencies
% imhist(fig)
    imd = imq(:,:,k); imd = imd(:);
    pixelValue = unique(imd);
% calculate the frequency of each pixel
    probability = hist(imd,unique(imd)) / length(imd);

% create a dictionary
    dict = huffmandict(pixelValue,probability);

%calculate length
    U=[];
    huft=0
    for i=1:size(dict)
        u=length(cell2mat(dict(i,2)))
```

```
64          huft=huft+u
65          U=[U,u]
66      end
67
68  % get the image pixels in 1D array
69      imageOneD = fig(:) ;
70
71  % encoding
72      testVal = imageOneD ;
73      encodedVal = huffmanenco(testVal,dict);
74
75  % decoding
76      decodedVal = huffmandeco(encodedVal,dict);
77
78  %bit per pixel
79      filesize=numel(encodedVal)
80      Filesize=[Filesize,filesize]
81      Bpp= filesize/numel(fig);
82      BPP=[BPP,Bpp]
83  %Entropy estimation
84      [symbol,c]=size(pixelValue)
85      probability1= (hist(imd,unique(imd))+1)/ (length(imd)+ 1*symbol)%(MAP of
            the probability)
86      ie=-mean(log2(probability1))%self-entropy
87      Ie=[Ie,ie]
88      h = -sum(probability1.*log2(probability1)); %entropy
89      H=[H,h]
90      codelength=(sum(sort(U,'descend').*sort(probability1)))*symbol+huft %(
            average codelength(bit/symbol)*symbol)
91      Codelength=[Codelength,codelength]
92      L=-sum(log2(probability1))%adaptive codelength
93      EL=(-sum(log2(probability1)))/length(probability)
94      AL=[AL,EL]
95  %PSNR
96      MSE=immse(imd(:,:,k),fingerprint)
97      F=10*log10((2^8-1)^2/MSE)
98      f=[f,F]
99
100 %reconstruction
101     for iter = 3:-1:1
102         clear lowC highC ll hl lh hh
103         right = size(fingerprint,2)/2^iter;
104         bot = size(fingerprint,1)/2^iter;
105
```

```
106         ll = imd(1:bot,1:right,k);
107         lh = imd(1:bot,right+1:right*2,k);
108         hl = imd(bot+1:bot*2,1:right,k);
109         hh = imd(bot+1:bot*2,right+1:right*2,k);
110     for i = 1:size(ll,2)
111         lowC(:,i) = idwt(ll(:,i),hl(:,i),filter);
112         highC(:,i) = idwt(lh(:,i),hh(:,i),filter);
113     end
114     for i = 1:size(lowC,1)
115         imd(i,1:right*2,k) = idwt(lowC(i,:),highC(i,:),filter);
116     end
117     end
118 end
119 
120 %plot PSNR with different filter
121 % load 'bior.mat'
122 % load 'db4.mat'
123 % bior=mean(bitrate)
124 % db4=mean(bitrate1)
125 
126 load 'psnr.mat'
127 load 'psnrbior.mat'
128 db4=mean(g)
129 bior=mean(p)
130 x=[1 10 20 30 40 50 60 70 80 90 ]
131 plot(x,db4,'r')
132 hold on
133 plot(x,bior,'b')
134 
135 
136 %plot rate distortion
137 load 'psnrbior.mat'
138 load 'db4.mat'
139 [row,col] = size(g);
140 color=[1 0 0;0 1 0;0 0 1;0.5 1 1;1 1 0.5;1 0.5 1; 0 0 0.5; 0.5 0 0;0 0.5 0;1
        0.5 0.5; 0.5 1 0.5;0.5 0.5 1;1 1 0;0 1 1;1 0 1];
141 
142 for j=1:1:10
143 
144     plot(sort(bitrate(j,:)),sort(p(j,:)),'color',color(j,:));
145 %     plot(sort(bitrate(j,:)),sort(p(j,:)),'o');
146 %     h2=plot(sort(g(j,:)),sort(bitrate1(j,:)),'o');
147      leg_str{j}=[num2str(j),' fingerprint'];
148     hold on
```

```
149 end
150  legend(leg_str)
```