Méthodes hybrides pour la compression d'image
Hybrid Methods for Image Compression

par

Xin WANG

Mémoire présenté au Département de mathématiques
en vue de l'obtention du grade de maître ès sciences (M.Sc.)

FACULTÉ DES SCIENCES
UNIVERSITÉ DE SHERBROOKE

Sherbrooke, Québec, Canada, décembre 2020

Le 16 décembre 2020

Le jury a accepté le mémoire de Madame Xin Wang dans sa version finale.

Membres du jury

Professeur Tomasz Kaczynski
Directeur de recherche
Département de mathématiques

Professeur Madjid Allili
Codirecteur de recherche
Département de mathématiques

Professeur Mohammed Ayoub Alaoui Mhamdi
Évaluateur externe à l'Université
Département d'informatique
Université Bishop's

Professeur Shiping Liu
Président-rapporteur
Département de mathématiques

Abstract

The storage and transmission of images is the basis of digital electronic communication. In order to communicate a maximum amount of information in a given period of time, one needs to look for efficient ways to represent the information communicated. Designing optimal representations is the subject of data compression.

In this work, the compression methods consist of two steps in general, which are encoding and decoding. During encoding, one expresses the image by less data than the original and stores the data information; during decoding, one decodes the compressed data to show the decompressed image.

In Chapter 1, we review some basic compression methods which are important in understanding the concepts of encoding and information theory as tools to build compression models and measure their efficiency. Further on, we focus on transform methods for compression, particularly we discuss in details Discrete Cosine Transform (DCT) and Discrete Wavelet Transform (DWT). We also analyse the hybrid method which combines DCT and DWT together to compress image data. For the sake of comparison, we discuss another total different method which is fractal image compression that compresses image data by taking advantage of self-similarity of images. We propose the hybrid method of fractal image compression and DCT based on their characteristic. Several experimental results are provided to show the outcome of the comparison between

the discussed methods. This allows us to conclude that the hybrid method performs more efficiently and offers a relatively good quality of compressed image than some particular methods, but also there is some improvement can be made in the future.

SOMMAIRE

Le stockage et la transmission d'images sont à la base de la communication électronique numérique. Afin de communiquer un maximum d'informations dans un laps de temps donné, il faut rechercher des moyens efficaces de représenter les informations communiquées. L'objectif de base de la compression de données est la conception d'algorithmes qui permettent des représentations optimales des données.

Dans ce travail, les méthodes de compression consistent en deux étapes en général, qui sont l'encodage et le décodage. Lors du codage, on exprime l'image par moins de données que l'image originale et stocke les informations obtenues; lors du décodage, on décode les données compressées pour montrer l'image décompressée.

Dans le chapitre 1, nous passons en revue quelques méthodes de compression de base qui sont importantes pour comprendre les concepts d'encodage et de théorie de l'information en tant qu'outils pour construire des modèles de compression et mesurer leur efficacité. Plus loin, nous nous concentrons sur les méthodes de transformation pour la compression, en particulier nous discutons en détail des méthodes de transformée en cosinus discrète (DCT) et Transformée en ondelettes discrète (DWT). Nous analysons également la méthode hybride qui combine DCT et DWT pour compresser les données d'image. À des fins de comparaison, nous discutons d'une autre méthode totalement différente qui est la compression d'image fractale qui comprime les données d'image en tirant partie de

l'autosimilarité des images. Nous proposons la méthode hybride de compression d'image fractale et DCT en fonction de leurs caractéristiques. Plusieurs résultats expérimentaux sont fournis pour montrer le résultat de la comparaison entre les méthodes discutées. Cela nous permet de conclure que la méthode hybride fonctionne plus efficacement et offre une qualité d'image compressée relativement meilleure que certaines méthodes, mais il y a aussi des améliorations qui peuvent être apportées à l'avenir.

Acknowledgements

First, I would like to express my sincere gratitude to my supervisors Professor Tomasz Kaczynski and Professor Madjid Allili for their continuous support of my studies and related research, for their patience, motivation, and immense knowledge. I am very grateful for the opportunity to complete my Master program.

I would like to thank all the colleagues and professors for sharing knowledge during my study. I would like to thank our university for providing the good environment.

Last but not the least, I would like to thank my family and friends for supporting me.

Xin Wang

Sherbrooke, November 2020

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

INTRODUCTION

Nowadays, the Internet and mobile communication lead to a huge demand for storing, transmitting, and receiving data. Data compression is the most effective way to save on storage space and network bandwidth. The compression removes duplicates of data sets and then decodes back to the original data set as needed by applying various algorithms. Therefore, it can help us to reduce the need for new hardware, improve database performance, speed up backups, and provide more secure storage. As images are constructed of pixels, which means images are constructed by a bunch of data, we will talk about image compression in this thesis.

The main purpose of image compression is to reduce the redundancy and irrelevancy present in the image so that it can be stored and transferred more efficiently. As we know, an image is composed of pixels and if the image is grey level, then it can be completely represented by encoding the intensity and position of each pixel. How to reduce the storage space of the image?

Here is an example of a naive approach (this approach can't solve the problem of image compression in general). First, we can represent the intensity of each pixel in the previous order in the coordinate plane; secondly, fit these points with Fourier series [1, page 203] to interpolate their values; finally, these points can be represented approximately by a function consisting of sine and cosine components.

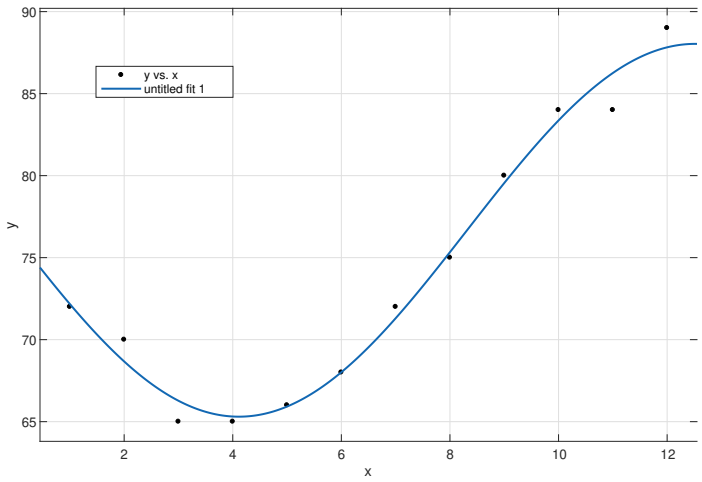


Figure 1: Points fitted by Fourier Series

Here is an example shown in Fig1. Suppose that we have the intensity of 12 pixels, then we represent them in the coordinate plane, to simplify the representation we can apply Fourier Series with the general function

$$f(x) = a_0 + a_1 \cos(x\omega) + b_1 \sin(x\omega).$$

As a result, we can get $a_0 = 76.67, a_1 = -0.3744, b_1 = -11.36, \omega = 0.3739$. Originally, we need to use 12 numbers for storage, but now we only need four numbers. In this thesis, we will introduce the transform method to do the compression. Generally, transform means to map the original image in space domain to the expected domain such as frequency domain and wavelet domain by a specific kernel. This process can be seen as decorrelation. Therefore, the purpose of reducing data can be achieved partly. We can store or transmit the compressed data. When reopening the image from the compressed data, we need to decode by the inverse process.

This thesis is divided as follows: Chapter 1 introduces the fundamentals of image compression including the concept of information theory, some basic compression methods and some standards for measuring the effectiveness of the compression. Chapter 2 explains Discrete Cosine Transform (DCT) algorithm and Discrete Wavelet Transform

(DWT) algorithm. The hybrid method of DCT and DWT algorithm and the hybrid method of fractal image compression and DCT explained in Chapter 3. We compare the efficiency of several image compression methods in Chapter 4. At the end, we give the conclusion.

CHAPTER 1

Fundamentals of Image Compression

Data compression may be viewed as a branch of information theory in which the primary objective is to minimize the amount of data to be represented [3]. Information theory is the mathematical treatment of the concepts, parameters and rules governing the transmission of messages through communication systems. It is devoted to discover mathematical laws that govern the behavior of data as it is transferred, stored, or retrieved. [4]

More specifically, a characterization of data compression is that it involves transforming a string of characters in some representation (such as ASCII) into a new string (of bits, for example) which contains the same information but whose length is as small as possible. Thus, a certain amount of information can be represented by different amounts of bits.

1.1 Basic Notions

1.1.1 What is encoding

The way we achieve compression is through computer encoding, so we start from explaining what is encoding. Encoding is the process of converting data from one form to another. There are several types of encoding, including image encoding, audio and video encoding, and character encoding.

In this thesis we are concerned about image encoding, which is an encoder that writes image data to a stream. Encoders can compress, encrypt, and alter the image pixels in a number of ways prior to writing them to the stream. For example, a BMP format image that is converted to a JPEG format image maybe 1/16 the size of the original BMP image [2].

A code is a mapping of source messages into codewords. The source messages are the basic units into which the string to be represented is partitioned. [3, page 4] For example, we can suppose the source message is "a", and its corresponding codeword is "000".

1.1.2 Image Compression Models

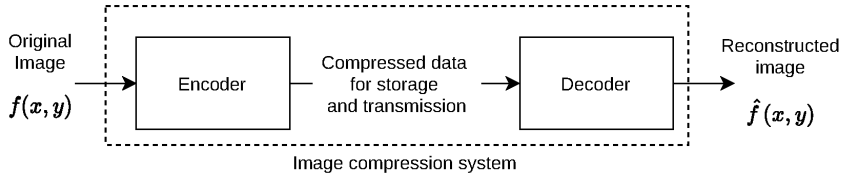


Figure 1.1: Image compression system in general

Now we will explain more concretely the process of image compression which is a branch of data compression. As Fig.1.1 shows, an image compression system is composed of two

distinct functional components: an encoder and a decoder. Note that when we reflect on a stuck of data to decipher its core meaning, we are decoding; when we determine its appropriate code and label it, we are encoding [5]. The encoder performs compression, and the decoder performs the complementary operation of decompression. We use codec which is a device or program to process the encoding and decoding. Here is the process of how the system works.

1. Input image $f(x, y)$ is fed into the encoder, which creates a compressed representation of the input. This representation will be stored for later use.

2. When the compressed representation is presented to its complementary decoder, a reconstructed output image $\hat{f}(x, y)$ is generated.

The function $\hat{f}(x, y)$ may or may not be an exact replica of $f(x, y)$. If it is, the compression system is called error free, lossless, or information preserving. If not, the reconstructed output image is distorted and the compression system is referred to as lossy. From Fig.1.2, we can see encoding and decoding process in details.

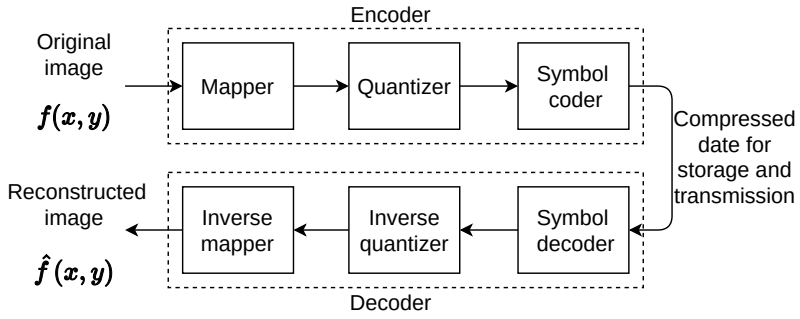


Figure 1.2: Image compression system in detail

For encoding, a mapper transforms $f(x, y)$ into a format which is designed to reduce spatial and temporal redundancy. Then the quantizer processes the transformed data, to keep irrelevant information out of the compressed representation by dividing a

quantization table. The quantization step is lossy and irreversible. The final step is symbol coding which assigns a sequence of bits called a codeword, to each level produced by the quantizer. After encoding, we get a bitstream consisting of the compressed data. The bitstream can be used for storage and transmission of image data, but it is not usually used to visualize the data. The decoding process allows to process the bitstream in order to recover the original information. It is the inverse of encoding.

1.1.3 Redundant data

To represent the information with less amount of data, we need to figure out which kind of data can be reduced in the representation, which is called redundant data. Representations that contain irrelevant or repeated information are said to contain redundant data which is what we don't want to represent. There are three principal types of data redundancies which can be identified and exploited that two-dimensional intensity arrays suffer from:

1. Coding redundancy

   Each piece of information or event is assigned a code word. The number of symbols in each code word is its length. If the gray levels of an image are coded in a way that uses more code symbols than absolutely necessary to represent each gray level, then the resulting image is said to contain coding redundancy.

   Typically, we use 8-bits codes to represent pixel intensities which often require less than 8-bits to be represented.

2. Spatial redundancy

   In most situations, the pixels in 2-D intensity arrays are correlated spatially, which means each pixel is similar to or dependent on neighboring pixels, that leads to unnecessary replication.

3. Irrelevant information

   Most 2-D intensity arrays contain information which is ignored by the human visual system or which is not correlated to the usage.

The important question now is to determine if we can represent image data with smaller amount of data without losing relevant information. Information theory provides the mathematical framework to answer this question.

1.1.4 Self-Information

In information theory, self-information of a random variable or signal is the amount of information gained when it is received. Given a random event E, we can say that the amount of self-information it contains is:

$$I(E) = \log \frac{1}{P(E)} = -\log P(E).$$

For example, if $P(E) = 1$, then $I(E) = 0$, which means the event E doesn't have any uncertainty, so when it happens, 0 bit of information have been received.

The unit of self-information is depended on the logarithmic base. If the base 2 is selected, the unit of information is bit which we will stick with.

A zero-memory information source S is a source that emits source symbols from an alphabet $\{s_{1,2}, \ldots, s_k\}$ with probabilities $\{p_1, p_2, \ldots, p_k\}$ respectively, where the symbols emitted are statistically independent [6].

For the zero-memory information source S, what is the average amount of information in observing the output of it? We can define the entropy of the source based on self-information, which represents the average information per symbol of a discrete set of events [6]:

$$H(S) = \sum_{j=1}^{n} p(s_j) \cdot I(s_j) = -\sum_{j=1}^{n} p(s_j) \lg p(s_j)$$

where lg denotes the base 2 logarithm, s_j in this equation is called source symbol, it's like the event E.

In conclusion, self-information denotes the uncertainty of an event, while entropy denotes the average self-information of a set of events. We can use "entropy" to measure the least amount of data that we need use to store an image.

1.2 Classification of Methods

Image compression techniques fall into two categories: lossless or lossy image compression. Choosing which of these two categories depends on the application and on the compression degree required.

1.2.1 Lossless compression

Lossless compression is a class of data compression algorithms that allows the original data to be perfectly reconstructed from the compressed data. It is used where it is important that the original and the decompressed data to be identical, or where deviations from the original data would be unfavorable. Typical examples are executable programs, text documents, and source code. Some image file formats, like PNG and BMP, use only lossless compression, while others like JPEG and GIF formats are lossy.

Lossless compression techniques [7]

Most lossless compression programs do two things in sequence:

1. generate a statistical model for the input data;

2. use this model to map input data into bit sequences in such a way that the frequently encountered data will produce shorter output than infrequently encountered data.

The primary encoding algorithms used to produce bit sequences are Huffman coding and arithmetic coding. Arithmetic coding achieves compression rates close to the best possible for a particular statistical model, which is given by the information entropy. Whereas Huffman coding is simpler and faster but produces poor results for models that deal with symbol probabilities close to 1.

Run length coding (RLC)

Using this encoding method, we can replace consecutive characters that appear repeatedly in a character string with two bytes, where the first byte represents the number of repetitions and the second byte represents the repeated character string. For example, (3, 7) represents the string "777"

Huffman coding

Huffman coding is an algorithm developed by David A. Huffman [8]. In computer science and information theory, a Huffman code is a particular type of optimal prefix code that is commonly used for lossless data compression.

By definition, a prefix code is a uniquely decodable code with the prefix property, which requires that no code word is a proper prefix of any other code words. For example, a code with code words $\{2, 11\}$ has the prefix property; a code consisting of $\{2, 11, 21\}$ does not, because "2" is a prefix of "21".

Here is the process of Huffman coding:

1. Build a priority queue including all unique characters organized by weight (frequency).
2. Remove two nodes with the smallest weight in the queue, w_i and w_j (usually we put the one with smaller weight on the left), merge them into an internal node whose weight is $w_i + w_j$.
3. Insert the internal node with weight $w_i + w_j$ into the queue.
4. Repeat steps 2-3 until the queue contains one node which is the root node.
5. Encoding.
   (a) Give all left links 0 and right links 1 in the Huffman tree.
   (b) Record the codes of all letters in order from the root to the leaves

Example

There is a stream of data including 100 symbols, and the probability of each symbol is shown in Tab.1.1.

Symbol	f	e	c	d	b	a
Frequency	0.04	0.06	0.1	0.1	0.3	0.4

Table 1.1: The initial priority queue

Symbol	a	b	c	d	e	f
Code	1	01	0000	0001	0011	0010

Table 1.2: Original source and codes after Huffman Coding

1. Direct coding: if we use ASCII codes, each symbol will be represented by 8 bits (for example, "a" will be represented as "01100001"), the result will be $100 \times 8 = 800 bits$

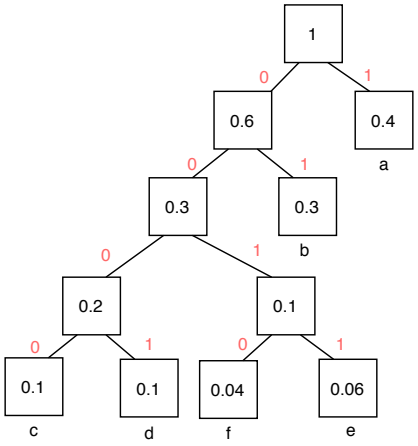


Figure 1.3: The process of Huffman coding

2. Huffman coding: after the process of Huffman coding shown in Fig.1.3, we can get the code for each symbol shown in Tab.1.1. The result will be $40 \times 1 + 30 \times 2 + 10 \times 4 + 10 \times 4 + 6 \times 4 + 4 \times 4 = 220$bits

In conclusion, Huffman coding allocates shorter codes to the symbols with higher probabilities and longer codes to the symbols with lower probabilities to reduce the length of total codes.

Arithmetic coding

The method of arithmetic coding was suggested by Elias, and presented by Abramson on Information Theory and coding [9].

In arithmetic coding, a source ensemble is represented by an interval between 0 and 1 on the real number line. Each symbol of the ensemble narrows this interval depending on their frequencies. As the interval becomes smaller, the number of bits needed to specify it grows.

Here is an example that illustrates the idea of arithmetic coding.

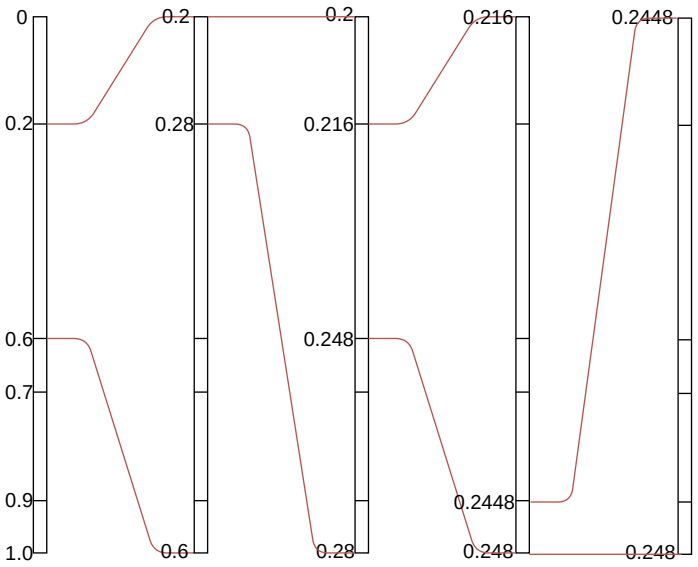


Figure 1.4: The process of Arithmetic coding

Given source messages $\{A, B, C, D, E\}$ with probabilities $\{0.2, 0.4, 0.1, 0.2, 0.1\}$. In the interval $[0, 1)$, A corresponds to $[0, 0.2)$, B corresponds to $[0.2, 0.6)$, C corresponds to $[0.6, 0.7)$, D corresponds to $[0.7, 0.9)$ and E corresponds to $[0.9, 1.0)$.
To represent the ensemble $BABE$, B narrows the interval from $[0, 1)$ to $[0.2, 0.6)$. A narrows the interval from $[0.2, 0.6)$ to $[0.2, 0.28)$. The second B narrows the interval from $[0.2, 0.28)$ to $[0.216, 0.248)$. E yields a final interval of $[0.2448, 0.248)$. The final interval or any number within the interval may be used to represent the ensemble.

Lempel-Ziv-Welch

Lempel-Ziv-Welch (LZW) [10] is a universal lossless data compression algorithm created by Abraham Lempel, Jacob Ziv, and Terry Welch. It was published by Welch in 1984 as an improved implementation of the LZW algorithm published by Lempel and Ziv in 1978. The algorithm is simple to implement and has the potential for very high throughput in hardware implementations. It is the algorithm that is widely used in Unix file compression

utility and is also used in the GIF image format. It is also used in the dictionary coder which is operated by searching for matches between the text to be compressed and a set of strings contained in a data structure maintained by the encoder.

- Encoding

Step 1 Initialize the dictionary to contain all strings of length one.

Step 2 The string encoding table (dictionary) will gradually expand as the strings are inputted.

Here is an example. The plain text to be encoded is *ababccab*#.

Symbol	Decimal
#	0
a	1
b	2
c	3

Step 1

Current Sequence	Output Code	Next Char	Extended Dictionary
a	1	b	4: ab
b	2	a	5: ba
ab	4	c	6: abc
c	3	c	7: cc
c	3	a	8: ca
ab	4	#	9: ab#
#	0		

Step 2

Figure 1.5: The process of LZW encoding

As Fig.1.5 shows, buffer input characters in a sequence ω until ω+next character is not in the dictionary. Emit the code for ω, and add ω+next character to the dictionary. Start buffering again with the next character. Thus, the encoded data is $1, 2, 4, 3, 3, 4, 0$.

- Decoding

For decoding, we just look up the dictionary to find out the corresponding characters for the encoded data. The first code is 1, so we look it up in the dictionary which

is the table in Fig.1.5, and output a. The second code is 2 which corresponds to b in the dictionary. 4 corresponds to ab in the extended dictionary. Then read the next code, find the responding sequence in the dictionary. We can get the original plain text by repeating the steps above.

1.2.2 Lossy methods

By contrast, lossy compression permits reconstruction only of an approximation of the original data, though usually with greatly improved compression ratios and therefore reduced images sizes. Thus, repeatedly compressing and decompressing an image results in a poor quality of the image. An advantage of this technique is that it allows for a higher compression ratio than the lossless.

A commonly used method of lossy compression for digital images is JPEG [11] based on Discrete cosine transform. The degree of compression can be adjusted, allowing a selectable tradeoff between storage size and image quality. JPEG typically achieves 10:1 compression ratios with little perceptible loss in image quality. Another lossy compression method is JPEG 2000 (JP2) [12] which is based on Wavelet compression, and it was developed from 1997 to 2000 by a Joint Photographic Experts Group committee. The third lossy compression method we will introduce is Fractal compression [13] based on fractals is best suited for textures and natural images, relying on the fact of similarities among different parts in an image. We will talk about these methods more in detail later.

Lossy compression techniques

There are two basic types of lossy compression methods.

1. Transform coding method

In lossy transform codecs, samples of picture or sound are taken, chopped into small segments, transformed into a new basis space, and quantized. The resulting quantized values are then entropy coded.

2. Predictive coding method

   In lossy predictive codecs, previous and/or subsequent decoded data is used to predict the current sound sample or image frame. The error between the predicted data and the real data, together with any extra information needed to reproduce the prediction, is then quantized and coded.

In some systems, the two techniques are combined, with transform codecs being used to compress the error signals generated by the predictive stage.

1.3 Performance Measures

There are some standards to measure the effectiveness of the compression of the image.

1.3.1 Compression Ratio

Data compression ratio, also known as compression power, is a measurement of the relative reduction in the size of data representation produced by a data compression algorithm. It is typically expressed as the following formula

$$Compression \quad Ratio = \frac{Uncompressed \quad Data \quad Bits}{Compressed \quad Data \quad Bits} \quad .$$

Here the size of uncompressed data and compressed data both refer to the storage size.

For example, a representation that compresses a file's storage size from 10 bits of data to 2 bits has a compression ratio of $10/2 = 5$, often notated as an explicit ratio, 5:1 (read "five" to "one"), or as an implicit ratio, 5/1.

1.3.2 Fidelity and Quality

We notice that lossy image compression will lead to different qualities of images, so a means of measuring the loss is needed. There are two standards for measuring: subjective fidelity criteria and objective fidelity criteria.

Subjective Fidelity Criteria

When information loss can be expressed as a mathematical function of the input and output of a compression process, it is said to be based on an objective fidelity criterion. Here are several symbols that can measure the objective fidelity criterion.

- Mean squared error (MSE)

  We can use the mean squared error (MSE) to measure the difference between the original image and the reconstructed image

  $$MSE = \left[\frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} [\hat{f}(x,y) - f(x,y)]^2 \right].$$

  Above, $f(x,y)$ is an input image and $\hat{f}(x,y)$ is the compressed image, the error $e(x,y)$ between $f(x,y)$ and $\hat{f}(x,y)$ is $e(x,y) = \hat{f}(x,y) - f(x,y)$.

- Peak signal-to-noise ratio (PSNR)

  Peak signal-to-noise ratio, often abbreviated PSNR, is an engineering term for the ratio between the maximum possible power of a signal and the power of corrupting noise that affects the fidelity of its representation. Because many signals have a

very wide dynamic range, PSNR is usually expressed in terms of the logarithmic decibel scale

$$PSNR = 10 \cdot \log_{10} \left(\frac{L^2}{MSE} \right).$$

Here, L is the maximum possible pixel value of the image which is 255 that we consider in this paper. When the value of PSNR is higher than 30dB, it will be difficult to find the difference by human subjective feeling. The typical peak signal-to-noise ratio in image compression is between 30 and 40dB, the higher the more approximate to the original picture.

Objective Fidelity Criteria

Objective fidelity criteria is executed by showing the decompressed image to the observers and then averaging their evaluation. The observers can give the evaluation from a level list such as much worse, worse, slightly worse, the same, slightly better, better, much better; or by comparing different degrees of decompressed images with the input one.

1.3.3 Time complexity

As we wish to transmit the images with relatively high efficiency, the time complexity is also a factor that we need to measure during image compression. In computer science, the time complexity is the computational cost that describes the amount of computations (number of executed steps) required to run an algorithm as a function of the size of the input. It is commonly expressed using big O notation, typically $O(n), O(n \log n), O(n^\alpha), O(2^n)$, etc., where n is the input size in units of bits needed to represent the input. We recall that a computational cost $f(n)$ is in $O(g(n))$ if $f(n)$ cannot grow faster than $g(n)$ when n is large. Using the big O notation, we can measure the time complexity of the algorithms that described in section 1.2.1.

The time complexity of Huffman algorithm is $O(k \log k)$ where k is the size of the unique characters. Since all the nodes are stored in a priority queue, then extracting node with minimum frequencies has a computational cost of $O(\log k)$, and it happens $2 \times (k-1)$ times [14].

The time complexity of Arithmetic algorithm is $O(n)$ where n is the size of input data. Since for each input character, we implement the algorithm one time to achieve a result of a more precise value based on the frequency of the characters. Arithmetic code achieves optimal compression rate [18]. Experimental results showed that the compression ratio of the arithmetic coding for text files is better than Huffman coding, while the performance of the Huffman coding is better than Arithmetic coding [15], as we can analyse that with the information of frequency, the representation of each symbol is fixed using Huffman coding, at the same time, we need to continuously process the presentation using Arithmetic coding. However, the shortage of both these two methods is that they can only compress data statically which means that they require information about the input stream of data prior to the compression process.

The time complexity of LZW algorithm is $O(n)$ which is linearly dependent on the input message size [17]. Because we process the same implementation: check if the string exists in the dictionary, output the corresponding code of the string, and put the combination of the string with next character into the dictionary table. LZW algorithms can provide faster execution than Huffman algorithm. Moreover, LZW algorithm doesn't require prior information about the input stream. We need to notice that for the usage of this algorithm, the efficiency of it increases as the number of long, repetitive words in the input data increases because it is a dictionary method [16].

CHAPTER 2

Compression Using Transforms

In Chapter 1 Section 1.2.2, we mentioned the lossy compression techniques, one of which is transform coding method. In this chapter, we will introduce some of the transform coding methods more specifically.

In image processing, an image transform means to convert the image from one to another, for example, we can transform an image from spatial domain representation to frequency domain representation or vice versa. There are two objectives of the transformation process:

1. to decorrelate the pixels of the image, which then lead to higher efficiency than the coding of the original image;

2. to pack as much information as possible into the smallest amount of transform coefficients, and for the other coefficients, we can do rough quantification or make them be 0, provided that the image is not distorted.

Transforms often applied are Discrete Fourier Transform (DFT), Karhunen-Loeve Transform (KLT), Walsh-Hadamard Transform (WHT), Discrete Cosine Transform

(DCT), Discrete Wavelet Transform (DWT) and so on. Their common point is to use the transformation kernel to do the transform and the inverse transform kernel to do the inverse transform.

For image $f(x, y)$ of size $N \times N$, the discrete transform $T(u, v)$ can be presented as

$$T(u,v) = \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x,y) r(x,y,u,v), \quad u, v = 0, 1, 2, \ldots, N-1.$$

Reciprocally, given $T(u, v)$, $f(x, y)$ can be obtained by the inverse transform

$$f(x,y) = \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} T(u,v) s(x,y,u,v), \quad x, y = 0, 1, 2, \ldots, N-1.$$

$r(x, y, u, v)$ and $s(x, y, u, v)$ are called forward and inverse transform kernel functions respectively, and the essence of different transformation methods is that the transformation kernels are different.

Thus, the main idea of the transforms is to express the signals on different bases or kernels to simplify the data. We will analyze in detail about the different bases used in different transforms.

The transform methods we mainly discuss in this thesis are discrete cosine transform and discrete wavelet transform. After the execution of discrete cosine transform, the energy of the image will concentrate to the upper left corner shown in Fig.2.1, then we set a threshold and set the values less than the threshold to be zero, so that now the matrix can be recorded with fewer code words than before. When reading the compressed data, we need to decode the recorded matrix. We can compare the original image and the processed image in Fig.2.2.

Having some knowledge of transform, we can get to know about the coding process based on transform. Here are the general steps:

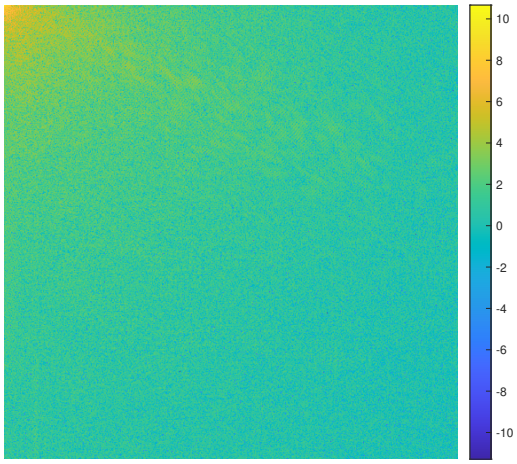


Figure 2.1: The transformed image using a logarithmic scale

Figure 2.2: Original Image (Left) and Processed Image (Right)

1. The original image undergoes a certain transformation to obtain the transformation coefficients.

   The transformation is always lossless to the image information which means it allows the original data to be perfectly reconstructed from the transformed data.

2. Quantize the transform coefficients, and output the index symbol stream of the quantization interval.

   Quantization, in general, is the process of constraining an input from a continuous or otherwise large set of values (such as the real numbers) to a discrete set (such as

the integers). The quantization operation is irreversible, so it will cause information loss. If the influence of noise is not considered, all the information loss occurs in the quantizer.

3. Entropy encodes the quantized symbol stream to obtain a more compact bitstream.

   Entropy coding, such as Huffman coding, LZW coding, and Arithmetic coding, is a lossless compression method based on information theory.

After encoding, the space occupied by the original image will be significantly reduced, then we need to decode to read the compressed data.

2.1 Discrete Cosine Transform (DCT)

Discrete cosine transform is used in a block transform coding system for compression (in block transform coding, a reversible, linear transform is used to map each block or subimage into a set of transform coefficients, which are then quantized and coded) [1, page 566], and we can also choose other transform methods in this system, such as the Walsh-Hadamard transform (WHT) and the discrete Fourier transform (DFT). The reason why we choose to discuss the DCT is that it has a stronger ability to carry information than WHT and DFT, and is easier to execute than KLT.

2.1.1 Fourier Series

The DCT comes from the concepts of Fourier Transform and Fourier Series developed by French mathematician and physicist Jean Baptiste Joseph Fourier. Fourier introduced the series for the purpose of solving the heat equation in a metal plate. Through Fourier's

research, the fact was established that an arbitrary continuous periodic function can be represented by a trigonometric series.

A function $f(t)$ of a continuous variable t that is periodic with period, T, can be expressed as the sum of sines and cosines multiplied by appropriate coefficients. This sum, known as a Fourier series [1, page 203], has the form

$$f(t) = \sum_{n=-\infty}^{\infty} c_n e^{i\frac{2\pi n}{T}t}$$

where the expansion coefficient c_n can be calculated as

$$c_n = \frac{1}{T}\int_{-T/2}^{T/2} f(t)e^{-i\frac{2\pi n}{T}t}dt, \quad n = 0, \pm 1, \pm 2, \ldots$$

Now we discuss the basis of Fourier series [20]. The complex exponentials in Fourier series form a basis for $L^2([0,T])$ and they are orthonormal.

Suppose $b_n(t) = e^{i\frac{2\pi n}{T}t}$, then the inner product of $b_n(t)$ and $b_m(t)$ should be

$$\begin{aligned}
\langle b_n, b_m\rangle &= \int_0^T e^{i\frac{2\pi n}{T}t}\overline{e^{i\frac{2\pi m}{T}t}}dt \\
&= \int_0^T e^{i\frac{2\pi n}{T}t}e^{-i\frac{2\pi m}{T}t}dt \\
&= \int_0^T e^{i\frac{2\pi(n-m)}{T}t}dt \\
\text{if } m = n,& \\
&= \int_0^T 1dt \\
&= T,
\end{aligned}$$

if $m \neq n$,

$$\begin{aligned}
\langle b_n, b_m \rangle &= \frac{T}{i2\pi(n-m)} e^{i\frac{2\pi(n-m)}{T}t}\Big|_0^T \\
&= \frac{T}{i2\pi(n-m)}(e^{i2\pi(n-m)} - e^0) \\
&= \frac{T}{i2\pi(n-m)}(1-1) \\
&= 0.
\end{aligned}$$

Therefore, the complex exponentials with period T are orthogonal. The orthogonality allows faster implementations, as there is less redundancy in computation. Furthermore, to get them orthonormal, we can scale them by the coefficient $\frac{1}{\sqrt{T}}$, then the new basis is

$$b_n(t) = \frac{1}{\sqrt{T}} e^{i\frac{2\pi n}{T}t}.$$

Any periodic function f can be expressed as the sum of the new complex exponentials with corresponding expansion coefficients, and we can get the expansion coefficients as the inner or scalar product of f and each basic function b_n,

$$\langle f, b_n \rangle = \frac{1}{\sqrt{T}} \int_0^T f(t) e^{i\frac{2\pi n}{T}t} dt.$$

Now f can be expressed as

$$\begin{aligned}
f(t) &= \sum_{n=-\infty}^{\infty} \langle f, b_n \rangle b_n \\
&= \sum_{n=-\infty}^{\infty} \left(\frac{1}{\sqrt{T}} \int_0^T f(s) e^{i\frac{2\pi n}{T}s} ds \right) \frac{1}{\sqrt{T}} e^{i\frac{2\pi n}{T}t} \\
&= \sum_{n=-\infty}^{\infty} c_n e^{i\frac{2\pi n}{T}t},
\end{aligned}$$

where

$$c_n = \frac{1}{T} \int_0^T f(t) e^{i\frac{2\pi n}{T}t} dt.$$

Since the result is the same as the Fourier series formulas, it follows that the basis b_n is orthonormal in the space $L^2([0,T])$.

That means, any periodic function $f(t)$ can be decomposed into a linear combination of mutually orthogonal elements $b_n(t)$. We can regard the basis of $L^2(2\pi)$ consists of the function $b_1(t) = e^{i\frac{2\pi}{T}t}$ which expands or shrinks by the integer n: $b_n(t) = b_1(nt)$ [21].

2.1.2 Fourier Transform

What happens if the function is not periodic? Is it possible to decompose it in a similar way as a periodic function? In this case, we can consider f as periodic with infinite period, i.e. $T \to \infty$, and $\omega = \frac{2\pi}{T}$, therefore, we can have $\omega \to 0$. Thus, Fourier transform can be derived from Fourier series. Here is the derivation process [22].

$$c_n = \frac{1}{T}\int_{-T/2}^{T/2} f(t)e^{-i\frac{2\pi n}{T}t}dt,$$
$$Tc_n = \int_{-T/2}^{T/2} f(t)e^{-in\omega_0 t}dt.$$

As $T \to \infty$, we can replace the discrete quantity Tc_n by a continuous quantity $F(\omega)$, therefore,

$$F(\omega) = \int_{\mathbb{R}} f(t)e^{-i\omega t}dt. \tag{2.1}$$

Likewise, with the Fourier Series, $f(t)$ can be expressed as

$$\begin{aligned} f(t) &= \sum_{n=-\infty}^{+\infty} c_n e^{in\omega_0 t} \\ &= \sum_{n=-\infty}^{+\infty} Tc_n e^{in\omega_0}\frac{1}{T} \\ &= \sum_{n=-\infty}^{+\infty} Tc_n e^{in\omega_0}\frac{\omega_0}{2\pi}. \end{aligned}$$

Here, we can replace Tc_n by $F(\omega)$ in Eq. (2.1)

$$f(t) = \frac{1}{2\pi} \sum_{n=-\infty}^{+\infty} F(\omega) e^{in\omega_0} \omega_0.$$

As $T \to \infty$, and $\omega_0 \to 0$, also we can regard the last $\omega_0 = \Delta\omega$ therefore

$$f(t) = \frac{1}{2\pi} \int_{\mathbb{R}} F(\omega) e^{i\omega t} d\omega. \tag{2.2}$$

Eq. (2.1) and Eq. (2.2) are Fourier transform and inverse Fourier transform respectively. Together, they make up the Fourier transform pair:

$$F(\omega) = \int_{-\infty}^{+\infty} f(t) e^{-i\omega t} dt,$$
$$f(t) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} F(\omega) e^{i\omega t} d\omega.$$

By performing the transform, it changes the variable from t to ω which denotes frequencies of the signal, therefore, the function $f(t)$ is transformed from the time domain to the frequency domain.

The comparison between the inverse Fourier transform and the Fourier series formulas suggest that $e^{i\frac{2\pi n}{T}t}$ is the analogous discrete version of the basic function $e^{i\omega t}$. But unlike $\{b_n(t) = e^{i\frac{2\pi n}{T}t}; n \in \mathbb{Z}\}$ which is the orthonormal basis of the space $L^2(0, 2\pi)$, $e^{i\omega t}$ can't generate the orthonormal basis of $L^2(\mathbb{R})$, as all the signals in $L^2(\mathbb{R})$ should have finite energy while the energy of $e^{i\omega t}$ is infinite. Therefore, we can infer that Fourier transform can't express the signals in $L^2(\mathbb{R})$ concisely enough, and we will discuss this in more details in the section of wavelet transform.

2.1.3 Discrete Fourier Transform

When we use computer to implement the algorithm, we need to discretize the time and frequency domains where discretization means sampling. We can get Discrete Fourier transform (DFT) by performing Fourier Transform on the sampled function [23].

Suppose $f(t)$ is a continuous signal, and there are M sampled points on $f(t)$, which are denoted as $f[0], f[1], f[2], \ldots, f[k], \ldots, f[M-1]$ (it happens if the data of the sampled points), the time gap between each sampled points is T. The Fourier Transform of $f(t)$ is

$$F(\omega) = \int_{\mathbb{R}} f(t) e^{-i\omega t} dt.$$

As we only consider the sampled points,

$$F(\omega) = \int_{0}^{(M-1)T} f(t) e^{-i\omega t} dt.$$

We could regard each sample $f[k]$ as an impulse

$$\begin{aligned} &= f[0] e^{-i0} + f[1] e^{-i\omega T} + \cdots + f[k] e^{-i\omega kT} + \cdots + f[M-1] e^{-i\omega(M-1)T} \\ &= \sum_{k=0}^{M-1} f[k] e^{-i\omega kT}. \end{aligned}$$

Since there are only a finite number of input signal points, we regard them as periodic in the DFT. For example, $f(0)$ to $f(M-1)$ is the same as $f(M)$ to $f(2M-1)$. As Fig. 2.3 shows, (a) is one period of the periodic sequence in (b). As we treat the sampled points on $f(t)$ as a sequence and periodic, we can consider the sequence by one period and we can evaluate the DFT value of each points by setting

$$\omega = 0, \frac{2\pi}{MT}, \frac{2\pi}{MT} \times 2, \ldots, \frac{2\pi}{MT} \times k, \ldots, \frac{2\pi}{MT} \times (M-1).$$

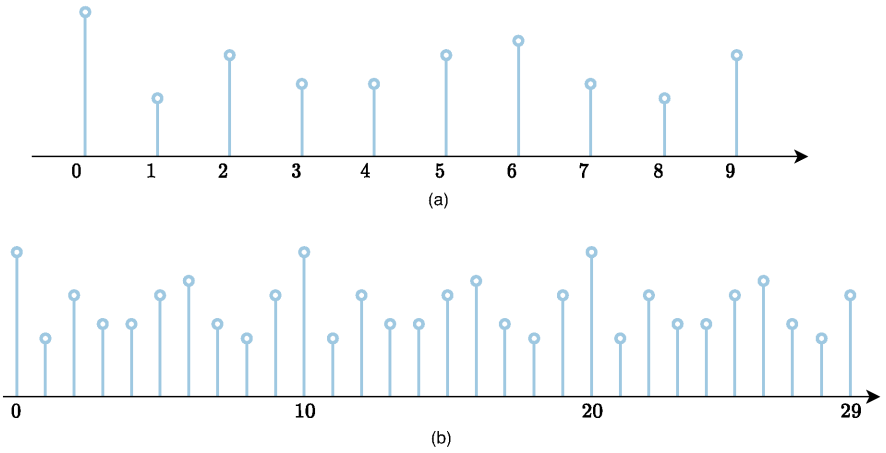


Figure 2.3: (a)Sequence of $M = 10$. (b)Implicit periodicity in DFT.

Thus we can have the DFT equation in general

$$F(\mu) = \sum_{k=0}^{M-1} f(k) e^{-i\frac{2\pi}{M}\mu k}, \qquad \mu = 0, 1, 2, ..., M-1.$$

The inverse DFT is

$$f(k) = \frac{1}{M} \sum_{\mu=0}^{M-1} F(\mu) e^{-i\frac{2\pi}{M}\mu k}, \qquad k = 0, 1, 2, ..., M-1.$$

Specifically, when $\mu = 0$, we have

$$\begin{aligned} F(0) &= \sum_{k=0}^{M-1} f(k) \\ &= M\frac{1}{M}\sum_{k=0}^{M-1} f(k) \\ &= M\overline{f}(k). \end{aligned}$$

$$|F(0)| = M|\overline{f}(k)|. \tag{2.3}$$

From Eq.(2.3), it is very obvious that $|F(0)|$ is the largest component of the spectrum.

As we can see in the equations, DFT transforms a complex signal into its complex spectrum. However, if the signal is real as in most of the applications, half of the data is redundant.

To simplify the operation process and also to have a more effective result, we use the Discrete cosine transform (DCT).

2.1.4 Discrete Cosine Transform

DCT is a real-valued transform that transforms a sequence of real data points into its real spectrum and therefore avoids the problem of redundancy.

Also, as DCT is derived from DFT, all the desirable properties of DFT (such as the fast algorithm) are preserved. From Fourier Transform to DCT, some mathematical theory is needed. The theory states that a continuous real symmetric function that satisfies Dirichlet's condition in a given interval can be expanded into a Fourier series containing only the cosine terms.

We discuss now the derivation process from DFT to DCT [24].

To derive the DCT of an M-point real signal sequence $\{x[0], \cdots, x[M-1]\}$, we first construct a new sequence of $2M$ points:

$$x'[m] \triangleq \begin{cases} x[m] & (0 \leq m \leq M-1) \\ x[-m-1] & (-M \leq m \leq -1). \end{cases}$$

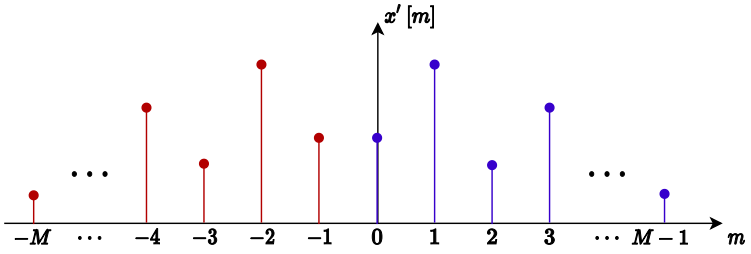


Figure 2.4: The sequence of 2M points

As Fig.2.4 shows, this 2M-point sequence $x'[m]$ is assumed to repeat itself outside the range $-M \leq m \leq M-1$, i.e., it is periodic with period $2M$, and it is even symmetric

with respect to the point at $m = -1/2$:

$$x'[m] = x'[-m-1] = x'[2M-m-1].$$

If we shift the points $x'[m]$ to the right by $1/2$ by defining another index $m' = m + 1/2$, then $x'[m'] = x'[m+1/2]$ is symmetric with respect to the origin at $m' = 0$ as Fig.2.5 shows.

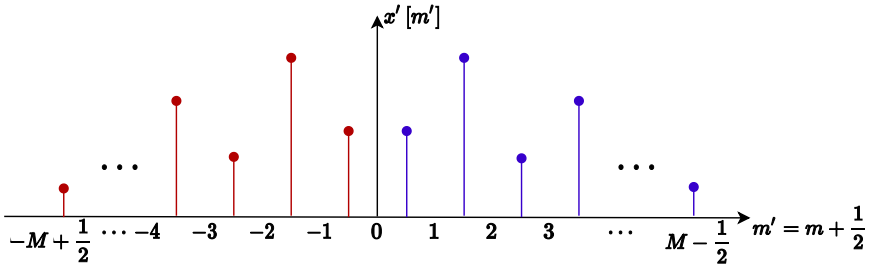


Figure 2.5: The shifted sequence

Perform DFT to this 2M-point symmetric sequence (for convenience, we denote $x'[m]$ as $x[m]$):

$$\begin{aligned} X[\mu] &= \frac{1}{\sqrt{2M}} \sum_{m'=-M+1/2}^{M-1/2} x\left[m' - \frac{1}{2}\right] e^{-j2\pi m'\mu/2M} \\ &= \frac{1}{\sqrt{2M}} \sum_{m'=-M+1/2}^{M-1/2} x\left[m' - \frac{1}{2}\right] \cos\left(\frac{2\pi m'\mu}{2M}\right) - \frac{j}{\sqrt{2M}} \sum_{m'=-M+1/2}^{M-1/2} x\left[m' - \frac{1}{2}\right] \sin\left(\frac{2\pi m'\mu}{2M}\right) \quad (2.4) \\ &= \frac{1}{\sqrt{2M}} \sum_{m'=-M+1/2}^{M-1/2} x\left[m' - \frac{1}{2}\right] \cos\left(\frac{2\pi m'\mu}{2M}\right) \quad (2.5) \\ &= \sqrt{\frac{2}{M}} \sum_{m'=1/2}^{M-1/2} x\left[m' - \frac{1}{2}\right] \cos\left(\frac{2\pi m'\mu}{2M}\right) \quad (\mu = 0, \cdots, 2M-1). \quad (2.6) \end{aligned}$$

Here we have used the fact that $x[m'-1/2]$ is even, $\cos(2\pi m'\mu/2M)$ and $\sin(2\pi m'\mu/2M)$ are respectively even and odd, all with respect to $m' = 0$ or $m = -1/2$. Therefore, we can simplify the equation from Eq. (2.4) and Eq. (2.5).

Substituting $m' = m + 1/2$ into Eq. (2.6), we get the Discrete Cosine Transform (DCT):

$$X[\mu] = \sqrt{\frac{2}{M}} \sum_{m=0}^{M-1} x\,[m] \cos\left(\frac{(2m+1)\mu\pi}{2M}\right) \quad (\mu = 0, \cdots, M-1). \tag{2.7}$$

We can define the coefficient $c[\mu, m]$ as

$$c[\mu, m] \triangleq \sqrt{\frac{2}{M}} \cos\left(\frac{(2m+1)\mu\pi}{2M}\right), \quad (m, \mu = 0, 1, \cdots, M-1).$$

which can be considered as the component on the μth row and mth column of an $M \times M$ matrix $\mathbf{C}$, called the DCT matrix.

We can show that all row vectors of $\mathbf{C}$ are orthogonal and normalized, except the first one ($\mu = 0$):

$$\sqrt{\sum_{m=0}^{M-1} c^2[\mu, m]} = \sqrt{\frac{2}{M} \sum_{m=0}^{M-1} \cos^2\left(\frac{(2m+1)\mu\pi}{2M}\right)} = \begin{cases} \sqrt{2} & \mu = 0 \\ 1 & \mu = 1, 2, \cdots, M-1. \end{cases}$$

To make DCT a orthonormal transform, we define a coefficient

$$a[\mu] = \begin{cases} \sqrt{1/M} & \mu = 0 \\ \sqrt{2/M} & \mu = 1, 2, \cdots, M-1, \end{cases}$$

so that the DCT now becomes

$$X[\mu] = a[\mu] \sum_{m=0}^{M-1} x[m] \cos\left(\frac{(2m+1)n\pi}{2M}\right) = \sum_{m=0}^{M-1} x[m] c[\mu, m] \quad (\mu = 0, \cdots, M-1),$$

where $c[\mu, m]$ is modified with $a[\mu]$, which is also the component in the μth row and mth column of the M by M cosine transform matrix:

$$\begin{bmatrix} \cdots & \cdots & \cdots \\ \vdots & c[\mu, m] & \vdots \\ \cdots & \cdots & \cdots \end{bmatrix} = \begin{bmatrix} \mathbf{c}_0^T \\ \vdots \\ \mathbf{c}_{M-1}^T \end{bmatrix} = \mathbf{C}^T.$$

Here $\mathbf{c}_i^T = [c[i,0], \cdots, c[i,M-1]]$ is the ith row of the DCT transform matrix $\mathbf{C}$. As these row vectors are orthogonal:

$$\mathbf{c}_i \cdot \mathbf{c}_j = \mathbf{c}_i^T \mathbf{c}_j = \delta_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j. \end{cases}$$

The DCT matrix $\mathbf{C}$ is orthogonal:

$$\mathbf{C}^{-1} = \mathbf{C}^T, \qquad \text{i.e.} \qquad \mathbf{C}^T\mathbf{C} = \mathbf{I},$$

and it is real $\mathbf{C} = \mathbf{C}^*$. Now the DCT can be expressed in matrix form as:

$$\mathbf{X} = \mathbf{C}^T\mathbf{x}.$$

Left multiplying both sides by $\mathbf{C}$ we get

$$\mathbf{C}\mathbf{X} = \mathbf{C}\mathbf{C}^T\mathbf{x} = \mathbf{C}\mathbf{C}^{-1}\mathbf{x} = \mathbf{x}.$$

This is the inverse DCT:

$$\mathbf{x} = \mathbf{C}\mathbf{X}$$

or in component form:

$$\begin{aligned} x[m] &= \sum_{\mu=0}^{M-1} a[\mu] X[\mu] \cos\left(\frac{(2m+1)\mu\pi}{2M}\right) \\ &= \sum_{\mu=0}^{M-1} X[\mu] c[\mu, m] \quad (m = 0, \cdots, M-1). \end{aligned}$$

Here is an example where DCT and FFT(fast Fourier transform is an algorithm that computes DFT) are performed on the same signal. Comparing the results in Fig.2.6, it can be observed that DCT transform has only real part, and after DFT transform, there is imaginary part. In this example, DCT uses only 3 points in the frequency domain to represent the signal, and after the DFT transform, 5 points are required in the frequency domain to represent the signal. Thus, DCT is more efficient than DFT.

```
1  x = [0 0 3 3 3 3 0 0];
2  X = dct(x)

   X = 1×8
        4.2426         0   -3.9197         0         0         0    1.6236         0

3  y=fft(x)

   y = 1×8 complex
       12.0000 + 0.0000i  -7.2426 - 3.0000i   0.0000 + 0.0000i   1.2426 + 3.0000i   0.0000 + 0.0000i   1.2426 - 3.0000i ...

4  A=idct(X)

   A = 1×8
        0.0000    0.0000    3.0000    3.0000    3.0000    3.0000    0.0000    0.0000

5  B=ifft(y)

   B = 1×8
                0   -0.0000    3.0000    3.0000    3.0000    3.0000         0   -0.0000
```

Figure 2.6: A signal transformed by DCT and FFT respectively

2.1.5 Application of DCT

DCT is applied in the JPEG image compression algorithm [1, page 566-584]. Among all image coding standards, the JPEG format (short for Joint Photographic Experts Group) is the most suitable for both color and binary still images of different types and different resolutions. JPEG image compression algorithm is a widely used still image compression standard, and is a lossy compression algorithm based on DCT.

Figure 2.7 is the graphic description of a block transform coding system which contains encoder and decoder.

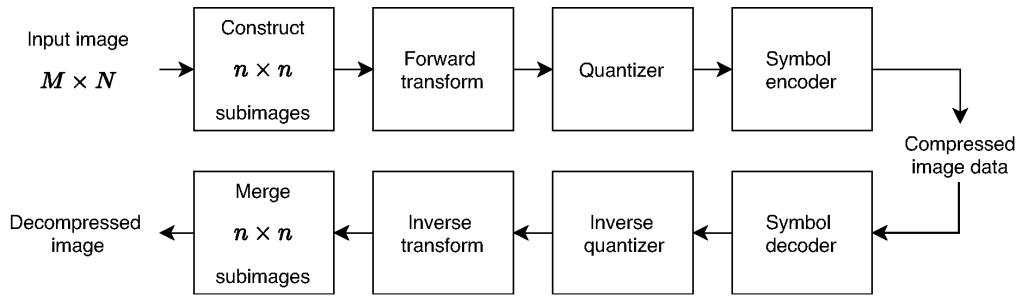


Figure 2.7: A block transform coding system

We suppose that the original image consists of 256 or 2^8 possible intensities. The block transform coding involves several steps:

1. Decompose the input image into data blocks of 8×8;
2. Do forward DCT to each block, and transform each of them into 64 DCT coefficients among which one coefficient with zero frequency in both dimensions is DC coefficient and the other 63 with non-zero frequencies are AC coefficients;
3. Quantize the DCT coefficients;
4. Encode the result of quantization and then transmit the compressed image data.

We will explain these steps in details.

Step 1

Construct $n \times n$ subimages, each subimages with the size 8×8. Then shift the pixels of the original subimage by -2^7 or -128 intensity levels to get a new array, because the range of numbers accepted by the DCT formula is between -128 and 127.

Step 2

Let $g(x, y)$ denote the subimage with the size 8×8. The discrete transform of $g(x, y)$ is denoted as $T(u, v)$, and the relation between them is

$$T(u, v) = \sum_{x=0}^{n-1} \sum_{y=0}^{n-1} g(x, y) r(x, y, u, v), \tag{2.8}$$

where $u, v = 0, 1, 2, ..., n-1$. Similarly, with $T(u, v)$, we can get $g(x, y)$ by the inverse discrete transform:

$$g(x, y) = \sum_{u=0}^{n-1} \sum_{v=0}^{n-1} T(u, v) s(x, y, u, v). \tag{2.9}$$

In these equations, $r(x, y, u, v)$ and $s(x, y, u, v)$ are called the forward and inverse transformation kernels, respectively. Furthermore, they are also referred to as basic functions or basis images. At the same time, $T(u, v)$ for $u, v = 0, 1, 2, ..., n-1$ in Eq. (2.8) are called transform coefficients; they can be viewed as the expansion coefficients of a series expansion of $g(x, y)$ with respect to the basic function $s(x, y, u, v)$.

The idea is that we multiply the original subimage with the basic function pixel by pixel to obtain the transformed coefficients which will work for compression. The kernels are the cores for the transform process. They determine the type of transformation and the complexity of the overall calculation. The best-known pair of transform kernels is

$$r(x, y, u, v) = e^{-j2\pi(ux+vy)/n}$$

and

$$s(x, y, u, v) = \frac{1}{n^2} e^{j2\pi(ux+vy)/n},$$

as we referred before, this is the basic function of DFT pair in 2-D. When implementing JPEG algorithm, we focus on the explanation of DCT.

In the last section, we have already analyzed 1-D DCT, the kernels are $\mathbf{C}$ and $\mathbf{C^T}$. 2-D DCT transformation whose kernel is shown in Fig.2.8 is to perform another DCT transformation based on 1-D DCT. Here is the formula:

$$F(u, v) = \alpha(u)\alpha(v) \sum_{x=0}^{n-1} \sum_{y=0}^{n-1} f(x, y) \cos\left[\frac{(2x+1)u\pi}{2n}\right] \cos\left[\frac{(2y+1)v\pi}{2n}\right],$$

where

$$\alpha(u) = \begin{cases} \sqrt{1/n} & u = 0 \\ \sqrt{2/n} & u = 1, 2, ..., n-1 \end{cases}$$

and similarly for $\alpha(v)$.

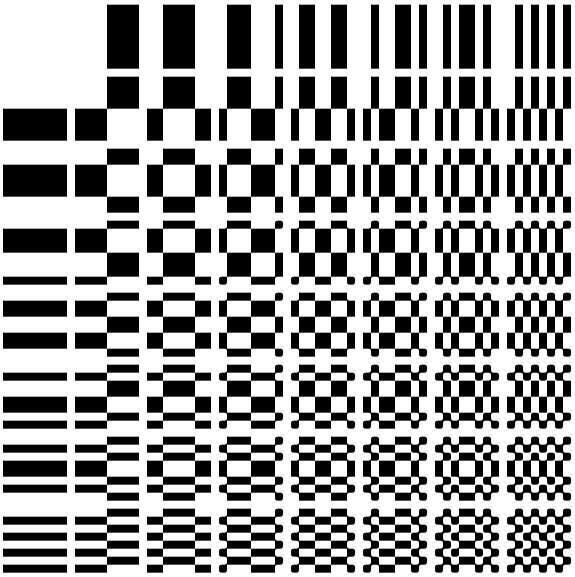


Figure 2.8: DCT basic function

As we implement the algorithm with Matlab, and the computation in Matlab is in matrix form, we show DCT formulas in matrix form:

$$\mathbf{F} = \mathbf{A}\mathbf{f}\mathbf{A}^T,$$

$$A(i,j) = \alpha(i)\cos\left[\frac{(2i+1)j\pi}{2n}\right],$$

where i stands for u, v and j stands for x, y. Then the inverse DCT can be derived from the following process:

$$\mathbf{A^{-1}} = \mathbf{A^T},$$

$$\mathbf{f} = \mathbf{A^{-1}F(A^T)^{-1}} = \mathbf{A^T F A}.$$

So we can define IDCT as:

$$f(x,y) = \frac{1}{n}F(0,0) + \frac{\sqrt{2}}{n}\sum_{v=0}^{n-1}F(0,v)\cos\frac{(2y+1)v\pi}{2n} + \frac{\sqrt{2}}{n}\sum_{u=0}^{n-1}F(u,0)\cos\frac{(2x+1)u\pi}{2n} + \frac{2}{n}\sum_{u=1}^{N-1}\sum_{v=1}^{n-1}F(u,v)\cos\frac{(2x+1)u\pi}{2n}\cos\frac{(2y+1)v\pi}{2n}. \tag{2.10}$$

Above, $f(x, y)$ is the intensity of the pixel in row x and column y, $x, y = 0, 1, 2, ..., N - 1$; $F(u, v)$ is the DCT coefficient, u, v are the generalized frequency variables, $u, v = 0, 1, 2, ..., N - 1$.

As shown in Fig.2.2, for most images, after DCT is performed, most of the signal energy lies at low frequencies that appear in the upper left corner and the lower right coefficients represent higher frequencies are often small enough to be neglected with little visible distortion.

Step 3

In this step, we will implement quantization, the object of which is to decrease the magnitudes of non-zero coefficients and to increase the number of zero coefficients so that we can present the image with fewer data. Coefficient quantization is a very important process, although it is the reason for the loss of information during encoding and decoding. The quantization process can be shown by the equation below:

$$Q(u, v) = round\left[\frac{F(u, v)}{S(u, v)}\right],$$

where $Q(u, v)$ means quantized coefficient magnitude, $F(u, v)$ represents the DCT coefficient, and $S(u, v)$ is an element of the quantization table. The size of the quantization table is the same as that of the subimage, it is corresponding to the DCT coefficients.

Since the human eye is more sensitive to low-frequencies (upper left corner) than high-frequencies (low right corner), the quantization steps in the upper left corner of the table are smaller than the quantization steps in the lower right corner. In addition, the quantization table shown in Tab.2.1 is the default luminance quantization table, custom quantization tables can also be set.

There are two points to note about the quantization step:

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

Table 2.1: Quantization table

1. The luminance quantization table can be multiplied by a scaling factor to control the compression rate. The larger the coefficient is, the higher the compression ratio is, and the worse the quality of the resulting image is.

2. After quantization, the coefficients can be masked, the more AC coefficients the mask matrix retains, the higher the quality the resulting image is.

Step 4

Execute the zigzag ordering pattern shown in Fig.2.9 on the coefficients after the transformation and normalization,and get the resulting 1-D coefficient sequence.

Step 5

Code the 1-D coefficient sequence that we get from step 4 and save. Here are the coding methods applied.

1. Difference Pulse Code Modulation(DPCM)

   The DC coefficient of the 8×8 image blocks after DCT transformation has two characteristics:

   (a) The coefficient value is relatively large;

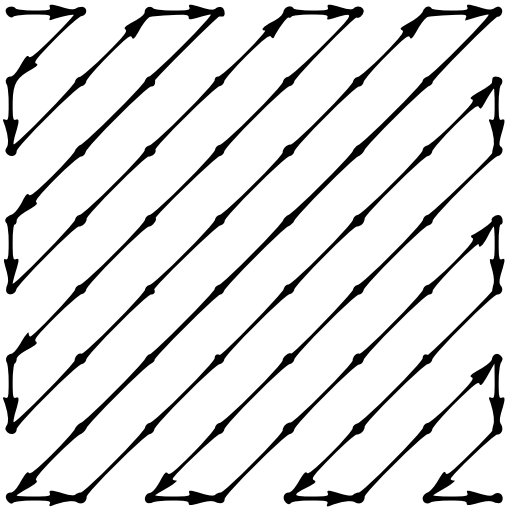


Figure 2.9: Zigzag ordering pattern

(b) The DC coefficients of adjacent 8×8 image blocks do not change much.

According to these two characteristics, DC coefficients are generally processed with DPCM, that is: take the difference between each DC value and the previous DC value in the image component to encode. For example, assuming that the DC coefficient of an 8×8 image block is 17, and the DC coefficient of the previous 8×8 image block is 15, the difference between the two is 2.

2. Run Length Coding (RLC)

The characteristic of the AC coefficient after quantization is that there are many coefficients with the value 0 among all 63 coefficients. Therefore, RLC which is mentioned in the section1.2.1 can be used to further reduce the amount of data transmission.

However, in JPEG encoding, the meaning of RLC is slightly different from its original meaning. In JPEG encoding, it is assumed that after RLC encoding, a (M, N) data pair is obtained, where M is the number of consecutive 0s between two non-zero AC coefficients (that is, run length), and N is the value of the next

non-zero AC coefficient.

3. Entropy coding

   After obtaining the intermediate format of DC coefficients and the intermediate format of AC coefficients, it is necessary to encode both of them to further compress the image data with the entropy encoder which is a general lossless data compression method that encodes symbols by using an amount of bits inversely proportional to the probability of the symbols.

   The JPEG standard relies on two entropy encoding methods: Huffman encoding and arithmetic encoding. The JPEG basic system uses the use of Huffman encoding which we introduce in the section1.2.1 in Chapter 1. Thus, by processing Huffman coding, binary codes with a shorter character length are assigned to characters with a high probability of occurrence, and binary codes with a longer character length are assigned to characters with a low probability of occurrence so that the average code length of a character is the shortest.

2.2 Wavelet Transform (WT)

2.2.1 The shortages of Fourier Transform

As there are some shortcomings of Fourier Transform, they lead to the foundation of wavelet transform. First, we will discuss the shortages of Fourier transform. As what we can see in the Fourier transform equation, $F(\omega)$ presents the amount of energy that the signal $f(t)$ has in total at a specified frequency ω. However, with Fourier transform, we are unable to know the instantaneous frequency of $f(t)$ at a certain moment. Here is an example [25]

There is a signal consisting of two sinusoids of 5 and 10 Hertz and it is corrupted by random noise. As Fig. 2.10 shows, in time domain, it's difficult to detect when the

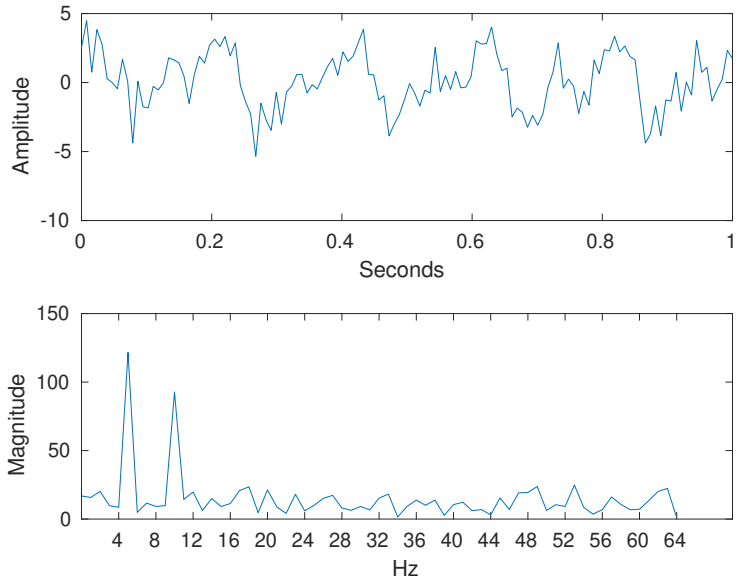


Figure 2.10: A signal in time domain and in frequency domain

significant oscillations happen as we don't know the specific frequency for a certain moment; likewise, while in frequency domain, the dominant oscillations at 5 Hz and 10 Hz are easy to detect, it's hard to isolate the frequencies in time, as each frequency exists across all the time. Fourier transform acts as a summary of the signal's frequency over that time period [26], but can't provide details for any specific time points.

Unlike the basic function composed of sin and cos which lasts forever and is infinite in Fourier Transform , wavelets are limited in time and frequency. There are some examples of different wavelets shown in Fig.2.11.

In practice, most signals processed are considered finite and the finite property of wavelet provides us a more accurate and flexible representation of the signals. Next, we will get to know more about wavelet [1, page 477-510].

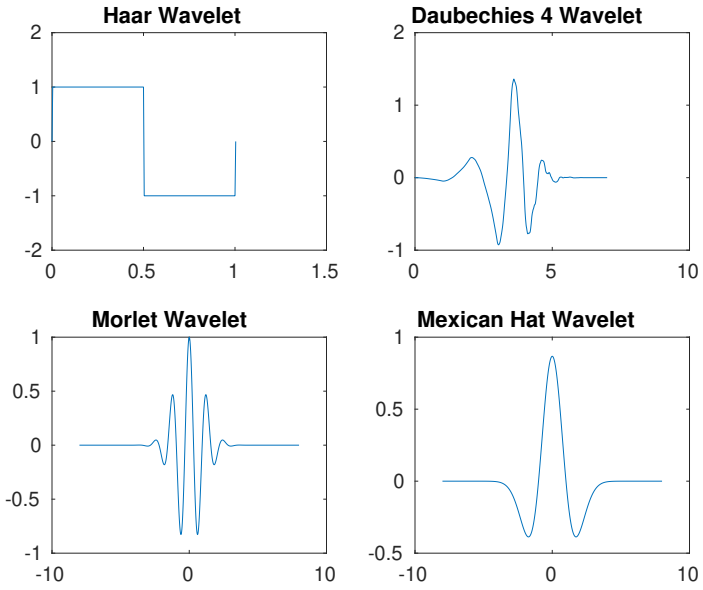


Figure 2.11: Different types of wavelets

2.2.2 Multi-resolution Expansions

Series Expansions

In functional analysis, a function $f(x)$ is often expressed as a finite or infinite sum of given functions

$$f(x) = \sum_k \alpha_k \phi_k(x), \tag{2.11}$$

where k is an integer index, the α_k are real-valued expansion coefficients, and the $\phi_k(x)$ are real-valued expansion functions.

If there is a unique set of α_k for any given $f(x)$, the $\phi_k(x)$ are called basic functions, and the expansion set, $\{\phi_k(x)\}$, is called a basis for the class of functions that can be so expressed.

We can construct a function space V with the basic functions

$$V = \overline{\underset{k}{Span}\{\phi_k(x)\}}, \tag{2.12}$$

where $\underset{k}{Span}\{\phi_k(x)\}$ is the set of all finite linear combination of $\{\phi_k(x)\}$, and $\overline{A}$ means the closure of a set A. The set V is referred as the closed span of the expansion set $\{\phi_k(x)\}$.

Now, any function $f(x) \in V$ can be expanded in the form of Eq.(2.11).

Scaling Functions

Given a real, square-integrable function $\phi(x)$, we can compose a set $\{\phi_{j,k}(x)\}$ of expansion functions, where

$$\phi_{j,k}(x) = 2^{j/2}\phi(2^j x - k), \quad \forall j,k \in \mathbb{Z}, \phi(x) \in L^2(\mathbb{R}). \tag{2.13}$$

Here, k determines the position of $\phi_{j,k}(x)$ along the x-axis, j determines the width of $\phi_{j,k}(x)$, and the term $2^{j/2}$ controls the amplitude of the function. Because the shape of $\phi_{j,k}(x)$ is determined by j, $\phi(x)$ is called a scaling function.

Based on Eq.(2.12), we can define the subspace V_j spanned over k for any fixed j as

$$V_j = \overline{\underset{k}{Span}\{\phi_{j,k}(x)\}}.$$

Specifically, when $j = j_0$,

$$V_{j_0} = \overline{\underset{k}{Span}\{\phi_{j_0,k}(x)\}}.$$

If $f(x) \in V_{j_0}$, then it can be expressed as

$$f(x) = \sum_k \alpha_k \phi_{j_0,k}(x), \tag{2.14}$$

as j increases, the width of $\phi_{j,k}(x)$ becomes narrower and the subspace becomes bigger. For more properties of scaling function, we can check out the four fundamental requirements of multi-resolution analysis:

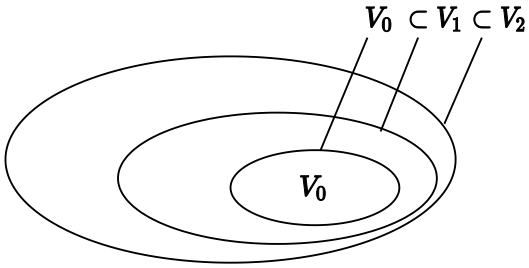


Figure 2.12: The nested function spaces spanned by a scaling function.

1. The scaling function is orthogonal to its integer translates.

   This means that

   $$\langle \phi_{j,k}, \phi_{j,l} \rangle = \delta(k-l).$$

2. The subspaces spanned by the scaling function at low scales are nested within those spanned at higher scales:

   $$V_{-\infty} \subset \cdots \subset V_{-1} \subset V_0 \subset V_1 \subset V_2 \subset \cdots \subset V_{\infty}.$$

   It is easy to find it out by observing the scaling function that if $f(x) \in V_j$, then $f(2x) \in V_{j+1}$.

3. The only function that is common to all V_j is $f(x) = 0$.

   This means that

   $$V_{-\infty} = \{0\},$$

   or

   $$\bigcap_{j \in \mathbb{Z}} V_j = \{0\}.$$

4. Any function can be represented with arbitrary precision.

   This is because all measurable, square-integrable functions can be represented by the scaling functions in the limit as $j \to \infty$. That is,

   $$V_{\infty} = L^2(\mathbb{R}),$$

or

$$\bigcup_{j\in\mathbb{Z}} V_j = L^2(\mathbb{R}).$$

With these requirements, we can know that $V_j \subset V_{j+1}$, therefore, the expansion functions of subspace V_j can be expressed as a weighted sum of the expansion functions of subspace V_{j+1}:

$$\phi_{j,k}(x) = \sum_n \alpha_n \phi_{j+1,n}(x), \tag{2.15}$$

then we substitute the scaling function as Eq.(2.13) shows into $\phi_{j+1,n}(x)$ and replace α_n with $h_\phi(n)$. Thus, Eq. (2.15) becomes

$$\phi_{j,k}(x) = \sum_n h_\phi(n) 2^{(j+1)/2} \phi(2^{j+1}x - n).$$

Furthermore, because $\phi(x) = \phi_{0,0}(x)$, we can let $j = 0, k = 0$ to obtain the simpler non-subscript expression

$$\phi(x) = \sum_n h_\phi(n) \sqrt{2} \phi(2x - n), \tag{2.16}$$

where the $h_\phi(n)$ coefficients are called scaling function coefficients and h_ϕ is referred to as a scaling vector. Eq. (2.16) is fundamental to multi-resolution analysis (MRA) and is called the refinement equation, the MRA equation or the dilation equation. It means that the expansion function of any subspace can be constructed by the expansion functions of the next higher resolution space.

Wavelet Functions

Given a scaling function that meets the MRA requirements, we can see from Fig.2.12 that there is always a gap between any two adjacent scaling subspaces V_j and V_{j+1}. Likewise,

we can define a wavelet function $\psi(x)$ that, together with its integer translates and binary scaling, spans the gap

$$\psi_{j,k}(x) = 2^{j/2}\psi(2^j x - k), \quad \forall j,k \in \mathbb{Z}, \psi(x) \in L^2(\mathbb{R}). \tag{2.17}$$

We denote the gap as the subspace W_j and it is spanned over k for any fixed j as

$$W_j = \overline{\underset{k}{Span}\{\psi_{j,k}(x)\}},$$

which means that for any function $f(x) \in W_j$, it can be expressed as

$$f(x) = \sum_k \alpha_k \psi_{j,k}(x). \tag{2.18}$$

The relation between the scaling and wavelet function subspaces, as Fig.2.13 shows, can be denoted as

$$V_{j+1} = V_j \oplus W_j,$$

where $\oplus$ denotes the union of spaces. The orthogonal complement of V_j in V_{j+1} is W_j and all members of V_j are orthogonal to the members of W_j, so we can have the equation

$$\langle \phi_{j,k}(x), \psi_{j,l}(x) \rangle = 0, \quad j,k,l \in \mathbb{Z}.$$

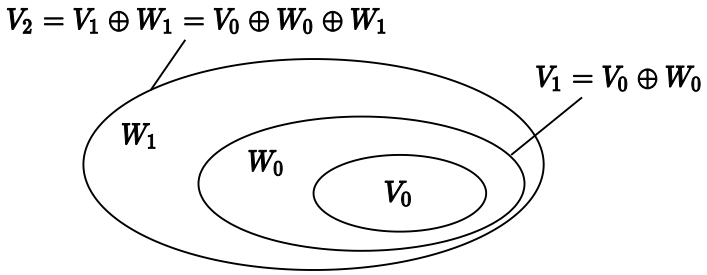


Figure 2.13: The relationship between scaling and wavelet function spaces.

After defining the wavelet function sub-space W_j and knowing the relation between the subspaces, we can now express the space of all measurable square-integrable functions as

$$L^2(\mathbb{R}) = V_0 \oplus W_0 \oplus W_1 \oplus \ldots, \tag{2.19}$$

or

$$L^2(\mathbb{R}) = V_1 \oplus W_1 \oplus W_2 \oplus \ldots,$$

or even

$$L^2(\mathbb{R}) = \ldots \oplus W_{-2} \oplus W_{-1} \oplus W_0 \oplus W_1 \oplus W_2 \oplus \ldots.$$

As $V_{j+1} = V_j \oplus W_j$, the expansion functions of sub-space W_j can also be represented by a weighted sum of the expansion functions of sub-space V_{j+1}

$$\psi(x) = \sum_n h_\psi(n)\sqrt{2}\phi(2x - n), \tag{2.20}$$

where the $h_\psi(n)$ are called the wavelet function coefficients and h_ψ is the wavelet vector.

Observing Eq.2.20 and Eq.2.16, they are expressed by the same basic functions but with different expansion coefficients. The relation between the coefficients $h_\psi(n)$ and $h_\phi(n)$ is

$$h_\psi(n) = (-1)^n h_\phi(1 - n).$$

2.2.3 Wavelet Transform in One Dimension

The Wavelet Series Expansions

From Eq.(2.19), any function $f(x)$ in the space $L^2(\mathbb{R})$ can be represented by a scaling function expansion of subspace V_{j_0} and some number of wavelet function expansions of

subspaces $W_{j_0}, W_{j_0+1}, \ldots$

$$f(x) = \sum_k c_{j_0}(k)\phi_{j_0,k}(x) + \sum_{j=j_0}^{\infty}\sum_k d_j(k)\psi_{j,k}(x), \tag{2.21}$$

where j_0 is an arbitrary starting scale and the $c_{j_0}(k)$ and $d_j(k)$ are relabeled α_k from Eqs. (2.14) and (2.18) respectively. The $c_{j_0}(k)$ are normally called approximation and/or scaling coefficients. The $d_j(k)$ are referred to as detail and/or wavelet coefficients, and we can calculate them by the formulas

$$c_{j_0}(k) = \langle f(x), \phi_{j_0,k}(x)\rangle = \int f(x)\phi_{j_0,k}(x)dx, \tag{2.22}$$

$$d_j(k) = \langle f(x), \psi_{j,k}(x)\rangle = \int f(x)\psi_{j,k}(x)dx. \tag{2.23}$$

The Discrete Wavelet Transform

Like the Fourier series expansion, the wavelet series expansion in Eq.(2.21) maps a function of a continuous variable into a sequence of coefficients. When the decomposed function is discrete which means it is sampled, the resulting coefficients are called discrete wavelet transform (DWT). Here is the detail.

We sample M elements from the continuous signal $f(x)$

$$f(n) = f(x_0 + n\Delta x), \quad n = 0, 1, \ldots, M-1,$$

where x_0 is the initial time and Δx is the sampling period.

The $\phi_{j,k}(n)$ and $\psi_{j,k}(n)$ are also sampled from the basic functions $\phi_{j,k}(x)$ and $\psi_{j,k}(x)$ and both contain M elements.

Therefore, the wavelet series expansion coefficients for $f(x)$ become the forward DWT

coefficients for sequence $f(n)$:

$$W_\phi(j_0,k) = \frac{1}{\sqrt{M}} \sum_n f(n)\phi_{j_0,k}(n),$$
$$W_\psi(j,k) = \frac{1}{\sqrt{M}} \sum_n f(n)\psi_{j,k}(n), \quad j \geq j_0,$$

where $W_\phi(j_0,k)$ and $W_\psi(j,k)$ correspond to the $c_{j_0}(k)$ and $d_j(k)$ of the wavelet series expansion in Eq. (2.22) and Eq. (2.23) respectively. The complementary inverse DWT which corresponds to the wavelet series expansion in Eq. (2.21) is

$$f(n) = \frac{1}{\sqrt{M}} \sum_k W_\phi(j_0,k)\phi_{j_0,k}(n) + \frac{1}{\sqrt{M}} \sum_{j=j_0}^{\infty} \sum_k W_\psi(j,k)\psi_{j,k}(n).$$

Normally, we let $j_0 = 0$ and select M to be a power of 2 (i.e., $M = 2^J$) so that the summations in equations of DWT and inverse DWT are performed over $n = 0, 1, 2, \ldots, M-1, j = 0, 1, 2, \ldots, J-1$, and $k = 0, 1, 2, \ldots, 2^j - 1$.

Here is an example of application of DWT [27].

Given a sampled signal $\boldsymbol{f} = [f[0], \cdots, f[3]]^T = [0, 3, 1, -1]^T$, thus, $M = 4$. The scaling and wavelet functions we use are discrete Haar scaling and wavelet functions

$$\begin{bmatrix} \phi_{0,0}[n] \\ \psi_{0,0}[n] \\ \psi_{1,0}[n] \\ \psi_{1,1}[n] \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ \sqrt{2} & -\sqrt{2} & 0 & 0 \\ 0 & 0 & \sqrt{2} & -\sqrt{2} \end{bmatrix}.$$

Then we can start calculating the scaling coefficients and the wavelet coefficients

$$W_\phi(0,0) = \frac{1}{2}\sum_{n=0}^{3} f[n]\,\phi_{0,0}[n] = \frac{1}{2}[1\cdot 0 + 1\cdot 3 + 1\cdot 1 + 1\cdot(-1)] = 1.5,$$

$$W_\psi(0,0) = \frac{1}{2}\sum_{n=0}^{3} f[n]\,\psi_{0,0}[n] = \frac{1}{2}[1\cdot 0 + 1\cdot 3 + (-1)\cdot 1 + (-1)\cdot(-1)] = 1.5,$$

$$W_\psi(1,0) = \frac{1}{2}\sum_{n=0}^{3} f[n]\,\psi_{1,0}[n] = \frac{1}{2}\Big[\sqrt{2}\cdot 0 + (-\sqrt{2})\cdot 3 + 0\cdot 1 + 0\cdot(-1)\Big] = -1.5\sqrt{2},$$

$$W_\psi(1,1) = \frac{1}{2}\sum_{n=0}^{3} f[n]\,\psi_{1,1}[n] = \frac{1}{2}\Big[0\cdot 0 + 0\cdot 3 + \sqrt{2}\cdot 1 + (-\sqrt{2})\cdot(-1)\Big] = \sqrt{2}.$$

With the resulting coefficients, we can get back the original sampled signal by applying the inverse DWT

$$f[n] = \frac{1}{2}\Big[W_\phi(0,0)\phi_{0,0}[n] + W_\psi(0,0)\psi_{0,0}[n] + W_\psi(1,0)\psi_{1,0}[n] + W_\psi(1,1)\psi_{1,1}[n]\Big].$$

The Continuous Wavelet Transforms

For the continuous wavelet transform, we just need to change Fourier basis to wavelet basis in the transform equation:

$\forall f(t) \in L^2(\mathbb{R})$,

$$W_\psi(a,b) = \int_{-\infty}^{+\infty} f(x)\overline{\psi_{a,b}}(t)dx = \frac{1}{\sqrt{|a|}}\int_{-\infty}^{+\infty} f(x)\overline{\psi}\left(\frac{x-b}{a}\right)dx. \tag{2.24}$$

It is called the continuous wavelet transform of $f(x)$, where $f(x)$ is the real signal, ψ is an arbitrary wavelet, a is the scale parameter and b is the translation parameter and W is the processed signal. The inverse wavelet transform:

$$f(x) = \frac{1}{C_\psi}\int\int_{\mathbb{R}\times\mathbb{R}^*} W_\psi(a,b)\psi_{(a,b)}(x)\frac{dadb}{a^2},$$

where $C_\psi = \int_{\mathbb{R}^*}\frac{|\Psi(\omega)|^2}{|\omega|}d\omega$.

There are some points that should be noted:

1. The continuous translation parameter b takes the place of the integer translation parameter k.

2. The continuous scale parameter a is inversely related to the binary scale parameter 2^j.

2.2.4 The Fast Wavelet Transform

The fast wavelet transform (FWT), also called Mallat's herringbone algorithm [29], is a computationally efficient implementation of the discrete wavelet transform (DWT). It simplifies the computation by exploiting the relationship between the coefficients of the DWT at adjacent scales. Here is the derivation process of FWT.

Based on MRA equation (Eq.(2.16)), replace x by $2^j x - k$, and let $m = 2k + n$ i.e. $n = m - 2k$, then we can have

$$\begin{aligned}\phi(2^j x - k) &= \sum_n h_\phi(n)\sqrt{2}\phi\left(2(2^j x - k) - n\right) \\ &= \sum_m h_\phi(m - 2k)\sqrt{2}\phi(2^{j+1}x - m).\end{aligned}$$

Similarly, we can obtain the following equation based on the wavelet function (Eq.(2.20))

$$\psi(2^j x - k) = \sum_m h_\psi(m - 2k)\sqrt{2}\phi(2^{j+1}x - m). \tag{2.25}$$

For the coefficient $W_\psi(j,k)$ of DWT,

$$W_\psi(j,k) = \frac{1}{\sqrt{M}} \sum_{n=0}^{M-1} f(x)\psi_{j,k}(x) \tag{2.26}$$

$$= \frac{1}{\sqrt{M}} \sum_{n=0}^{M-1} f(x)2^{j/2}\psi(2^j x - k) \tag{2.27}$$

$$= \frac{1}{\sqrt{M}} \sum_{n=0}^{M-1} f(x)2^{j/2}\Big[\sum_m h_\psi(m-2k)\sqrt{2}\phi(2^{j+1}x - m)\Big] \tag{2.28}$$

$$= \sum_m h_\psi(m-2k)\Big[\frac{1}{\sqrt{M}} \sum_{n=0}^{M-1} f(x)2^{(j+1)/2}\phi(2^{j+1}x - m)\Big] \tag{2.29}$$

$$= \sum_m h_\psi(m-2k)W_\psi(j+1,k). \tag{2.30}$$

$$\tag{2.31}$$

Substitue wavelet function into Eq.(2.26) to get Eq.(2.27), and substitute Eq.(2.25) into Eq.(2.27) to get Eq.(2.28). Change order of Eq.2.28 to get Eq.2.29. In Eq.(2.30), the part in the square brackets equals to $W_\psi(j+1,k)$, so we finally get Eq.(2.30) representing the coefficient $W_\psi(j,k)$ of DWT.

Similarly, for the coefficient $W_\phi(j,k)$ of DWT, we can have

$$W_\phi(j,k) = \sum_m h_\phi(m-2k)W_\psi(j+1,k). \tag{2.32}$$

We compare Eq.(2.30) and Eq.(2.32) with the discrete convolution formula,

$$y(n) = \sum_{i=-\infty}^{+\infty} h(n-i)x(i) = h(n) * x(n),$$

therefore,

$$W_\psi(j,k) = h_\psi(-n) * W_\phi(j+1,n)\Big|_{n=2k,k\geq 0},$$

$$W_\phi(j,k) = h_\phi(-n) * W_\phi(j+1,n)\Big|_{n=2k,k\geq 0}.$$

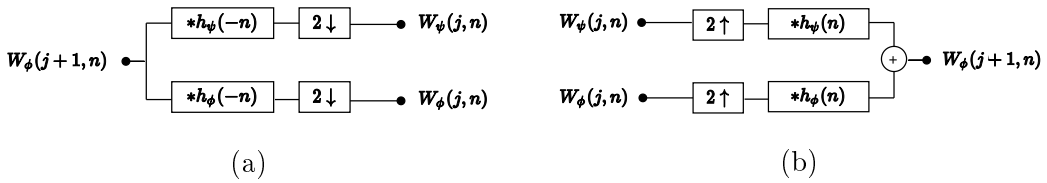


Figure 2.14: (a)The FWT analysis bank, (b)The FWT^{-1} synthesis filter bank.

An FWT analysis bank can be generated after performing downsampling by 2, so that the size of $W_\psi(j,n)$ and $W_\phi(j,n)$ is half of $W_\phi(j+1,n)$, shown in Fig.2.14a. Then we can infer the FWT^{-1} filter bank shown in Fig.2.14b to reconstruct $W_\phi(j+1,k)$ from the results of the forward transform.

In the two-band subband coding and decoding system as Fig. 2.15 shows, perfect reconstruction [28] requires $g_i(n) = h_i(-n)$ for $i = \{0,1\}$. In the FWT analysis filter, $h_0(n) = h_\phi(-n)$ and $h_1(n) = h_\psi(-n)$, the required FWT^{-1} synthesis filters are $g_0(n) = h_0(-n) = h_\phi(n)$ and $g_1(n) = h_1(n) = h_\psi(n)$ [1, page 499]. The computation of

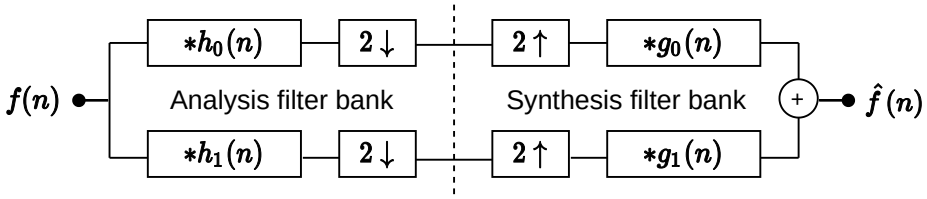


Figure 2.15: A two-band subband coding and decoding system.

the FWT^{-1} filter bank

$$W_\phi(j+1,k) = h_\phi(k) * W_\phi^{2\uparrow}(j,k) + h_\psi(k) * W_\psi^{2\uparrow}(j,k) \Big| k \geq 0,$$

where $2\uparrow$ means upsampling by 2, this is inserting zeros in W so that it is twice its original length.

2.2.5 Wavelet Transforms in Two Dimensions

In this section, we extend the one-dimensional transforms to two-dimensional ones. There is one scaling function $\phi(x,y)$ and three two-dimensional wavelets $\psi^H(x,y), \psi^V(x,y)$ and $\psi^D(x,y)$:

$$\begin{aligned}\phi(x,y) &= \phi(x)\phi(y),\\ \psi^H(x,y) &= \psi(x)\phi(y),\\ \psi^V(x,y) &= \phi(x)\psi(y),\\ \psi^D(x,y) &= \psi(x)\psi(y).\end{aligned}$$

Then we can define the scaled and translated basic functions:

$$\phi_{j,m,n}(x,y) = 2^{j/2}\phi(2^j x - m, 2^j y - n), \tag{2.33}$$

$$\psi^i_{j,m,n}(x,y) = 2^{j/2}\psi(2^j x - m, 2^j y - n), \quad i = \{H,V,D\}. \tag{2.34}$$

where index i identifies the directional wavelets as a superscript.

The discrete wavelet transform of an image $f(x,y)$ of size $M \times N$ is

$$W_\phi(j_0,m,n) = \frac{1}{\sqrt{MN}}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)\phi_{j_0,m,n}(x,y), \tag{2.35}$$

$$W^i_\psi(j_0,m,n) = \frac{1}{\sqrt{MN}}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)\psi^i_{j_0,m,n}(x,y), \quad i = \{H,V,D\}, \tag{2.36}$$

where j_0 is an arbitrary starting scale and the $W_\phi(j_0,m,n)$ coefficients are an approximation of $f(x,y)$ at scale j_0; the $W^i_\psi(j_0,m,n)$ coefficients add details for scales $j \geq j_0$ in directions determined by index i.

The inverse discrete wavelet transform is given by

$$\begin{aligned}f(x,y) = &\frac{1}{\sqrt{MN}}\sum_m\sum_n W_\phi(j_0,m,n)\phi_{j_0,m,n}(x,y)\\ &+ \frac{1}{\sqrt{MN}}\sum_{i=H,V,D}\sum_{j=j_0}^{\infty}\sum_m\sum_n W^i_\psi(j,m,n)\psi^i_{j,m,n}(x,y)\end{aligned} \tag{2.37}$$

Similarly, we can expand FWT to 2-D shown in Fig.2.16. After applying 2-D wavelet

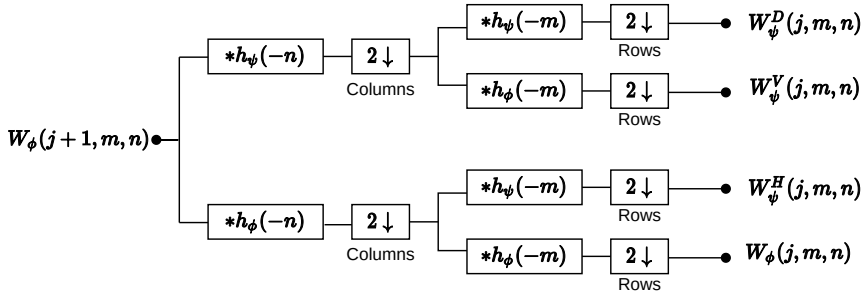


Figure 2.16: The analysis filter bank of the 2-D fast wavelet transform.

transform, the image is decomposed into 4 frequency bands: horizontal (HL), vertical (LH), diagonal (HH), and low frequency (LL) which can be further decomposed into 4 frequency bands. Now, the image energy is mainly concentrated in the low frequency part, while the energy of the horizontal, vertical and diagonal part is relatively small and has obvious directivity. As Fig.2.17 shows, we can process 2-D FWT to the approximation part (the low frequency part, which include most information of the image) of the resulting decomposition again and again.

In Fig.2.18a, we decompose the image "Lena" three times using Daubechies 4

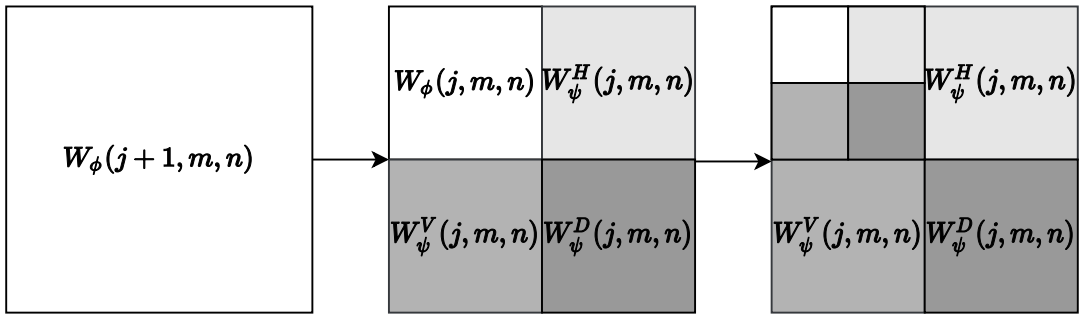


Figure 2.17: The resulting decomposition of 2-D fast wavelet transform.

wavelet function, and then reconstruct it using the multi-layer subimage obtained using decomposition shown in 2.18b.

(a) (b)

Figure 2.18: (a)Image after wavelet decomposition for 3 times, (b)Reconstructed image.

2.2.6 Implementation of image compression using wavelet transform

As DCT is one of the methods used in a block transform coding system, image needs to be divided into subimages. While wavelet transform is used in a wavelet coding system [33] [34], which doesn't require constructing subimages. Because wavelet transforms use wavelets basic functions which are limited in duration and decayable, subdivision of the original image is unnecessary, and the removal of the subdivision step eliminates the blocking artifact that characterizes DCT-based approximations at high compression ratios [30].

DWT is applied in the JPEG 2000 (JP2) image compression algorithm which is an image compression standard and coding system, and was designed to supersede the DCT with wavelet-based method. The standardized filename extension normally is .jp2.

There are some improvements over the JPEG standard:

- The pyramid representation [1, page 462-466] can offer multiple resolutions.
- By choosing the type of wavelet transform, we can choose lossless or lossy compression.
- JPEG 2000 provides efficient code-stream organizations which are progressive by pixel accuracy and by image resolution (or by image size). This way, once a smaller part of the whole file has been received, the viewer can see a lower quality version of the final picture. The quality then improves progressively through downloading more data bits from the source.

The graphic description of a wavelet coding system which contains encoder and decoder is shown in Fig.2.19.

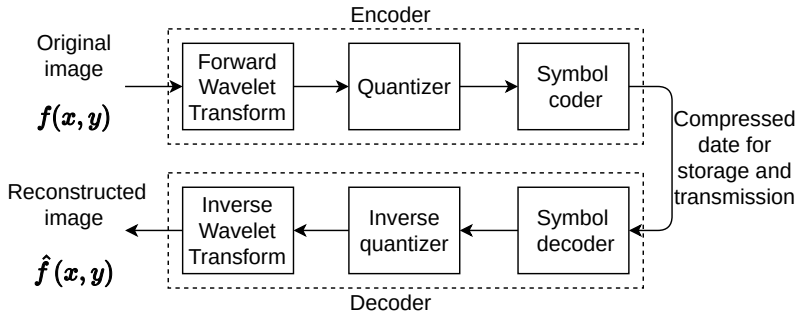


Figure 2.19: A wavelet coding system.

Step 1 Image Tiling

The term tiling refers to the partition of the original image into same sized rectangular non-overlapping tiles except the boundary blocks, and these tiles are compressed independently. After tiling, all the other steps are processed on each tile independently. Arbitrary tile sizes are allowed, up to and including the whole image.

Here is the reason why we process the tiling step on the original image firstly instead of processing directly on the whole image directly (i.e., we can also regard the whole image as one tile).

- Tiling reduces the memory requirement and computation complexity.
- Tiling allows random access of different parts of an image, so that we can decode specific parts of the image.
- We can control the coding efficiency and complexity by changing the tile size.

Step 2 DC Level Shifting

Shift all data of the tiles by subtracting the same quantity 2^{P-1}, where P is the component's precision. Thus, the range of pixel value changes from $[0, 255]$ to $[-128, 127]$, and the values become 0 centered.

At the decoder side, inverse DC level shifting is performed on reconstructed data by adding 2^{P-1} back.

Step 3 The forward 2D-DWT

Perform the forward 2D-DWT on each tiles as discussed in the previous section, by filtering the original tiles with scaling function coefficients and wavelet function coefficients and down sampling to get the four resulting quarter-size decomposition outputs (i.e., the approximation and horizontal, vertical, and diagonal details).
There are two filters that can be chosen:

- Daubechies 9-tap/7-tap filter is for lossy compression;
- Le Gall 5tap/3-tap filter is for lossless compression.

Step 4 Quantization

We recall that quantization is the process by which the coefficients are reduced in precision. A quantizer can be described as a function Q that maps each element in a subset of the real line to a particular value. Here is the formula to calculate the quantization result.

$$q_b(u,v) = sign(a_b(u,v)) \left[\frac{|a_b(u,v)|}{\Delta_b} \right], \tag{2.38}$$

where $a_b(u,v)$ denotes the coefficients of the subband b, Δ_b is the step-size relative to the dynamic range of subband b. The dynamic range is determined by the number of bits used to represent the original image tile component and by the choice of the wavelet transform, and $q_b(u,v)$ denotes the corresponding result. As Fig. 2.20 shows, the wavelet coefficients inside the interval $(-\Delta, \Delta)$ are quantized to zero, thus, the interval $(-\Delta, \Delta)$ is called the "deadzone". The JPEG 2000 standard supports separate quantization step-

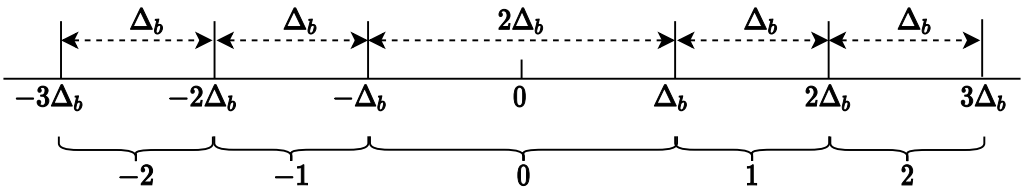


Figure 2.20: Dead-zone uniform scalar quantizer.

sizes for each subband and one quantization step-size is allowed per subband. Specifically, for lossless compression, $\Delta_b = 1$.

An example of dead-zone uniform scalar quantization is shown in Fig.2.21. $\Delta_b = 10$, given a wavelet coefficient $a = 25.35$, then the quantization result is $q_{10} = 2$.

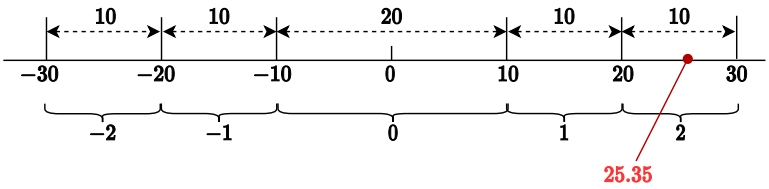


Figure 2.21: Dead-zone uniform scalar quantization example.

Step 5 Embedded Block Coding with Optimal Truncation(EBCOT)

The quantized sub-bands are split into precincts which are rectangular regions in the wavelet domain. Then precincts are split into code blocks which are in a single sub-band and have equal sized which is typically 64×64 and no less than 32×32. Each bit plane

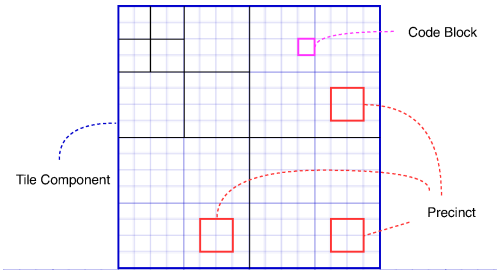


Figure 2.22: Partition of a tile component into code blocks and precincts.

of the code block will be encoded in three coding passes:

1. Significance Propagation Pass. Encoding bits of insignificant coefficients with significant neighbors.

2. Magnitude Refinement Pass. Encoding bits of significant coefficients.

3. Cleanup Pass. Encoding coefficients without significant neighbors.

The bits selected by these coding passes then get encoded by the binary MQ-coder which is a context-driven binary arithmetic coder. The result is a bit-stream that is split into

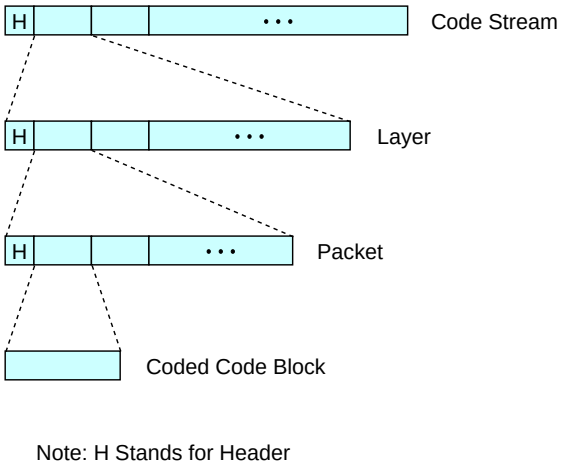


Figure 2.23: The bit stream representation.

packets where a packet groups selected passes of all code blocks from a precinct into one indivisible unit. Packets from all sub-bands are then collected in layers. This process is shown in Fig.2.23. For each code block, a separate bit stream is generated. The bit stream has the property that it can be truncated to a variety of discrete lengths which lead to a distortion, when reconstructing from each of these truncated subsets, and the distortion is estimated and denoted by the mean squared error.

Thus, we need to find the optimal packet length for all code blocks which minimizes the overall distortion in a way that the generated target bit rate equals the demanded bit rate. Rate distortion optimization is used to allocate truncation points to each code block. By applying the rate distortion optimization, a better compression can be achieved.

2.3 Summary

In this chapter, we have introduced the transforms which can provide data compression. What they have in common is that they all use the transform kernel to perform the transform and the inverse transform. We distinguish different types of transform by

different kernels.

In practice, computer only processes finite data which motivates the use of discrete transforms. In this thesis we introduced DCT and DWT. DCT is with the Fourier kernel and DWT is with the wavelet kernel. From mathematics, we know that the most simplified way to express a signal in a special space is by the orthogonal basis. The DCT basis is fixed and it is the orthonormal basis of the space $L^2(2\pi)$, while wavelet basis are multiple as there are multiple type of wavelets and we introduced how to construct wavelet function which is the orthonormal basis of the space $L^2(\mathbb{R})$.

As the wavelet basis is energy-limited in the time domain and so do the signals in reality, hence DWT is a better option than DCT for expressing signals.

CHAPTER 3

Hybrid Methods for Image Compression

As we discussed in Chapter 2, DCT has the capability of information compacting and is easy to implement, however, at the same time DCT always leads to blocking artifact at relatively high compression ratio which is one of the most annoying problems. For example, in Fig.3.1, compressed by JPEG, the decompressed image has obvious blocking artifacts. The blocking artifact occurs due to the loss of high frequency components that are eliminated during quantization. Thus, we consider the combination of DCT and other compression methods in order to provide efficiency as well as reduce the blocking artifact.

In this chapter, we present two hybrid methods for image compression: the hybrid image compression method based on DCT and DWT and the hybrid method based on fractal image compression and DCT.

(a) (b)

Figure 3.1: Implementing JPEG on the original image, (a) Original image, (b)Compressed image, $PSNR = 25.25dB, CR = 7.00$

3.1 Hybrid image compression method based on DCT and DWT

From Chapter 2, we know that using particular basis to represent an image can help to reduce the quantity of data required for the representation. We mainly introduced two transform methods for image compression: DCT and DWT. Both of these two transforms are widely applied in the domain of image compression. The DCT algorithm can concentrate most of the energy on a smaller quantity of the coefficients and offer simpler computation than other compression methods: DFT, DST, WHT and DWT [36], but it causes the block effect at higher compression ratio. The DWT algorithm can provide multi-resolution compression by discarding the detail coefficients and processing the approximate coefficients continuously, but the energy compact characteristic of DWT is lower comparing to DCT [36] and the computation process is more complicated. Therefore, hybrid methods are proposed to take advantage of both of DCT and DWT. In

this work, we want to use a hybrid method which is the combination of DCT and DWT for image compression.

Here is the main motivation for the DWT-DCT hybrid method. DWT can perform multi-resolution decomposition to divide the image into hierarchical sub-band structures with different spaces and frequencies. The energy of the image is mainly concentrated in the low-frequency part which lies on the upper left corner, while the energy of the high-frequency parts is much smaller. Therefore, we can perform DCT on the low-frequency part. Now we will explain this method in details [37].

3.1.1 The process of image compression

Step 1 Perform three level DWT to image.

By this step, the image is decomposed into 10 sub-bands which are LL3, LH3, HL3, HH3, HL2, LH2, HH2, HL1, LH1, HH1. Among them, only LL3 is the low-frequency sub-band, the others are high-frequency sub-bands in horizontal direction, vertical direction or diagonal direction. We discard the coefficients of HL1,LH1 and HH1 by setting them zeros.

Step 2 Process the rest high-frequency sub-bands

For the rest high-frequency bands LH3, HL3, HH3, HL2, LH2, HH2, quantize and eliminate zeros from each sub-band, and then compress them by Arithmetic Coding.

Step 3 Process the low-frequency sub-band LL3

Compress LL3 by DCT, then quantize and convert the coefficient matrix to an array and use Arithmetic Coding to compress it.

Step 4 Store all data in a file

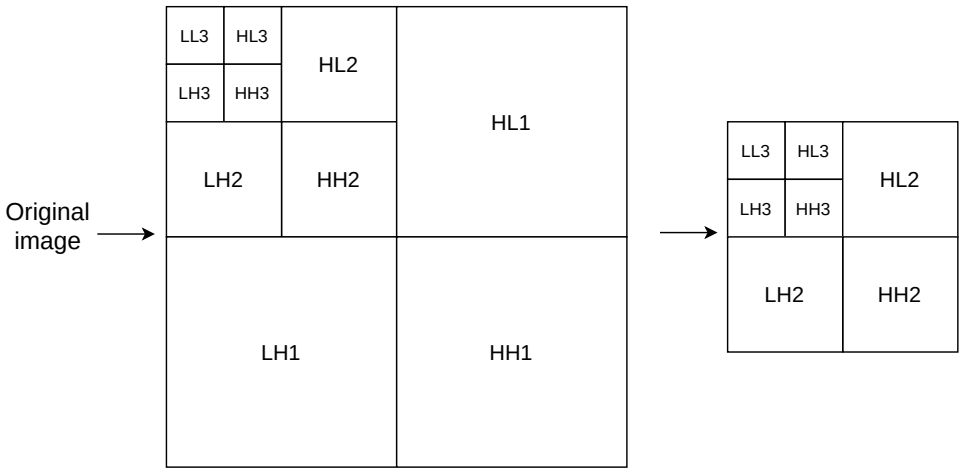


Figure 3.2: Step 1: Perform three level DWT to image.

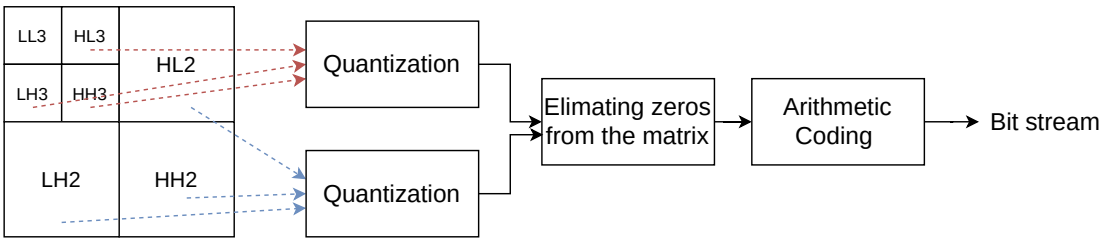


Figure 3.3: Step 2: Process the rest high-frequency sub-bands.

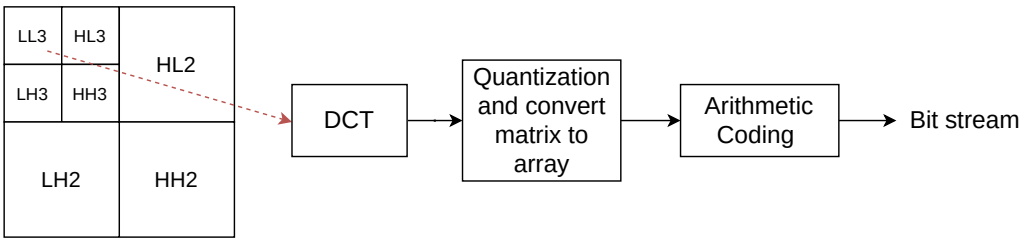


Figure 3.4: Step 3: Process the low-frequency sub-band LL3.

In this step, we save the bit stream that we get as header information and compressed data in a file.

3.1.2 The process of reconstruction

After the process of image compression, we can get the compressed data as a bit stream in a file. The file is used to store and transmit and we need to process the reconstruction to get the reconstructed image. Here is the process.

- Load the compressed data from the corresponding file.
- Read and decompress the header information with arithmetic decoding, inverse quantize the sub-bands.
- Process the inverse transformation to the coefficients respectively to get the decompressed image.

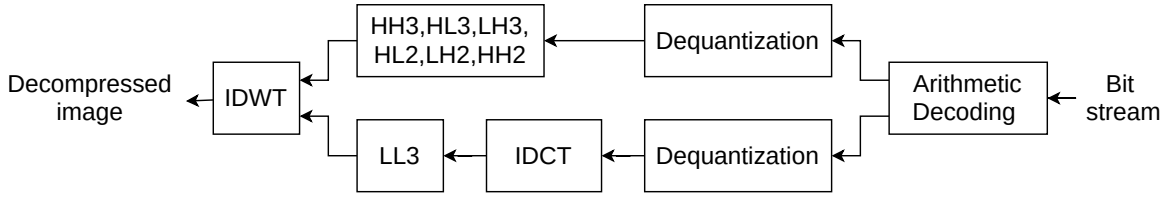


Figure 3.5: The process of reconstruction

3.2 Hybrid image compression method based on fractal image compression and DCT

In this chapter, another hybrid method which is based on fractal image compression and DCT is also proposed. Why do we combine Fractal image compression and DCT? Fractal image compression encoding can provide relatively high compression ratios, which is very attractive, however, the quality of images is sometimes very poor and it requires a long compression time. As DCT has the capability of information compacting, it is

widely used in the field of image compression. By combining DCT and Fractal image compression encoder, we sacrifice a small loss in compression ratios to gain better quality.

Before explaining the algorithm of this hybrid method, we describe the fractal image compression.

3.2.1 Fractal image compression

Fractal image compression is a relatively new technique of lossy image compression which uses Iterated Function System (IFS) to extract the self-similarity in an image to achieve the purpose of compression coding [43]. Its mathematical foundation is fractal geometry.

Fractal image compression has achieved rapid development because of its high compression ratio potential, while it also has fatal weaknesses. Although the decoding process is very fast, the encoding process is very time-consuming, which causes great restrictions on its application. Moreover, images do not necessarily have a strict fractal structure and cannot achieve the expected high compression ratio.

Fundamental of fractal image compression

The fractal image compression algorithm achieves the purpose of compression by removing the redundant part of the image according to the local self-similarity of the image itself at different scales and spaces.

Particularly, here we concentrate on IFS fractal image compression which is to use iteration to get an approximation of the original image. The algorithm is based on some notations which will be represented below.

The first notation is affine transformation through which we can skew, stretch, rotate,

scale or translate an input image. It can be denoted as

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} e \\ f \end{bmatrix} \tag{3.1}$$

where a, b, c, d, e and f are constants that determine the exact nature of the affine transformation. a, b, c and d control rotation, scaling and skew, and e, f control the translation. x, y denote the pixels of the input image and x', y' denote the pixels of the output image.

One special type of affine transformation is contraction transformation or contraction mapping.

Definition 3.1. *[39] Let (X, d) be a complete metric space [40]. Then a map $T : X \to X$ is called a **contraction mapping** on X if there exists $s \in [0, 1)$ such that*

$$d(T(x), T(y)) \leq sd(x, y)$$

for all x, y in X.

In this definition, T denotes a contraction mapping with contractivity factor s which is a constant and shows how contractive the transformation is. The effect of contraction mapping shows in Fig.3.6, which is that the mapped points are getting closer to each other. What if we iterate a contractive mapping at any point? It converges to a fixed

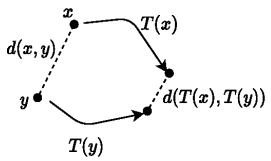


Figure 3.6: Contraction mapping

point, and this is what Banach Fixed Point Theorem states.

Banach Fixed Point Theorem. *[39] Let (X, d) be a non-empty complete metric space with a contraction mapping $T : X \to X$. Then T admits a unique fixed-point x^* in X (i.e. $T(x^*) = x^*$). Furthermore, x^* can be found as follows: start with an arbitrary element x_0 in X and define a sequence $\{x_n\}$ by $x_n = T(x_{n-1})$ for $n \geq 1$. Then $x_n \to x^*$.*

This theorem means that there is always a unique fixed point x^* of a contraction mapping T, so that after any point in the space undergoes the iteration of this contraction mapping T, the formed point sequence converges to x^*.

If a set of contraction mappings work together, they form an iterated function system.

Definition 3.2. *[41, p.82] Formally, an **iterated function system**(IFS) is a finite set of contraction mappings on a complete metric space. Symbolically,*

$$\{f_i : X \to X | i = 1, 2, \ldots, N\}, N \in \mathbb{N}$$

is an iterated function system if each f_i is a contraction on the complete metric space X.

From this definition, we can know that a finite collection of contraction mappings form an IFS. The contraction factor of an IFS is the largest of all the individual contraction factors.

Collage Theorem. *[41] Let $\mathbb{X}$ be a complete metric space. Suppose L is a non-empty, compact subset of $\mathbb{X}$ and let $\epsilon > 0$ be given. Choose an iterated function system (IFS) $\{\mathbb{X}; w_1, w_2, \ldots, w_N\}$ with contraction factor $0 \leq s < 1$. Suppose*

$$h\left(L, \bigcup_{n=1}^{N} w_n(L)\right) \leq \varepsilon$$

where $h(\cdot, \cdot)$ is the Hausdorff metric between sets. Then

$$h(L, A) \leq \frac{\varepsilon}{1 - s}$$

where A is the attractor of the IFS. Equivalently,

$$h(L, A) \leq (1-s)^{-1} h\left(L, \cup_{n=1}^{N} w_n(L)\right)$$

for all non-empty, compact subsets L of $\mathbb{X}$.

Informally, if L is close to being stabilized by the IFS, then L is also close to being the attractor of the IFS.

Implementation of fractal image compression

We follow [42] to implement fractal image compression, we take a digital binary image $A = [0,1]^{2^N \times 2^N}$ as an example.

Step 1 Image partition

Divide the image A into non-overlapping range blocks $\{R_i, i = 1, 2, \ldots, n\}$ of size $2^R \times 2^R$, let $A = \bigcup_{i=1}^{N} R_i$, and $R_i \bigcap R_j = \emptyset$, $i \neq j$. Then divide A into larger domain blocks which are not necessarily disjoint, denoted by D, with size $2^D \times 2^D \quad (D > R)$.

Step 2 Search for the best match

For any range block R_i, seek a domain block D_j, so that D_j approximates R_i through some affine transformation ω_i, that is $R_i \approx \omega_i(D_j)$. The sizes of D_j and R_i are much smaller than the entire image, therefore, as long as the sub-blocks are sufficiently small, local self-similarity always exists.

Step 3 Compression

We generate all possible affine transformations of domain blocks. Then we try all the generated domain blocks, optimize their contrast and the brightness to find a best match for each range block.Record the parameters that determine the transformation ω.

Step 4 Decompression

The decoding process start from a completely random image to which we apply the contraction several times. After a limited number of iterations, the image will stabilize, it will approach the attractor of the IFS, the attractor is edited as the decoded image of the original image.

3.2.2 The algorithm of the hybrid method

The fractal image compression method can offer a very nice compression ratio while it will consume a relatively longer time to encode. One of the main purposes of this hybrid method [43] is to reduce the encoding time.

First, we discuss why we choose fractal image compression to combine with DCT instead of other transform methods. As we know, the common point of DCT and fractal image compression is that they both implement within blocks. What's more, DCT has a very good capacity of compacting the sub-image, and it can concentrate the energy of the sub-image into the left corner. These features of DCT offer good condition for fractal image compression to implement based on it. Next, we develop the algorithm of the combination of DCT and fractal image compression.

Step 1 Partition the image

Partition the image into range blocks of size $X \times X$, then the corresponding domain blocks are of size $2X \times 2X$. The position of corresponding domain block is fixed that it expands to $2X \times 2X$ with its range block as the center, as shown in Fig.3.7. Get mean of the domain blocks by 4 neighbor pixels, then the processed domain blocks have the same size as the range blocks.

Step 2 Perform DCT on each block

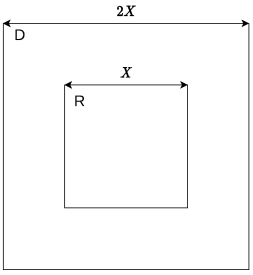


Figure 3.7: Position of range block and corresponding domain block

By doing this step, we transform the sub-images from space domain to frequency domain, and we get $F_R(u, v)$ and $F_D(u, v)$. Then save the DC coefficient of each range block which can be denoted as $F_R(1, 1)$.

Step 3 Search the best value for scaling parameter

Before searching, each range block corresponds to its own domain block and we suppose that $F_R(1, 1) = F_D(1, 1) = 0$. While searching, we should save the scaling parameter only if the error between the range block and the mapped domain block is less than a predefined threshold. Consider $E_i, i = 1, 2, 3$ as the error threshold for each level, and in this step we compare the error with the error threshold for level one E_1.

Here is the formulas for calculating the error:

$$E(R, D) = \sum_{i=1}^{n} (sd_i + o - r_i)^2$$

$$o = \overline{R} - s\overline{D}$$

s is the scaling parameter; $\overline{R}$ and $\overline{D}$ are mean of the range block and mean of the domain block; r_i and d_i are the pixel value of the range block and that of the domain block respectively.

Step 4 Perform quad-tree algorithm on the remaining range blocks

For the remaining range blocks, partition them for the second time, then they have

the size of $\frac{X}{2} \times \frac{X}{2}$, and their corresponding domain blocks which share the same centers with the range blocks are of size $X \times X$. Save $F_R(1,1)$, search like in Step 3, and save the scaling parameter s if the error is less than E_2. For the remaining range blocks, repeat the steps. If this error is smaller than corresponding predefined value E_i or the algorithm is in level 4 of quad-tree scheme, we store the contrast scaling, s and brightness, else split the range block into four equal size blocks and continue search process for them.

During the encoding process, changing the predefined thresholds E_i of each level leads to different PSNR and compression ratio. Here the range block of succeeding step have size of 16×16 , 8×8 , 4×4 and 2×2 respectively. The obvious advantage of this method compared with fractal image compression is that, for any range block, the domain block is fixed and they share the same center, so that it can reduce the encoding time efficiently without processing the real searching.

CHAPTER 4

Experimental Results and Analysis

In this chapter, we conduct several experiments to evaluate the performance of the hybrid algorithms proposed in chapter 3. The proposed algorithms are applied on several images and the obtained results are compared to results of some well known algorithms on the same sequence of images.

The experiments are performed on a CPU @ 1.10GHz×2 with an Intel(R) Celeron(R) processor. All the algorithms discussed here are implemented in Matlab which has several toolkit (e.g. "dct2", "dwt2", "jp2") dedicated to image compression. For comparison, we will use the same gray-scale image "Lena" with size $512 \times 512 \times 8 - bit$. The experiment consists of implementing the comparison methods (e.g. DCT, DWT, the technique JPEG based on DCT transform, the technique JPEG 2000 based on DWT and also the hybrid method), and use the comparison standards proposed in the first chapter (e.g. peak-signal-to-noise ratio (PSNR) and compression ratio (CR)) to measure the reconstructed images by different methods. Finally, we analyse the results of the experiments and draw the necessary conclusions.

4.1 Implementation of the compression methods

4.1.1 Implementation of the DCT

Explanation of the algorithm of DCT

First, we split the original image into small blocks of size 8×8. Then we implement DCT on each block. Here we take the small block in the upper right corner of the image as an example. As we can see from Tab.4.2, the low frequency coefficients are concentrated

48	48	53	49	53	57	51	53
47	47	55	53	50	55	52	51
48	48	57	52	54	54	50	57
54	54	48	47	53	53	50	55
46	46	50	50	52	49	50	51
43	43	50	47	54	47	50	49
44	44	55	51	54	47	50	51
44	44	55	66	54	47	50	51

Table 4.1: Original block

405	-10.8267	-11.3968	-7.0106	2.75	-0.13334	5.8031	4.9489
6.5547	-0.58421	7.1679	6.3594	-6.1965	-4.2586	5.4765	-2.8904
2.6528	-0.84955	-8.5962	-4.9226	0.20776	5.8556	1.3964	-1.7081
-8.7151	-2.5673	1.118	4.3468	-3.0874	-1.8359	-0.50688	4.2166
2.75	1.9393	1.64	2.3399	2.5	-0.28145	-2.5735	-4.4226
1.8866	0.58672	5.4275	3.0961	-0.40922	-5.6574	1.0281	2.2967
0.52478	-1.7393	-2.1036	-1.1607	1.8081	-0.83721	0.59619	0.71926
0.63035	-1.4584	-2.3707	-3.6053	0.16937	0.5561	2.3427	1.3949

Table 4.2: Block after implementing DCT

in the upper left corner and most high frequency coefficients are very close to zero so that they have the potential of requiring fewer bits to code as we will implement the quantization with Tab.2.1 in Chapter 2. Here we can multiply the quantization table

with a certain scaling factor (SF) to control quality of the image

$$Q(u,v) = round\left[\frac{F(u,v)}{SF \times S(u,v)}\right], \tag{4.1}$$

where $Q(u,v)$ means quantized coefficient magnitude, $F(u,v)$ means the DCT coefficient, $S(u,v)$ is an element of the quantization table, and SF is the scaling factor. From the

25	-1	-1	0	0	0	0	0
1	0	1	0	0	0	0	0
0	0	-1	0	0	0	0	0
-1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Table 4.3: Block after quantization when $SF = 1$

13	0	-1	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Table 4.4: Block after quantization when $SF = 2$

blocks after quantization, the representation of the coefficients should require fewer bits to code than the original ones. As the value of SF becomes greater, the proportion of 0 becomes bigger.

For decoding, we need to process the inverse quantization and then the inverse DCT on the coefficients to get the reconstructed blocks.

From the reconstructed blocks and their corresponding images with scaled colors in Fig.4.1, we can see that the change of the coefficients is more gentle, and the larger the SF is, the smoother the change is, which also means that the quality of the image is lower.

Experiment using DCT on the image

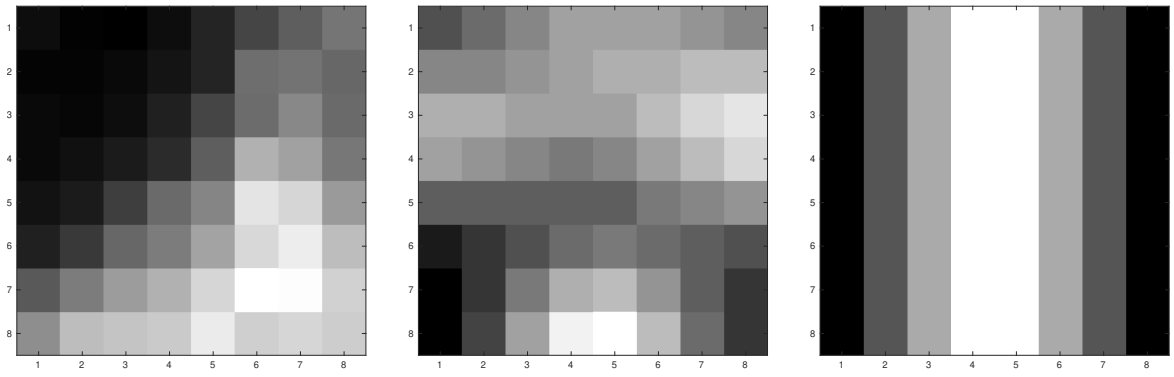
To be more clear, we process the experiment with images in Fig. 4.2. It is very obvious by human perception that the reconstructed image becomes blurrier. This also can be

46	48	50	52	52	52	51	50
50	50	51	52	53	53	54	54
53	53	52	52	52	54	56	57
52	51	50	49	50	52	54	56
47	47	47	47	47	49	50	51
42	44	46	48	49	48	47	46
40	44	49	53	54	51	47	44
40	45	52	58	59	54	48	44

Table 4.5: Reconstructed block after inverse quantization and IDCT when $SF = 1$

49	51	53	55	55	53	51	49
49	51	53	55	55	53	51	49
49	51	53	55	55	53	51	49
49	51	53	55	55	53	51	49
49	51	53	55	55	53	51	49
49	51	53	55	55	53	51	49
49	51	53	55	55	53	51	49
49	51	53	55	55	53	51	49

Table 4.6: Reconstructed block after inverse quantization and IDCT when $SF = 2$



(a) (b) (c)

Figure 4.1: Images with scaled colors corresponding to the related blocks, (a) Original block, (b) Reconstructed block when $SF = 1$, (c) Reconstructed block when $SF = 2$

shown by the measurement parameter $PSNR$ becoming smaller when SF value becomes greater. When SF reaches 10, the block effect of DCT algorithm is shown very obviously.

4.1.2 Implementation of the DWT

Explanation of the algorithm of DWT

First, we use the function 'wavedec2' to decompose the image into high frequency parts and low frequency part. In order to highlight the result, we use the block of size 16×16 pixels in the upper right corner of the original image instead of the original image itself. Then, we perform a level 1 wavelet decomposition of the block using the Haar wavelet. To show the decomposition parts of the block, we extract the level 1 approximation and

50	50	58	56	54	47	65	67	78	85	87	88	83	100	117	121
58	58	59	56	59	58	61	63	73	78	82	81	82	87	110	117
50	50	67	58	63	63	64	61	69	77	77	81	76	91	110	118
53	53	58	58	68	65	59	60	66	62	69	73	74	84	103	116
52	52	56	59	58	58	52	61	66	60	61	71	68	72	102	117
50	50	52	56	54	53	50	55	69	63	66	67	64	72	96	121
48	48	56	63	61	51	50	55	60	63	62	62	74	74	104	123
49	49	54	55	56	49	48	52	55	60	59	62	61	72	105	124
48	48	53	49	53	57	51	53	60	67	57	59	58	74	104	130
47	47	55	53	50	55	52	51	57	61	56	55	60	71	106	128
48	48	57	52	54	54	50	57	63	51	55	58	64	70	104	129
54	54	48	47	53	53	50	55	57	61	51	55	56	64	96	120
46	46	50	50	52	49	50	51	59	55	58	54	55	68	87	119
43	43	50	47	54	47	50	49	60	53	45	55	56	64	82	112
44	44	55	51	54	47	50	51	59	51	47	51	53	53	79	109
44	44	55	66	54	47	50	51	59	51	47	51	53	53	79	109

Table 4.7: Original block of size 16×16 pixels

detail coefficients, then rescale the coefficients based on their absolute values. Fig.4.4 shows very clearly that the respective characters of approximation coefficients and detail

(a) Original image

(b) $SF = 1, PSNR = 35.78dB$

(c) $SF = 2, PSNR = 33.68dB$

(d) $SF = 5, PSNR = 30.39dB$

(e) $SF = 10, PSNR = 27.33dB$

Figure 4.2: Implementing DCT on the original image with different SF value

35	47	37	70	121	142	154	253
27	57	73	60	86	109	131	237
25	42	42	37	72	78	88	228
16	46	36	26	55	61	92	245
13	30	35	28	61	45	76	255
25	25	34	32	49	38	69	239
2	19	23	21	45	32	59	196
1	45	23	23	39	18	32	175

Table 4.8: Approximation Coef. of Level 1

227	15	227	114	171	171	199	156
86	128	100	86	255	227	128	128
57	100	128	114	86	15	57	29
29	142	100	71	114	43	213	29
29	85	71	15	128	71	15	1
171	199	29	29	57	100	199	241
86	43	1	29	15	171	43	171
1	213	1	1	1	1	1	1

Table 4.9: Horizontal Detail Coef. of Level 1

1	21	33	17	50	1	91	46
1	38	13	9	17	33	103	87
1	29	5	58	50	46	50	165
1	33	70	38	33	13	46	157
1	25	38	5	46	5	112	198
1	25	1	50	33	29	58	202
1	13	42	1	46	25	87	255
1	29	58	9	66	33	1	247

Table 4.10: Vertical Detail Coef. of Level 1

1	16	96	1	32	32	192	48
1	144	48	64	192	1	80	80
1	16	16	64	1	144	64	160
1	96	48	16	32	48	176	1
1	32	16	48	48	48	80	64
1	64	1	32	255	16	32	16
1	48	64	32	48	224	80	32
1	240	1	1	1	1	1	1

Table 4.11: Diagonal Detail Coef. of Level 1

coefficients, and the size of the original block is two times of each of the decomposition part of level 1. Moreover, the image of the approximation coefficients is smoother as it contains less detail coefficients comparing to the image with scaled colors of the original block in Tab.4.3.

If we want to get the decomposition coefficients of level 2, we need to perform wavelet decomposition on the approximation coefficients of level 1, but we do not perform that here.

Now we reconstruct the image based on the decomposition coefficients. The reconstructed image is exactly the same as the original one as we don't perform the quantization step, so there is no information lost.

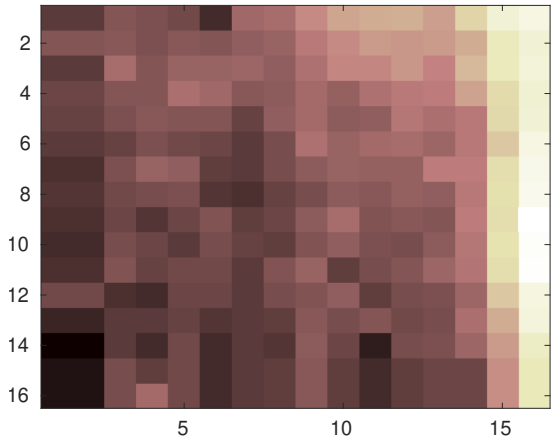


Figure 4.3: Corresponding image with scaled colors of the original block

Experiment using DWT on the image

In this experiment, we perform a level 2 wavelet decomposition of the image using the Haar wavelet. We use 'wcodemat' (a toolkit of Matlab) to rescale the coefficients based on their absolute values to make the images more visible. As we can see from Fig.4.5, the approximation coefficients become fewer and fewer as the decomposition level becomes greater and greater, so that DWT can offer multi-resolution images.

4.1.3 Implementation of the JPEG

Explanation of the algorithm of JPEG

As we discussed in Chapter 2, JPEG is a lossy image compression method based on DCT. After DCT, it performs quantization, and entropy coding which is Huffman Coding to get the compressed data. In the decompression step, we just decode the compressed data to get the reconstructed image. To measure the effect of JPEG, we use the parameters PSNR and CR.

As before, to be more specific, we use the upper right corner block of size 8×8 in the

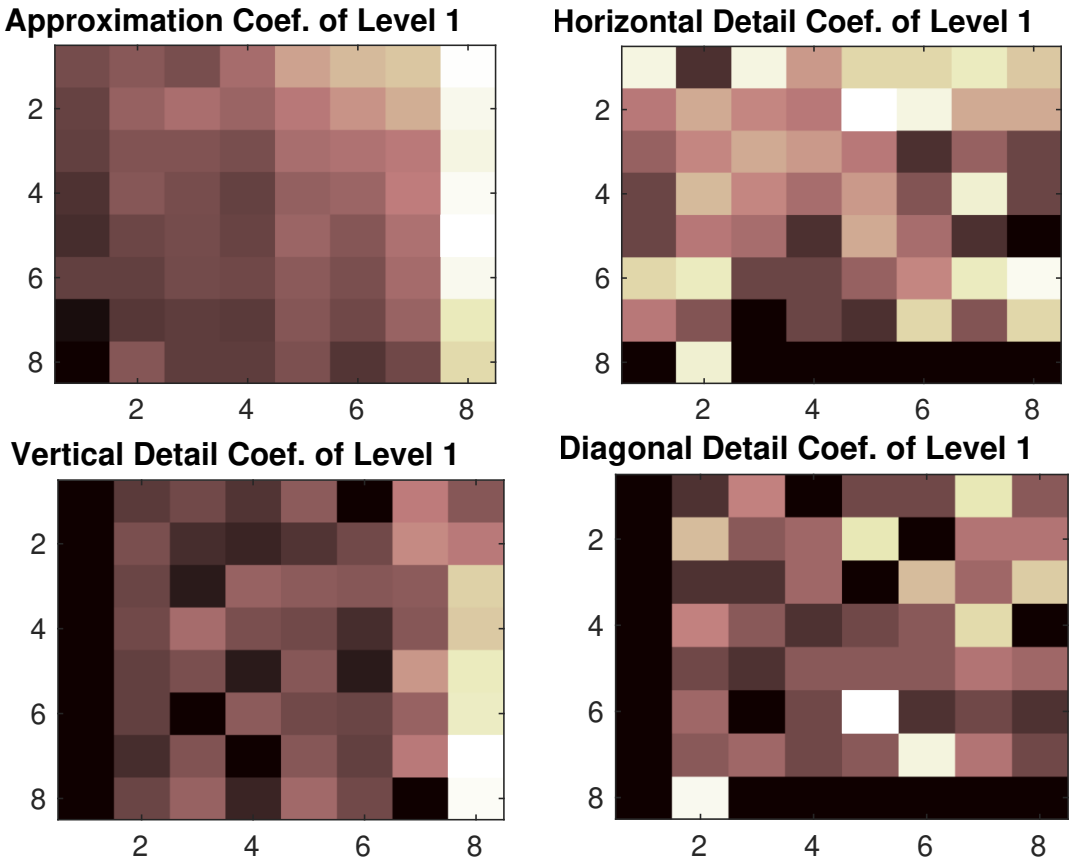


Figure 4.4: Corresponding images with scaled colors of level 1 decomposition coefficients

original image "Lena" to explain the algorithm step by step. As the block is represented in Tab.4.1, we do not represent it here.

First, we substract 128 from every pixel value in the block, the result is shown in Tab.4.12. Then, we apply DCT to the current block (Tab.4.13) and perform quantization by dividing the current block by the quantization table shown as Tab.2.1 element by elment (we can also divide by SF times the quantization table as Eq.(4.1) shows). It's this quantization step that makes JPEG a lossy compression method.

With the matrix in Tab.4.14, we can denote it by the Zigzag order we mentioned in Chapter 2 Tab.2.1. After the Zigzag step, the coefficients are denoted in one row, then we

(a) (b) (c)

Figure 4.5: DWT decomposition using Haar wavelet on image "Lena", (a) Original image, (b) Level 1 DWT decomposition, (c) Level 2 DWT decomposition

-80	-80	-75	-79	-75	-71	-77	-75
-81	-81	-73	-75	-78	-73	-76	-77
-80	-80	-71	-76	-74	-74	-78	-71
-74	-74	-80	-81	-75	-75	-78	-73
-82	-82	-78	-78	-76	-79	-78	-77
-85	-85	-78	-81	-74	-81	-78	-79
-84	-84	-73	-77	-74	-81	-78	-77
-84	-84	-73	-62	-74	-81	-78	-77

Table 4.12: After every pixel subtracting 128

can perform Huffman coding, a lossless compression method, to compress the coefficients. The last step of encoding is to save the result of Huffman coding as a file which is the compression data.

The process of decoding just performs the encoding process in reverse order which is Huffman decoding, inverse Zigzag, inverse quantization (Tab.4.15), inverse DCT (Tab.4.16) and adding back 128 for each pixel, thus we can finally get the reconstructed block (Tab.4.17). As the quantization process which makes JPEG a lossy compression method, the PSNR is 37.52 dB.

-619	-10.8267	-11.3968	-7.0106	2.75	-0.1333	5.8031	4.9489
6.5547	-0.5842	7.1679	6.3594	-6.1965	-4.2586	5.4765	-2.8904
2.6528	-0.8496	-8.5962	-4.9226	0.2078	5.8556	1.3964	-1.7081
-8.7151	-2.5673	1.118	4.3468	-3.0874	-1.8359	-0.5069	4.2166
2.75	1.9393	1.64	2.3399	2.5	-0.2814	-2.5735	-4.4226
1.8866	0.5867	5.4275	3.0961	-0.4092	-5.6574	1.0281	2.2967
0.5248	-1.7393	-2.1036	-1.1607	1.8081	-0.8372	0.5962	0.7193
0.6303	-1.4584	-2.3707	-3.6053	0.1694	0.5561	2.3427	1.3949

Table 4.13: After applying DCT on the shifted block

-39	-1	-1	0	0	0	0	0
1	0	1	0	0	0	0	0
0	0	-1	0	0	0	0	0
-1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Table 4.14: After quantization ($SF = 1$)

Experiment using JPEG on the image

After performing the JPEG code on the image in Matlab, the following results are explicit in Tab.4.17. By switching the value of SF, we can get images with different CR and PSNR. The result of the decompressed images are exactly the same with the ones in Fig.4.2. But with DCT, we didn't perform the entropy coding on the processed data, so there is no compression. For JPEG, we perform Huffman coding to compress the data and we get an array which is the compressed data, so there is parameter CR to measure the degree of compression, and it's shown in Tab.4.18. To be more specific, we use the bytes of the original image dividing by the bytes of the compressed data obtaining in the encoding process to get CR. From Tab.4.18, we see that when SF increases, CR increases and $PSNR$ decreases, that means we lose the image quality when we want to

-624	-11	-10	0	0	0	0	0
12	0	14	0	0	0	0	0
0	0	-16	0	0	0	0	0
-14	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Table 4.15: After inverse quantization

-82	-80	-78	-76	-76	-76	-77	-78
-78	-78	-77	-76	-75	-75	-74	-74
-75	-75	-76	-76	-76	-74	-72	-71
-76	-77	-78	-79	-78	-76	-74	-72
-81	-81	-81	-81	-81	-79	-78	-77
-86	-84	-82	-80	-79	-80	-81	-82
-88	-84	-79	-75	-74	-77	-81	-84
-88	-83	-76	-70	-69	-74	-80	-84

Table 4.16: After inverse DCT

46	48	50	52	52	52	51	50
50	50	51	52	53	53	54	54
53	53	52	52	52	54	56	57
52	51	50	49	50	52	54	56
47	47	47	47	47	49	50	51
42	44	46	48	49	48	47	46
40	44	49	53	54	51	47	44
40	45	52	58	59	54	48	44

Table 4.17: Reconstructed block

store or transmit the image with less compressed data. From the result, we see that JPEG

SF	$PSNR$	CR
1	35.78	5.61
2	33.68	6.13
5	30.39	6.64
10	27.33	6.89

Table 4.18: The $PSNR$ and CR results of JPEG when switching the value of SF

provides good image compression quality when the compression ratio is high. However, when the compression ratio is very close to 7, the compressed image shows an obvious block effect, which is mainly caused by the block DCT.

4.1.4 Implementation of the JPEG 2000

Explanation of the algorithm of JPEG 2000

Experiment using JPEG 2000 on the image

Unlike JPEG, JPEG 2000 use multi-analysis coding method based on wavelet transform instead of blocking coding method based on DCT.

To perform the compression with JPEG 2000, we just use the function given directly by Matlab toolbox "j2k". By setting different compression ratio, we get images with widely different quality. CR is calculated by the size in bytes of the original image divided by that of the j2k file. The results are shown in Fig.4.6. By the plots, we can see that for the condition of low and medium quality, JPEG 2000 can provide an extremely high compression ratio with relatively good quality of image; for the high quality, JPEG has a little bit higher compression ratio, while JPEG 2000 shows a higher PSNR initially. Through the trend of the curves, we can deserve that JPEG 2000 provides a finer distinction for low and medium quality. Generally, it is not difficult to see that JPEG 2000 is more advanced than JPEG for image compression in spite of its more complicated process.

4.1.5 Implementation of the the hybrid method

Explanation of the algorithm of the hybrid method

To show the process of performing the hybrid method, we use the block of size 16×16 pixels located in the upper right corner of the original image in Tab.4.3.

First, perform DWT with Haar wavelet on the original block, we can get four decomposition sub-bands of level 1 which are LL, HL, LH and HH. L stands for low

(a) Original image

(b) $PSNR = 39.16dB$

(c) $PSNR = 31.82dB$

(d) $PSNR = 29.12dB$

(e) $PSNR = 26.46dB$

Figure 4.6: Implementing JPEG 2000 on the original image with different CR, (b) $CR = 10$, (c) $CR = 50$, (d) $CR = 100$, (e) $CR = 200$

frequency, H stands for high frequency, and LL means approximation coefficients, HL means horizontal detail coefficients, LH means vertical detail coefficients and HH means diagonal detail coefficients, each of them are of size 8×8 pixels. Divide all the coefficients in LL by 2 shown in Tab.4.19, and regard all detail coefficients as zero. Perform DWT

54	57	55	64	79	85	88	116
52	60	65	61	69	75	81	112
51	56	56	55	65	66	69	109
49	57	54	51	60	61	70	114
48	53	54	52	61	57	66	117
51	51	54	53	58	55	64	112
45	49	51	50	57	53	61	100
44	57	51	51	55	49	53	94

Table 4.19: Divide all the coefficients in LL by 2

again on the new sub-band LL to get the decomposition sub-bands of level 2, LL2, HL2, LH2 and HH2, each of them are of size 4×4. Then divide all the coefficients in LL2 by 2, and perform quantization on the detail coefficients by dividing the product of the maximum coefficient in LH2 and SF (we set $SF = 0.01$), the results are shown in Tab.4.21. Again, perform DWT on the new sub-band LL2 to get the decomposition sub-bands of level 3, LL3, HL3, LH3 and HH3, each of them are of size 8×8. Then divide all the coefficients in LL3 by 2, and perform quantization on the detail coefficients by dividing the product of the maximum coefficient in LH3 and SF. For the new sub-band LL3, perform one dimensional DCT, then quantization and finally Arithmetic Coding on it. The next step is to compress each detail sub-band of level 2 and level 3 by converting them to arrays.

The final step of encoding process is to code all the information and store all the data in a file. Let the bytes of the original block divide by the bytes of this file, we can get CR.

For decoding, we first read the stored file and then perform the inverse steps in encoding.

56	61	77	99
53	54	63	91
51	53	58	90
49	51	54	77

(a)

-10	-70	200	110
10	60	100	-60
-10	-10	50	70
-70	-10	60	140

(b)

-110	-50	-120	-590
-130	40	-20	-840
-50	30	70	-990
-170	10	100	-800

(c)

50	-130	0	30
30	-20	0	40
-50	10	10	-30
90	10	-20	20

(d)

Table 4.20: Process of decomposition level 2 (a) Divide all coefficients in LL2 by 2, (b) HL2 after quantization, (c) LH2 after quantization, (d) HH2 after quantization

56	83
51	70

(a)

-250	-550
-100	-425

(b)

150	1250
100	1375

(c)

100	-150
0	225

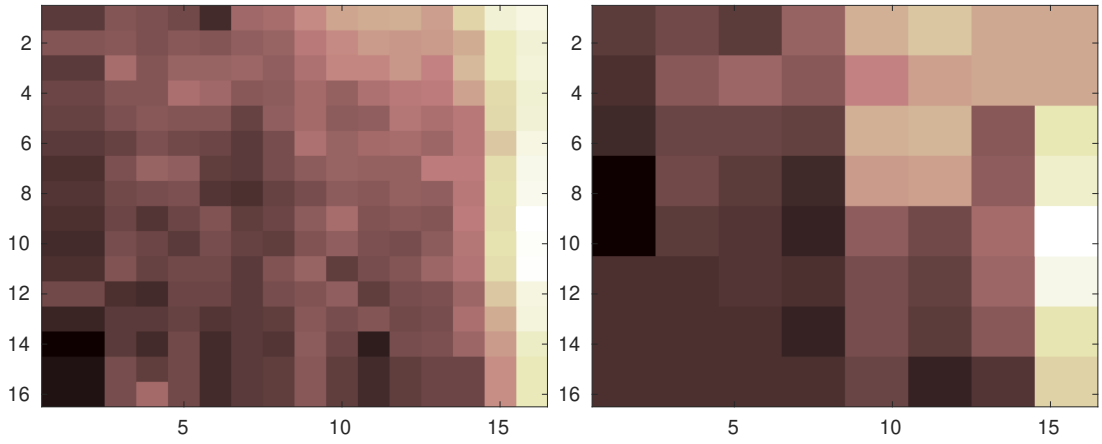
(d)

Table 4.21: Process of decomposition level 3 (a) Divide all coefficients in LL3 by 2, (b) HL3 after quantization, (c) LH3 after quantization, (d) HH3 after quantization

What is needed to remark here is that during the quantization process, we can still switch SF to control $PSNR$ and CR of the image.

Experiment using the hybrid method on the image

During the experiment, we can change the value of SF and the type of wavelets to get different results. In the final process of encoding, we store the processed data; and load the data in the beginning of decoding. The results are shown in Fig.4.8. From the results, we can see that when SF reaches 0.2, there are block artifacts due to the implementation of DCT in the reconstructed image; and when SF reaches 0.5, the block artifacts become very obvious.



(a) (b)

Figure 4.7: Implementing the hybrid method on the original block, (a) Original block, (b) Reconstructed block

4.1.6 Implementation of the Fractal image compression

As we discussed in Chapter 3, Fractal compression is a lossy image compression method based on fractals. By changing the size of range blocks or the space between domain blocks and bits for quantizing brightness, we can get multi-resolution images, shown in Fig.4.9.

	Variable parameters			Results		
	Size of Range Blocks	Space between Domain Blocks	Bits for quantizing brightness	PSNR	CR	Encoding time
b	4	2	8	30.46	14.21	3.59
c	4	2	4	25.83	16.69	4.31
d	8	2	8	25.53	56.71	3.81
e	8	2	4	23.41	66.53	2.30

Table 4.22: Related data for Fig.4.9

From Tab.4.22, we can see that when the other parameters are fixed, PSNR decreases as

the bits for quantizing brightness decreases, or as the size of range blocks increase. We can achieve a relatively high CR with fractal compression method. If we change the size of range blocks to 16, we can achieve a very high CR, but we lose the clarity of the image. In general, for fractal compression, we can only choose from the size of range blocks 4 or 8 to get a qualified image. Moreover, we will need a extremely long encoding time when process tons of images.

4.1.7 Implementation of the the hybrid method of fractal and DCT

As we discussed in Chapter 3, the hybrid method of fractal and DCT doesn't require to search for the responsive range blocks of the transformed domain blocks, so that the encoding time is reduced. For fractal compression, we change the variable parameters to get multi-resolution images; for this hybrid method, we can change the error of range block and transformed domain block to access different quality of fractal coded image.

First, we define the size of range block 16 and domain block 32. Then average 4 neighbour pixels of domain blocks so that domain blocks have the same size with the range blocks. We perform DCT and quantization to all blocks respectively. We find best match in terms of contrast, if the distance between range and domain block is more than error, then divide the range block again, all block (with size of 2×2) in stage 4 should be coded regardless distance between range and domain block.

We can see from Fig.4.10 and the related Tab.4.23 that the encoding time of the hybrid method decreases a lot and CR increases with the similar PSNR comparing to fractal compression. Now we know that the hybrid method of fractal and DCT can achieve a better result than fractal compression for the image Lena. How about other images? We will do the perform evaluation of all methods mentioned with multiple kinds of images

	Error	PSNR	CR	Encoding time
b	7	31.23	22.96	0.0419
c	10	30.42	25.06	0.031
d	15	28.78	31.32	0.03
e	20	26.91	45.70	0.02

Table 4.23: Related data for Fig.4.10

to get a result more reliable.

4.2 Performance evaluation

In this section, different methods of image compression will be evaluated and compared using certain parameters. Fifteen different gray level images of size 512×512 are picked up from [38] to be compressed and then reconstructed using JPEG, JPEG 2000, fractal image compression, the hybrid method of DCT and DWT, and the hybrid method of fractal and DCT.

To get a more reliable result, we choose the fifteen gray level images of different types which are shown in Tab.4.24, and these images used for experiments are shown in Appendix.

	Image type
Type 1	Faces
Type 2	Nature and architectures
Type 3	Objects

Table 4.24: Types of images used for experiments

To compare the performance, the values of *PSNR* are observed at a constant *CR*. Fig.4.11 shows the *PSNR* values for the original tested images at a constant *CR* around 6 by

performing JPEG, JPEG 2000, the hybrid method of DCT and DWT, fractal image compression, and the hybrid method of fractal and DCT.

During the experiment, we notice that CR of the hybrid method of fractal and DCT is mostly more than 6, so we can't compare it with the other method with the same CR. While we still want to compare the hybrid method of fractal and DCT with other methods, we show the related data in Fig.4.11 when error is fixed to 7 and the related CR is shown in Fig.4.12. We can see that for the hybrid method of fractal and DCT, even the error is fixed, the CR is various for different images, furthermore, the quality of decompressed image will not change significantly even we assign a lower value of error. Also, we notice that Fractal image compression can't reach the CR of value 6 for most tested images, and it has a big limitation when come across relatively low CR or relatively high PSNR, so we can't plot that out in this case. Generally, by observing Fig.4.11, PSNR changes in the similar trend for any tested image applying all the methods, while JPEG 2000 can achieve the highest PSNR and the other three methods reach approximative PSNR.

Another case is that we can fix the PSNR and observe the value of CR. Here we fix the value of PSNR to be around 25dB and the result is shown in Fig.4.13. From the plot, we can see that JPEG 2000 can achieve the best CR generally except for the image 9, JPEG has the lowest and stable CR around 6 to 7, Fractal image compression has the stable CR around 22 to 23, and the two hybrid methods have approximate trend. During this experiment, we notice that for some images with lots of details, applying the hybrid method of fractal and DCT can't achieve a high PSNR even if we change the error to be very small; at the same time, for some images without lots of details (for example, the faces images), applying the hybrid method of fractal and DCT can always achieve a PSNR more than 25dB even we change the error to be very big. Moreover, during the experiment, we can find that by applying the hybrid method of fractal and DCT, the

PSNR can't really represent the quality of some images (for example, the decompressed image zelda, even the PSNR is 27dB, the quality of the image is very low).

At the same time, we find out that for the hybrid method of DWT and DCT, there is a limitation for PSNR: as for the images with lots of details, the highest PSNR is limited to 23dB; for the images without lots of details, the lowest PSNR is limited to 26dB or 28dB for instance.

(a) Original image

(b) $PSNR = 31.39dB$

(c) $PSNR = 30.14dB$

(d) $PSNR = 28.67dB$

(e) $PSNR = 25.61dB$

Figure 4.8: Implementing the hybrid method of DWT and DCT on the original image with different SF value, (a) Original image, (b) $SF = 0.01, CR = 6.82$, (c) $SF = 0.1, CR = 18.88$, (d) $SF = 0.2, CR = 27.26$, (e) $SF = 0.5, CR = 42.84$

(a) Original image

(b) $PSNR = 30.46dB$

(c) $PSNR = 25.83dB$

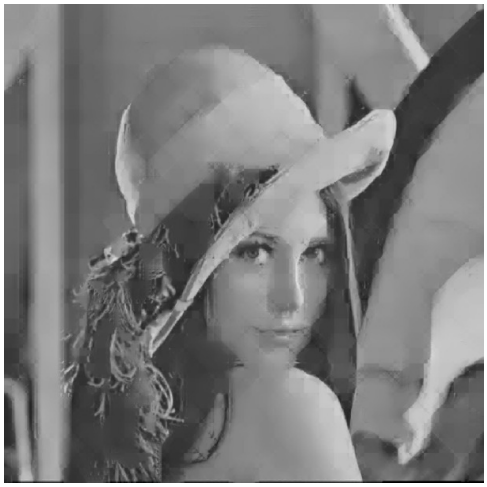
(d) $PSNR = 25.53dB$

(e) $PSNR = 23.41dB$

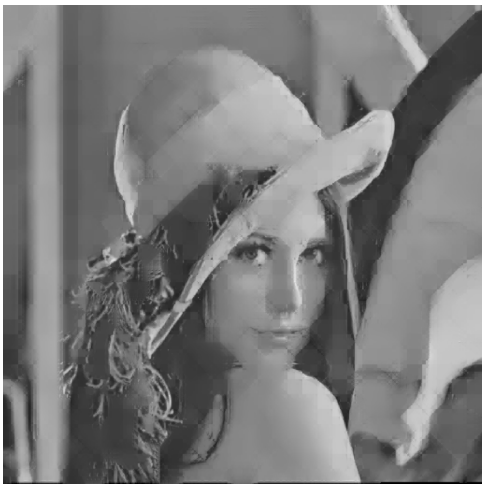
Figure 4.9: Implementing fractal compression on the original image with different values of parameters, (a) Original image, (b) $CR = 14.21$, (c) $CR = 16.69$, (d) $CR = 56.71$, (e) $CR = 66.53$

(a) Original image

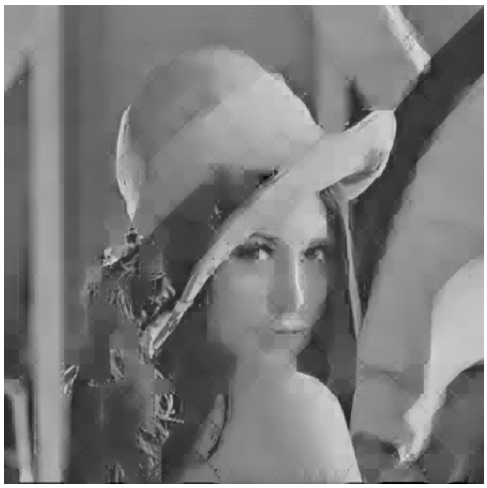
(b) $PSNR = 31.23dB$



(c) $PSNR = 30.42dB$



(d) $PSNR = 28.78dB$



(e) $PSNR = 26.91dB$

Figure 4.10: Implementing the hybrid method of fractal and DCT on the original image with different values of error, (a) Original image, (b) $CR = 22.96$, (c) $CR = 25.06$, (d) $CR = 31.32$, (e) $CR = 45.70$

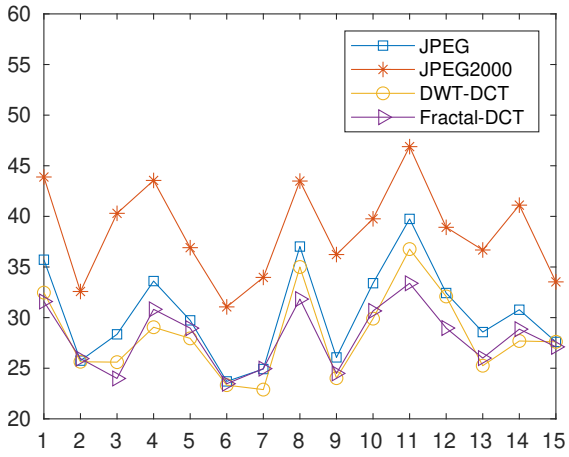


Figure 4.11: *PSNR* of the test images when *CR* is around 6, except the hybrid method of fractal and DCT

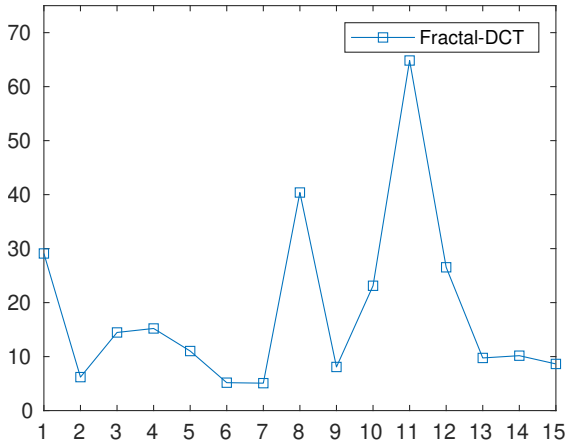


Figure 4.12: *CR* of the images using the hybrid method fractal and DCT in Fig.4.11

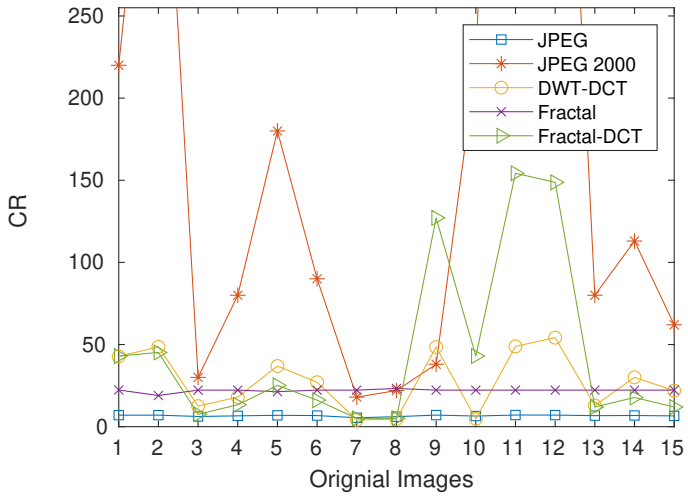


Figure 4.13: CR of the images with $PSNR$ fixed to 25dB

CONCLUSION

In Chapter 4, we carried out experiments of image compression using several methods, which are JPEG, JPEG 2000, the hybrid of DCT and DWT, fractal image compression and the hybrid of fractal and DCT. Fractal image compression can provide a good compression ratio, while it requires to spend much more time and has some limitation at a relatively high PSNR. The hybrid method of fractal and DCT reduces the encoding time successfully and provides good compression ratio and PSNR. The hybrid method of DCT and DWT has similar PSNR to JPEG when CR is fixed and provides higher CR when PSNR is fixed than JPEG, while the expense of time is also acceptable. Moreover, both hybrid methods can provide better CR at the fixed PSNR 25dB than JPEG or fractal image compression in general.

Also, we can see the big potential of the basic transforms like DCT, DWT and fractal image compression, as they can combine with other compression methods and are easy to implement. In addition, we should also consider using proper coding methods depending on the characteristics of different transforms. The software we use and how we code in details also influence the result. Taking advantage of a better algorithm or making improvements in coding can surely provide more advanced techniques in the domain of image compression.

At last, the results presented in this thesis can provide a direction for the development

of image compression. Some future work can be considered:

1. The hybrid algorithms can be further improved to reduce the encoding and decoding time.

2. The two hybrid methods have the limitation of CR and PSNR for particular images. The future work can try to break the limitation.

3. Figure out how to widely use hybrid methods and JPEG2000 in practice.

REFERENCES

[1] Gonzalez, R. C. and Woods, R. E., *Digital Image Processing*, 3rd ed. Pearson, 2007.

[2] Aguilera, P., *Comparison of different image compression formats*, ECE 533 project report, University of Wisconsin-Madison, https://homepages.cae.wisc.edu/~ece533/project/f06/aguilera_rpt.pdf.

[3] Lelewer, D. A. and Hirschberg, D. S., *Data compression*, ACM Computing Surveys (CSUR), 19(3), 1987, ACM New York, NY, USA, p.261–296.

[4] Martignon, L., *Information Theory*, International Encyclopedia of the Social & Behavioral Sciences, 2001, p.7476–7480. https://www.sciencedirect.com/science/article/pii/B0080430767006082?via%3Dihub.

[5] Saldaña, J., *An introduction to codes and coding*, The coding manual for qualitative researchers, 3, 2009, p.4. https://www.sagepub.com/sites/default/files/upm-binaries/24614_01_Saldana_Ch_01.pdf.

[6] Fenwick, P., *Information Theory*, Lecture COMPSCI 314, The University of Auckland, 2004. https://www.cs.auckland.ac.nz/compsci314s1t/resources/InfoTheory.pdf, p.1-2.

[7] Sindhu, M. and Rajkamal, R., *Images and its compression techniques-A Review*, International Journal of Recent Trends in Engineering, 2(4), p.71, 2009, Citeseer.

[8] Huffman, D. A., *A method for the construction of minimum-redundancy codes*, IEEE Proceedings of the IRE, 1952, 40(9): 1098-1101.

[9] Abramson, N., *Information theory and coding*, McGraw-Hill, p.61–62, 1963.

[10] Welch, T., *A high-performance technique for data compression*, IEEE Computer, p.8–19, 1984.

[11] Stanković, R. S. and Astola, J. T., *Reprints from the early days of information sciences*, Tampere International Center for Signal Processing, 2012.

[12] Rabbani, M., *JPEG2000: Image compression fundamentals, standards and practice*, Journal of Electronic Imaging, 2002, 11(2): 286. International Society for Optics and Photonics.

[13] Xiao, H., *Fractal Compression*, Queen's University, Canada, 2004.

[14] *Huffman Coding*, `https://www.programiz.com/dsa/huffman-coding`.

[15] Shahbahrami, A. and Bahrampour, R. and Rostami, M. S. and Mobarhan, M. A., *Evaluation of Huffman and arithmetic algorithms for multimedia compression standards*, arXiv preprint arXiv:1109.0216, 2011.

[16] *LZW compression technique*, `https://www.geeksforgeeks.org/lzw-lempel-ziv-welch-compression-technique/`.

[17] Beglaryan, G., *Lossless compression of satellite telemetry data for a narrow-band downlink*, California State University, Northridge, 2014.

[18] Tse, D., *EE/Stats 376A: Information theory, Lecture 7*, Stanford University, 2017. `https://tselab.stanford.edu/mirror/ee376a_winter1617/Lecture_7.pdf`.

[19] Haezendonck, J., *Abstract Lebesgue-Rohlin spaces*, Bull. Soc. Math. Belg, 25: 243–258, 1973.

[20] Osgood, B., *Lecture notes for EE 261: the Fourier transform and its applications*, Electrical Engineering Department, Stanford University, p. 31–32, 2013.

[21] Ran, Q. W. and Tan, L. Y. *Theory and application of wavelet transform and fractional Fourier transform*, Harbin Institute of Technology Press, p.11–13, 2002.

[22] Cheever, E., *Aperiodic Functions: From Fourier Series to Fourier Transform*, Department of Engineering,Swarthmore College. `https://lpsa.swarthmore.edu/Fourier/Series/FTFS.html`.

[23] Roberts, S., *Lecture 7 The discrete Fourier transform, Signal Processing.* `http://www.robots.ox.ac.uk/~sjrob/Teaching/SP/l7.pdf`, p.82–85.

[24] Wang, R., *Definition of DCT, Lecture E161*, Engineering, Harvey Mudd College. `http://fourier.eng.hmc.edu/e161/lectures/dct/node1.html`.

[25] *From Fourier Analysis to Wavelet Analysis*, `https://www.mathworks.com/help/wavelet/gs/from-fourier-analysis-to-wavelet-analysis.html`.

[26] Dallas, G., *Wavelets 4 Dummies: Signal Processing, Fourier Transforms and Heisenberg*, `https://georgemdallas.wordpress.com/2014/05/14/wavelets-4-dummies-signal-processing-fourier-transforms-and-heisenberg/`.

[27] Wang, R., *Discrete Wavelet Transform, Lecture E161*, Engineering, Harvey Mudd College. `http://fourier.eng.hmc.edu/e161/lectures/wavelets/node6.html`.

[28] Baraniuk, R., *Signals and Systems*, Orange Grove Books, Sampling and Reconstruction, Perfect Reconstruction, 2009.

[29] Mallat, S. G., *A theory for multiresolution signal decomposition: the wavelet representation*, IEEE transactions on pattern analysis and machine intelligence, 11(7), p.674-693, 1989, Ieee.

[30] Lai,W., *Chapter 8 Image Compression*, Department of Electrical Engineering, National Chung Cheng University, 2016. `http://www.dsp.ee.ccu.edu.tw/wnlie/File/96-IP/Lie-Gonzalez_ch8.pdf`, p.8-67.

[31] Hasan, K. and Harada, K., *Haar Wavelet based approach for image compression and quality assessment of compressed image*, IAENG International Journal of Applied Mathematics, 36:1, IJAM_36_1_9, 2007.

[32] Mallat, S., *A Wavelet Tour of Signal Processing*, Academic Press, Second Edition, 1999.

[33] Skodras, A. and Christopoulos, C. and Ebrahimi, T., *The JPEG 2000 still image compression standard*, IEEE Signal processing magazine, 18(5), p.36–58, 2001, IEEE.

[34] Rabbani, M., *JPEG2000: Image compression fundamentals, standards and practice*, Journal of Electronic Imaging, 11(2), p.286, 2002, International Society for Optics and Photonics.

[35] Stollnitz, E. and DeRose, T. and Salesin, D., *Wavelets for computer graphics: a primer*, IEEE Computer Graphics and Applications, 15(3) and 15(4), 1995.

[36] Shrestha, S., *Hybrid DWT-DCT algorithm for image and video compression applications*, A Thesis, University of Saskatchewan, Electrical and Computer Engineering Dept., Canada, Citeseer, p.29, 2010.

[37] Siddeq, M. M., *Color Image Compression*, Software Engineering Department, Technical College, Kirkuk, IRAQ, 2011. http://read.pudn.com/downloads404/sourcecode/zip/1725229/CIC2011/Readme%20First.pdf.

[38] *"C/Python/Shell programming and image/video processing/compression"*, Aug 5 2020. [Online]. Available:https://www.hlevkin.com/06testimages.html.

[39] Ciesielski, K., *On Stefan Banach and some of his results*, Banach Journal of Mathematical Analysis, 1(1), p.1–10, 2007, Tusi Mathematical Research Group.

[40] Grünbaum, B., *Some applications of expansion constants*, Pacific Journal of Mathematics, 10(1), p.193–201, Pacific Journal of Mathematics, 1960.

[41] Barnsley, M., *Fractals Everywhere*, Academic Press, 1988.

[42] Anson, L.F., *Fractal Image Compression*, BYTE, Oct. 1993, p.195–202.

[43] Salarian, M. and Hassanpour, H., *A new fast no search fractal image compression in DCT domain*, IEEE, International Conference on Machine Vision, 2007, p.62–66.

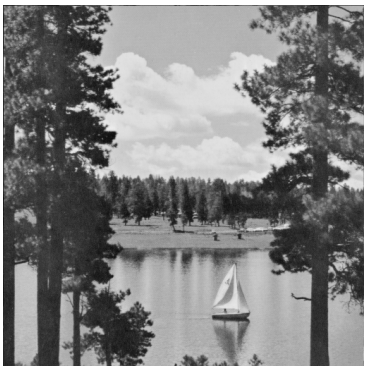
APPENDIX

1

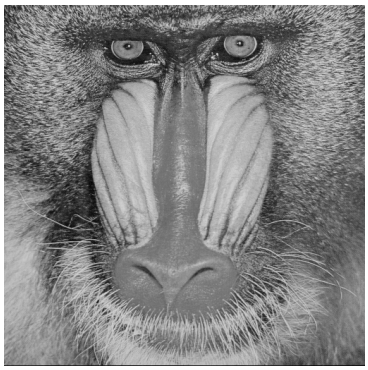
2

3

4



5



6

7

8

9

10

11

12

13

14

15