COMPUTATIONAL ASPECTS OF MATHEMATICAL MODELS IN IMAGE COMPRESSION

WENJING CHEN[1] AND WEI DUAN[2]

Abstract

Image compression is an application of data compression on digital images, which is in high demand as it reduces the computational time and consequently the cost in image storage and transmission. The basis for image compression is to remove redundant and unimportant data while to keep the compressed image quality in an acceptable range. In this paper we will introduce three different still image compression methods: Fast Fourier transform (FFT), wavelet transform (WT) and singular value decomposition (SVD). We apply these three lossy compression techniques to different images and compare their performances in terms of compression ratio, L_2-norm error, mean squared error (MSE), peak signal-to-noise ratio (PSNR) and visual quality. As the result, we get the advantages, drawbacks and potential application areas for each method.

Acknowledgements

Our deepest gratitude goes first and foremost to Mohammad Asadzadeh and Ivar Gustafsson, our supervisors, for their constant encouragement and guidance. They have walked with us through all the stages of the writing of this thesis. Without their consistent and illuminating instruction, this thesis could not have reached its present form. Second, we would like to express our heartfelt gratitude to the professors and teachers at department of Communication Engineering and department of Biomedical Engineering, they have instructed and helped us a lot in the past two years. A very special gratitude to Billy Lång for providing us the very gorgeous color picture of the bird on the front cover as well as elsewhere through all compression methods in this thesis. Last our thanks would go to our beloved families for their loving considerations and great confidence in us all through these years. We also owe our sincere gratitude to our friends and our fellow classmates who gave us their help and time in listening to us and helping us work out our problems during the difficult course of the thesis.

Key words and phrases. Image Compression, Fourier transforms, Wavelets, Singular value decomposition (SVD).

[1] Department of Biomedical Engineering, Chalmers University of Technology, SE–412 96 Göteborg, Sweden.

[2] Department of Communication Engineering, Chalmers University of Technology, SE–412 96 Göteborg, Sweden.

CONTENTS

LIST OF FIGURES

List of Tables

void!

1. Introduction

Uncompressed image data requires considerable storage capacity and transmission bandwidth. The recent growth of multimedia-based data-intensive web applications have not only sustained the need for more efficient ways to encode signals and images, but also have made compression of these data central to storage and communication technology.

A common characteristic for most digital images is that the neighboring pixels are correlated and contain redundant information. Therefore the most important task when compressing an image is to find less correlated and yet recognizable representation of the image. The algorithmic tools developed to take this approach are called image compression, which can be viewed as an early step in image processing. Two fundamental components of image compression are *redundancy* and *irrelevancy* reduction. Redundancy reduction aims at removing duplication from the signal source (image/video). Irrelevancy reduction omits parts of the signal that will not be noticed by the signal receiver, namely the Human Visual System (HVS). In general, three types of redundancy can be identified: Spatial Redundancy or correlation between neighboring pixel values, Spectral Redundancy or correlation between different color planes or spectral bands, Temporal Redundancy or correlation between adjacent frames in a sequence of images (in video applications).

Image compression research aims at reducing the number of bits needed to represent an image by removing the spatial and spectral redundancies as much as possible. Since we will focus only on still image compression, we will not worry about temporal redundancy.

The procedure of the image compression can be performed in one of the following two approaches: the lossy or lossless image compression. In this note we consider methods used in the study of the lossy approach: the crossed domains in the Table 1 below,

There are, mainly, four types of methods used to study the lossy compression:

- Reducing the color space to the most common colors in the image
- Chroma subsampling
- Transform coding
- Fractal compression

Among these methods the *Transform coding* is the one which is most widely used. In this thesis, we compare the results of image compression using three different mathematical transforms:

- Fourier transform in the form of *discrete cosine transform* (DCT) and Fast Fourier Transform (FFT)
- Wavelet transform based on *Haar, db2 and db4 wavelet basis*
- A numerical linear algebra transform presented as *singular value decomposition* (SVD).

These three techniques are applied to a variety of images for which the compression is, in one or other way, of interest in science and technology as well as in daily life. More specifically we have studied the images with application in:

Approaches \ Redundancy	Spatial	Spectral	Temporal
lossy	$\times$	$\times$	-
lossless	-	-	-

TABLE 1. The image compression procedure

I) Electromagnetic field (Coil)
II) Identification and Criminology (Finger print)
III) Medical Physics (Fungus, & MRI)
IV) Nature and environment (Wood & bird)
V) People in general (Duan)

In each of the above application areas we choose a related example as the original image, apply the three transforms to the image, compute compression degree, L_2-norm error, mean squared error (MSE), peak signal-to-noise ratio (PSNR) and visual quality, and finally compare the outcoming results.

As a general summarizing comment, we find out that FFT, an image compression procedure based on DCT, has the advantages of simplicity, with satisfactory performance, and availability of special purpose for implementation. However, the DCT is block-based leading to “blocking artifacts”, especially for low bit rates images. This is the most serious drawback in FFT. As for SVD, the quality of the compressed image is not as good as the other two approaches, but SVD is more stable so that we can save the cost in having less oscillations, which appear otherwise. Furthermore, the compression speed in SVD is also very high. We found out that among these three methods, wavelet is superior in most situations, it is the best way to compress still images and avoid most of the problem arising in FFT and SVD.

2. Mathematical modeling

2.1. **Fourier transform (FT).** Fourier transform is a useful tool for signal processing and analysis. It transfers a signal from its original ’time domain’ (or ’spatial domain’) into ’frequency domain’, describing the frequency components in the signal [6].

Before applying Fourier transform for 2D image compression, let us first take a look at the Fourier approach for one dimensional signals. Like decomposing a vector into the sum of basis vectors in Euclidean space, a signal can be projected onto a set of basis functions in frequency domain. For Fourier transform, the basis used in the frequency domain are given by $\{\cos(2\pi x\omega), \sin(2\pi x\omega)\}$, where $\omega \in \mathbb{R}$ is the frequency. We can write the basis as $e^{-2\pi i x\omega}$, since

$$e^{-2\pi i x\omega} = \cos(2\pi x\omega) - i\sin(2\pi x\omega). \tag{1}$$

Here we can see, like the standard basis in Euclidean space, that the basis functions are orthogonal to each other if they have different frequency ω, for their scalar products are all 0. For example, for integer ω_1 and ω_2, $\omega_1 \neq \omega_2$;

$$\int_0^1 cos(2\pi x\omega_1)cos(2\pi x\omega_2)dx = 0, \tag{2}$$

while for $\omega_1 = \omega_2$, $\int_0^1 cos^2(2\pi\omega_1)dx = 1/2$.
Assume $f(x)$ is a function in the space (or time for $x > 0$) domain $\mathbb{R}$, using the basis discussed above, its Fourier transform is given by

$$\mathcal{F}[f(x)] = F(\omega) = \int_{-\infty}^{\infty} f(x)e^{-2\pi i x\omega}dx. \tag{3}$$

$\mathcal{F}$ represents the Fourier transform, an integrable function, $f(x)$ is a function in space domain $\mathbb{R}$ or time domain $\mathbb{R}^+$, and the independent variable x represents space or time. $F(\omega)$ is corresponding to the function in the frequency domain with ω as frequency variable. After processing and analysis in frequency domain, the signal can be transformed back into time domain, which is called Inverse Fourier transform, given by

$$\mathcal{F}^{-1}[F(\omega)] = f(x) = \int_{-\infty}^{\infty} F(\omega) e^{2\pi i x \omega} d\omega, \tag{4}$$

As there are imaginary parts in the basis functions, the signal in the frequency domain $F(\omega)$ is complex and can be expressed as

$$F(\omega) = a(\omega) + ib(\omega) = |F(\omega)| \, e^{i\Phi(w)}, \tag{5}$$

where

$$|F(\omega)| = \sqrt{a^2 + b^2}, \qquad \Phi(\omega) = \tan^{-1}(b/a). \tag{6}$$

The absolute value of the amplitude is the Fourier spectrum, and $\Phi(\omega)$ represents phase information. Although in many applications phase information is not as important as amplitude spectrum, in image processing however, phase spectrum carries a lot of information. Here is an example, (see Figures 1 to 4).

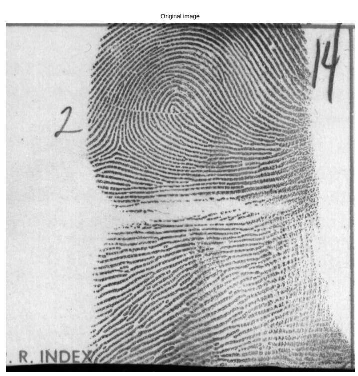

FIGURE 1. Finger Print

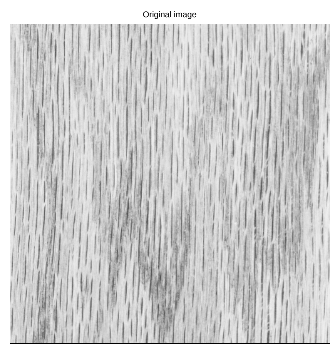

FIGURE 2. Wood

We make Fourier transform for the first two pictures, finger print and wood, extract their amplitude and phase information. Then we reconstruct the third picture with amplitude information from finger print and phase information from wood, while reconstruct the fourth picture with amplitude information from wood and phase information from finger print. From these reconstructed pictures we can see that phase information dominates the picture. This example shows that phase is as important as, or even more important than the amplitude information, in image applications.

The Fourier transform can be used as an image processing tool to decompose an image into its sine and cosine components, transforming image from its spatial domain into frequency domain with each point representing a particular frequency contained in the image. As images are two-dimensional (2D) functions, here we introduce the 2D Fourier transform,

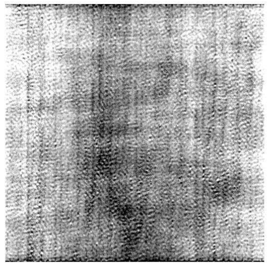

FIGURE 3. Reconstructed image with amplitude information of finger print and phase information of wood

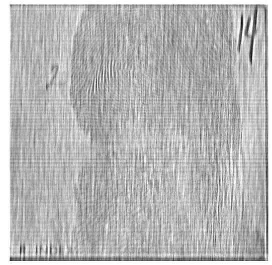

FIGURE 4. Reconstructed image with amplitude information of wood and phase information of finger print

$$\mathcal{F}(f(x,y)) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)e^{-2\pi i(x\omega+y\nu)}dxdy. \tag{7}$$

Here (x,y) are variables in a 2D space domain, and ω, ν represent the variables in the corresponding frequency domain. The 2D $inverse\ Fourier\ transform$ is given by

$$\mathcal{F}^{-1}[F(\omega,\nu)] = f(x,y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} F(\omega,\nu)e^{2\pi i(x\omega+y\nu)}d\omega d\nu. \tag{8}$$

2.1.1. **Discrete Fourier transform (DFT).** As the images we are dealing with are digital images, the signals are discrete. Then, the relevant Fourier transform is the $discrete\ Fourier\ transform$:

The discrete Fourier transform is a linear mapping that operates on N-dimensional vectors in the same way that the Fourier transform operates on functions in $\mathbb{R}$. As the image is of finite size, we approximate the Fourier transform by a finite number of algebraic operations performed on a finite set of data. First we replace the integral over $(-\infty, \infty)$ by the integral over a finite interval $[0, \Omega]$: We may assume that f vanishes outside the bounded interval $[0, \Omega]$. Thus we define

$$F(\omega) = \int_0^{\Omega} f(x)e^{-ix\omega}dx. \tag{9}$$

Using the sampling points $x = \Omega/N$ we approximate $F(\omega)$ by the Riemann sum

$$F(\omega) \approx \sum_{n=0}^{N-1} f\Big(\frac{n\Omega}{N}\Big)e^{-in\frac{\Omega}{N}\omega} \times \frac{\Omega}{N}. \tag{10}$$

The sum is periodic in ω with the period $\frac{2\pi N}{\Omega}$. Now we calculate $F(\omega)$ at the points $\omega = \frac{2\pi m}{\Omega},\ m = 0, 1, ..., N-1$:

$$F\Big(\frac{2\pi m}{\Omega}\Big) \cong \frac{\Omega}{N}\sum_{n=0}^{N-1} e^{\frac{-2\pi i n m}{N}} f\Big(\frac{n\Omega}{N}\Big), \tag{11}$$

and let $a_n = f\left(\frac{n\Omega}{N}\right)$, then we get

$$F\Big(\frac{2\pi m}{\Omega}\Big) \cong \frac{\Omega}{N}\hat{a}_m, \quad \text{where} \quad |m| << N \quad \text{and} \quad \hat{a}_m = \sum_{n=0}^{N-1} e^{-i\frac{2\pi n m}{N}} a_n.$$

We have therefore a mapping that transforms a given N-dimensional vector $\mathbf{a} = (a_0, a_1, ..., a_{N-1})$ into another N-dimensional vector $\hat{\mathbf{a}} = (\hat{a}_0, \hat{a}_1, ..., \hat{a}_{N-1})$, and the definition of "N-point discrete Fourier transform" $\mathcal{F}_N$ is given by

$$\mathcal{F}_N(\mathbf{a}) = \hat{\mathbf{a}}, \quad \text{with} \quad \hat{a}_m = \sum_{n=0}^{N-1} e^{-i\frac{2\pi n m}{N}} a_n \quad . \tag{12}$$

For a square image of size $N \times N$, the 2D DFT is defined as:

$$\mathcal{F}_N(\mathbf{a}) = \hat{\mathbf{a}}, \quad \text{and} \quad \hat{a}_{k,l} = \sum_{n=0}^{N-1}\sum_{n=0}^{N-1} e^{-i\frac{2\pi(kn+lm)}{N}} a_{n,m} \quad \text{where} \quad 0 \le k, l < N. \tag{13}$$

$\mathbf{a}$ is the image in the space domain and $\hat{a}$ is corresponding to its discrete Fourier transform. The basis functions are sine and cosine waves with increasing frequencies. The 2D Inverse Discrete Fourier transform then reads as follows:

$$f(a,b) = \frac{1}{N^2}\sum_{k=0}^{N-1}\sum_{l=0}^{N-1} F(l,k) e^{i2\pi(ka+lb)/N}, \quad n = 0, 1, ..., N. \tag{14}$$

2.1.2. **Fast Fourier transform (FFT).** Define an "elementary operation" as a multiplication of two real numbers followed by an addition of two real numbers. From the definition of $\hat{a}_m$ we have that the calculation of each $\hat{a}_m$ requires N elementary operations. There are N such $\hat{a}_m$'s, hence the calculation of all $\hat{a}_m$ requires a total of N^2 elementary operations. So the discrete Fourier transform may become computationally unmanageable for large N. To compute the DFT efficiently, here we introduce the fast Fourier transform (FFT) algorithm [1].

When N is prime, not much can be done about this. But when N is composite we can write $N = N_1 N_2$ and the indexes m and n in the definition of $\hat{a}_m$ as multiples of $N_1$and N_2 plus remainders. Let us assume that

$$m = m' N_1 + m'', \quad \text{where} \quad 0 \le m'' \le N_1 - 1 \quad \text{and} \quad 0 \le m' \le N_2 - 1$$

$$n = n' N_2 + n'', \quad \text{where} \quad 0 \le n'' \le N_2 - 1 \quad \text{and} \quad 0 \le n' \le N_1 - 1.$$

Then it follows that

$$e^{-i\frac{2\pi n m}{N}} = e^{-2\pi i\left(\frac{m'n'N_1N_2}{N} + \frac{m'n''N_1}{N} + \frac{m''n'N_2}{N} + \frac{m''n''}{N}\right)} = e^{-2\pi i\left(\frac{m'n''}{N_2} + \frac{m''n'}{N_1} + \frac{m''n''}{N}\right)}.$$

Thus we have

$$\hat{a}_m = \sum_{n''=0}^{N_2-1} C(m'', n'') e^{-2\pi i \left(\frac{m' n''}{N_2} + \frac{m'' n''}{N} \right)} \tag{15}$$

where

$$C(m'', n'') = \sum_{n'=0}^{N_1-1} e^{-2\pi i \frac{m'' n'}{N_1}} \cdot a_{n' N_2 + n''} = \sum_{n'=0}^{N_1-1} e^{-2\pi i \frac{m'' n'}{N_1}} \cdot a_n. \tag{16}$$

Each $C(m'', n'')$ requires N_1 elementary operations and there are $N_1 N_2 = N$ different $C(m'', n'')$'s, so NN_1 elementary operations are needed to calculate them all.

Then N_2 elementary operations are required to calculate each $\hat{a}_m$, ($\hat{a}_m = \sum_{n''=0}^{N_2-1} \ldots$), and there are N of those, hence NN_2 elementary operations [5] are required to compute is all $\hat{a}_m$.

The total number of elementary operations is thus $NN_1 + NN_2 = N(N_1 + N_2)$.

Suppose N_1 can be factored further, such that $N_1 = N_{11} N_{12}$. For a fixed n'', $C(m'', n'')$ is a discrete Fourier transform in m''. Then all $C(m'', n'')$ can be calculated with

$$N_2 N_1 (N_{11} + N_{12}) = N(N_{11} + N_{12}),$$

elementary operations, where N_2 is the number of n'' and N_1 is the number of m'' for a fixed n''. Totally it requires $N(N_{11} + N_{12}) + NN_2 = N(N_{11} + N_{12} + N_2)$ elementary operations, where NN_2 is the number of all $\hat{a}_m : s$.

If $N = N_1 N_2 \cdot ... \cdot N_k$, then it requires $N(N_1 + N_2 + ... + N_k)$ elementary operations. In particular, if N is a power of 2, say $N = 2^k$ it requires $2kN = 2N \log_2 N$ elementary operations. The resulting algorithm for calculating discrete transforms is called the *Fast Fourier transform, FFT*.

2.1.3. **Discrete cosine transform (DCT).** Suppose we have a periodic signal $f(x)$ with period N. In DFT, if $f(0) \neq f(N)$ we will have a discontinuity at $x = 0$ (or $x = N$), which will cause the Fourier coefficients to decay slower towards large frequencies and the packing of the coefficients is decreased. If we use the 2N-periodic even extension of $f(x)$:

$$\tilde{f}(x) := \begin{cases} f(x), & 0 \leq x < N \\ f(-x), & -N < x \leq 0, \end{cases} \tag{17}$$

then the signal will be continuous at time levels $x = -N, 0,$ and N. Due to the symmetry of $f(x)$ (even extension of f) the sine terms in the Fourier series will disappear, and the cosine terms are left. This is the concept of the discrete cosine transform (DCT) [8].

DCT is similar to DFT, but with twice the length in the spatial domain than DFT. DCT uses only real numbers and transforms a sequence of finite data into a sum of cosine functions at different frequencies.

There are 8 types of DCTs, the most common used is type-II DCT, which is referred to as the DCT. It is often used in signal and image processing, which ensures that the data are implicitly continuous at the boundaries. In this thesis we use this transform for image compression.

The one-dimensional discrete cosine transform $C(u)$ of a function f(x), with the discrete vector x of length N, is defined by

$$C(u) = \alpha(u) \sum_{x=0}^{N-1} f(x) cos[\frac{\pi(2x+1)u}{2N}], \tag{18}$$

where $u = 0, 1, 2..., N-1$. The inverse cosine transform is given by

$$f(x) = \sum_{u=0}^{N-1} \alpha(u) C(u) cos[\frac{\pi(2x+1)u}{2N}], \tag{19}$$

where $x = 0, 1, 2..., N-1$. The $\alpha(u)$ in both equations is defined as

$$\alpha(u) = \begin{cases} \sqrt{\frac{1}{N}}, u = 0 \\ \sqrt{\frac{2}{N}}, u \neq 0. \end{cases} \tag{20}$$

In the definition of DCT, $cos[\frac{\pi(2x+1)u}{2N}], u = 0, \ldots, N-1$ is the basis for the transform. Here we plot the basis for $N = 8$, see Figure 5:

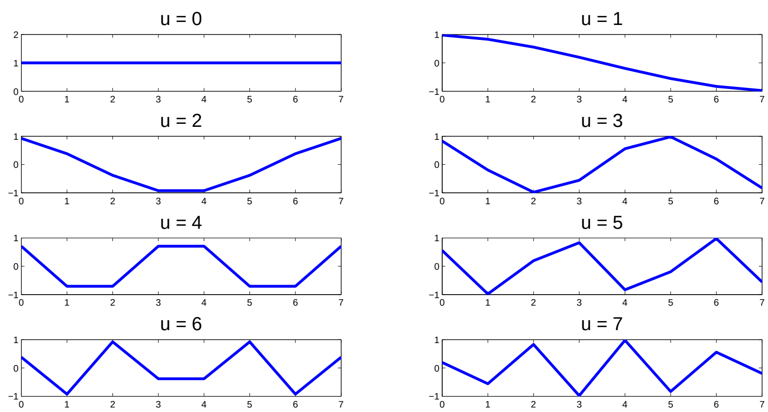

FIGURE 5. DCT basis for N = 8

The basis element corresponding to $u = 0$ is always 1 for all x, and $C_{u=0} = \sqrt{\frac{1}{N}} \sum_{x=0}^{N-1} f(x)$ is an average value of f(x). This value is called detail coefficient (DC). Other transform coefficients are referred to as the approximation coefficients [9].
Now we extend DCT into two dimensional space and define

$$C(u, v) = \alpha(u)\alpha(v) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x, y) cos[\frac{\pi(2x+1)u}{2N}] cos[\frac{\pi(2y+1)v}{2N}], \tag{21}$$

where $u, v = 0, 1, 2..., N-1$, and $\alpha(u)$ and $\alpha(v)$ are defined as $\alpha(u)$ in 1D DCT. The two-dimensional inverse transform is then given by

$$f(x,y)=\sum_{u=0}^{N-1}\sum_{v=0}^{N-1}\alpha(u)\alpha(v)C(u,v)cos[\frac{\pi(2x+1)u}{2N}]cos[\frac{\pi(2y+1)v}{2N}], \tag{22}$$

where $x, y = 0, 1, 2..., N-1$.
As we can see in the DCT definition, the 2D basis functions are generated by multiplying the horizontal 1D basis function with the vertical ones. For $N = 8$ (8×8 block), the 2D basis are shown in the chess box, in Figure 6:

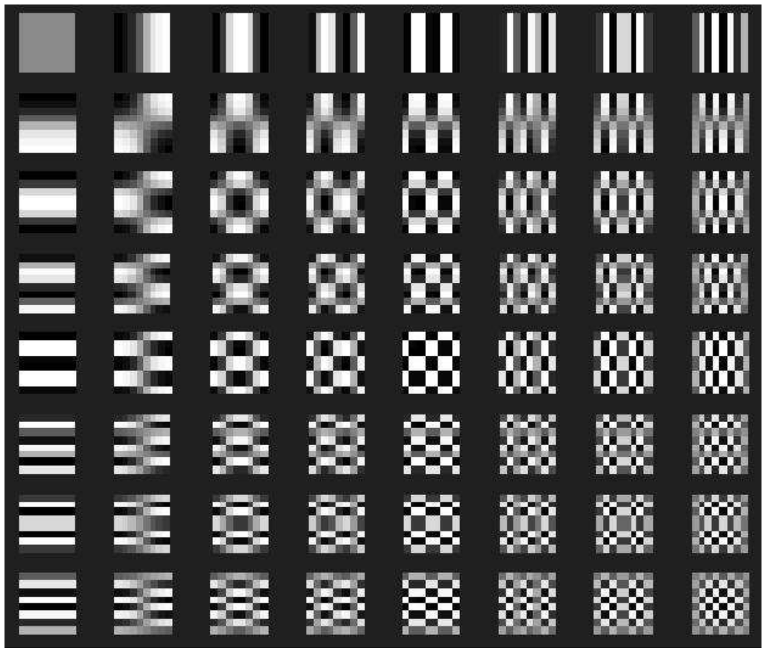

FIGURE 6. 2D DCT basis for N = 8

Similar to the 1D basis, the 2D basis on the top left is a DC component, while frequency increases both in the vertical and horizontal directions to get refined approximation coefficient components.

2.1.4. **Fourier transform in image compression.** Fourier transform is one of the most common techniques used in different imaging procedures. Based on discrete cosine transform (DCT), ISO (International Standards Organization) and IEC (International Electro-Technical Commission) have established the 'Joint Photographic Experts Group'(JPEG) standard for image compression [3].

The reason for using DCT is that it has the ability to deal with the boundary coefficients in DFT. In this thesis, we apply the DCT to each distinct 8×8 block of the 2D image, padding the image with zeros if the number of elements in the column and row are not 2^N (N is a positive integer).

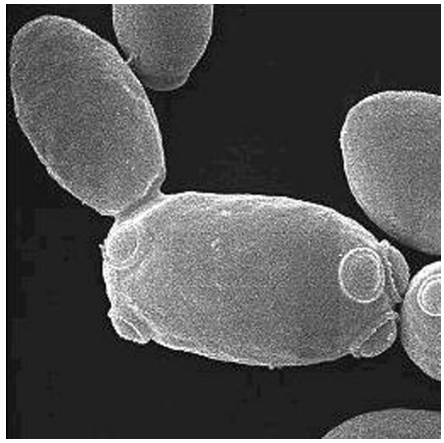

FIGURE 7. Original Fungus Image

As we can see from the original fungus image, see Figure 7, there are higher frequencies in the fungus cells, and lower frequencies in the background. So in the DCT result, see Figure 8, the cell part appears brighter than the other part, which indicates a higher frequency in the cells. Now we set a threshold and get rid of the high frequencies which representing the details of the image block by block. Then the inverse DCT is applied to the compressed matrix, and we get the compressed image in Section 5.

2.2. **Wavelet Transform.** As described above, Fourier Transform could transform a signal into the sum of infinite series of sines and cosines, which corresponds to the frequencies in the signal. However, one disadvantage of Fourier Transform is that we only know which frequencies are presented in the signal, but we don't know when the frequencies occur. Here we introduce a method, wavelet transform, which could represent both frequency and space (or time).

In the image compression field, wavelet methods has advantages over Fourier methods in the applicants where the signal contains discontinuities and sharp spikes.The wavelet-based image compression has been developed and implemented over the few past years, which has a better performance in many applications than DCT. Like DCT, wavelet transform (WT) belongs to unitary transforms, a class of transforms which are linear, invertible. Wavelet functions are defined over a finite interval with zero average value. Wavelet transform represents any signal f(t) as a superposition of a set of wavelet basis functions ('mother wavelet'). The difference between WT and DCT is that the WT has a realization is more flexible we can use any mother wavelets, which are with different properties.

2.2.1. **Continuous Wavelet Transform (CWT).** First we introduce the continuous wavelet transform (CWT). In analogy to FT, we can construct CWT as follow,

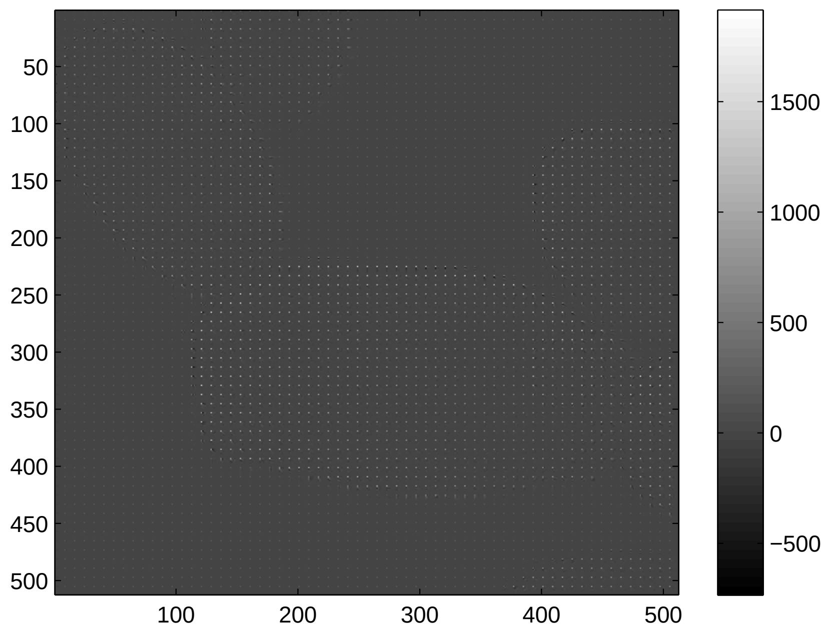

FIGURE 8. Fangus, DCT

$$\mathcal{W}(a,b) = \int_{-\infty}^{\infty} f(x)\Psi_{a,b}(x)dx, \tag{23}$$

where f(x) is the original signal, and $\mathcal{W}(a,b)$ is the signal after wavelet transform. $\Psi_{a,b}$ is a set of basis functions, called wavelets. The wavelets are generated from the 'mother wavelet', Ψ, by scaling and shift translation:

$$\Psi_{a,b}(x) = \frac{1}{\sqrt{a}}\Psi\left(\frac{x-b}{a}\right), \tag{24}$$

where a is the scale factor, and b is the translation factor. $\frac{1}{\sqrt{a}}$ is for energy normalization for different scales.

Wavelet Properties

One of the most significant feature of wavelet is that its average value in spatial domain is zero:

$$\int_{-\infty}^{\infty} \Psi(x)dx = 0, \tag{25}$$

It is a wave-like oscillation with an amplitude that starts with zero, increases, and then decreases back to zero. This is why it is called wavelet. Wavelet functions also satisfy the admissibility condition,

$$\int_{-\infty}^{\infty} \frac{\left|\hat{\Psi}(\omega)\right|^2}{|\omega|} d\omega < +\infty; \tag{26}$$

where $\hat{\Psi}(\omega)$ represents the Fourier transform of $\Psi(x)$. But we shall denote it by $\Psi(\omega)$ instead of $\hat{\Psi}$, where the frequency variable (ω) dependence indicates that it is transformed. This condition indicates that $\Psi(\omega)$ vanishes at zero frequency,

$$|\Psi(\omega)|^2\big|_{\omega=0} = 0, \tag{27}$$

which means wavelet functions have band-pass spectrums.

2.2.2. **Discrete Wavelet Transform.** Discrete wavelet transform (DWT) is wavelet transform where wavelets are discretely sampled. Note like CWT, DWT are also continuous-time transforms. CWTs operate over every possible scale and translation while DWTs can only be scaled and translated in discrete (finite) number of steps. The sampled wavelets of DWT are showed as below,

$$\Psi_{j,k}(x) = \frac{1}{\sqrt{a_0^j}} \Psi\Big(\frac{x - kb_0 a_0^j}{a_0^j}\Big), \tag{28}$$

where j and k are integers, $a_0 > 1$ is the dilation step, and b_0 is the a translation factor which depends on the dilation step. Usually we use $a_0 = 2$ and $b_0 = 1$ for dyadic sampling for both frequency axis and time axis, which makes it easier to process by computers.

If the functions $\Psi_{j,k}$ form a dense frame of $L^2(R)$, then any signal $f(x)$ of finite energy can be reconstructed.

In CWT the signals are analyzed using a set of basis functions that are related to each others by simple scaling and translation, while in DWT the transformed signal is obtained by digital filter banks with different cutoff frequencies at different scales.

As shown in Figure 9, DWT is computed by iteration of filters with rescaling. The filtering operations determine the resolution of the signal, and supersampling and subsampling operations determine the scale. The signal is denoted by the sequence $x[n]$, where n is an integer. The low pass filter is denoted by G_l while the high pass filter is denoted by H_l, where l means the level of decomposition. At each level, the high pass filter produces detail information, $d[n]$, while the low pass filter associated with scaling function producing coarse approximations, $a[n]$. This is called the Mallat algorithm or Mallat-tree decomposition, which connects the continuous-time multiresolution to discrete-time filters.

According to the Nyquists rule, if the highest frequency of the original signal is ω, the lowest sampling frequency should be 2ω multiple. Note that the band filters at each decomposition level produce signals spanning only half the frequency band, it now has a highest frequency of $\omega/2$ multiple. the signal can be sampled at a frequency of ω multiple thus discarding half the samples with no loss of information. This decimation by 2 halves the time resolution as the entire signal is now represented by only half the number of samples. The half band low pass filtering removes half of the frequencies and halves the resolution, the decimation by 2 doubles the scale. So the time resolution becomes

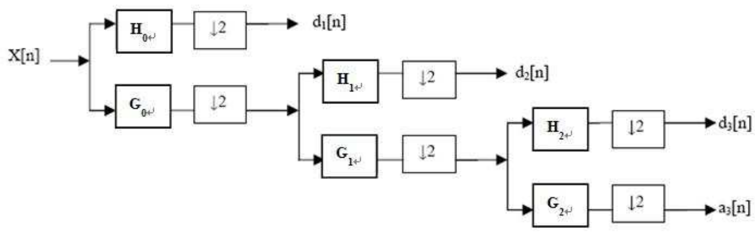

FIGURE 9. 3 level wavelet decomposition tree

arbitrarily good at high frequencies, while the frequency resolution becomes arbitrarily good at low frequencies. The filtering and decimation process is continued until the aim level is reached. The maximum number of levels depends on the length of the signal. The DWT of the original signal is then obtained by all the coefficients, $a[n]$ and $d[n]$, starting from the last level of decomposition.

The reconstruction of the original signal from the wavelet coefficients is the reverse process of decomposition, as shown in Figure 10. The approximation and detail coefficients at every level are supersampled by two, passed through the inverses low pass filters Gi and high pass filters Hi and then added. This process is continued through the same number of levels as in the decomposition process to obtain the original signal.

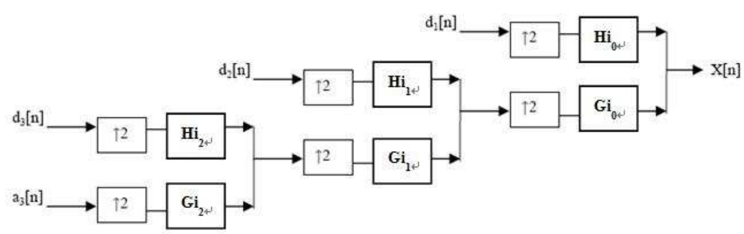

FIGURE 10. 3 level wavelet reconstruction tree

In image compression, the images are 2D signals. Assume I^0 is the original image, I^n is decomposed into a set of images A_0^{n+1} , A_1^{n+1}, A_2^{n+1} and I^{n+1}, each image is the result of a convolution operation performed between I^n and each 2D discrete filters GG, GH, HG and HH, respectively. After convolution, each image is subsampled, removing one column and one row; the result is a wavelet representation at resolution n composed by the four images. The decomposition can be done repeatedly preserving A elements and decomposing the I element.

Reconstruction algorithm starts from taking the last obtained decomposition set A_0^n , A_1^n, A_2^n and I^n. Each element is expanded introducing zeros between rows and columns.

Next, a convolution operation is performed at each image with their respective reconstruction filters GGi, GHi, HGi and HHi. At the end, image addition is done in order to obtain the I^{n-1} image. Once I^n is obtained the algorithm ends, it represents the reconstructed image.

2.2.3. **Haar Wavelet. (Haar Scaling Function and Wavelets)**
In this part,we are using Haar wavelet for image compression. The Haar wavelet is the first known wavelet, which was proposed in 1909 by Alfred Haar. It is the simplest wavelet function, but it is not continuous, which means Haar wavelet is not differentiable [7].

The two- dimensional parametrization is achieved from the function $\psi(t)$ which is called the generating or mother wavelet

(29) $$\psi_{j,k}(t) = 2^{j/2}\psi(2^j t - k), \quad j, k \in Z$$

where Z is the set of all integers and the factor $2^{j/2}$ maintains a constant norm independent of scale *j*. This parametrization of the time or space location by *k* and the frequency or scale by *j* turns out to be extraordinarily effective.

In our approach, Haar is the most important wavelet. The multiresolution formulation needs two closely related basic functions. In addition to the wavelet ψ that has been discussed, we will need another basic function called the $scaling function\, \varphi(t)$. The simplest orthogonal wavelet system is generated from the Haar scaling function and wavelet. Haar wavelet function is shown in the Figure 11.

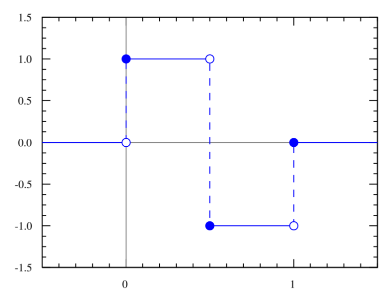

FIGURE 11. Haar wavelet

Here is an example of the Haar wavelet system which may help for a quick understanding. We choose the scaling function to have compact support over $0 \leq t \leq 1$, then we can get a simple rectangle function

(30) $$\phi(t) = \begin{cases} 1 & \text{if} \quad 0 \leq t \leq 1 \\ 0 & \text{otherwise}, \end{cases}$$

with only two nonzero coefficients $h(0) = h(1) = 1/\sqrt{2}$, which is the Haar scaling function. Another kind of Haar requires that the wavelet to be

$$\psi(t) = \begin{cases} 1 & \text{for } 0 \leq t \leq 0.5 \\ -1 & \text{for } 0.5 \leq t \leq 1 \\ 0 & \text{otherwise,} \end{cases} \tag{31}$$

with only two nonzero coefficients $h_1(0) = 1/\sqrt{2}$ and $h_1(1) = -1/\sqrt{2}$, which is Haar wavelet.

Haar Decomposition and Reconstruction Algorithms

Decomposition is the most important part of using wavelet function. We illustrate this in an example.

Lemma 1. *The following relations hold for all x∈ R:*

$$\phi(2^j x) = (\psi(2^{j-1}x) + \phi(2^{j-1}x))/2 \tag{32}$$

$$\phi(2^j x - 1) = (\phi(2^{j-1}x) - \psi(2^{j-1}x))/2. \tag{33}$$

This lemma can be used to decompose $\phi(2^j x - l)$ into its W_l-components for $l < j$. So the description of f in the example in terms of $\phi(2^2 x - l)$ is given by

$$f(x) = 2\phi(4x) + 2\phi(4x-1) + \phi(4x-2) - \phi(4x-3). \tag{34}$$

We want to decompose f into its W_1, W_0, and V_0 components. Before we do that, we should introduce the W and V components. Let V_0 be the space of all functions of the form

$$V_0 = \{\sum_{k\in Z} a_k \phi(x-k), \quad a_k \in R\} \tag{35}$$

where k can range over any finite set of positive or negative integers. $\phi(x-k)$ is discontinuous at $x = k$ and $x = k+1$, V_0 consists of all piecewise constant functions whose discontinuities are contained in the set of integers. All the elements outside the range are set zero. In this way, we can set V_1 as the space of functions of the form

$$V_1 = \{\sum_{k\in Z} a_k \phi(2x-k), \quad a_k \in R,\} \tag{36}$$

with possible discontinuities at $\{0, \pm 1/2, \pm 1, \pm 3/2, \ldots\}$.

A more general definition can be given as follows.

Definition 1. *Suppose j is any nonnegative integer. The space of step functions at level j, denoted by V_j, is defined to be the space spanned by the set*

$$\{\phi(2^j x+1), \phi(2^j x), \phi(2^x - 1), \phi(2^j x - 2)\} \tag{37}$$

over the real numbers. V_j is the space of piecewise constant functions of finite support whose discontinuities are contained in the set

$$\{\ldots, -1/2^j, 0, 1/2^j, 2/2^j, 3/2^j, \ldots\}. \tag{38}$$

Any function in V_0 is contained in V_1 , the same applies to $V_1 \subset V_2$ and so forth:

$$V_0 \subset V_1 \subset \ldots \subset V_{j-1} \subset V_j \subset V_{j+1} \ldots . \tag{39}$$

These inclusions are strict. For example, the function $\phi(2x)$ belongs to V_1 but does not belong to V_0 , because $\phi(2x)$ is discontinuous at $x = 1/2$. When j gets larger, the resolution will be finer. There is a spike of width $1/2^j$ in the function $\phi(2^j x)$. When j becomes large, the $\phi(2^j x)$ will be similar to one of the spikes of a signal which we want to remove. We have an efficient algorithm to decompose a signal into its V_j -components. To construct an orthogonal basis for V_j is a quite efficient way. But this kind of orthogonal basis of V_j is only half of the function's graph, so we need to find a way to isolate the 'spikes' which belong to V_j but not V_{j-1}. And at this point the wavelet ψ enters the picture. Let us start with j=1, V_0 is generated by ϕ and its translates, so one expects that the orthogonal complement of V_0 is generated by the translates of some functions ψ.

To construct ψ we need two components:
$\Delta\psi$ is an element of V_1 and ψ can be expressed as $\psi(x)$ = $\sum_l a_l \phi(2x - l)$ for some choice of $a_l \in R$ (note that only a finite number of the a_l are nonzero).
$\Delta\psi$ is orthogonal to V_0. This is equivalent to $\int \psi(x)\phi(x - k)dx$ = 0 for all integers k.
For example, there is a function consisting of two blocks

$$\psi(x) = \phi(2x) - \phi(2(x - 1/2)) = \phi(2x) - \phi(2x - 1) \tag{40}$$

satisfying the first requirement. In addition,

$$\int_{-\infty}^{\infty} \phi(x)\psi(x)dx = \int_0^{1/2} 1dx - \int_{1/2}^{1} 1dx = 1/2 - 1/2 = 0. \tag{41}$$

So, we can see that ψ is orthogonal to ϕ . Therefor, ψ belongs to V_1 and is orthogonal to V_0 ; ψ is called the $Haar\ wavelet$.

Definition 2. *The function of Haar is*

$$\psi(x) = \phi(2x) - \phi(2x - 1). \tag{42}$$

In other words, a function in V_1 is orthogonal to V_0 if and only if it is of the form $\Sigma_k a_k \psi(x - k)$. Let W_0 be the space of all functions of the form

$$\sum_{k \in Z} a_k \psi(x - k), \quad a_k \in R \tag{43}$$

and assume that only a finite number of the a_k are nonzero. W_0 is the orthogonal complement of V_0 in V_1 or we can say V_1 = $V_0 \oplus W_0$, here $\oplus$ means that V_0 and W_0 are orthogonal to each others. In this way, more general results can be established.

Theorem 1. *Let W_j be the space of functions of the form*

$$\sum_{k \in Z} a_k \psi(2^j x - k), \qquad a_k \in R, \tag{44}$$

W_j is the orthogonal complement of V_j in V_{j+1} and

$$V_{j+1} = V_j \oplus W_j . \tag{45}$$

According to this theorem, we can get

$$V_j = W_{j-1} \oplus V_{j-1} = W_{j-1} \oplus W_{j-2} \oplus V_{j-2} = \cdots = W_{j-1} \oplus W_{j-2} \oplus \cdots \oplus W_0 \oplus V_0. \tag{46}$$

So each f in V_j can be decomposed uniquely as a sum

$$f_j = w_{j-1} + w_{j-2} + \cdots + w_0 + f_0 \tag{47}$$

When j goes to infinity, there is a limiting theorem.

Theorem 2. *The space $L^2(R)$ can be decomposed as an infinite orthogonal direct sum*

$$L^2(R) = V_0 \oplus W_0 \oplus W_1 \oplus \cdots . \tag{48}$$

In particular, each f $\in L^2(R)$ can be written uniquely as

$$f = f_0 + \sum_{i=0}^{\infty} w_j, \tag{49}$$

where f_0 belongs to V_0 and w_j belongs to W_j.

This result can be seen as

$$f = f_0 + \lim_{N\to\infty} \sum_{j=0}^{N} w_j \tag{50}$$

Now we return to the previous example. Using of the equation

$$f(x) = 2\phi(4x) + 2\phi(4x-1) + \phi(4x-2) - \phi(4x-3), \tag{51}$$

We decompose f into its W_1, W_0, and V_0 components. So we can get

$$\begin{aligned} \phi(4x) &= (\psi(2x) + \phi(2x))/2 \\ \phi(4x-1) &= (\phi(2x) - \psi(2x))/2 \\ \phi(4x-2) = \phi(4(x-1/2)) &= (\psi(2(x-1/2)) + \phi(2(x-1/2)))/2 \\ \phi(4x-3) = \phi(4(x-1/2)-1) &= (\phi(2(x-1/2)) - \psi(2(x-1/2)))/2. \end{aligned} \tag{52}$$

Inserting these equations in the previous one and collecting terms yields

$$\begin{aligned} f(x) &= [\psi(2x) + \phi(2x)] + [\phi(2x) - \psi(2x)] \\ &\quad + [\psi(2x-1) + \phi(2x-1)]/2 - [\phi(2x-1) - \psi(2x-1)]/2 \\ &= \psi(2x-1) + 2\phi(2x). \end{aligned} \tag{53}$$

Here the W_1 - component of $f(x)$ is $\psi(2x-1)$, since W_1 is the linear span of $\{\psi(2x-k);\ k \in Z\}$. And the V_1 - component of $f(x)$ is $2\phi(2x)$. We also can use the equation $\phi(2x) = (\phi(x) + \psi(x))/2$ to decompose more into V_0 - component and W_0 - component. The final result is

$$f(x) = \psi(2x-1) + \psi(x) + \phi(x). \tag{54}$$

That means the components of f should be

$$W_1 = \psi(2x-1)$$
$$W_0 = \psi(x) \tag{55}$$
$$V_0 = \phi(x)$$

We summarize the previous decomposition scheme in the following theorem.

Theorem 3. (Haar Decomposition) *Suppose*

$$f_j(x) = \sum_{k \in Z} a_k^j \phi(2^j x - k) \quad \in V_j. \tag{56}$$

Then f_j can be decomposed as

$$f_j = w_{j-1} + f_{j-1} \tag{57}$$

where

$$\begin{aligned} w_{j-1} &= \sum_{k \in Z} b_k^{j-1} \psi(2^{j-1}x - k) \;\in\; W_{j-1} \\ f_{j-1} &= \sum_{k \in Z} a_k^{j-1} \phi(2^{j-1}x - k) \;\in\; V_{j-1} \end{aligned} \tag{58}$$

with

$$b_k^{j-1} = \frac{a_{2k}^j - a_{2k+1}^j}{2}, \qquad a_k^{j-1} = \frac{a_{2k}^j + a_{2k+1}^j}{2}. \tag{59}$$

This process can be repeated for $j-1$ to decompose f_{j-1} as $w_{j-2} + f_{j-2}$. In this way, we get the decomposition

$$f_j = w_{j-1} + w_{j-2} + \cdots + w_0 + f_0. \tag{60}$$

Finally, we can summarize all as follows: a signal is first discredited which produces an approximate signal $f_j \in V_j$. Then the decomposition algorithm can produce a decomposition of f_j into its various frequency components: $f_j = w_{j-1} + w_{j-2} + \cdots + w_0 + f_0$.

Reconstruction

Our goal is image compression. To this approach after decomposing a signal f into its V_0 - and W_j - components, the W_j - components that are small enough can be removed without significant changes in the original signal. The information that we need to transmit is only the significant W_j - components, and significant data compression can be achieved. The size of 'small' components depend on the tolerance for error for a particular application.

In order to rebuild the compressed or filtered signal in terms of the basis elements $\phi(2^j x - l)$ of V_j , we need a reconstruction algorithm using

$$f(x) = \sum_{l \in Z} a_l^j \phi(2^j x - l). \tag{61}$$

That means we can rewrite the signal f as a linear combination of step functions with amplitudes a_l^j over the intervals $l/2^j \leq x \leq (l+1)/2^j$. Now we assume a signal of the form

$$f(x) = f_0(x) + w_0(x) + \cdots + w_{j-1}(x), \quad w_l \in W_l \tag{62}$$

where

$$f_0(x) = \sum_{k\in Z} a_k^0 \phi(x-k) \in V_0 \;\; and \;\; w_l = \sum_k b_k^l \psi(2^l x - k) \in W_l \tag{63}$$

for $0 \leq l \leq j-1$. There are two equations

$$\phi(x) = \phi(2x) + \phi(2x-1), \psi(x) = \phi(2x) - \phi(2x-1). \tag{64}$$

which follow from the definitions of ψ and ϕ. Now we replace x by $2^{j-1}x$ to get

$$\phi(2^{j-1}x) = \phi(2^j x) + \phi(2^j x - 1), \psi(2^{j-1}x) = \phi(2^j x) - \phi(2^j x - 1). \tag{65}$$

In this way, we have

$$f_0(x) = \sum_{k\in Z} a_k^0 \phi(x-k) = \sum_{k\in Z} a_{2k}^0 \phi(2x-2k) + a_{2k+1}^0 \phi(2x-2k-1) \tag{66}$$

So

$$f_0(x) = \sum_{k\in Z} \hat{a}_l^1 \phi(2x-l), \text{where} \;\; \hat{a}_l^1 = \begin{cases} a_{2k}^0, & \text{if } l = 2k, \\ a_{2k+1}^0, & \text{if } l = 2k+1. \end{cases} \tag{67}$$

In a similar way, $w_0 = \sum_k b_k^0 \psi(x-k)$ can be written as

$$w_0(x) = \sum_{l\in Z} \hat{b}_l^1 \phi(2x-l), \text{where} \;\; \hat{b}_l^1 = \begin{cases} b_{2k}^0, & \text{if } l = 2k, \\ b_{2k+1}^0, & \text{if } l = 2k+1. \end{cases} \tag{68}$$

Hence, we can get a formula of the form

$$f_0(x) + w_0(x) = \sum_{l\in Z} a_l^1 \phi(2x-l), \text{where} \;\; a_l^1 = \hat{a}_l^1 + \hat{b}_l^1 = \begin{cases} a_k^0 + b_k^0, & \text{if } l = 2k \\ a_k^0 - b_k^0, & \text{if } l = 2k+1. \end{cases} \tag{69}$$

According to the form of the signal, the next step is to get $w_1 = \sum_k b_k^1 \psi(2x-k)$, and add it to the sum in the same way as above, i.e.

$$f_0(x) + w_0(x) + w_1(x) = \sum_{l\in Z} a_l^2 \phi(2^2 x - l), \text{where} \;\; a_l^2 = \begin{cases} a_k^1 + b_k^1, & \text{if } l = 2k \\ a_k^1 - b_k^1, & \text{if } l = 2k+1. \end{cases} \tag{70}$$

Here the a_l^1- coefficient is determined by the a_l^0- and b_l^0- coefficient. Then the a_l^2- coefficients is determined by the a_l^1- and b_l^1- coefficients, and so on in a recursive manner. The previous reconstruction algorithm can be summarized in the following theorem.

Theorem 4. *Haar Reconstruction, Suppose*

$$f = f_0 + w_0 + w_1 + w_2 + \cdots + w_{j-1} \tag{71}$$

with

(72)
$$f_0(x) = \sum_{k\in Z} a_k^0 \phi(x-k) \in V_0, \quad w_{j'}(x) = \sum_{k\in Z} b_k^{j'} \psi(2^{j'}x-k) \in W_{j'}, \quad 0 \le j' \le j.$$

Then

(73)
$$f(x) = \sum_{l\in Z} a_l^{j'} \phi(2^j x - l) \in V_j$$

where the $a_l^{j'}$ *are determined recursively for* $j' = 1$, *then* $j' = 2$, *and so on until* $j' = j$, *using the algorithm*

(74)
$$a_l^{j'} = \begin{cases} a_k^{j'-1} + b_k^{j'-1}, & \text{if } l = 2k, \\ a_k^{j'-1} - b_k^{j'-1}, & \text{if } l = 2k+1. \end{cases}$$

Summary

A format in a step-by-step procedure used to process a given signal, we let ϕ and ψ be the Haar scaling function and wavelet.

Step.1 $Sample.$

If the signal is analog, $y = f(t)$, where t represents time, set $j = J$ as the top level, so that 2^J is larger than the Nyquist rate for the signal. Get $a_k^J = f(k/2^J)$. In fact, the range of k is a finite interval determined by the duration of the signal, i.e. if the duration of the signal is $0 \le t \le 1$, then the range of k will be $0 \le k \le 2^J - 1$.

If the signal is discrete, then this step is not necessary. We can set the top level a_k^J as the $k_t h$ term in the sampled signal, then the sampling rate will be 2^J. But in any case, we have the highest-level approximation of f given by

(75)
$$f_J(x) = \sum_{k\in Z} a_k^J \phi(2^J x - k)$$

Step.2 $Decomposition.$

We use the decomposition algorithm and to decompose f_J into

(76)
$$f_J = w_{J-1} + \cdots + w_{j-1} + f_{j-1} + \ldots + w_0 + f_0,$$

where

(77)
$$w_{j-1} = \sum_{l\in Z} b_l^{j-1} \psi(2^{j-1}x - l), f_{j-1} = \sum_{l\in Z} a_l^{j-1} \phi(2^{j-1}x - l).$$

The coefficients a_l^{j-1} and b_l^{j-1} are determined by the algorithm

(78)
$$a_l^{j-1} = DL(a^j)_k, \qquad b_l^{j-1} = DH(a^j)_k,$$

where H and L are the high-pass and low-pass filters. When $j = J$, a_k^J determines a_k^{J-1} and b_k^{J-1}. Then for $j = J-1$, a_k^{J-1} determines a_k^{J-2} and b_k^{J-2}. Then j becomes $J-2$, and so on, until there are too few coefficients to continue. Or otherwise stated, the decomposition algorithm will continue until the level $j = 0$.

Step.3 *Processing.*
After decomposition, the signal will be of the form

$$f_J(x) = \sum_{j=0}^{J-1} w_j + f_0 = \sum_{j=0}^{J-1}\left(\sum_{k\in Z} b_l^j \psi(2^j x - k)\right) + \sum_{k\in Z} a_k^0 \phi(x-k). \tag{79}$$

Now the signal can be filtered by modifying the wavelet coefficients b_k^j . To filter out all high frequencies, all the b_k^j would be set to zero for j above a threshold. Maybe there is only a certain segment of the signal corresponding to particular values of k to be filtered. Our goal is data compression, then the b_k^j that are below a certain absolute value would be set to zero.

Step.4 *Reconstruction.*
To take the modified signal, f_J, we can reconstruct it as

$$f_J = \sum_{k\in Z} a_k^J \phi(2^J x - k). \tag{80}$$

We use the reconstruction algorithm

$$a^j = \tilde{L}Ua^{j-1} + \tilde{H}Ub^{j-1}, \tag{81}$$

for $j = 1, \cdots J$. When $j = 1$, a_k^1 is obtained from a_k^0 and b_k^0. For $j = 2$, the a_k^1 and b_k^1 can be computed from a_k^2 and so forth. When j has reached the top level, a_k^J represents the approximate value of the processed signal at $x = k/2^J$.

2.2.4. **Daubechies wavelet.** Daubechies wavelets (dbN) are a family of orthogonal wavelets, named after Ingrid Daubechies. N is the order. Some authors also use 2N instead of N.

With a given support width, the Daubechies wavelets have the maximal number of vanishing moments. It is impossible to write down these wavelets in an explicit expression, except for db1, which is the Haar wavelet discussed before. This is so because they are not defined in terms of resulting scaling and wavelet functions. db1-db10 are the most commonly used Daubechies wavelets. Here are the wavelet functions Ψ :

Each wavelet has a number of vanishing moments equal to the number of coefficients, which is also the order N of dbN. The vanishing moments are the number of zeros at π of z-transformed coefficients. Actually N determines the accuracy of the wavelet. Because wavelet order N means that the polynomial signal up to order N-1 can be represented completely in scaling space, while when the order is equal or larger than N, the coefficients of the polynomial will be zero. For example, db4 represents a polynomial signal with 4 coefficients, and db8 encodes the signal with 8 coefficients. So large order (more vanishing moments) means the wavelet can represent more complex signals with higher accuracy, see Figure 12.

Though the dbN wavelets ($N > 1$) are not explicit, however, the square modulus of the transfer function of h can be expressed as following,

$$P(y) = \sum_{k=0}^{N-1} C_k^{N-1+k} y^k, \tag{82}$$

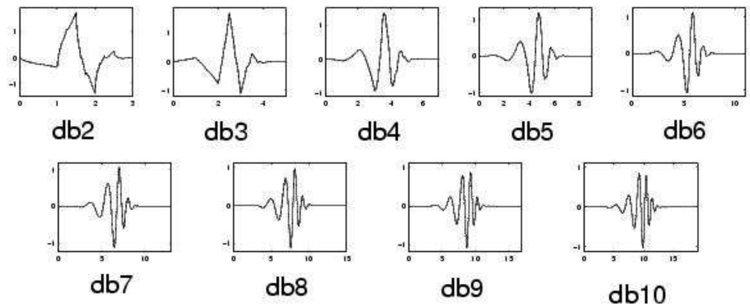

FIGURE 12. db2-db10, Image comes from MATLAB Help

where $P(y)$ means the polynomial signal, and C_k^{N-1+k} is the binomial coefficients,

$$|m_0(\omega)|^2 = |(cos^2(\frac{\omega}{2}))^N P(sin^2(\frac{\omega}{2}))|, \tag{83}$$

where

$$m_0(\omega) = \frac{1}{\sqrt{2}} \sum_{k=0}^{2N-1} h_k e^{-ik\omega}. \tag{84}$$

2.2.5. **Wavelet in Image Compression.** Wavelet-based compression has basis functions with variable length, and does not block the input image. This property leads to a kind of compression with higher compression ratio while avoiding blocking artifacts. Furthermore, it is more robust under transmission and decoding errors, and also facilitates progressive transmission of images. Because of all these advantages, the JPEG-2000 standard prescribes wavelet-based compression algorithms [2].

2.3. **Singular Value Decomposition (SVD).** The decomposition has been known since the late 19th century, and is wildly used in signal processing and statistics. Many methods have been given to decompose a matrix into more useful elements. One of the most popular factorization has been the singular value decomposition (SVD), which can be applied to both real and complex rectangular matrices. It is one of the most useful tools of linear algebra, it is a factorization and approximation technique. The SVD works wonderfully with both under - and over - determined matrices.

Let A denote an $m \times n$ matrix of real-valued or complex-valued data with rank r. Here the rank r is the maximal number of linearly independent rows or columns of A, which is at most min (m, n). Then the real-valued matrix could be presented in the form

$$A_{m\times n} = U_{m\times m} S_{m\times n} V_{n\times n}^T, \tag{85}$$

where U denotes an $m \times m$ orthogonal matrix that is $\mathrm{U}^T\mathrm{U} = \mathrm{I}_{m\times m}$, with I being the $m \times m$ identity matrix. S is a $m \times n$ diagonal matrix with nonnegative real numbers, and the matrix $V^T$is the transposed matrix of the $n \times n$ orthogonal matrix V that is $(V^TV = I_{n\times n})$. This factorization is called the singular-value decomposition of A. That means the matrix can be decomposed as the product of three matrices, see equation (85).

The diagonal elements of S are ordered in a non-increasing way, and S is uniquely determined by A. The diagonal entries of S are called singular values of A. However, the matrices U and V are not uniquely determined by A. The columns of U is a set of orthogonal 'output' basis vector directions for A, which is called the left singular vectors; and the rows of V^T form a set of orthogonal 'input' basis vector, called the right singular vectors.

2.3.1. **SVD in Image Compression.** In linear algebra, SVD is a very powerful technique dealing with sets of equation or matrices that are either singular or numerically very close to singular. It is an important factorization of a rectangular matrix, which can be applied in image compression. It has also several applications in signal processing and statistics [4].

In this project, there are several steps that should be carefully performed in order to successfully compress an image with SVD. Firstly, we set an $m \times n$ pixel image as an $m \times n$ matrix A. In particular, we illustrate SVD with low-rank approximations of the original image. An $m \times n$ image is an $m \times n$ matrix, where the entry (i, j) is interpreted as the brightness of pixel (i, j). This means that the matrix entries are interpreted as pixels ranging from black (0) through various shades of gray to white (1). It can present a colorful image too.

Let $A = USV^T$ be the SVD of A. We write

$$U = [u_1, u_2, \ldots, u_m], \text{and } \; V = [v_1, v_2, \ldots, v_n]. \tag{86}$$

so that the matrix A could be written as

$$A = USV^T = \sum_{i=1}^{n} \sigma_i u_i v_i^T. \tag{87}$$

Since $\sigma_j = 0$ for $j > r$ where r is the rank of the matrix A, we may define a compact SVD as

$$A = \sum_{i=1}^{r} \sigma_i u_i \nu_i^T. \tag{88}$$

The best rank-k approximation of matrix A can be written as

$$A_k = \sum_{i=1}^{k} \sigma_i u_i v_i^T. \tag{89}$$

It is the best approximation in the sense of minimizing the L_2 -norm of the error

$$\|A - A_k\|_2 = \sigma_{k+1}. \tag{90}$$

We may also write

$$A_k = US_kV^T, \tag{91}$$

where $S_k = diag(\lambda_1, \ldots, \lambda_k, 0, \ldots, 0)$.
Here we should explain the 2-norm of a matrix. The length of a vector $x = (x_1, x_2, \ldots, x_n)^T$ is usually given by the Euclidean norm

$$\|x\|_2 = ({x_1}^2 + {x_2}^2 + \ldots + {x_n}^2)^{1/2}. \tag{92}$$

In the case of the Euclidean norm and square matrices, the induced matrix norm is the spectral norm. The spectral norm of a matrix A is the largest singular value of A or the square root of the largest eigenvalue of the positive-semidefinite matrix A^*A.

$$\|A\|_2 = \sqrt{\lambda_{max}(A^*A)}, \tag{93}$$

where A^* denotes the conjugate transpose of A. In this way, A$_k$ has rank k and can be represented as

$$A_k = \sum_{i=1}^{k} \sigma_i u_i v_i^T = U \begin{bmatrix} \sigma_1 & \ldots & \ldots & \ldots & \ldots \\ \ldots & \sigma_2 & \ldots & \ldots & \ldots \\ \ldots & \ldots & \ddots & \ldots & \ldots \\ \ldots & \ldots & \ldots & \sigma_k & \ldots \\ \ldots & \ldots & \ldots & \ldots & \ddots \end{bmatrix} V^T \tag{94}$$

The L_2 norm of the error is given by

$$\|A - A_k\|_2 = \| \sum_{i=k+1}^{n} \sigma_i u_i v_i^T \|_2 = \sigma_{k+1} \tag{95}$$

Here we only need $m \cdot k + n \cdot k = (m+n) \cdot k$ memory places to store u_1 through u_k and $\lambda_1 v_1$ through$\lambda_k v_k$. Later we can use these to reconstruct the image A_k or the matrix A_k. Compared with the storage places needed for the original matrix A namely $m \times n$. The storage requirement for the decomposed matrix is much less when k is small. So now, A_k is our compressed image, only using $(m+n) \cdot k$ memory places. By changing k, we can get different errors $||A - A_k||_2/||A||_2$ and compression degrees defined as $1 - (m+n) \cdot k/(m \cdot n)$.

2.3.2. **The SVD method for image compression (*An Example*).** It is quite obvious that the mathematics behind the SVD would become extraordinarily involved rather quickly. Once the theory has been understood, it is a good idea to use a mathematical software. MATLAB works quite nicely. From above, it is clear that the matrix A_k provides less information than the original matrix A. In fact, considering the requirements of human visual, choosing a suitable value $k < r$ for the image file A_k, we can get a good approximation of A from A_k. The smaller value of k, the less data to present the A_k. When k gets close to rank r, the matrix A_k will approach the original image matrix A. That means, if we can choose an appropriate number of singular values, the compressed matrix A_k can show a reasonably nice image, sufficiently close to the original one, which can satisfy the human visual.

After several rounds of tests, we get a result. Usually, for $A_{m \times n}$, with $256 \leq n \leq 2048$, we can get a good quality image choosing $25 \leq k \leq 100$. For a nearly square matrix i.e $m \approx n$ and $r \approx n$, when k is in the range of $r/5$ to $r/30$, the compression ratio will be between 3/5 to 14/15.

The SVD Matlab commands are very simple:

(96)
```
load A.mat;
[U, S, V]=svd(X);
colormap('gray');
image(U(:,1:k)*S(1:k,1:k)*V(:,1:k)');
```

First of all, we load the image in MATLAB as a matrix A. Then we use the svd function to decompose the matrix into U, S and V and save them. $colormap$ is an m-by-3 matrix of real numbers between 0.0 and 1.0. Each row is an RGB (red, green, blue) vector that defines one color. In our project, we just use 'gray' colormap to set all the values between 0 to 1. In the end, we reconstruct the image with U, V and the S_k.

To find out more about these commands and others while working in MATLAB use the $help$ command. For example, if the command is $linspace(0, 5)$, type $help\ linspace$ to find out more about the $linspace$ command.

Here is an example: Create a random 8×10 matrix A with integer values ranging from -64 to 64 and use MATLAB's svd command to find the matrices U, S, V corresponding to A.

To create a matrix of random integers, the easiest way is to use the $randint$ command. The command with these parameters reads:

$$\gg A = randint(8, 10, [-64, 64])$$

The function randint(m,n,rg) which we use here generates an 8-by-10 (m-by-n) integer matrix with element in the range [-64, 64] (rg).
We now get the SVD by

$$\gg [U, S, V] = svd(A)$$

and we can check the rank of the matrix A by

$$\gg rank(A),$$

or by

$$\gg diag(S)$$

Below is an example with a much smaller matrix than our images but it can be helpful to explain the process. The image matrix $A_{9\times10}$ is decomposed by SVD into three matrices: $U(9 \times 9)$, $S(9 \times 10)$, $V(10 \times 10)$.

$$A = \begin{bmatrix} 68 & 71 & 63 & 63 & 61 & 64 & 60 & 67 & 66 & 63 \\ 67 & 64 & 64 & 61 & 63 & 65 & 66 & 77 & 70 & 66 \\ 69 & 63 & 64 & 63 & 69 & 194 & 201 & 197 & 193 & 92 \\ 67 & 67 & 65 & 65 & 81 & 112 & 54 & 87 & 85 & 147 \\ 66 & 68 & 68 & 72 & 59 & 90 & 57 & 54 & 84 & 139 \\ 67 & 61 & 70 & 75 & 83 & 90 & 96 & 101 & 107 & 64 \\ 68 & 72 & 77 & 68 & 84 & 92 & 100 & 101 & 70 & 145 \\ 65 & 65 & 62 & 72 & 84 & 93 & 104 & 130 & 101 & 134 \\ 65 & 61 & 62 & 69 & 81 & 88 & 123 & 113 & 105 & 122 \end{bmatrix} \tag{97}$$

(98)
$$U = \begin{bmatrix} -.239 & -.146 & +.519 & +.147 & -.050 & -.297 & -.344 & +.306 & +.573 \\ -.248 & -.103 & +.457 & +.031 & +.062 & -.197 & -.255 & -.039 & -.780 \\ -.489 & +.781 & +.218 & +.240 & -.058 & -.196 & -.062 & -.024 & +.000 \\ -.320 & -.344 & -.331 & -.395 & +.582 & +.007 & +.175 & +.373 & -.063 \\ -.288 & -.386 & -.202 & +.656 & -.593 & +.229 & -.128 & -.297 & -.005 \\ -.310 & +.085 & +.522 & +.066 & +.140 & +.417 & +.581 & -.271 & +.124 \\ -.337 & -.277 & -.159 & +.394 & -.114 & -.628 & +.393 & -.252 & +.069 \\ -.356 & -.083 & -.149 & -.420 & +.367 & +.309 & -.517 & -.380 & +.159 \\ -.348 & -.009 & -.060 & -.458 & -.363 & +.344 & +.080 & +.629 & -.115 \end{bmatrix},$$

(99)
$$S = \begin{bmatrix} 833.208 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 164.686 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 76.291 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 55.314 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 29.909 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 25.309 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 16.026 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 6.704 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3.756 & 0 \end{bmatrix}$$

(100)
$$V = \begin{bmatrix} -.236 & -.186 & +.339 & +.117 & -.051 & -.142 & -.264 & +.424 & -.546 & +.458 \\ -.231 & -.230 & +.307 & +.112 & -.085 & -.342 & -.413 & +.229 & +.526 & -.402 \\ -.233 & -.216 & +.311 & +.070 & -.138 & -.255 & +.233 & -.540 & -.427 & -.397 \\ -.238 & -.216 & +.317 & +.034 & -.127 & +.335 & -.024 & -.470 & +.399 & +.535 \\ -.263 & -.219 & +.251 & -.235 & +.417 & +.226 & +.973 & +.394 & +.125 & -.104 \\ -.373 & +.217 & -.283 & +.625 & +.190 & -.377 & +.303 & -.008 & +.158 & +.220 \\ -.367 & +.434 & -.010 & -.471 & -.582 & -.197 & +.209 & +.132 & +.089 & +.092 \\ -.389 & +.314 & -.022 & -.381 & +.605 & -.089 & -.388 & -.269 & -.087 & +.001 \\ -.370 & +.319 & +.030 & +.361 & -.154 & +.661 & -.159 & +.101 & -.140 & -.340 \\ -.387 & -.579 & -.670 & -.158 & -.132 & +.091 & -.102 & +.007 & -.070 & -.032 \end{bmatrix}.$$

If $k = 4$, then we get the U_k, S_k, V_k as

$$U_k = \begin{bmatrix} -0.239 & -0.146 & 0.519 & 0.147 \\ -0.248 & -0.103 & 0.457 & 0.031 \\ -0.489 & 0.781 & -0.218 & 0.240 \\ -0.320 & -0.344 & -0.331 & 0.395 \\ -0.288 & -0.386 & -0.331 & 0.395 \\ -0.310 & 0.085 & 0.522 & 0.066 \\ -0.337 & -0.277 & -0.159 & 0.394 \\ -0.356 & -0.083 & -0.149 & -0.420 \\ -0.347 & -0.009 & -0.060 & -0.458 \end{bmatrix},$$

$$S_k = \begin{bmatrix} 833.208 & 0 & 0 & 0 \\ 0 & 164.6863 & 0 & 0 \\ 0 & 0 & 76.291 & 0 \\ 0 & 0 & 0 & 55.3135 \end{bmatrix}, \tag{101}$$

$$V_k = \begin{bmatrix} -0.236 & -0.186 & 0.339 & 0.117 \\ -0.231 & -0.230 & 0.307 & 0.112 \\ -0.233 & -0.216 & 0.314 & 0.070 \\ -0.238 & -0.216 & 0.317 & 0.034 \\ -0.263 & -0.219 & 0.251 & -0.235 \\ -0.373 & 0.217 & -0.283 & 0.625 \\ -0.367 & 0.434 & -0.010 & -0.471 \\ -0.389 & 0.314 & -0.022 & -0.381 \\ -0.370 & 0.319 & 0.030 & 0.361 \\ -0.387 & -0.579 & -0.670 & -0.158 \end{bmatrix}.$$

Calculating $A_k = U_k S_k V_k^T$ now gives

$$\begin{bmatrix} 65.707 & 64.423 & 64.463 & 65.310 & 65.548 & 62.788 & 58.344 & 65.824 & 69.962 & 63.063 \\ 63.812 & 62.367 & 62.750 & 63.858 & 66.289 & 64.469 & 67.238 & 73.526 & 72.457 & 66.073 \\ 68.013 & 60.785 & 62.694 & 64.343 & 71.513 & 192.761 & 199.242 & 194.181 & 195.863 & 192.121 \\ 67.463 & 69.248 & 67.907 & 68.478 & 71.068 & 107.979 & 63.281 & 78.233 & 87.714 & 49.529 \\ 66.202 & 168.102 & 66.524 & 66.821 & 67.082 & 96.157 & 48.500 & 63.912 & 77.312 & 135.940 \\ 72.198 & 68.973 & 69.844 & 71.171 & 73.947 & 190.277 & 98.761 & 102.587 & 102.403 & 64.588 \\ 68.092 & 69.134 & 69.897 & 72.151 & 85.919 & 84.692 & 93.782 & 103.648 & 81.175 & 146.667 \\ 65.885 & 65.446 & 66.744 & 69.134 & 83.513 & 96.270 & 113.956 & 120.204 & 69.547 & 133.947 \\ 63.996 & 62.855 & 64.446 & 66.889 & 81.163 & 93.056 & 117.597 & 121.943 & 97.289 & 119.852 \end{bmatrix}. \tag{102}$$

From these matrices, we can see that A_k is almost equal to A. The size of the image (matrix) is large, in practice, say 1024×768 matrix. For a larger matrix we may choose relatively smaller value of k. Hence the compression ratio will become very large. For example, if $k = 100$, the compression degree is 77% for a 1024×768 matrix. Normally, the image can be compressed to $80 - 90\%$ of the original one, and the distortion is still not serious.

There are several reasons why the SVD has become so popular. First, it is very stable. Small change in the input A result in small change in the singular matrix S, and vice versa. Second, the singular values σ_i provide an easy way to approximate A.

2.4. **Error Estimates.** In order to measure the quality of image compression, we choose three methods to calculate the difference in distortion. Suppose that $f(x,y)$ represents the original image, and $g(x,y)$ is the compressed image,both of size are $M \times N$. Then we will have the following formulas.

Average absolute difference:

$$D_{aad} = 1/NM \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} |g(x,y) - f(x,y)| \tag{103}$$

L_p-norm:

$$D_{L_p} = \|f(x,y) - g(x,y)\|_p = \{1/NM \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} |g(x,y) - f(x,y)|^p\}^{1/p} \tag{104}$$

where we will use p = 2.

Note the p = 1 gives Daad.

Signal-to-noise ratio:

$$D_{SNR} = \frac{\sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f^2(x,y)}{\sum_{x=0}^{M-1} \sum_{y=0}^{N-1} [g(x,y) - f(x,y)]^2} \tag{105}$$

In these three norms, we can compare the effect of different compression methods more accurately. The average absolute difference and SNR can be understood easily.

In image processing, the SNR of an image is usually defined as the ratio of the mean pixel value to the standard deviation of the pixel values. Related measures are the ”contrast ratio” and the ”contrast-to-noise ratio”.

3. **Coding Methods**

3.1. **Fourier Transform.** For gray picture, first we load the data, and set the threshold of compression. Then we use the commend $blkproc$ in MATLAB.

As the help of MATLAB describes, the commend

$$B = blkproc(A, [mn], fun) \tag{106}$$

processes the image A by applying the function fun to each distinct m by n (here we use 8-by-8) block of A, padding A with 0's if necessary. fun is a function handle that accepts an m by n matrix, x, and returns a matrix, vector, or scalar y, i.e.

$$y = fun(x). \tag{107}$$

$blkproc$ does not require that y be of the same size as x. However, B is of the same size as A only if y is of the same size as x. Here we use 'dct2' as the fun, which represents the two-dimensional discrete cosine transform.

Now we get the image in Fourier domain. As the major information of an image relies on the low frequency part, we set a threshold and remove the high frequency part, which contains mostly details and noises. We do this also by m by n blocks (here we use 8-by-8).

We reconstruct the image again by the commend $blkproc$, but with the fun as $idct2$, which returns the two-dimensional inverse discrete cosine transform of the threshold data.

To get the error rate, we calculate the difference between the original image and the compressed image by subtraction. Then in order to make different compressed images comparable, we calculate the L_2-norm of the difference matrix and normalize it with the L_2-norm of the original image. The result is the error rate of the image compression.

For pictures with color, we first divide the picture into three layers: red, green and blue. Then we process the three layers respectively as the method we used for gray pictures. After we get 3 compressed layers, we combined them back into a single color picture.

3.2. **Wavelet Transform.** For gray pictures, first we load the data, and set the threshold and the level of Wavelet compression. We use the $wavedec2$ commend in MATLAB:

$$[C, S] = wavedec2(X, N,' wname') \tag{108}$$

As the MATLAB help describes, $wavedec2$ is a two-dimensional wavelet analysis function. The function returns the wavelet decomposition of the matrix X at level N, using the wavelet named in string 'wname' (here we are using Haar wavelet). Outputs are the decomposition vector C and the corresponding book keeping matrix S, see Figure 13.
The vector C is organized as

(109)
$$C = [A(N)|H(N)|V(N)|D(N)|...H(N-1)|V(N-1)|D(N-1)|...|H(1)|V(1)|D(1)]$$

where A, H, V, D, are row vectors with entries described as follows:

- A = approximation coefficients
- H = horizontal detail coefficients
- V = vertical detail coefficients
- D = diagonal detail coefficients

The matrix S is such that

$$\begin{aligned} S(1,:) &= \text{size of approximation coefficients (N)}, \\ S(i,:) &= \text{size of detail coefficients (N-i+2) for i = 2, ...N+1}, \\ S(N+2,:) &= \text{size(X)} \end{aligned} \tag{110}$$

Now we take the first $S(1,1) * S(1,2)$ elements in the decomposition vector C, which contain the coarsest approximation. Then we threshold these elements to get the compressed data.

To inverse the Wavelet Transform back to image, we use the MATLAB command

$$X = waverec2(C, S,' wname') \tag{111}$$

which performs a multilevel wavelet reconstruction of the matrix X based on the wavelet decomposition structure [C,S]. Use the same 'wname' as in the $wavedec2$.

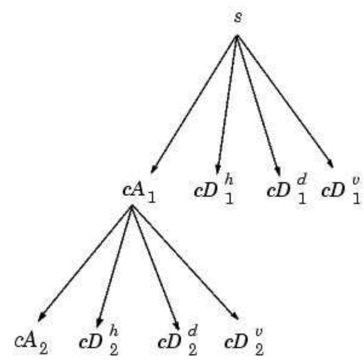

FIGURE 13. The structure of S

As we did before in Fourier Transform, we calculate the difference between the original image and the compressed image by subtraction. Then we calculate the L_2-norm of the difference matrix and normalize it with the L_2-norm of the original image, to make different compressed images comparable. The result is the error rate of the image compression.

For pictures with color, we first divide the picture into three layers: red, green and blue. Then we process the three layers respectively with the method used for gray pictures. After that we get 3 compressed layers and we combined them back into a single colorful picture.

3.3. **Singular Value Decomposition.** In order to get a better understanding of the coding method for SVD, it is necessary to include a discussion about how MATLAB constructs images. Normally, each entry in the matrix corresponds to a small square of the image. The value of the entry corresponds to a color. We can get the color spectrum easily in MATLAB.

$\gg A = 1 : 64; \gg image(A);$

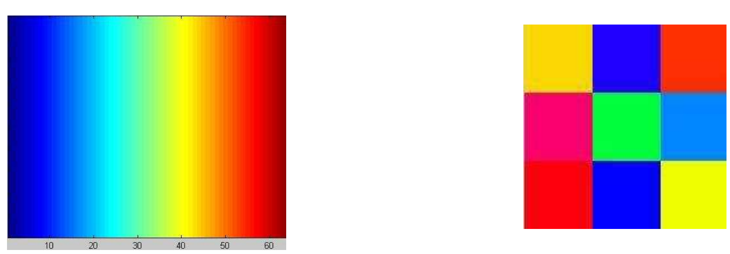

FIGURE 14. Colour spectrum and blocked image

The right figure shows a 3× 3 matrix of random integers which has 9 square blocks comprising one large block. The code we used is

$\gg A = randint(3,3); \gg image(A);$

According to our previous theory, any matrix A can be approximated using a smaller number of iterations (singular values) when calculating the approximate SVD of A. These images we can get by our SVD code. We choose three iterations images with three different parameters of σ_k :σ_1,σ_2,σ_3, and the number of iterations equals the rank of the approximate SVD matrix A_k.

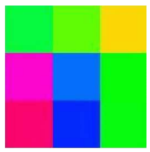

FIGURE 15. One iteration

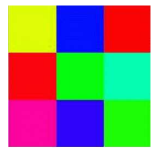

FIGURE 16. Two iterations

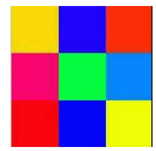

FIGURE 17. Three iterations

Through the figures, we can see that the original image does not appear until the third iteration. Note that a more detailed image A which is an $m \times n$ matrix, can be approximated using the same techniques.

For another example, let A be a 15×20 matrix of random integers ranging from -64 to 64, with rank 12. So the original image should be represented by the twelfth iteration. But for human vision, it is possible to get a good quality approximation in ten iterations. Here we use the MATLAB commands

$\gg A = randint(15,20,64); \gg [U,S,V] = svd(A);$

and compute the approximate A_k by $A_k = U_k S_k V_k^T$ for different values of k (iterations).

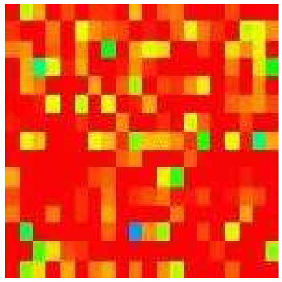

FIGURE 18. Five iterations

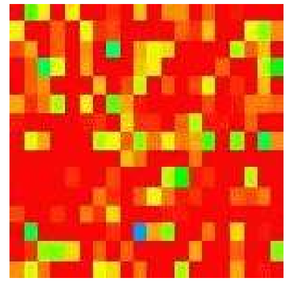

FIGURE 19. Ten iterations

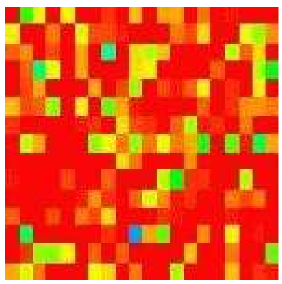

FIGURE 20. Twelve iterations

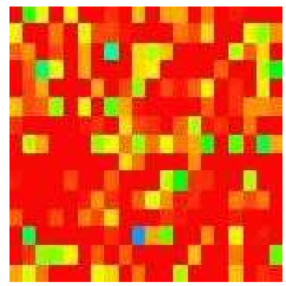

FIGURE 21. Original image

For the actual random matrix with $r = 12$, A_{12} is an exact copy of the original matrix A. The ten iteration image A_{10} has a good enough quality for human vision, see Figure 19.

Now we use a real nature image which is represented by 512×512 matrix for analysis, the group images of vegetables. As we can see, after 10 iterations we can already make out what the image is, see Figure 24. By 25 iterations the figure is much clear and with 75 iterations the figure is mostly the same as the original one. The compression degree for this image is $1 - \frac{(512+512)\times 75}{512\times 512} = 71\%$.

Our project coding command computes the matrix singular value decomposition, it produces a diagonal matrix S of the same dimension as A, with nonnegative diagonal elements in decreasing order, and orthogonal matrices U and V so that $A = USV^T$. When we get the decomposed matrices, it means that we choose a value k for image compression. The value of k decides the content of compressed image which combines by three new matrices. In MATLAB-code this reads

$$A_k = U(:, 1:k) * S(1:k, 1:k) * V(:, 1:k)' \tag{112}$$

The difference between the original image and the compressed one is calculated by the L_2-norm, $\|A - A_k\|_2 = \sigma_{k+1}$. For the colorful image, at the first step we always divide the image into three layers (red green and blue). For each layer, we process the matrix as above, decompose them and compress them separately. Finally to get a colorful figure, we combine three compressed matrices together.

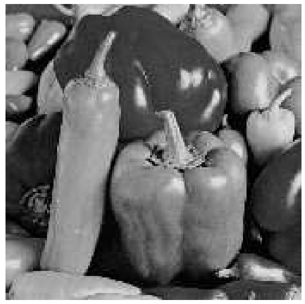

FIGURE 22. Original image

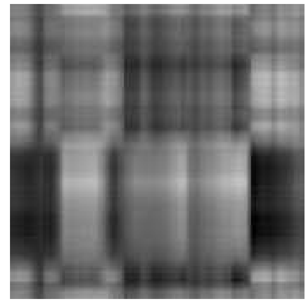

FIGURE 23. 2 iterations

FIGURE 24. 10 iterations

FIGURE 25. 25 iterations

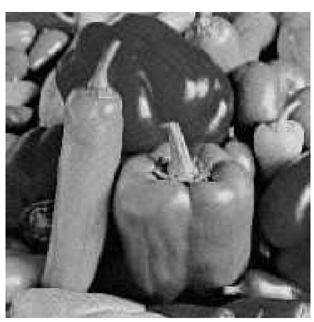

FIGURE 26. 50 iterations

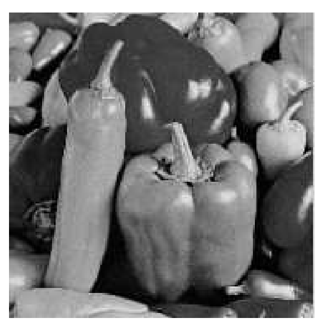

FIGURE 27. 75 iterations

4. **Result**

4.1. **The processed figures.** We apply the MATLAB methods described above to the chosen pictures. First we set the threshold of Wavelet and Fourier Transform. The level of Wavelet is set as 3 to be comparable to Fourier Transform. Then we choose different compression ratios: about 80%, 90% and 98.5%, and according these compressions choose the singular value of SVD. We can get three different quality levels of the compressed figures: **A+**: recognizable from original one, **A**: acceptable, **A-**: can't be accepted.

We compress all the images in five methods with three quality levels, and calculate their compression ratios and error ratios. The order of the methods is FFT, SVD, Wavelet(db1), Wavelet(db2), Wavelet(db4).

In this way, there are three figures for each method, and five methods for each image, which means there are fifteen figures for each image. We processed seven different kinds of images, so there are one hundred and five figures in all. There are abbreviations some used in this part: CR means compression ratio, ER means error ratio, FP means Finger Print.

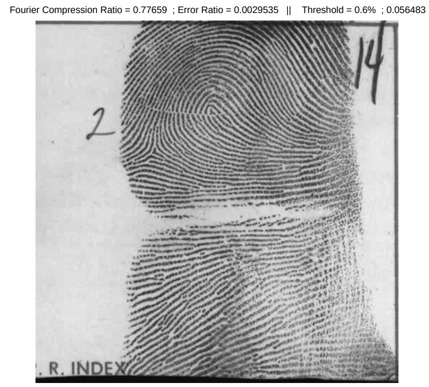

FIGURE 28. A+, FP, FFT, CR = 0.77659, ER = 0.0029535

FIGURE 29. A, FP, FFT,CR = 0.94727, ER = 0.0107

FIGURE 30. A-, FP, FFT, CR = 0.98207, ER = 0.031263

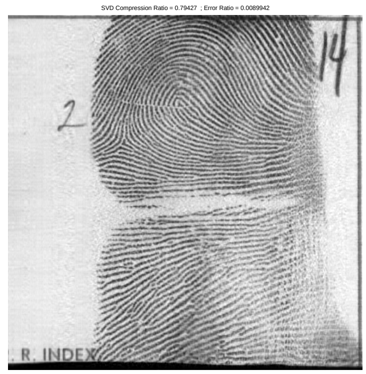

FIGURE 31. A+, FP, SVD, CR = 0.79427, ER = 0.0089942

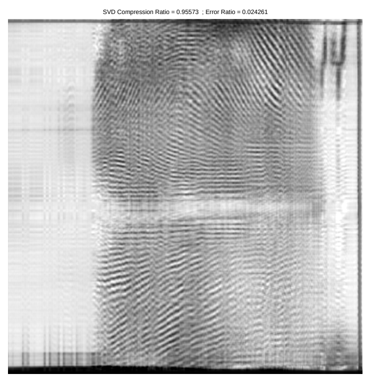

FIGURE 32. A, FP, SVD, CR = 0.95573, ER = 0.024261

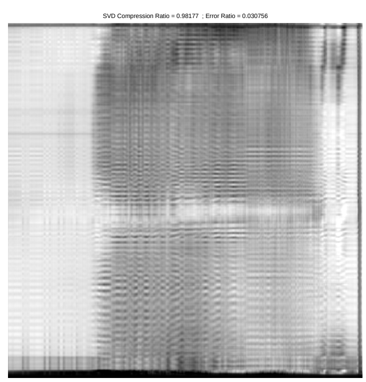

FIGURE 33. A-, FP, SVD, CR = 0.98177, ER = 0.030756

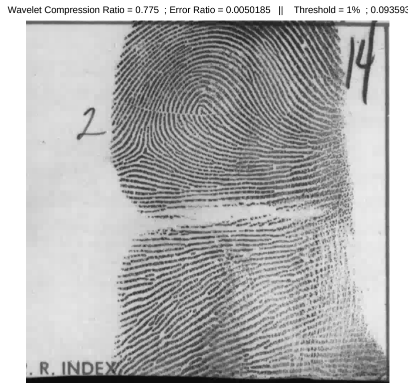

FIGURE 34. A+, FP, Haar, CR = 0.775, ER = 0.0050185

FIGURE 35. A, FP, Haar, CR = 0.94222; ER = 0.016428

FIGURE 36. A-, FP, Haar, CR = 0.98172, ER = 0.032037

FIGURE 37. A+, FP, db2, CR = 0.7791, ER = 0.004197

FIGURE 38. A, FP, db2, CR = 0.94745, ER = 0.013493

FIGURE 39. A-, FP, db2, CR = 0.98206, ER = 0.034217

FIGURE 40. A+, FP, db4, CR = 0.7597, ER = 0.0030893

FIGURE 41. A, FP, db4, CR = 0.94488, ER = 0.011193

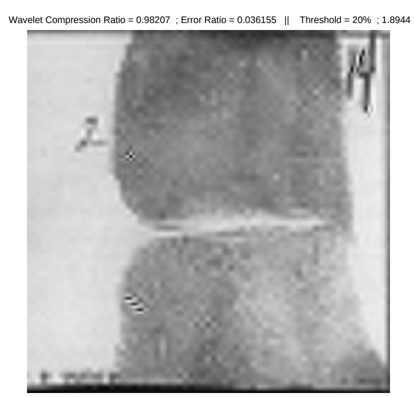

FIGURE 42. A-, FP, db4, CR = 0.98207, ER = 0.036156

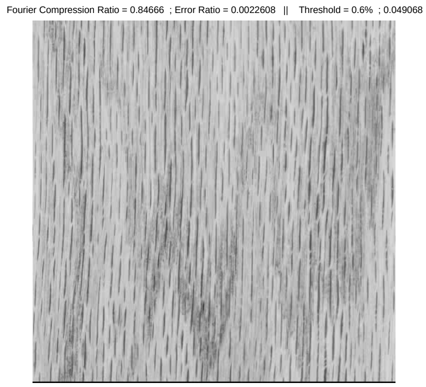

FIGURE 43. A+, Wood, FFT, CR = 0.84666, ER = 0.0022608

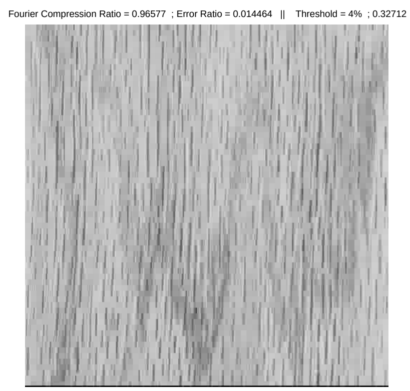

FIGURE 44. A, Wood, FFT, CR = 0.96577, ER = 0.014464

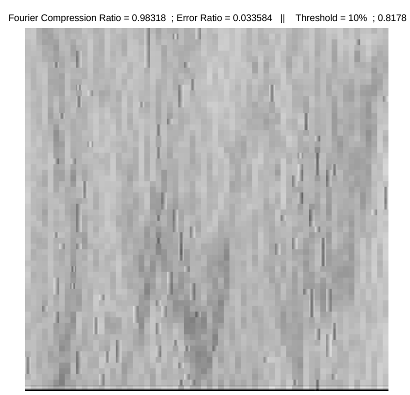

FIGURE 45. A-, Wood, FFT, CR = 0.98318, ER = 0.033584

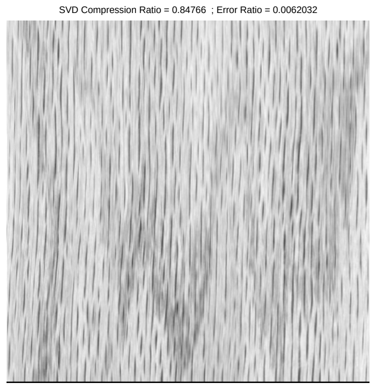

FIGURE 46. A+, Wood, SVD, CR = 0.84766, ER = 0.0062032

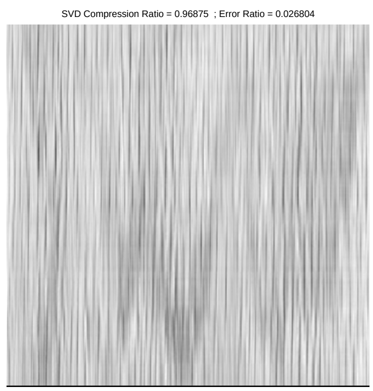

FIGURE 47. A, Wood, SVD, CR = 0.96875; ER = 0.026804

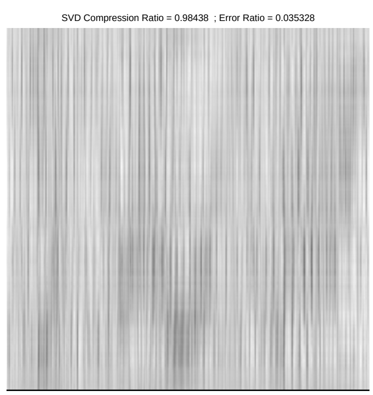

FIGURE 48. A-, Wood, SVD, CR = 0.98438, ER = 0.035328

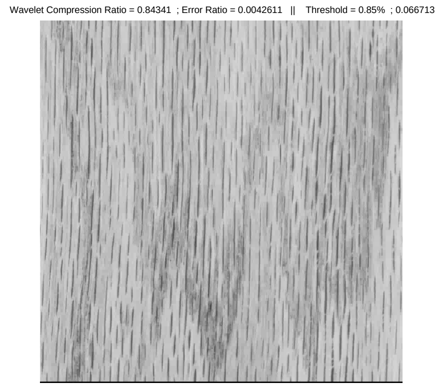

FIGURE 49. A+, Wood, Haar, CR = 0.84341; ER = 0.0042611

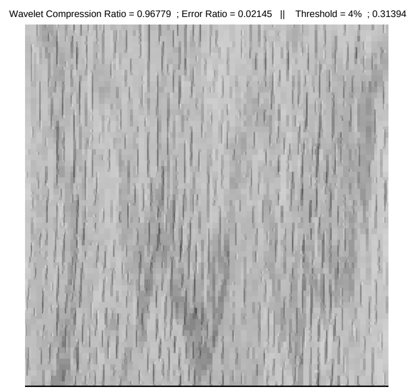

FIGURE 50. A, Wood, Haar, CR = 0.96779, ER = 0.02145

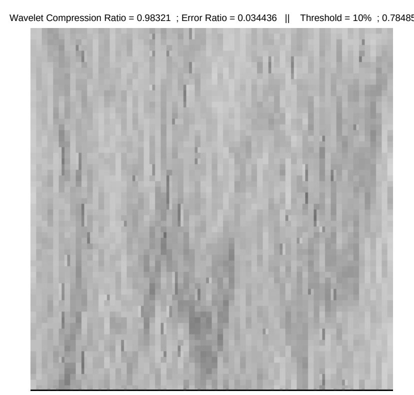

FIGURE 51. A-, Wood, Haar, CR = 0.98321, ER = 0.034436

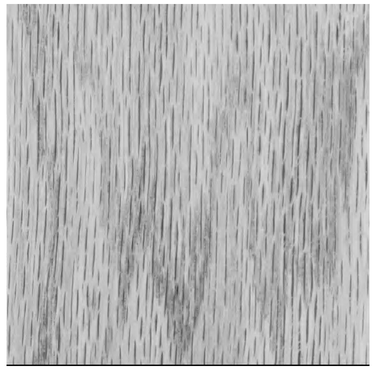

FIGURE 52. A+, Wood, db2, CR = 0.84547, ER = 0.005229

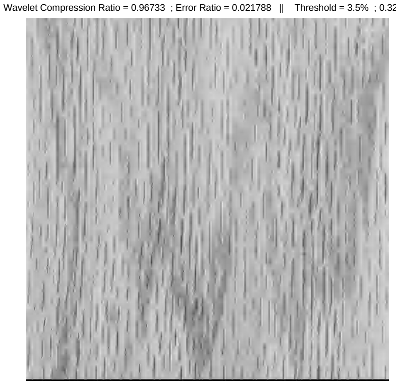

FIGURE 53. A, Wood, db2, CR = 0.96733, ER = 0.021788

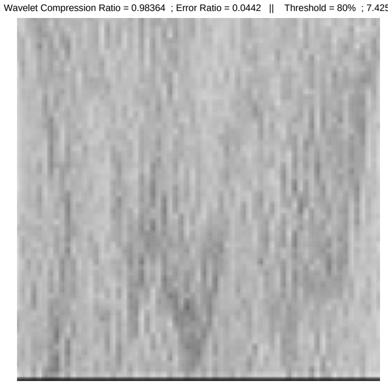

FIGURE 54. A-, Wood, db2, CR = 0.98364; ER = 0.0442

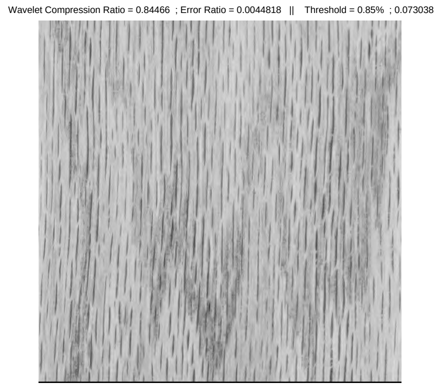

FIGURE 55. A+, Wood, db4, CR = 0.84466, ER = 0.0044818

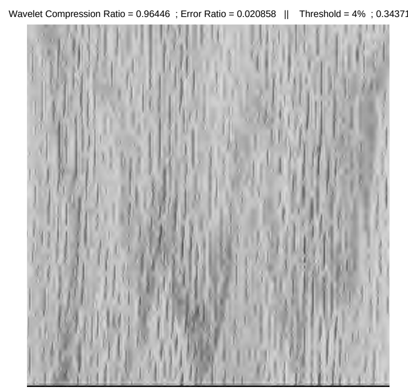

FIGURE 56. A, Wood, db4, CR = 0.96446, ER = 0.020858

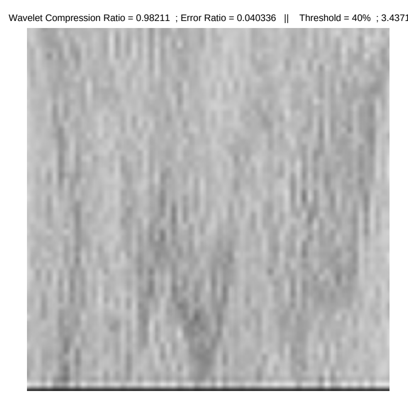

FIGURE 57. A-, Wood, db4, CR = 0.98211, ER = 0.040336

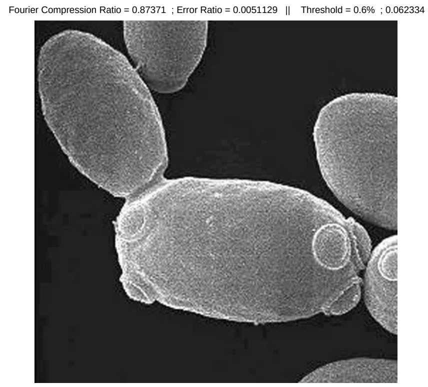

FIGURE 58. A+, Fungus, FFT, CR = 0.87371, ER = 0.0051129

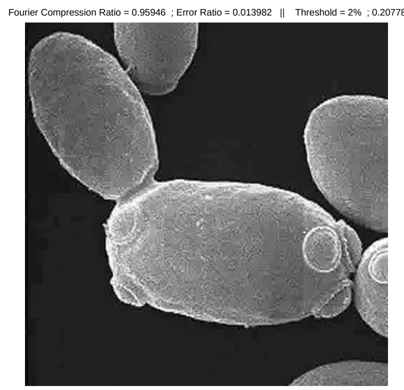

FIGURE 59. A, Fungus, FFT, CR = 0.95946, ER = 0.013982

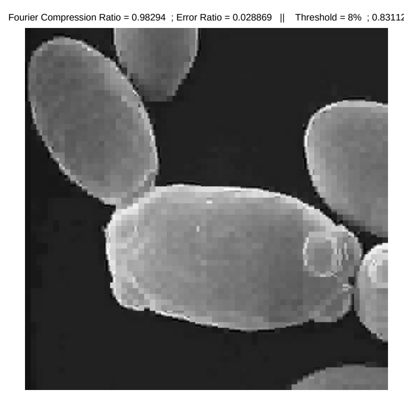

FIGURE 60. A-, Fungus, FFT, CR = 0.98294, ER = 0.028869

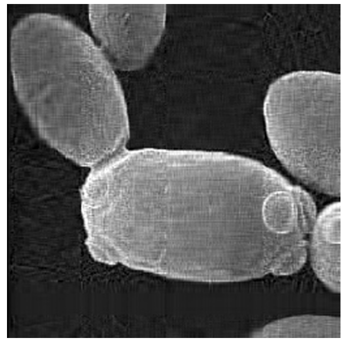

FIGURE 61. A+, Fungus, SVD, CR = 0.875, ER = 0.027916

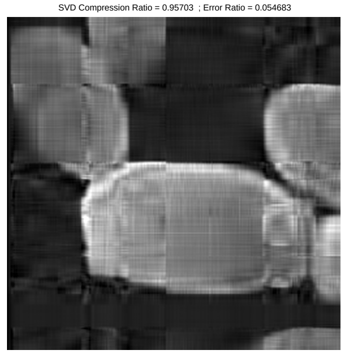

FIGURE 62. A, Fungus, SVD, CR = 0.95703, ER = 0.054683

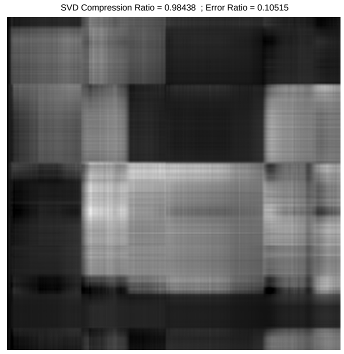

FIGURE 63. A-, Fungus, SVD, CR = 0.98438, ER = 0.10515

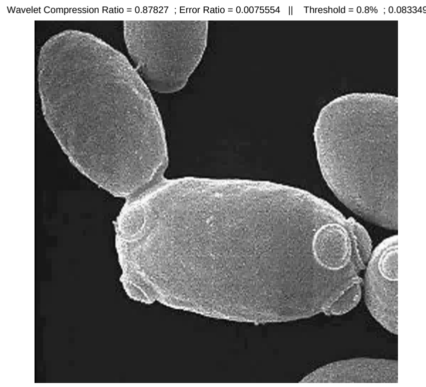

FIGURE 64. A+, Fungus, Haar, CR = 0.87827, ER = 0.0075554

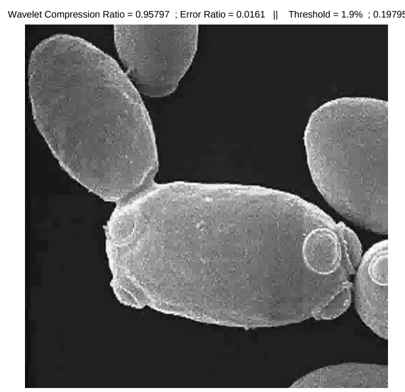

FIGURE 65. A, Fungus, Haar, CR = 0.95797, ER = 0.0161

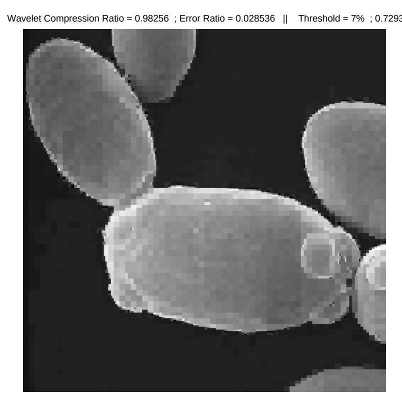

FIGURE 66. A-, Fungus, Haar, CR = 0.98256, ER = 0.028536

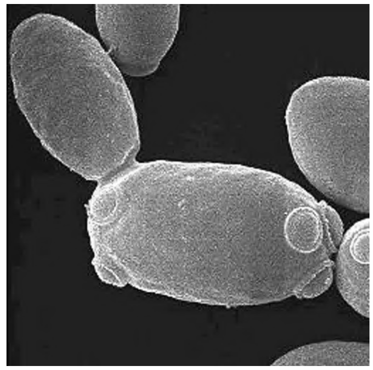

FIGURE 67. A+, Fungus, db2, CR = 0.87409, ER = 0.0067255

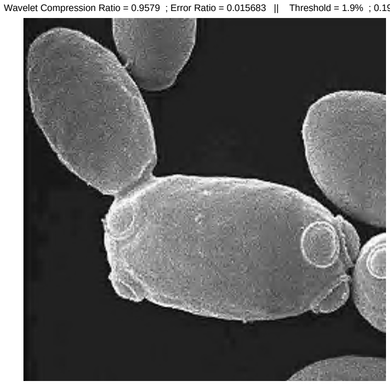

FIGURE 68. A, Fungus, db2, CR = 0.9579, ER = 0.015683

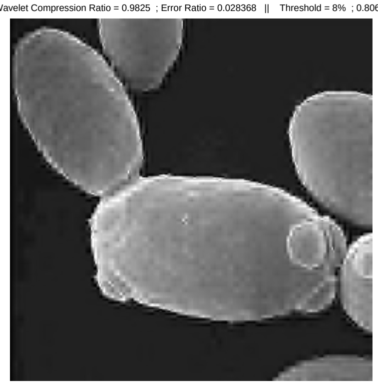

FIGURE 69. A-,Fungus,db2,CR=0.9825,ER=0.028368

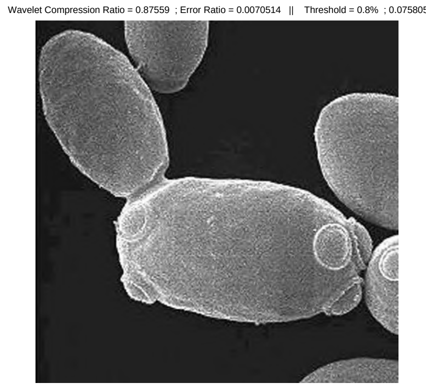

FIGURE 70. A+, Fungus, db4, CR = 0.87559, ER = 0.0070514

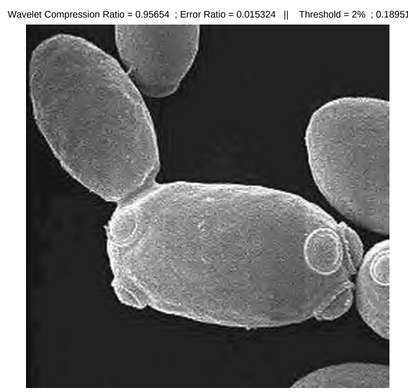

FIGURE 71. A, Fungus, db4, CR = 0.95654, ER = 0.015324

FIGURE 72. A-, Fungus, db4, CR = 0.98206, ER = 0.043639

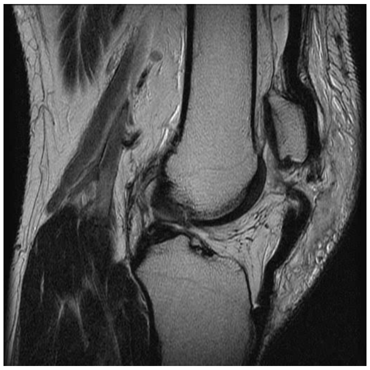

FIGURE 73. A+, MRI, FFT, CR = 0.80042, ER = 0.0021607

FIGURE 74. A, MRI, FFT, CR = 0.95477, ER = 0.01124

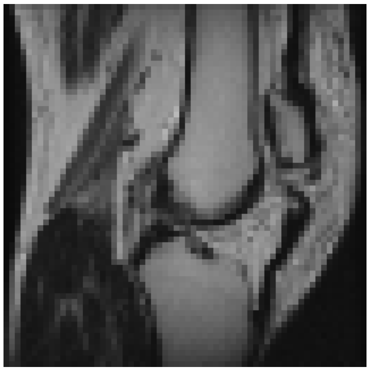

FIGURE 75. A-, MRI, FFT, CR = 0.98438, ER = 0.069299

FIGURE 76. A+, MRI, SVD, CR = 0.80469, ER = 0.011546

FIGURE 77. A, MRI, SVD, CR = 0.96875, ER = 0.05091

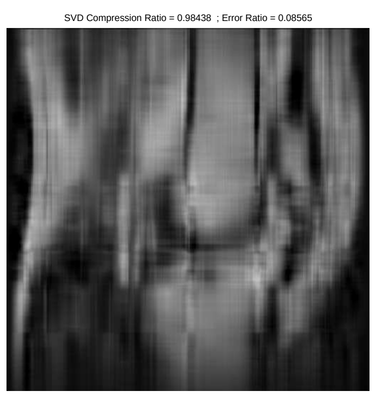

FIGURE 78. A-, MRI, SVD, CR = 0.98438, ER = 0.08565

FIGURE 79. A+, MRI, Haar, CR = 0.80277, ER = 0.0049436

FIGURE 80. A, MRI, Haar, CR = 0.95078, ER = 0.014788

FIGURE 81. A-, MRI, Haar, CR = 0.98434, ER = 0.067778

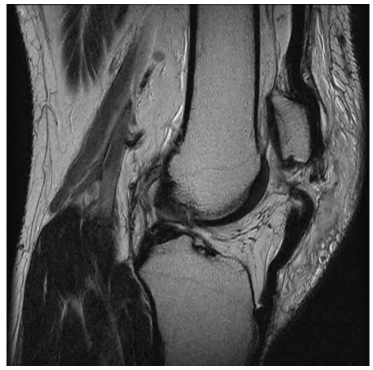

FIGURE 82. A+, MRI, db2, CR = 0.8098, ER = 0.0042916

FIGURE 83. A, MRI, db2, CR = 0.95767, ER = 0.013892

FIGURE 84. A-, MRI, db2, CR = 0.98364, ER = 0.050141

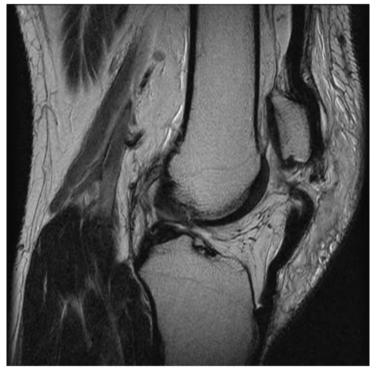

FIGURE 85. A+, MRI, db4, CR = 0.8035, ER = 0.0039866

FIGURE 86. A, MRI, db4, CR = 0.95575, ER = 0.012429

FIGURE 87. A-, MRI, db4, CR = 0.98211, ER = 0.045417

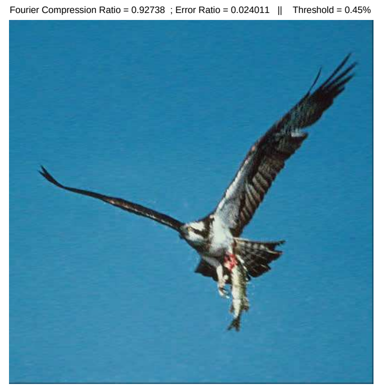

FIGURE 88. A+, bird, FFT, CR = 0.92738, ER = 0.024011

FIGURE 89. A, bird, FFT, CR = 0.97652, ER = 0.024908

FIGURE 90. A-, bird, FFT, CR = 0.98231, ER = 0.028418

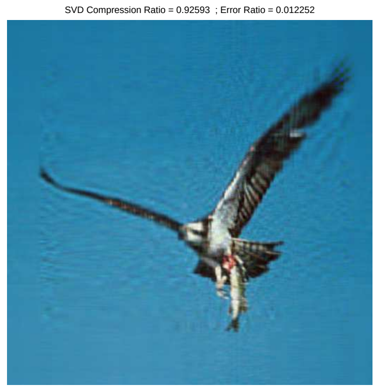

FIGURE 91. A+, bird, SVD, CR = 0.92593, ER = 0.012252

FIGURE 92. A, bird, SVD, CR = 0.97957, ER = 0.032357

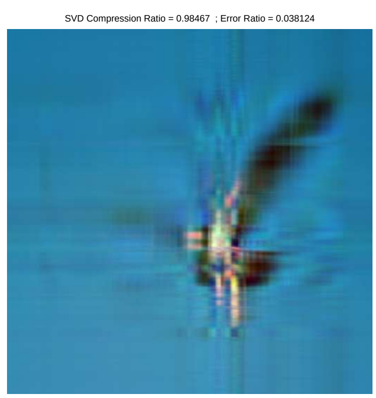

FIGURE 93. A-, bird, SVD, CR = 0.98467, ER = 0.038124

Wavelet Compression Ratio = 0.92214 ; Error Ratio = 0.003742 || Threshold = 0.45%

FIGURE 94. A+, bird, Haar, CR = 0.92214, ER = 0.003742

Wavelet Compression Ratio = 0.97567 ; Error Ratio = 0.0092727 || Threshold = 2%

FIGURE 95. A, bird, Haar, CR = 0.97567, ER = 0.0092727

Wavelet Compression Ratio = 0.98284 ; Error Ratio = 0.019097 || Threshold = 6%

FIGURE 96. A-, bird, Haar, CR = 0.98284, ER = 0.019097

FIGURE 97. A+, bird, db2, CR = 0.92618, ER = 0.0035689

FIGURE 98. A, bird, db2, CR = 0.97816, ER = 0.0089658

FIGURE 99. A-, bird, db2, CR = 0.98299, ER = 0.017445

FIGURE 100. A+, bird, db4, CR = 0.9279, ER = 0.0037105

FIGURE 101. A, bird, db4, CR = 0.97743, ER = 0.0088242

FIGURE 102. A-, bird, db4, CR = 0.98225, ER = 0.03381

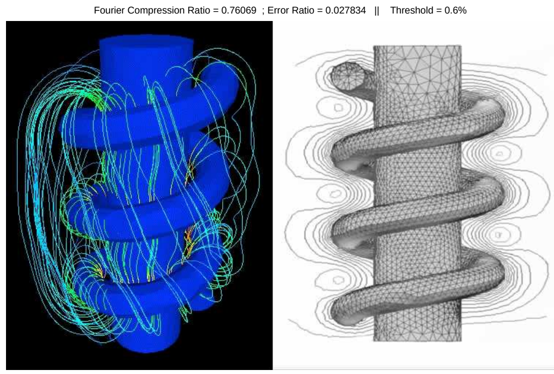

FIGURE 103. A+, coil, FFT, CR = 0.76069, ER = 0.027834

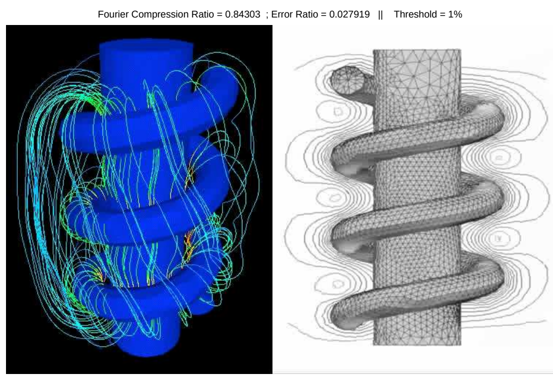

FIGURE 104. A, coil, FFT, CR = 0.84303, ER = 0.027919

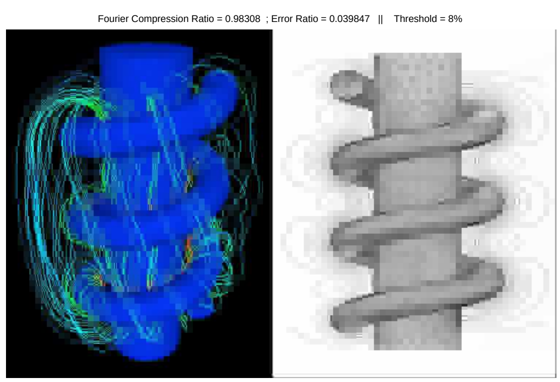

FIGURE 105. A-, coil, FFT, CR = 0.98308, ER = 0.039847

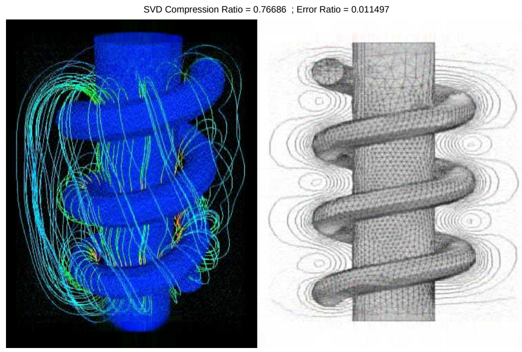

FIGURE 106. A+, coil, SVD, CR = 0.76686, ER = 0.011497

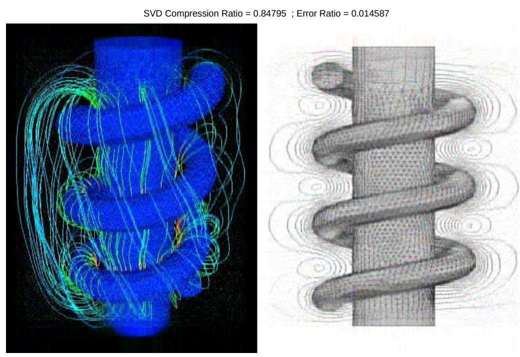

FIGURE 107. A, coil, SVD, CR = 0.84795, ER = 0.014587

FIGURE 108. A-, coil, SVD, CR = 0.98986, ER = 0.053735

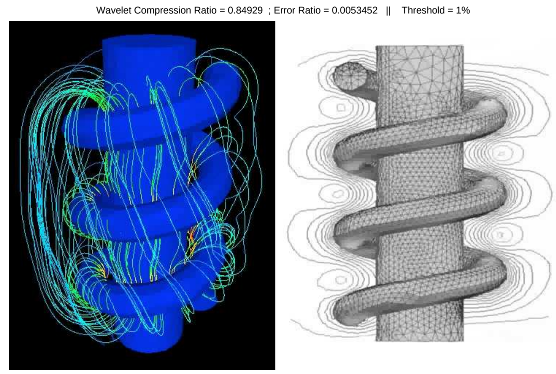

FIGURE 109. A+, coil, Haar, CR = 0.84929, ER = 0.0053452

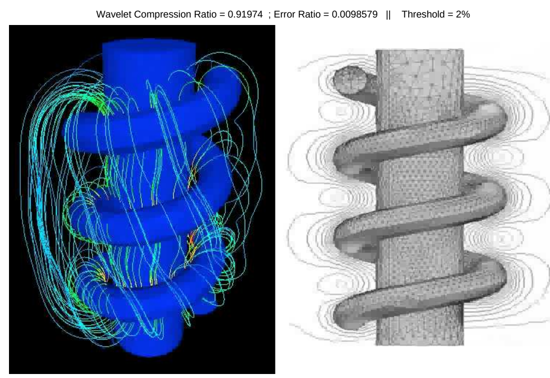

FIGURE 110. A, coil, Haar, CR = 0.91974, ER = 0.0098579

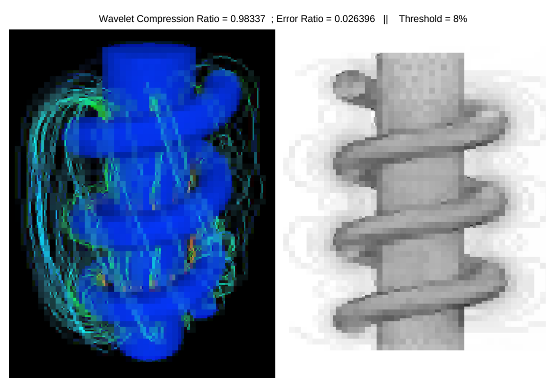

FIGURE 111. A-, coil, Haar, CR = 0.98337, ER = 0.026396

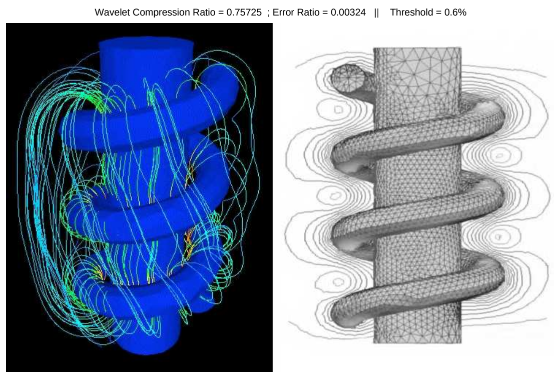

FIGURE 112. A+, coil, db2, CR = 0.75725, ER = 0.00324

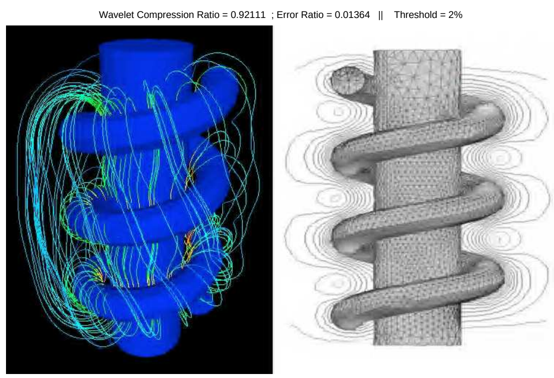

FIGURE 113. A, coil, db2, CR = 0.92111, ER = 0.01364

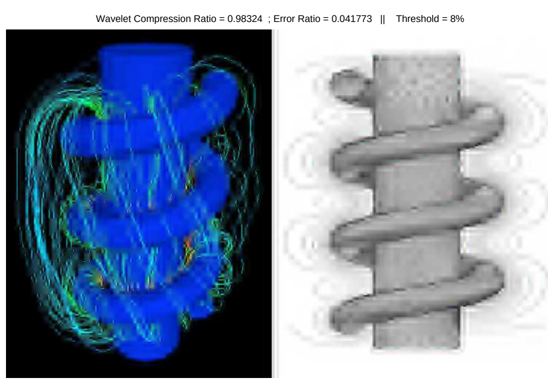

FIGURE 114. A-, coil, db2, CR = 0.98324, ER = 0.041773

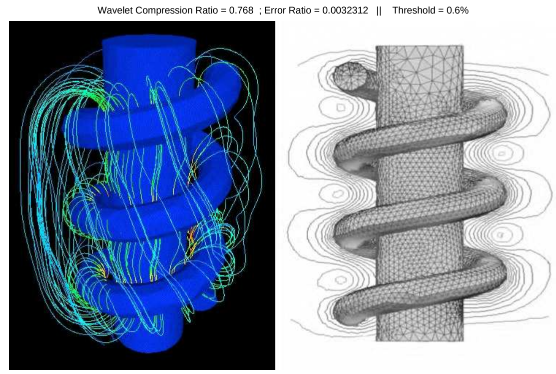

FIGURE 115. A+, coil, db4, CR = 0.768, ER = 0.0032312

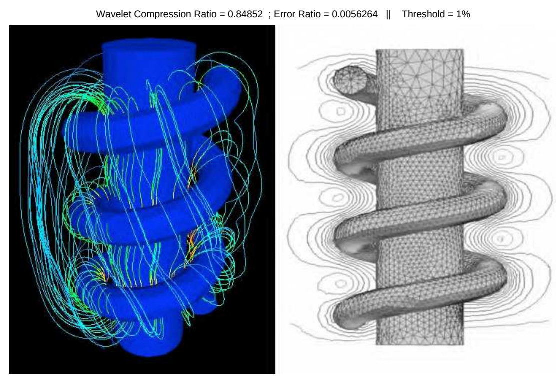

FIGURE 116. A, coil, db4, CR = 0.84852, ER = 0.0056264

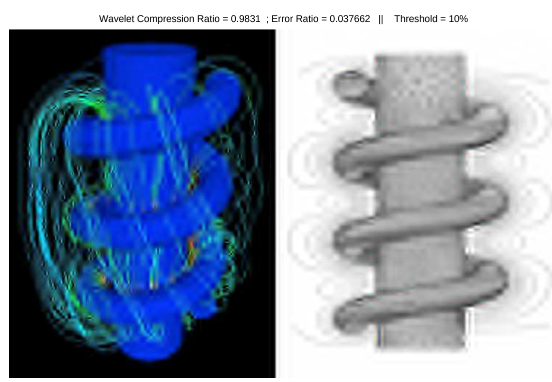

FIGURE 117. A-, coil, db4, CR = 0.9831, ER = 0.037662

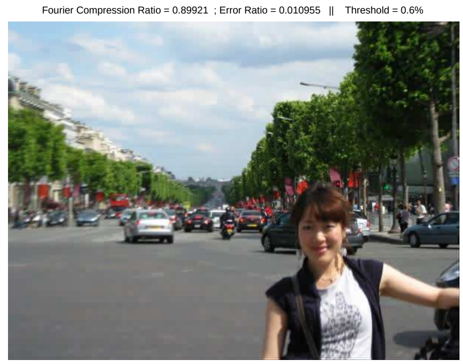

FIGURE 118. A+, Duan, FFT, CR = 0.89921, ER = 0.010955

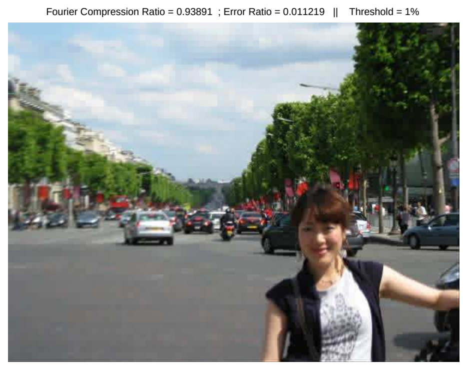

FIGURE 119. A, Duan, FFT, CR = 0.93891, ER = 0.011219

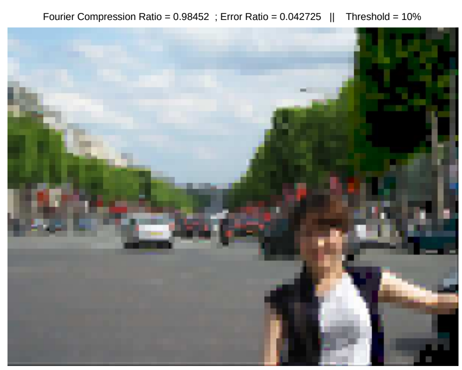

FIGURE 120. A-, Duan, FFT, CR = 0.98452, ER = 0.042725

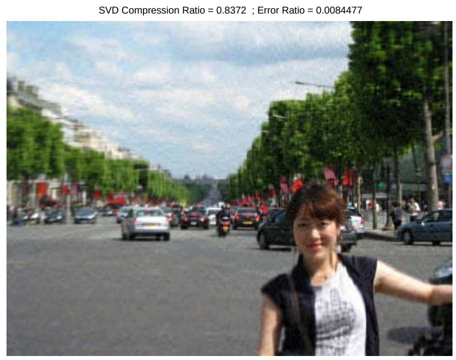

FIGURE 121. A+, Duan, SVD, CR = 0.8372, ER = 0.0084477

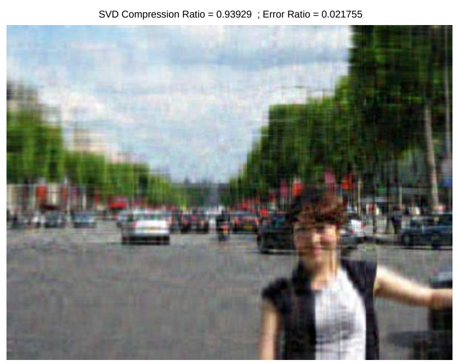

FIGURE 122. A, Duan, SVD, CR = 0.93929, ER = 0.021755

FIGURE 123. A-, Duan, SVD, CR = 0.98344, ER = 0.068952

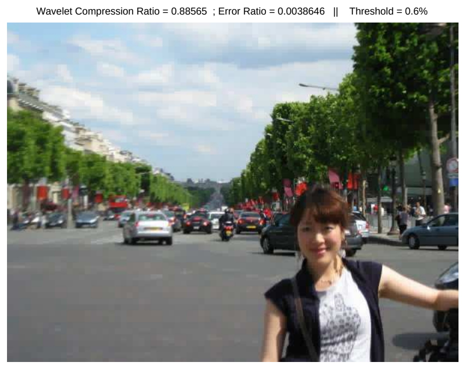

FIGURE 124. A+, Duan, Haar, CR = 0.88565, ER = 0.0038646

FIGURE 125. A, Duan, Haar, CR = 0.93062, ER = 0.0059593

FIGURE 126. A-, Duan, Haar, CR = 0.9842, ER = 0.034342

FIGURE 127. A+, Duan, db2, CR = 0.88475, ER = 0.0028814

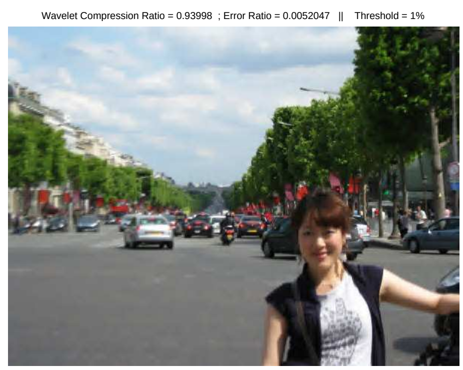

FIGURE 128. A, Duan, db2, CR = 0.93998, ER = 0.0052047

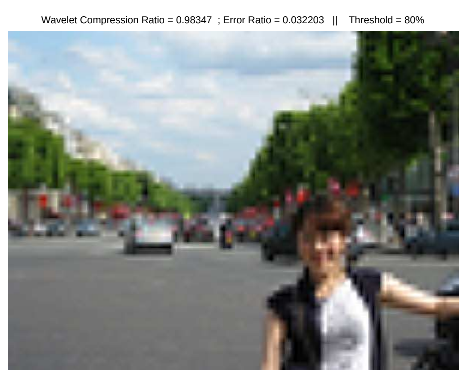

FIGURE 129. A-, Duan, db2, CR = 0.98347; ER = 0.032203

FIGURE 130. A+, Duan, db4, CR = 0.8997, ER = 0.0029361

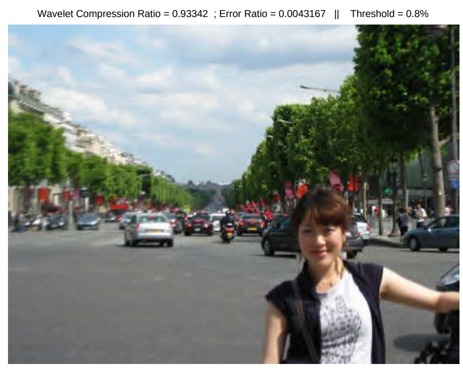

FIGURE 131. A, Duan, db4, CR = 0.93342, ER = 0.0043167

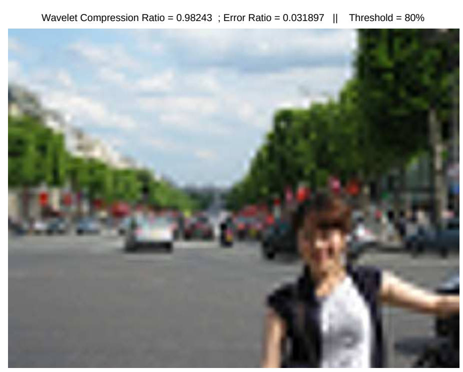

FIGURE 132. A-, Duan, db4, CR = 0.98243, ER = 0.031897

4.2. **Data Analysis.** These eight images almost include all different kinds of image types, gray, color, texture, animal, human face, detail, rough. In this way, we can analysis the data more professionally and more persuasively. In this section, we will do some study in different interesting perspectives, and we can see a quite interesting result.

4.2.1. **Comparison of different compression methods.** In order to feel more directly the effects of different compression methods, we gathered the characteristic curves of all the methods in one graph for each compressed image. Finding some phenomena of each image. The plots are shown after these paragraphs.

Fingerprint

There are lots of curve texture in finger print image. In this figure we can see that
1. As a whole, FFT is the best method for the image. At the same compression degree, FFT always has the lower error ratio.
2. SVD does not work very well, the error ratio is higher than the others.
3. Among the wavelets way, db4 is the best one, and Haar wavelet is the worst.
4. With the raising of the compression degree, the error ration for all the methods raise very quickly. Wavelet and FFT can't compress the image beyond a certain compression degree, but SVD continue compressing.

Wood

There are lots of vertical texture in wood image. In this figure we can see that
1. As a whole, FFT is the best method for Wood image. At the same compression degree, FFT always has the lower error ratio.
2. SVD does not work very well, the error ratio is higher than the others.
3. The effects of three wavelet methods are quite similar to each others.
4. With the raising of the compression degree, the error ration for all the methods raise very quickly. Wavelet and FFT can't compress the image beyond a certain compression degree, but SVD continues compressing.

Fungus

There are some clear objects in fungus image. In this figure we can see that
1. As a whole, FFT is still the best one, but the difference is not so clear now. FFT and wavelet are quite similar to each others.
2. SVD does not work very well, the error ratio is much higher than the others.
3. The effects of three wavelet methods are almost the same.
4. Wavelet and FFT stop compressing the image beyond a certain compression degree, but SVD still can compress a lot.

MRI

MRI is a gray scale image here. In this figure we can see that
1. As a whole, FFT is still the best one, but now the difference is not obvious. FFT and wavelet are quite similar to each others.
2. SVD does not work very well, the error ratio is much higher than the others.
3. The effects of three wavelet methods are almost the same.
4. Wavelet and FFT stop compressing the image at a certain compression degree, but SVD still can compress a lot.

Bird

This is a color image with a lot of blue color. In this figure we can see that
1. With the compression ratio ≤ 0.9675, wavelet is much better than FFT and SVD, the FFT is the worst. For the compression ration > 0.9675, FFT turns out to be better than SVD.
2. With raising compression degree SVDs behavior becomes poorer
3. The effects of three wavelet methods are almost the same.
4. Wavelet and FFT stop compressing the image at a certain compression degree, but SVD still can compress a lot.

Coil

This image combines two parts: a color one and a gray one. In this figure we can see that
1. Wavelet is better than both FFT and SVD. Now the FFT is performance the worst. The error ratio in FFT hardly changes.
2. SVD works better, when the compression degree is raising.
3. The effects of three wavelet methods are almost the same.
4. Wavelet and FFT stop compressing the image beyond a certain compression degree, but SVD still can compress a lot.

Duan

'Duan' is a common photo of Wei Duan with true colors. In this figure we can see that
1. As a whole, wavelet is the best method, SVD is the worst one. FFT is in the middle and doesn't change a lot when the compression degree is lower than a certain value.
2. SVD does not work very well, the error ratio is quite higher than the other methods.
3. The effects of three wavelet methods are similar, but the db4 seems to be the best one, the second best is db2, then comes the Haar wavelet in the last place.
4. Wavelet and FFT stop compressing the image beyond a certain compression degree, but SVD still can compress a lot.

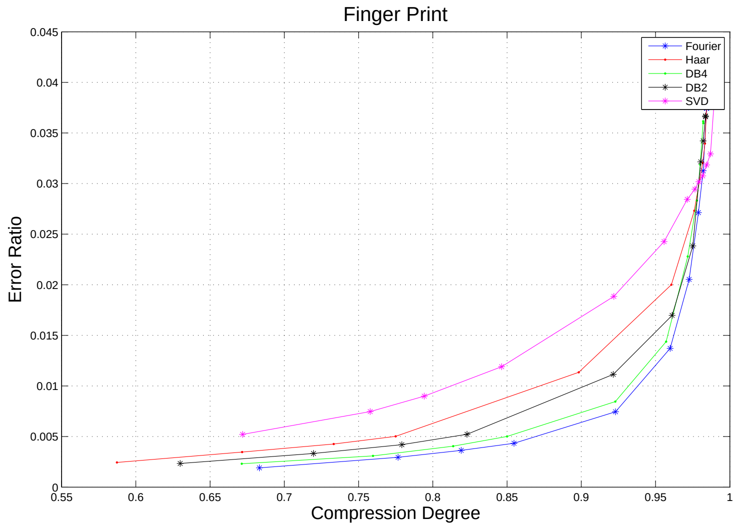

FIGURE 133. Five compressed methods of bird

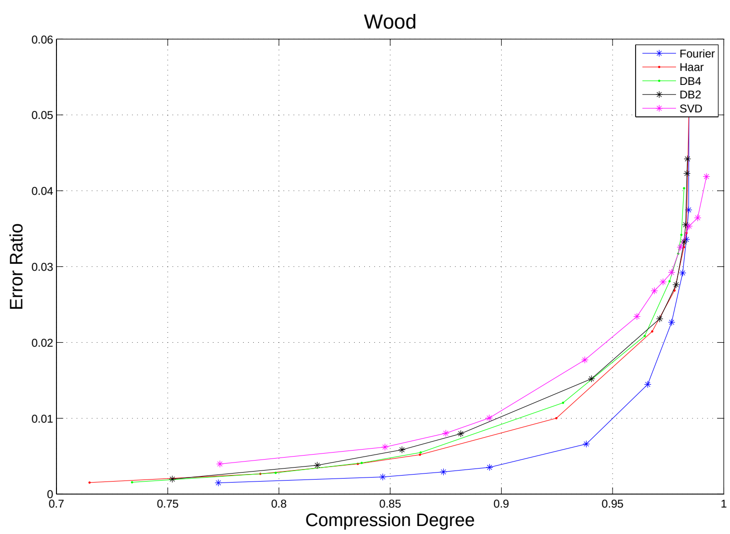

FIGURE 134. Five compressed methods of Wood

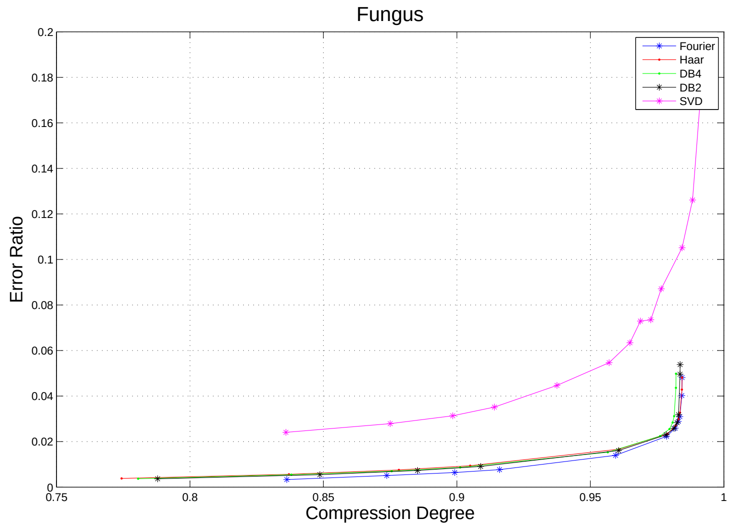

FIGURE 135. Five compressed methods of Fungus

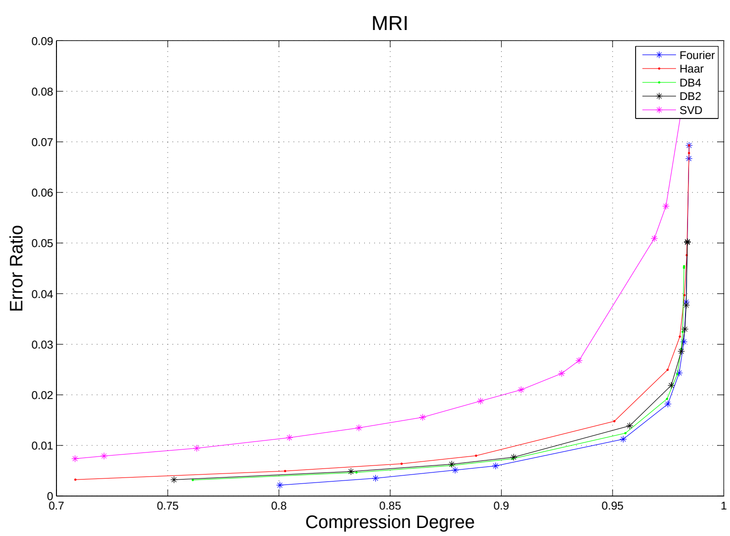

FIGURE 136. Five compressed methods of MRI

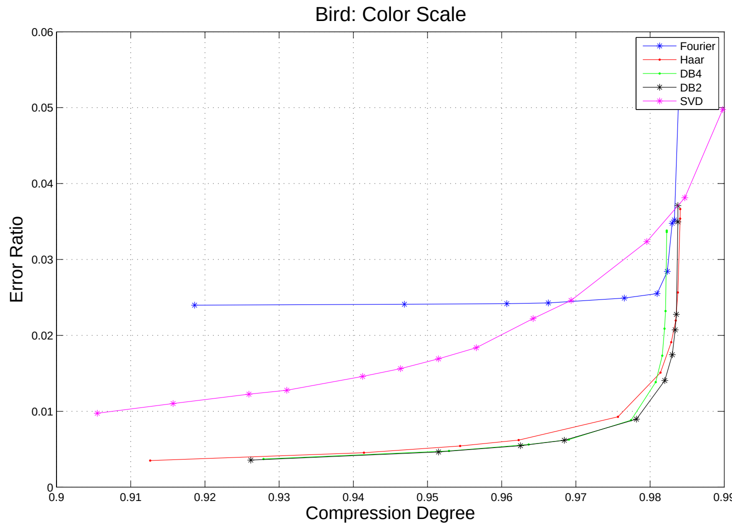

FIGURE 137. Five compressed methods of bird

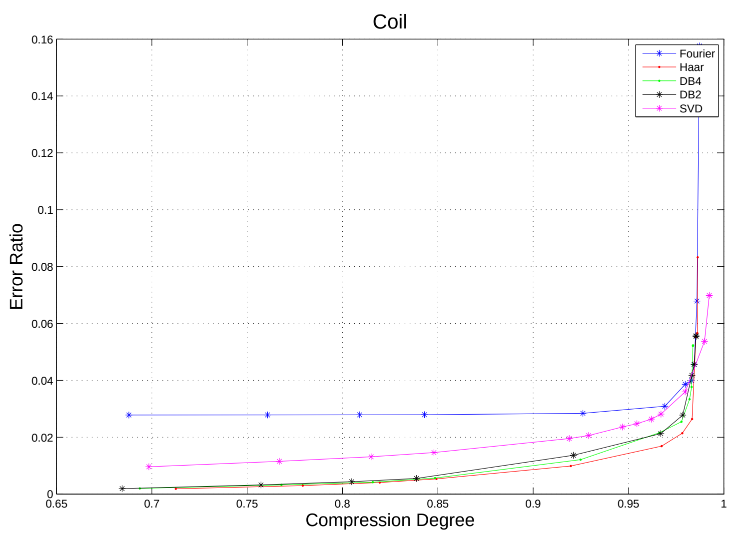

FIGURE 138. Five compressed methods of Coil

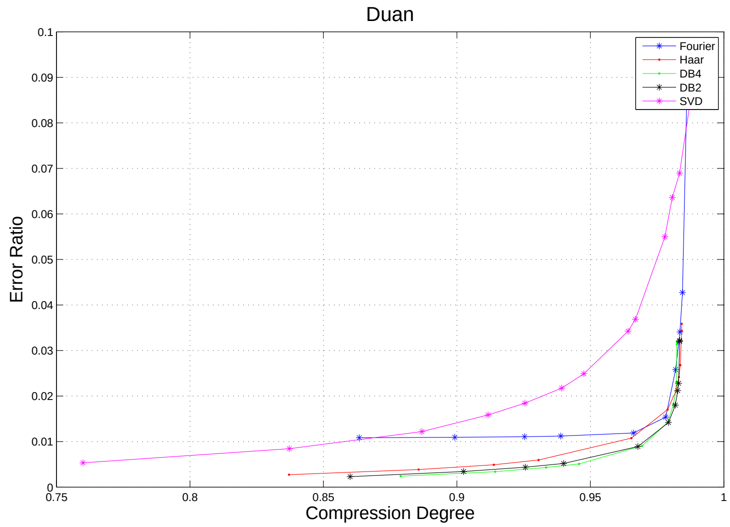

FIGURE 139. Five compressed methods of Duan

4.2.2. **The effect of compressed gray image versus color image.** Through compressing the 'bird' images, we found that even using the same methods to compress, the effect of gray one is different from the color one.

Below we start considering the **graybird**

FIGURE 140. A+, graybird, FFT, CR = 0.92396, ER = 0.023294

FIGURE 141. A, graybird, FFT, CR = 0.97081, ER = 0.02342

FIGURE 142. A-, graybird, FFT, CR = 0.9828, ER = 0.024201

FIGURE 143. A+, graybird, SVD, CR = 0.92337, ER = 0.012332

FIGURE 144. A, graybird, SVD, CR = 0.9719, ER = 0.026432

FIGURE 145. A-, graybird, SVD, CR = 0.98467, ER = 0.037873

FIGURE 146. A+, graybird, Haar, CR = 0.92738, ER = 0.0025672

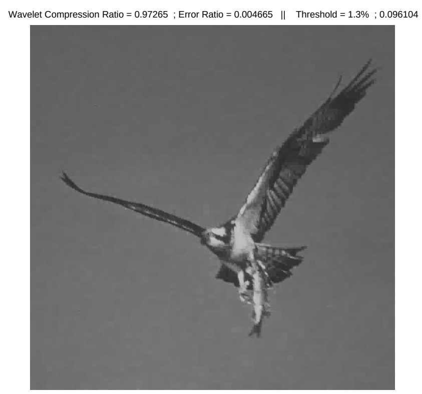

FIGURE 147. A, graybird, Haar, CR = 0.97265, ER = 0.004665

FIGURE 148. A-, graybird, Haar, CR = 0.98257, ER = 0.010394

FIGURE 149. A+, graybird, db2, CR = 0.9262, ER = 0.0023367

FIGURE 150. A, graybird, db2, CR = 0.9723, ER = 0.0037908

FIGURE 151. A-, graybird, db2, CR = 0.98265, ER = 0.0089934

FIGURE 152. A+, graybird, db4, CR = 0.92652, ER = 0.0021438

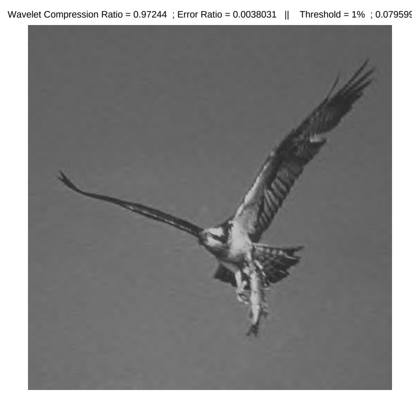

FIGURE 153. A, graybird, db4, CR = 0.97244, ER = 0.0038031

FIGURE 154. A-, graybird, db4, CR = 0.98225, ER = 0.020159

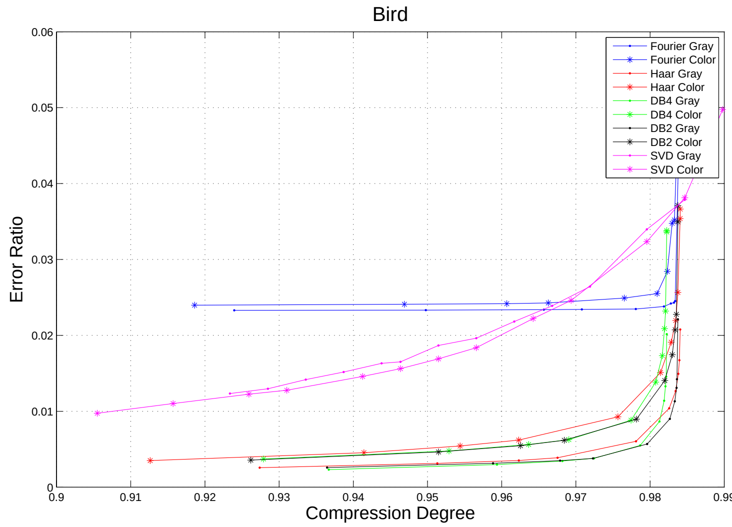

FIGURE 155. Five compressed methods of color bird and gray bird

In Figure 155 we can see that

1. The rough trends of the two images' methods are similar.
2. Mostly, the compression methods work better for the gray one then the color one.
3. SVD has an opposite behavior, it works better for the color image than for the gray one.

4.2.3. **Comparison of various parameters.** Up to now, we have calculated the compression degree and error ratio of three-level images for different methods. These two parameters can only show how deep we can compress the image. For human vision, in reality, a high quality image is demanded. Now a natural question is: How can one judge the quality of an image? MSE and PSNR answer to this question easily.

In statistics, the mean squared error,i.e. the L_2-error squared, see page 27, of an estimator is one of many ways to quantify the amount by which an estimator differs from the true value of the quantity being estimated. For a loss function, L_2 is called squared error loss. L_2 measures the average of the square of the "error." The error is the amount by which the estimator differs from the quantity to be estimated. L_2 is one of the ways to get the difference between compressed image and original one.

It is most easy to define PSNR: the peak signal-to-noise ratio, by the mean squared error.

$$PSNR = 20 \cdot log_{10}(\frac{MAX_I}{L_2}) \tag{113}$$

Here, MAX_I is the maximum possible pixel value of the image. In our project, the pixels are represented from 0 to 1, so this value is 1 here. The PSNR is most commonly used as a measure of quality of reconstruction of lossy compressions, such as image compression. The signal here is the original image, and the noise is the error introduced by compression. When comparing compression codes PSNR is used as an approximation to human perception of reconstruction quality, therefore in some cases one reconstruction may appear to be closer to the original than the another, even though it has a lower PSNR. Normally, a higher PSNR would indicate that the reconstruction is of higher quality. One has to pay extra attention to the range of validity of this metric. It is only conclusively valid when it is used to compare results from the same content.

Typical values for the PSNR in compressed image are between 30 and 50 dB, where higher is better. Acceptable values for wireless transmission quality loss are considered to be about 20 dB to 25 dB. When the two images are identical the MSE will be equal to zero, resulting in an infinite PSNR.

Now we get some figures and tables to show the data of seven images in different methods, and in this way we can get the point of L_2 and PSNR directly. In the pictures we use the notation MSE (mean square error) for the L_2-error squared.

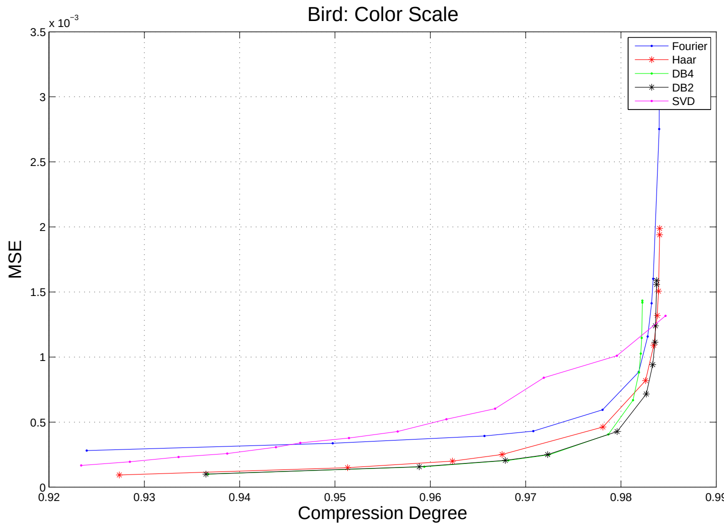

FIGURE 156. MSE-bird-Color

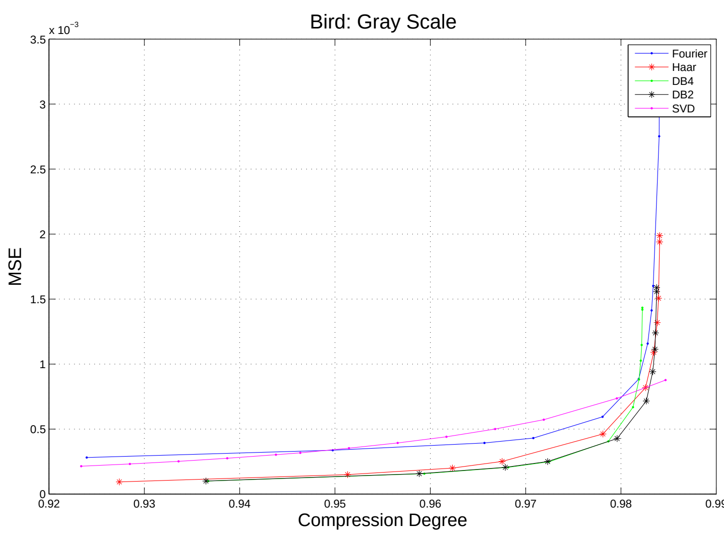

FIGURE 157. MSE-bird-Gray

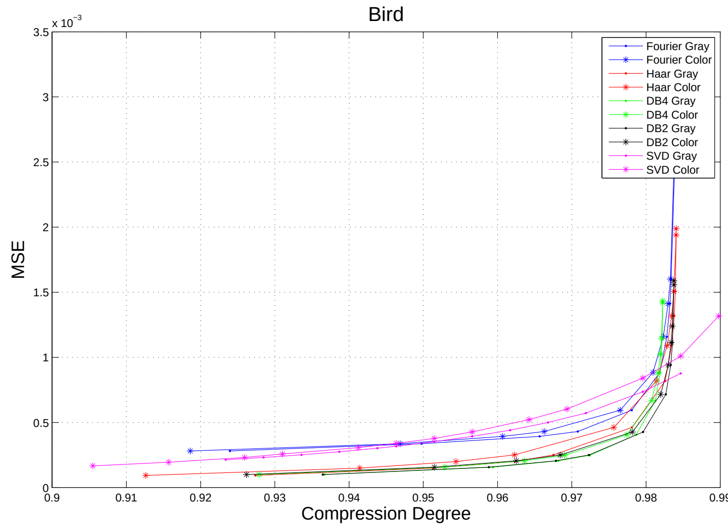

FIGURE 158. MSE-bird-Gray and Color

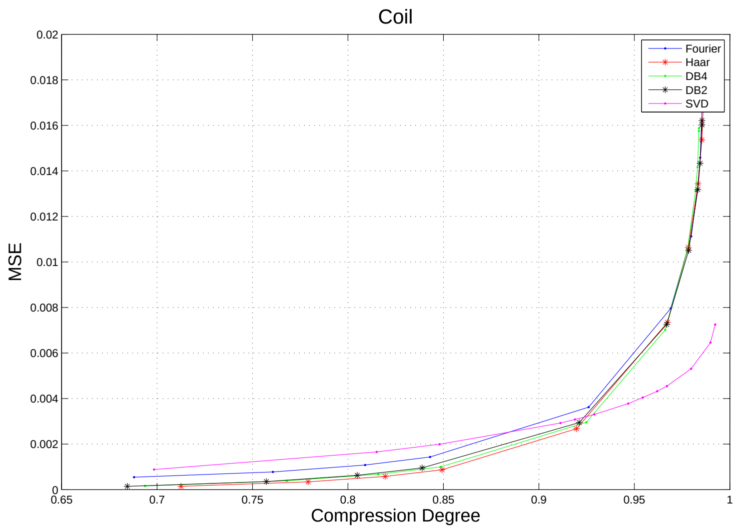

FIGURE 159. MSE-Coil

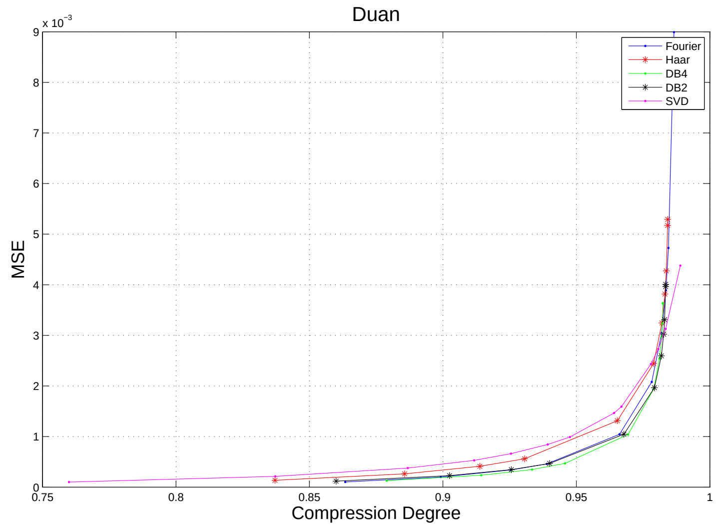

FIGURE 160. MSE-Duan

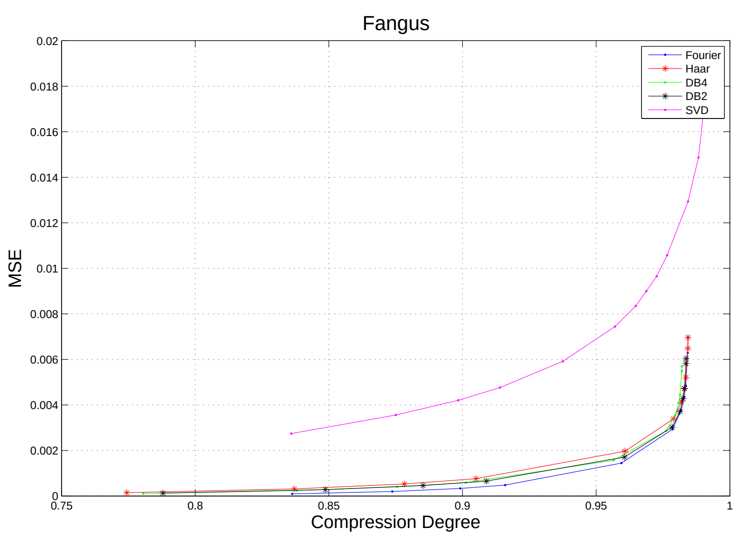

FIGURE 161. MSE-Fungus

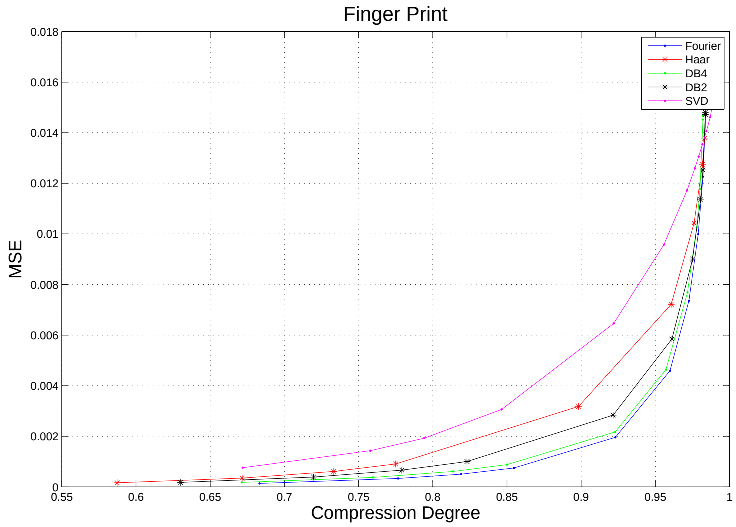

FIGURE 162. MSE-FingerPrint

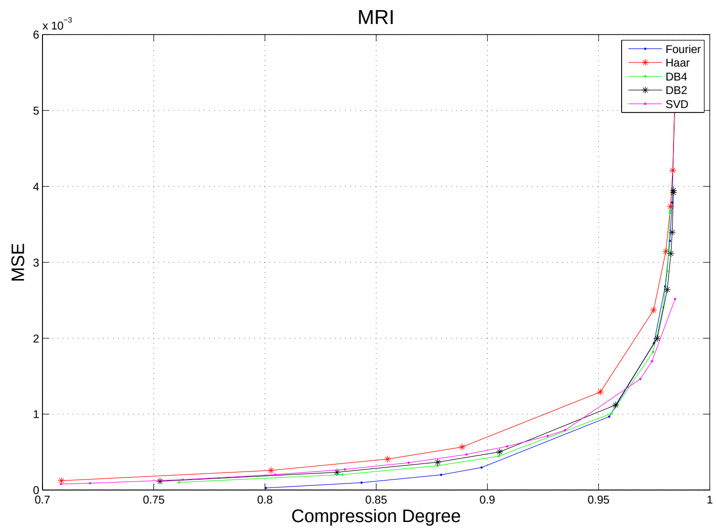

FIGURE 163. MSE-MRI

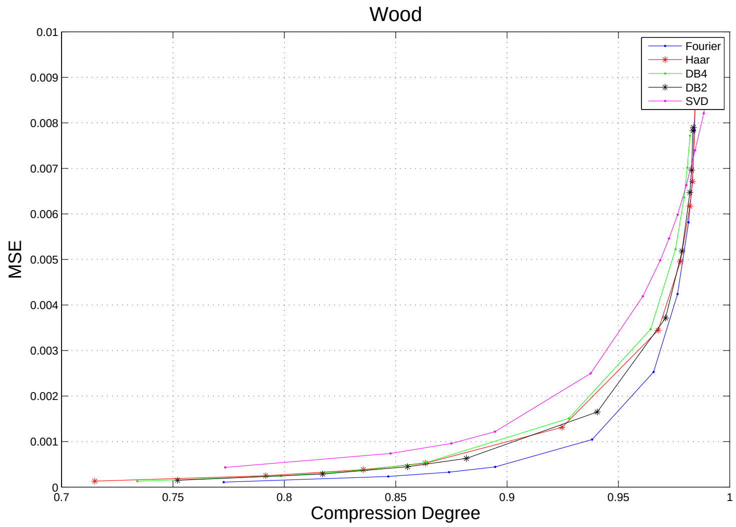

FIGURE 164. MSE-Wood

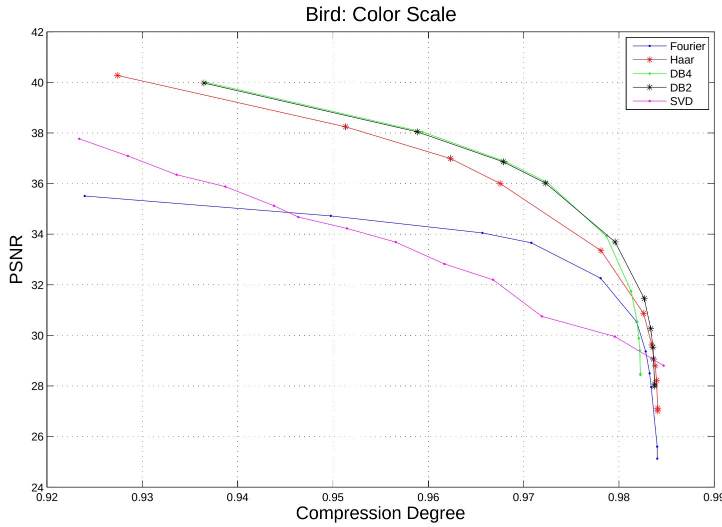

FIGURE 165. PSNR-bird-C

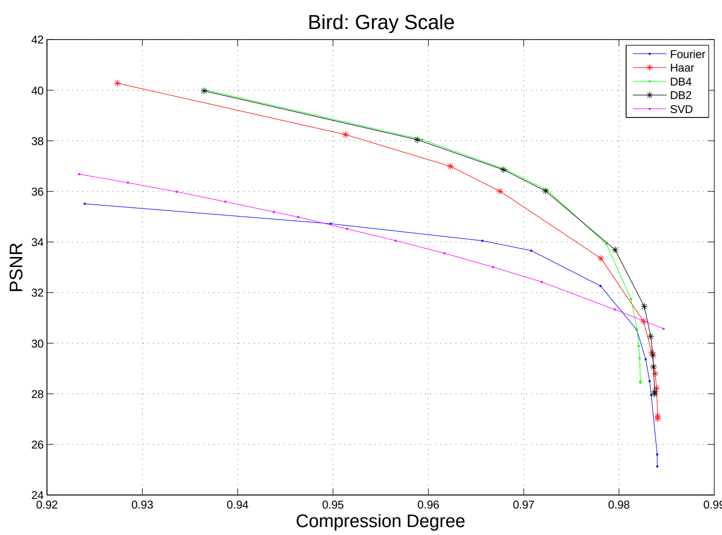

FIGURE 166. PSNR-bird-G

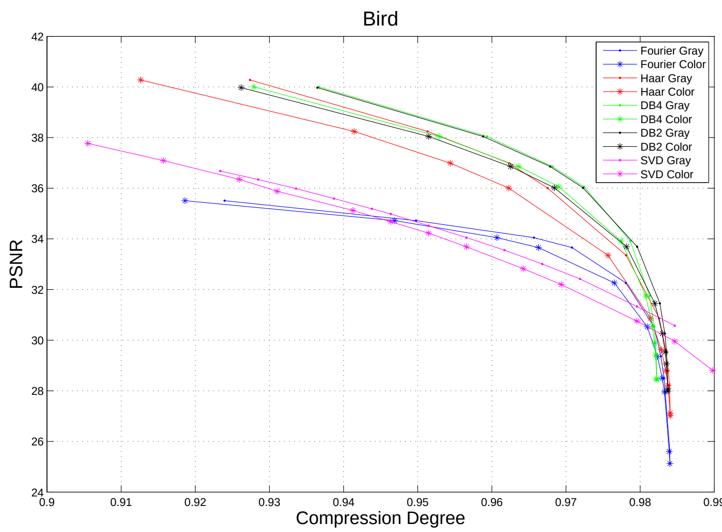

FIGURE 167. PSNR-bird-Gray and Color

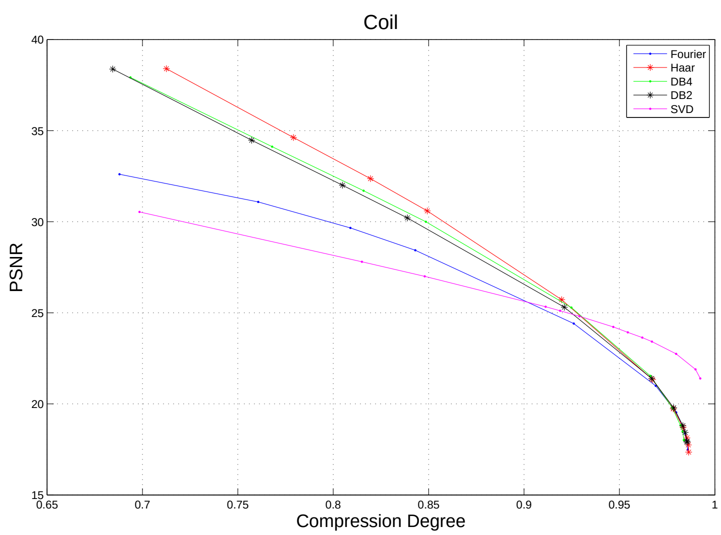

FIGURE 168. PSNR-Coil

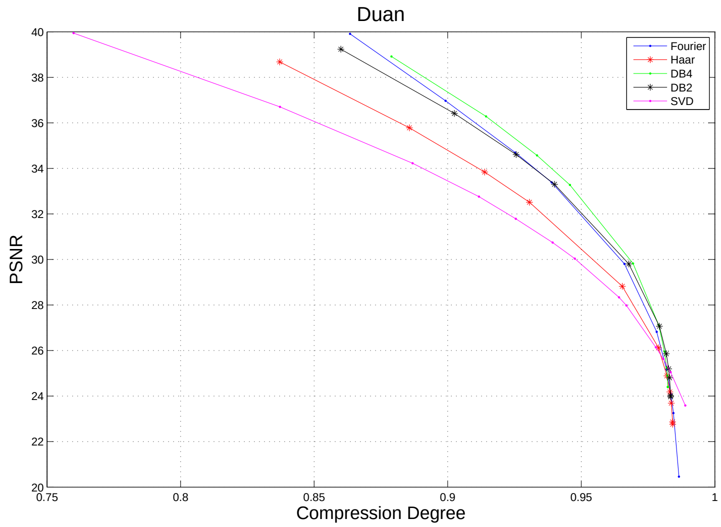

FIGURE 169. PSNR-Duan

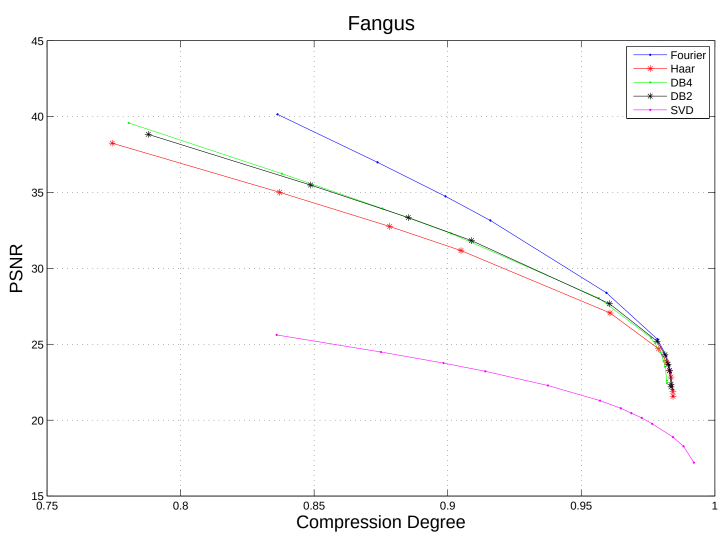

FIGURE 170. PSNR-Fungus

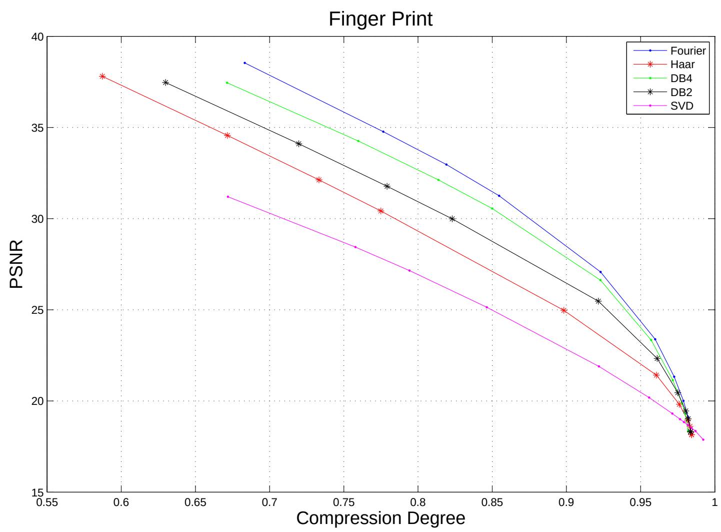

FIGURE 171. PSNR-FingerPrint

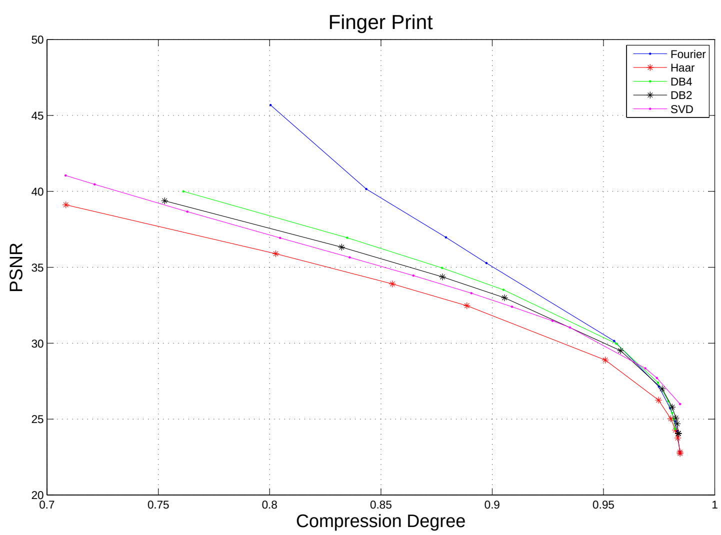

FIGURE 172. PSNR-MRI

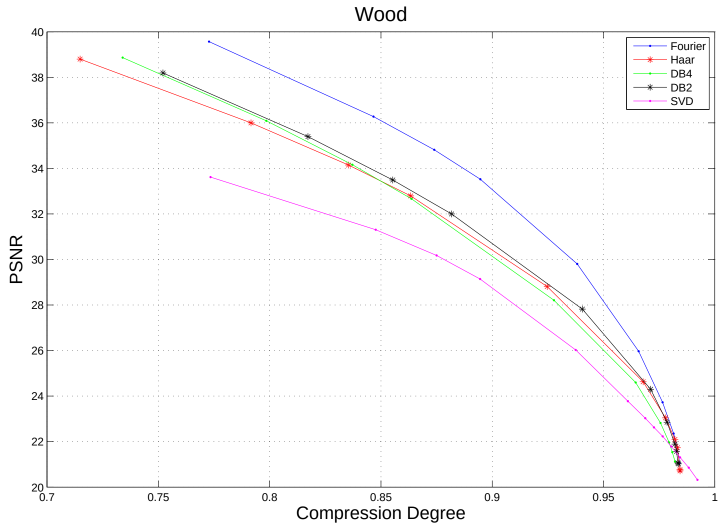

FIGURE 173. PSNR-Wood

CD ≈ 0.923	FFT	Haar	wavelet(db2)	wavelet(db4)	SVD
PSNR(Finger Print)	27.073	24.969	25.472	26.625	21.896
PSNR(Wood)	29.804	28.809	27.820	28.208	26.024
PSNR(Fungus)	33.155	31.171	31.828	32.295	23.220
PSNR(MRI)	30.146	28.889	29.507	29.967	35.652
PSNR(Bird-gray)	37.089	41.920	42.0877	42.381	36.682
PSNR(Bird-color)	35.508	40.275	39.974	40.004	37.770
PSNR(Coil)	24.409	25.719	25.310	25.288	25.112
PSNR(Duan)	34.691	33.841	34.608	34.572	34.22

TABLE 2. PSNR for images whose compression degrees are all around 0.923

CD ≈ 0.984	FFT	Haar	wavelet(db2)	wavelet(db4)	SVD
PSNR(Finger Print)	18.147	18.147	18.296	18.340204	17.875
PSNR(Wood)	20.717	20.717	21.025	21.125	20.323
PSNR(Fungus)	21.56	21.570	22.189	22.442	17.201
PSNR(MRI)	22.733	22.733	24.038	24.344	25.992
PSNR(Bird-gray)	28.038	30.836	31.575	31.979	30.571
PSNR(Bird-color)	25.128	27.015	27.990	28.436	28.806
PSNR(Coil)	18.363	18.135	18.436	18.487	23.643
PSNR(Duan)	23.253	23.690	24.801	24.898	27.981

TABLE 3. PSNR for images whose compression degrees are all around 0.984

4.2.4. **Integrating all methods.** PSNR and compression degree are two most important parameters to measure the quality and efficiency of compressed images. Now we integrate the images with different methods to see what happens to the PSNR value if we set compression degree to a constant (Tables 2 and 3). For each group of tables, the PSNR of three quality levels of images (A+, A and A-) will be compared together and for all compression methods having the same compression ratio. In this way, we see how the compressed images look like when they occupy similar amount of storage. In the tables, CD = Compression Degree.

Subsequently, we integrate the images with different methods to see what happens to the compression degree if PSNR is set to as a constant (Tables 4 and 5). For each group of tables, the compression degree of three quality levels of images (A+, A and A-) is compared together for all compression methods with the same PSNR. In this way, we see how much storage the compressed images occupy.

5. Discussion

5.1. **Comparisons between Fourier Transform, Wavelet Transform, and SVD.** From our experiments, we got a series of graphs to compare all five methods on each image. In this way we can easily predict the characteristic effects of compression methods. In those graphs, the x-axis is set as the compression degree and the y-axis is set as the error ratio.

PSNR $\approx$ 32.608	FFT	Haar	wavelet(db2)	wavelet(db4)	SVD
CD(Finger Print)	0.819	0.733	0.779	0.814	0.794
CD(Wood)	0.895	0.8634	0.882	0.864	0.895
CD(Fungus)	0.916	0.905	0.909	0.901	0.914
CD(MRI)	0.955	0.951	0.958	0.956	0.836
CD(Bird-gray)	0.982	0.983	0.983	0.981	0.946
CD(Bird-color)	0.977	0.976	0.978	0.977	0.941
CD(Coil)	0.688	0.713	0.684	0.694	0.911
CD(Duan)	0.939	0.930	0.940	0.946	0.912

TABLE 4. Compression Degree for images whose PSNR are all around 32.608

PSNR $\approx$ 28.038	FFT	Haar	wavelet(db2)	wavelet(db4)	SVD
CD(Finger Print)	0.923	0.898	0.922	0.923	0.922
CD(Wood)	0.938	0.925	0.941	0.928	0.938
CD(Fungus)	0.959	0.961	0.961	0.957	0.938
CD(MRI)	0.975	0.975	0.976	0.974	0.865
CD(Bird-gray)	0.984	0.984	0.984	0.982	0.985
CD(Bird-color)	0.983	0.983	0.983	0.982	0.957
CD(Coil)	0.843	0.849	0.839	0.849	0.937
CD(Duan)	0.966	0.965	0.968	0.969	0.926

TABLE 5. Compression Degree for images whose PSNR are all around 28.038

The roughly trend of those graphs are curves, showing how the error ratio is raised with growing compression degree.

Finger print, Fungus, MRI and wood are gray images. For these images DCT compression provides lowest error ratio among our considered the compression methods. Whereas, the outcome is different for color images: Bird, coil and Duan. Wavelet including Haar, db2, db4 work well in different images. Among all the wavelets we used, the performance of compression does not necessarily gets better with the raise of filter length. For example, in the color image coil, Haar is better than db2 and db4. The reason for this is that the longer the length a the wavelet filter is, the more details can be kept. The performance of SVD is as good as the first two methods. The error ratio of SVD compression is often exceeding the other two, except for Bird image. With the same compression degree, mostly, DCT and wavelet have lower error ratio than SVD. But they (DCT and wavelet) always stuck at certain compression degrees and can not compress the image any further. SVD method compresses the image more than DCT and wavelet. This can be explained easily according to the SVD theory: the compression degree is decided by the number of terms in the truncated SVD sum and this number can be very small. For Bird image, certain compression degrees, SVD even works better than DCT.

5.2. **Gray images versus Color images.** In the graphs comparing the compressed Bird images with gray and color scales, it is easily seen that the error ratio of the gray one is lower than the color one, except for the SVD case. The error ratio differences between wavelet compressed gray image and color image are especially obvious. The reason for these differences is based on the pixels of the images. The gray image has only one layer, whereas the color image has three layers to represent tricolor: red, green and blue (RGB). When the color image is compressed, all three layers are compressed simultaneously. This means that there is a risk of increasing error as the effect of accumulated layer errors.

Note that in the Figures 158 and 167 we have used different points than in the Figures 156 and 165, repectively. This is the resason for discripecies appearing in the corresponding figures.

5.3. **The relationship in parameters.** The graphs above show the rough trend of MSE and PSNR. While MSE becomes smaller with the decrease of the compression degree, the PSNR grows larger. This inverse relationship between MSE and PSNR is in accordance with the Equation of PSNR. Furthermore, MSE is just another way to check the quality of the compressed image, which is similar to error ratio. So the trends of MSE and error ratio are correlated. They also have some similar characteristics. For example, in those MSE graphs, DCT method provides the gray images with the lowest mean square error, except for the Bird image. For the wavelet methods, the images also get low errors. SVD is still not good here but images can be compressed further than in the previous two methods.

5.4. **Different methods fit different images.** Through the tables on PSNR and compression degree, we get some ideas about the compression methods that are suitable for each kind of image. Through the tables, we can see that the values of PSNR for image Bird, is obviously larger than others: over 35db for high compression level and over 25db for low compression level. Images Duan and MRI seem to work well too. This means that normally, for the common three layers color pictures, the quality of the compressed images are better than those of the corresponding gray one. With same compression degree, three layers image can provide more information than in the gray case, so that human vision will be more satisfied.

There is one case that we should pay a particular attention to: as the DCT has directional property, for the texture image with a similar directional pattern, such as the Wood image, DCT method has a much better performance than in the other images such as Finger print and Fungus. Its error ratio is much lower than the other two compression methods, and its PSNR is much higher.

Wavelet has a higher PSNR while the compression degree is good enough for the color images. SVD is good at a kind of image compression which can sacrifice the quality of the picture but on the other hand compresses up to the contour. For example, in medical image processing, sometimes the doctors just want to know the shape of tumor rather than the quality of the image. For this purpose, SVD will compress the images to a very high degree and save a huge amount of memory.

6. **Conclusions**

In this paper we focused on three image compression methods: Fourier, wavelet and singular value decomposition (SVD). Applying these methods to the selected gray and color images from different application fields, we compared them in terms of compression ratio, error ratio, peak signal-to-noise ratio (PSNR).

Discrete cosine transform (DCT) compression provides the gray images with the lowest error ratio, lowest MSE and the highest PSNR in all three compression ways, except for the gray image of the Bird. It does not perform as good as wavelet for Bird, Coil and Duan, but if performs better than SVD. As the DCT has directional property, the performance for the texture image with a similar directional pattern is much better and the error ratio is much lower than for the other two compression methods, and the PSNR is much higher.

Wavelet provides a high PSNR and its compression degree is sufficiently good for both gray and color images. Considering the very same image with the same compression degree, MSE and PSNR of the three wavelets used Haar, db2 and db4 in here are close to each others. In most cases, with the raise of filter length, the performance of compression gets better. But this rule breaks down for the color image of coil.

The performance of SVD is not as good as the other two compression methods. To get the same compression degree, the MSE error is much higher and PSNR is much lower than the results of the other two compression methods. However, SVD can compress the image much further while Fourier and Wavelet have a limitation of the maximal compression degree. In the application where only the contour matters while image quality is not so significant, SVD is a good choice. In the SVD case, compared to the other two methods, the image have somewhat low quality. On the other hand, the error ratio in SVD is much lower than the other two cases. Further SVD is more stable than DCT transform and wavelet transform.

References

[1] Mohammad Asadzadeh *lecture notes*, `http://www.math.chalmers.se/~mohammad/teaching/Fourier/LectureNotes_A3/traft_1.ps`

[2] Sarah Betz *Wavelet-Based Image Compression*, Elec 301 Final Project, 1994.

[3] Ingrid Daubechies *Ten lectures wavelets*, Siam, 1992.

[4] James Demmel *Applied Numerival Linear Algebra*, Siam, 1997.

[5] R. Fisher *Image Processing Learing Resources HIPR2 explore with java*, `http://homepages.inf.ed.ac.uk/rbf/HIPR2/hipr_top.htm`, 2003.

[6] G.G.Folland *Fourier analysis and its applications*, Wadsworth and Cole, 1992.

[7] Lewis *Image compression using the 2-D wavelet transform*, Dept. of Electr.Eng.Imperial Coll.London, 1992.

[8] Subhasis Saha *Image Compression - from DCT to wavelets: A Revies*, ACM/Crossroads/Xrds6-3, 1997.

[9] Andrew B. Watson *Wavelet-Based Image Compression*, NASA Ames Research Center, Mathematica Journal, 4(1), 1994.

[10] www.wikipedia.com