City University Of Hong Kong

Final Year Project

Wavelet Transform and Its Applications to Image Processing

Author:

Zhigang Wang

SID:52165928

Supervisor:

Dr. Xiaosheng Zhuang

A thesis submitted in fulfilment of the requirements

for the degree of Bachelor of Science

in the

Department of Mathematics

May 2014

CITY UNIVERSITY OF HONG KONG

Abstract

Department of Mathematics

Bachelor of Science

Wavelet Transform and Its Applications to Image Processing

by Zhigang WANG

In this paper, we study wavelet transform and its application in image denoising and image inpainting. In the denoising and inpainting experiments, two types of filter banks named orthogonal filter bank and tight framelet filter bank respectively are considered. Also, different values of the threshold function and decomposition levels are carefully selected as they are the key to optimal results attained in extensive simulations of image denoising and impainting. Section 2 demonstrates the main methods and algorithms used in the experiments and Section 3 concentrates on comparing the results of different filter banks. This paper focuses on the property of different filter banks and shows the advantages of redundancy property of tight framelet filter banks in image inpainting.

Contents

Abstract i

Contents ii

List of Figures iv

1 Introduction 1
- 1.1 Introduction to Wavelet Transform . . . 1
  - 1.1.1 The Scaling Function . . . 2
  - 1.1.2 The Wavelet Functions . . . 2
- 1.2 Introduction to Filter Banks and Discrete Wavelet Transform . . . 3
  - 1.2.1 Analysis - From Fine Scale to Coarse Scale . . . 4
  - 1.2.2 Synthesis - From Coarse Scale to Fine Scale . . . 5
  - 1.2.3 Discrete Wavelet Transform . . . 5

2 Methods 6
- 2.1 Discrete Framelet Transform . . . 6
  - 2.1.1 Mallat's algorithm . . . 6
  - 2.1.2 One Level Discrete Framelet Transform . . . 7
  - 2.1.3 Multi-level discrete wavelet transform . . . 8
  - 2.1.4 Implement the algorithm on Images . . . 9
- 2.2 Wavelet Transform in Image Denoising . . . 10
  - 2.2.1 Problem Formulation . . . 11
  - 2.2.2 Wavelet Thresholding . . . 12
  - 2.2.3 Algorithm for image denoising . . . 13
- 2.3 Wavelet Transform in Image Inpainting . . . 14
  - 2.3.1 Problem Formulation . . . 14
  - 2.3.2 Algorithm for image inpainting . . . 15

3 Results 17
- 3.1 Image Denoising . . . 17
  - 3.1.1 PSNR results in image denoising . . . 17
  - 3.1.2 Performance analysis . . . 20
- 3.2 Image Inpainting . . . 22
  - 3.2.1 PSNR results in image inpainting . . . 22

3.2.2 Performance analysis 24

4 Conclusion **26**

A Some Important Matlab codes **30**

List of Figures

1.1 A Sine Wave 1
1.2 Daubechies' Wavelet ψ_{D20} 1

2.1 One Level Analysis Bank 7
2.2 One Level Synthesis Bank 7
2.3 Implementation of one-level discrete wavelet transform 8
2.4 Two-level Analysis Tree 8
2.5 Two-level Discrete Framelet Transform 9
2.6 Sub-bands of the 2D orthogonal wavelet transform 10
2.7 Image Denoising using hard and soft thresholding with $\sigma = 50$ . . . 14
2.8 Image Inpainting using hard and soft thresholding 15

3.1 Denoising Result with $\sigma = 30$ for Barbara. Top: Original images and Zoom. Middle: 'db2' Denoising. Bottom: 'tf2' denoising. 21
3.2 Mask Figures 22
3.3 Inpainting Result with 'text' mask for Boat. Top: Original images and Zoom. Middle: 'db1' Inpainting. Bottom: 'tf1' inpainting. . . . 25

Section 1 Introduction

1.1 Introduction to Wavelet Transform

Generally speaking, a wave is defined to be an oscillating function of time or space, for example, sinusoid and cosinousoid are typical waves used in many fields such as Fourier analysis, which expands signals in terms of sinusiods. In reality, fourier transform has proven to be extremely useful in mathematics, science and engineering especially in dealing with periodic signals. A wavelet is a "small wave" with its energy concentrated at a small area and is examined to be more efficient in the analysis of nonstationary phenomena. Figure 1.1 and Figure 1.2 illustrate the main difference between a wave and a wavelet, where a wavelet has its finite energy concentrated around a point compared to wave.

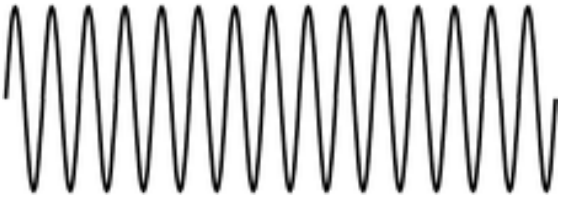

FIGURE 1.1: A Sine Wave

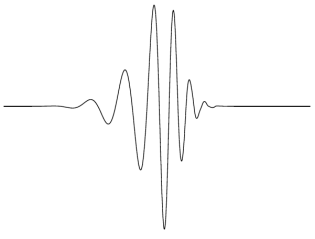

FIGURE 1.2: Daubechies' Wavelet ψ_{D20}

1.1.1 The Scaling Function

Scaling functions are defined in order to implement the idea of multiresolution and wavelets are defined in terms of the scaling functions. Moreover, a two-dimensional family of scaling functions are transformed from basic scaling function through translation and scaling by

$$\varphi_{j,k}(t) = 2^{j/2}\varphi(2^j t - k) \qquad k \in \mathbb{Z} \qquad \varphi \in L^2 \tag{1.1}$$

and the subspace of $L^2(\mathbb{R})$ spanned by these functions is

$$\mathcal{V}_j = \overline{Span_k\{\varphi_{j,k}(t)\}} \tag{1.2}$$

In addition, scaling functions with a lower scale level can be expressed in terms of those at a higher scale by

$$\varphi(t) = \sum_n h_0(n)\sqrt{2}\varphi(2t - n) \tag{1.3}$$

1.1.2 The Wavelet Functions

Instead of using $\varphi_{j,k}(t)$ and incresing j to increase the size of the subspace spanned by those scaling functions, a slightly different set of functions $\psi_{j,k}(t)$ which span the differences between the spaces spanned by the various scales of the scaling

function is defined by

$$\psi(t) = \sum_n h_1(n)\sqrt{2}\varphi(2t - n) \tag{1.4}$$

A signal can be better described by the combination of scaling functions and wavelet functions, especially when the scaling functions and wavelets are orthogonal. Therefore, any function $f(t) \in L^2(\mathbb{R})$ can be represented as

$$f(t) = \sum_k c_{j_0}(k)\varphi_{j_0,k}(t) + \sum_k \sum_{j=j_0}^{\infty} d_j(k)\psi_{j,k}(t) \tag{1.5}$$

1.2 Introduction to Filter Banks and Discrete Wavelet Transform

The coefficients $h_0(n)$ and $h_1(n)$ in the equations (1.3) and (1.4) can be viewed as digital filters and the coefficients $c_{j_0}(k)$ and $d_j(k)$ in equations (1.5) can be viewed as digital signals. In signal processing, filter banks contain analysis bank and synthesis bank. Orthogonal wavelet filter banks and tight framelet filter banks will be introduced in the next section and the efficiency of different filter banks will be studied through various experiments.

1.2.1 Analysis - From Fine Scale to Coarse Scale

In the process of Decomposition, analysis bank separates the input signal into multiple components with each one carrying different frequency sub-band of the original signal. These multiple components (a collection of sub-signals) can be used to emphasize aspects of the original signal and may help us work with the original signal in an easier and more efficient way. For instance, analysis banks are widely used in signal compression since most useless information can be easily discarded. Given that $\varphi_{j,k}(t)$ and $\psi_{j,k}(t)$ are orthonormal, the j^{th} level of scaling coefficients are derived from

$$c_j(k) = \langle f(t), \varphi_{j,k}(t) \rangle = \int f(t) 2^{j/2} \varphi(2^j t - k) dt \tag{1.6}$$

Substituting $\varphi_{j,k}(t)$ with equation (1.1) giving the scaling coefficients

$$c_j(k) = \sum_m h_0(m - 2k) c_{j+1}(m) \tag{1.7}$$

The corresponding formula for the wavelet coefficients can be derived in the same way

$$d_j(k) = \sum_m h_1(m - 2k) c_{j+1}(m) \tag{1.8}$$

1.2.2 Synthesis - From Coarse Scale to Fine Scale

The output of analysis can be reconstructed in the process of Reconstruction with the synthesis bank. Moreover, the desire for perfect reconstruction (i.e., the output signal is exactly the same as the input signal) imposes extra constraints on the analysis and synthesis filters.

$$c_{j+1}(k) = \sum_m c_j(m)h_0(k-2m) + \sum_m d_j(m)h_1(k-2m) \tag{1.9}$$

1.2.3 Discrete Wavelet Transform

The classical discrete wavelet transform (DWT) provides means of implementing analysis on the basis of filter banks with the attribute of perfect reconstruction. It has proven to be practically efficient in the process of a certain classes of signals, especially for piecewise smooth signals which will be presented in the following sections.

Section 2 Methods

2.1 Discrete Framelet Transform

2.1.1 Mallat's algorithm

The Mallat's algorithm was formally developed in 1988 by Mallat which proves to be a fast algorithm for wavelet decomposition and reconstruction. In the case of discrete wavelet transform, suppose that the length of the signal is $N = 2^J$, the transform can be efficiently computed by implementing Mallat's algorithm with the time complexity $O(N)$. Basically, the algorithm is a classical scheme for deriving the sub-signals using a set of low pass and high pass filters followed by a down-sampling process. To be specific, the low and high pass filters are predetermined by the mother wavelet. Moreover, the low pass filter generates the approximation coefficients and the output of the high pass filters is named detail coefficients. Figure 2.3 and Figure 2.5 shows the diagram of implementing the Mallat's algorithm. After the process of decomposition, the original signal is subsequently decomposed into several sub-signals with different scales.

2.1.2 One Level Discrete Framelet Transform

Equation (1.7) and equation (1.8) indicate that the scaling and wavelet coefficients at level of $j-1$ can be derived from convolving the expansion coefficients at scale j by the coefficients $h_0(-n)$ and $h_1(-n)$ then down-sampling by two, which means taking even terms in the signal sequence and discard the odd terms. Namely, the scale-j coefficients are filtered by two digital filters $h_0(n)$ and $h_1(n)$ and down-sampling the results gives the expansion coefficients at level of $j-1$.(as Figure 2.1 shows)

For the synthesis procedure, it is required to first up-sampling by two and then filtering, where up-sampling by two means inserting zeros between each of the original signal sequence.(as Figure 2.2 shows)

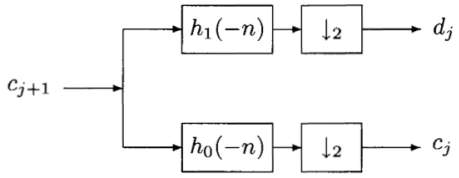

FIGURE 2.1: One Level Analysis Bank

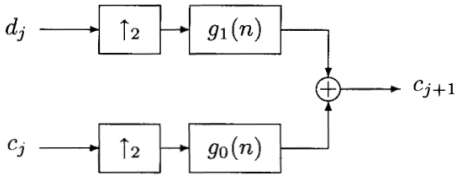

FIGURE 2.2: One Level Synthesis Bank

As illustrated in section 1.2, one-level standard discrete framelet transform includes two separate steps named decomposition and reconstruction. Figure 2.3 shows the implementation of one-level discrete framelet transform with a dual framelet filter bank $\{u_0, ..., u_s\}, \{\tilde{u}_0, ..., \tilde{u}_s\}$, the analysis filter $\{u_i\}$ plays the role of filtering the input signal while $\{\tilde{u}_i\}$ is the synthesis filter. The sub-signals are

down-sampled after filtered so that the data rates will be the same in the sub-signals as in the original signal. To ease notation, the complex conjugate sequence of u reflected about the origin is denoted as v^*, which means $u^*(k) := \overline{u(-k)}$

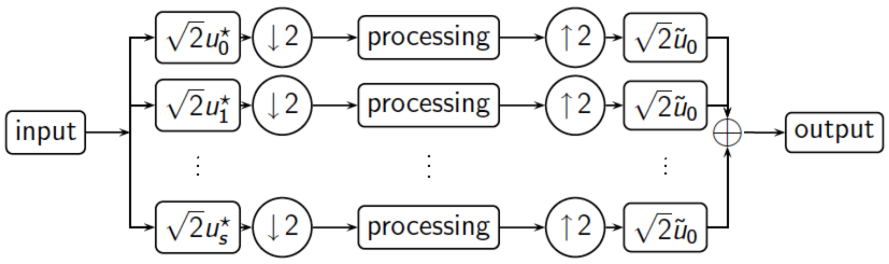

FIGURE 2.3: Implementation of one-level discrete wavelet transform

2.1.3 Multi-level discrete wavelet transform

The filtering and down-sampling procedure can be repeated for several times on the scaling coefficients to give the multi-level structure as illustrated in Figure 2.4. Note that the total number of the sub-signal data will be the same as the number in original data as a result of the down-sampling process. Iterating the filter bank generates more subsets of sub-signals which can be reconstructed through the multi-level synthesis structure later on.

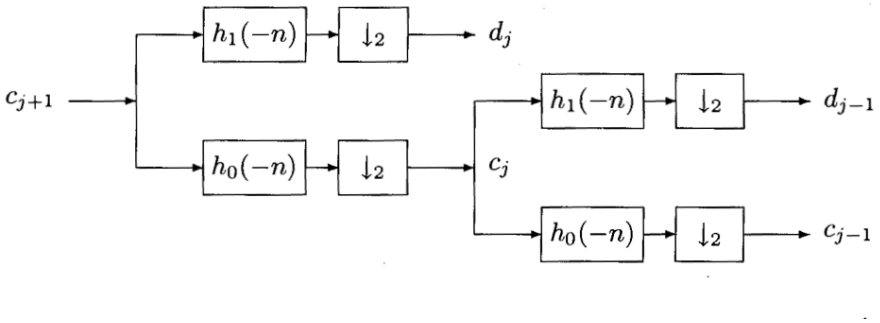

FIGURE 2.4: Two-level Analysis Tree

Figure 2.5 shows the implementation of a two-level discrete framelet transform with a dual framelet filter bank $\{a, b_1, ..., b_s\}, \{\tilde{a}, \tilde{b}_1, ..., \tilde{a}_s\}$, also noting the value of the decomposition levels and of the reconstruction levels for a signal should be the same and is particularly determined by the user according to the characters of the original signal and the expected result.

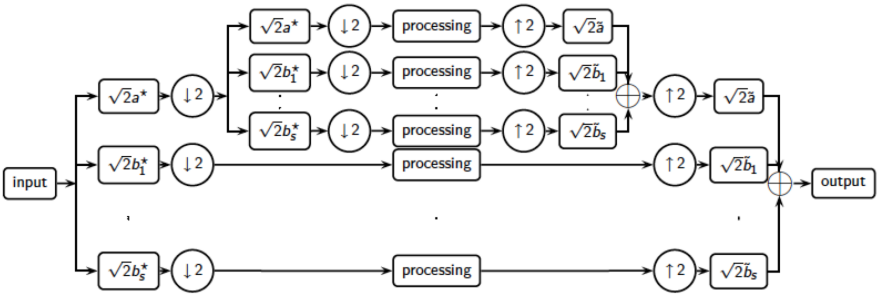

FIGURE 2.5: Two-level Discrete Framelet Transform

2.1.4 Implement the algorithm on Images

Images is interpreted as a two dimensional matrix with each element representing a pixel in the original image. Therefore, implementing discrete wavelet transform on images requires processing wavelet transform on rows and columns of the 2D matrix. Algorithm 1 briefly illustrates the method for implementing discrete framelet transform on images .

Figure 2.6 shows the output of the decomposition process, the sub-bands are organized as rows and columns of the new matrix. With the low pass filter denoted as LP and the high pass filters denoted as HP, it is convenient to denote the output as shown in Figure 2.6. The sub-band LL_3 is called the low resolution residual and the others are named details.

Algorithm 1 Discrete Framelet Transform on Images

```
// Transform (Decomposition)
parse image signal into a matrix C
for each row in C do
   decompose the row signal according to levels of scale
   sort the data into a row and store it into a new matrix D
end for
for each column in C do
   decompose the column signal according to the same levels of scale
   sort the data into a column and store it into D
end for
// Inverse Transform (Reconstruction)
for each column in D do
   using synthesis bank to reconstruct the column signal from the column in D
end for
for each row in D do
   using synthesis bank to reconstruct the row signal from the row in D
end for
```

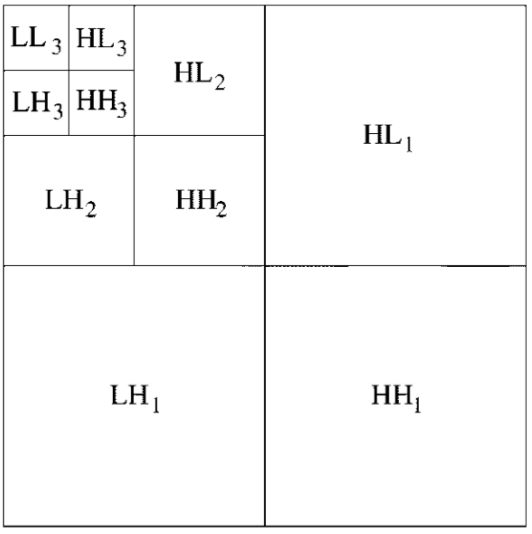

FIGURE 2.6: Sub-bands of the 2D orthogonal wavelet transform

2.2 Wavelet Transform in Image Denoising

It seems often that an image is corrupted by noise when it is transmitted. In that case, the main goal of image denoising is to discard the noise and retain the important signal features as much as possible. Recent work on image denoising through

wavelet thresholding has indicated that various wavelet thresholding schemes and different thresholding values for denoising perform well and some of them have near-optimal properties.

2.2.1 Problem Formulation

Assume that there are n noisy samples of function $f(t)$ in total, then denoising problem is usually posed as follows.

$$y_i = f(t_i) + \sigma\varepsilon_i, \qquad i = 1, ..., n \tag{2.1}$$

Here ε_i are independent and identically normal distributed $N(0, 1)$ and σ is called the noise level. As described, our goal is to discard the noise as much as possible and recover the function f, where the optimization criterion is the mean squared error (MSE). Namely, it is required to optimize $\hat{f}$ such that $\|\hat{f} - f\|_2$ is minimized.

In the 2D scenario, peak signal-to-noise ratio, often abbreviated PSNR, is often used as the optimization criterion. PSNR is directly related to MSE and is defined to be

$$PSNR = 10 \cdot log_{10}(\frac{MAX_I^2}{MSE}) \tag{2.2}$$

where MAX_I is the maximum possible pixel value of the image.

2.2.2 Wavelet Thresholding

Usually the output coefficients of the wavelet transform are sparse, in other words, most values in the coefficients are approximately zero in wavelet transform without the noise. Therefore, in the case of wavelet transform with noise, coefficients with small magnitude are typically noise and ought to be discarded in the denoising problem. The value used to identify noise in this approach is called threshold value and it determines whether a coefficient should be retained. The threshold value denoted by λ is usually determined by the decomposition level j, e.g. $\lambda = \lambda(j)$

In practice, the thresholding method is only applied to the detail coefficients rather than the approximation coefficients since the approximation coefficients usually contain important components of the signal. Hard thresholding and soft thresholding are often used to thresholding the wavelet coefficients and the rules are given respectively as follows

$$\delta_\lambda^H(d_{jk}) = \begin{cases} 0 & |d_{jk}| \leqslant \lambda \\ d_{jk} & |d_{jk}| > \lambda \end{cases} \tag{2.3}$$

$$\delta_\lambda^S(d_{jk}) = \begin{cases} 0 & |d_{jk}| \leqslant \lambda \\ d_{jk} - \lambda & d_{jk} > \lambda \\ d_{jk} + \lambda & d_{jk} < -\lambda \end{cases} \tag{2.4}$$

2.2.3 Algorithm for image denoising

Algorithm 2 Image Denoising

Input:
Original Image C; noise level σ; Decomposition level l; threshold value λ
Output:
Image D Recovered from the noise; PSNR value;
parse image signal into a matrix C
add noise to the image C according to σ
for each row in C **do**
  decompose the row signal according to l
  sort the data into a row and store it into a new matrix D
end for
for each column in C **do**
  decompose the column signal according to l
  sort the data into a column and store it into D
end for
hard/soft thresholding the matrix D according to λ
// Inverse Transform (Reconstruction)
for each column in D **do**
  using synthesis bank to reconstruct the column signal from the column in D
end for
for each row in D **do**
  using synthesis bank to reconstruct the row signal from the row in D
end for
Calculate PSNR for C and D

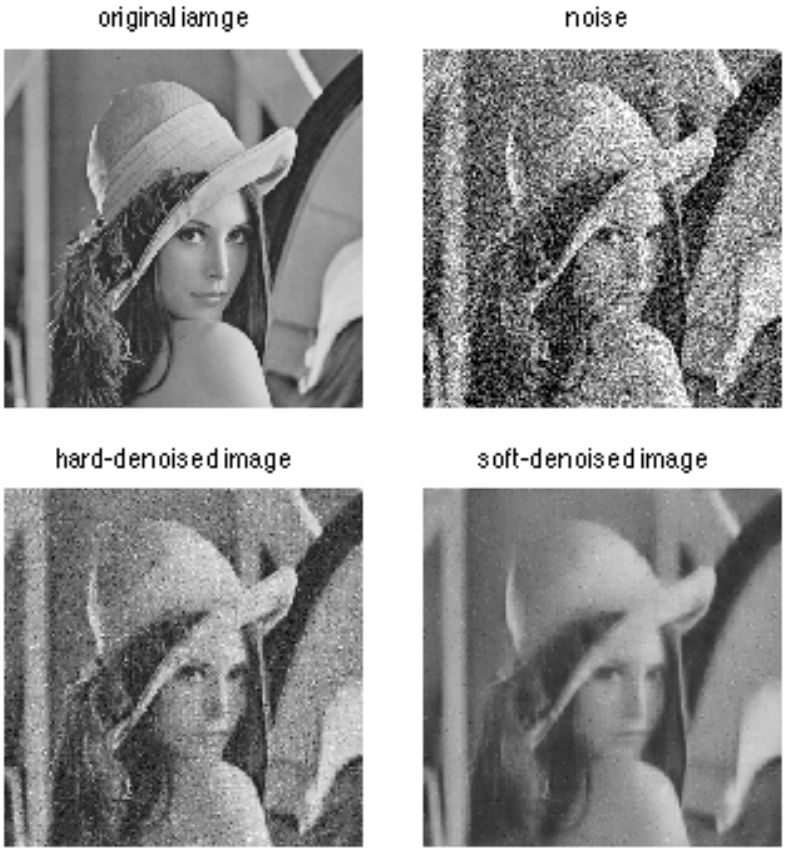

FIGURE 2.7: Image Denoising using hard and soft thresholding with $\sigma = 50$

2.3 Wavelet Transform in Image Inpainting

2.3.1 Problem Formulation

Typically, inpainting is described as the way reconstructing deteriorated part of an image. An effective inpainting process involves application of sophisticated algorithms to replace the corrupted part of the image data. Compared to image denoising, a mask matrix representing the corrupted area of the image is required for image inpainting.

Given an image data matrix C and a region Ω inside it, the inpainting problem consists in reconstructing the data in Ω in order to eliminate some outstanding difference between the region and its surroundings. A complete process of image inpainting usually requires implementing the wavelet transform recursively for several times in order to reconstruct the image perfectly. Figure 2.8 shows the ouput of the implementation for image inpainting using both hard and soft thresholding.

2.3.2 Algorithm for image inpainting

FIGURE 2.8: Image Inpainting using hard and soft thresholding

Algorithm 3 Image Inpainting

Input:
Original Image C; Mask Matrix M;Recursion times n; Decomposition level l; threshold value λ

Output:
Reconstructed Image; PSNR value;
parse image signal into a matrix C
read a mask M
C = A.*B is the image with missing information
for i = 1:n **do**
  for each row in C **do**
    decompose the row signal according to l
    sort the data into a row and store it into a new matrix D
  end for
  for each column in C **do**
    decompose the column signal according to l
    sort the data into a column and store it into D
  end for
  hard/soft thresholding the matrix D according to λ
  // Inverse Transform (Reconstruction)
  for each column in D **do**
    using synthesis bank to reconstruct the column signal from the column in D
  end for
  for each row in D **do**
    using synthesis bank to reconstruct the row signal from the row in D
  end for
  Update the missing information of C by $C_{new} = (1 - M). * D + M. * C$ and replace C by C_{new}
  replace threshold value λ by a smaller value
end for
Calculate PSNR for C

Section 3 Results

3.1 Image Denoising

3.1.1 PSNR results in image denoising

In order to assess the performance of various filter banks in image denoising, three images named Lena, Boat and Barbara respectively were chosen as the original image. Moreover, five different filter banks including two Daubechies filters and three tight framelet filters. In the following analysis, the "Haar" wavelet filter is referred as 'db1' and the Daubechies 4-tap filter is referred as 'db2'. Also, 'tf1','tf2' and 'tf3' are representing three different kinds of tight framelet wavelet filters.

- Orthogonal Filter Bank 'db1'
  - $h_0 = \{\frac{1}{2}, \frac{1}{2}\}$ $\quad h_1 = \{-\frac{1}{2}, \frac{1}{2}\}$
  - $g_0 = \{\frac{1}{2}, \frac{1}{2}\}$ $\quad g_1 = \{-\frac{1}{2}, \frac{1}{2}\}$
- Orthogonal Filter Bank 'db2'
  - $h_0 = \{\frac{1-\sqrt{3}}{8}, \frac{-3+\sqrt{3}}{8}, \frac{3+\sqrt{3}}{8}, \frac{-1-\sqrt{3}}{8}\}$ $\quad h_1 = \{\frac{1+\sqrt{3}}{8}, \frac{3+\sqrt{3}}{8}, \frac{3-\sqrt{3}}{8}, \frac{1-\sqrt{3}}{8}\}$

- $g_0 = \{\frac{1-\sqrt{3}}{8}, \frac{-3+\sqrt{3}}{8}, \frac{3+\sqrt{3}}{8}, \frac{-1-\sqrt{3}}{8}\}$ $\quad g_1 = \{\frac{1+\sqrt{3}}{8}, \frac{3+\sqrt{3}}{8}, \frac{3-\sqrt{3}}{8}, \frac{1-\sqrt{3}}{8}\}$

- Tight Framelet Filter Bank 'tf1'

  - $h_0 = \{\frac{1}{4}, \frac{1}{2}, \frac{1}{4}\}$ $\quad h_1 = \{-\frac{\sqrt{2}}{4}, 0, \frac{\sqrt{2}}{4}\}$ $\quad h_2 = \{-\frac{1}{4}, \frac{1}{2}, -\frac{1}{4}\}$

  - $g_0 = \{\frac{1}{4}, \frac{1}{2}, \frac{1}{4}\}$ $\quad g_1 = \{-\frac{\sqrt{2}}{4}, 0, \frac{\sqrt{2}}{4}\}$ $\quad g_2 = \{-\frac{1}{4}, \frac{1}{2}, -\frac{1}{4}\}$

- Tight Framelet Filter Bank 'tf2'

  - $h_0 = \{\frac{5}{1024}, 0, -\frac{63}{1024}, \frac{75}{1024}, \frac{495}{1024}, \frac{495}{1024}, \frac{75}{1024}, -\frac{63}{1024}, 0, \frac{5}{1024}\}$

    $h_1 = \{0, 0, -\frac{3\sqrt{15}}{512}, 0, \frac{22\sqrt{15}}{512}, -\frac{45\sqrt{15}}{512}, \frac{45\sqrt{15}}{512}, -\frac{22\sqrt{15}}{512}, 0, \frac{3\sqrt{15}}{512}\}$

    $h_2 = \{\frac{5}{1024}, 0, -\frac{117}{1024}, \frac{75}{1024}, \frac{315}{1024}, -\frac{315}{1024}, -\frac{75}{1024}, \frac{117}{1024}, 0, -\frac{5}{1024}\}$

  - $g_0 = \{\frac{5}{1024}, 0, -\frac{63}{1024}, \frac{75}{1024}, \frac{495}{1024}, \frac{495}{1024}, \frac{75}{1024}, -\frac{63}{1024}, 0, \frac{5}{1024}\}$

    $g_1 = \{0, 0, -\frac{3\sqrt{15}}{512}, 0, \frac{22\sqrt{15}}{512}, -\frac{45\sqrt{15}}{512}, \frac{45\sqrt{15}}{512}, -\frac{22\sqrt{15}}{512}, 0, \frac{3\sqrt{15}}{512}\}$

    $g_2 = \{\frac{5}{1024}, 0, -\frac{117}{1024}, \frac{75}{1024}, \frac{315}{1024}, -\frac{315}{1024}, -\frac{75}{1024}, \frac{117}{1024}, 0, -\frac{5}{1024}\}$

- Tight Framelet Filter Bank 'tf3'

  - $h_0 = \{\frac{63}{5120}, 0, -\frac{-429}{5120}, 0, \frac{1309}{5120}, \frac{1617}{5120}, \frac{1617}{5120}, \frac{1309}{5120}, 0, -\frac{429}{5120}, 0, \frac{63}{5120}\}$

    $h_1 = \{0, 0, \frac{3\sqrt{231}}{2560}, 0, \frac{10\sqrt{231}}{2560}, 0, -\frac{77\sqrt{231}}{2560}, \frac{77\sqrt{231}}{2560}, 0, -\frac{10\sqrt{231}}{2560}, 0, -\frac{3\sqrt{231}}{2560}\}$

    $h_2 = \{\frac{63}{5120}, 0, -\frac{495}{5120}, 0, \frac{385}{5120}, \frac{1617}{5120}, -\frac{1617}{5120}, -\frac{385}{5120}, 0, -\frac{495}{5120}, 0, -\frac{63}{5120}\}$

  - $g_0 = \{\frac{63}{5120}, 0, -\frac{-429}{5120}, 0, \frac{1309}{5120}, \frac{1617}{5120}, \frac{1617}{5120}, \frac{1309}{5120}, 0, -\frac{429}{5120}, 0, \frac{63}{5120}\}$

    $g_1 = \{0, 0, \frac{3\sqrt{231}}{2560}, 0, \frac{10\sqrt{231}}{2560}, 0, -\frac{77\sqrt{231}}{2560}, \frac{77\sqrt{231}}{2560}, 0, -\frac{10\sqrt{231}}{2560}, 0, -\frac{3\sqrt{231}}{2560}\}$

    $g_2 = \{\frac{63}{5120}, 0, -\frac{495}{5120}, 0, \frac{385}{5120}, \frac{1617}{5120}, -\frac{1617}{5120}, -\frac{385}{5120}, 0, -\frac{495}{5120}, 0, -\frac{63}{5120}\}$

In table 3.1, 3.2 and 3.3, the columns refer to five filter banks respectively and the rows represent cases in different noise levels. In the experiment, several possible combination of different coefficients (e.g. Decomposition Levels, Threshold values...) were tried in order to obtain the near optimal performance of a method as much as possible. Moreover, both soft thresholding and hard thresholding schemes were implemented and the results can be easily compared in the cells. Finally, the following three tables present the nearly optimal PSNR values for different methods and various noise levels. The result of the method with the best performance is highlighted in bold font for each test set.

Algorithm 4 Obtaining the optimum denoising result

Input:
A choices set L for possible levels (e.g. L= [3:7]) A choices set TV for possible threshold values (e.g. TV = [50:25:300])
Output:
The optimum $PSNR_{opt}$ and the optimum Indices i_{opt}, j_{opt}
let $PSNR_{opt} = 0$
let $i_{opt}, j_{opt} = 0$
for i = 1: length(L) **do**
  $l = L(i)$
  for j = 1: length(TV) **do**
    $th = TV(j)$
    Using denoising algorithm to compute the PSNR with Decomposition levels = l and threshold value = th
    if $PSNR > PSNR_{opt}$ **then**
      $PSNR_{opt} = PSNR,\ i_{opt} = i,\ j_{opt} = j$
    end if
  end for
end for

Lena	db1		db2		tf1		tf2		tf3	
	hard	soft	hard	soft	hard	soft	hard	soft	hard	soft
$\sigma = 5$	**29.56**	26.61	33.01	19.94	29.07	25.59	30.04	26.38	28.69	25.35
$\sigma = 5$	29.43	26.61	**31.02**	19.81	29.04	25.58	29.98	26.35	28.67	25.34
$\sigma = 15$	28.91	26.65	28.76	19.63	**29.02**	25.58	29.83	26.32	28.66	25.34
$\sigma = 20$	27.04	26.65	26.06	19.36	28.92	25.56	**29.60**	26.25	28.49	25.31
$\sigma = 30$	26.00	25.98	21.67	18.69	27.80	25.51	**28.51**	26.12	26.69	25.25
$\sigma = 50$	23.85	23.45	17.51	16.68	25.54	24.60	**25.94**	25.30	24.74	24.11
$\sigma = 100$	18.43	19.98	11.60	12.47	21.78	20.59	**22.05**	21.22	21.80	20.50

Table 3.1: Lena Denoising

Boat	db1		db2		tf1		tf2		tf3	
	hard	soft	hard	soft	hard	soft	hard	soft	hard	soft
$\sigma = 5$	28.28	25.49	**30.07**	19.80	27.23	24.09	27.98	24.81	26.83	23.88
$\sigma = 10$	28.12	25.49	**28.88**	19.70	27.23	24.09	27.91	24.79	26.83	23.87
$\sigma = 15$	27.68	25.52	27.33	19.54	27.20	24.08	**27.84**	24.78	26.78	23.87
$\sigma = 20$	26.18	25.52	25.26	19.31	27.16	24.09	**27.67**	24.76	26.69	23.87
$\sigma = 30$	24.81	25.02	22.09	18.67	26.29	24.03	**26.66**	24.65	25.42	23.83
$\sigma = 50$	22.65	22.58	17.65	16.64	23.90	23.37	**24.28**	23.86	23.29	22.97
$\sigma = 100$	17.99	19.57	11.71	12.34	**20.84**	19.76	21.09	20.53	**20.84**	19.63

Table 3.2: Boat Denoising

Barbara	db1		db2		tf1		tf2		tf3	
	hard	soft	hard	soft	hard	soft	hard	soft	hard	soft
$\sigma = 5$	26.62	24.06	**29.65**	19.50	25.32	22.96	26.65	23.67	25.48	22.98
$\sigma = 10$	26.48	24.10	**28.41**	19.40	25.35	22.97	26.65	23.66	25.50	22.99
$\sigma = 15$	26.10	24.14	**26.82**	19.25	25.41	22.98	26.58	23.66	25.50	23.00
$\sigma = 20$	24.92	24.17	24.84	19.04	25.41	23.00	**26.48**	23.66	25.46	23.02
$\sigma = 30$	23.24	23.92	20.94	18.46	24.98	23.05	**25.81**	23.65	24.52	23.05
$\sigma = 50$	21.72	21.76	17.14	16.61	22.87	22.72	**23.27**	23.22	22.59	22.55
$\sigma = 100$	17.68	19.14	11.33	12.56	20.29	19.91	**20.58**	20.18	20.39	19.89

Table 3.3: Barbara Denoising

3.1.2 Performance analysis

In terms of thresholding schemes, the choice of hard thresholding gives a better performance evaluated by PSNR. In contrast, soft thresholding generates a poorer result in almost every test case. As the optimum data highlighted in bold illustrates, the filter banks 'db2' and 'tf1' have demonstrated a great advantage in image denoising over the other filter banks. To be specific, the data indicates

that the tight framelet filter banks are preferable in the circumstances with higher noise levels and 'db2' is preferable in the cases with lower noise levels. Figure 3.1 shows the difference between methods using orthogonal filter bank 'db2' and tight framelet filter bank 'tf2' in the performance of denoising with $\sigma = 30$. Moreover, the second column gives the zoom in graphs in order to show the details.

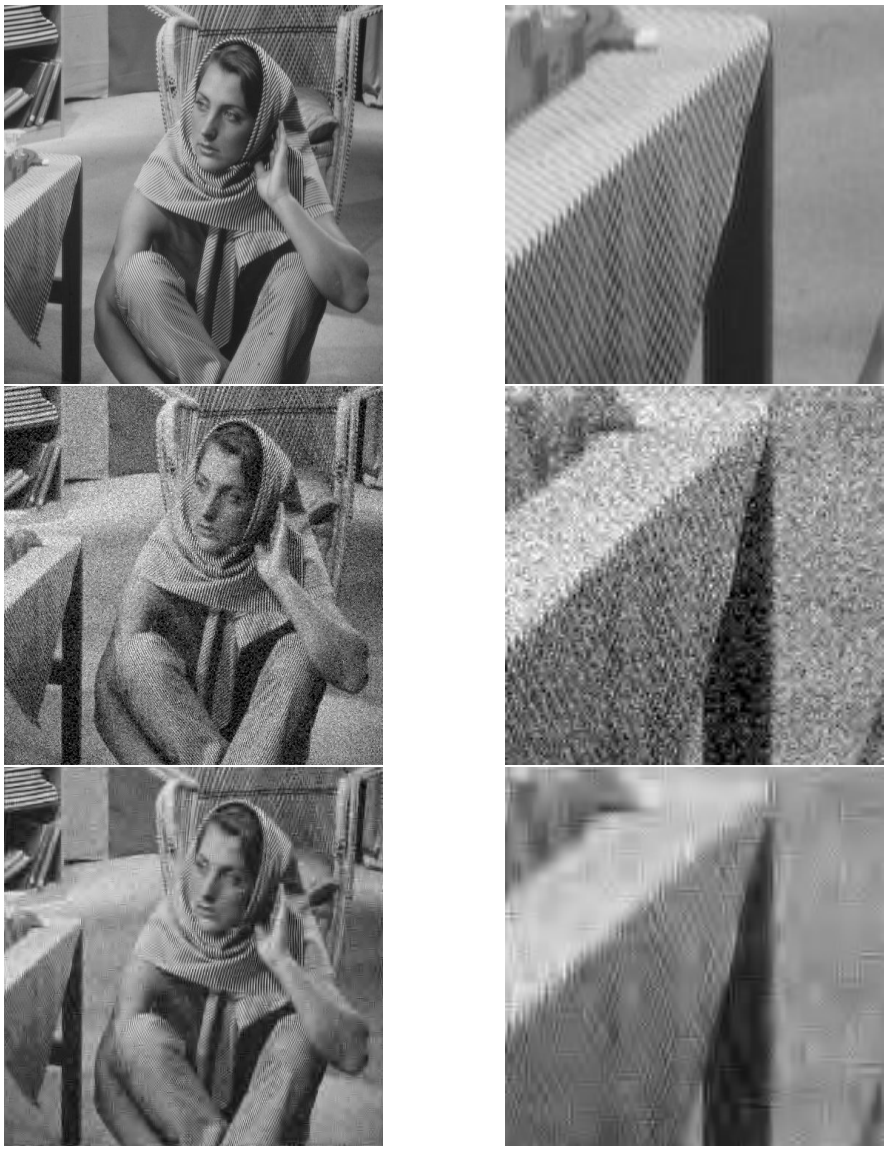

FIGURE 3.1: Denoising Result with $\sigma = 30$ for Barbara. Top: Original images and Zoom. Middle: 'db2' Denoising. Bottom: 'tf2' denoising.

3.2 Image Inpainting

3.2.1 PSNR results in image inpainting

Inpainting process is more complicated than the denoising process in the way that recursive implementation is required for an effective inpainting process. Similar to the way we assessing the variables in denoising, several simple masks containing basic geometric graphs are selected and five filter banks are chosen in order to construct a comparision scheme. To further justify the choice of soft thresholding over hard thresholding in the process of inpainting, each test case is implemented for both thresholding methods. Similarly, the decomposition levels, initial threshold values and recursion times are selected form a range of feasible numbers for many times in order to reach the optimal result in extensive simulations.

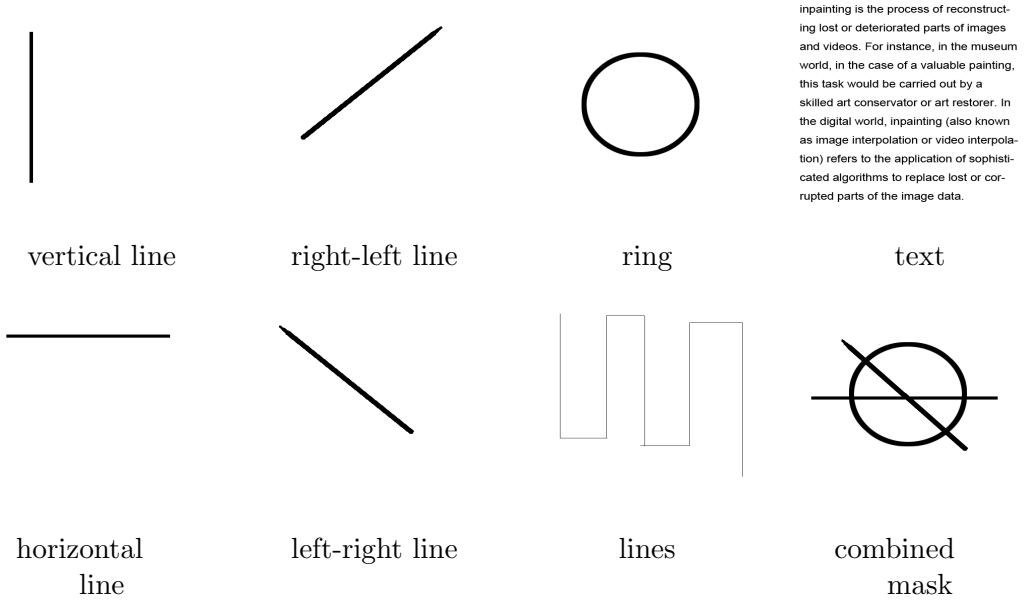

FIGURE 3.2: Mask Figures

Figure 3.2 shows eight different masks used in our inpainting experiment including straight lines with four possible directions, a ring mask, text mask and a complicated mask combined of others.

In the table 3.4, 3.5 and 3.6, the columns refer to fiver different filter banks, 'db','db2','tf1','tf2' and 'tf3', the rows represent cases with different masks. And the second column named "mask PSNR" represent the corruption level of the mask image before the implementation of inpainting, surrounding values in the same row should be compared with the "mask PSNR" in order to justify the effectiveness of the corresponding method on the mask.

Algorithm 5 Obtaining the optimum inpainting result

Input:
A choices set L for possible levels (e.g. L= [3:7]) A choices set TV for possible threshold values (e.g. TV = [50:25:300]) A choices set R for possible recursion times (e.g. R =[3,5,7,9])

Output:
The optimum $PSNR_{opt}$ and the optimum Indice i_{opt}, j_{opt}
let $PSNR_{opt} = 0$
let $i_{opt}, j_{opt} = 0, k_{opt} = 0$
for i = 1: length(L) **do**
  $l = L(i)$
  for j = 1: length(TV) **do**
    $th = TV(j)$
    for k = 1: length(R) **do**
      $rt = R(k)$
      Using inpainting algorithm to compute the PSNR with Decomposition levels = l, threshold value = th, recursion time = rt
      if $PSNR > PSNR_{opt}$ **then**
        $PSNR_{opt} = PSNR,\ i_{opt} = i,\ j_{opt} = j, k_{opt} = k$
      end if
    end for
  end for
end for

Lena	mask	db1		db2		tf1		tf2		tf3	
	PSNR	hard	soft	hard	soft	hard	soft	hard	soft	hard	soft
vertical line	26.88	27.21	33.70	26.77	26.75	28.96	36.58	29.31	36.05	29.37	**36.62**
horizontal line	24.39	24.65	32.43	25.03	24.98	25.81	37.00	25.91	39.16	26.51	**40.66**
right-left line	23.01	25.75	32.03	23.32	23.29	**34.13**	33.72	28.89	33.94	33.35	33.58
left-right line	24.62	27.92	32.99	24.98	24.87	38.41	37.82	33.03	37.37	**39.51**	38.41
ring	21.44	23.57	28.78	21.64	21.64	29.87	33.52	25.69	31.39	29.21	**33.72**
lines	27.18	28.51	43.32	33.69	32.64	32.74	43.36	37.23	**44.86**	34.79	43.67
text	15.75	26.52	30.95	16.08	16.13	33.75	33.13	33.71	**34.02**	33.87	33.55
combine	19.23	21.36	26.65	19.43	19.42	27.22	**31.39**	24.16	30.50	27.00	31.25

TABLE 3.4: Lena Inpainting

Boat	mask	db1		db2		tf1		tf2		tf3	
	PSNR	hard	soft	hard	soft	hard	soft	hard	soft	hard	soft
vertical line	25.76	26.16	34.27	25.76	25.64	28.50	40.99	28.10	41.23	29.28	**41.28**
horizontal line	24.96	25.22	33.91	25.60	25.55	26.44	39.68	26.35	**41.08**	27.24	41.04
right-left line	23.39	26.26	31.67	23.66	23.63	**34.96**	34.39	29.86	33.39	34.74	34.09
left-right line	23.39	26.29	33.45	23.73	23.67	**38.43**	37.86	30.15	36.98	36.84	38.18
ring	22.30	24.51	29.43	22.55	22.54	31.43	33.86	27.09	32.54	30.49	**34.16**
lines	27.28	28.73	42.03	32.67	32.15	33.87	41.59	37.06	43.01	35.75	**41.73**
text	15.39	26.73	30.39	15.72	15.73	**32.53**	32.08	32.00	32.51	31.68	31.92
combine	18.96	20.81	24.85	19.14	19.13	24.80	**29.38**	22.79	28.50	24.90	29.26

TABLE 3.5: Boat Inpainting

Barbara	mask	db1		db2		tf1		tf2		tf3	
	PSNR	hard	soft	hard	soft	hard	soft	hard	soft	hard	soft
vertical line	25.14	25.34	32.71	25.12	25.02	26.80	**45.54**	26.44	42.06	27.06	43.35
horizontal line	25.49	25.79	31.95	26.13	26.07	26.76	36.31	27.31	36.39	27.53	**37.81**
right-left line	24.37	27.62	33.38	24.63	24.60	**37.67**	37.30	32.55	36.54	37.44	37.06
left-right line	22.92	25.66	32.37	23.22	23.17	36.19	35.97	29.52	35.79	**36.10**	35.92
ring	20.85	23.05	28.11	21.09	21.08	29.79	33.39	25.34	32.07	28.84	**33.70**
lines	27.53	29.18	40.95	31.12	31.92	34.52	40.84	39.67	**43.05**	36.72	41.32
text	16.31	26.98	30.03	16.54	16.59	**32.30**	31.72	31.69	32.17	32.49	32.01
combine	18.02	19.91	24.72	18.21	18.19	24.78	30.36	22.43	28.71	24.72	30.19

TABLE 3.6: Barbara Inpainting

3.2.2 Performance analysis

In terms of thresholding schemes, the soft thresholding method shows a slight advantage in the process of image inpainting, whereas the hard thresholding also gets credit in some circumstances. As the optimal data highlighted in bold illustrates, the family of tight framelet filter banks demonstrated a great advantage in each test case. Figure 3.3 shows the comparison of inpainting results with text mask

using orthogonal filter bank Haar and the tight framelet filter bank 'tf1' correspondingly, and the second column shows the zoom in details. As we see, tight framelet filter bank is an effective tool since it has the property of redundancy and Figure 3.3 also indicates redundancy is desirable in some applications such as image inpainting.

FIGURE 3.3: Inpainting Result with 'text' mask for Boat. Top: Original images and Zoom. Middle: 'db1' Inpainting. Bottom: 'tf1' inpainting.

Section 4 Conclusion

In the applications of wavelet transform such as image denoising and image inpainting, careful choice of a suitable filter bank as well as coefficients such as threshold values and decomposition values can be very essential for obtaining a desirable result. Theoretically, frames are an over-complete version of a basis set and the tight frames are regarded as an over-complete version of an orthogonal basis set.

In practical, the algorithms and computing methods used for different filter banks are basically the same, while the implementation with the tight framelet method can result in a certain amount of redundancy. However, this redundancy property of a tight framelet is desirable in the application of image inpainting since it gives a robustness in the transform and the errors consequently become less destructive. In the context of image denoising, the redundancy property seems not so effective as it behaves in the inpainting. Moreover, tight framelet filter banks are proved to be undesirable in denoising especially when the noise level is low (As shown in table 3.1).

Bibliography

[1] B. Han and X. Zhuang, Algorithms for matrix extension and orthogonal wavelet filter banks over algebraic number fields, Mathematics of Computation. 82(281): 459-490, 2013

[2] B. Han and X. Zhuang, Matrix extension with symmetry and its applications to symmetric orthonormal multiwavelets. SIAM Journal on Mathematical Analysis. 42(5): 2297-2317, 2010

[3] B. Han, Properties of discrete framelet transforms, Mathematical Modelling of Natural Phenomena, 8, 18-47, 2013

[4] C. Sidney Burrus, Ramesh A. Gopinath and Haitao Guo, Introduction to Wavelets and Wavelet Transforms, Prentice Hall, 1998.

[5] C. K. Chui, An introduction to wavelets. Academic Press, Inc., Boston, MA, 1992.

[6] D. L. Donoho, Denoising by soft-thresholding, IEEE Transactions on Information Theory. Theory, 41, pp. 613–627, May 1995.

[7] F. Abramovich, T. Sapatinas, and B. W. Silverman, Wavelet thresh- olding via a Bayesian approach J. R. Statist. Soc., 60, pp. 725–749, 1998.

[8] G. Strang and T. Nguyen, Wavelets and filter banks, Wellesley College, 2nd edition, 1996.

[9] I. Daubechies. Different Perspectives on Wavelets. American Mathematical Society, 47, January 11-12,1993.

[10] I. Daubechies. Ten lectures on wavelets. CBMS-NSF Regional Conference Series in Applied Mathematics, 61, December 1992.

[11] J. S. Walker and Ying-Jui Chen. Image denoising using tree-based wavelet subband correlations and shrinkage. Opt. Eng. 39, 2000.

[12] M. Steinbuch and M.J.G. van de Molengraft. Wavelet theory and applications: A literature study. Technical report, Eindhoven University of Technology, 2005.

[13] Q. Mo and X. Zhuang, Matrix splitting with symmetry and dyadic framelet filter banks over algebraic number fields, Linear Algebra and Its Applications, 437(10): 2650-2679, May 2012.

[14] S. Mallat, A wavelet tour of signal processing. Third edition. Elsevier/Academic Press, Amsterdam, 2009.

[15] S. Tsai. Wavelet transform and denoising. Master's thesis, Chapter 4, pp. 35-42, 2002.

[16] S. Grace Chang, Bin Yu, and Martin Vetterli, Adaptive Wavelet Thresholding for Image Denoising and Compression, IEEE Transactions On Image Processing, 9(9), September 2000.

Appendix A Some Important Matlab codes

Funtion *Subdivision.m*

```
%Subdivision operator plays the row of refining and predictiong the data to
    higher resolution levels
%Input: Filter is the related filter bank cell, W is the original input signal
%Output: V is a cell, with rows v0,v1,v2... each row is one result of the
%        transition Process using the ith element of Filter Bank
function V=Subdivision(Filter,W,shift)
    V=cell(1,length(Filter));
    for i=1:length(Filter)
        v=cell2mat(W(i,1));
        %each v is the result of the procedure of upsample and convolution, sqrt
    (2) and 2 can be replaced by other numbers instead.
        v_up=upsample(v,2);
        v=fconv(v_up,sqrt(2)*cell2mat(Filter(1,i)));
        v=circshift(v,shift);
        V(1,i)={v};
    end
end
```

Funtion *Transition.m*

```
%Transition operator plays the role of coarsening and frequency seperating the
    data to lower resolution levels
%Input: Filter is the related filter bank cell, V is the original input signal
%Output: W is a cell, with rows w0,w1,w2... each row is one result of the
%        transition Process using the ith element of Filter Bank
function W=Transition(Filter,V)
    W=cell(length(Filter),1);
    for i=1:length(Filter)
        %for each ui, we calculate its complex conjugate sequence reflected about
     the origin,u_star
        u_star=conj(fliplr(cell2mat(Filter(1,i))));
        %each w is the result of the procedure of convolution and downsample,
    sqrt(2) and 2 can be replaced by other numbers instead.
        w=fconv(V,sqrt(2)*u_star);
        w=downsample(w,2);
```

```
        %keep the vector w in each row of cell W
        W(i,1)={w};
    end
end
```

Funtion *Decomposition.m*

```
function [co_L,co_H_sets]=Decomposition(v,Filter,levels)
    s=length(Filter)-1; %number of High Pass filters
    co_L=v; %coefficent after Lower pass filter
    co_H_sets=cell(0);   %coefficients after high pass filters
    %Decomposition
    for i=1:levels
        w=Transition(Filter,co_L);
        if isempty(co_H_sets)  %First iteration
            co_H_sets=w(2:end)';
        else
            co_H_sets(1,s+1:end+s)=co_H_sets(1:end);
            co_H_sets(1,1:s)=w(2:end);
        end
        co_L=cell2mat(w(1));  % for next iteration
    end
end
```

Funtion *Reconstruction.m*

```
function y=Reconstruction(co_L,co_H_sets,Filter,shift)

    s=length(Filter)-1; %number of High Pass filters
    w=cell(s+1,1); %To store the values of signals
    %Iterating while co_H_sets is not empty
    while ~isempty(co_H_sets)
        w(1)={co_L};
        w(2:end)=co_H_sets(1:s);
        co_H_sets=co_H_sets(s+1:end); %eliminate the used s elements
        v=Subdivision(Filter,w,shift);
        co_L=sum(cell2mat(v),2);  %sum up horizonally
    end
    y=co_L;
end
```

Funtion *multiLevel_DFrT.m*

```
function [co_L,co_H_sets,dY]=multiLevel_DFrT(v,filtertype,levels)
    [FILTER_D,FILTER_R,SHIFT]=matchFilter(filtertype);
    %Final Result
    [co_L,co_H_sets]=Decomposition(v,FILTER_D,levels);
    dY=Reconstruction(co_L,co_H_sets,FILTER_R,SHIFT)-v;
end
```

Funtion *D2_dwt.m*

```
function y=D2_dwt(x,filtertype,levels);
    % x is the original 2D signal
    % Output: LL,HL,LH,HH are 4 coefficient matrices with the same size
    % LL:diag. detail coefficients  HL:diag. detail coefficients
```

```
    % LH:diag. detail coefficients  HH:diag. detail coefficients
    x0=x;
    %Define the filter type
    global FILTER_D;global FILTER_R;global SHIFT;
    [FILTER_D,FILTER_R,SHIFT]=matchFilter(filtertype);
    [row,col]   =  size(x);                      % Get the size of the original
    matrix
    % x_D will be the decomposed matrix of X, basically the length of x_D will be
    % (n-1)l0-(n-2)l0/(levels)^n, where n is the length of filter_D, l0 is the
    length of x.
    n = length(FILTER_D);
    len_x_D  = (n-1)*col-(n-2)*col/(2^levels);
    x_D=zeros(len_x_D);
    for j = 1:row                              % Implement one Dimensional DFRT to
     each row
        tmp1 = x(j,:)';
        [ca1,cd1] = Decomposition(tmp1,FILTER_D,levels);
        % size(ca1)
        tmp = [ca1; cell2mat(cd1')]';          % Restore the data into x, [L|H]

        x_D(j,:) = tmp(1:len_x_D);
    end

    for k = 1:len_x_D                              % Implement one Dimensional
    DFRT for each column
        tmp2 = x_D(1:row,k);
        [ca2,cd2] = Decomposition(tmp2,FILTER_D,levels);
        tmp = [ca2;cell2mat(cd2')]';                  % Restore the data into x [
    LL,HL;LH,HH]
        x_D(:,k) = tmp(1:len_x_D);
    end
    y = x_D;
    % disp('y-x is');
    % D2_idwt(x)-x;
end
```

Funtion *D2_idwt.m*

```
function y=D2_idwt(x,levels);
    global FILTER_D;
    global FILTER_R;
    global SHIFT;
    % This function mainly reconstruct the signal
    % get the size of the original signal
    org_row = 2^(floor(log2(length(x))));
    org_col = 2^(floor(log2(length(x))));

    ca1_len  =  org_row/(2^levels);
    ca2_len  =  org_col/(2^levels);
    y=zeros(org_row,org_col);
    n = length(FILTER_D);
    [row,col]=size(x);

    for k = 1:col                      % Split the x to get the low pass
    coefficient and the high pass coefficient
        tmp = x(:,k);
        ca1 = vector_pop(tmp,ca1_len,'tmp');
        cd1 = cell(0);
```

```
        for m = 1:levels
            len1 = ca1_len*2^(m-1);
            for jt=1:n-1
                cd1(end+1) = {vector_pop(tmp,len1,'tmp')};
            end
        end
        tmp1 = Reconstruction(ca1,cd1,FILTER_D,SHIFT);   % Reconstruction
        yt(:,k) = tmp1;                 %  Now , yt = [L|H]
    end
    for j = 1:org_row
        tmp = yt(j,:);
        ca2 = vector_pop(tmp,ca2_len,'tmp')';
        cd2 = cell(0);
        for m = 1:levels
            len1 = ca2_len*2^(m-1);
            for jt=1:n-1
                cd2(end+1) = {vector_pop(tmp,len1,'tmp')'};
            end
        end
        tmp2 = Reconstruction(ca2,cd2,FILTER_R,SHIFT);
        y(j,:) = tmp2;
    end
end
function y = vector_pop(x,k,name)
% x is the value of vector, name is the vector name is the caller workspace
    ind = [1:k];
    y = x(ind);
    x(ind) = [];
    assignin('caller',name,x)
end
```

Funtion *decreaseTV.m*

```
function y = decreaseTV(TV,schemetype)
    %Decrease the threshold value for next iteration according to variant methods
    y = TV;
    switch schemetype
        case {1,}
            if TV >= 50
                y  =  TV-20 ;
            end
        case {2,}
            if TV > 10
                y = TV/2;
            end
        otherwise,
            warndlg(['undefined scheme type:',schemetype]);
            y = nan; %not a number
    end
    fprintf('                        TV_old = %.0f, TV_new = %.0f, Jump to next
    iteration...\n',TV,y);
end
```

Funtion *matchFilter.m*

```
function [FILTER_D,FILTER_R,SHIFT]=matchFilter(filtertype)
    switch filtertype
        case 'db1'  % testing the second set of Filter Bank, biorthogonal wavelet
     filter bank
```

```
            LO_D = [1/2,1/2];
            HI_D = [-1/2,1/2];
            FILTER_D = {LO_D,HI_D};
            FILTER_R = {LO_D,HI_D};
            SHIFT = -1;
        case 'db2'  % testing the second set of Filter Bank, Daubechies 4-tap
    filter bank
            LO_D = [1-sqrt(3), -3+sqrt(3), 3+sqrt(3), -1-sqrt(3)]*1/8;
            HI_D = [1+sqrt(3), 3+sqrt(3), 3-sqrt(3), 1-sqrt(3)]*1/8;
                        FILTER_D = {LO_D,HI_D};
            FILTER_R = {LO_D,HI_D};
            SHIFT = -3;
        case 'tf1'  % testing the 3rd set of Filter Bank, tight framlet wavelet
    filter bank
            u0 = [1/4,1/2,1/4];
            u1 = [-sqrt(2)/4,0,sqrt(2)/4];
            u2 = [-1/4,1/2,-1/4];
            FILTER_D = {u0,u1,u2};
            FILTER_R = {u0,u1,u2};
            SHIFT = -2;
        case 'tf2' %Tight framlet wavelet filter bank, from Example1
            u0 = [5, 0,  -63, 75, 495, 495, 75, -63, 0, 5]*1/1024;
            u1 = [0, 0,   -3,  0,  22, -45, 45, -22, 0, 3]*sqrt(15)/512;
            u2 = [5, 0, -117, 75, 315,-315,-75, 117, 0,-5]*1/1024;
            FILTER_D = {u0,u1,u2};
            FILTER_R = {u0,u1,u2};
            SHIFT = -9;
        case 'tf3' %Tight framlet wavelet filter bank, from Example2
            u0 = [63, 0, -429, 0, 1309, 1617, 1617, 1309, 0, -429, 0, 63]*1/5120;
            u1 = [ 0, 0,    3, 0,   10,    0,  -77,   77, 0,  -10, 0, -3]*sqrt
    (231)/2560;
            u2 = [63, 0, -495, 0,  385, 1617,-1617, -385, 0,  495, 0,-63]*1/5120;
            FILTER_D = {u0,u1,u2};
            FILTER_R = {u0,u1,u2};
            SHIFT = -11;
        case 'tf4' %Tight framlet wavelet filter bank, from Example3
            u0 = [ 21, 539, -825, -3927,  6930, 30030, 30030, 6930, -3927, -825,
    539, 21]*1/65536;
            u1 = [-21,-539, 1023,  9009,-12474, -9702,  9702,12474, -9009,-1023,
    539, 21]*1/65536;
            u2 = [  3,  77, -108,  -308,   898,  -898,   308,  108,   -77,   -3,
      0,  0]*sqrt(231)/32768;
            FILTER_D = {u0,u1,u2};
            FILTER_R = {u0,u1,u2};
            SHIFT = -11;
        otherwise
            warndlg(['undefined filter type: ',filtertype]);
    end

end
```

Funtion *psnr.m*

```
function p = psnr(x,y, vmax)
    if nargin <=2
        m1 = max( abs(x(:)) );
        m2 = max( abs(y(:)) );
        vmax = max(m1,m2);
    end
```

```
    d = mean( (x(:)-y(:)).^2 );
    p = 10*log10( vmax^2/d );
end
```

Funtion *fconv.m*

```
function y=fconv(v,a)
    if length(v) < length(a)
        warndlg('signal length smaller than low pass filter length','!! Warning
    !!')
    end
    x=zeros(size(v));
    x(1:length(a))=a;
    y=ifft(fft(v).*fft(x));
end
```

Funtion *test_deno.m*

```
function test_deno(filtertype,levels)
    pic=imread('pics/Lena','png');
    %subplot(2,2,1), imshow(uint8(pic)),title('original iamge');
    %Denoising Process
    sig=50;
    V=(sig/256)^2;
    npic=imnoise(pic,'gaussian',0,V);
    %subplot(2,2,2), imshow(uint8(npic)),title('noise');
    %Doing the wavelet decomposition
    C=D2_dwt(npic,filtertype,levels);
    figure(2), imshow(uint8(C));
    %Define the threshold(universal threshold)
    TV=50;
    %Hard thresholding
    hardC=[hthresh(C,TV)];
    %Reconstructing the image from the hard-thresholded wavelet coefficients
    newpich=D2_idwt(hardC,levels);
    %Displaying the hard-denoised image
    %subplot(2,2,3),imshow(uint8(newpich)),title('hard-denoised image');
    PSNR_H = psnr(double(pic),newpich,255)
    %Soft thresholding
    softC=[sthresh(C,TV)];
    %Reconstructing the image from the soft-thresholded wavelet coefficients
    newpics=D2_idwt(softC,levels);
    %Displaying the soft-denoised image
    %subplot(2,2,4), imshow(uint8(newpics)),title('soft-denoised image');
    PSNR_S = psnr(double(pic),newpics,255)
end

function op=sthresh(X,T);
%A function to perform soft thresholding on a
%given an input vector X with a given threshold T
% S=sthresh(X,T);
    ind=find(abs(X)<=T);
    ind1=find(abs(X)>T);
    X(ind)=0;
    X(ind1)=sign(X(ind1)).*(abs(X(ind1))-T);
    op=X;
end
```

```
function op=hthresh(X,T);
%A function to perform hard thresholding on a
%given an input vector X with a given threshold T
% H=hthresh(X,T);
    ind=find(abs(X)<=T);
    X(ind)=0;
    op=X;
end
function res=constructMask(mpic)
        res = mpic > 100;
        % size(res)
end
```

Funtion *test_inp.m*

```
function test_inp(filtertype,rt)
        %Note: Figure window 1 displays the original image, fig 2 the noisy img
        %fig 3 denoised img by hard thresholding, fig 4 denoised by soft
    thresholding
    %Reading the image
    pic=imread('pics/bagua','png');
    % size(pic)
    subplot(2,2,1), imshow(uint8(pic)),title('original iamge');
    %Inpainting Process
    mpic = imread('pics/masks/m_text','png');
    B=constructMask(mpic);
    %npic is the noise picture
    npic=double(pic).*B;
    subplot(2,2,2), imshow(uint8(npic)),title('missing Data');
    levels=5;
    %Define the threshold(universal threshold)
    TV=200;
    C=npic;
    for j=1:rt
        %Doing the wavelet decomposition
        C_dec=D2_dwt(C,filtertype,levels);
        %Hard thresholding
        hardC=[hthresh(C_dec,TV)];
        %Reconstructing the image from the hard-thresholded wavelet coefficients
        C_thr=D2_idwt(hardC,levels);
        %C_thr=waverec2(hardC,S,'db1');
        C_new = (1-B).*C_thr+B.*C;

        C = C_new;
        if TV>300
            TV = TV-20 ;
        end
    end
    %Displaying the hard-denoised image
    subplot(2,2,3),imshow(uint8(C)),title('hard-denoised image');
    PSNR = psnr(double(pic),C,255)

    C1=npic;
    for j=1:5
        %Doing the wavelet decomposition
        C_dec=D2_dwt(C1,filtertype,levels);
        %Soft thresholding
        softC=[sthresh(C_dec,TV)];
```

```
        %Reconstructing the image from the soft-thresholded wavelet coefficients
        C_thr=D2_idwt(softC,levels);
        %Update the missing information of C by
        C_new = (1-B).*C_thr+C1;

        C1 = C_new;
        TV = TV/2 ;
    end
    %Displaying the soft-denoised image
    subplot(2,2,4), imshow(uint8(C1)),title('soft-denoised image');
end

function op=sthresh(X,T)
%A function to perform soft thresholding on a
%given an input vector X with a given threshold T
% S=sthresh(X,T);
    ind=find(abs(X)<=T);
    ind1=find(abs(X)>T);
    X(ind)=0;
    X(ind1)=sign(X(ind1)).*(abs(X(ind1))-T);
    op=X;
end

function op=hthresh(X,T)
%A function to perform hard thresholding on a
%given an input vector X with a given threshold T
% H=hthresh(X,T);
    ind=find(abs(X)<=T);
    X(ind)=0;
    op=X;
end
function res=constructMask(mpic)
        res = mpic > 150;
        % size(res)
end
```

Funtion *test_inp_c.m*

```
function test_inp_c(filtertype,rt)
    %Note: Figure window 1 displays the original image, fig 2 the noisy img
    %fig 3 denoised img by hard thresholding, fig 4 denoised by soft thresholding
    levels = 5;
    %Define the threshold(universal threshold)
    TV = 200;
    %Reading the image
    pic = imread('pics/1','png');
    pic_R = pic(:,:,1);
    pic_G = pic(:,:,2);
    pic_B = pic(:,:,3);
    subplot(2,2,1), imshow(uint8(pic)),title('original iamge');
    %Inpainting Process
    %mpic is the mask picture
    mpic = imread('pics/masks/m_text','png');
    B = constructMask(mpic);
    C_RGB_H = zeros(size(pic));
    C_RGB_S = zeros(size(pic));
    B3 = zeros(size(pic)); for i = 1:3; B3(:,:,i) = B; end;
    n_pic = double(pic).*B3;
    PSNR1  =  psnr(double(n_pic),double(pic),255)
```

```
    subplot(2,2,2), imshow(uint8(n_pic)),title('Corrupted Image');
    for i = 1:3
        %npic is the polluted picture
        npic = double(pic(:,:,i)).*B;
        C_H = npic;
        C_S = npic;
        TV_TMP = TV;
        for j = 1:rt
            C_dec_H = D2_dwt(C_H,filtertype,levels);
            C_dec_S = D2_dwt(C_S,filtertype,levels);
            %Hard thresholding
            hardC = [hthresh(C_dec_H,TV_TMP)];
            C_thr_H = D2_idwt(hardC,levels);
            %Soft thresholding
            softC = [sthresh(C_dec_S,TV_TMP)];
            C_thr_S = D2_idwt(softC,levels);
            %Update the missing information of C by
            C_new_H = (1-B).*C_thr_H + B.*C_H;
            C_H = C_new_H;
            C_new_S = (1-B).*C_thr_S + B.*C_S;
            C_S = C_new_S;
            TV_TMP = decreaseTV(TV_TMP,1);
        end
        C_RGB_H(:,:,i) = C_H;
        C_RGB_S(:,:,i) = C_S;
    end
    %Displaying the hard-denoised image
    subplot(2,2,3),imshow(uint8(C_RGB_H)),title('hard-denoised image');
    %assignin('base','C_RGB_H',C_RGB_H);
    PSNR  =  psnr(double(pic),C_RGB_H,255)
    subplot(2,2,4),imshow(uint8(C_RGB_S)),title('soft-denoised image');
    %assignin('base','C_RGB_S',C_RGB_S);
    PSNR  =  psnr(double(pic),C_RGB_S,255)
end
function op = sthresh(X,T)
    %A function to perform soft thresholding on a
    %given an input vector X with a given threshold T
    % S = sthresh(X,T);
    ind = find(abs(X) <= T);
    ind1 = find(abs(X) > T);
    X(ind) = 0;
    X(ind1) = sign(X(ind1)).*(abs(X(ind1))-T);
    op = X;
end
function op = hthresh(X,T)
    %A function to perform hard thresholding on a
    %given an input vector X with a given threshold T
    % H = hthresh(X,T);
    ind = find(abs(X) <= T);
    X(ind) = 0;
    op = X;
end
function res = constructMask(mpic)
        res  =  mpic > 150;
        % size(res)
end
```

Funtion *auto_test_deno.m*

```
function [PSNR_opt_H, PSNR_opt_S, PSNR_opt_H_index, PSNR_opt_S_index] =
    auto_test_deno()
    IMG_S = {'pics/Lena', 'pics/Boat', 'pics/Barbara'};
    SIGMA_S = [5,10,15,20,30,50,100];
    FILTERTYPE_S = {'db1','db2','tf1','tf2','tf3','tf4'};
    LEVELS_S = [3,5];
    TV_S = [50,100,200]; %Threshold values set
    %For test
    %IMG_S = {'pics/Lena','pics/Boat'};
    %SIGMA_S = [5,10];
    %FILTERTYPE_S = {'db1','db2'};
    %LEVELS_S = [1];
    %TV_S = [50]; %Threshold values set
    VAR_S = {IMG_S, SIGMA_S,FILTERTYPE_S, LEVELS_S, TV_S,};
    VARNAME_S = {'IMG_S','SIGMA_S','FILTERTYPE_S','LEVELS_S','TV_S'};
    %optiomal PSNR values
    PSNR_opt_H = zeros(length(SIGMA_S),length(FILTERTYPE_S),length(IMG_S));
    PSNR_opt_H_index = PSNR_opt_H;
    PSNR_opt_S = zeros(length(SIGMA_S),length(FILTERTYPE_S),length(IMG_S));
    PSNR_opt_S_index = PSNR_opt_S;
    c_time = clock;
    for i = 1:length(IMG_S)
        img_name = IMG_S(i);
        img_name = img_name{1};
        pic = imread(img_name,'png');
        fprintf('\n\nTESTING PICTURE %s ...\n\n',img_name);
        %pic=rgb2gray(pic);
        %subplot(2,2,1), imshow(uint8(pic)),title('original iamge');
        for j = 1:length(SIGMA_S)
            sig = SIGMA_S(j);
            V = (sig/256)^2;
            fprintf('testing denoising with sigma %d ...\n\n',sig);
            npic=imnoise(pic,'gaussian',0,V);
            %subplot(2,2,2), imshow(uint8(npic)),title('noise');
            for k = 1:length(FILTERTYPE_S)
                filtertype = FILTERTYPE_S(k);
                filtertype = filtertype{1};
                fprintf('   testing filtertype %s ...\n',filtertype);
                PSNR_H = [];PSNR_S = [];
                for levels = LEVELS_S
                    for TV = TV_S
                        fprintf('           using levels = %d and Threshold Value
     =%d ...\n',levels,TV);
                        C=D2_dwt(npic,filtertype,levels);
                        %Hard thresholding
                        hardC=[hthresh(C,TV)];
                        newpich=D2_idwt(hardC,levels);
                        %Displaying the hard-denoised image
                        %subplot(2,2,3),imshow(uint8(newpich)),title('hard-
    denoised image');
                        PSNR_H(end+1) = psnr(double(pic),newpich,255);
                        %Soft thresholding
                        softC=[sthresh(C,TV)];
                        newpics=D2_idwt(softC,levels);
                        %Displaying the soft-denoised image
                        %subplot(2,2,4), imshow(uint8(newpics)),title('soft-
    denoised image');
                        PSNR_S(end+1) = psnr(double(pic),newpics,255);
                    end
```

```
                end
                [maxv,index] = max(PSNR_H);
                PSNR_opt_H(j,k,i) = maxv; PSNR_opt_H_index(j,k,i) =index;
                [maxv,index] = max(PSNR_S);
                PSNR_opt_S(j,k,i) = maxv; PSNR_opt_S_index(j,k,i) =index;
            end
        end
    end
    printRes(IMG_S,c_time,VAR_S,VARNAME_S, PSNR_opt_H,PSNR_opt_S,PSNR_opt_H_index
    ,PSNR_opt_S_index);
end
function printRes(IMG_S,c_time,VAR_S,VARNAME_S, H,S,H_i,S_i)
    assignin('base','H',H);
    assignin('base','S',S);
    assignin('base','H_i',H_i);
    assignin('base','S_i',S_i);
    fprintf('Beginning To Write File...\n');
    fid = fopen('result_deno.tmp','a');
    %print parameter infomation
    fprintf(fid,'*************Parameter Info********');
    for i = 1:length(VAR_S)
        var = VAR_S{i};
        varname = VARNAME_S(i); varname = varname{1};
        vartype = class(var);
        fprintf(fid,'%15s = ',varname);
        if strcmp(vartype,'cell')
            fprintf(fid,'| %s ',var{:}); fprintf(fid,' |\n');
        elseif strcmp(vartype,'char')
            fprintf(fid,'| %s |\n',var);
        elseif strcmp(vartype,'double')
            fprintf(fid,'| %.0f ',var(:));fprintf(fid,' |\n');
        else
            disp(vartype)
        end
    end
    fprintf(fid,'*************Parameter Info End**************\n\n');
    fprintf(fid,'Started @ %s',datestr(datenum(c_time(1),c_time(2),c_time(3),
    c_time(4),c_time(5),c_time(6))));
    fprintf(fid,'\n');
    [row_sig, col_filter, h_pic] = size(H);
    %assignin('base','H',H);
    for k = 1:h_pic
        img_name = IMG_S(k); img_name = img_name{1};
        fprintf(fid,'-----Picture: %s Infomation-----\n',img_name);
        fprintf(fid,'Hard Table for Picture: %s \n',img_name);
        for i = 1:row_sig
            fprintf(fid,'%.2f ',H(i,:,k));
            fprintf(fid,'\n');
        end
        fprintf(fid,'Hard Index Table for Picture: %s \n',img_name);
        for i = 1:row_sig
            fprintf(fid,'%d ',H_i(i,:,k));
            fprintf(fid,'\n');
        end
        fprintf(fid,'Soft Table for Picture: %s \n',img_name);
        for i = 1:row_sig
            fprintf(fid,'%.2f ',S(i,:,k));
            fprintf(fid,'\n');
        end
        fprintf(fid,'Soft Index Table for Picture: %s \n',img_name);
```

```
        for i = 1:row_sig
            fprintf(fid,'%d ',S_i(i,:,k));
            fprintf(fid,'\n');
        end
    end
    c_time = clock;
    fprintf(fid,'------------------------\n Ended @ %s\n\n\n\n\n\n\n\n\n',datestr(
    datenum(c_time(1),c_time(2),c_time(3),c_time(4),c_time(5),c_time(6))));
    fclose(fid);
end
function op=sthresh(X,T);
%A function to perform soft thresholding on a
%given an input vector X with a given threshold T
% S=sthresh(X,T);
    ind=find(abs(X)<=T);
    ind1=find(abs(X)>T);
    X(ind)=0;
    X(ind1)=sign(X(ind1)).*(abs(X(ind1))-T);
    op=X;
end
function op=hthresh(X,T);
%A function to perform hard thresholding on a
%given an input vector X with a given threshold T
% H=hthresh(X,T);
    ind=find(abs(X)<=T);
    X(ind)=0;
    op=X;
end
function res=constructMask(mpic)
        res = mpic > 100;
        % size(res)
end
```

Funtion *auto_test_inp.m*

```
function [PSNR_opt_H, PSNR_opt_S, PSNR_opt_H_index, PSNR_opt_S_index] =
    auto_test_inp()
    IMG_S = {'pics/Lena', 'pics/Boat', 'pics/Barbara'};
    FILTERTYPE_S = {'db1','db2','tf1','tf2','tf3'};
    MASK_PREFIX = 'pics/masks/';
    MASK_S = { 'm_l_v','m_l_h', 'm_l_rl', 'm_l_lr','m_ring', 'm_line', 'm_text',
    'm_combine', };
    LEVELS_S = [5];
    TV_S = [50,100,200]; %Threshold values set
    SCHEME_S = [1]; %Scheme for decreasing TV
    RT_S = [5]; %Total Recursion Time set
    VAR_S = {IMG_S, FILTERTYPE_S, MASK_PREFIX, MASK_S, LEVELS_S, TV_S, SCHEME_S,
    RT_S};
    VARNAME_S = {'IMG_S','FILTERTYPE_S','MASK_PREFIX','MASK_S','LEVELS_S','TV_S',
    'SCHEME_S','RT_S'};
    %optiomal PSNR values
    % the last column stores the value of original PSNR
    PSNR_opt_H = zeros(length(MASK_S),length(FILTERTYPE_S)+1,length(IMG_S));
    PSNR_opt_H_index = PSNR_opt_H;
    PSNR_opt_S = zeros(length(MASK_S),length(FILTERTYPE_S)+1,length(IMG_S));
    PSNR_opt_S_index = PSNR_opt_S;
    c_time = clock;
    for i = 1:length(IMG_S)
        img_name = IMG_S(i);
```

```
    img_name = img_name{1};
    pic = imread(img_name,'png');
    fprintf('\n\nTESTING PICTURE %s ...\n\n',img_name);
    %pic=rgb2gray(pic);
    %subplot(2,2,1), imshow(uint8(pic)),title('original iamge');
    for j = 1:length(MASK_S)
        mask = MASK_S(j);
        mask = strcat(MASK_PREFIX,mask{1});
        fprintf('testing inpainting with mask %s ...\n\n',mask);
        mpic = imread(mask,'png');
        B=constructMask(mpic);
        %npic is the noise picture
        npic=double(pic).*B;
        %subplot(2,2,2), imshow(uint8(npic)),title('missing Data');
        PSNR_0 = psnr(double(pic),npic,255);
        PSNR_opt_H(j,end,i) = PSNR_0 ;
        PSNR_opt_S(j,end,i) = PSNR_0 ;
        for k = 1:length(FILTERTYPE_S)
            filtertype = FILTERTYPE_S(k);
            filtertype = filtertype{1};
            fprintf('   testing filtertype %s ...\n',filtertype);
            PSNR_H = [];PSNR_S = [];
            for levels = LEVELS_S
                for TV = TV_S
                    fprintf('           using levels = %d and Threshold Value
 =%d\n',levels,TV);
                    for rt = RT_S
                        fprintf('               Setting Recursion time = %d
...\n',rt);
                        for scheme = SCHEME_S
                            fprintf('                   Using %d th scheme
...\n',scheme);
                            TV_tmp =TV;
                            C_H = npic;
                            C_S = npic;
                            for tmp_int = 1:rt
                                %Doing the wavelet decomposition
                                C_dec_H = D2_dwt(C_H,filtertype,levels);
                                C_dec_S = D2_dwt(C_S,filtertype,levels);
                                %Hard thresholding
                                hardC = [hthresh(C_dec_H,TV_tmp)];
                                C_thr_H = D2_idwt(hardC,levels);
                                %Soft thresholding
                                softC = [sthresh(C_dec_S,TV_tmp)];
                                C_thr_S = D2_idwt(softC,levels);
                                %Update the missing information of C by
                                C_new_H = (1-B).*C_thr_H + B.*C_H;
                                C_H = C_new_H;
                                C_new_S = (1-B).*C_thr_S + B.*C_S;
                                C_S = C_new_S;
                                TV_tmp = decreaseTV(TV_tmp,scheme);
                            end
                            %Displaying the hard-denoised image
                            %subplot(2,2,3),imshow(uint8(C)),title('hard-
denoised image');
                            PSNR_H(end+1) = psnr(double(pic),C_H,255);
                            %Displaying the soft-denoised image
                            %subplot(2,2,4), imshow(uint8(newpics)),title('
soft-denoised image');
                            PSNR_S(end+1) = psnr(double(pic),C_S,255);
```

```
                        end
                    end
                end
            end
            [maxv,index] = max(PSNR_H);
            PSNR_opt_H(j,k,i) = maxv; PSNR_opt_H_index(j,k,i) =index;
            [maxv,index] = max(PSNR_S);
            PSNR_opt_S(j,k,i) = maxv; PSNR_opt_S_index(j,k,i) =index;
        end
    end
end
printRes(IMG_S,c_time,VAR_S,VARNAME_S, PSNR_opt_H,PSNR_opt_S,PSNR_opt_H_index
,PSNR_opt_S_index);
end
function printRes(IMG_S,c_time,VAR_S,VARNAME_S, H,S,H_i,S_i)
    assignin('base','H',H);
    assignin('base','S',S);
    assignin('base','H_i',H_i);
    assignin('base','S_i',S_i);
    fprintf('\nBeginning To Write File...\n');
    fid = fopen('result_inp.tmp','a');
    %print parameter infomation
    fprintf(fid,'*************Parameter Info********\n');
    for i = 1:length(VAR_S)
        var = VAR_S{i};
        varname = VARNAME_S(i); varname = varname{1};
        vartype = class(var);
        fprintf(fid,'%15s = ',varname);
        if strcmp(vartype,'cell')
            fprintf(fid,'| %s ',var{:}); fprintf(fid,' |\n');
        elseif strcmp(vartype,'char')
            fprintf(fid,'| %s |\n',var);
        elseif strcmp(vartype,'double')
            fprintf(fid,'| %.0f ',var(:));fprintf(fid,' |\n');
        else
            disp(vartype)
        end
    end
    fprintf(fid,'*************Parameter Info End*************\n\n');
    fprintf(fid,'Started @ %s \n',datestr(datenum(c_time(1),c_time(2),c_time(3),
    c_time(4),c_time(5),c_time(6))));
    [row_sig, col_filter, h_pic] = size(H);
    for k = 1:h_pic
        img_name = IMG_S(k); img_name = img_name{1};
        fprintf(fid,'-----Picture: %s Infomation-----\n',img_name);
        fprintf(fid,'Hard Table for Picture: %s \n',img_name);
        for i = 1:row_sig
            fprintf(fid,'%.2f ',H(i,:,k));
            fprintf(fid,'\n');
        end
        fprintf(fid,'Hard Index Table for Picture: %s \n',img_name);
        for i = 1:row_sig
            fprintf(fid,'%d ',H_i(i,:,k));
            fprintf(fid,'\n');
        end
        fprintf(fid,'Soft Table for Picture: %s \n',img_name);
        for i = 1:row_sig
            fprintf(fid,'%.2f ',S(i,:,k));
            fprintf(fid,'\n');
        end
```

```
        fprintf(fid,'Soft Index Table for Picture: %s \n',img_name);
        for i = 1:row_sig
            fprintf(fid,'%d ',S_i(i,:,k));
            fprintf(fid,'\n');
        end
    end
    c_time = clock;
    fprintf(fid,'------------------------\n Ended @ %s\n\n\n\n\n\n\n\n\n',datestr(
    datenum(c_time(1),c_time(2),c_time(3),c_time(4),c_time(5),c_time(6))));
    fclose(fid);
end
function op=sthresh(X,T)
%A function to perform soft thresholding on a
%given an input vector X with a given threshold T
% S=sthresh(X,T);
    ind=find(abs(X)<=T);
    ind1=find(abs(X)>T);
    X(ind)=0;
    X(ind1)=sign(X(ind1)).*(abs(X(ind1))-T);
    op=X;
end
function op=hthresh(X,T)
%A function to perform hard thresholding on a
%given an input vector X with a given threshold T
% H=hthresh(X,T);
    ind=find(abs(X)<=T);
    X(ind)=0;
    op=X;
end
function res=constructMask(mpic)
        res = mpic > 150;
        % size(res)
end
```