MEDICAL IMAGE REGISTRATION USING ARTIFICIAL NEURAL NETWORK

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Electrical Engineering

by

Hyunjong Choi

December 2015

© 2015

Hyunjong Choi

ALL RIGHTS RESERVED

COMMITTEE MEMBERSHIP

TITLE:	Medical Image Registration using Artificial Neural Network
AUTHOR:	Hyunjong Choi
DATE SUBMITTED:	December 2015
COMMITTEE CHAIR:	Xiao-Hua (Helen) Yu, Ph.D.
	Professor of Electrical Engineering
COMMITTEE MEMBER:	Jane Zhang, Ph.D.
	Professor of Electrical Engineering
COMMITTEE MEMBER:	Wayne Pilkington, Ph.D.
	Associate Professor of Electrical Engineering

ABSTRACT

Medical Image Registration using Artificial Neural Network

Hyunjong Choi

Image registration is the transformation of different sets of images into one coordinate system in order to align and overlay multiple images. Image registration is used in many fields such as medical imaging, remote sensing, and computer vision. It is very important in medical research, where multiple images are acquired from different sensors at various points in time. This allows doctors to monitor the effects of treatments on patients in a certain region of interest over time. In this thesis, artificial neural networks with curvelet keypoints are used to estimate the parameters of registration. Simulations show that the curvelet keypoints provide more accurate results than using the Discrete Cosine Transform (DCT) coefficients and Scale Invariant Feature Transform (SIFT) keypoints on rotation and scale parameter estimation.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1. Introduction

Image registration is the process of transforming a multiple image set of the same subject that was taken from different points of view, times, depths, or sensors into one coordinate system [1]. Image registration has its applications in various fields such as remote sensing, environmental monitoring, change detection, weather forecasting, integrating information into geographic information system (GIS), and in medical fields. Image registration is a spatial transform and it can be divided into three groups based on how the reference and the sensed image could be taken: [1]

1. The images were taken at different points of time.
2. The images were taken using different devices like MRI, CT, PET, SPECT etc. (multi modal).
3. The images were taken from different angles in order to have 2D or 3D perspective (multi temporal).

In medical fields, image registration is one of the most important techniques combining data from different modalities, e.g. Computer Tomography (CT) and Magnetic Resonance Imaging (MRI), to obtain complete information about the patient, and monitoring tumor or cancer growth. This allows doctors to monitor the effects of treatments on patients in a certain region of interest over time.

There exist many methods to carry out the registration depending on the problem [2] [3]. However, most of the algorithms basically follow the same approach as Figure 1.1 [4].

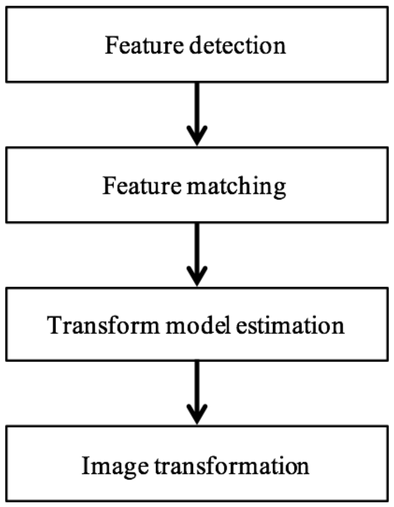

Figure 1.1 Basic approach of Image registration

At the feature detection step, salient and distinctive features such as edges, contours, closed-boundary regions are detected in both reference and sensed images. A comparison between the features in the reference and sensed images is established in feature matching. Then, mapping functions are estimated that aligning the sensed image with the reference image. At an image transformation stage, the sensed image is transformed based on the mapping function.

This thesis consists of five chapters. In chapter 2, we review the previous work of image registration methods and compare their advantage and drawbacks. Then, we define the problem and the range of research based on the neural network approach. Fundamental algorithms and the suggested approach using curvelet transform is explained in chapter 3. Simulation results are shown in chapter 4. According to the results in chapter 4, we conclude our research and discuss potential future work.

Chapter 2. Literature review

There exist many methods to perform image registration. These methods can be categorized with respect to various criteria including the application fields, dimensionality of data, computational cost, and the basis of the registration algorithm. Essentially, registration methods can be categorized into two types, depending on whether they use feature extraction or not [1]. They are area-based and feature-based methods. In this chapter, we're going to review previous literature in those two categories.

2.1. Area-based approaches

Area-based methods, also called correlation methods, deal with the images without detecting significant objects. They use correlation, mutual information (MI), cross-correlation, or Fourier analysis method to measure the similarity of images [5] [6]. The area can be the entire image or part of image with a predefined rectangular window, which is most often used, to estimate the correspondence. The rectangular window works well for registration when only translation differences exist between images. If images are deformed by more complex transformations, this type of windows is not able to work properly. Another limitation of the area-based methods is 'remarkableness' of window content. If a window only contains a smooth area without any prominent details, it may be matched incorrectly with the smooth, but totally unrelated another image [1]. Fourier methods exploit the representation of the images in the frequency domain. They are preferred rather than the spatial correlation methods if images are corrupted by frequency-dependent noise or acquired under time varying illumination. However, the computational cost and time expense grows greatly as the image size increases or the condition of transformation is more severe.

2.1.1. Correlation methods

Cross-correlation (CC) method exploits image intensities directly, without any structural analysis. So, they are sensitive to image intensity changes, noise, varying illumination, and sensor types. The representative algorithm of area-based methods is to use normalized cross-correlation (CC): [7]

$$CC(u,v)=\frac{\sum_x\sum_y T(x,y)I(x-u,y-v)}{\sqrt{\sum_x\sum_y I^2(x-u,y-v)}} \tag{2.1}$$

This function is computed for window pairs from the reference and the sensed images, and is searched for its maximum value. The parts of the two images producing the maximum cross-correlation value are considered the corresponding ones. This method works successfully when rotation and scaling exist in small degrees. The main drawbacks of this method are the flatness due to self-similarity of the images and high computational cost if the transformation is complex. Some modification methods are suggested in order to overcome its limitations. Pratt applied image filtering to improve CC performance on noisy or highly correlated images [8], and Van Wie and Anuta exploited correlation on the edge extraction instead of using the entire original images [9] [10] . The advantage of these modification methods is that the results are sensitive to intensity variation and noisy images. A sequential similarity detection algorithm (SSDA) is also suggested which is similar to the CC methods. It uses the sums of the differences in the intensity values for the window pairs and if the difference is too high, the pair is rejected as a non-match. The advantage of this method is that computation of SSDA is simpler than the CC. As mentioned above, even though there exist some limitations on correlation-like methods; they are still one of the most useful registration methods due to being easy to implement in hardware systems [1].

2.1.2. Fourier methods

Fourier methods are preferred over the correlation-like methods when the images are acquired under severe conditions and include noise [1]. It employed Fourier representation of the images. The representative algorithm in this approach is to use phase correlation [11]. It calculates the cross-power spectrum of the reference and sensed images and finds the peak point in its inverse domain. The method shows strong robustness against the correlated and frequency dependent noise and non-uniform, time varying illumination disturbances. However, this method, like the previous area methods, is just strong when there is only translation present. De Castro and Morandi introduced an extension of the phase correlation for additional rotation transform [12]. If the change of image scale is present, the images can be registered using the combination of polar-log mapping of the spectral magnitude and the phase correlation.

2.1.3. Mutual information (MI) methods

Mutual information (MI) technique is a measure of the dependence between two random variables and a very useful algorithm in multimodal registration. MI between two random variables X and Y is given by: [1]

$$MI(x,y) = H(Y) - H(Y \mid X) = H(x) + H(y) - H(x,y) H(x) = -E(\log_2(P(x))) \qquad (2.2)$$

Where $H(x) = -E(\log_2(P(x))$ represents entropy of random variable $P(x)$ and is the probability distribution of X .

Especially, MI methods have been widely used in medical imaging applications and have been proved to be very effective. However, they are not so successful in other fields such as military target recognition system. First, it takes a long time to search the global transformation parameters by optimizing MI similarity measure. So, it could not be used for real time systems.

In addition, it could fall into a local extreme not correlated in the global measurement. Several methods are proposed to solve this issue, which usually combined the MI with other feature-based methods to get higher robustness [4].

2.2. Feature-based approaches

Feature-based matching methods are typically used when the local structural information is more significant than the image's intensities [1] [13] [14]. They allow registering completely different images and can handle complex between-image distortions. The common drawback of the feature-based methods is that the respective features might be hard to detect or unstable in time. The crucial point of feature-based matching methods is that prominent features are invariant to transformations. So, the important task is to find the pair-wise correspondence between the reference and sensed images using their spatial relations or descriptors of features. Also, the descriptors should have met the conditions such as *invariance*, *uniqueness*, *stability*, and *independence*. Figure 2.3 shows an example of feature-based methods with two satellite images [1]. The descriptors (control points) in this example are invariant to the shift and scale parameters so they could be used effectively for registration.

2.2.1. Neural network methods

An artificial neural network (ANN) is a computational model based on biological human brain. The ANN is an adaptive system that can change its internal structure using the external information that flows through the network during learning phase. They can be used to map out complex relationships between inputs and outputs or pattern recognition in computer vision. The architecture of ANN adapts a parallel computational structure that is composed of a number of neurons connected through a set of links, which have some weight associated with them. Neurons consist of a number of inputs, summing junction, and an activation function to pass information to another neuron [15].

The structure of feedforward neural network is shown at Figure 2.1. The number of hidden layers depends on the complexity of the problem. Generally, if the number of hidden layers is increasing, it is able to handle more complex situation. A weight is assigned to each input and the results of the input times the weight are summed up. Then, the sum is passed through an activation function as shown at Figure 2.2. The most commonly used activation function is a sigmoid function and in this thesis a hyper-tangent function is used as an activation function.

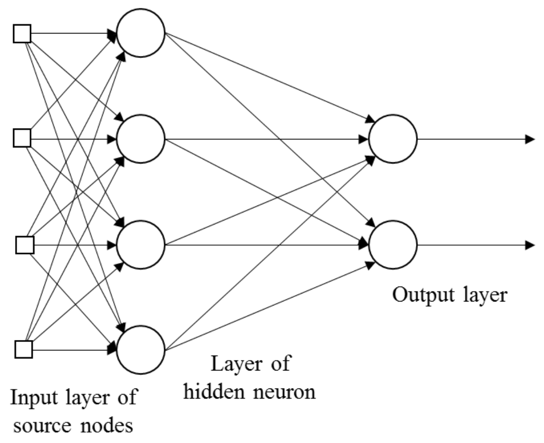

Figure 2.1 Artificial neural network architecture

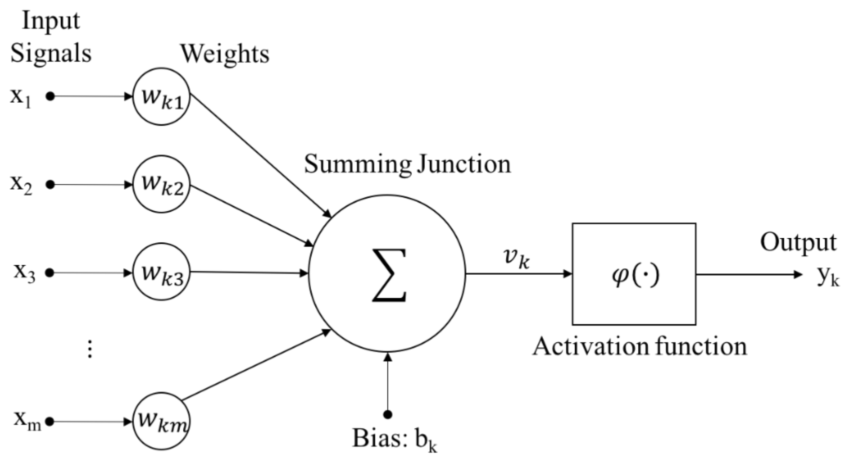

Figure 2.2 The structure of a neuron

There have been researches using neural network for image registration. In 2000, Sheinfeld, Tal, and Trisoh exploited Discrete Cosine Transform (DCT) coefficients as inputs to train the network [16]. The majority information of DCT coefficients is stored in the top left corner, so they used a rectangular window to select coefficients. This method gives a good result when there exist a small amount of rotation, scaling, and translation and shows robustness to Gaussian and salt and pepper noise. However, it is vulnerable to scaling factor. In 2006, Abche, Maalouf, Karam used Fourier coefficients instead of using DCT [17]. They used an 8 by 8 window to select coefficients around zero frequency. This method performed far more accurate than DCT coefficients approach with respect to the scaling factor. Jianzhen Wu and Jianying Xie trained their networks using low order Zernike moments [18]. Zernike moments are a sequence of polynomials that form an orthogonal basis on the unit disc. They are the projection of the image onto the orthogonal basis functions and have been used greatly in pattern recognition. An advantage using Zernike moments is that the absolute values of the moments are rotation invariant, robust to noise, and fast to compute. The previous approaches used general feedforward neural network. Pramod used the wavelet neural network which employed a wavelet function as an activation function [19]. Wavelet function can be trained more adaptively at each neuron node with its parameters, that are scale and translation parameters, comparing to hyper-tangent function which is the same at all neurons. Wavelet neural network method performed more accurate and faster than general feedforward neural network due to its fewer neurons and a single hidden layer.

This thesis exploits artificial neural network approaches for image registration. The advantage of using neural network is that it requires less time to estimate parameters. Neural network is able to output all transformation parameters at once while conventional methods only estimate parameters one at a time. So, once the network is trained it is a simple work of passing an input vector to the network. Another advantage of neural network is that it is a very useful method to map non-linear relationships between inputs and outputs. This is because neural network uses a non-linear activation function for each neuron's output. This enables the network to cover the severe cases when there exist high distortion of the images.

2.3. Summary

Area-based methods are preferably applied when the images do not have many prominent details and the distinctive information is provided by gray levels/colors rather than by shapes and structure. They have two principal limitations. Reference and sensed images must have somehow ‘similar’ intensity functions, either identical (and then correlation-like methods can be used) or at least statistically dependent (this typically occurs in multimodal registration). Area-based methods often employ pyramidal image representations and sophisticated optimization algorithms to find the maximum of the similarity matrix [1] [4].

Feature-based matching methods are typically applied when the local structural information is more significant than the information carried by the image intensities. They allow registering images of completely different nature (like aerial photograph and map) and can handle complexity between-image distortions. The common drawback of the feature-based methods is that the respective features might be hard to detect and/or unstable in time. The crucial point of all feature-based matching methods is to have discriminative and robust feature descriptors that are invariant to all assumed differences between the images [1] [4].

2.4. Problem definition

A challenge still remained in medical image registration is that how fast and accurate to estimate parameters with robustness to noise of the images. Neural networks inspired by biological brain structure are parallel learning algorithms can be used to estimate parameters and approximate functions that need a large number of inputs. It is most commonly used in medical image registration because once it is trained, any input pattern can be computed through the networks extremely fast to get an output. In this study, the registration of MRI medical image is investigated using Scale-Invariant Feature Transform (SIFT), Discrete Cosine Transform (DCT), Discrete Wavelet Transform (DWT), and Discrete Curvelet Transform (CT) as inputs for artificial neural network. The problem is to reduce the training and testing error of the network.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N}(t_i - a_i)^2}{N}} \tag{2.3}$$

Where, N is the number of training or testing sets, t is target of outputs, and a is actual outputs from the network.

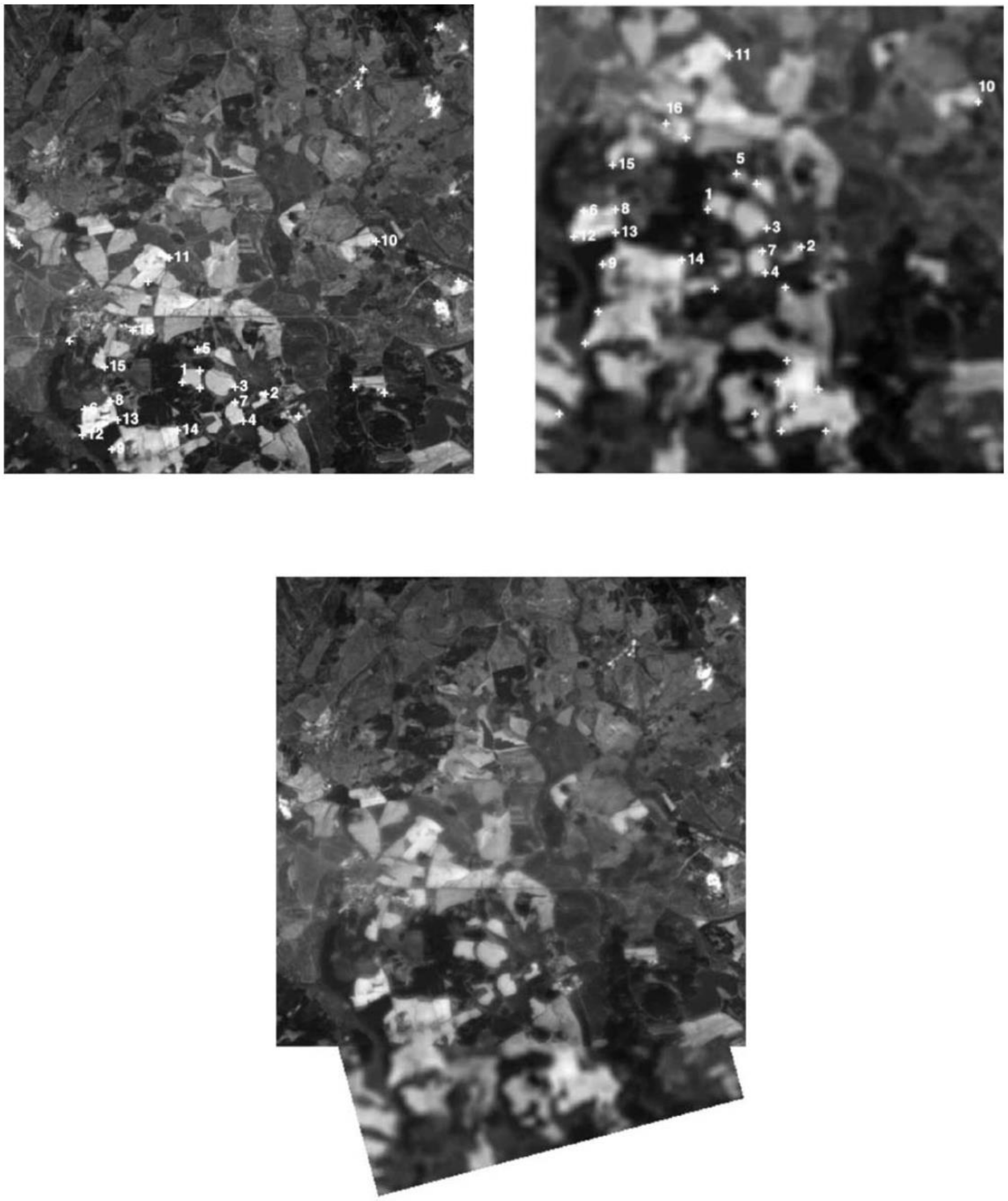

Figure 2.3 Feature-based image registration; Feature-based image registration; in these two satellite images, control points (corners) were matched using invariants based on complex moments [20].The numbers identify corresponding control points. The bottom image shows the registration result

Chapter 3. Background

The main points of neural network approaches in image registration are the selection of parameters to train the network and the type of neural network to use. Previous work used Discrete Cosine Transform (DCT) coefficients, keypoints from Scale Invariant Feature Transform (SIFT), and wavelet neural network [17] [19]. In this chapter, we're going to review those algorithms and discuss how it can be meaningful solution to solve medical image registration problem.

3.1. Affine transform introduction

Before moving on to the main algorithm, images are used for inputs, should be affined based on several transformations. In geometry, an affine transformation is a function between affine spaces that preserves points, line, and planes. Examples of affine transformations include rotation, scaling, translation, shear mapping, and compositions of them in any combination. An affine map is composition of two functions: a translation and a linear map. If the linear map is represented as a multiplication by a matrix A and the translation as the addition of a vector $\vec{b}$, an affine map f acting on a vector $\vec{x}$ can be represented as below.

$$\vec{y} = f(\vec{x}) = A\vec{x} + \vec{b} \qquad (3.1)$$

Where $\vec{y}$ is the transformed coordinates and $\vec{x}$ is the original coordinates. A 3 by 3 matrix form is usually used instead of 2 by 2 matrixes for translation parameter which is not a linear operation in 2-D representation. By adding another row of translation enables to be performed as a linear operation as below.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{3.2}$$

A rotation operation can be represented as (3.3). The sample image that applied a rotation transform is shown at Figure 3.1.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{3.3}$$

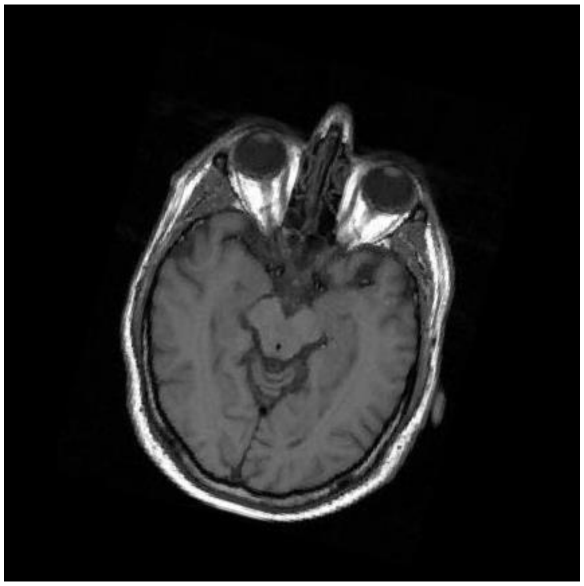

Figure 3.1 Affine transformation (Rotation)

Scale, shear, and translation transform can be performed using equations (3.4), (3.5), and (3.6), respectively. The example images applied those transform are shown below.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} scale_x & 0 & 0 \\ 0 & scale_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{3.4}$$

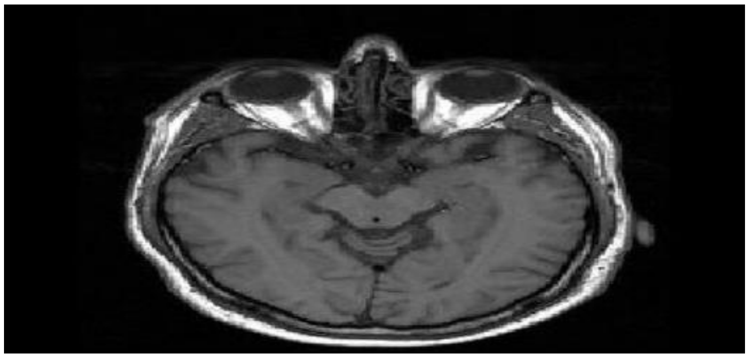

Figure 3.2 Affine transformation (Scale)

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & shear_x & 0 \\ shear_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{3.5}$$

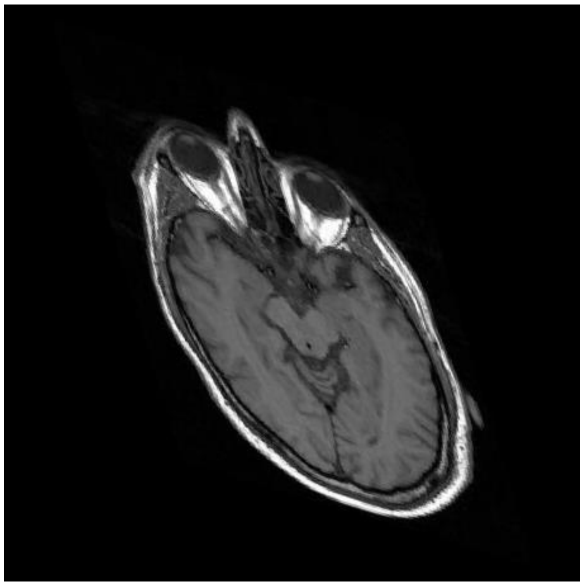

Figure 3.3 Affine transformation (Shear)

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & translation_x \\ 0 & 1 & translation_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{3.6}$$

In this study, all above transforms are applied at the same time with specific ranges. Some of sample images that applied transformation are shown at Figure 3.4. The images used in this study are MRI images of brain that were taken from an image database at Magnetic Resonance - Technology Information Portal [21]. Table 3-1 shows parameter values used at Figure 3.4 and the range of transformation parameters used in this thesis.

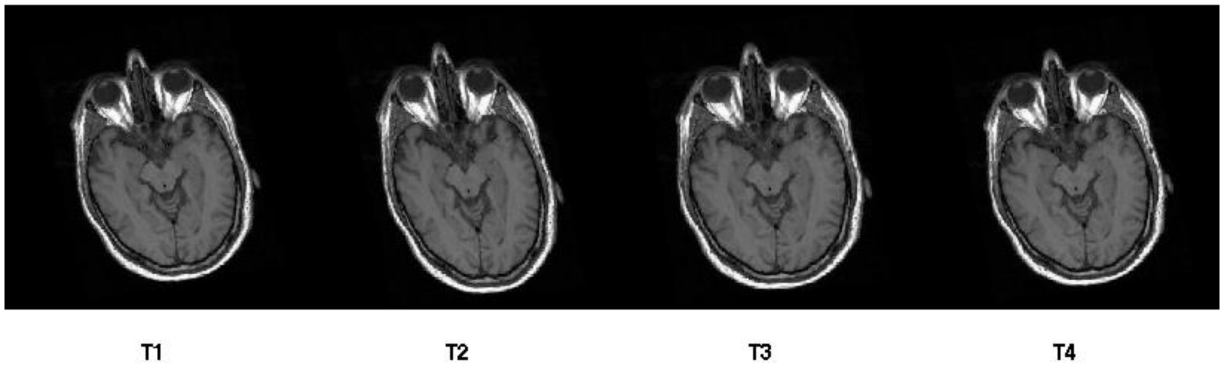

Figure 3.4 Examples of affine transformation

Table 3-1 Sample images with affine transformation

	T1	T2	T3	T4	Max. Range
Rotation(Degree)	-11	-8	-7	-11	-12 ~ 12
Scale(X-Axis)	0.9697	1.0915	0.9446	1.0878	0.9 ~ 1.1
Scale(Y-Axis)	0.9354	1.0485	1.0231	0.9394	0.9 ~ 1.1
Translation(X-Axis)[pixel]	0	3	1	5	-5 ~ 5
Translation(Y-Axis)[pixel]	-4	3	-2	2	-5 ~ 5
Shear(X-Axis)	0.0195	0.0523	0.0149	0.0301	-0.1 ~ 0.1
Shear(Y-Axis)	0.0394	0.0990	0.0147	0.0029	-0.1 ~ 0.1

3.2. Discrete cosine transform (DCT)

There have been several researches on using DCT coefficients for image registration and given decent results [17]. Frequency information can be found for any image and provide global information of the image. When viewing translation and rotation properties in the spectral domain, some interesting points arise. For example, the Fourier shift translation property of Fourier transform, which states that if I1 and I2 are two images differ only in their translation, is related by:

$$I_2(x,y) = I_1(x - x_0, y - y_0) \leftrightarrow F_2(u,v) = F_1(u,v)e^{-j2\pi(ux_0 + vy_0)} \tag{3.7}$$

It can also be shown that other affine transformations have distinct representation in Fourier domain. Thus, it can be concluded that taking an appropriate amount of coefficients result in providing the necessary information for estimating registration parameters.

The advantage of using DCT instead of Discrete Fourier Transform (DFT) is that DCT can compact information better than DFT, which means ability to pack the energy of the spatial sequence into as few frequency coefficients as possible. Figure 3.5 shows how DCT coefficients are compacted to top-left corner compared to DFT coefficients. This is very important in image compression and beneficial in other applications, too.

The reason DCT is more efficient than DFT is that it uses different boundary conditions which lead to remove discontinuities containing a lot of high frequencies. So, it has a smooth continuous transition. In this study, we use a 10 by 10 window to extract coefficients from top-left corner of DCT.

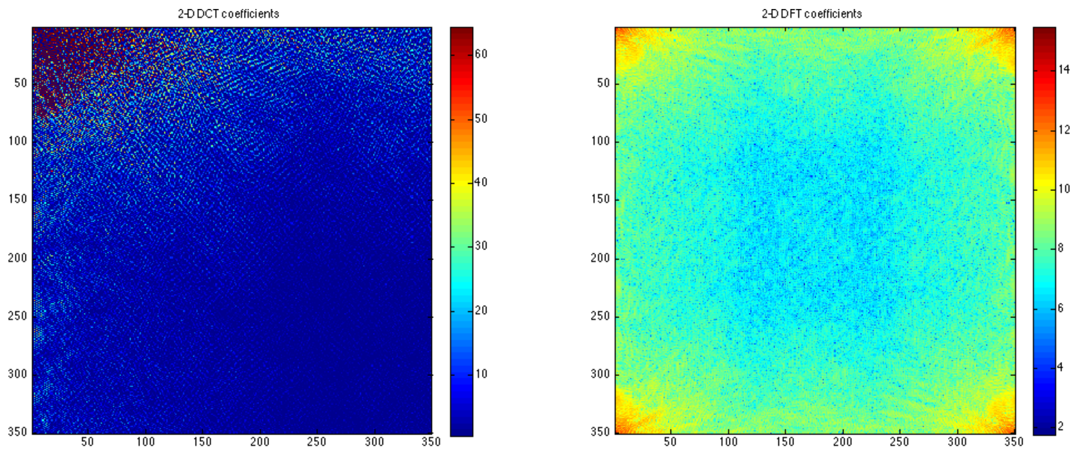

Figure 3.5 Comparison of DCT and DFT

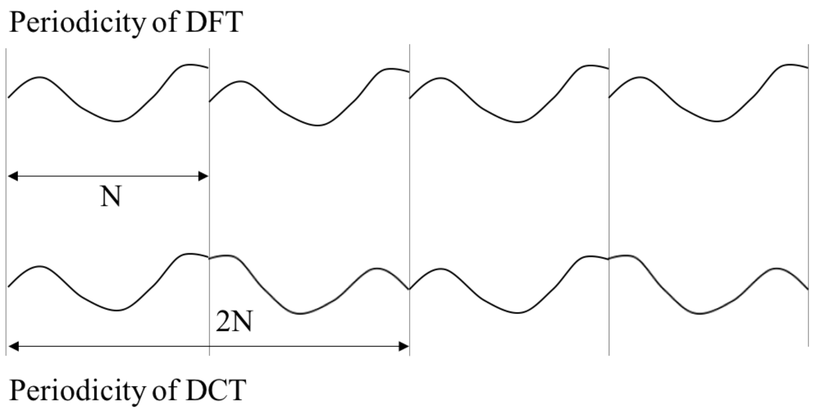

Figure 3.6 Periodicity of DFT and DCT

3.3. Scale invariant feature transform (SIFT) algorithm

SIFT algorithm, was developed in 1999 by Lowe David G., is usually used in computer vision such as image matching and object recognition [22]. The advantage of this algorithm is that the features are invariant to scale, rotation, and translation. It has been used in image registration as well [23]. In this study, SIFT keypoints are considered as inputs for neural network training.

SIFT algorithm mainly consists of two steps. Finding key points and assigning descriptor to it as the first step, and in a second step, matching original image and target image using descriptors is performed as Figure 3.7. In other words, features, which are defined as extrema, could be found from various scaled images. Then, the information including orientation and magnitude that can account for the features is assigned. In a matching step, features from original image and target features are matched by comparing the distance between them.

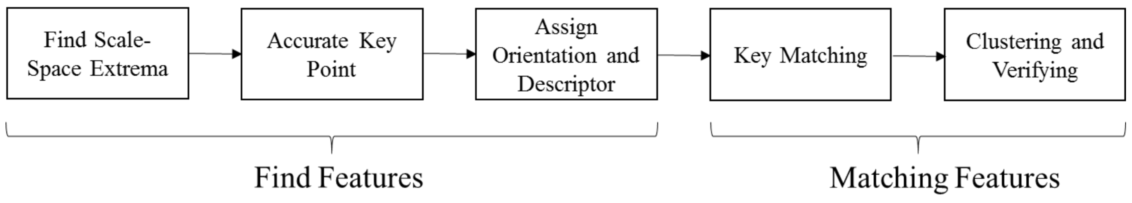

Figure 3.7 SIFT algorithm sequences

3.3.1. Scale-space extrema

The advantage of SIFT algorithm is that it can extract keypoints that are invariant to scale, orientation, luminance, and affine transform. This step is the process to extract interest point to be expected to be invariant scale and orientation. Difference of Gaussian (DOG) is used to improve the processing time. Gaussian pyramid is the process to calculate Gaussian convolution with various scales as Figure 3.8. The difference of pyramid image is the similar result with the second derivative of Laplacian of Gaussian (LOG). So, we can measure the features of the original image.

In order to be invariant to the change of illuminance, it figures out to select the object’s maximum or minimum. After building Gaussian pyramid, key candidate is extracted from the point of blob using DOG. Key candidate is to be considered when it is the maximum or the minimum comparing to its 26 around points in upper or lower scale levels like Figure 3.9.

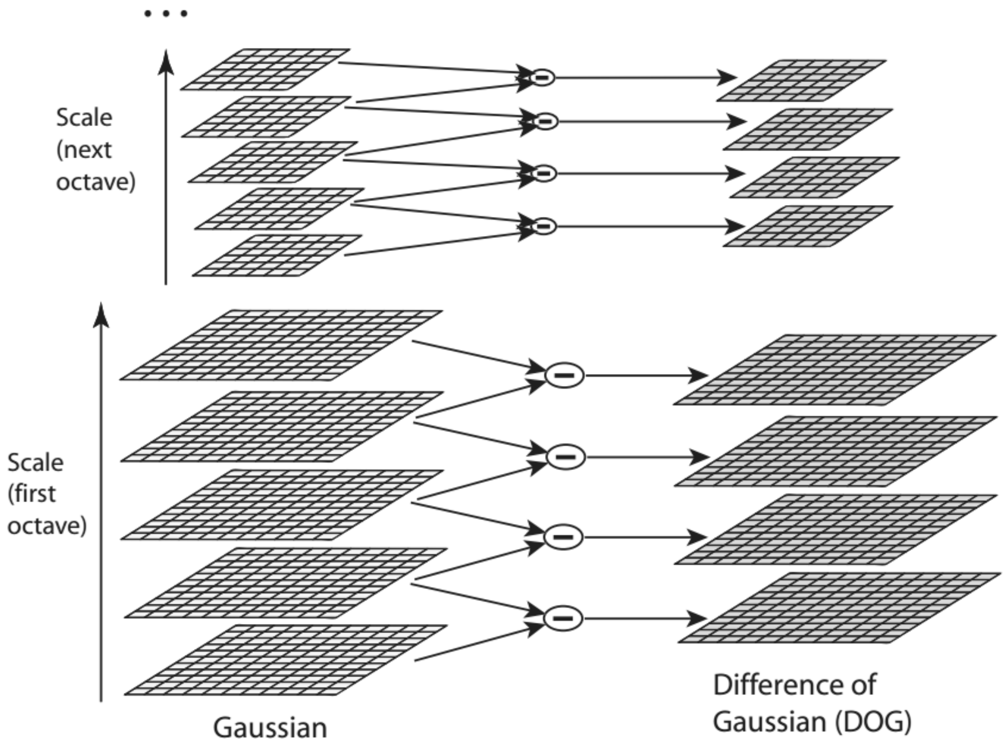

Figure 3.8 Gaussian pyramid & DOG [22]

Due to the comparison method, if DOG generates n, n+3 Gaussian images should be created for 1 Octave, and the rate of scale change is $2^{\frac{1}{n}}$.

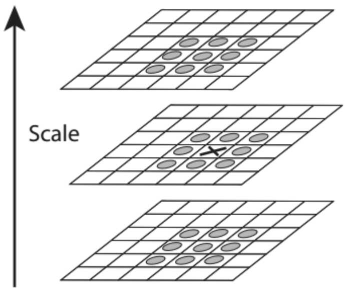

Figure 3.9 Key point [22]

3.3.2. Accurate keypoints

Once key candidates are selected, more accurate points should be extracted which represents features clearly. Among the candidates, there might be low contrast points or edge key points which are not good features to represent.

Lowe applied Taylor expansion, Brown suggested, to find more accurate keypoints and calculate contrast of the point using interpolation to filter out keypoints. Equation (3.8) is to find extrema using interpolation with Taylor second expansion.

$$\begin{aligned} D(X) &= D + \frac{\partial D^T}{\partial X} X + \frac{1}{2} X^T \frac{\partial^2 D}{\partial X^2} X, (X = (x - x_0, y - y_0, \sigma - \sigma_0)) \\ \hat{X} &= -\frac{\partial^2 D^{-1}}{\partial X^2} \frac{\partial D}{\partial X} \end{aligned} \tag{3.8}$$

This value is applied to Taylor first expansion (3.9) and if the contrast is below 0.03, the points are discarded.

$$\begin{aligned} D(\hat{X}) &= D + \frac{1}{2} \frac{\partial D^T}{\partial X} \hat{X} \\ if \left| D(\hat{X}) \right| &< 0.03 => discard \end{aligned} \tag{3.9}$$

3.3.3. Eliminating edge responses

It is not sufficient to reject keypoints with low contrast. So, Lowe used Harris corner detection method to remove edges. He set the threshold $r = 10$ as an appropriate value.

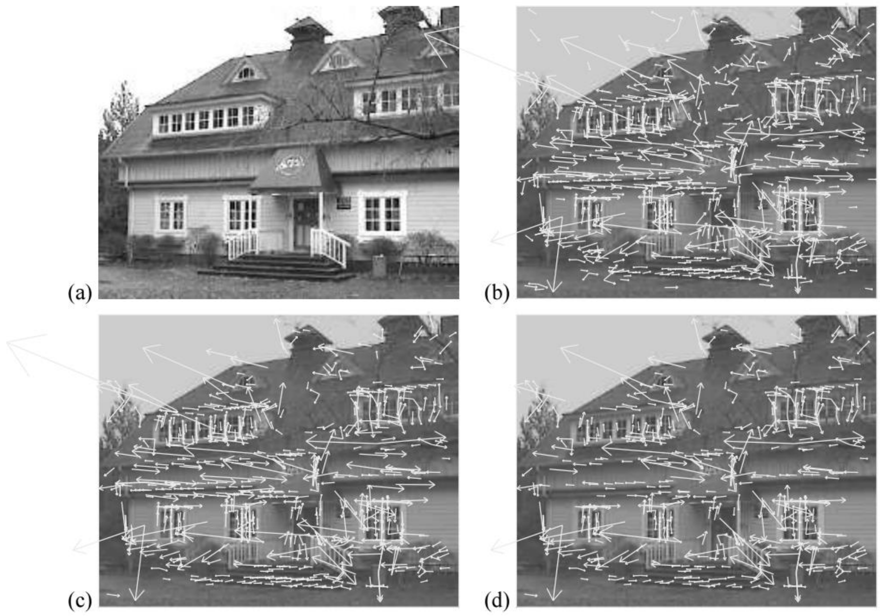

Figure 3.10 Keypoints selection steps [22]

Figure 3.10 shows the stages of keypoint selection. (a) is the 233x189 pixel original image. (b) is the initial 832 keypoints locations at maxima and minima of the difference of Gaussian function. Keypoints are displayed as vectors indicating scale, orientation, and location. After applying a threshold on minimum contrast, 729 keypoints remain as in (c). The final 536 keypoints are remained in (d).

3.3.4. Orientations

By assigning a consistent orientation to each keypoint based on local image properties, the keypoint descriptor can be represented relative to this orientation. Gaussian blurring is applied to the image of 16 by 16 around keypoint then assigned the orientation and scale of the gradient to each point. Equation (3.10) is used where L is the Gaussian blurred image.

$$\begin{aligned} m(x,y) &= \sqrt{(L(x+1,y)-L(x-1,y))^2+(L(x,y+1)-L(x,y-1))^2} \\ \theta(x,y) &= \tan^{-1}((L(x+1,y)-L(x-1,y))/(L(x,y+1)-L(x,y-1))) \end{aligned} \tag{3.10}$$

The orientation of keypoint is determined by using all magnitudes and orientations of the 16 by 16 image. An orientation histogram is formed with 36 bins covering 360 degree range of orientation. So, the peak value is the orientation of the keypoint. If there exist more than 1 peak point over 80% of the peak, this keypoint is considered with multiple keypoints.

3.3.5. Keypoint descriptors

The first three steps ensured invariance to location, scale, and rotation by assigning to each keypoint a location, scale, and orientation. So, the last step is to ensure invariant to the change of illumination. This can be done by sampling gradient magnitudes and orientations around each keypoint. The Gaussian image is selected based on scale of the keypoint, then orientation histogram with 8 bins is formed over 4 by 4 neighborhood regions from sampled magnitude and orientation values in 16 by16 image. A Gaussian weighting function is used to assign a weight to the magnitude of each sample point. The descriptor is a feature vector consisting of all the values in the histogram.

3.3.6. Image matching

In this step, the Euclidean distance of keypoints between original image and target image is calculated as (3.11).

$$D = \sqrt{\sum_{i=1}^{n}(a_i - b_i)^2} \qquad (3.11)$$

The keypoint which has the minimum distance D is matching keypoint. Dr. David Lowe suggested a method to check whether the matching is done correctly using the ratio between the closest keypoint and the second closest keypoint. When the value is over 0.8, there is high probability of miss matching so they are discarded. Figure 3.11 shows the keypoints and its descriptors of original MRI brain image we used in this study. Figure 3.12 is for the affine transformed image. Matched keypoints between these two images are shown at Figure 3.13.

There are 23 matched keypoints for these images. The coordinate information of the affined image is used to train neural network.

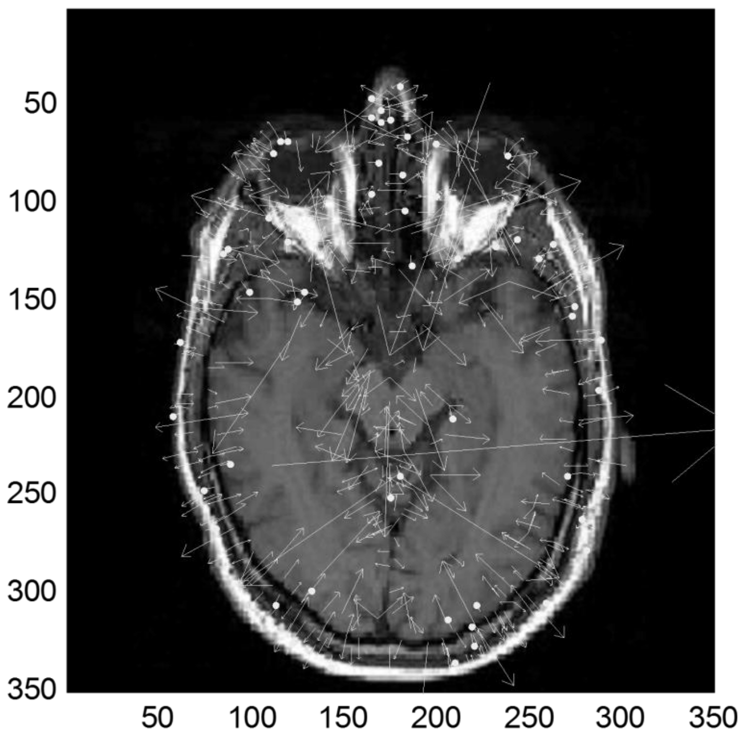

Figure 3.11 SIFT keypoints of the original image

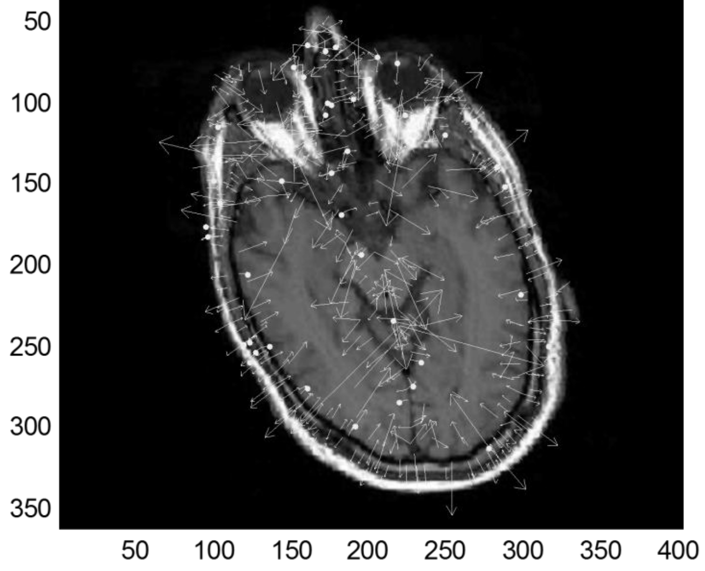

Figure 3.12 SIFT keypoints of the affine transformed image

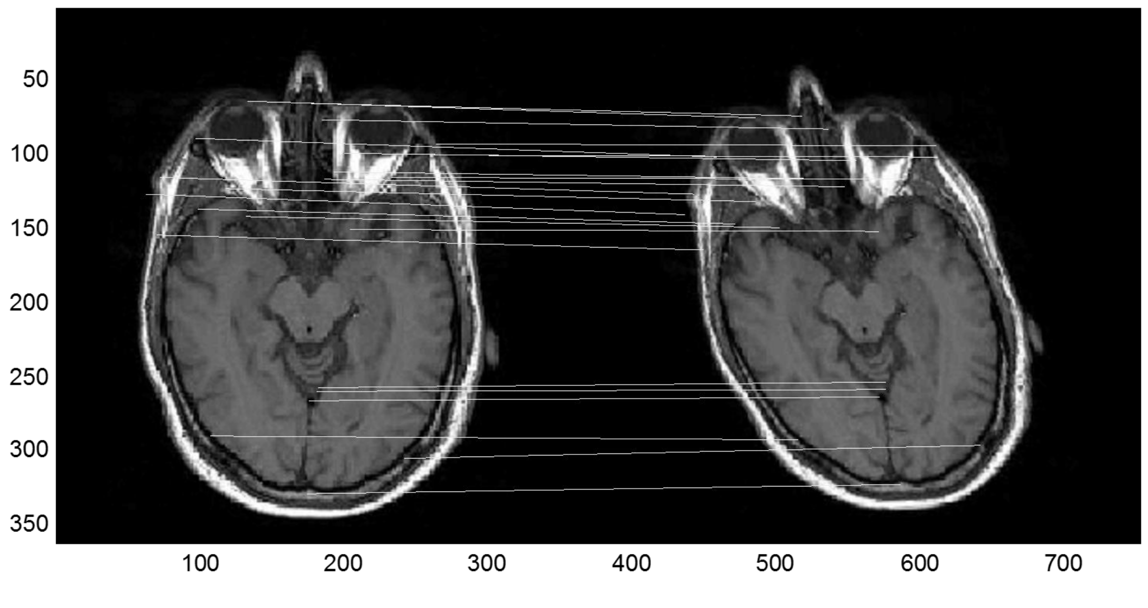

Figure 3.13 23 matched keypoints between Figure 3.11 and Figure 3.12

3.4. Wavelet neural network (WNN)

Wavelet neural networks combine the theory of wavelets and neural networks into one [24]. A wavelet neural network generally consists of a feedforward neural network with one hidden layer and activation functions are from an orthonormal wavelet family. The structure of wavelet neural network is similar to the feedforward neural network as Figure 3.14. The neurons of wavelet are usually referred to as wavelons [24].

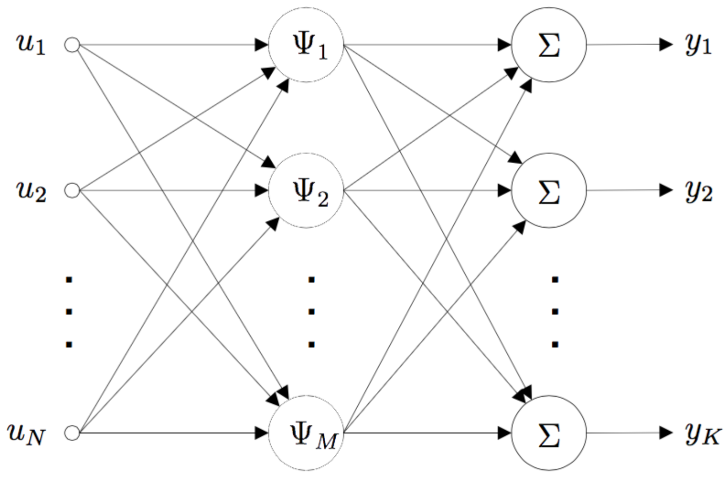

Figure 3.14 Structure of wavelet neural network

The simplest form of wavelet neural network is one with a single input and a single output. The hidden layer of neurons consists of wavelons whose parameters include the wavelet dilation and translation values. The output of a wavelet neural network is a linear weighted combination of the wavelet activation functions. The single input wavelon is defined as:

$$\varphi_{\lambda,t}(u) = \varphi(\frac{u-t}{\lambda}) \tag{3.12}$$

Where λ and t are dilation and translation parameters respectively. The architecture of single wavelet network is shown in Figure 3.15. The hidden layer consists of M wavelons and the output is just a summer. So, the output can be defined with a weighted sum of the wavelon outputs as (3.13).

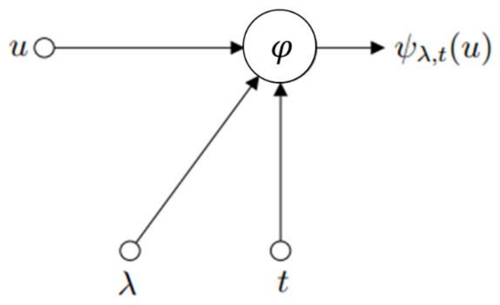

Figure 3.15 Wavelet neuron

$$y(u) = \sum_{i=1}^{M} w_i \varphi_{\lambda_i t_i}(u) + \bar{y} \qquad (3.13)$$

In a wavelet network all parameter $\bar{y}, w, \lambda, t$ are adjustable by learning algorithm. A multidimensional wavelet network is an extension of single input and output architecture into multi-input and outputs system. Therefore the input-output mapping of the network is defined as:

$$y_\theta(u) = \sum_{i=1}^{M} w_i \varphi\left(\frac{u - t_i}{\lambda_i}\right) + \bar{y} \qquad (3.14)$$

The parameters $\bar{y}, w, \lambda, t$ should be formed into one vector θ . The objective function to be minimized is:

$$C(\theta) = \frac{1}{2} E\{(y_\theta(u) - f(u))^2\} \qquad (3.15)$$

The minimization is performed using a stochastic gradient algorithm. This recursively modifies θ , in the opposite direction of the gradient of

$$c(\theta, u_k, f(u_k)) = \frac{1}{2}(y_\theta(u_k) - f(u_k))^2 \qquad (3.16)$$

The gradient for each parameter of θ can be found by calculating the partial derivatives of $c(\theta, u_k, f(u_k))$ as follows:

$$\begin{aligned}
\frac{\partial c}{\partial \bar{y}} &= e_k \\
\frac{\partial c}{\partial w_i} &= e_k \varphi(z_{ki}) \\
\frac{\partial c}{\partial t_i} &= -e_k w_i \frac{1}{\lambda_i} \varphi'(z_{ki}) \\
\frac{\partial c}{\partial \lambda_i} &= -e_k w_i \left(\frac{u_k - t_i}{\lambda_i^2}\right) \varphi'(z_{ki})
\end{aligned} \tag{3.17}$$

Where $e_k = y_\theta(u_k) - f(u_k), z_{ki} = \dfrac{u_k - t_i}{\lambda_i}$ and $\varphi'(z) = \dfrac{d\varphi(z)}{dz}$.

In this study, the Morlet wavelet function is exploited for activation functions of neuron.

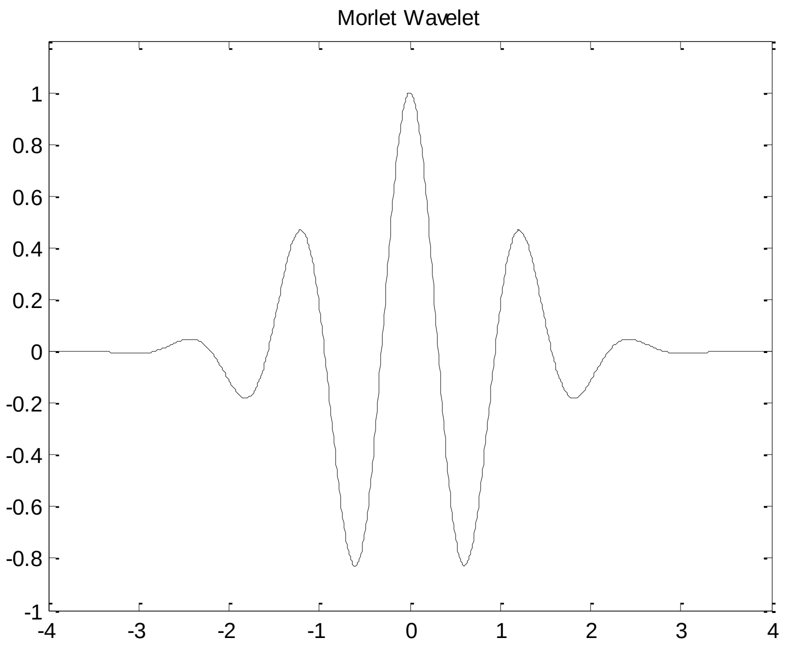

Figure 3.16 Morlet wavelet function

3.5. Curvelet transform

In this chapter, a new approach using neural network for image registration is suggested. The Curvelet Transform (CT) is used to extract features from the images sets as inputs to train the network [25]. As being extracted coordinate information from SIFT algorithm, we suggest a method to extract curvelet keypoints information instead of using statistical measure of curvelet coefficients.

3.5.1. Introduction of curvelet transform (CT)

Multiresolution methods are deeply related to image processing, biological and computer vision. The curvelet transform is a multiscale directional transform that allows an almost optimal non-adaptive sparse representation of objects with edges [25]. Although wavelet transform becomes increasingly popular in engineering fields, it only performs well at representing point singularities since they ignore the geometric properties of structures and do not exploit the regularity of edges. So, wavelet-based applications are computationally inefficient for geometric features with line and surface singularities. In other words, in the field of biological and computer vision, wavelets do not supply good direction selectivity, which is an important response property of objects. To overcome the missing directional selectivity of conventional 2-D discrete wavelet transforms, a multiresolution geometric analysis (MGA), named curvelet transform, was proposed by Candes and Donoho in 2000 [26]. Unlike the isotropic elements of wavelets, the curvelet transform possess very high directional sensitivity and anisotropy with the needle-shaped elements. So, it can represent edges and other singularities along curves much more efficiently than traditional transforms, i.e. using significantly fewer coefficients for a given accuracy of reconstruction.

The implementation of the first generation of curvelet uses a pre-processing step involving a special partitioning of phase-space followed by the ridgelet transform. However, in the second generation of curvelet, the algorithm is redesigned to make them easier to use. As a result, Fast Discrete Curvelet Transforms (FDCTs) is considerably simpler, faster, and less redundant than the first proposals [26]. There are two distinct implementation ways of FDCT: unequispaced FFTs (USFFT-based) and Wrapping-based. In this study, we used a wrapping-based method, which is faster than USFFT one.

3.5.2. Digital curvelet transforms

3.5.2.1. Digital coronization

The digital transformation is linear and takes as input Cartesian arrays of the form $f[t_1,t_2], 0 \le t_1,t_2 < n$. The coefficients $c^D(j,l,k)$ are obtained by:

$$c^D(j,l,k) := \sum_{0 \le t_1,t_2 < n} f[t_1,t_2]\overline{\varphi^D_{j,l,k}[t_1,t_2]} \qquad (3.18)$$

Where each $\varphi^D_{j,l,k}$ is a digital curvelet waveform (the superscript D stands for "digital"). In the continuous-time definition, the window U_j smoothly extracts frequencies near dyadic corona $\{2^j \le r \le 2^{j+1}\}$ and near the angle $\{-\pi 2^{-j/2} \le \theta \le \pi 2^{-j/2}\}$ as Figure 3.17. In discrete-time domain, it is convenient to replace the concepts by Cartesian equivalents based on concentric squares and shears as Figure 3.18. The Cartesian analog to the family $W_j(\omega) = W(2^{-j}\omega)$, would be a window of the form:

$$\widetilde{W}_j(\omega) = \sqrt{\Phi^2_{j+1}(\omega) - \Phi^2_j(\omega)},\ j \ge 0,$$

Where Φ is defined as the product of low-pass one dimensional windows. This is how to separate scales in a Cartesian-friendly fashion and the angular localization V is set as:

$$V_j(\omega) = V(2^{|j/2|}\omega_2/\omega_1)$$

Then, the Cartesian window is defined as:

$$\tilde{U}_j(\omega) := \tilde{W}_j(\omega)V_j(\omega) \qquad (3.19)$$

However, $\tilde{U}_j(\omega)$ is hard to represent frequencies near the wedge, so a modified window is suggested as:

$$\tilde{U}_{j,l}(\omega) := \tilde{W}_j(\omega)V_j(S_{\theta_l}\omega) \qquad (3.20)$$

Where S_θ is the shear matrix,

$$S_\theta := \begin{pmatrix} 1 & 0 \\ -\tan\theta & 1 \end{pmatrix}$$

When completed by symmetry around the origin and rotation by $\pm\pi/2$ radians, the $\tilde{U}_{j,l}$ defines the Cartesian to the family which is a concentric tiling whose geometry is displayed in Figure 3.18.

3.5.2.2. Curvelet transform using wrapping method

The process of curvelet transform with wrapping method is as follow.

(1) Both image and curvelet filter bank are transformed into Fourier domain.

$$\hat{f}[n_1, n_2], -n/2 \le n_1, n_2 < n/2$$

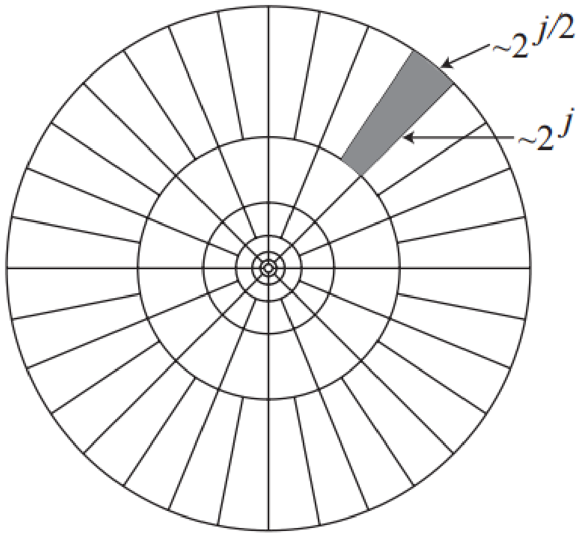

Figure 3.17 Induced tiling of the frequency plane. In Fourier space, curvelets are supported near a “parabolic” wedge, and the shaded are represents a generic wedge [26]

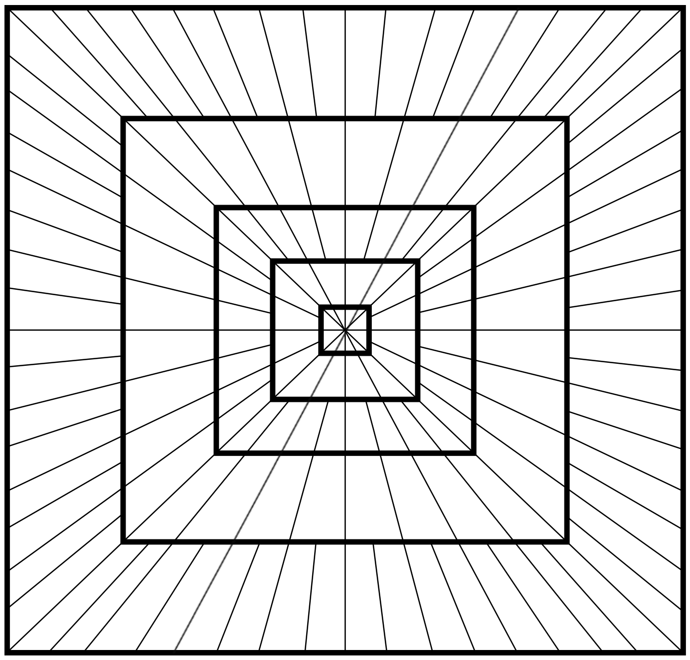

Figure 3.18 Digital tiling. The windows smoothly localize the Fourier transform near the sheared wedges obeying the parabolic scaling [26]

(2) Then the convolution of the curvelet with the image in spatial domain becomes the product in Fourier domain.

$$\tilde{U}_{j,l}[n_1,n_2]\hat{f}[n_1,n_2]$$

(3) Compute the curvelet coefficient by applying inverse Fourier transform. The wedge must be wrapped into rectangular to perform inverse Fourier transform because the frequency response of curvelet is non rectangular wedge.

$$\tilde{f}_{j,l}[n_1,n_2] = W(\tilde{U}_{j,l}\hat{f})[n_1,n_2]$$

Where the range for n_1 and n_2 is $0 \le n_1 < L_{1,j}$ and $0 \le n_2 < L_{2,j}$.

(4) By periodic wedge tiling wrapping is done, then collecting the rectangular coefficient ($c^D(j,l,k)$) area in the center by applying the inverse 2D FFT to each $\tilde{f}_{j,l}$. The overall procedure is described in Figure 3.20.

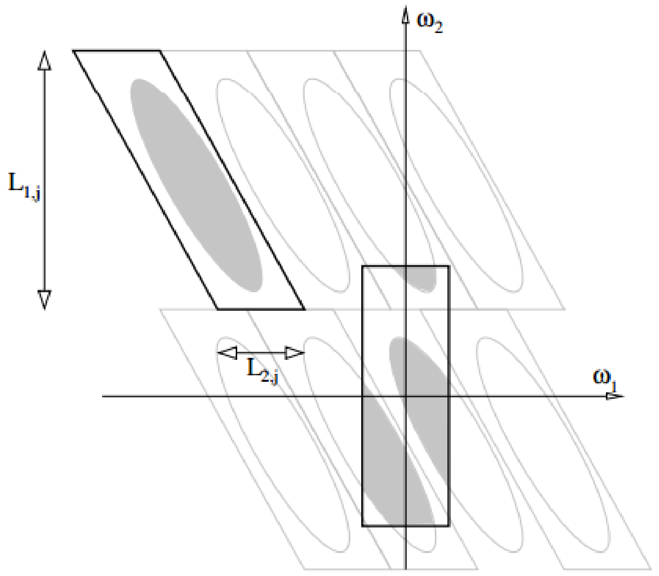

Figure 3.19 Wrapping data, initially inside a parallelogram, into a rectangle by periodicity [26]

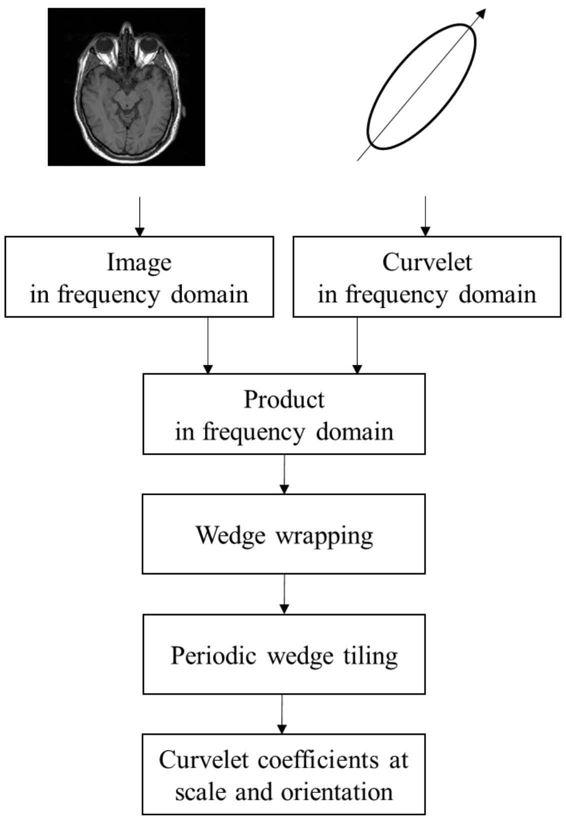
Image in frequency domain
Curvelet in frequency domain
Product in frequency domain
Wedge wrapping
Periodic wedge tiling
Curvelet coefficients at scale and orientation

Figure 3.20 Fast discrete curvelet transform via wrapping (FDCT)

3.5.2.3. Curvelet transform for brain MRI image

In this study, we resized the medical image to size 512 by 512. The feature extraction using curvelet is applied to the original brain image. For image size of 512 by 512, the maximum number of levels is 6. So the image is decomposed into 6 levels of scales using curvelet transform. The numbers of sub-bands at different scales are different. For 6 levels of decomposition, there are 1, 16, 32, 32, 64, and 1 sub-bands at decomposition level 1, 2, 3, 4, 5, and 6 respectively. Therefore, 6 levels decomposition creates 146 sub-bands of curvelet coefficients. Figure 3.21 shows the original image and coefficients of sub-bands at scale 1(center) and 6(right). From Figure 3.22 to Figure 3.25, curvelet coefficients of sub-bands at level 2, 3, 4, and 5 are displayed. All curvelet coefficients of sub-bands are figured in Figure 3.26.

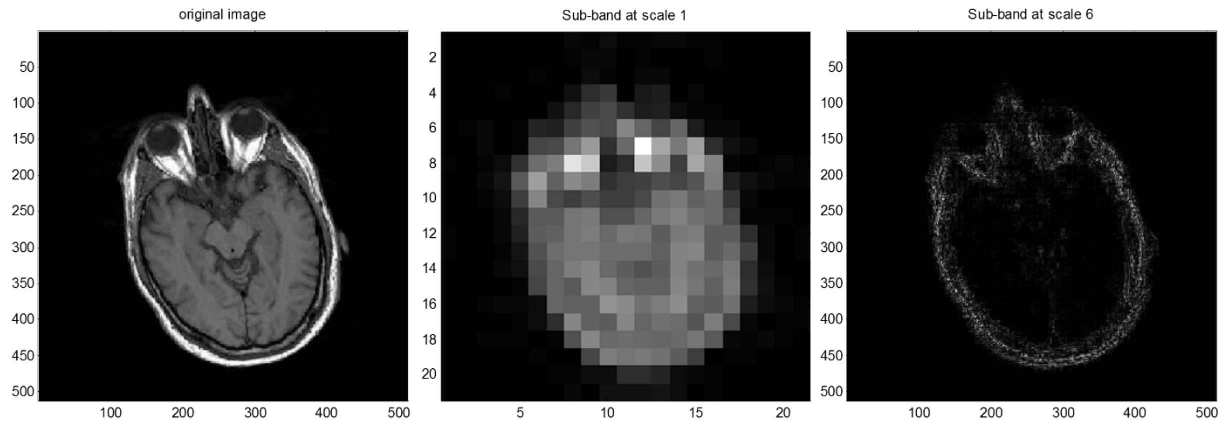

Figure 3.21 Curvelet coefficients (Original: left, Scale 1: center, Scale 6: right)

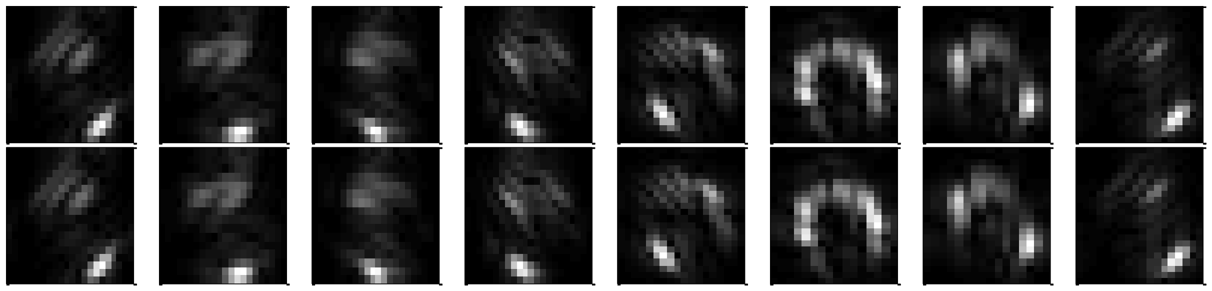

Figure 3.22 Curvelet coefficients at level 2 (16)

Table 3-2 The number of sub-bands of curvelet transform

Scale	Level 1	Level 2	Level 3	Level 4	Level 5	Level 6	Total
Number of Bands	1	16	32	32	64	1	146

Figure 3.26 displays all curvelet coefficients at a single image. Low frequency coefficients are positioned at the center of the figure. The outer coronae correspond to higher

frequencies. Each corona has four strips further subdivided in angular panels. And each panel represents coefficients at a specified scale and orientation. The displayed coefficients in Figure 3.26 are normalized at each scale level which means divided by maximum value of the level.

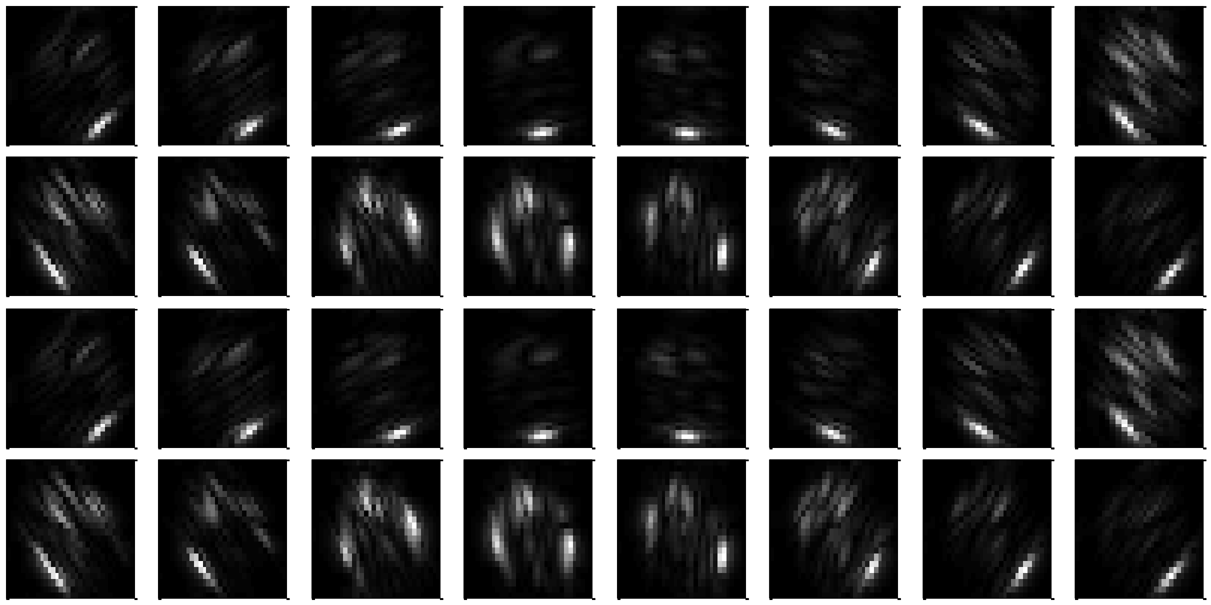
Figure 3.23 Curvelet coefficients at level 3 (32)

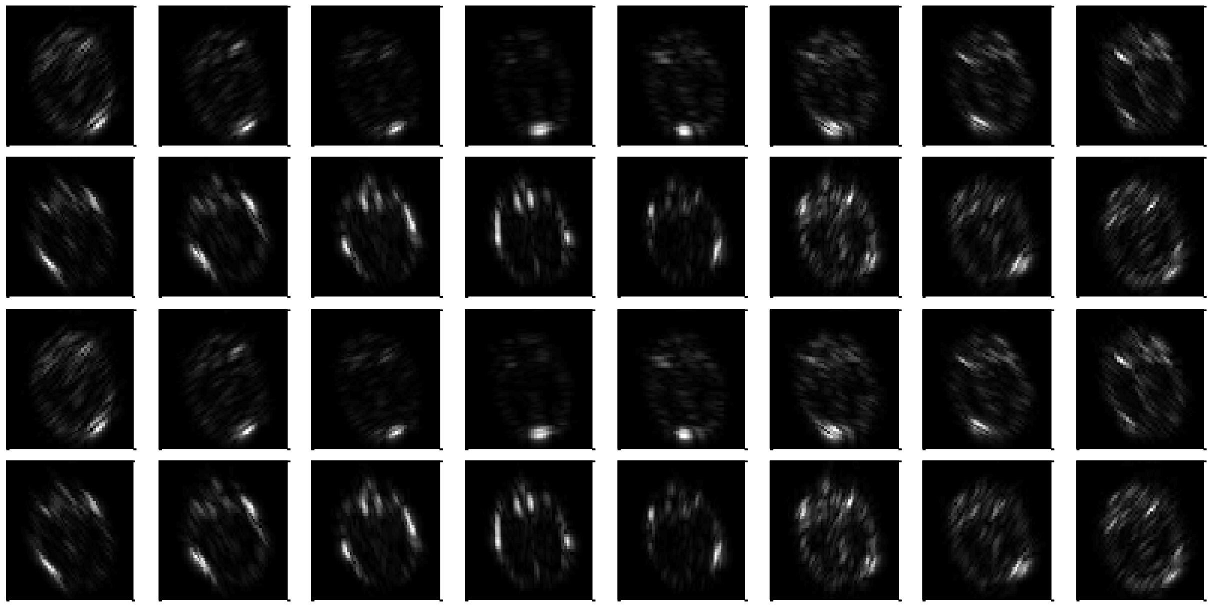
Figure 3.24 Curvelet coefficients at level 4 (32)

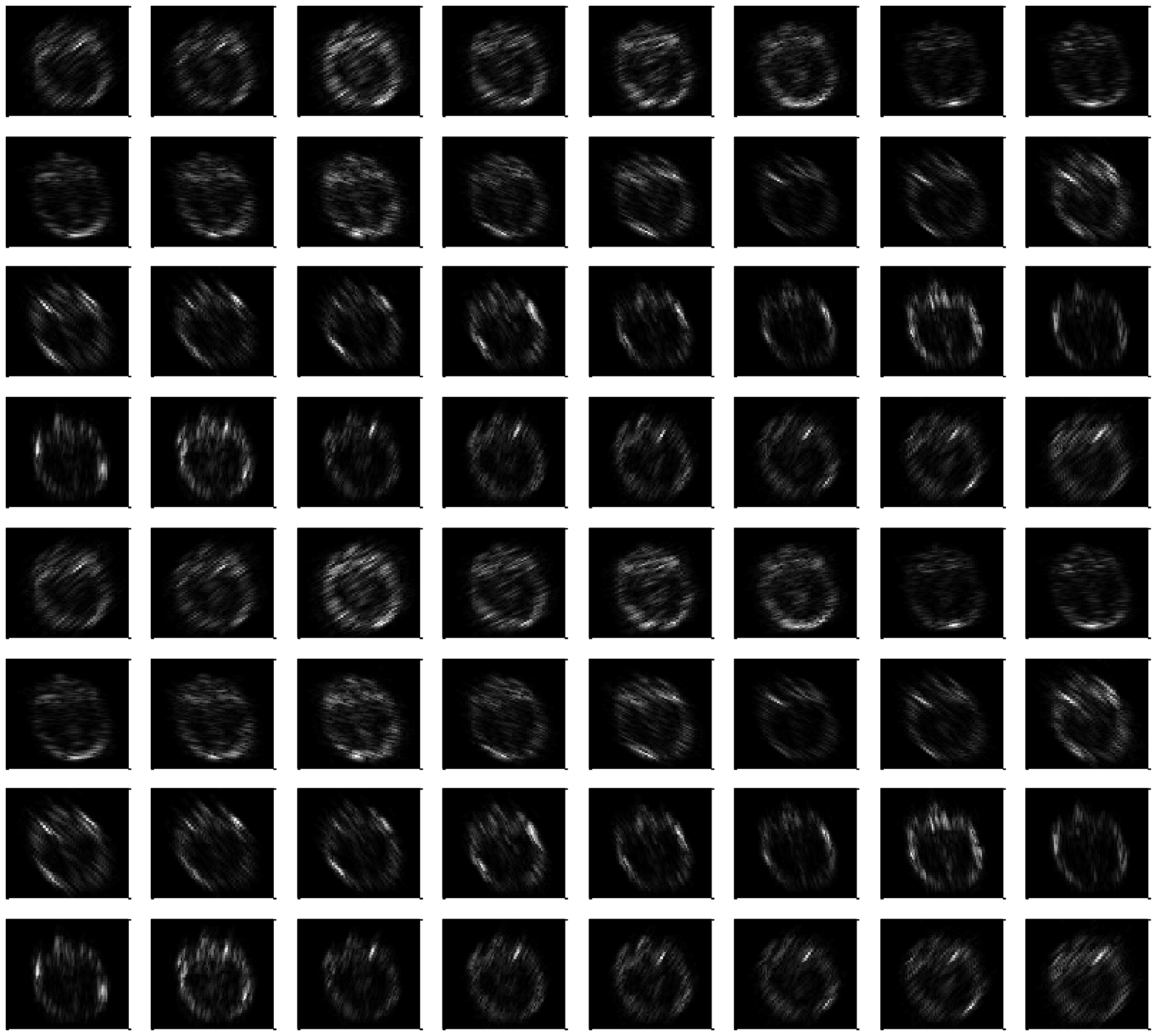

Figure 3.25 Curvelet coefficients at level 5 (64)

3.5.2.4. Curvelet coefficients via thresholding approach

If we use curvelet coefficients from all sub-bands of each scale for neural network inputs, there exist too many inputs for the network. Even the number of coefficients at level 1 is 21 by 21 (441 values), which needs a larger size of network structure. So, it might need extremely high computational cost if we take coefficients from scale 2 to 6 together.

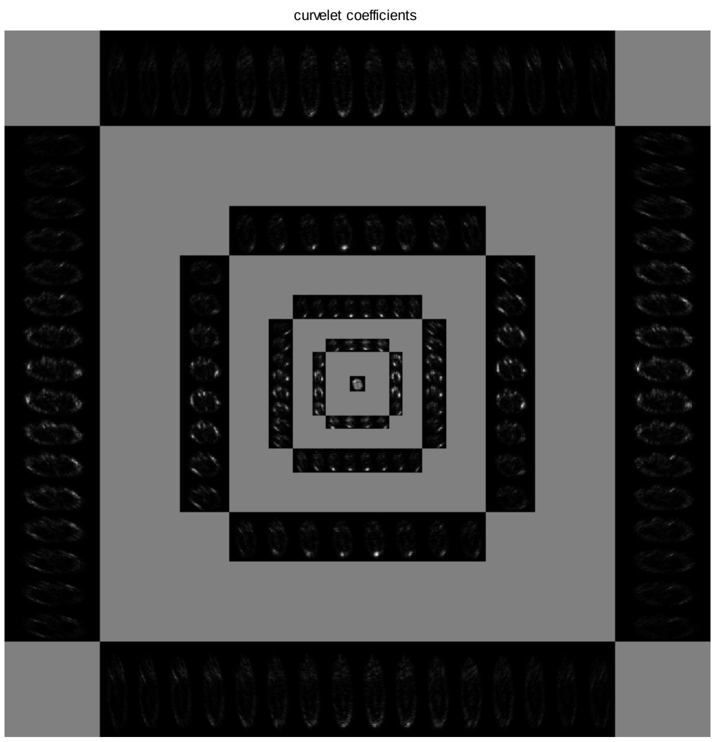

Figure 3.26 Curvelet coefficients plot

Statistical measures of curvelet coefficients such as mean, variance and entropy have been used to extract features using curvelet transform [27] [28] [29] [30] [31]. However, we found a poor accuracy using these approaches. So, the alternative approach is based on thresholding the curvelet coefficients [32]. We extract image features of MRI image by applying curvelet transform then take coefficients above threshold value from all sub-bands. Figure 3.27 shows an example, when we assume that we use 10 percent of the strongest coefficients from all sub-bands. The right image is reconstructed image with those 10 percent coefficients. The selected coefficients have enough information to reconstruct the image, so the right image shows pretty detail features of the original image. The problem we use these coefficients is that there still too many coefficients. With the 512 by 512 image, there are 73900 coefficients from 10 percent of curvelet coefficients. It still needs too big neural network structure which is ineffective comparing to 100 DCT and 100 keypoints from SIFT algorithm.

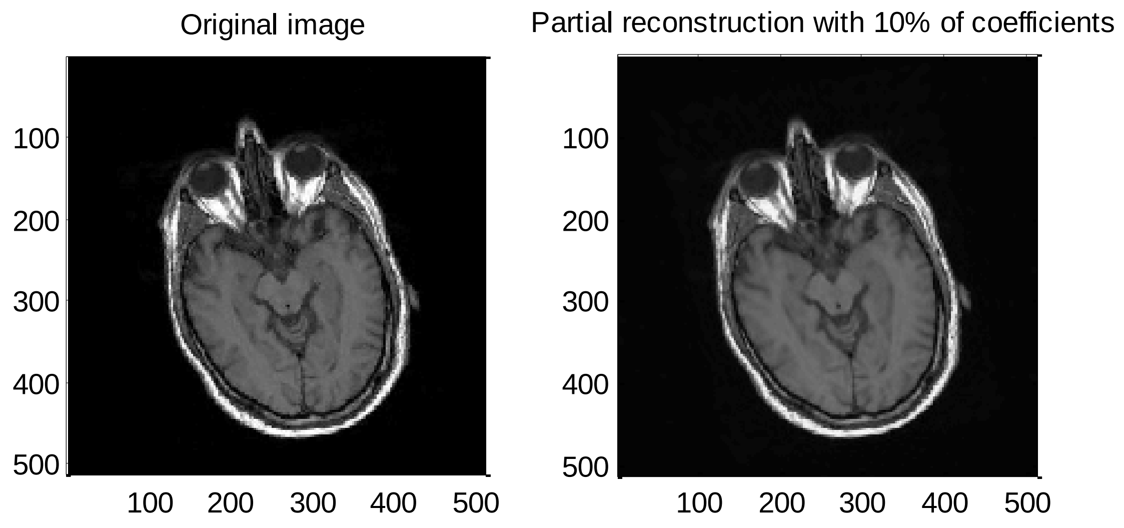

Figure 3.27 Reconstruction with the strongest 10% of coefficients from all sub-bands

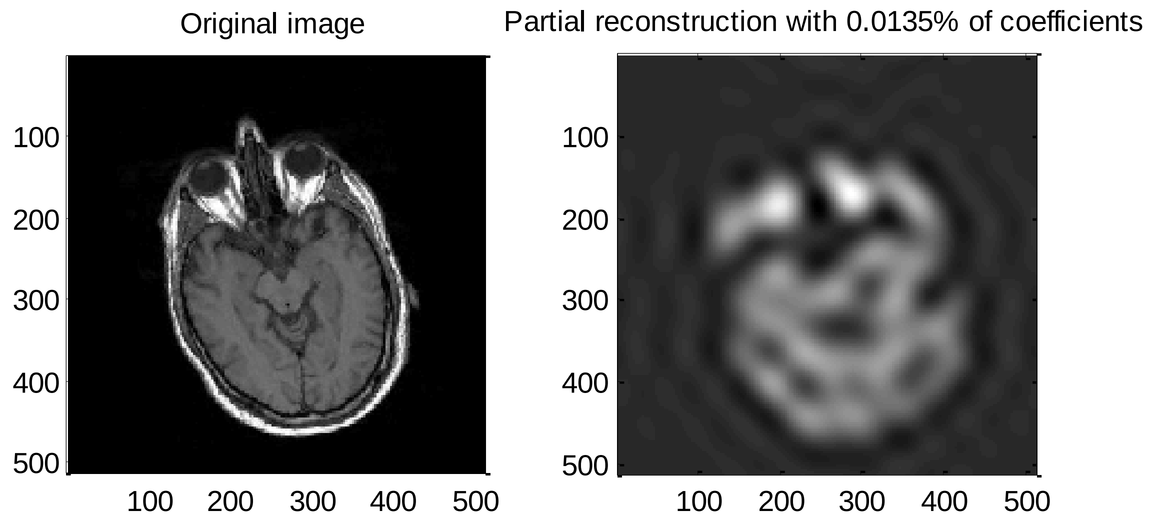

Figure 3.28 Reconstruction with the strongest 0.0135% of coefficients from all sub-bands

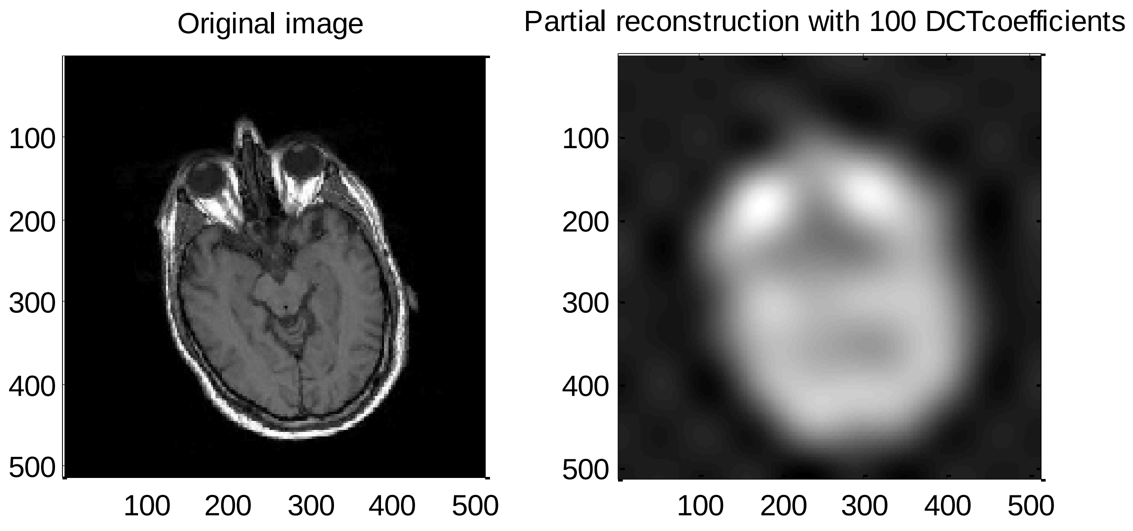

Figure 3.29 Reconstruction with 100 DCT coefficients

In order to compare the 100 DCT coefficients, we take 0.0135% of curvelet coefficients from all sub-bands which could generate exactly 100 coefficients. It shows that the number of coefficients to be extracted is appropriate for training the network, however it loses lots of information especially on curves, lines shown as Figure 3.28. If we compare the reconstruction image with 100 DCT coefficients, curvelet coefficients seem to have more features on curves and edges while DCT coefficients have coarse features of the image.

3.5.2.5. Curvelet keypoints based on modified thresholding method

Now, we suggest a method that can have critical features as well as keep the limitation of the number of coefficients based on modified thresholding approach of curvelet coefficients. In other words, the purpose of this algorithm is to get significant features with minimum number of coefficients. There are three steps for this algorithm as follow:

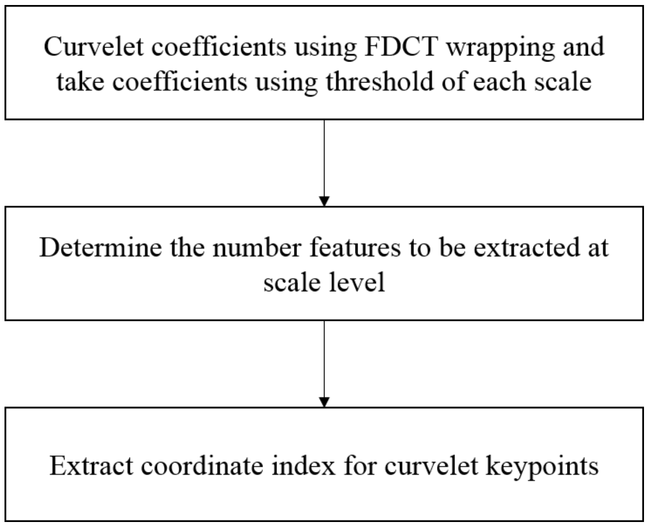
Curvelet coefficients using FDCT wrapping and take coefficients using threshold of each scale
Determine the number features to be extracted at scale level
Extract coordinate index for curvelet keypoints

Figure 3.30 Curvelet keypoints based on modified thresholding method

Modified thresholding method

At first step, we apply discrete curvelet transform to the image and take coefficients using modified thresholding method. The thresholding method in the previous section is to take a certain percentage value of coefficients from all sub-bands. This method might have possibility of ignoring important coefficients at specific scale level, if the value of coefficients from another scale is significantly high. Therefore, we suggest a way to take coefficients above threshold based on each scale level. This method can avoid ignoring coefficients at a certain scale level. In addition, we only consider the mid-bands (from 2 to 5 scales) of the curvelets because the first scale level (1st scale level) coefficients have coarse resolution and high local variations at fine resolutions at the highest level (6th scale level). Those two scale level do not have directional information which means that they are not dominant coefficients. Figure 3.31 is a bar graph of coefficients at each scale level and its threshold which is 70% of its maximum value. The number of coefficients that are taken from this method is 48, 34, 54, and 16 respectively and totally 152 coefficients at this example.

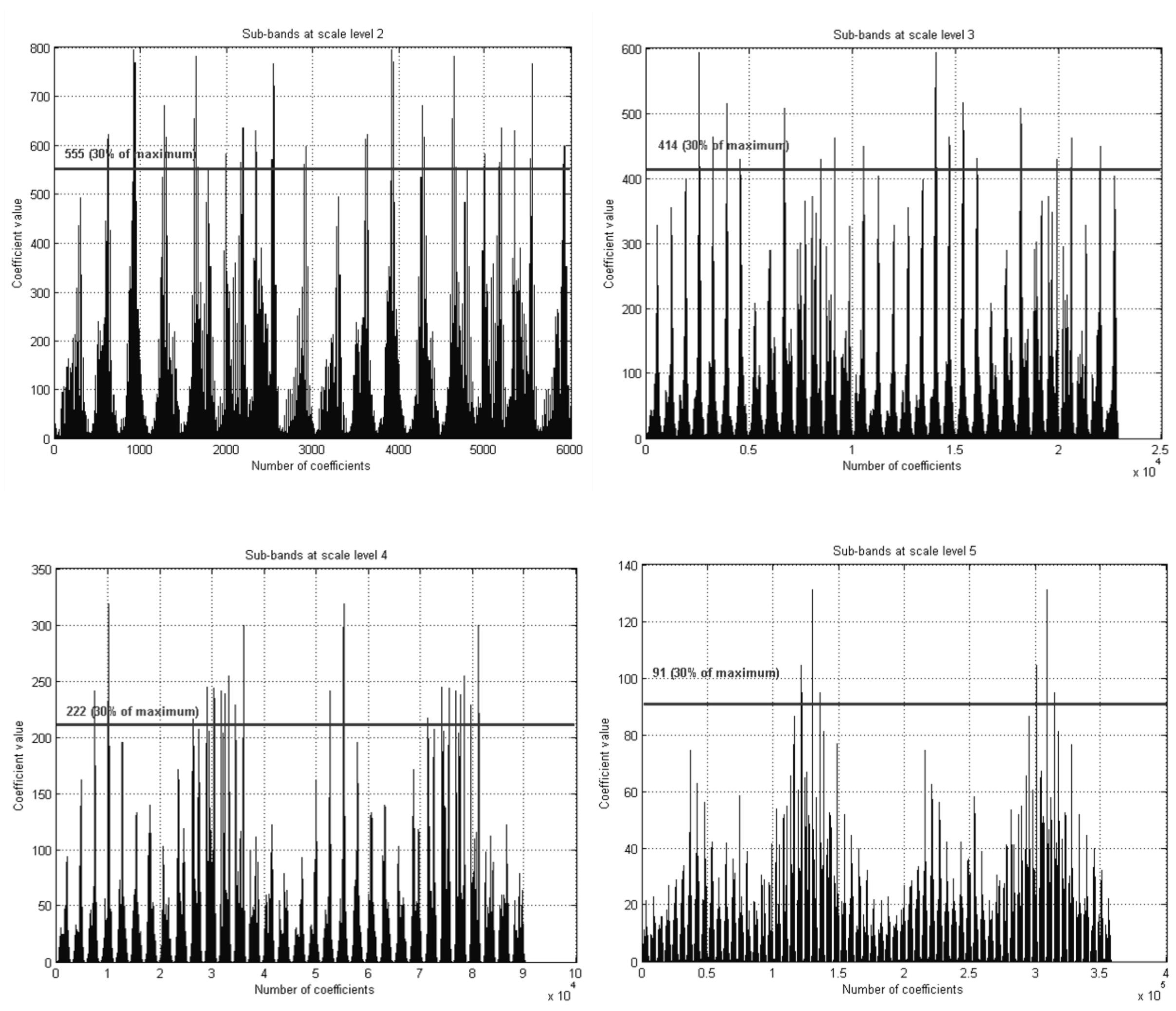

Figure 3.31 Threshold at each scale level. Take the highest 30% of coefficients from each scale. Threshold is 555, 414, 222, and 91 at scale level 2, 3, 4, and 5 respectively

Table 3-3 The number of coefficients at each scale level

	Scale 2	Scale 3	Scale 4	Scale 5	Total
Original coefficients	5,984	22,880	90,144	357,408	476,416
Threshold	48	34	54	16	152
Ratio (%)	31.58	22.37	35.53	10.53	100
Dynamic arrange	31	22	37	10	100

Determine the number of coefficients dynamically

In order to meet the number of coefficients we want to take (100), we apply a method that can re-arrange the number of coefficients we should take based on the ratio to all scale levels. The reason we called it dynamic re-arrangement is that the algorithm is to assign the number of coefficients no matter it lacks or over the maximum number of coefficients. Basically, the higher ratio scale has higher probability to extract more coefficients while lower ratio scale has higher chance to discard values when the total number is over the limitation. Table 3-3 is an example of the result at step 2. Therefore, 31, 22, 37, and 10 coefficients are taken from each scale level respectively based on its ratio. The number of total coefficients is 100. And from now on, we only consider those 100 coefficients. The selected coefficients at scale level 2 are shown at Figure 3.32. There are 31 coefficients which represented by small white circles.

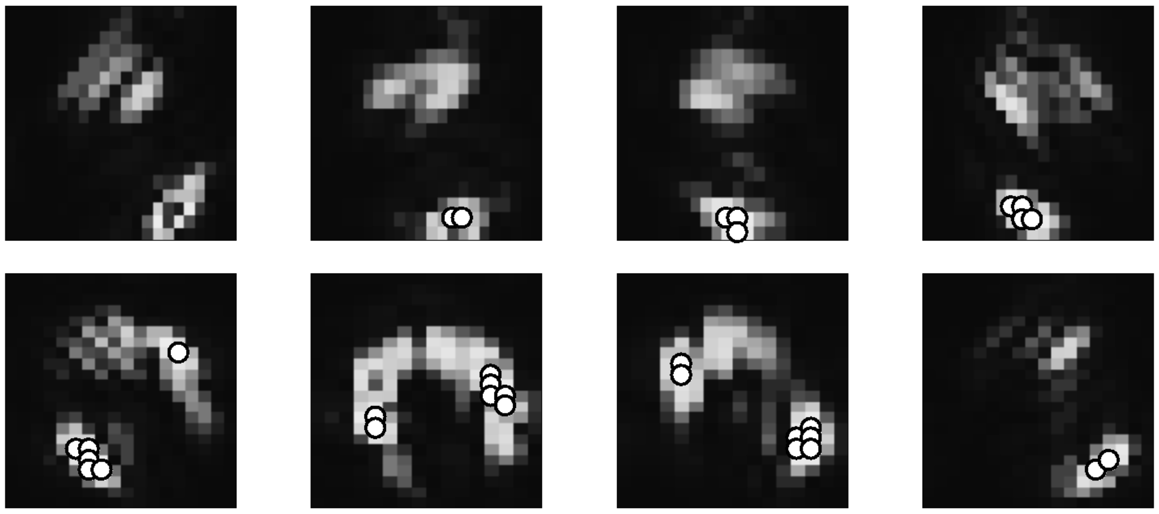

Figure 3.32 Selected coefficients (31) at scale level 2

Extract coordinate index for curvelet keypoints

We are able to use selected coefficients directly from above section. However, as we mentioned from previous work about SIFT algorithm, coordinate information of features in spatial domain can impact for training neural network significantly. So, we extract coordinate information using selected coefficients at step 2. Basically, a single point in the image can have multiple coefficients at different sub-bands as Figure 3.33. We are able to exploit this characteristic oppositely. In other words, a single coefficient at a sub-band has its matching coordinate information in the image. Some coefficients might point to the same spatial coordinates but we only take 100 coefficients which is very small portion compare to the number of all coefficients. Thus, it has low chance for multiple coefficients to point the same spatial coordinates.

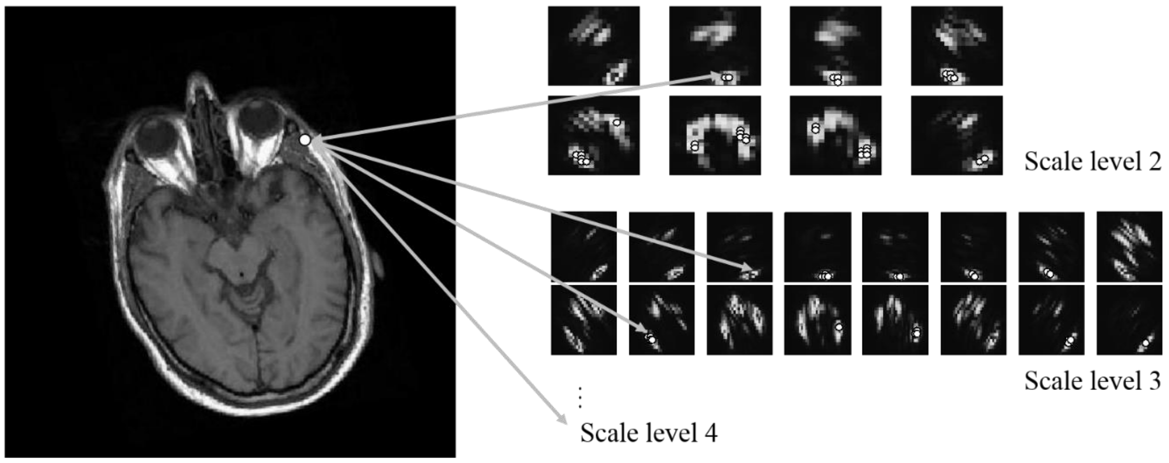

Figure 3.33 Keypoints matching between spatial domain and coefficients of sub-bands

The result of curvelet keypoints extraction is shown at Figure 3.34. Yellow markers are keypoints from scale level 2. Red, green, and magenta represents keypoints from scale level 3, 4, and 5 respectively. If we compare to keypoints from SIFT algorithm, curvelet keypoints represent the curves and directional features rather than keypoints of SIFT (Figure 3.13).

In addition, 100 curvelet keypoints via our algorithm has much more information than just using a general threshold method. The right image of Figure 3.35 is the reconstructed image

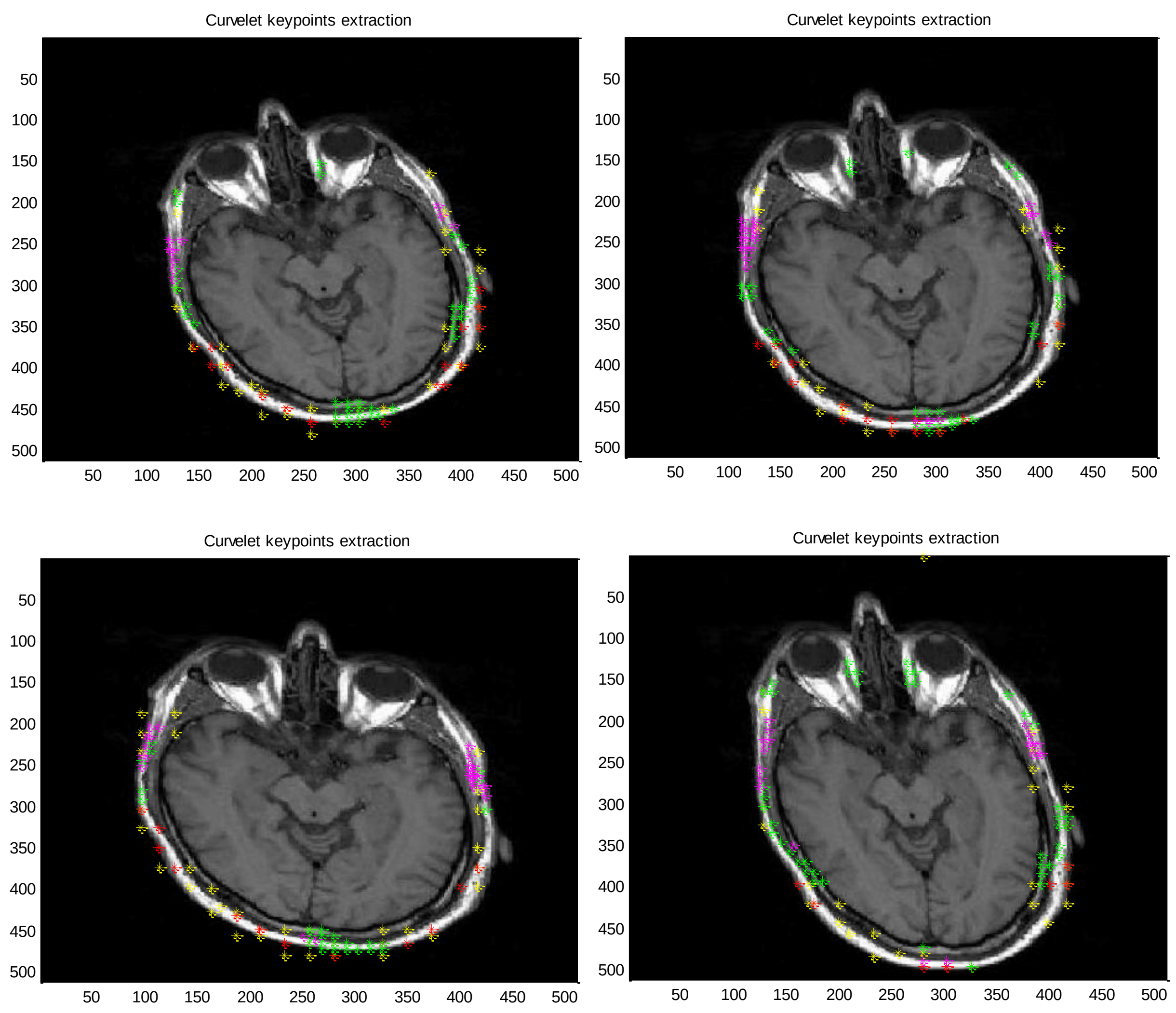
Curvelet keypoints extraction
Curvelet keypoints extraction
Curvelet keypoints extraction
Curvelet keypoints extraction

Figure 3.34 Curvelet keypoints extraction examples

using 100 coefficients via suggested algorithm. If we compare it to the reconstructed image from in Figure 3.28, our algorithm can represent the curve and line features much better than thresholding method. We also can see that it has lots of directional information which expect a good performance for rotation and translation parameters in training neural network.

In chapter 4, we're going to train neural network using DCT coefficients, keypoints from SIFT algorithm, wavelet coefficients, and curvelet keypoints we suggested in this chapter. We compare the results and which one has the best performance in terms of computational cost and accuracy.

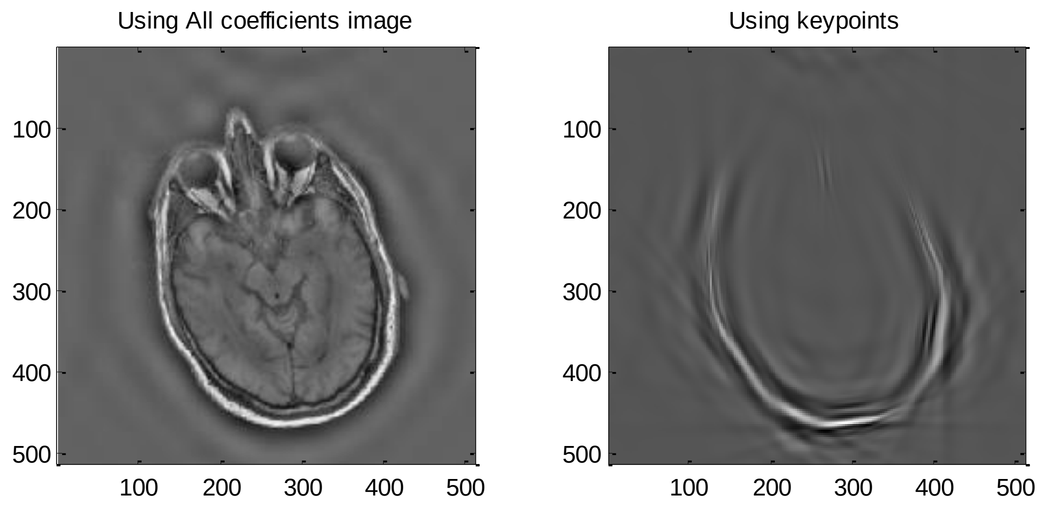

Figure 3.35 Reconstructed image using suggested algorithm

Chapter 4. Simulation

In this chapter, we simulate the image registration based on the algorithm suggested in chapter 3. Simulation are performed in two stages. First, the pre-registration stage is to synthesize original image to make training sets using affine transformations. Then, features such as DCT coefficients, SIFT keypoints, wavelet coefficients, or curvelet keypoints are extracted from the affined image. These inputs, normalized with zscore method, are used to train the network. In registration phase, which is a testing stage, features are extracted from registered image and feed to the trained network. The overall process is shown at Figure 4.1.

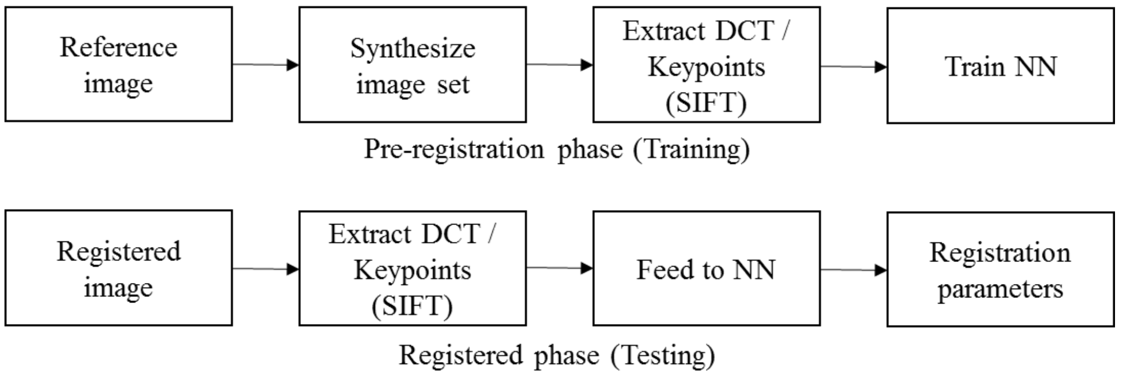
Reference image
Synthesize image set
Extract DCT / Keypoints (SIFT)
Train NN
Pre-registration phase (Training)
Registered image
Extract DCT / Keypoints (SIFT)
Feed to NN
Registration parameters
Registered phase (Testing)

Figure 4.1 Simulation process

The neural network we used for the simulation is general artificial neural network and wavelet neural network we mentioned in chapter 3. However, the performance of two structures is not significantly different, so most simulations are only performed on artificial neural network. Root Mean Square Error (RMSE) between parameters of target images and outputs of trained networks is compared to measure the performance.

4.1. Neural network structure

Before training the neural network, it is important to determine the structure of neural network. Coefficients from DCT, wavelet, or keypoints from SIFT and curvelet transform could be the inputs to train the network. The outputs of the network are the transformation parameters such as rotation, scale, translation, and shear. The general feedforward network uses sigmoid function as an activation function in hidden layers and linear function in output layer. The network is trained using 500 source-target image pairs. Among 500 image pairs, 50 pairs of images are used to validate the network that is performed at every 5 iteration during training. Another 50 image pairs then generated to test the network. Backpropagation algorithm is used to train the networks until the desired error is below 0.00001 or the maximum iteration reaches 15000. Learning rate (η) is set to 0.0001 which is considered appropriate value that doesn't make the error oscillate and the network fall into the local minimum. A small learning rate means the network will change at a slower rate and will take longer to converge while a large learning rate makes the network change faster but invoke oscillation to converge. A large learning rate might not converge at all as it could lead to 'over-shooting' the solution.

In order to determine the number of hidden layers and neurons of each layer, several simulations are performed with various numbers of layers and the number of neurons. Table 4-7 shows the result of training error (RMSE) with various neural network structures. A hundred DCT coefficients are used for this simulation. The performance of training error with two or three hidden layers is better than the structure of single hidden layer or four hidden layers. However, the testing errors of the trained network are similar each other. Therefore, the size of neural network used in the simulation is a single hidden layer or three hidden layers that have 64, 40, and 15 neurons at each layer depending on the simulations. The training error graph is shown at Figure 4.2. It shows that the network with two or three hidden layers is better than a single layer network.

Wavelet neural network is also used in this study. As mentioned in chapter 3, the difference between general artificial neural network and wavelet neural network is the activation function as an output of each neuron. General feedforward neural network has the same sigmoid function for all neurons, on the other hand, wavelet function has different dilation and translation parameters. So, wavelet neural network can be trained more adaptively than general neural network. Table 4-1 and Figure 4.2 shows that wavelet neural network has slightly better performance in some parameters than general feedforward network but it’s hard to measure that wavelet neural network is better. Thus, we only used a general artificial neural network for simulations.

Table 4-1 Neural network training error with various hidden layers and neurons

	Training error				
	ANN				WNN
RMSE	[64]	[64 40]	[64 40 15]	[64 40 15 9]	[64]
Rotation	1.0349	0.2702	0.1693	1.1700	0.8709
Scale(X-Axis)	0.0520	0.0523	0.0562	0.0580	0.0543
Scale(Y-Axis)	0.0582	0.0561	0.0592	0.0601	0.0581
Translation(X-Axis)	0.2987	0.2210	0.1892	2.5129	0.3039
Translation(Y-Axis)	0.3061	0.2304	0.1675	1.8517	0.3202
Shear(X-Axis)	0.0194	0.0149	0.0269	0.0289	0.0164
Shear(Y-Axis)	0.0191	0.0127	0.0235	0.0294	0.0186

Table 4-2 Neural network testing error with various hidden layers and neurons

	Testing error				
	ANN				WNN
RMSE	[64]	[64 40]	[64 40 15]	[64 40 15 9]	[64]
Rotation	1.1961	1.2005	1.2392	1.2096	1.2081
Scale(X-Axis)	0.0515	0.0555	0.0513	0.0501	0.0529
Scale(Y-Axis)	0.0607	0.0584	0.0579	0.0571	0.0582
Translation(X-Axis)	0.3691	0.3526	0.3714	3.2338	0.4076
Translation(Y-Axis)	0.3681	0.4125	0.4199	1.2110	0.4593
Shear(X-Axis)	0.0242	0.0345	0.0328	0.0319	0.0221
Shear(Y-Axis)	0.0232	0.0286	0.0283	0.0281	0.0240

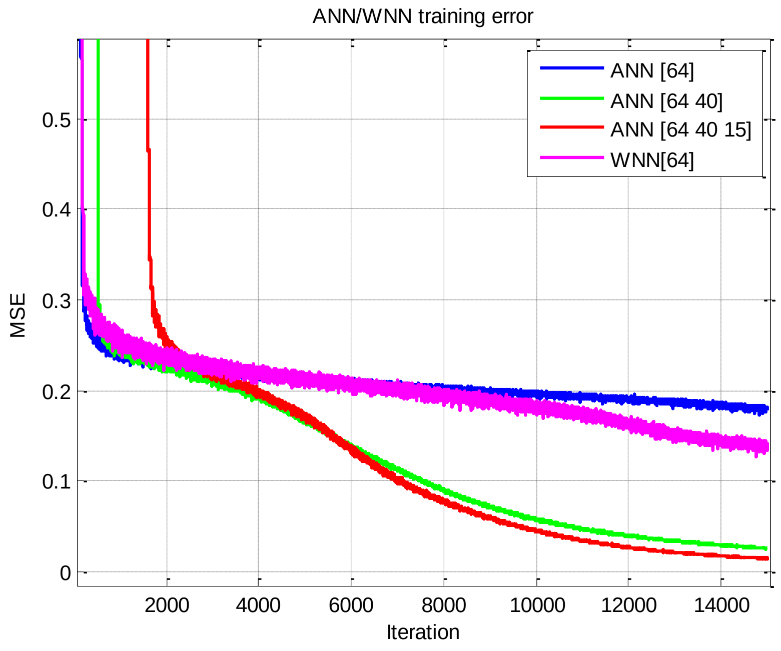

Figure 4.2 Training error of ANN/WNN according to the structures

4.2. Simulation with single transformation parameter

At this section, before applying all affine transformations together, images that are transformed with a single parameter is used to train the network: rotation, scale, translation, and shear respectively. The performance of three cases, DCT coefficients, DCT with SIFT keypoints, and curvelet keypoints that we suggested in chapter 3, are compared.

4.2.1. Rotation

Table 4-3 shows the result of three simulation cases estimating rotation parameter. The threshold for convergence is 10e^{-6}. Curvelet keypoints has the best performance in estimating rotation parameters in terms of accuracy and computational cost. It is straightforward because curvelet keypoints that we suggested in chapter 3 has strong characteristic of directional selectivity compared to DCT and SIFT keypoints features. From Figure 4.3 to Figure 4.5 shows the error graph for each case. Training, validation, and testing error are represented by asterisk, triangle, and square marker respectively. Curvelet keypoints case even converged at 123 iteration. Thus, we recognize that curvelet keypoints is the most effective features to estimate rotation parameter.

Table 4-3 Training and Testing error of rotation applied images

	DCT		DCT + SIFT		Curvelet keypoints	
	Train	Test	Train	Test	Train	Test
Rotation	0.0377	0.0287	0.0153	0.0176	0.0031	0.0003
Total inputs	100		200		200	
Training time	162 sec		209 sec		26 sec	
Iteration	1000 (no converge)		1000 (no converge)		123 (converge)	

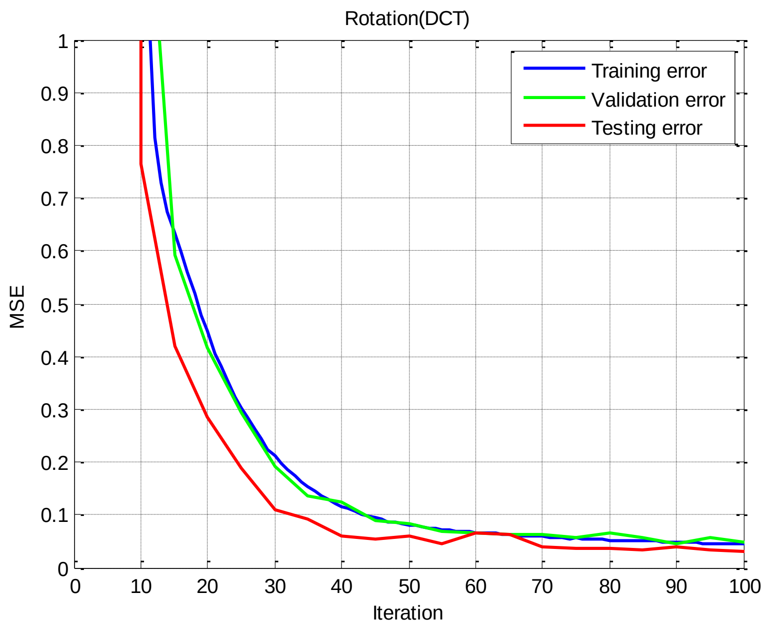

Figure 4.3 Rotation simulation with DCT coefficients

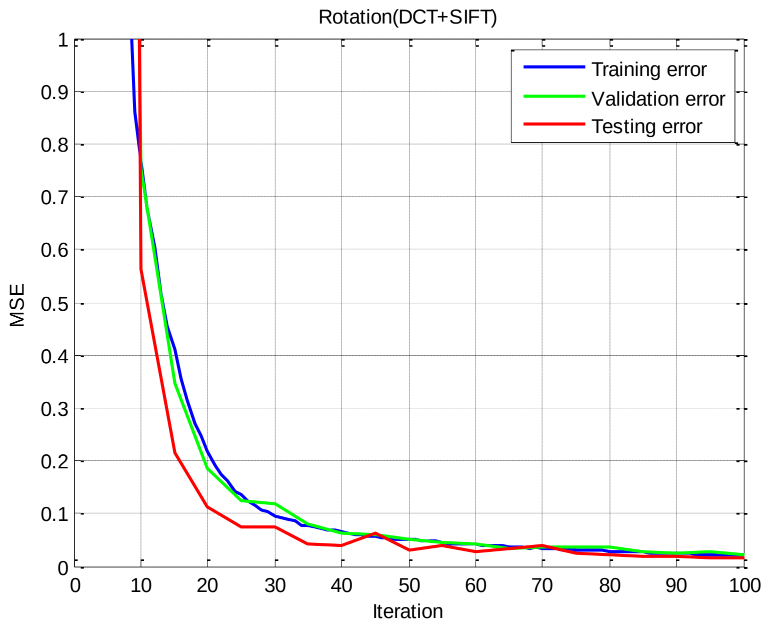

Figure 4.4 Rotation simulation with DCT coefficients and SIFT keypoints

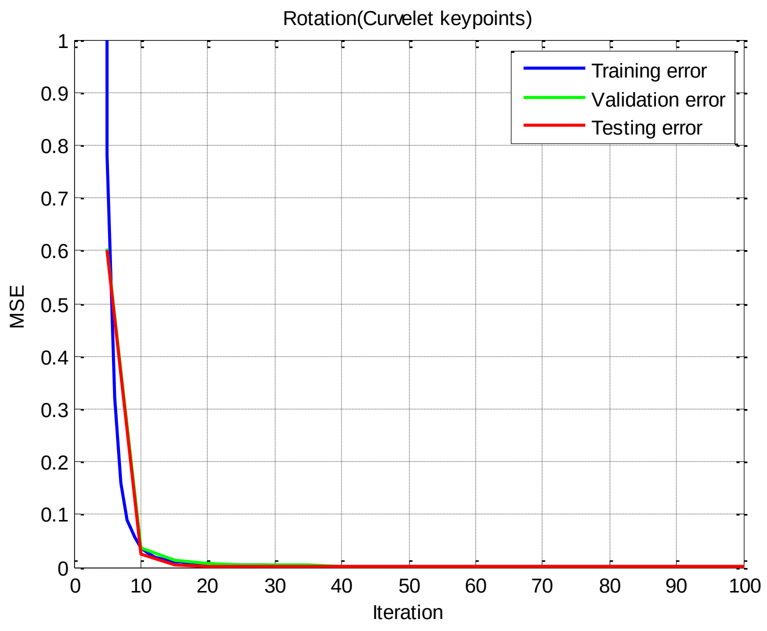

Figure 4.5 Rotation simulation with curvelet keypoints

4.2.2. Scaling

The range of scale parameter is 0.9 to 1.1 for both x-axis and y-axis. Table 4-4 shows the result. As the same with rotation simulation, curvelet keypoints has the best performance among the three cases. Curvelet transform is one of multi-resolution analysis methods, so one of its characteristics is invariant to scales. Figure 4.6, Figure 4.7, and Figure 4.8 shows the mean square error plot during training, validation, and testing.

Table 4-4 Training and Testing error of scale applied images

	DCT		DCT + SIFT		Curvelet keypoints	
	Train	Test	Train	Test	Train	Test
Scale X	0.0562	0.0506	0.0560	0.0507	0.0236	0.0362
Scale Y	0.0578	0.0539	0.0572	0.0535	0.0291	0.0386
Total inputs	100		200		200	
Training time	163 sec		208 sec		206 sec	
Iteration	1000 (no converge)		1000 (no converge)		1000 (no converge)	

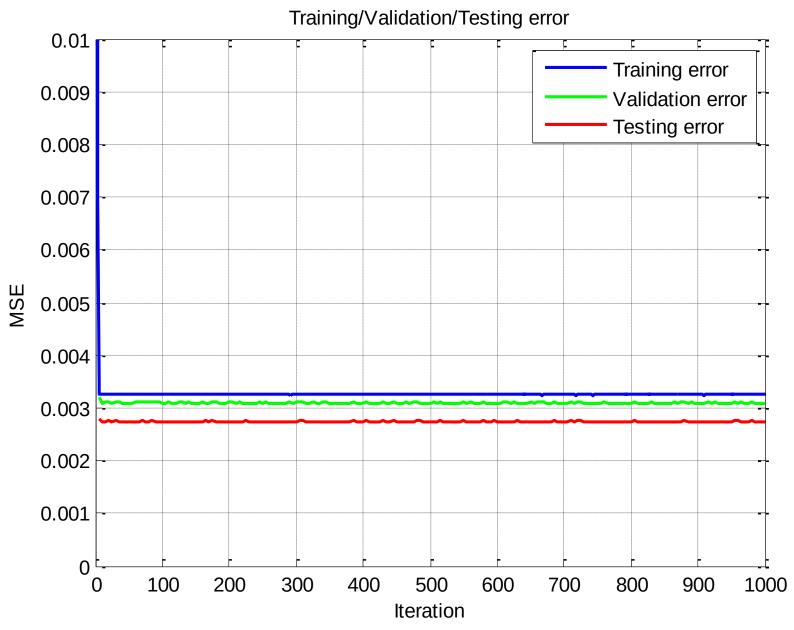

Figure 4.6 Scale simulation with DCT coefficients

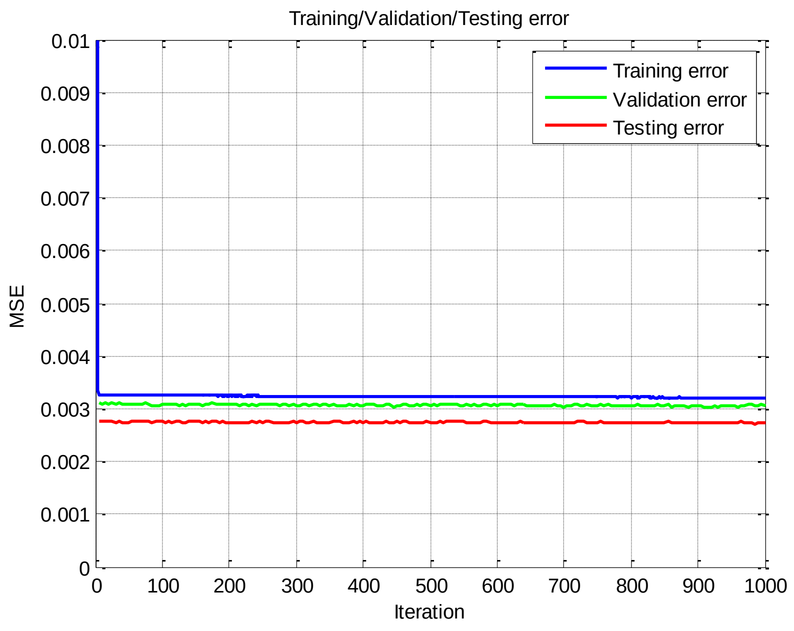

Figure 4.7 Scale simulation with DCT coefficients and SIFT keypoints

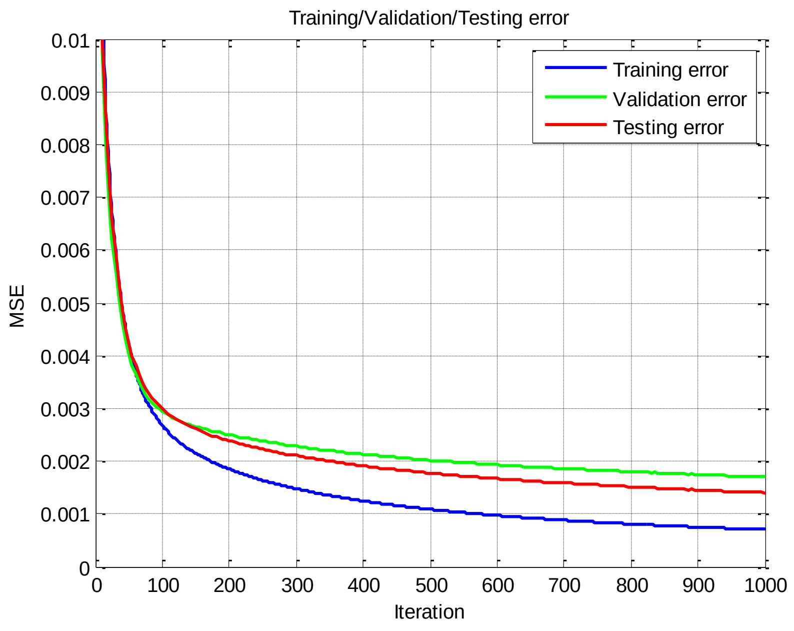

Figure 4.8 Scale simulation with curvelet keypoints

4.2.3. Translation

The range of translation parameter is -5 to 5 for both x-axis and y-axis. The performance of translation parameters is the best when the training inputs are DCT coefficients and DCT with SIFT keypoints. As we noticed at Figure 3.17 and Figure 3.18, curvelet transform has strong features to extract directional information from the image and it exploited a polar coordinate instead of using rectangular coordinate in the process of curvelet transform. Table 4-5 shows

Table 4-5 Training and Testing error of translation applied images

	DCT		DCT + SIFT		Curvelet Keypoints	
	Train	Test	Train	Test	Train	Test
Translation X	0.0443	0.0439	0.0389	0.0401	0.0686	0.2108
Translation Y	0.0457	0.0410	0.0406	0.0389	0.0315	0.4396
Total inputs	100		200		200	
Training time	415 sec		207 sec		208 sec	
Iteration	1000 (no converge)		1000 (no converge)		1000 (no converge)	

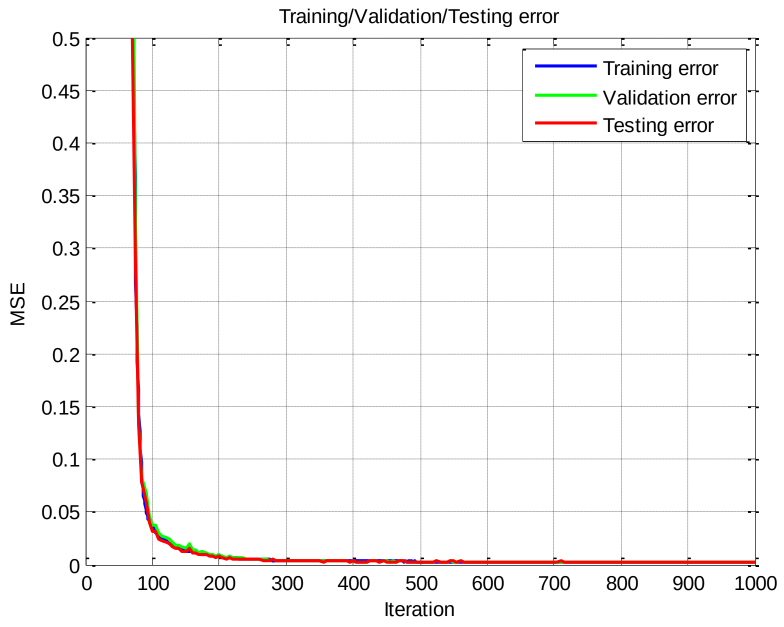

Figure 4.9 Translation simulation with DCT coefficients

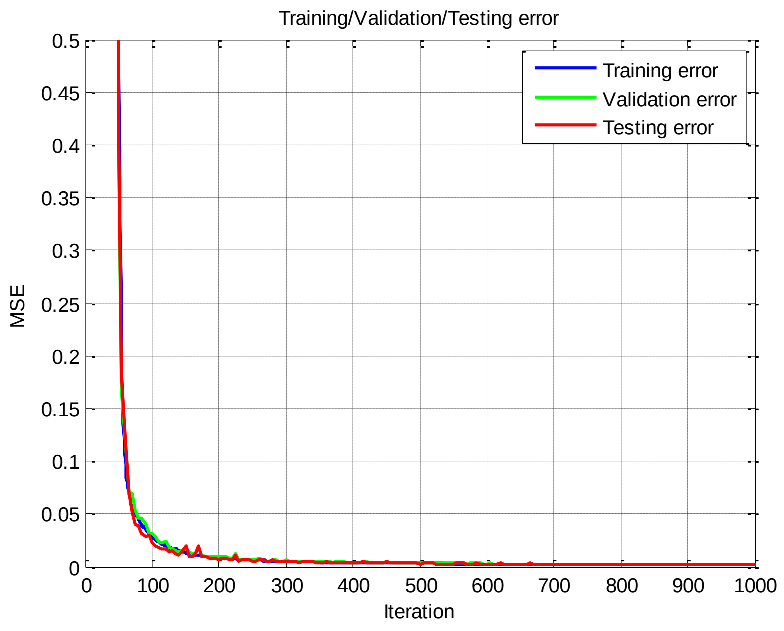

Figure 4.10 Translation simulation with DCT coefficients and SIFT keypoints

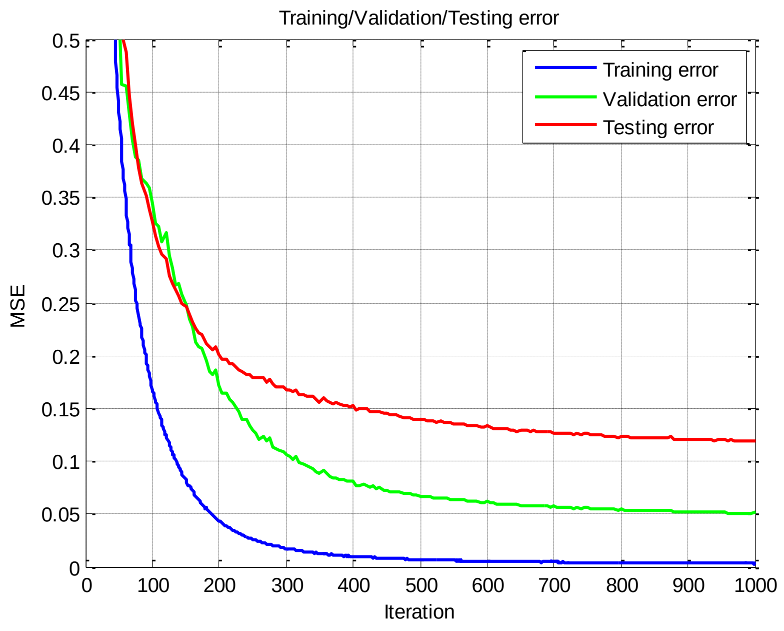

Figure 4.11 Translation simulation with curvelet keypoints

the result. The MSE for training is shown at Figure 4.9, Figure 4.10, and Figure 4.11

4.2.4. Shear

The range of shear parameter used in the simulation is from 0 to 0.1. DCT coefficients and DCT with SIFT keypoints cases are better than curvelet keypoints on shear transformation. We expected the curvelet keypoints have better performance due to its strong directional features. The performance of curvelet keypoints is slightly better than DCT coefficients case but worse than DCT with SIFT keypoints case. We can see that the SIFT keypoints can have a decisive effect on estimating shear parameter. The result is shown at Table 4-6. The MSE for training and testing is shown at Figure 4.12, Figure 4.13, and Figure 4.14.

Table 4-6 Training and Testing error of shear applied images

	DCT		DCT + SIFT		Curvelet Keypoints	
	Train	Test	Train	Test	Train	Test
Shear X	0.0136	0.0121	0.0073	0.0068	0.0121	0.0113
Shear Y	0.0155	0.0143	0.0076	0.0072	0.0100	0.0133
Total inputs	100		200		200	
Training time	167 sec		203 sec		204 sec	
Iteration	1000 (no converge)		1000 (no converge)		1000 (no converge)	

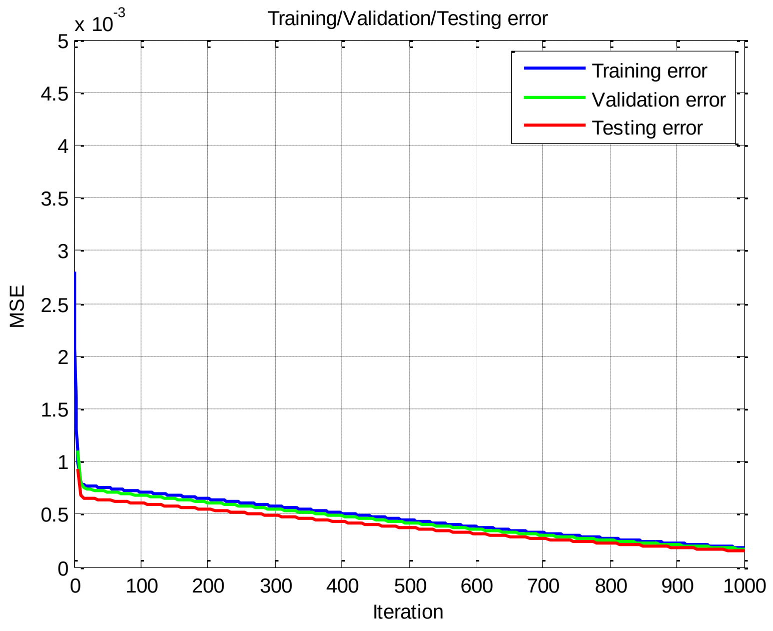

Figure 4.12 Shear simulation with DCT coefficients

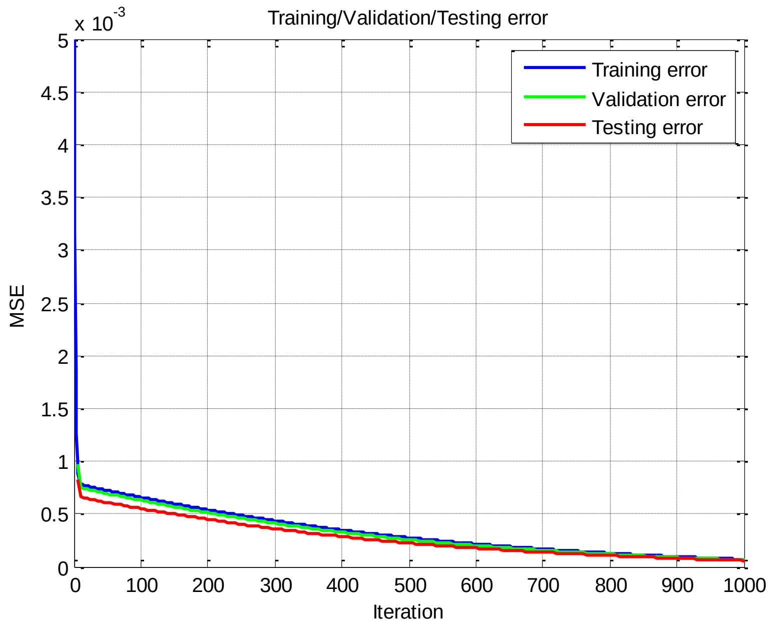

Figure 4.13 Shear simulation with DCT coefficients and SIFT keypoints

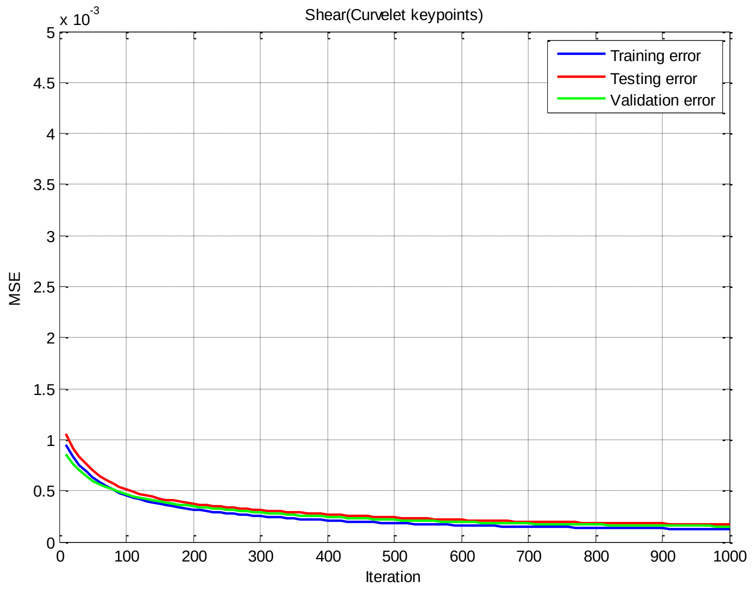

Figure 4.14 Shear simulation with curvelet keypoints

4.3. Simulation with multiple transformation parameters

In this section, simulations are performed using images that several affine transformation parameters are applied together.

4.3.1. DCT coefficients and SIFT keypoints

The simulation starts with the case of a hundred DCT coefficients. The case of a hundred DCT coefficients with fifty pairs of keypoints from SIFT algorithm is also simulated at this section. Keypoints are coordinate information from the image which is applied affine transformations. Fifty pairs of keypoints means a hundred inputs including x, and y coordinate information. Thus, totally two hundred inputs are used for training/testing and there are seven output parameters. Table 4-7 shows the result of using DCT coefficients only and DCT coefficients with SIFT keypoints. The overall training performance of DCT with SIFT keypoints is better than DCT only case. Especially, training of rotation and translation parameters are trained well comparing to other parameters. This does make sense because the features from SIFT keypoints are to be invariant to rotation, scale, and translation. However, the testing errors are similar for both cases. If we compare the error graph in detail in order to analyze the difference between the training and the testing error, it seems to be a generalization issue on DCT with SIFT keypoints case [Figure 4.15, and Figure 4.16]. Generally, there could be occurred a generalization problem when the network is over-trained or the complexity of the system is higher than the inputs. If it is happened due to over-training, there are several methods to avoid over-training issues of neural network. One of the simplest ways is to stop training early before the testing error will be increased. So, we can use trained weights and bias at the point the testing error has the minimum. The testing error at Table 4-7 is measured by applying an early stop method. However, the testing error does not follow the training error exactly. We'll investigate this problem more detail after performing curvelet keypoints simulation.

Table 4-7 Training and testing error with DCT and DCT with SIFT

	DCT		DCT + SIFT	
RMSE	Training	Testing	Training	Testing
Rotation	1.1594	1.2282	1.0646	1.2433
Scale(X-Axis)	0.0576	0.0541	0.0569	0.0527
Scale(Y-Axis)	0.0545	0.0520	0.0542	0.0537
Translation(X-Axis)	0.3320	0.3560	0.3013	0.3757
Translation(Y-Axis)	0.3373	0.3178	0.3168	0.3394
Shear(X-Axis)	0.0211	0.0225	0.0216	0.0258
Shear(Y-Axis)	0.0216	0.0221	0.0202	0.0264

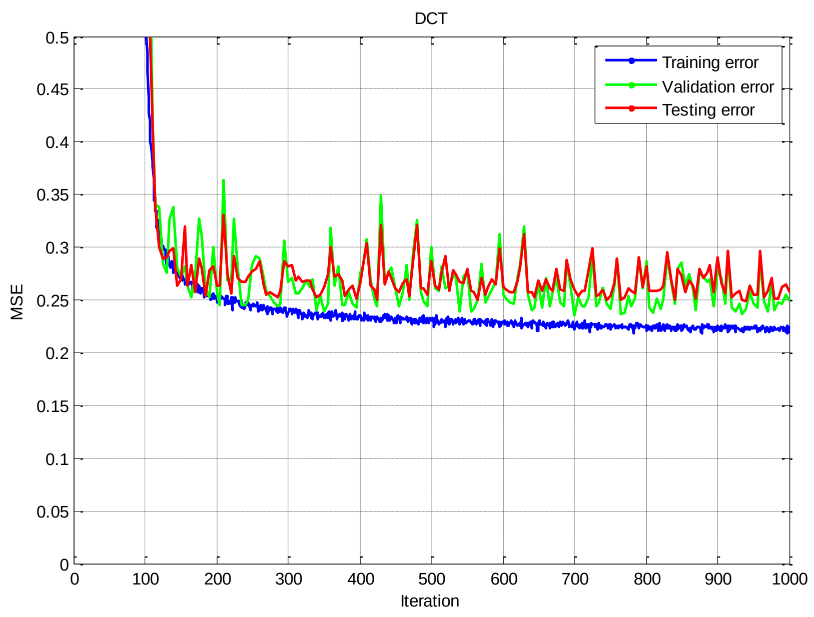

Figure 4.15 Difference between training and testing of DCT

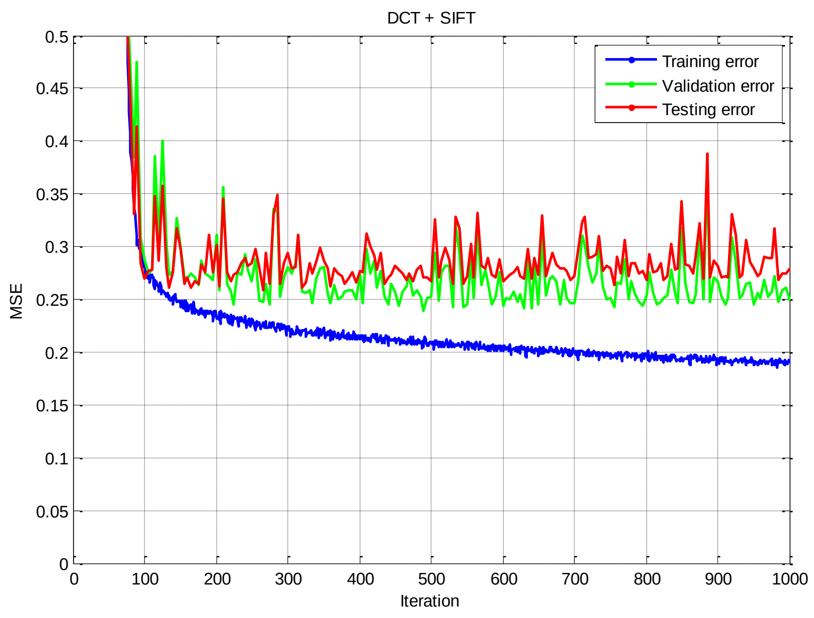

Figure 4.16 Difference between training and testing of DCT + SIFT

4.3.2. Curvelet coefficients

We mentioned that how to take a hundred curvelet coefficients from the scale level 2 to scale level 5 using a modified threshold method. We used curvelet coefficients as inputs to training the neural network. The training error of rotation and scale is much better than DCT coefficients and DCT with SIFT keypoints, but the testing error much worse than the others.

Table 4-8 Training and testing error of curvelet coefficients

	Curvelet coefficients	
RMSE	Training	Testing
Rotation	0.2655	3.1592
Scale(X-Axis)	0.0487	0.0639
Scale(Y-Axis)	0.0453	0.0548
Translation(X-Axis)	0.3797	3.1193
Translation(Y-Axis)	0.3958	3.4386
Shear(X-Axis)	0.0259	0.0351
Shear(Y-Axis)	0.0281	0.0314

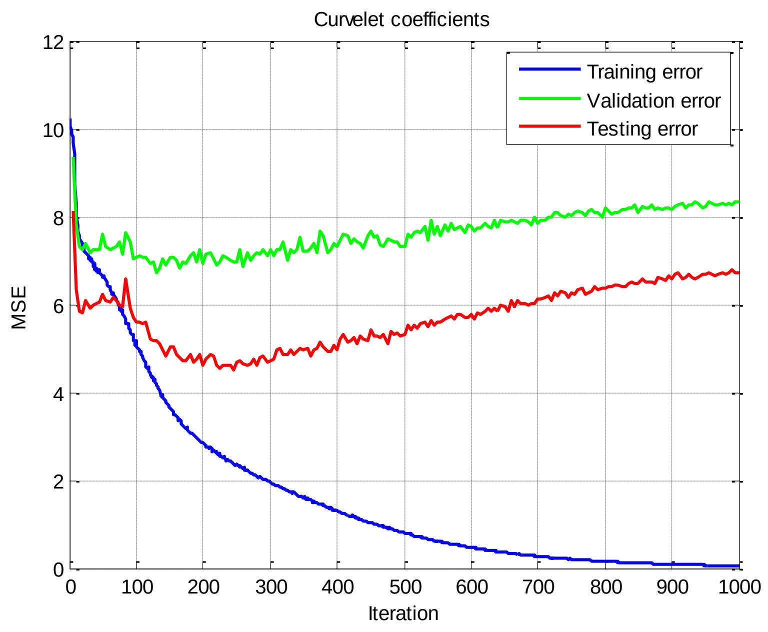

Figure 4.17 Difference between training and testing of curvelet coefficients

If we see the Figure 4.17 the testing error does not follow the training error. The generalization capability is much worse than the DCT coefficients with SIFT keypoints case.

4.3.3. Curvelet keypoints

Curvelet keypoints are coordinate information of the images and used for training neural network as inputs. As we mentioned at chapter 3, a hundred keypoints pairs are extracted from curvelet transformation which means two hundred inputs including x-axis and y-axis coordinate information. Those keypoints are the position information of curvelet coefficients as we used at the above section. The training result is at Table 4-9. Training error is significantly better than the DCT coefficients and DCT with SIFT keypoints simulations same as we noticed in a single parameter estimation. However, the testing error is much worse than the others, which seems to be the same generalization issue with curvelet coefficients simulation. Figure 4.18 shows a huge difference between training and testing error. So, now we’ll look over about the generalization issue of neural network more detail.

Table 4-9 Training and testing error of curvelet keypoints

	Curvelet keypoints	
RMSE	Training	Testing
Rotation	0.00095	3.7093
Scale(X-Axis)	0.0364	0.1249
Scale(Y-Axis)	0.0368	0.1539
Translation(X-Axis)	0.00098	3.1551
Translation(Y-Axis)	0.00099	2.9657
Shear(X-Axis)	0.0243	0.0313
Shear(Y-Axis)	0.0270	0.0317

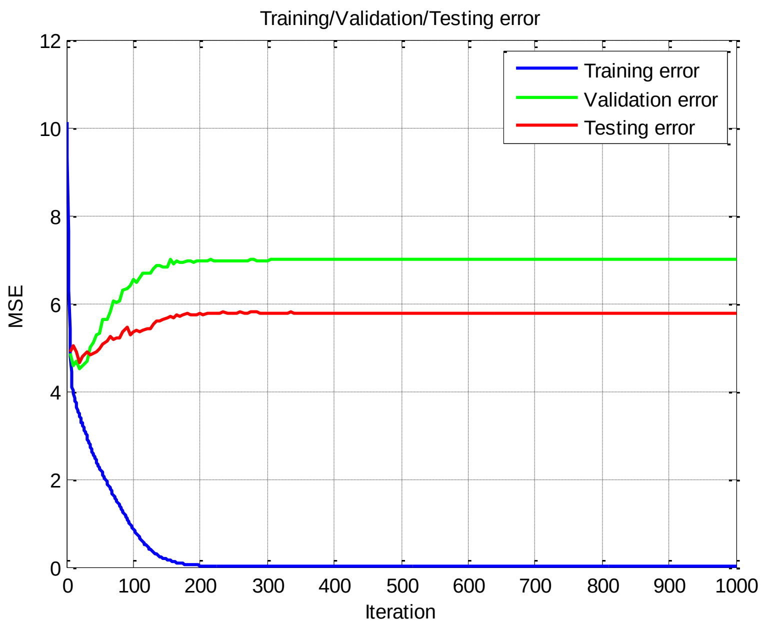

Figure 4.18 Difference between training and testing of curvelet keypoints

4.3.4. The generalization issue of neural network

The performance of artificial neural networks is mostly depends upon its generalization capability. Generalization of the neural network is ability to handle unseen data. The generalization capability of the network is usually determined by system complexity and training of the network. Poor generalization is observed when the network is over-trained or system complexity (or degree of freedom) is relatively more than the training data. There exist several approaches to avoid over-training of neural network such as an early stop training, cross-validation, or weight decaying methods. We tried to apply an early stop training approach but it does not improve the capability of generalization. When the system complexity is high than the training data, pruning method is able to reduce the system complexity. Reducing the number of neurons of hidden layers could have the similar effects of pruning approach to the network. Even though we used 32 neurons, which is a half of network structure, it did not solve the generalization issue. Now, we looked into the input data that are DCT coefficients, SIFT keypoints, and curvelet keypoints. Figure 4.19 shows the 20 input data sets for DCT with SIFT keypoints (left) and curvelet keypoints (right). The range from 0 to 100 in x-axis at left plot of Figure 4.19 represents DCT coefficients and the data from index 101 to 200 in x-axis represents coordinate information of SIFT keypoints. The scale of DCT coefficients is relatively larger than curvelet keypoints.

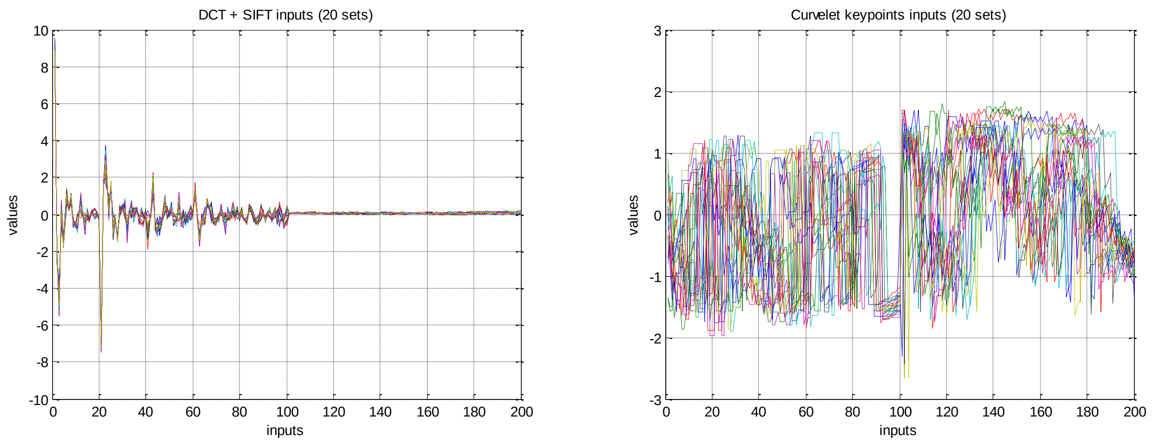

Figure 4.19 Input data sequences of DCT with SIFT and curvelet keypoints

From Figure 3.29, we can also guess that DCT coefficients can represent the overall information of the image while curvelet coefficients and keypoints can express details of the image such as curves and lines. Therefore, we may think that DCT coefficients could play an important role to generalize the unseen data. In Figure 4.19, even though the transformation parameters of 20 sets of images are different, the DCT coefficients seem to have a consistent pattern while the pattern of curvelet keypoints seem to be hard to predict. In order to increase generalization capability, we might need to add some kinds of overall information of the image for training the neural network.

4.3.5. Using DCT and curvelet coefficients

At this section, we tried to use DCT coefficients and curvelet coefficients together. As we mentioned above, we might need overall information of the image in order to solve generalization issue that was occurred in curvelet coefficients and curvelet keypoints simulation.

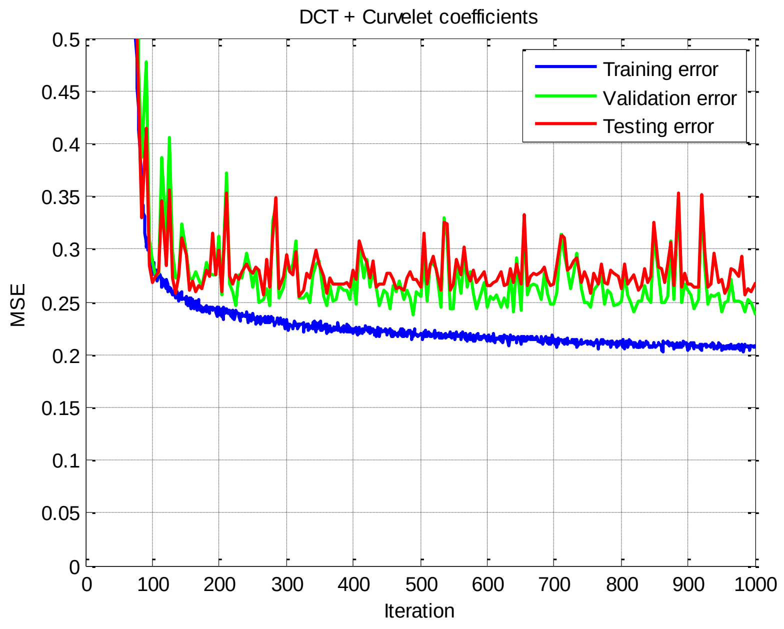

Figure 4.20 DCT with curvelet coefficients simulation

Table 4-10 Training and testing error of DCT with curvelet coefficients

	DCT + Curvelet coefficients	
RMSE	Training	Testing
Rotation	1.1115	1.2322
Scale(X-Axis)	0.0572	0.0519
Scale(Y-Axis)	0.0531	0.0537
Translation(X-Axis)	0.3262	0.3697
Translation(Y-Axis)	0.3260	0.3572
Shear(X-Axis)	0.0211	0.0252
Shear(Y-Axis)	0.0210	0.0258

When we used DCT coefficients of the image, it improves the generalization capability as we expected. However, DCT coefficients effect on training the neural network dominantly. Thus, the training and testing results are similar with DCT coefficients case.

4.3.6. Using DCT and curvelet keypoints

We also use a hundred DCT coefficients of the image and a hundred curvelet keypoints to training the neural network. The results is shown at Table 4-11. DCT coefficients are also able to increase generalization capability of the neural network. We can see that curvelet keypoints are more effective to train the neural network. Even though DCT coefficients played dominantly by reducing generalization error, we can see training error is better than the others. The testing error is also slightly better than the others in all parameters.

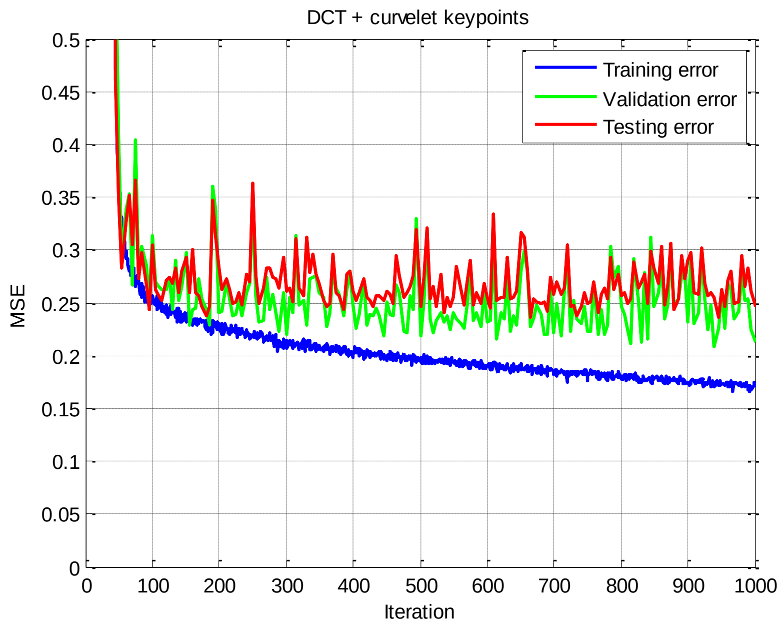

Figure 4.21 Training and testing error of DCT with curvelet keypoints

Table 4-11 Training and testing error of DCT with curvelet keypoints

	DCT + Curvelet keypoints	
RMSE	Training	Testing
Rotation	0.9690	1.1909
Scale(X-Axis)	0.0519	0.0502
Scale(Y-Axis)	0.0534	0.0515
Translation(X-Axis)	0.2654	0.3440
Translation(Y-Axis)	0.2734	0.3285
Shear(X-Axis)	0.0222	0.0227
Shear(Y-Axis)	0.0196	0.0215

4.3.7. Cross validation

We can increase generalization capability by adding DCT of the image for training the neural network at the above section. However, there still seems to be the over-training at Figure 4.21. As mentioned above, early stop training approach does not significantly improve the over-training. We applied a cross-validation method for training the network. First, we combine training and testing data set and mix it. Then the data sets are divided into 10 sub sets that each set has 50 inputs and outputs pairs. At the first round, we train sub sets from 2 to 10 and test with set number 1. We train sub sets 1 and from 3 to 10, and test with set number 2 at a second round of training. We continue to train the network using data set rotationally. After 5 round of training, we could get a better training result comparing without applying cross-validation. [Table 4-12]. In testing, it has significantly better performance to estimate rotation parameter.

Table 4-12 Cross validation of DCT with curvelet keypoints

	DCT + Curvelet keypoints after cross-validation	
RMSE	Training	Testing
Rotation	0.9401	0.8197
Scale(X-Axis)	0.0603	0.0836
Scale(Y-Axis)	0.0608	0.0791
Translation(X-Axis)	0.4828	0.7696
Translation(Y-Axis)	0.4837	0.6324
Shear(X-Axis)	0.0281	0.0303
Shear(Y-Axis)	0.0301	0.0295

4.4. Robustness to noisy images

At this section, we test neural network with noisy images instead of clear images. We add two types of noise: Gaussian noise, and salt and pepper noise [Figure 4.22]. For Gaussian noise, signal to noise ratio is described as Equation (4.1). 5dB, 2dB, and 1dB noisy images are tested in this simulation. Salt and pepper noise is added by based on percentage of image pixels and 2%, 5%, and 10% images are tested.

$$SNR = 10\log_{10}\frac{P_{signal}}{P_{noise}} = 10\log_{10}\frac{Var(image)}{Var(noise)} \qquad (4.1)$$

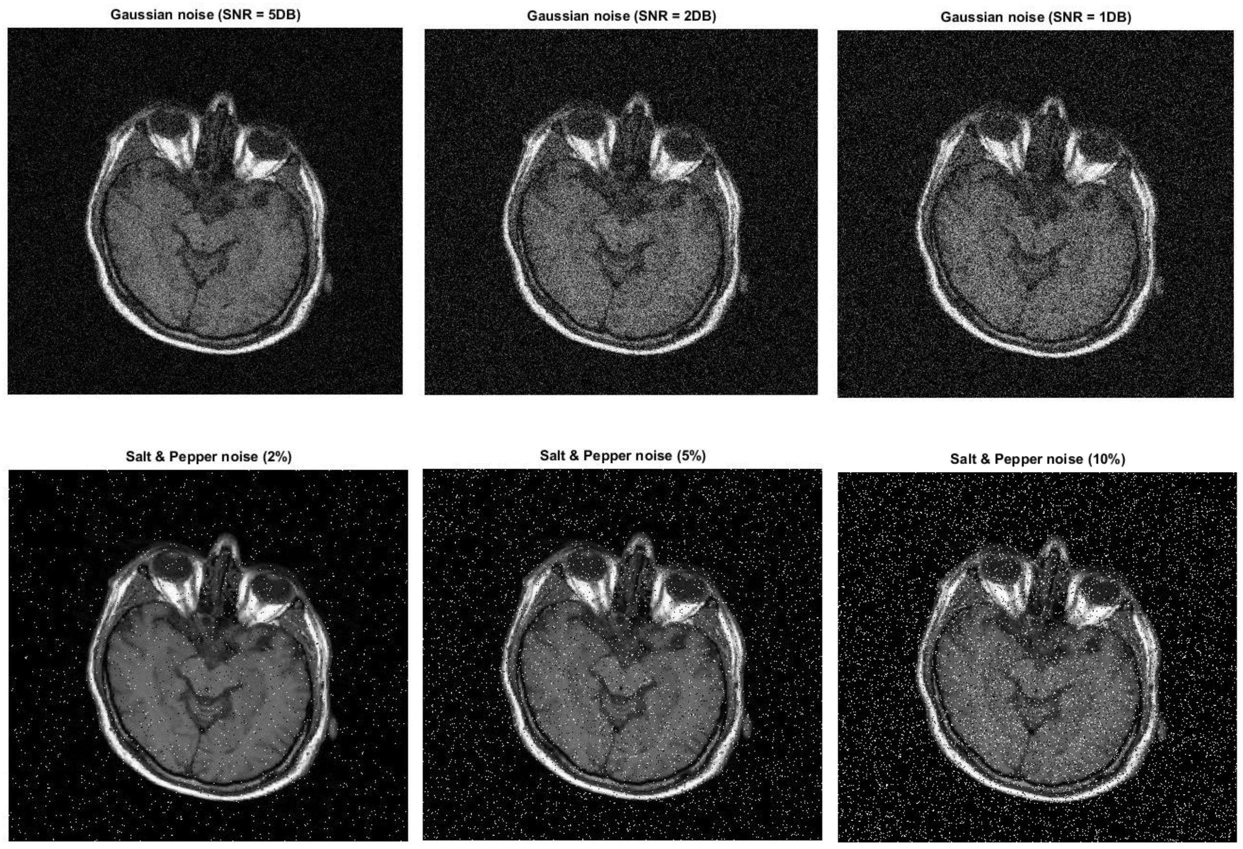
Gaussian noise (SNR = 5DB)
Gaussian noise (SNR = 2DB)
Gaussian noise (SNR = 1DB)
Salt & Pepper noise (2%)
Salt & Pepper noise (5%)
Salt & Pepper noise (10%)

Figure 4.22 Noisy images; Gaussian noise (top), Salt & Pepper noise (bottom)

Trained parameters at above sections are used for testing noisy images. We compare the results with DCT coefficients, DCT coefficients with SIFT keypoints, and DCT with curvelet keypoints cases.

4.4.1. DCT coefficients with noisy images

Table 4-13 and Table 4-14 show the results of simulation with noisy images. As the amount of noise is increasing, the performance becomes worse for both Gaussian and salt & pepper type noise. The rotation and translation parameters become worse than other parameters.

Table 4-13 Testing results with Gaussian noise images (DCT coefficients)

Gaussian Noise	w/o noise	Noise (5dB)	Noise (2dB)	Noise (1dB)
Rotation	1.2282	1.5937	1.9068	2.0620
Scale(X-Axis)	0.0541	0.0588	0.0627	0.0642
Scale(Y-Axis)	0.0520	0.0556	0.0594	0.0612
Translation(X-Axis)	0.3560	1.0515	1.3996	1.5609
Translation(Y-Axis)	0.3178	0.7274	0.7726	0.7824
Shear(X-Axis)	0.0225	0.0231	0.0231	0.0230
Shear(Y-Axis)	0.0221	0.0232	0.0239	0.0242

Table 4-14 Testing results with Salt & Pepper noise images (DCT coefficients)

	w/o noise	Noise (2%)	Noise (5%)	Noise (10%)
Rotation	1.2282	1.2920	1.4885	1.8839
Scale(X-Axis)	0.0541	0.0550	0.0561	0.0591
Scale(Y-Axis)	0.0520	0.0524	0.0531	0.0562
Translation(X-Axis)	0.3560	0.4976	0.8798	1.3400
Translation(Y-Axis)	0.3178	0.3338	0.4369	0.7345
Shear(X-Axis)	0.0225	0.0225	0.0229	0.0231
Shear(Y-Axis)	0.0221	0.0223	0.0225	0.0233

4.4.2. DCT coefficients and SIFT keypoints with noisy images

Table 4-15, and Table 4-16 show the results of DCT coefficients and SIFT keypoints. The overall performance is better than DCT coefficients only case[Table 4-13, Table 4-14]

Table 4-15 Testing results with Gaussian noise images (DCT + SIFT)

	w/o noise	Noise (5dB)	Noise (2dB)	Noise (1dB)
Rotation	1.2433	1.5335	1.8420	1.9813
Scale(X-Axis)	0.0527	0.0554	0.0587	0.0602
Scale(Y-Axis)	0.0537	0.0545	0.0564	0.0582
Translation(X-Axis)	0.3757	0.9252	1.2478	1.4120
Translation(Y-Axis)	0.3394	0.8677	0.9017	0.8987
Shear(X-Axis)	0.0258	0.0258	0.0258	0.0258
Shear(Y-Axis)	0.0264	0.0266	0.0267	0.0268

The estimation for translation parameters is worse than the other ones, which is the similar trend with DCT coefficients case.

Table 4-16 Testing results with Salt & Pepper noise images (DCT + SIFT)

	w/o noise	Noise (2%)	Noise (5%)	Noise (10%)
Rotation	1.2433	1.2450	1.4332	1.8133
Scale(X-Axis)	0.0527	0.0532	0.0541	0.0568
Scale(Y-Axis)	0.0537	0.0538	0.0531	0.0549
Translation(X-Axis)	0.3757	0.4657	0.7785	1.1808
Translation(Y-Axis)	0.3394	0.3693	0.4688	0.7634
Shear(X-Axis)	0.0258	0.0257	0.0258	0.0260
Shear(Y-Axis)	0.0264	0.0264	0.0265	0.0265

4.4.3. DCT coefficients and curvelet keypoints with noisy images

One of the applications of curvelet transform is to reduce noise of the image. However, in this simulation, curvelet transform does not improve the estimation performance so much. The result is slightly better when we exploit DCT coefficients and curvelet keypoints together [Table 4-11]. Table 4-17 and Table 4-18 show the estimation results using DCT and curvelet keypoints together. The testing error of noisy images does not show the improvement compared to DCT coefficients and DCT with SIFT keypoints, rather slightly worse in some parameters.

Table 4-17 Testing results with Gaussian noise images (DCT with curvelet keypoints)

	w/o noise	Noise (5dB)	Noise (2dB)	Noise (1dB)
Rotation	1.1909	1.5606	1.9340	2.1279
Scale(X-Axis)	0.0502	0.0556	0.0588	0.0600
Scale(Y-Axis)	0.0515	0.0554	0.0588	0.0610
Translation(X-Axis)	0.3440	1.0869	1.4571	1.6173
Translation(Y-Axis)	0.3285	0.7615	0.7892	0.8141
Shear(X-Axis)	0.0227	0.0223	0.0222	0.0224
Shear(Y-Axis)	0.0215	0.0224	0.0234	0.0236

Table 4-18 Testing results with Salt & Pepper noise images (DCT with curvelet keypoints)

	w/o noise	Noise (2%)	Noise (5%)	Noise (10%)
Rotation	1.1909	1.2060	1.4057	1.8441
Scale(X-Axis)	0.0502	0.0508	0.0527	0.0550
Scale(Y-Axis)	0.0515	0.0522	0.0534	0.0557
Translation(X-Axis)	0.3440	0.5074	0.8846	1.3553
Translation(Y-Axis)	0.3285	0.3501	0.4689	0.7598
Shear(X-Axis)	0.0227	0.0221	0.0224	0.0225
Shear(Y-Axis)	0.0215	0.0217	0.0217	0.0224

The reason we might guess is that if curvelet keypoints are extracted well from noisy image, it can estimate parameters better. However, it also has possible to get worse results, when the curvelet keypoints do not represent its features well as displayed in Figure 4.23.

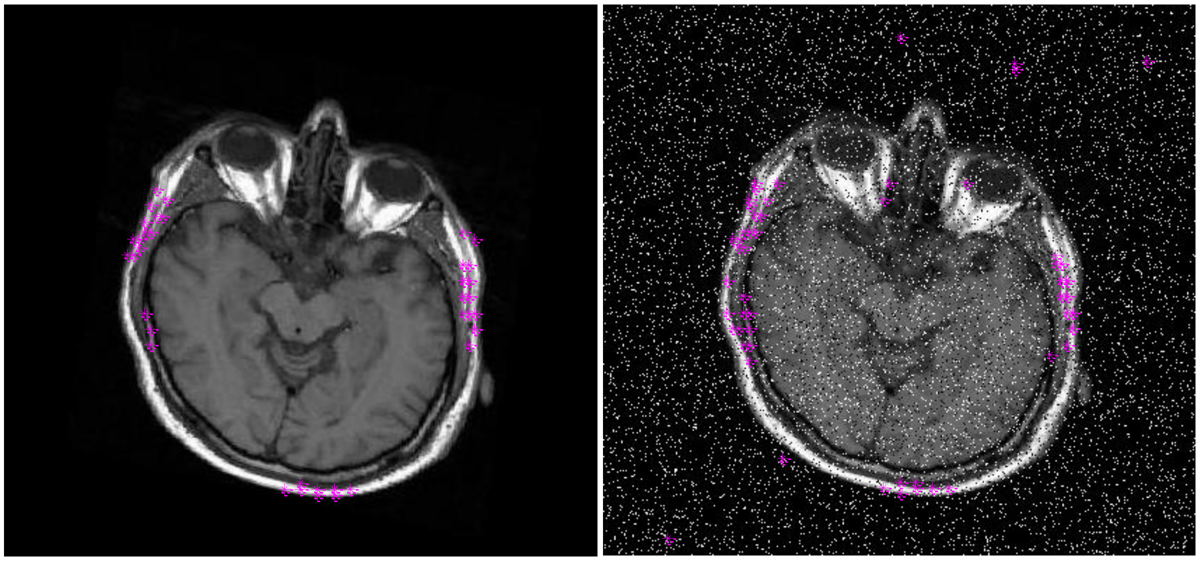

Figure 4.23 False curvelet keypoints of salt & pepper noisy image (10%)

Chapter 5. Conclusions and future work

In this thesis, neural networks are used to estimate the parameters needed for registration of brain images. The inputs to the networks are DCT coefficients and coordinate information. We suggested to use curvelet keypoints that are extracted by using curvelet transformation as coordinate information. Curvelet transform is a multiresolution geometric analysis that is able to exploit the geometric properties of structures better than wavelet transform due to its high directional sensitivity. As discussed in chapter 4, the training of neural network with curvelet keypoints approach performed significantly better than other methods on the estimation of rotation or scale parameters. The training and testing errors of ANN with DCT coefficients and curvelet keypoints are slightly better than other methods when all image registration parameters are applied.

For future work, we will apply this approach (DCT with curvelet keypoints) to more images to further test its performance.

BIBLIOGRAPHY

[1] Barbara Zitová, Jan Flusser, "Image Registration Methods: a survey," *Image and Vision Computing,* vol. 21, no. 11, pp. 977-1000, 2003.

[2] S. Damas, O. Cordon, J. Santamaria, "Medical Image Registration Using Evolutionary Computation: An Experimental Survey," *IEEE Computational Intelligence Magazine,* pp. 26-42, 2011.

[3] W. Rui, L. Minglu, "An Overview of Medical Image Registration," *Proceedings of the Fifth International Conference on Computational Intelligence and Multimedia Applications,* pp. 385-390, 2003.

[4] Medha V. Wyawahare, Dr. Pradeep M. Patil, and Hemant K. Abhyankar, "Image Registration Techniques: An overview," *International Journal of Signal Processing, Image Processing and Pattern Recognition,* vol. 2, 2009.

[5] A. Malviya, S.G. Bhirud, "Wavelet based Image Registration using Mutual Information," *International Conference on Emerging Trends in Electronic and Photonic Devices & Systems,* pp. 241-244, 2009.

[6] Y. Yamamura, "A Method for Image Registration by Maximization of Mutual Information," *SICE_ICASE International Join Conference,* pp. 1469-1472, 2006.

[7] W. Pratt, Digital Image Processing, Wiley, 1991.

[8] W. Pratt, "Correlation Techniques of Image Registration," *IEEE Transactions on Aerospace and Electronic Systems,* vol. 10, pp. 353-358, 1974.

[9] P. van Wie, M. Stein, "A Landsat Digital Image Rectification System," *IEEE Transactions on Geoscience Electronics,* vol. 15, pp. 130-136, 1977.

[10] P. Anuta, "Spatial Registration of Multispectral and Multitemporal Digital Imagery using Fast Fourier Transform," *IEEE Transactions on Geoscience Electronics,* vol. 8, pp. 353-368, 1970.

[11] R. Bracewell, The Fourier Transform and Its Applications, New York: McGraw-Hill, 1965.

[12] E.D. Castro, C. Morandi, "Registration of Translated and Rotated Images using Finite Fourier Transform," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 9, pp. 700-703, 1987.

[13] H.S. Alhichri, M. Kamel, "Virtual Circles: a new set of features for fast image registration," *Pattern Recognition Letters,* vol. 24, pp. 1181-1190, 2003.

[14] A. Noble, "Finding corners," *Image and Vision Computing,* vol. 6, pp. 121-128, 1988.

[15] S. Haykin, Neural Networks and Learning Machines, New York: Prentice Hall, 2009.

[16] I Elhanany, M Sheinfeld, A Beck, Y Kadmon, N Tal, D Tirosh, "Robust Image Registration Based on Feedforward Neural Networks," *IEEE International Conference on Systems, Man, and Cybernetics,* pp. 1507-1511, 2000.

[17] A.B. Abche, F. Yaacoub, A. Maalouf, E. Karam, "Image Registration based on Neural Network and Fourier Transform," *Proceedings of the 28th IEEE EMBS Annual International Conference,* pp. 4803-4806, 2006.

[18] J. Wu, J. Xie, "“Zernike Moment-based Image Registration Scheme Utilizing Feedforward Neural Networks," *Proceedings of the 5th World Congress on Intelligent Control and Automation,* pp. 4046-4048, 2004.

[19] P. Gadde, "Affine Image Registration Using Artificial Neural Networks," California Polytechnic State University, San Luis Obispo, 2013.

[20] J. Flusser, B. Zitova, "Combined Invariants to Linear Filtering and Rotation," *International Journal of Pattern Recognition and Artificial Intelligence,* vol. 13, pp. 1123-1136, 1999.

[21] "Magnetic Resonance –Technology Information Portal," 2012. [Online]. Available: http://www.mr-tip.com/serv1.php.

[22] D. G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," University of British Columbia, 2004.

[23] I. A. El Rube, M. A. Sharks, A. R. Salem, "Image Registration Based on Multi-Scale SIFT for Remote Sensing Images," *International Conference on Signal Processing and Communication Systems,* pp. 1-5, 2009.

[24] D. Veitch, "Wevelet Neural Networks and Their Application in The Study of Dynamical Systems," Department of Mathematics, University of York, 2005.

[25] Jianwei Ma, Gerlind Plonka, "The Curvelet Transform," *IEEE Signal Processing Magagine*, 2010.

[26] Emmanuel Cand`es, Laurent Demanet, David Donoho and Lexing Ying, "Fast Discrete Curvelet Transforms," *Applied and Computational Mathematics*, 2006.

[27] Alzubi, S., Islam, N., Abbod, M., "Multiresolution Analysis Using Wavelet, Ridgelet, and Curvelet Transforms for Medical Image Segmentation," *Int J Biomed Imaging,* p. 136034, 2011.

[28] Ankita Kaushal, Paramjeet Kaur, "Curvelet and Wavelet Image Fusion using Neural Network Algorithm," *International Journal of Computer Science and Information Technologies,* pp. 7508-7512, 2014.

[29] Bhawna Gupta, Shamik Tiwari, "Lung Cancer Detection using Curvelet Transform and Neural Network," *International Journal of Computer Applications,* pp. 15-17, 214.

[30] Khaled Abu Mahmoud, Adel Al-Jumaily, Maen Takruri, "Wavelet and Curvelet Analysis for Automatic Identification of Melanoma Based on Neural Network Classification," *International Journal of Computer Information Systems and Industrial Management,* vol. 5, pp. 606-614, 2013.

[31] Khalil al-saif, Nagham ajeel al-ajeely, "Normalized Cut Algorithm on Curvelet Coefficient for Digital Image Segmentation," *International Journal of Engineering and Innovative Technology,* vol. 3, no. 11, 2014.

[32] S. Elaiwat, M. Bennamoun, F. Boussaid, and A. El-Sallam, "3-D Face Recognition Using Curvelet Local Features," *IEEE Signal Processing Letters,* vol. 21, 2014.

[33] L. Y. Laurent Demanet, "Curvelets and Wave Atoms for Mirror-Extended Images," Stanford University, University of Texas at Austin, 2007.

[34] U. Qayyum, "Phase Efficient Neural Network using Curvelet Features for Face Recognition," *IEEE International Multitopic Conference,* 23 December 2008.

[35] A. Graps, "An Introduction to Wavelets," *IEEE Computational Science and Engineering,* vol. 2, 1995.

[36] Michel Misiti, Yves Misiti, Georges Oppenheim, Jean-Michel Poggi, "Wavelet Toolbox 4 user's guide," The MathWorks, Inc., 1997.

[37] Hana Hejazi, Mohammed Alhanjouri, "Face Recognition Using Curvelet Transform," pp. 388-396, 2010.

[38] Tanaya Guha, Q. M. Jonathan Wu, "Curvelet Based Feature Extraction," *Face Recognition,* 2010.

[39] P.-Y. Lin, "An introduction to Wavelet Transform," Graduate Institute of Communication Engineering, National Taiwan University, 2007.

[40] C.-L. Liu, "A Tutorial of Wavelet Transform," 2010.

[41] Mark Hudson Beale, Martin T. Hagan, Howard B. Demuth, "Neural Network Toolbox User Guide," 1992.

[42] L. Y. Laurent Demanet, "Curvelets, Wave Atoms, and Wave Equations," 2006.

[43] SHREEJA R, KHUSHALI DEULKAR, "Neuro Fuzzy Model for Face Recognition with Ccurvelet Based Feature Image," *International Journal of Engineering Science and Technology,* vol. 3, no. 6, pp. 5306-5312, 2011.

[44] Amol D. Rahulkar, Dattatraya V. Jadhav, Raghunath S. Holambe, "Fast Discrete Curvelet Transform Based Anisotropic Feature Extraction for Iris Recognition," *ICTACT Journal on Image and Video Processing,* pp. 69-75, 2010.

[45] Elaiwat, S., Bennamoun, M., Boussaid, F., El-Sallam, A., "A Curvelet-based Approach for Textured 3D Face Recognition," *Pattern Recognition,* pp. 1235-1246, 2015.

[46] N. Vujovic, D. Brzakovic, "Establishing The Correspondence Between Control Points," *IEEE Transactions on Image Processing*, vol. 6, pp. 1388-1399, 1997.

[47] Liu Zhaoying, Zhou Fugen, Bai Xiangzhi, Wang Hui, Tan Dongjie, "Multi-Modal Image Registration by Mutual Information Based on Optimal Region Selection," *International Conference on Information, Networking and Automation*, pp. 249-253, 2010.

[48] Alman, D.H. and L. Ningfang, "Overtraining in Back-propagation Neural Networks: A CRT Color Calibration Example," *Color Research & Application*, pp. 122-125, 2002.

[49] R. Vidakovic, "Wavelets for Kids, a tutorial introduction," *Wavelet Digest*, 1991.

APPENDICES

Appendix A Matlab codes

- Neural network training script

```
%% Image registration Simulation
% Author - Hyunjong Choi

clc; clear all; close all;% delete H.mat bias.mat weights.mat;

% Load training inputs/outputs files
load train_test_data/326_inputs_cv_test5.mat;
load train_test_data/326_outputs_cv_test5.mat;

% Train parameters
H = [64];    % Hidden neurons
rate = 0.0001;  % Learning rate
epoch = 1000;   % Iteration

% Train weighting factors and parameters
[ weights, layer_out, error, val_error, bias, actcnt, train_time, save_w,
save_b] = NeuralNetwork_Train_Validation(inputs, outputs, H, rate, epoch);

%% Calculate RMSE of trained networks
Err = abs(outputs(:,1:400) - layer_out{end});
rot_E = sqrt(mean(Err(1,:).^2));
scale_x_E = sqrt(mean(Err(2,:).^2));
scale_y_E = sqrt(mean(Err(3,:).^2));
trans_x_E = sqrt(mean(Err(4,:).^2));
trans_y_E = sqrt(mean(Err(5,:).^2));
shear_x_E = sqrt(mean(Err(6,:).^2));
shear_y_E = sqrt(mean(Err(7,:).^2));
Train_result =
[rot_E;scale_x_E;scale_y_E;trans_x_E;trans_y_E;shear_x_E;shear_y_E];
```

- Neural network testing script

```
%% 2. Testing networks
%clc; %clear all;

load train_test_data/327_t_inputs_cv_test4_S_10.mat;
load train_test_data/327_t_outputs_cv_test5.mat;

% Testing network
idx = 1;
for s = 5 : 5: length(save_w)
    test_out = NeuralNetwork( t_inputs, save_w{s}, H, save_b{s});

    t_Err = t_outputs' - test_out{end}';

    mean_Err = mean(t_Err)';
    absmean_Err = mean(abs(t_Err))';
    std_Err = std(t_Err)';
    rmse = sqrt(mean(t_Err.^2))';
    max_Err = max(abs(t_Err))';
    min_Err = min(abs(t_Err))';

    tmp_test_E = t_outputs - test_out{end};
    test_E(idx) = mean(sum((tmp_test_E)'.^2)/50);

    idx = idx + 1;
end
% Find minimum error and check the error values
[Y I] = min(test_E);

test_out = NeuralNetwork( t_inputs, save_w{I*5}, H, save_b{I*5});
t_Err = t_outputs' - test_out{end}';

mean_Err = mean(t_Err)';
absmean_Err = mean(abs(t_Err))';
std_Err = std(t_Err)';
rmse = sqrt(mean(t_Err.^2))';
max_Err = max(abs(t_Err))';
min_Err = min(abs(t_Err))';
result = [rmse mean_Err absmean_Err std_Err max_Err min_Err];

fig = figure(1);
x = 5:5:actcnt;
plot(error, 'k--', 'LineWidth', 1, 'Marker', '*'); grid on; hold on;
plot(x, val_error, 'k:', 'LineWidth', 1, 'Marker', '^');
tline = plot(x, test_E, 'k-.', 'LineWidth', 1, 'Marker', 's');
xlabel('Iteration'); ylabel('MSE'); title('Training/Validation/Testing
error');
legend('Training error','Validation error','Testing error');
xlim([0 1000]);

dcm_obj = datacursormode(fig);
set(dcm_obj,
'DisplayStyle','datatip','SnapToDataVertex','off','Enable','on');
target = handle(tline);
dtip = dcm_obj.createDatatip(target);
position = [I*5, Y];
update(dtip, position);
```

- Neural network training function

```
function [weights, layer_out, E, val_E, bias, cnt, train_time, save_w,
save_b] = NeuralNetwork_Train_Validation(raw_inputs, raw_outputs, H, rate,
epoch)
% function [weights, actOutputs] = NeuralNetwork_Train(inputs, d_outputs, H,
rate, epoch)
% Description of the function : The function is to train weights and
parameters of general artificial neural network.
%
```

```
%   Input parameters(arguments) are :
%       inputs : Input vectors
%       outputs : Desired outputs
%       H : hidden layer & the number of neurons
%       iteration : the number of iteration to execute
%       rate : learning rate
%
%   Output values returned are :
%       weights : Weights factors
%       layer_out : Actual output vectors
%       E : training error
%       bias : bias
%
% Developed by: Hyunjong Choi
% Revised: 6/1/2015
% Updated: 10/4/2015
% - a bug based on handcalculation code:
% - remove useless parameters

rand('seed',1);

% Divide training and validation set
inputs = raw_inputs(:,1:400);
outputs = raw_outputs(:, 1:400);
val_inputs = raw_inputs(:, 401:450);
val_outputs = raw_outputs(:, 401:450);

% Get NN dimensions
HLayer = size(H, 2);
TLayer = HLayer + 2;
M = size(inputs, 1);
N = size(outputs, 1);
Q = size(inputs, 2);
a = 1; b = 1; alpha = 0.5;

% Initialize parameters
% weights : It includes weights for bias at first row)
for h = 1 : HLayer+1
    if h == 1
        weights{h} = 0.01*rands(H(h), M);
    elseif h == (HLayer+1)
        weights{h} = 0.01*rands(N, H(h-1));
    else
        weights{h} = 0.01*rands(H(h), H(h-1));
    end
end

% load 'init_weights.mat';
n_weights = weights;
p_weights = weights;
p_d_w = weights;
d_w = weights;
save_w = {};
save_b = {};
for l = 1 : TLayer
    if l == 1
```

```
        layer_out{l} = ones(M, Q);
    elseif l == TLayer
        layer_out{l} = ones(N, Q);
        net{l} = ones(N, Q);
        bias{l} = 0.01*rands(N, 1);
    else
        layer_out{l} = ones(H(l-1), Q);
        net{l} = ones(H(l-1), Q);
        bias{l} = 0.01*rands(H(l-1), 1);
    end
end

% load 'init_bias.mat';
n_bias = bias;
p_bias = bias;
p_d_b = bias;
d_b = bias;

cnt = 1; val_cnt = 1;
tic
for i = 1 : epoch
    order = randperm(Q);
    for q = 1 : Q
        % For one iteration, it includes the number of input SETs
        % A set of inputs has the number of input elements
        % Generate noise and add it to training inputs
        layer_out{1}(:,order(q)) = inputs(:,order(q));

        % Forward path of networks
        for n = 1 : (TLayer-1)
            % Summation part
            net{n+1}(:,order(q)) = weights{n}*layer_out{n}(:,order(q)) +
bias{n+1};

            % Activation part
            if n == TLayer-1
                layer_out{n+1}(:,order(q)) = purelin(net{n+1}(:,order(q)));
            else
                layer_out{n+1}(:,order(q)) =
hyperbolic(net{n+1}(:,order(q)), a, b);
            end
        end

        % Backward path (Update parameters)
        for n = 1 : (TLayer-1)
            % Derivative of output of neurons (edited 10/4/2015)
            out_der = (b/a).*(a-layer_out{TLayer-
n+1}(:,order(q))).*(a+layer_out{TLayer-n+1}(:,order(q)));

            % Output layer neurons
            if n == 1
                % Calculate errors of output neurons (edited 10/4/2015)
                error = outputs(:,order(q)) - layer_out{TLayer}(:,order(q));
                delta{TLayer}(:,order(q)) = error;
            else
```

```
                delta{TLayer-n+1}(:,order(q)) = weights{TLayer-
n+1}'*delta{TLayer-n+2}(:,order(q)).*out_der;
            end
        end

        for n = 1 : (TLayer-1)
            if n == 1
                d_w{TLayer-1} = alpha*p_d_w{TLayer-1} +
rate.*delta{TLayer}(:,order(q))*layer_out{TLayer-1}(:,order(q))';
                n_weights{TLayer-1} = weights{TLayer-1} + d_w{TLayer-1};

                d_b{TLayer} = alpha*p_d_b{TLayer} +
rate.*delta{TLayer}(:,order(q));
                n_bias{TLayer} = bias{TLayer} + d_b{TLayer};
            else
                d_w{TLayer-n} = alpha*p_d_w{TLayer-n} + rate.*delta{TLayer-
n+1}(:,order(q))*layer_out{TLayer-n}(:,order(q))';
                n_weights{TLayer-n} = weights{TLayer-n} + d_w{TLayer-n};
                d_b{TLayer-n+1} = alpha*p_d_b{TLayer-n+1} +
rate.*delta{TLayer-n+1}(:,order(q));
                n_bias{TLayer-n+1} = bias{TLayer-n+1} + d_b{TLayer-n+1};
            end
        end

        % Save as previous stage value
        p_weights = weights;
        weights = n_weights;
        p_bias = bias;
        bias = n_bias;
        p_d_w = d_w;
        p_d_b = d_b;

    end

    % Denormalising
    tmp_E = outputs - layer_out{TLayer};
    E(cnt) = mean(sum((tmp_E)'.^2)/Q);
    E(1,cnt)

    if (mod(cnt,5) == 0)
        val_act = NeuralNetwork( val_inputs, weights, H, bias);
        tmp_E_val = val_outputs - val_act{TLayer};
        tmp_val_E = mean(sum((tmp_E_val)'.^2)/50);
        val_E(val_cnt) = tmp_val_E;

        save_w{cnt} = weights;
        save_b{cnt} = bias;

        val_cnt = val_cnt + 1;
    end

    if (E(1,cnt) < 0.00001)
        break;
    end
    cnt = cnt + 1;
```

```
end
train_time = toc
end
```

- Neural network forward loop function

```
function [ layer_outs ] = NeuralNetwork( inputs, weights, H, bias)
% function [ outputs ] = NeuralNetwork( inputs, weights, H, a, b);
% Description of function : This function is to get output of wavelet
% neural network with trained weighting factors and parameters of wavelet
% function
%
% Input parameters (arguments) are:
%     inputs : input vectors (N X M), N:elements, M:# of input sets
%     weights : weights factors
%     H : hidden layer & the number of neurons
%     bias : bias values
%
% Output values returned are:
%     layer_out: outputs of NN
%
% Developed by: Hyunjong Choi
% Revised: 6/1/2015

HLayer = size(H, 2);
TLayer = HLayer + 2;
M = size(inputs, 1);
Q = size(inputs, 2);
P = H(1);
a = 1; b = 1;

% tic
for q = 1 : Q
    % For one iteration, it includes the number of input SETs
    % A set of inputs has the number of input elements
    layer_outs{1}(:,q) = inputs(:,q);

    % Forward path of networks
    for n = 1 : (TLayer-1)
        % Summation part
        net{n+1}(:,q) = weights{n}*layer_outs{n}(:,q) + bias{n+1};

        % Activation part
        if n == TLayer-1
            layer_outs{n+1}(:,q) = purelin(net{n+1}(:,q));
        else

            layer_outs{n+1}(:,q) = hyperbolic(net{n+1}(:,q), a, b);
        end
    end
end
% test_time = toc;
end
```

- Generate training/testing image sets

```
%% Image registration Generate train and test sets
clc; clear all; close all;

%% 1. Read image file and set the number of training and testing
filename = 'brain2';
im = imread([filename, '.jpg']);

% Train parameters
numTrain = 450;      % Number of training pairs
numTest = 50;        % Number of testing pairs
M = numTrain + numTest;
keypoints = 50;      % Number of keypoints coordinates from SIFT

%% 2. Generate random variables for rotation, scale, translation, shear
rand('seed',2);
rotation = randi([-12 12],M,1);
scale_x = 0.9 + (1.1 - 0.9).*rand(M, 1);
scale_y = 0.9 + (1.1 - 0.9).*rand(M, 1);
trans_x = randi([-5 5],M,1);
trans_y = randi([-5 5],M,1);
shear_x = 0.1.*rand(M, 1);
shear_y = 0.1.*rand(M, 1);

%% 3. Save variables & generate affined images

affine_para = [rotation scale_x scale_y trans_x trans_y shear_x shear_y];

for n = 1 : M
    outputImg = Affine_Images(im, rotation(n), scale_x(n), scale_y(n),
trans_x(n), trans_y(n), shear_x(n), shear_y(n));

    imwrite(outputImg, ['../train_images/', filename, '-', num2str(n), '-of-
1000.jpg']);
end

%% 4. Find out keypoints using SIFT (should run in window PC)
for n = 1 : M
     affined = ['../train_images/', filename, '-', num2str(n), '-of-
1000.jpg'];
     [matched_num(n) origin_pos{n} matched_pos{n}] = match([filename,
'.jpg'], affined);
     origin_pairs = origin_pos{n}(1:keypoints, :);
     affined_pairs = matched_pos{n}(1:keypoints,:);

     origin_crd{n} = [origin_pairs(:,1); origin_pairs(:,2)];
     affined_crd{n} = [affined_pairs(:,1); affined_pairs(:,2)];
end


%% 5. Get DCT coefficients from affined images
```

```
dct_tmp1 = {}; dct_coeffi = {};
for m = 1 : M
    im2 = imread(['../train_images/', filename, '-', num2str(m), '-of-
1000.jpg']);
    im2_gray = rgb2gray(im2);
    dct_origin = dct2(im2_gray);
    dct_tmp1{m} = dct_origin(1:10, 1:10);
end

%% 6. Normalization inputs and structuring for network inputs
for m = 1 : M
     input = reshape(dct_tmp1{m}, [], 1);

     tmp_inputs(:, m) = [input; affined_crd{m}];
     inputs(:, m) = zscore(tmp_inputs(:,m));
     outputs(:, m) = [rotation(m) scale_x(m) scale_y(m) trans_x(m)
trans_y(m) shear_x(m) shear_y(m)]';
end

%% 7. Divide for training and testing samples
t_inputs = inputs(:,numTrain+1:numTrain+numTest);
t_outputs = outputs(:,numTrain+1:numTrain+numTest);

inputs = inputs(:,1:numTrain);
outputs = outputs(:,1:numTrain);
```

- Affine transform function

```
function [ outputImage ] = Affine_Images( Im, r_angle, sx, sy, tx, ty, shx,
shy )

% Rotation matrix
R = [cos(r_angle*pi/180) sin(r_angle*pi/180) 0;
    -sin(r_angle*pi/180) cos(r_angle*pi/180) 0;
    0 0 1];

% Scaling matrix
S = [sx 0 0;
    0 sy 0;
    0 0 1];

% % Shearing matrix
SH = [1 shx 0;
    shy 1 0;
    0 0 1];

% Transformation function
TR = R*S*SH;

tform = maketform('affine', TR);
[I X Y] = imtransform(Im, tform, 'XYScale', 1);
outputImage = imtranslate(I, [tx, ty],'OutputView','full');
```

```
end
```

- Extract curvelet keypoints script

```
%% Extract keypoints based on curvelet transform
clc; clear all; close all;



% Train parameters
numTrain = 450;      % Number of training pairs
numTest = 50;        % Number of testing pairs
M = numTrain + numTest;
filename = 'brain2';

for m = 1 : 100
    % Load image
    im = imread(['../../../train_images/brain2-500/', filename, '-',
num2str(m), '-of-500.jpg']);
    im_gray = rgb2gray(im);
    [M N] = size(im_gray);

%     % Noisy image pre-processing
%     disp('Compute all thresholds');
%     F = ones(M);
%     X = fftshift(ifft2(F)) * sqrt(prod(size(F)));
%     Cn = fdct_wrapping(X,0,2);
%
%     % Compute norm of curvelets (exact)
%     E = cell(size(Cn));
%     for s=1:length(Cn)
%       E{s} = cell(size(Cn{s}));
%       for w=1:length(Cn{s})
%         A = Cn{s}{w};
%         E{s}{w} = sqrt(sum(sum(A.*conj(A))) / prod(size(A)));
%       end
%     end
%     Cn = fdct_wrapping(im_gray,1,2);
%     % Apply thresholding
%     Cnt = Cn; sigma = 20;
%     for s = 2:length(Cn)
%       thresh = 3*sigma + sigma*(s == length(Cn));
%       for w = 1:length(Cn{s})
%         Cnt{s}{w} = Cn{s}{w}.* (abs(Cn{s}{w}) > thresh*E{s}{w});
%       end
%     end
%
%     % Take inverse curvelet transform
%     im_gray = real(ifdct_wrapping(Cnt,1));
%


    % Discrete Curvelet transform
    tic; C = fdct_wrapping(double(im_gray), 0); toc;
    Ct = C;
    tt = (real(Ct{1}{1}));
    cv_Dct = dct2(tt);
    curveletDCT = reshape(cv_Dct(1:5,1:5), [], 1);
    % Get Parameters from the result of Curvelet
    [X_rows, X_cols, F_rows, F_cols, N_rows, N_cols] =
fdct_wrapping_param(C, M, N);

    % Get Mean value of each scale
    for s = 1 : length(C)
        cfs = [];
        for w = 1 : length(C{s})
            cfs = [cfs; abs(C{s}{w}(:))];
        end
        scale{s} = sort(cfs, 'descend');
    %     scale_mean{s} = mean(cfs);
        scale_mean{s} = max(cfs)*0.7;
    end
    [s1r s1c] = size(C{1}{1});
    [sr sc] = size(C{length(C)}{1});

    C{1}{1} = zeros(s1r, s1c);
    C{length(C)}{1} = zeros(sr, sc);

    % Set small coefficients to zero
    for s = 1 : length(C)
        for w = 1 : length(C{s})
            C_Keypoints{s}{w} = C{s}{w}.* (abs(C{s}{w})>scale_mean{s});
        end
    end

    % Get keypoints index information of each scale,
    % Take the index information of non-zero values
    origin_key_num = zeros(1, length(C_Keypoints));
    for s = 1 : length(C_Keypoints)
        for w = 1 : length(C_Keypoints{s})
           [tmp_KeyidxR{s}{w} tmp_KeyidxC{s}{w}] = find(C_Keypoints{s}{w} ~=
0);
           origin_key_num(1, s) = length(tmp_KeyidxR{s}{w}) +
origin_key_num(1, s);
        end
    end

    % Determine the number of keypoint to extract
    ratio = floor(origin_key_num./sum(origin_key_num)*150);
    while sum(ratio) < 150
        [val_ratio midx] = max(ratio);
        ratio(midx) = ratio(midx) + 1;
    end
    ratio(1) = 1; ratio(6) = 1;

    % Set small coefficients to zero
    for s = 1 : length(C)
        for w = 1 : length(C{s})
            CS_Keypoints{s}{w} = C{s}{w}.*
(abs(C{s}{w})>=scale{s}(ratio(s)*2));
        end
    end

    for s = 1 : length(CS_Keypoints)
        for w = 1 : length(CS_Keypoints{s})
           [KeyidxR{s}{w} KeyidxC{s}{w}] = find(CS_Keypoints{s}{w} ~= 0);
        end
    end

    figure(m);
    imshow(im); hold on;
    % Get keypoints index in the original image
    num_key = 0; num_key_2 = 0; num_key_3 = 0; num_key_4 = 0; num_key_5 = 0;
    cnt = 1;
    for s = 2 : length(C)-1
        for w = 1 : length(C{s})/2
            for i = 1 : length(KeyidxR{s}{w})
                % Curvelet coefficients
                cv_coeff(cnt, 1) = abs(CS_Keypoints{s}{w}(KeyidxR{s}{w}(i),
KeyidxC{s}{w}(i)));
                cnt = cnt + 1;
                % Coordinate information in the image
                imgIdx{s}{w}(i, 1) = (X_rows{s}{w}(KeyidxR{s}{w}(i),
KeyidxC{s}{w}(i)));
                imgIdx{s}{w}(i, 2) = (X_cols{s}{w}(KeyidxR{s}{w}(i),
KeyidxC{s}{w}(i)));
                % Total number of keypoints
                num_key = num_key + 1;
                keypoints_crd(num_key, 1) = imgIdx{s}{w}(i, 2);
                keypoints_crd(num_key, 2) = imgIdx{s}{w}(i, 1);

                if s == 2   % Scale 2
                    plot(imgIdx{s}{w}(i, 2), imgIdx{s}{w}(i, 1), 'y*');
                    num_key_2 = num_key_2 + 1;
                elseif s == 3   % Scale 3
                    plot(imgIdx{s}{w}(i, 2), imgIdx{s}{w}(i, 1), 'r*');
                    num_key_3 = num_key_3 + 1;
                elseif s == 4   % Scale 4
                    plot(imgIdx{s}{w}(i, 2), imgIdx{s}{w}(i, 1), 'g*');
                    num_key_4 = num_key_4 + 1;
                elseif s == 5   % Scale 5
                    plot(imgIdx{s}{w}(i, 2), imgIdx{s}{w}(i, 1), 'm*');
                    num_key_5 = num_key_5 + 1;
                end
            end
        end
    end

    input_cvKey = [keypoints_crd(:, 1); keypoints_crd(:, 2)];
    inputs(:, m) = zscore([curveletDCT; input_cvKey]);
    outputs(:,m) = affine_para(m,:)';

end

%% Divide for training and testing samples
t_inputs = inputs(:,numTrain+1:numTrain+numTest);
```

```
t_outputs = outputs(:,numTrain+1:numTrain+numTest);

inputs = inputs(:,1:numTrain);
outputs = outputs(:,1:numTrain);
```

- Wavelet neural network training function

```
function [weights, layer_out, E, val_E, bias, lamda, tau, cnt, train_time,
save_w, save_b, save_t, save_l] = Wavelet_NN_Train_Validation(raw_inputs,
raw_outputs, H, rate, epoch)
% function [weights, layer_out, E, bias, lamda, tau, cnt] = Wavelet_NN_Train
(inputs, outputs, H, rate, epoch)
% Description of the function : The function is to train weights and
parameters
% of wavelet neural network.
%
%   Input parameters(arguments) are :
%       inputs : Input vectors
%       outputs : Desired outputs
%       H : hidden layer & the number of neurons
%       epoch : the number of iteration to execute
%       rate : learning rate
%
%   Output values returned are :
%       weights : Weights factors
%       Layer_out : Actual output vectors
%       E : training error
%       bias : bias
%
% Developed by: Hyunjong Choi
% Revised: 6/1/2015

% Divide training and validation set
inputs = raw_inputs(:,1:250);
outputs = raw_outputs(:, 1:250);
val_inputs = raw_inputs(:, 251:300);
val_outputs = raw_outputs(:, 251:300);

% Get NN dimensions
HLayer = size(H, 2);
TLayer = HLayer + 2;
M = size(inputs, 1);
N = size(outputs, 1);
Q = size(inputs, 2);
a = 1; b = 1;
alpha = 0.4;

% Initialize parameters
% weights : It includes weights for bias at first row)
for h = 1 : HLayer+1
    if h == 1
        weights{h} = 0.01*rands(H(h), M);
    elseif h == (HLayer+1)
        weights{h} = 0.01*rands(N, H(h-1));
    else
        weights{h} = 0.01*rands(H(h), H(h-1));
    end
end
n_weights = weights;
p_weights = weights;
p_d_w = weights;
d_w = weights;
tmp_weights = weights;

for l = 1 : TLayer
    if l == 1
        layer_out{l} = ones(M, Q);
    elseif l == TLayer
        layer_out{l} = ones(N, Q);
        net{l} = ones(N, Q);
        bias{l} = rands(N, 1);
        lamda{l} = rands(N, 1);
        tau{l} = rands(N, 1);

        p_lamda{l} = lamda{l};
        p_tau{l} = tau{l};
    else
        layer_out{l} = ones(H(l-1), Q);
        net{l} = ones(H(l-1), Q);
        bias{l} = rands(H(l-1), 1);
        lamda{l} = rands(H(l-1), 1);
        tau{l} = rands(H(l-1), 1);
        p_lamda{l} = lamda{l};
        p_tau{l} = tau{l};

    end
end
n_bias = bias;
p_bias = bias;
p_d_b = bias;
d_b = bias;

d2 = max(max(inputs)); d1 = min(min(inputs));
for l = 2 : TLayer-1
    for h = 1 : H(l-1)
        if h == 1
            a = rands(1);
            tau{l}(h,1) = (d2 - d1)*a;
            lamda{l}(h,1) = 0.5*(d2 - d1);
        elseif mod(h,2) == 0
            tau{l}(h,1) = tau{l}(h/2,1)*rands(1);
            lamda{l}(h,1) = 0.5*(tau{l}(h/2,1) - d1);
        elseif mod(h,2) == 1
            tau{l}(h,1) = d2 - tau{l}((h-1)/2,1)*rands(1);
            lamda{l}(h,1) = 0.5*(d2 - tau{l}((h-1)/2, 1));
        end
    end
end
p_d_tau = tau;
p_d_lamda = lamda;
```

```
d_tau = tau;
tmp_tau = tau;
d_lamda = lamda;
tmp2_lamda = lamda;

save_w = {}; save_b={}; save_l={}; save_t={};
cnt = 1; val_cnt = 1; E(1) = 1000; p_E = E;
tic
for i = 1 : epoch
    order = randperm(Q);
    for q = 1 : Q
        % For one iteration, it includes the number of input SETs
        % A set of inputs has the number of input elements
        layer_out{1}(:,order(q)) = inputs(:,order(q));

        % Forward path of networks
        for n = 1 : (TLayer-1)
            % Summation part
            net{n+1}(:,order(q)) = weights{n}*layer_out{n}(:,order(q)) +
bias{n+1};

            % Activation part
            if n == TLayer-1
                layer_out{n+1}(:,order(q)) = purelin(net{n+1}(:,order(q)));
            else
                layer_out{n+1}(:,order(q)) =
Morlet_Wavelet(net{n+1}(:,order(q)), lamda{n+1}, tau{n+1}, '');
            end
        end

        % Backward path (Update parameters)
        for n = 1 : (TLayer-1)
            % Output layer neurons
            out_der{TLayer-n+1}(:,order(q)) = Morlet_Wavelet(net{TLayer-
n+1}(:,order(q)), lamda{TLayer-n+1}, tau{TLayer-n+1}, 'de');

            if n == 1
                % Calculate errors of output neurons
                error = outputs(:,order(q)) - layer_out{TLayer}(:,order(q));
                delta{TLayer} = error;
            else
                tmp_lamda = (net{TLayer-n+1}(:,order(q)) - tau{TLayer-
n+1})./(lamda{TLayer-n+1}.^2);

                delta{TLayer-n+1} = weights{TLayer-
n+1}'*delta{TLayer}.*out_der{TLayer-n+1}(:,order(q));
                delta_lamda{TLayer-n+1} = weights{TLayer-
n+1}'*delta{TLayer}.*tmp_lamda;
                delta_tau{TLayer-n+1} = weights{TLayer-
n+1}'*delta{TLayer}.*(1./lamda{TLayer-n+1});

            end
        end
        for n = 1 : (TLayer-1)
            if n == 1
                d_w{TLayer-1} = rate.*delta{TLayer}*layer_out{TLayer-
1}(:,order(q))';% + alpha*p_d_w{TLayer-1};
                n_weights{TLayer-1} = weights{TLayer-1} + d_w{TLayer-1};

                d_b{TLayer} = rate.*delta{TLayer};% + alpha*p_d_b{TLayer};
                n_bias{TLayer} = bias{TLayer} + d_b{TLayer};
            else
                d_w{TLayer-n} = rate.*delta{TLayer-n+1}*layer_out{TLayer-
n}(:,order(q))';% + alpha*p_d_w{TLayer-n};
                n_weights{TLayer-n} = weights{TLayer-n} + d_w{TLayer-n};
                d_b{TLayer-n+1} = rate.*delta{TLayer-n+1};% +
alpha*p_d_b{TLayer-n+1};
                n_bias{TLayer-n+1} = bias{TLayer-n+1} + d_b{TLayer-n+1};

                d_tau{TLayer-n+1} = rate.*delta_tau{TLayer-
n+1}.*out_der{TLayer-n+1}(:,order(q));% + alpha*p_d_tau{TLayer-n+1};
                tmp_tau{TLayer-n+1} = tau{TLayer-n+1} + d_tau{TLayer-n+1};
                test_out = Wavelet_NN( inputs(:,order(q)), weights, H, bias,
lamda, tmp_tau);
                tmp_E = mean(sum((outputs(:,order(q)) -
test_out{TLayer})'.^2));
                if (tmp_E < p_E(end))
                    tau{TLayer-n+1} = tmp_tau{TLayer-n+1};
                    p_E(end) = tmp_E;
                end

                d_lamda{TLayer-n+1} = rate.*delta_lamda{TLayer-
n+1}.*out_der{TLayer-n+1}(:,order(q));% + alpha*p_d_lamda{TLayer-n+1};
                tmp2_lamda{TLayer-n+1} = lamda{TLayer-n+1} + d_lamda{TLayer-
n+1};
                test_out = Wavelet_NN( inputs(:,order(q)), weights, H, bias,
tmp2_lamda, tau);
                tmp_E = mean(sum((outputs(:,order(q)) -
test_out{TLayer})'.^2));
                if (tmp_E < p_E(end))
                    lamda{TLayer-n+1} = tmp2_lamda{TLayer-n+1};
                    p_E(end) = tmp_E;
                end
            end
        end

        % Save as previous stage value
        p_weights = weights;
        weights = n_weights;
        p_bias = bias;
        bias = n_bias;
        p_d_w = d_w;
        p_d_b = d_b;

        p_d_w = d_w;
        p_d_b = d_b;
        p_d_tau = d_tau;
        p_d_lamda = d_lamda;
    end
    p_E = E;
    err(1) = sum((outputs(1,:) - layer_out{TLayer}(1,:)).^2)./Q;
    err(2) = sum((outputs(2,:) - layer_out{TLayer}(2,:)).^2)./Q;
    err(3) = sum((outputs(3,:) - layer_out{TLayer}(3,:)).^2)./Q;
    err(4) = sum((outputs(4,:) - layer_out{TLayer}(4,:)).^2)./Q;
    err(5) = sum((outputs(5,:) - layer_out{TLayer}(5,:)).^2)./Q;
    err(6) = sum((outputs(6,:) - layer_out{TLayer}(6,:)).^2)./Q;
    err(7) = sum((outputs(7,:) - layer_out{TLayer}(7,:)).^2)./Q;
    E(cnt) = (err(1)+err(2)+err(3)+err(4)+err(5)+err(6)+err(7))/7;
    E(1,cnt)

    if (mod(cnt,5) == 0)
        val_act = Wavelet_NN( val_inputs, weights, H, bias, lamda, tau);
        tmp_val_E = mean(sum((val_outputs - val_act{TLayer})'.^2)/50);
        val_E(val_cnt) = tmp_val_E;
        save_w{cnt} = weights;
        save_b{cnt} = bias;
        save_t{cnt} = tau;
        save_l{cnt} = lamda;
        val_cnt = val_cnt + 1;
    end

    if (E(1,cnt) < 0.00001)
        break;
    end
    p_E = E(1, cnt);
    cnt = cnt + 1;

end
train_time = toc
end
```

- Wavelet neural network forward loop function

```
function [ layer_out ] = Wavelet_NN( inputs, weights, H, bias, lamda, tau)
% function [layer_out] = Wavelet_NN(inputs, weights, H, bias, lamda, tau);
% Description of function : This function is to get output of wavelet
% neural network with trained weighting factors and parameters of wavelet
% function
%
% Input parameters (arguments) are:
%     inputs : input vectors (N X M), N:elements, M:# of input sets
%     weights : weights factors
%     H : hidden layer & the number of neurons
%     tau: parameters of wavelet function
%     lamda: parameters of wavelet function
%     bias : bias for output layer
%
% Output values returned are:
%     outputs : outputs of WNN
%
% Developed by: Hyunjong Choi
% Revised: 6/1/2015
```

```
HLayer = size(H, 2);
TLayer = HLayer + 2;
epsil = 0.1;
M = size(inputs, 1);
Q = size(inputs, 2);
P = H(1);

% tic
for q = 1 : Q
    layer_out{1}(:,q) = inputs(:,q);

    for n = 1 : (TLayer-1)
        net{n+1}(:,q) = weights{n}*layer_out{n}(:,q) + bias{n+1};
        % Activation part
        if n == TLayer-1
            layer_out{n+1}(:,q) = purelin(net{n+1}(:,q));
        else
            layer_out{n+1}(:,q) = Morlet_Wavelet(net{n+1}(:,q), lamda{n+1}, 
tau{n+1}, '');
        end
    end

end
% test_time = toc
end
```