AFFINE IMAGE REGISTRATION USING ARTIFICIAL NEURAL NETWORKS

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

Pramod Gadde

May 2013

© 2013
Pramod Gadde
ALL RIGHTS RESERVED

COMMITTEE MEMBERSHIP

TITLE: Affine Image Registration Using Artificial Neural Networks

AUTHOR: Pramod Gadde

DATE SUBMITTED: May 2013

COMMITTEE CHAIR: Dr. Xiao-Hua (Helen) Yu, Professor
Electrical Engineering Department

COMMITTEE MEMBER: Dr. Jane Zhang, Associate Professor
Electrical Engineering Department

COMMITTEE MEMBER: Dr. Wayne Pilkington, Associate Professor
Electrical Engineering Department

ABSTRACT

AFFINE IMAGE REGISTRATION USING ARTIFICIAL NEURAL NETWORKS

Pramod Gadde

This thesis deals with image registration of MRI images using neural networks. Image registration combines multiple images of the same subject that were taken at different points in time, from different sensors, or from different points of views into a single image and coordinate system. Image registration is widely used in medical imaging and remote sensing. In this thesis feed forward neural networks and wavelet neural networks are used to estimate the parameters of registration. Simulations show that the wavelet networks provide significantly more accurate results than feed forward networks and other proposed methods including genetic algorithms. Both methods are also shown to be robust to noise and changes in parameter ranges.

TABLE OF CONTENTS

Title	Page
LIST OF TABLES	vii
LIST OF FIGURES	ix
Chapter 1: Image Registration Introduction	1
Chapter 2: Literature Review	4
2.1. Introduction and Affine Transformation	4
2.2. Literature Review: Traditional Registration Methods	7
2.3. Neural Network Registration Methods	10
2.4. Genetic Algorithm Registration Methods	13
2.5. Summary	15
Chapter 3: Algorithm Development	16
3.1. Initial Development.....	16
3.2. Edge Detection to Scale Invariant Feature Transform	18
3.3. SIFT: Scale Invariant Feature Transform.....	20
3.3.1. SIFT: Scale-Space Peak Selection.....	20
3.3.2. SIFT: Keypoint Localization	23
3.3.3. SIFT: Orientation Assignment	26
3.3.4. SIFT: Keypoint Descriptor	26
3.3.5. SIFT: Image Matching	28
3.4. Wavelet Neural Networks	31
3.4.1. Wavelet Neural Network Design.....	33
3.4.2. Learning and Parameter updates	34
3.4.3. Initial Parameters.....	35
Chapter 4: Simulation Results	37

4.1. Discrete Cosine Transform.....	38
4.2. Initial Test Results.....	40
4.3. Wavelet Coefficients vs. DCT coefficients.....	47
4.4. Wavelet Networks	51
4.5. Noise Results.....	54
4.6. Genetic Algorithms	56
4.7. Different Input Images	61
4.8. Recap and Comparison to Other Methods	67
Chapter 5: Conclusion	73
Bibliography	75
Appendix 1: Training Results	80
Appendix 2: Matlab and C-Code	94

LIST OF TABLES

Table	Page
Table 4.1 Testing results for various number of DCT coefficients used as inputs	41
Table 4.2 Testing results for multi layer networks with 100 DCT coefficients inputs.....	42
Table 4.3 Testing Results for various number of matched key points as inputs	43
Table 4.4 Testing Results for multi layer networks using 50 matched points.....	43
Table 4.5 Testing Results for multi layer networks using DCT + Matched Keypoints....	45
Table 4.6 Testing Results for DCT+Matched keypoints network with Shear added	46
Table 4.7 Testing Results for networks w/ 100 Wavelet Coefficients	49
Table 4.8 Testing Results for networks w/ 100 coeff + 50 matched points.....	50
Table 4.9 Testing Results for WNN using Matched Keypoints	52
Table 4.10 Testing Results for WNN using DCT+Matched Keypoints	53
Table 4.11 Testing Results for WNN using DCT+Keypoints with shearing added	54
Table 4.12 Testing results for FFNN trained with 100 coeffs + 50 matched points w/ Noise	55
Table 4.13 Testing results for wavelet networks with 100 coeffs+50 matched points w/ Noise	56
Table 4.14 Error of top solution after 5, 100, 200 iterations on 1 image pair	59
Table 4.15 RMS error after 5, 100, 200 iterations on 10 image pairs	60
Table 4.16 FFNN w/ 64, 40, 15 hidden neurons testing results (no noise)	62
Table 4.17 FFNN w/ 64, 40, 15 hidden neurons testing results (with noise)	63
Table 4.18 Testing results for Wavelet network w/ 31 wavelons (no noise).....	64
Table 4.19 Testing results for Wavelet network w/ 31 wavelons (with noise).....	65
Table 4.20 FFNN w/ 64, 40, 15 hidden neurons testing results.....	66
Table 4.21 Testing results for Wavelet network w/ 31 wavelons.....	67

Table 4.22 Results for FFNN w/ 100 DCT coefficients+50 matched points (no noise) ..	68
Table 4.23 FFNN w/ 100 DCT coefficients + 50 matched points (with noise).....	69
Table 4.24 Results for WNN w/ 100 DCT coeffs + 50 matched points (no noise)	70
Table 4.25 Results for WNN w/ 100 DCT coeffs + 50 matched points (with noise).....	71
Table 4.26 144 DCT coefficient network	71
Table A.1 Training results for various number of DCT coefficients used as inputs	80
Table A.2 Training results for multi layer networks with 100 DCT coefficients inputs ..	80
Table A.3 Training Results for various number of matched key points as inputs	81
Table A.4 Training Results for multi layer networks using 50 matched points	81
Table A.5 Results for multi layer networks using DCT + Matched Keypoints	82
Table A.6 Training Results for DCT+Matched keypoints network with Shear added.....	83
Table A.7 Training Results for networks w/ 100 Wavelet Coefficients.....	84
Table A.8 Training Results for networks w/ 100 coeff + 50 matched points	85
Table A.9 Training Results for WNN using Matched Keypoints	86
Table A.10 Training Results for WNN using DCT+Matched Keypoints	86
Table A.11 Training Results for WNN using DCT+Keypoints with shearing added	87
Table A.12 FFNN trained w/ 100 coeffs + 50 matched points w/ Noise	88
Table A.13 Wavelet networks w/ 100 coeffs+50 matched points w/ Noise	89
Table A.14 FFNN w/ 64, 40, 15 hidden neurons Training results (no noise)	90
Table A.15 FFNN w/ 64, 40, 15 hidden neurons Training results (with noise)	91
Table A.16 Training results for Wavelet network w/ 31 wavelons (no noise).....	92
Table A.17 Training results for Wavelet network w/ 31 wavelons (with noise)	93
Table A.18 Training Results for FFNN trained with 25 Strongest Edges.	93

LIST OF FIGURES

Figure	Page
Fig 2.1: 15° Rotation applied on source image	5
Fig 2.2: Shear factor of 0.3 applied on source image	6
Fig 2.3: Four steps of Image Registration.....	7
Fig 2.4: A single neuron with m inputs.....	11
Fig 3.1: Original Image of the Brain before applying an Affine Transformation	17
Fig 3.2: Source image after applying rotation and scaling	17
Fig 3.3: Edges using Sobel operator	19
Fig 3.4: Showing the Difference of Gaussians	21
Fig 3.5: Neighborhood comparison for DoG images.....	22
Fig 3.6: Repeatability for various number of scales per octave.....	23
Fig 3.7: Repeatability for various smoothing (sigma) values	23
Fig 3.8: Correct matches for various orientations.....	27
Fig 3.9: SIFT (a) Original Image, (b) Image with keypoints and orientations displayed.....	29
Fig 3.10: SIFT (a)Transformed image and its 636 keypoints and orientations, (b) Displaying the first 25 matched points, (c) Displaying all 445 matched keypoints.....	30
Fig 3.11: Wavelet Networks, (a)A wavelon, (b)design of wavelet neural network	33
Fig 3.12: Morlet Wavelet.....	36
Fig 4.1: Logistic Function.....	37
Fig 4.2: Neural Network Structure Used	38
Fig 4.3: Periodicity of DFT and DCT	39
Fig 4.4: Original Image and DCT compressed image	40
Fig 4.5: (a)2-D wavelet Decomposition, (b) 3-level decomposition	47

Fig 4.6: 2-Level Wavelet Decomposition.....	48
Fig 4.7: RMSE Plots, (a) Rotation RMSE vs. # Iterations, (b) plot of all parameters RMSE vs. # iterations.....	52
Fig 4.8: Input Image with Noise Added.....	55
Fig 4.9: One Solution/Chromosome	57
Fig 4.10: Cross Over Example, (a) Parent 1, (b) Parent 2, (c) Child after crossover with n=3	58
Fig 4.11: Genetic Algorithm Results after various iterations	59
Fig 4.12: Two new source images for testing	61

Chapter I: Image Registration Introduction

Image registration is the transformation of different sets of data into one coordinate system in order to align up and overlay multiple images. [8] One image is referred to as the source image and the other image(s) is referred to as the target image(s) to be registered. Image registration is used greatly 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. Each sensor provides new information and so combining the results would be beneficial for doctors. Also it is not likely for multiple images to have the region of interest (a tumor, for example) positioned in exactly the same spot or orientation. The images would need to all be transformed to the same coordinate plane in order to overlap them. This allows doctors to monitor the effects of treatments on patients in a certain region of interest over time. Image registration applications can be broken up into one of four groups depending on how the images were acquired: [8]

1. The images were taken in different points of time. Examples include landscape planning, motion tracking, and monitoring of tumors.
2. The images were taken from different sensors. Examples include radar imaging in remote sensing and combination of magnetic resonance imaging (MRI), ultrasound, or computed tomography (CT) images in medical imaging.
3. The images were taken from different viewpoints. Examples include mosaicing of images in remote sensing and shape recovery in computer vision.

4. The image of a scene and a model of the scene are to be registered. Examples include registering satellite and aerial data into maps in remote sensing, template matching with images in computer vision, and specimen classification in medical imaging.

There exists many methods to carry out the registration depending on the problem [27][8][28]. However most of the algorithms follow the same approach: feature detection, feature matching, transform model estimation, and image transformation. The problems with current methods are that they are extremely time consuming. There are 2 types of registration methods: [8] area based (e.g. correlation, mutual information [20][41]) and feature based (e.g. region features, line features [40][1][2]). This thesis looks at monomodal registration. Monomodal registration is registering images taken by the same imaging modality/sensor (MRI, CT, etc.). Fourier domain based cross-correlation methods are used often for monomodal image registration. [8][42] These methods are effective when the transformation parameters aren't very large. However as the image size is large, or the conditions become more severe (higher rotation, scale, etc.) the computational cost and time expense grows greatly. Correlation methods and mutual information methods also tend to mismatch smooth areas that are not rich in detail. [8] Feature based methods of registration are preferred if the images to be registered are rich in detail. The important task in feature based methods is to have robust features that are invariant to transformations. Details on current algorithms are explained in the next chapter.

This thesis deals with neural network approaches to image registration. The advantage to using neural network approaches is that they require far less time to estimate

the parameters. Once the networks are trained, it is a simple case of passing an input vector through (not an iterative method like traditional methods). Most traditional methods (such as correlation and Fourier methods) only estimate the transformation parameters one at a time. Neural networks will output all the transformation parameters at once. [13][15][32] Another advantage of neural networks is that they are able to map complex input/output relationships due to their non-linear activation function. This allows the networks to better handle the cases when there is high distortion (where traditional methods tend to fail or take a long time).

This thesis consists of 5 chapters. Chapter 2 is a literature review on current registration methods (traditional approaches and neural network based approaches). Chapters 3 discusses the algorithms used in this thesis. Chapter 4 includes computer simulation results. Chapter 5 summarizes the results and discusses future works and possible improvements.

Chapter 2: Literature Review

There are many methods to perform registration of images. An overview of most of the methods developed for registration is summarized in several detailed papers.

[27][8][28] Section 2.1 briefly talks about affine transformations. Section 2.2 focuses on traditional registration methods and section 2.3 talks about neural network and genetic algorithm approaches.

2.1 Introduction and Affine Transformations

There are two types of transformation models that can be used: linear transformations and non-rigid transformations. Linear transformations are global transformations of rotation, scaling, shearing, and translation. Non-rigid transformations allow for local distortions such as stretching lines and mapping straight lines to curves. [43] Non-rigid transformations can be divided into physical based models and function representations. Physical representations come from the theory of continuum mechanics and function representations come from interpolation and approximation theory. [43] This thesis deals with geometric affine transformations (linear transforms).

An affine transform maps pixel coordinates into new coordinates by applying a linear combination of rotation, scaling, shearing, and translation matrices. This type of transformation preserves straight lines and maps parallel lines to parallel lines. In other words, affine transformations are any transform that can be written as:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} ax + by + c \\ dx + ey + f \end{bmatrix} \quad (2.1)$$

Where the pair (x', y') are the transformed coordinates and (x, y) are the original coordinates. A 3x3 matrix representations instead of a 2x2 representation is because of the translation matrix. In a 2-D representation the translation is not a linear transform. However by making it into a 3-D matrix with an extra row of $(0,0,1)$ translations can be performed as a linear operation.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad (2.2)$$

Where c and f are the translation parameters. In a 2x2 representation, the translation would not be a linear transformation.

Rotation can be represented as:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad (2.3)$$

Fig. 2.1 15° Rotation applied on source image

Scaling can be represented as:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} scaleX & 0 & 0 \\ 0 & scaleY & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad (2.4)$$

Shearing can be represented as:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & shearX & 0 \\ shearY & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad (2.5)$$

Fig. 2.2 Shear factor of 0.3 applied on source image

Translation can be represented as:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & \delta x \\ 0 & 1 & \delta y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad (2.6)$$

A complex transformation can be represented as a combination of these matrices. For example performing a rotation after first scaling an image can be represented as:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos \theta & \sin \theta & 0 \\ -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \left(\begin{bmatrix} scaleX & 0 & 0 \\ 0 & scaleY & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \right)$$

2.2 Literature Review: Traditional Registration Methods

The four steps of registration are shown in figure 2.3. [8]

Fig. 2.3 Four steps of Image Registration

The first step is feature detection in which features are extracted from the image.

Examples on features in the spatial domain are edges, corners, and intersections.

Examples in the frequency domain are FFT and DCT coefficients. Ideally the features that are extracted should be resistant to noise and easy to detect. There are two ways to select the features: automatically and manually. In the first way the features are automatically selected by an algorithm (such as edge detection). In the manual way, the user selects the key points in each image. The manual way requires the user to be knowledgeable of the images and is a slower process. Previous works in feature detection dealt with extracting high contrast closed-boundary regions of an image using segmentation methods, [8] using virtual circles to extract a region, [40] affinely invariant neighborhoods based on Harris corner detector and edges, [1] representations of general line segments or elongated anatomic structures, [2] edge detection methods such as the Laplacian of Gaussian or Sobel operator, and local curvature discontinuities detected using the Gabor wavelets. [3] A challenge is ensuring that the features extracted are robust to noise and complex transformations. For example, virtual circles can be

extracted easily and used effectively for registration. However they are very susceptible to background noise and so might not be strong features to use in images with much noise.

In some medical images it is hard to extract strong features described above due to a lack of strong details in the image. In order to perform registration on these types of images, an area-based method is used to extract and match the features. [27] These methods use windows of a fixed size to match with the images. However this approach is susceptible to complex transformations and works best only when there is a simple transformation such as a translation. Another problem with a windowed approach is that areas of low intensity might be mismatched in the target and source image.

The classical area-based approach is the normalized cross correlation approach. [4] The normalized cross correlation function is:

$$C(u, v) = \frac{\sum_{x=0}^W \sum_{y=0}^H [T(x, y) - \bar{T}] [I(u+x, v+y) - \bar{I}(u+x, v+y)]}{\sqrt{\sum_{x=0}^W \sum_{y=0}^H [T(x, y) - \bar{T}]^2 [I(u+x, v+y) - \bar{I}(u+x, v+y)]^2}} \quad (2.7)$$

Where T is the template intensity, I is the image intensity, W is the template width minus 1, and H is the template height minus 1. C is the cross correlation coefficient at the points u and v and x and y are the points in the template window going from 0 to the width and height of the template minus 1. $\bar{T}$ is the average value of the template (computed once) and $\bar{I}(u+x, v+y)$ is the “average value of the of the image in the region coincident with the template” T . [30] This is calculated for window pairs from the target and source images. The windows with the highest cross correlation coefficient $C(u, v)$ are set to match. However as previously mentioned, this approach works well with translations and

a small amount of rotation and scaling. Two other issues with cross correlation (CC) methods are that the computational load grows very fast as the transformation becomes more complex and the flatness of the function maxima due to the self-similarity of the images. [8] One way to deal with these issues and improve the CC performance is by computing the CC on the edges of the images (target and source) rather than the images themselves. [11] The advantage to this approach is that it is less sensitive to intensity changes and more robust against noise. Another approach is the sequential similarity detection algorithm (SSDA). [5] The computation of SSDA is much simpler than the CC. It calculates the sums of the differences in the intensity values in the window pairs and if the difference is too high, the pair is rejected non-match. Despite the problems with the correlation methods, they are popular as they are easy to implement in hardware.

Another useful approach is Fourier methods. The phase correlation method is robust against narrow bandwidth noise and non-uniform time varying illumination disturbances. [8] The method works by performing a 2-D Fourier transform on the source and target images, calculating the cross-power spectrum of the source and target image, taking the inverse Fourier transform, and then finding the location of the peak response. This method, like the previous area methods, is strong when there is only translation present. The last area-based methods are mutual information methods. [8][20][41] This approach is less common than the previous two area-based approaches mentioned. Mutual information is a measure of the dependence between two random variables and is useful in registering images taken from different sensors (common in remote sensing and medical imaging).

The mutual information between random variables x and y is:

$$MI(x, y) = H(x) + H(y) - H(x, y) \quad (2.8)$$

$$H(x) = -E_x(\log_2(P(x)))$$

$H(x)$ is the entropy of the random variable and $P(x)$ is the probability distribution.

Wavelet decomposition is often used in accordance with mutual information techniques.

[9] The images to be registered are decomposed at least two levels. Using the lower band images (low-low band for a 2 level decomposition), the mutual information between the images is found. When the mutual information is maximum, the parameters for rotation and scaling are found for the level of decomposition. The process is then repeated for next lowest level until the mutual information and parameters are found for the original image. This is an iterative process and can be time consuming. Instead finding the MI on neighborhoods in the lowest band image provides similar results. [20] MI methods can work with features such as edges as well rather than intensities (most common approach).

2.3 Neural Network Registration Methods

Most of the traditional approaches to registration are iterative processes or depend greatly on the specific images features. Neural networks are commonly used in pattern recognition problems. They can be thought of as mapping a set of input features to the output. The advantage to neural network approaches in image registration is that once the network is trained, it should be able to accurately provide the transformation parameters even if the input image is changed. Also once the network is trained, any input pattern can be passed through to get an output extremely fast (just feed forward the inputs).

Artificial neural networks are based on the parallel architecture of the brain. The brain is able to process many pieces of differing information at the same time to draw a conclusion. The neurons in an artificial neural network are based on the neurons in your nervous system. The neurons consist of a number of inputs (granted far less complex than the neurons in your brain), an activation function (similar to the brains neurons deciding whether to transmit information or not), and an output (the information passed on to other cells).

An image of a neuron is shown in figure 2.4. [12] There are m inputs to the neuron in the picture. A weight is assigned to each input and the results of the input times the weight are summed up. A bias may also be added to the result of the sums. The sum is then passed through an activation function. The most commonly used activation function is a sigmoid function. This thesis deals with feed forward neural networks with at least one hidden layer.

Fig 2.4 A single neuron with m inputs [12]

There have been several studies on using neural networks for image registration. Elhanary, Sheinfeld, Beck, Kadmon, Tal, and Trisoh in 2000 used DCT coefficients to train their network. [15] They used a 12x12 window to select the coefficients from the top left corner (where most of the information is stored). Their network was a 3 layer network with 144 input nodes, 40 nodes in the hidden layer, and four output nodes (rotation, scale, and translation). This method produced reasonably strong results when small amounts of rotation, scaling, and translation were considered. It was also shown to be robust to additive white gaussian noise and salt-and-pepper noise. However this approach was very vulnerable to the scaling parameter. Abche, Yaacoub, Maalouf, Karam in 2006 trained their network with the Fourier coefficients instead. [13] Once again the transformation parameters used were rotation, scaling, and translation. An 8x8 window around the zero frequency was used to select the input coefficients. Results from this approach were compared to traditional FFT-based approaches. The neural network approach performed significantly better than the traditional approaches. This approach was also more far more accurate than the DCT approach with respect to the scale parameter.

Jianzhen Wu and Jianying Xie trained their networks using low order Zernike moments. [35] 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 to using Zernike moments is that the absolute values of the moments are rotation invariant, robust to noise, and fast to compute. Wu and Xie's approach produced stronger results than the DCT approach with respect to scale and rotation parameters. It produced similar results

with respect to the translation parameters. The previous approaches all used feed forward neural networks. Sarnel and Senlo used radial basis neural networks instead of feed forward networks in order to bypass the training step. [34] DCT coefficients were used as the feature vector for each transformed image. The networks performed similarly to feed forward networks when there was no noise. With noise the radial basis function networks performed better than feed forward networks and were able to accurately estimate the amount of rotation, scaling, and translation.

2.4 Genetic Algorithm Registration Methods

Genetic algorithms are an optimization technique inspired by natural evolution. The algorithm is an iterative process in which the population of solutions is repeatedly modified. Each iteration can be thought of as a ‘generation’. The population of solutions used are referred to as chromosomes. Each individual chromosome is of finite length and consists of the solution to the problem being solved (the transformation parameters for example). A fitness function is used in order to rate how strong each solution is.

In every generation ‘child’ chromosomes are selected to populate the next generation. The ‘children’ can be selected in a variety of ways. A common way is the ‘roulette wheel’ selection. In this selection stronger chromosomes (rated by the fitness function) have a higher chance of being selected. There is also a chance of crossover occurring. This is when two ‘parent’ chromosomes selected swap their solutions at a certain point. Another way is ‘elitism’ in which the best chromosomes from each generation are automatically copied over into the next generation. The remaining child chromosomes are selected by other methods such as the roulette wheel selection. There is also a chance of

mutation occurring in which one parameter in the chromosome is randomly modified.

Once the next generation is filled up, the new chromosomes are ranked according to the fitness function and the process repeats. The algorithm runs until the desired number of iterations is reached or the error is below a threshold. Over time all the individual solutions converge to an optimal solution.

Evolutionary computing approaches to image registration have been around since the 1980s. One of the earliest studies was by Yamnay who used binary coded chromosomes to detect rotation of up to 31 degrees and translation of up to 127 units. [39] More recently, Ignole, Joshi, Deshmukh, and Shete in 2009 were able to accurately estimate rotation, scale, translation, and skew parameters using genetic algorithms. [26] They did this by having the fitness function be the mean square error of the pixel intensities. The chromosomes were made up of the 6 parameters to be estimated. Stronger solutions were more likely to be chosen for the next generation with a crossover rate of 80%.

Battacharya in 2007 performed registration of same and different modality (MR and CT) images by maximizing the mutual information of the images using genetic networks. [38]

Huang and Zhang in 2010 estimated the rotation, scale, and translation methods by using genetic algorithms and the Powell algorithm. [36] The Powell algorithm depends on the initial value which is provided by the genetic algorithm. They had the fitness function be the maximum gradient of mutual information between the source and target image. The algorithm ran until the difference between the fitness function was the same in the child generation as it was in the parent generation. The solution provided by the genetic algorithm was then used as the initial value and the Powell algorithm was used to

obtain a global solution. Another approach is to use the Euclidean distance between control points in each image as the fitness function. [37]

The advantage to the genetic algorithm approaches over neural networks is that they are more robust for higher parameter ranges (rotation, scaling, etc) and provide more accurate results. Since genetic algorithms are an iterative process where the best solutions are used passed on to the next iteration, they eventually converge to an optimal solution. However genetic algorithm approaches are applied on one image at a time and so are much slower than neural networks. The genetic algorithm approaches do not learn any generalized information from each image whereas neural networks learn a way to map an input vector to an output vector, regardless of the image used. Because of this genetic algorithm approaches might be preferred when there is registration of only a small number of images and neural networks approaches are better when there are a lot of images to be registered.

2.5 Summary

The problem with correlation, Fourier, and mutual information approaches is that they become computationally heavy and time expensive as the transformation becomes more complex or the image grows in size. The problem with feature based methods is that they require the features to be easily detectable in the source and transform images (so ideally robust to transformations and noise) and also take longer as the transformations become complex. Neural network approaches are extremely fast once trained well and provide strong results. Genetic algorithms are another iterative method that provides strong results even when the transformations are complex.

Chapter 3: Algorithm Development

Three new approaches to solve the registration problem are presented in this chapter. The first approach is to extract keypoints from the images using the scale invariant feature transform (SIFT) algorithm [18] and then using those points as inputs to a feed forward neural network. The second approach is to use the keypoints along with extracted discrete cosine transform (DCT) coefficients to feed into the network. The size of the networks (number of hidden layers and neurons in each layer) is found through trial and error. The final approach is using DCT coefficients as the inputs to a wavelet neural network. Results from these approaches will be explained in detail in the chapter 4. The advantage to using SIFT keypoints is that they are invariant to linear transformations such as rotation, scaling, and translation. DCT coefficients have been used for registration previously and given decent results. Frequency information (DCT, DFT coefficients) can be found for any image and provide global information of the image (depending on which frequency ranges are used). Keypoints on the other hand are unique to each image and provide local information (local changes around each point). Using SIFT keypoints along with the DCT coefficients should improve the accuracy of registration even further as it provides local and global features.

3.1 Initial Development

The images used in this thesis are MRI images of the brain. Figure 3.1 shows the source image used for all the early tests. These images were taken from an image database at Magnetic Resonance – Technology Information Portal. [45]

Figure 3.1. Original Image of the Brain before applying an Affine Transformation

Figure 3.2. Source image after applying rotation and scaling

As was mentioned in the previous section, the motivation to use neural networks is their ability to learn a way to map input features to a desired output. Once the network is trained well, any new set of input features will be able to produce an accurate estimate of the transformation parameters. The first step to using a neural network approach is to determine the structure of the network.

All the methods mentioned in the previous chapter use a three layer network with one hidden layer (with different number of hidden neurons). There is no perfect number of

hidden layers to use for all cases. The rule of thumb is using as few hidden layers as possible since it is far less complex than many layers. As the number of hidden layers is increased, there is no guarantee that performance will improve. The first test before developing the algorithms is to see if using a larger layer network is better than using a standard 3 layer network. The number of neurons are varied in each hidden layer as well. Having too many or too few neurons will lead to overfitting and underfitting the problem respectively. The inputs to the network for these tests were various number of DCT coefficients taken from the top left corner of the DCT transformed image. The tests show that a 5 layer network with 3 hidden layers and 100 input DCT coefficients performs the best. The hidden layers consist of 44, 25, and 10 hidden neurons.

3.2 Edge Detection to Scale Invariant Feature Transform

A common set of features used to train neural networks are the frequency coefficients. DCT and DFT are used for this purpose. The advantage to using the DCT/DFT coefficients is that they can be extracted from any image. One disadvantage to this approach is that just the coefficients don't give an understanding of what is in the image. A different approach might be to extract and use the edges in the image as the features to train the network. The two common methods used are the gradient based detection and Laplacian based detection. [16] The gradient based methods look for the maximums and minimums in the first derivative to find edges. The Laplacian methods look for zero crossings in the second derivative to find edges. Another method is by using a wavelet algorithm to detect the edges. [24]

One common gradient method to find edges is by using the Sobel Operator. The steps are:

1. First find the gradient in each direction. This can be done by using two 3x3 convolution kernels (one for the gradient in the x-direction and one for the y). The kernels used are:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \text{ and } G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

2. Once the gradient in each direction is found, the two are combined to produce the absolute magnitude of the gradient at every point in the image.

$$|G| = \sqrt{G_x^2 + G_y^2}$$

3. Next threshold the image in order to get rid of the weak edges.

Fig 3.3 Edges using Sobel operator

The strongest edge points can be used to train the network (test results in table 18 in appendix). However, a problem with using edges is that they are not very reliable when

there is noise in the images or in certain types of transformations (shearing, for example). One popular algorithm used to extract distinct invariant features is the scale invariant feature transform (SIFT) developed by David Lowe.

3.3 SIFT: Scale Invariant Feature Transform

SIFT is commonly used in computer vision, image matching, and object recognition. [17][18][19] The biggest advantage to using SIFT to extract features is that the features are invariant to rotation, scaling, and translation, and are robust perspective changes. The SIFT algorithm has been used in image registration as well. [25] One way is to first perform wavelet decomposition of the target and source images. Next SIFT features are extracted from the lowest band sub-image. Then the features are matched and the transformation parameters are estimated. For example if a 2-level decomposition is performed, SIFT points are extracted from the low-low band image. This process is repeated until there is accurate registration. The four main steps to SIFT are: [17][18]

1. Scale-space peak selection
2. Keypoint localization
3. Orientation assignment
4. Keypoint Descriptor.

3.3.1. SIFT: Scale-Space Peak Selection

In the first step the keypoints are detected. This is done by using the difference of Gaussians (DoG). Difference of Gaussians is similar to the Laplacian of Gaussian (see equationn 3.1).

$$L(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}} * I(x, y) \quad (3.1)$$

I is the image and σ is the kernel. DoG as it sounds is simply the difference of Gaussian blurred images using a different σ . An image is first convolved with a Gaussian kernel with σ_1 , and then again with a σ_2 . The second smoothed image is then subtracted from the first in order to produce a DoG. The reason for using DoG is because it is easy and efficient to compute and is a close approximation to a scale-normalized LoG. The DoG convolution kernel is defined as:

$$DoG(x, y) = -\frac{1}{\sqrt{2\pi}} \left[\frac{1}{\sigma_1} e^{-\frac{(x^2+y^2)}{2\sigma_1^2}} - \frac{1}{\sigma_2} e^{-\frac{(x^2+y^2)}{2\sigma_2^2}} \right] \quad (3.2)$$

σ_1 and σ_2 are the widths of the Gaussian kernels. In SIFT the reference image is convolved with successive Gaussians to produce Gaussian blurred images separated by a constant factor k (so $\sigma_2 = \sigma$ and $\sigma_1 = k\sigma$). The images produced are grouped in octaves. An octave is doubling of σ . A constant integer number of images per octave is decided beforehand. [18] Next DoG images are produced by subtracting adjacent images (see Fig 3.4). The images on the left in the figures are the Gaussian images (smoothed with σ 's) and the ones on the right are the difference images. This process is repeated for every octave.

Fig 3.4 Showing the Difference of Gaussians [18]

After all the DoG images are created, the keypoints (maximum and minimum points) of the image are detected. This is done by checking each point against its 8 neighbor points (3x3 neighborhoods provide reliable and fast results) as well as the 9 corresponding neighborhood points in the next and previous images (Fig 3.5). The green dots in the image are the neighborhood points used for comparison and the black x is the point being checked. The minimum and maximum of the points are selected.

Fig 3.5. Neighborhood comparison for DoG images. [18]

Dr. Lowe ran experimental tests to determine the ideal number of scales sampled per octave, width of the kernels (σ), and scale. [18] These results are plotted and shown in figures 3.6 and 3.7. Figure 3.6 shows the repeatability for various number of scales per octave. The top line in the plot shows the percentage of keypoints that are correctly matched in original and transformed images (repeatability of keypoints). The plot shows that having 3 scales per octave produces the highest repeatability. Similarly figure 3.7 shows the repeatability for various values of σ . A higher σ provides for better repeatability. However a larger σ also provides a lower number of matched keypoints (bottom line). In general 1.6 is chosen as the value for σ and is $\sqrt{2}$ chosen as the value for the scale, k . [18]

Fig 3.6 Repeatability for various number of scales per octave [18]

Fig 3.7 Repeatability for various smoothing (sigma) values [18]

3.3.2. SIFT: Keypoint Localization

After the keypoints are all detected, the points with low contrast or localized along an edge are rejected. Each keypoint is fitted to nearby data in order to get location, edge response, and peak magnitude. This is done by using the Taylor expansion up to the quadratic term of the scale-space function (Difference of Gaussian D) shifted so that the

keypoint is at the origin.

$$D(\mathbf{x}) = D + \left(\frac{dD}{dx}\right)^T \mathbf{x} + \frac{1}{2} \mathbf{x}^T \frac{d^2D}{dx^2} \mathbf{x} \text{ where } \mathbf{x} = (x, y, \sigma)^T \quad (3.3)$$

D (difference of Gaussian matrix) and its derivatives are evaluated at the sample point (keypoint) and $\mathbf{x}$ is the offset from the keypoint. [44] Brown suggested calculating the derivatives and the Hessian of D by using, “differences of neighboring sample points” [18]. The location of the extremum, $\hat{\mathbf{x}}$, is found by setting the derivative with respect to $\mathbf{x}$ equal to 0. $\hat{\mathbf{x}}$ is where $D(\mathbf{x})$ is a minima or maxima. Equation 3.4 shows this below.

$$\hat{\mathbf{x}} = - \left(\frac{d^2D}{d\mathbf{x}^2} \right)^{-1} \frac{dD}{d\mathbf{x}} \quad (3.4)$$

Next the value at the extrema, $D(\hat{\mathbf{x}})$, is used to reject peaks with low contrast.

$$D(\hat{\mathbf{x}}) = D + \frac{1}{2} \left(\frac{dD}{dx} \right)^T \hat{\mathbf{x}} \quad (3.5)$$

All peaks with values of $D(x)$ below a threshold (a commonly used value and the one used in this thesis is 0.03) are discarded. [18]

Next, keypoints that have a high edge response are examined. The Difference of Gaussian function has a strong response along edges even if the location along the edge is unstable. [18] A weak keypoint in DoG function has a large principle curvature across the edge and a small curvature in the perpendicular direction. The principle curvatures are computed from the Hessian matrix (2x2):

$$H = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix} \quad (3.6)$$

$$D_{xx} = \frac{d^2D}{dx^2}, \quad D_{yy} = \frac{d^2D}{dy^2}, \quad D_{xy} = \frac{d^2D}{dxdy}$$

D_{xx} is the second derivative of D with respect to x , D_{yy} is the second derivative in y , D_{xy} and D_{yx} are the derivatives with respect to x and y . Once again the derivatives are approximated by the differences of neighboring sample points. The eigenvalues of H are proportional to the principle curvature. Let α and β be the eigenvalues of the H . Only the ratio of the eigenvalues (α and β) is needed. The sum of the eigenvalues is equal to the trace and the product is equivalent to the determinant.

$$Tr(H) = D_{xx} + D_{yy} = \alpha + \beta$$

$$Det(H) = D_{xx}D_{yy} - (D_{xy})^2 = \alpha\beta$$

The ratio of the square of the trace of H to the determinant of H is used to determine if the keypoint should be discarded or not. Let r be the ratio of the largest (α) and smallest (β) eigenvalues: $r = \frac{\alpha}{\beta}$

$$\frac{[Tr(H)]^2}{Det(H)} = \frac{(\alpha + \beta)^2}{\alpha\beta} = \frac{(r\beta + \beta)^2}{r\beta^2} = \frac{(r+1)^2}{r} \quad (3.7)$$

The advantage to using r is the eigenvalues no longer need to be explicitly computed. The equation is at a minimum when $\alpha=\beta$ ($r=1$). By using a threshold for r , it is simple to check the ratio of the principle curvatures.

$$\frac{[Tr(H)]^2}{Det(H)} < \frac{(r+1)^2}{r} \quad (3.8)$$

A common value of r is 10 which means keypoints that have a principle curvature ratio of greater than 10 are discarded. [18] This is done to eliminate strong edge responses. A

high value of r means that there is a large principle curvature across the edge and a small one in the perpendicular direction. Edges are unstable to small amounts of noise and so are not good keypoints.

3.3.3. SIFT: Orientation Assignment

Next, each keypoint is assigned at least one orientation based on local image properties. For each keypoint, the corresponding Gaussian smoothed image, L (see eqn 3.1), with the closest scale is selected. Next, the gradient magnitude, m , and orientations, θ , are calculated using pixel difference. [18]

$$m = \sqrt{(L(x+1, y) - L(x-1, y))^2 + (L(x, y+1) - L(x, y-1))^2}$$

$$\theta = \tan^{-1} \left(\frac{L(x, y+1) - L(x, y-1)}{L(x+1, y) - L(x-1, y)} \right) \quad (3.9)$$

An orientation histogram with 36 bins covering 360 degrees of orientations is created at all points in a circular region around the keypoint. Each point in the region added to the histogram is, “weighted by its gradient magnitude and by a Gaussian-weighted circular window with a σ three times the scale of the keypoint” [18]. The peak in the histogram is detected and the orientations corresponding to it, as well as orientations corresponding to peaks within 80%, are assigned to the keypoint. This means it is possible to have multiple orientations assigned to one keypoint which improves significantly the stability of matching.

3.3.4. SIFT: Keypoint Descriptor

The first three steps of the algorithm ensured invariance to location, scale, and rotation by assigning to each keypoint a location, scale, and orientations. The last step

ensures invariance for remaining parameters such as change in illumination. This is done by first sampling the gradient magnitudes and orientations around each keypoint. The Gaussian image selected is based on scale of the keypoint. Next, orientation histograms with 8 bins each are created over 4×4 neighborhood regions from the sampled magnitude and orientation values in a 16×16 neighborhood around each keypoint. [18] 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 histograms. Since there are 4×4 arrays of histograms with 8 bins each, there are $4 \times 4 \times 8 = 128$ elements in each feature vector descriptor. These numbers were found by Lowe to produce the best results. [18] Lowe's results are shown below in Fig 3.8.

Fig 3.8 Correct matches for various orientations [18]

The plot shows that the percentage of keypoints that give the correct match is strongest when using a 4×4 array of histograms with 8 orientations bins (top line). Lastly the 128 element feature vector is normalized to unit length, then a threshold of 0.2 (a commonly used value determined experimentally by Lowe) is applied in order to reduce non-linear

illumination effects, and is renormalized again to unit length. The renormalization is done in order to provide a better descriptor. This is because if there was a large gradient (before removing it by applying the threshold of 0.2), it would skew the descriptor vector. By removing the larger magnitudes of gradients and renormalizing, the descriptor vector is more robust and useful.

3.3.5 SIFT: Image Matching

SIFT is extremely useful for image matching. This is done by first finding the keypoints in the target and reference images. Next, each keypoint in the reference image is matched against the keypoints in the target image by the nearest neighbor method, that is, the keypoints with the minimum Euclidean distance are matched. However this will lead to errors when there are multiple matches with similar Euclidean distance for one point. In order to deal with this, the ratio of the Euclidean distance between the top two matches is looked at. If the ratio is greater than 0.8 (found by Lowe [18]), then the point is discarded and not matched. This way the number of unambiguous matches is greatly reduced. Figure 3.9 shows the original source image and all its keypoints and orientations. Figure 3.10 has 3 images. The top image is a target (transformed) image displaying its keypoints and orientations. The second image matches up the first 25 keypoints from the target image (the top image) to the original image (see figure 3.9). The last image shows all the 445 matched keypoints.

Fig 3.9 SIFT (a)Original Image, (b) Image with keypoints and orientations displayed

Fig 3.10 SIFT (a) Transformed image and its 636 keypoints and orientations, (b) Displaying the first 25 matched points, (c) Displaying all 445 matched keypoints

Neural Network Approach

In this thesis, the features that are used to train the feed forward neural networks are the keypoints extracted from the image using the SIFT algorithm. First, the SIFT is performed on the source and target images. Next, a list of matched keypoints is made. Fifty matched keypoints are used to train the networks. The first set of networks is trained with just the matched keypoints. The next set of networks is trained with the 50 matched keypoints as well as 100 DCT coefficients of the image. The networks are trained until the average error is below 0.01.

3.4 Wavelet Neural Networks

Wavelet neural networks combine neural networks and wavelet decomposition into one. In simple terms, a wavelet neural network is a neural network with wavelets for its activation functions rather than a sigmoid function. The activation function is what allows the networks to model non-linear functions, and in a wavelet neural network this is done by a combination of a series of wavelets. Wavelets are localized in both the time and frequency domain. An advantage to using a wavelet networks is their ability to identify the localization of signals. This is adjusted in the neural network by changing the scaling and translation parameters. A brief explanation of wavelets and the design of a wavelet network follow.

A wavelet is a wave-like function of finite length. The wavelet function is represented as $\psi(t)$. The three main properties of a wavelet function $\psi(t)$ are: [30]

1. The admissibility condition:

$$C_{\psi} \equiv \int \frac{|\psi(f)|^2}{f} df < \infty \quad (3.10)$$

Where $\psi(f)$ is the Fourier transform of $\psi(t)$. This condition ensures that signal can be reconstructed from its transform without a loss of information.

2. Property one implies that $\psi(t)$ is zero at zero frequency

$$\left(\left| \psi(f) \right|^2 \right) \Big|_{f=0} = 0 \quad (3.11)$$

3. The average value of the wavelet is zero in the time domain

$$\int_{-\infty}^{\infty} \psi(t) dt = 0 \quad (3.12)$$

This last property ensures that the waveform is oscillatory.

In a wavelet transform, $\psi(t)$ refers to the mother wavelet. This is the wavelet function from which all other ‘child’ wavelets are created. Other wavelets are generated by translating and scaling the mother wavelet.

$$\psi_{\lambda,\tau}(t) = \frac{1}{\sqrt{\lambda}} \psi\left(\frac{t-\tau}{\lambda}\right) \quad (3.13)$$

λ is the scaling factor and τ is the translation factor. The $\frac{1}{\sqrt{\lambda}}$ is for energy normalization across different λ and τ

$$\|\psi_{\lambda,\tau}\| = \|\psi\| \text{ for all } \lambda, \tau.$$

The τ factor can be thought of as shifting the wavelet and λ can be thought of as stretching or compressing the wavelet. A λ greater than 1 dilates the signal and a λ less than 1 compresses the signal.

3.4.1 Wavelet Neural Network Design

Wavelet neural networks are similar to a feed forward neural network with one hidden layer. [22] Fig. 3.11 shows the structure of a wavelet neural network. The key difference between a wavelet neural network and a regular neural network is in the activation functions. Regular networks use a non-linear function, such as the logistic function, for the activation function whereas the wavelet network uses a wavelet. [12][33] The neurons in the wavelet network are referred to as wavelons. This means that the output of a single input, single output network, with M wavelons is simply:

$$y(x) = \sum_{i=1}^M w_i \psi_{\lambda_i \tau_i}(x) + \bar{y} \quad (3.14)$$

where the second term is the bias added to deal with functions that have nonzero mean. x is the input data, w is the weight and $\psi_{\lambda\tau}$ is the wavelet function for a given τ and λ . As was mentioned, x is a combination of the inputs based on the weights of the network.

Fig 3.11 Wavelet Networks, (a)A wavelon [12], (b)design of wavelet neural network [23]

If the input to the network is a multidimensional vector, then the output of the wavelon is:

$$\Psi(x_1, \dots, x_N) = \psi_{\lambda_i \tau_i} \left(\sum_{i=1}^N w_{j,i} x_i \right) \quad (3.15)$$

Once again x is the input to the neuron, $w_{j,i}$ is the weight connecting i to j , and $\psi_{\lambda_i \tau_i}$ is the wavelet function used for the given τ and λ .

In a wavelet neural network the scaling factor, λ , and translation factor, τ , are modified along with the weights, w , after each iteration according to a learning algorithm. In the classical networks used so far, the weights were updated according to the back propagation algorithm. The same can be applied for wavelet networks.

3.4.2 Learning and Parameter updates

In the wavelet network the parameters to be updated are the weights, scaling, and translation factors. Let θ be the matrix consisting of these parameters (for a one neuron network θ would be a 3x1 vector). The cost function the learning algorithm seeks to minimize is:

$$J(\theta) = \frac{1}{2} \sum_{i=1}^N (y - y_{j,\theta})^2 \quad (3.16)$$

$y_{j,\theta}$ is the output of neuron j for the given parameters in θ and y is the desired output.

To minimize the error, gradient of the error is calculated and gradient descent is used once again (same as classical neural networks) to update θ . The gradients are shown below:

$$\Delta w_{j,i} = (y - y_{j,\theta}) \frac{d}{dw} \left(\sum_{i=1}^M w_i \psi_{\lambda_i \tau_i}(x) \right) = (y - y_{j,\theta}) \psi'_{\lambda,\tau}(x) \quad (3.17)$$

$$\Delta\tau_i = (y - y_{j,\theta}) \frac{d}{d\tau_i} \left(\sum_{i=1}^M w_{j,i} \psi_{\lambda_i \tau_i}(x) \right) = -(y - y_{j,\theta}) \frac{1}{\lambda} \psi'_{\lambda, \tau}(x) \quad (3.18)$$

$$\Delta\lambda_i = (y - y_{j,\theta}) \frac{d}{d\lambda_i} \left(\sum_{i=1}^M w_{j,i} \psi_{\lambda_i \tau_i}(x) \right) = -(y - y_{j,\theta}) \left(\frac{x_k - t_i}{\lambda_i^2} \right) \psi'_{\lambda, \tau}(x) \quad (3.19)$$

$w_{j,i}$ is the weight to neuron j from neuron i , τ_i is the translation factor for neuron i , and λ_i is the scale factor for neuron i . In this thesis ψ is the Morlet wavelet as it is a commonly used wavelet. [39][21]

$$\psi(x) = e^{\frac{-x^2}{2}} \cos(5x) \Rightarrow \psi'(x) = -e^{\frac{-x^2}{2}} (5\sin[5x] + x\cos[5x]) \quad (3.20)$$

Once the gradients are calculated, the parameters are all updated by an amount η .

$$w_{j,i} = w_{j,i} + \eta * \Delta w_{j,i}$$

$$\tau_i = \tau_i + \eta * \Delta \tau_i$$

$$\lambda_i = \lambda_i + \eta * \Delta \lambda_i$$

3.4.3 Initial Parameters

One of the most important steps in designing a network is initialization of the parameters. To initialize τ and λ , the range $D = [a, b]$ over which the function is approximated needs to be known. In this thesis all the data is normalized to unity which means a is 0 and b is 1. With this known, the steps are:

1. Initialize the weights by giving them random values.
2. Select a point p in the interval $[a, b]$. τ is initialized by selecting a point in the interval $[a, b]$. λ is initialized by: $\lambda = \varepsilon(b-a)$ where ε is a constant and is typically set to be 0.5.

3. Repeat step 2 for the remaining wavelets, except over the intervals $[a, \tau]$, and $[\tau, b]$ and so on.

The networks used have 31 or 63 wavelets and the family of wavelets is the Morlet wavelet (see Fig 3.12).

Fig 3.12 Morlet Wavelet

Wavelet Neural Network Approach

Recently Zernike moments have been used as features to train wavelet networks. In this thesis DCT coefficients and matched SIFT keypoints will be used as the input features for the wavelet network. The results are compared to traditional networks as well as other registration methods. The networks used are made up of 31 or 63 wavelets. Ideally the wavelet network should have a number of wavelets equal to $(2^L - 1)$ where L is an integer specifying how many resolution levels to cover. [12] 31 and 63 are chosen in order to make sure that the networks cover enough resolution levels.

Chapter 4: Simulation Results

Feedforward Network Results

The first test is to determine the structure of the network to use and size of networks to use. DCT coefficients (from a $N \times N$ window) and SIFT keypoints are the inputs to various structured networks. The outputs of the network are the various transformation parameters (Fig 4.2). For these first tests rotation angle, translation, and scale factor are modified. Rotation angle ranges from -12° to 12° , translation from -5 to 5 pixels, and scale factor from 0.9 to 1.1 in each direction. The values are drawn from uniform distributions and are chosen in order to be comparable with previous registration works (similar ranges for the parameters). [13][15][32] The networks use the logistic function as the activation function (Fig 4.1) and are trained using 300 source-target image pairs. 25 other image pairs shall then used to test the networks. Backpropagation is used to train the networks until the desired error is below 0.015. The learning rate for the networks is $\eta = 0.05$. A small η means the network will change at a slower rate and will take longer to converge. A large η means the network will change faster and may converge faster. However a large η might not converge at all as it could lead to ‘over-shooting’ the solution. A smaller rate such as 0.01 would take longer to converge and a rate such as 0.2 might not converge for the complex registration problem.

Fig 4.1. Logistic Function

Fig 4.2 Neural Network Structure Used

4.1 Discrete Cosine Transform

$$f(m, n) = \frac{1}{N * M} \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} F(k, l) e^{i2\pi \left(\frac{kn}{N} + \frac{lm}{M} \right)} \quad (4.1)$$

Before running the tests, the discrete cosine transform shall be briefly explained. A discrete cosine transform is similar to a Fourier transform. The key difference between the two is that the DCT expresses a signal as the sum of cosine functions at different frequencies whereas the DFT expresses it as a sum of cosine and sine functions. A 2-D DCT, like a 2-D DFT, is a separable function. This means a 2-D DCT it is the same as performing the 1-D DCT in each dimension.

The formula for a 2-D DCT is (size NxM image):

$$F(k, l) = \alpha(u)\alpha(v) \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} f(m, n) \cos\left(\frac{(2n+1)\pi u}{2N}\right) \cos\left(\frac{(2m+1)\pi v}{2M}\right)$$

$$\alpha(u) = \begin{cases} \sqrt{\frac{1}{n}} & \text{for } u = 0 \\ \sqrt{\frac{2}{n}} & \text{for } u = 1, \dots, n-1 \end{cases} \quad (4.2)$$

The importance of the DCT can be seen in image compression. The DCT has strong “energy compaction” whereas the DFT has poor “energy compaction” [14][29]. This basically means that a signal can be represented using fewer frequency coefficients. This is due to boundary conditions. The periodicity of the DFT generates false spectral components at boundaries as a signal is likely to have different values at its right and left boundaries. The DCT has even boundaries at both ends and so has smooth continuous transitions. Figure 4.3 displays this.

Figure 4.3 Periodicity of DFT and DCT

After transformation, most of the image’s energy is concentrated in the upper left corner.

Due to this, all the coefficients with values below a threshold can be discarded and still

maintain an accurate representation of the image. Once the coefficients have been discarded, the inverse discrete cosine transform can be performed in order to get back to the time domain.

$$f(m,n) = \alpha(u)\alpha(v) \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} F(k,l) \cos\left(\frac{(2n+1)\pi u}{2N}\right) \cos\left(\frac{(2m+1)\pi v}{2M}\right) \quad (4.3)$$

Figure 4.4 shows the source image and a DCT reduced image. The reduced image was obtained by first performing a 2-D DCT on the original image. Next a 50x50 sized window was used to get the DCT coefficients from the top left corner of the transformed image. Lastly an inverse transform was performed to produce the compressed image.

Figure 4.4. Original Image and DCT compressed image

4.2 Initial Test Results

DCT coefficients have been used as inputs to neural networks in previous works. [15] 144 coefficients from the top left corner (a 12x12 window) were used in order to train the network. [15] However those tests were performed on a 3 layer network with 44 hidden neurons. Before testing more complex networks, different sized windows are used to see

whether a smaller window (so fewer coefficients) would provide similar results. The results are shown in table 4.1. All tables in this thesis are broken up into 2 sub-tables in order to make the values easier to read. These results are the testing results (on 25 image pairs), the training results can be found in the appendix. To make accurate comparisons, the random number generator seed was set to be constant in order to produce the same image pairs for all tests. The networks were standard 3 layer networks with 44 hidden neurons (chosen to be consistent with Elhanany, Sheinfeld, Beck, Kadmon, and Tal's paper [15]). As stated previously, rotation angle ranges from -12° to 12° , translation from -5 to 5 pixels, and scale factor from 0.9 to 1.1 in each direction. The parameters were chosen randomly from uniform distributions.

# DCT Coeffs	49	64	81	100	121	144
Rotation RMSE	0.3681	0.3447	0.3687	0.3382	0.3961	0.3709
Min Rot Error	0.0785	0.0665	0.0072	0.049	0.1065	0.1015
Max Rot Error	3.5668	3.0001	2.0913	2.0711	3.1832	2.671
Scale x RMSE	0.0142	0.0131	0.0137	0.013	0.0121	0.0135
Min Scale x Error	0.001	0.0037	0.0008	0.0005	0.0016	0.0039
Max Scale x Error	0.1198	0.1024	0.0941	0.0627	0.0835	0.0859
Scale y RMSE	0.0114	0.0109	0.0134	0.0113	0.0089	0.0103
Min Scale y Error	0.0011	0.0009	0.0042	0.0011	0.0102	0.002
Max Scale y Error	0.0934	0.0952	0.0849	0.0813	0.0911	0.0876
Translate x RMSE	0.3008	0.161	0.2319	0.2119	0.2637	0.1682
Min Trans x Error	0.0120	0.0025	0.0037	0.0013	0.0283	0.0363
Max Trans x Error	1.6560	2.3073	2.5010	1.7035	2.2091	1.2113
Translate y RMSE	0.3177	0.3443	0.2637	0.2236	0.311	0.2592
Min Trans y Error	0.0085	0.0063	0.0874	0.0124	0.0328	0.0394
Max Trans y Error	1.4488	1.7448	1.4321	1.876	4.2188	1.9075

Table 4.1 Testing results for various number of DCT coefficients used as inputs

From the results it appears that using a 10x10 sized window provides as strong results as using a 12x12 window. Since the 10x10 window is smaller, it will be used for future tests. Next multi hidden layer networks are examined for the 100 coefficient input

network. The testing results are shown in table 4.2 and the training results are in the appendix.

# Hidden Layers and neurons per layer	2 layers 44,25 neurons	3 layers 44,25,10 neurons	4 layers 44,25,10,5 neurons
Rotation RMSE	0.3221	0.3148	0.3652
Min Rot Error	0.0681	0.0233	0.0004
Max Rot Error	2.1052	1.6972	2.1158
Scale x RMSE	0.0123	0.01325	0.0128
Min Scale x Error	0.0015	0.0007	0.0004
Max Scale x Error	0.0803	0.0894	0.1088
Scale y RMSE	0.0403	0.0807	0.0501
Min Scale y Error	0.0006	0.0022	0.0031
Max Scale y Error	0.0721	0.0911	0.1007
Translate x RMSE	0.2688	0.2343	0.2551
Min Trans x Error	0.0916	0.0238	0.019
Max Trans x Error	1.335	1.7761	2.1937
Translate y RMSE	0.3397	0.2548	0.3268
Min Trans y Error	0.0041	0.0043	0.0007
Max Trans y Error	2.2885	1.1602	1.7442

Table 4.2 Testing results for multi layer networks with 100 DCT coefficients inputs

These results show that a network with 3 hidden layers network perform the best when the DCT coefficients are the inputs.

The next step is to repeat the previous two tests using matched SIFT keypoints. Tests need to be run in order to determine the ideal number of matched keypoints to use and the number of hidden layers to use. Once again the hidden layer contains 44 neurons for the standard 3 layer network. 44 neurons are used in order to be consistent with the previous DCT tests.

Inputs (Matched Points*2)	60	80	100	120	140
Rotation RMSE	1.3985	1.496	1.347	1.3544	1.635
Min Rot Error	0.1651	0.1053	0.2663	0.3024	0.4752
Max Rot Error	11.3584	10.042	7.7553	8.8318	9.1200
Scale x RMSE	0.0099	0.0103	0.0113	0.0114	0.0099
Min Scale x Error	0.0009	0.0039	0.0034	0.0011	0.0006
Max Scale x Error	0.1176	0.0039	0.1102	0.0034	0.1305
Scale y RMSE	0.0104	0.0102	0.0093	0.01	0.0092
Min Scale y Error	0.001	0.0013	0.0022	0.003	0.0009
Max Scale y Error	0.1126	0.1028	0.108	0.0885	0.1000
Translate x RMSE	0.6331	0.6521	0.6708	0.6431	0.5463
Min Trans x Error	0.0439	0.0339	0.0416	0.1061	0.1078
Max Trans x Error	7.3882	7.1503	5.576	5.3478	5.1986
Translate y RMSE	0.6194	0.534	0.6515	0.6124	0.7195
Min Trans y Error	0.0707	0.1812	0.0222	0.3218	0.3485
Max Trans y Error	6.5346	5.1665	4.6355	5.2300	6.6665

Table 4.3 Testing Results for various number of matched key points as inputs

# Hidden Layers and neurons	2 layers 44,25 neurons	3 layers 44,25,10 neurons	4 layers 44,25,10,5 neurons
Rotation RMSE	1.5351	1.3415	1.3123
Min Rot Error	0.1093	0.1135	0.1036
Max Rot Error	8.7179	9.0103	8.7018
Scale x RMSE	0.0118	0.0094	0.0117
Min Scale x Error	0.0038	0.0034	0.0206
Max Scale x Error	0.0948	0.0913	0.081
Scale y RMSE	0.0099	0.0094	0.0109
Min Scale y Error	0.0021	0.0003	0.0007
Max Scale y Error	0.0822	0.0677	0.0853
Translate x RMSE	0.2925	0.3026	0.3541
Min Trans x Error	0.099	0.1247	0.0664
Max Trans x Error	5.9721	4.0535	4.2507
Translate y RMSE	0.3151	0.2962	0.3091
Min Trans y Error	0.1011	0.0652	0.0503
Max Trans y Error	5.5341	4.2105	4.3997

Table 4.4 Testing Results for multi layer networks using 50 matched points

Table 4.3 shows the testing results of the network from using the matched points. It appears that using 50 or 60 points gives the best results. Since it is better to use a simpler network, 50 matched points will be used for future tests. Table 4.4 shows the testing results of networks with more than one hidden layer (using 50 matched points as inputs). The number of hidden neurons in the layers was chosen to be the same as the number used for the DCT network tests. The results show that a three or four hidden layer network gives better results than a one or two hidden layer network. The three layer is less complex and so will be used for future tests. The training results for the previous two tests can be found in the appendix (Tables 3 and 4). One thing to note is that the networks trained with the matched points performed far worse than networks trained with DCT coefficients.

The previous tests used either DCT coefficients or matched keypoints as the inputs to the networks. Using just the matched keypoints resulted in poor network accuracy. The next tests use both 50 matched keypoints and 100 DCT coefficients as the input features. Since the number of inputs is now doubled (200 inputs), the number of neurons per layer is also modified.

# Hidden Layers and neurons per layer	1 layer 44 neurons	1 layer 64 neurons	2 layers 44,25 neurons	2 layers 64,40 neurons	3 layers 44,25,10 neurons	3 layers 64,40,15 neurons	4 layers 44,25,10,5 neurons	4 layers 64,40,15,10 neurons
Rotation RMSE	0.5645	0.4244	0.3384	0.3002	0.3098	0.2658	0.3979	0.2832
Min Rot Error	0.0365	0.0455	0.0748	0.0543	0.0287	0.0629	0.0483	0.0155
Max Rot Error	4.5991	2.4289	2.1104	2.196	1.9378	1.5718	1.5757	2.0754
Scale x RMSE	0.0128	0.0119	0.0134	0.0107	0.0132	0.0105	0.0136	0.0101
Min Scale x Error	0.0095	0.0007	0.0061	0.0009	0.0077	0.0062	0.0018	0.0011
Max Scale x Error	0.0942	0.073	0.0731	0.0694	0.0574	0.0614	0.0607	0.0714
Scale y RMSE	0.0115	0.0103	0.0111	0.0143	0.0108	0.0108	0.0109	0.0111
Min Scale y Error	0.0032	0.0041	0.0028	0.0032	0.0042	0.0094	0.0004	0.002
Max Scale y Error	0.0836	0.071	0.0643	0.0708	0.0616	0.0485	0.0602	0.0647
Translate x RMSE	0.2584	0.4241	0.2338	0.3262	0.2342	0.2096	0.2341	0.2296
Min Trans x Error	0.0542	0.1577	0.0155	0.0835	0.1385	0.1001	0.0084	0.0022
Max Trans x Error	2.1074	2.2421	1.9008	1.6307	1.2016	1.2394	1.4902	2.5743
Translate y RMSE	0.3041	0.4691	0.355	0.2858	0.3284	0.2168	0.3434	0.254
Min Trans y Error	0.172	0.0224	0.1083	0.1324	0.0271	0.0047	0.0097	0.0078
Max Trans y Error	1.6584	1.0318	1.5687	1.5132	1.5018	1.1134	1.6335	1.8509

Table 4.5 Testing Results for multi layer networks using DCT + Matched Keypoints

Using the matched keypoints along with the DCT coefficients improves the accuracy of the networks significantly. A five layer network with 64, 40, and 15 neurons in the hidden layers performed the best. The training results can be found in the appendix.

All the previous tests varied the rotation, scaling, and translation. Most registration papers examined in this thesis dealing with affine transformations only deal with these three parameters. One other parameter to consider is shear factor. Only a few papers deal with the shear parameter as well. [26] A shearing transformation preserves straight lines

and maps parallel lines to parallel lines. This type of affine transformation is less likely to occur. The 100 DCT coefficients and 50 matched point tests are rerun (200 inputs) except this time there is shearing from -0.10 to 0.10 as well. The full set of parameter ranges is: rotation angle range from -12° to 12° , translation from -5 to 5 pixels, scale factor from 0.9 to 1.1 in each direction, and shear factor from -0.10 to 0.10.

# Hidden Layers and neurons per layer	3 layers 64,40,15	4 layers 64,40,15,10
Rotation RMSE	0.5843	0.5737
Min Rot Error	0.133	0.0168
Max Rot Error	1.5129	2.3675
Scale x RMSE	0.0131	0.0128
Min Scale x Error	0.0078	0.0003
Max Scale x Error	0.0795	0.0921
Scale y RMSE	0.0117	0.0105
Min Scale y Error	0.0002	0.0034
Max Scale y Error	0.0725	0.0900
Translate x RMSE	0.379	0.3450
Min Trans x Error	0.0455	0.1732
Max Trans x Error	1.4506	1.7214
Translate y RMSE	0.5005	0.4945
Min Trans y Error	0.1562	0.1844
Max Trans y Error	3.3188	2.2211
Shear x RMSE	0.0069	0.0060
Min Shear x Error	0.0014	0.0001
Max Shear x Error	0.0701	0.0692
Shear y RMSE	0.0059	0.0074
Min Shear y Error	0.0011	0.0010
Max Shear y Error	0.0497	0.0716

Table 4.6 Testing Results for DCT+Matched keypoints network with Shear added

Table 4.6 shows the testing results for the networks with the shear parameter added. With shearing introduced, the accuracy of the network dropped. However the network still appears to be making accurate predictions.

4.3 Wavelet Coefficients vs. DCT coefficients

Another type of features that can be used instead of DCT coefficients are wavelet coefficients. Wavelet decomposition is used in image compression applications as well (JPEG-200). For image processing applications two dimensional wavelets are used. A 2-D wavelet decomposition (similar to 2-D DCT and FFT) is the same as performing a 1-D transform in each dimension. As was mentioned in the previous chapter, the wavelet transform can be thought of as passing a signal through low-pass and high-pass filters. Since the 2-D wavelet is the product of two 1-D wavelets, the image is decomposed into four sub-bands (low-low, low-high, high-low, high-high). Fig 4.5 shows the wavelet decomposition.

Fig 4.5 (a)2-D wavelet Decomposition, (b) 3-level decomposition [20]

The top left corner (low-low subband) is the approximation image. Fig 4.6 shows a two level decomposition for the input image used. Neural networks will be trained with the wavelet coefficients extracted from the approximation image. The results will be compared to the DCT coefficient networks. In order to compare the results the windows used to extract the coefficients will be the same size as they were in the DCT networks. A 5-level decomposition is used in order to decompose the input images. This is done so that the entire approximation image (or almost the entire image) can be used to train the networks.

Fig 4.6 2-Level Decomposition [2]

Table 4.7 shows the testing results for networks trained with 100 wavelet coefficients and table 4.8 shows the testing results for networks trained with 100 wavelet coefficients and 50 matched keypoints.

# Hidden Layers and neurons per layer	1 layers 44 Neurons w/o Shear	3 layers 44,25,10 w/o Shear	1 layers 44 Neurons with Shear	3 layers 44,25,10 with Shear
Rotation RMSE	0.6709	0.3873	0.7738	0.6127
Min Rot Error	0.0155	0.0678	0.038	0.0009
Max Rot Error	5.3777	3.6755	5.132	3.0861
Scale x RMSE	0.0056	0.0057	0.0059	0.0062
Min Scale x Error	0.0040	0.0023	0.0010	0.0018
Max Scale x Error	0.1025	0.063	0.0874	0.0606
Scale y RMSE	0.0073	0.0086	0.0112	0.0123
Min Scale y Error	0.0039	0.0029	0.0057	0.0013
Max Scale y Error	0.0951	0.0632	0.1016	0.0702
Translate x RMSE	0.5386	0.43	0.5988	0.5921
Min Trans x Error	0.0273	0.0215	0.0011	0.1123
Max Trans x Error	3.8356	1.498	4.0251	2.3294
Translate y RMSE	0.4634	0.4991	0.4505	0.6132
Min Trans y Error	0.0456	0.08155	0.0151	0.0253
Max Trans y Error	3.3017	1.7649	5.1663	2.0681
Shear x RMSE	N/A	N/A	0.0077	0.0093
Min Shear x Error	N/A	N/A	0.0006	0.0027
Max Shear x Error	N/A	N/A	0.1001	0.0604
Shear y RMSE	N/A	N/A	0.0063	0.0076
Min Shear y Error	N/A	N/A	0.0008	0.0051
Max Shear y Error	N/A	N/A	0.0841	0.0586

Table 4.7 Testing Results for networks w/ 100 Wavelet Coefficients

# Hidden Layers and neurons per layer	1 layers 44 Neurons w/o Shear	3 layers 64,40,10 w/o Shear	1 layers 44 Neurons with Shear	3 layers 64,40,10 w/o Shear
Rotation RMSE	0.7602	0.595	1.092	0.8184
Min Rot Error	0.0267	0.0338	0.3672	0.3034
Max Rot Error	4.8547	3.9124	7.5076	4.5279
Scale x RMSE	0.0059	0.0058	0.0086	0.0071
Min Scale x Error	0.0006	0.0059	0.0004	0.0027
Max Scale x Error	0.0846	0.0554	0.0805	0.0950
Scale y RMSE	0.0077	0.0073	0.0114	0.0114
Min Scale y Error	0.0002	0.0018	0.0051	0.0003
Max Scale y Error	0.0775	0.0704	0.1238	0.0987
Translate x RMSE	0.5299	0.4714	0.5927	0.5385
Min Trans x Error	0.0001	0.3143	0.1613	0.2414
Max Trans x Error	3.8929	2.1915	4.2009	2.4385
Translate y RMSE	0.6718	0.4326	0.5935	0.6209
Min Trans y Error	0.0208	0.1110	0.2886	0.1895
Max Trans y Error	3.2561	2.8783	4.4376	3.4821
Shear x Error	N/A	N/A	0.0092	0.0075
Min Shear x RMSE	N/A	N/A	0.0057	0.0016
Max Shear x Error	N/A	N/A	0.0806	0.0824
Shear y RMSE	N/A	N/A	0.0072	0.0075
Min Shear y Error	N/A	N/A	0.0027	0.0019
Max Shear y Error	N/A	N/A	0.0876	0.0863

Table 4.8 Testing Results for networks w/ 100 coeff + 50 matched points

From these results it appears that using wavelet coefficients provides far more accurate predictions on the scale parameter. However the networks trained with DCT coefficients

provided more accurate rotation, and translation predictions. Also when networks were trained with wavelet coefficients and matched keypoints, the networks didn't see any significant improvement in accuracy. Networks trained with DCT coefficients + matched points performed far better than the wavelet coefficient networks. Because of these results, DCT coefficients are used for all future tests.

4.4 Wavelet Networks

Wavelet networks are next examined. The networks are trained with the same inputs and same parameter ranges as the feedforward networks were in previous tests. The networks have either 31 or 63 wavelons and were run for 15,000 iterations. Fig 4.7(a) is a plot of the average rotation RMSE for a number of iterations. The error stops reducing after about 15,000 iterations. Similar plots for the scale rmse, shear RMSE, and translation RMSE are included in Fig 4.7(b).

The networks are trained with 25 image pairs and 5 other image pairs are used for testing. Only the testing results are shown in this chapter. All the training results are shown in the appendix. The first two tests shown below have rotation, scale, and translation varied. The first test uses 50 keypoints as the inputs and the second test uses 50 keypoints and 100 DCT coefficients.

(a)

(b)

Fig 4.7 RMSE Plots, (a) Rotation RMSE vs. # Iterations, (b) plot of all parameters RMSE vs. # iterations

# Wavelons	31 Wavelons	63 Wavelons
Rotation RMSE	0.4442	0.4835
Min Rot Error	0.0539	0.0727
Max Rot Error	3.5989	2.8358
Scale x RMSE	0.1071	0.1129
Min Scale x Error	0.0286	0.0319
Max Scale x Error	0.1658	0.1758
Scale y RMSE	0.1399	0.110288
Min Scale y Error	0.0128	0.0253
Max Scale y Error	0.1787	0.1586
Translate x RMSE	0.3190	0.2595
Min Trans x Error	0.0324	0.1499
Max Trans x Error	1.9567	2.1586
Translate y RMSE	0.3582	0.4072
Min Trans y Error	0.1211	0.1195
Max Trans y Error	1.5301	1.6491

Table 4.9 Testing Results for WNN using Matched Keypoints

# Wavelons	31 Wavelons	63 Wavelons
Rotation RMSE	0.1609	0.2209
Min Rot Error	0.0351	0.0832
Max Rot Error	0.3301	0.3638
Scale x RMSE	0.0089	0.0082
Min Scale x Error	0.0036	0.0052
Max Scale x Error	0.0307	0.0201
Scale y RMSE	0.0079	0.0098
Min Scale y Error	0.0051	0.0055
Max Scale y Error	0.0341	0.0402
Translate x RMSE	0.0997	0.1163
Min Trans x Error	0.2188	0.0841
Max Trans x Error	0.82	0.7924
Translate y RMSE	0.1076	0.1089
Min Trans y Error	0.0375	0.0793
Max Trans y Error	0.7179	0.5877

Table 4.10 Testing Results for WNN using DCT+Matched Keypoints

Once again adding the 100 DCT coefficients improved the accuracy of the network significantly. Also it is quite easy to see that the wavelet networks performed far better than the feedforward networks.

The next result includes shearing along with rotation, scaling, and translation. Only DCT coefficients and matched keypoints are used for the remaining tests.

The networks, as expected, perform worse when shearing is introduced. From all the tests conducted so far, it is clear to see that wavelet networks perform far better than feedforward networks. This is most easily seen when looking at the networks that used just matched keypoints as the input features. The feedforward networks performed extremely poorly when just matched points were used (average feedforward network rotation error of $\approx 1.3^\circ$ whereas the average rotation error for wavelet networks using matched points is $\approx 0.45^\circ$). Also it appears that increasing the number of wavelons doesn't impact the results significantly. All further tests from here will be done using only 31 wavelons (since it is less complex).

# Wavelons	31 Wavelons	63 Wavelons
Rotation RMSE	0.2462	0.2795
Min Rot Error	0.0079	0.0039
Max Rot Error	0.4877	0.5587
Scale x RMSE	0.0056	0.0078
Min Scale x Error	0.0007	0.0008
Max Scale x Error	0.0139	0.0142
Scale y RMSE	0.011	0.0099
Min Scale y Error	0.0014	0.0008
Max Scale y Error	0.0572	0.0197
Translate x RMSE	0.0867	0.1551
Min Trans x Error	0.0042	0.0149
Max Trans x Error	0.4552	0.0149
Translate y RMSE	0.1911	0.1058
Min Trans y Error	0.016	0.0437
Max Trans y Error	0.3993	0.4489
Shear x RMSE	0.0105	0.0099
Min Shear x Error	0.0047	0.0076
Max Shear x Error	0.0521	0.0429
Shear y RMSE	0.0097	0.0094
Min Shear y Error	0.0013	0.0079
Max Shear y Error	0.0597	0.0504

Table 4.11 Testing Results for WNN using DCT+Keypoints with shearing added

4.5 Noise Results

All the tests performed previously were on noise free images. In real life applications this will not be the case. The networks need to be robust and be able to give accurate predictions even when there is noise. In the next tests Gaussian white noise with zero mean and 0.01 variance (image intensities range from 0 to 1) is added to the images (see Fig 4.7). 100 DCT coefficients and 50 matched keypoints are used as the inputs for the feedforward and wavelet networks. The same parameter ranges are used. Table 4.12 shows the testing results for feedforward networks and table 4.13 shows the results for the wavelet networks.

Fig 4.8 Input Image with Noise Added

# Hidden Layers and neurons per layer	3 layers 64,40,15 w/o Shear	3 layers 64,40,15 w/ Shear
Rotation RMSE	0.4361	0.5744
Min Rot Error	0.0174	0.0202
Max Rot Error	3.5894	3.6563
Scale x RMSE	0.0129	0.0130
Min Scale x Error	0.0029	0.0106
Max Scale x Error	0.0901	0.1017
Scale y RMSE	0.0110	0.0108
Min Scale y Error	0.0002	0.0050
Max Scale y Error	0.0953	0.1042
Translate x RMSE	0.3188	0.4567
Min Trans x Error	0.0727	0.0189
Max Trans x Error	2.5367	1.7156
Translate y RMSE	0.5413	0.5174
Min Trans y Error	0.1453	0.076
Max Trans y Error	2.1306	2.5981
Shear x RMSE	N/A	0.0063
Min Shear x Error	N/A	0.0001
Max Shear x Error	N/A	0.0842
Shear y RMSE	N/A	0.0059
Min Shear y Error	N/A	0.0044
Max Shear y Error	N/A	0.0638

Table 4.12 Testing results for FFNN trained with 100 coeffs + 50 matched points w/ Noise

# Wavelons	31 Wavelons w/o Shear	31 Wavelons w/ Shear
Rotation RMSE	0.3039	0.3181
Min Rot Error	0.005	0.0714
Max Rot Error	0.535	0.4237
Scale x RMSE	0.0108	0.0139
Min Scale x Error	0.003	0.0109
Max Scale x Error	0.0653	0.0174
Scale y RMSE	0.0094	0.0149
Min Scale y Error	0.0046	0.0073
Max Scale y Error	0.0644	0.0266
Translate x RMSE	0.16064	0.2193
Min Trans x Error	0.0107	0.004
Max Trans x Error	0.6498	0.4821
Translate y RMSE	0.2332	0.3057
Min Trans y Error	0.0032	0.0629
Max Trans y Error	0.5669	0.2244
Shear x RMSE	N/A	0.0124
Min Shear x Error	N/A	0.0001
Max Shear x Error	N/A	0.0199
Shear y RMSE	N/A	0.0110
Min Shear y Error	N/A	0.0024
Max Shear y Error	N/A	0.0182

Table 4.13 Testing results for wavelet networks with 100 coeffs+50 matched points w/ Noise

Both types of networks gave accurate estimations even in the presence of noise. This is expected since the features used were low frequency coefficients (so no high frequency noise) and SIFT keypoints (which is robust to noise). The rotation parameter had the most accurate estimations. Also once again the wavelet networks performed far better than the feedforward networks.

4.6 Genetic Algorithms

This thesis focuses on using neural networks to solve the registration problem. However it would be useful to compare the neural network results with a different learning method. In this section a genetic algorithm method is used in order to compare its results with the neural network method results.

For each solution, there are seven parameters: 1 for rotation, 2 for scaling, 2 for shearing, and 2 for translation. [26] Solutions are encoded with the real values of the parameters. Fig 4.8 shows a solution and the 7 parameters. The population size is 100 in each generation. The parameter ranges are the same as those for the neural network case: rotation ranges from $[-12, 12]$, scale factor from $[0.9, 1.1]$, translation factor from $[-5, 5]$, and shear factor from $[-0.1, 0.1]$. The fitness function used is the mean absolute pixel error between the target image and the transformed image using the solution parameters.

1	θ	x-scale	y-scale	x-shear	y-shear	x-translation	y-translation
---	----------	---------	---------	---------	---------	---------------	---------------

Fig 4.9 One Solution/Chromosome

The rules to make up the next generation are:

1. Automatically pass the 10 best solutions from the current generation.
2. Select two random parents from the 35 best solutions from the next 90 and create a child solution. There is an 80% chance of crossover and 20% chance of no crossover (one of the parents is fully copied). To perform the crossover, a number n is randomly chosen from 1 to 6. The first n parameters from one parent are copied to the child and the next $(7-n)$ parameters are copied from the other parent. This process is shown in Fig 4.9. There is a 3% chance of mutation where one of the parameters is randomly modified. This is done by selecting a random number n from 1 to 7. The number chosen is the number that will be modified. There is a 50% chance that the number will increase and 50% chance it will decrease. If the rotation parameter is chosen then it is modified by 1° . The shear and scale factors are modified by 0.01 if selected and translation is modified by 1 pixel if selected.

3. Select two parents from the entire population of 100 solutions, giving more weight to solutions that performed better. There is an 80% chance of crossover and a 3% chance of mutation.

100°	1.1	1.4	.3	.7	30	40
------	-----	-----	----	----	----	----

(a)

131°	1.5	.9	.8	.2	70	60
------	-----	----	----	----	----	----

(b)

100°	1.1	1.4	.8	.2	70	60
------	-----	-----	----	----	----	----

(c)

Fig 4.10 Cross Over Example, (a) Parent 1, (b) Parent 2, (c) Child after crossover with n=3

This process is repeated for 200 iterations (generations). The reason for passing the 10 best solutions is to ensure that the best fitness value cannot get worse for the next generation (since the best solutions are passed). Proportionate selection is one of the most common methods (rule 3) used for selection. This method favors stronger solutions over weak ones. The reason for using the entire population of 100 solutions (so possibly repeating some solutions) is that in the first few generations, there is no set of solutions that stand out. Basically early on, the difference between the best and worst solution (according to the mean pixel intensity difference) is not high enough to favor the top solutions over the worst solutions. If the top solutions were excluded and proportionate selection was used, it takes more iteration to arrive at a reasonable solution. Similarly the purpose of rule 2 is to favor the stronger solutions. 80% is used for crossover rate as it is a common value for many problems. [26]

Results

Fig 4.8 shows the best solution at 5,100, and 200 iterations for one source-target image pair. Table 4.14 shows the error of the top solution after the same iterations. As the number of iterations increase, the top solution image approaches the target image and the error approaches 0.

Fig 4.11 Genetic Algorithm Results after various iterations

# Iterations	5	100	200
Rotation Error	1.0141	0.4944	0.4944
Scale x Error	0.1348	0.0086	0.0045
Scale y Error	0.1086	0.0087	0.0068
Translate x RMSE	2.0	1.0	1.0
Translate y RMSE	-4.0	2.0	1.0
Shear x Error	0.0575	0.0061	0.0048
Shear y Error	0.0407	0.0027	0.0034

Table 4.14 Error of top solution after 5, 100, 200 iterations on 1 image pair

Table 4.14 also gives an idea of the rate of convergence for the top solution. For example, from iteration 5 to 100, rotation error decreases by 0.5197° . From iteration 100 to 200, rotation error decreases by almost 0. This shows how there is initially a huge amount of improvement, however over time the rate of improvement decreases drastically. As the number of iterations increase, the error continues to decrease. However the computation time increases significantly as well.

The previous results were on only image pair. Table 4.15 shows the RMS error after 5, 100, and 200 iterations for 10 source-target image pairs.

# Iterations	5	100	200
Rotation RMSE	0.9278	0.8064	0.7924
Scale x RMSE	0.0062	0.0014	0.0014
Scale y RMSE	0.0088	0.002	0.0012
Translate x RMSE	1.1	0.8	0.8
Translate y RMSE	1.0	1.0	1.1
Shear x RMSE	0.0042	0.0021	0.0011
Shear y RMSE	0.0069	0.003	0.0026

Table 4.15 RMS error after 5, 100, 200 iterations on 10 image pairs

The genetic algorithm approach provides more accurate overall results than the neural network approach. The rotation and translation errors are lower by a bit in the neural network methods (both classical and wavelet), but scale factor and shear factor is significantly lower in the GA approach. However the problem with it is that it is an iterative process and so is extremely time consuming. For 200 iterations in the genetic algorithm approach, it took 1394.122902 seconds for one source-target image pair using Matlab. The wavelet neural network approach, also using Matlab, took 61.183346 seconds to train 25 source-target image pairs. The feedforward network approach took 10.311347 seconds to train 300 source-target image pairs (however it was in C and used the FANN libraries which are considerably faster than the code written in Matlab). All

tests were run on a laptop with an Intel Core i5-2410M processor operating at 2.30 GHz. Another problem is that the algorithm doesn't learn the input/output pattern to apply to a different image or variation. This means if the goal is to register many images, the GA approach would be extremely time consuming (since the entire process would be repeated for every image). The advantage to using neural networks is that once the network is trained, any input pattern can be passed through to get an output extremely fast (just feed forward the inputs).

4.7 Different Input Images

All the tests performed up till now used the same source image. In order to check the strength of the results and to perform a complete study; the tests need to be repeated on other source images. Fig 4.9 shows two other source images that will be used for feedforward network and wavelet network (method 2) tests. DCT coefficients and SIFT keypoints are the inputs to the networks. The parameter ranges are the same as were used for the original tests: rotation ranges from $[-12, 12]$, scale factor from $[.9, 1.1]$, translation from $[-5, 5]$, and shear factor from $[-0.1, 0.1]$.

Fig 4.12 Two new source images for testing

The testing results of the tests are summarized in the tables below. Table 4.11 and 4.12 show the results of feed forward neural networks using 100 DCT coefficients and 50 matched points. The networks are 5 layer networks with 64, 40, and 15 neurons per hidden layer. Table 4.16 has no noise and table 4.17 has white Gaussian noise with mean 0 and variance 0.01 added (on normalized pixel values from 0 to 1). Tables 4.18 and 4.19 show the testing results for wavelet networks (method 2) that have 31 wavelons. Table 4.18 has no noise and table 4.19 has noise added.

Source Image	Fig 4.9a w/o Shear	Fig 4.9a w/ Shear	Fig 4.9b w/o Shear	Fig 4.9b w/ Shear
Rotation RMSE	0.2615	0.5026	0.4312	0.5536
Min Rot Error	0.0848	0.1019	0.0588	0.0722
Max Rot Error	2.0784	2.4754	3.0774	4.9361
Scale x RMSE	0.0134	0.0139	0.0129	0.0128
Min Scale x Error	0.0050	0.0002	0.0017	0.0014
Max Scale x Error	0.0672	0.1064	0.0812	0.0912
Scale y RMSE	0.0098	0.0117	0.0106	0.0108
Min Scale y Error	0.0031	0.0023	0.0049	0.0013
Max Scale y Error	0.0513	0.0932	0.0716	0.1011
Translate x RMSE	0.2165	0.3101	0.2823	0.2301
Min Trans x Error	0.0023	0.0806	0.0068	0.0496
Max Trans x Error	1.6451	1.9021	2.8655	3.6104
Translate y RMSE	0.2314	0.4498	0.2919	0.2234
Min Trans y Error	0.0061	0.0048	0.0046	0.1184
Max Trans y Error	1.458	2.2161	2.1353	2.935
Shear x RMSE	N/A	0.008	N/A	0.0072
Min Shear x Error	N/A	0.0019	N/A	0.0016
Max Shear x Error	N/A	0.0794	N/A	0.07678
Shear y RMSE	N/A	0.0077	N/A	0.0066
Min Shear y Error	N/A	0.0028	N/A	0.0002
Max Shear y Error	N/A	0.0688	N/A	0.0656

Table 4.16 FFNN w/ 64, 40, 15 hidden neurons testing results (no noise)

Source Image	Fig 4.9a w/o Shear	Fig 4.9a w/ Shear	Fig 4.9b w/o Shear	Fig 4.9b w/ Shear
Rotation RMSE	0.3387	0.5220	0.4106	0.6122
Min Rot Error	0.0532	0.0324	0.0818	0.1414
Max Rot Error	2.9363	4.0359	4.0931	5.4375
Scale x RMSE	0.0139	0.0117	0.0137	0.0143
Min Scale x Error	0.0034	0.0034	0.0062	0.0023
Max Scale x Error	0.0604	0.1078	0.0858	0.0947
Scale y RMSE	0.0108	0.0127	0.0143	0.0131
Min Scale y Error	0.0033	0.0042	0.0011	0.0053
Max Scale y Error	0.0764	0.0732	0.0716	0.1059
Translate x RMSE	0.1816	0.3277	0.2969	0.3212
Min Trans x Error	0.0448	0.0077	0.0058	0.0039
Max Trans x Error	1.3014	2.7784	3.2233	4.6511
Translate y RMSE	0.2634	0.5734	0.3375	0.2949
Min Trans y Error	0.0936	0.0158	0.023	0.1086
Max Trans y Error	2.1410	3.8186	2.4751	3.317
Shear x RMSE	N/A	0.0098	N/A	0.0079
Min Shear x Error	N/A	0.0009	N/A	0.0008
Max Shear x Error	N/A	0.0687	N/A	0.0652
Shear y RMSE	N/A	0.0083	N/A	0.0073
Min Shear y Error	N/A	0.0002	N/A	0.0017
Max Shear y Error	N/A	0.0513	N/A	0.0675

Table 4.17 FFNN w/ 64, 40, 15 hidden neurons testing results (with noise)

Source Image	Fig 4.9a w/o Shear	Fig 4.9a w/ Shear	Fig 4.9b w/o Shear	Fig 4.9b w/ Shear
Rotation RMSE	0.2427	0.4185	0.413	0.4701
Min Rot Error	0.0835	0.0696	0.2782	0.0928
Max Rot Error	0.4359	0.8762	0.8321	0.945
Scale x RMSE	0.0072	0.0093	0.0097	0.0133
Min Scale x Error	4.03E-05	0.0001	0.0042	0.0002
Max Scale x Error	0.0074	0.0034	0.0168	0.0346
Scale y RMSE	0.0091	0.0103	0.0108	0.0108
Min Scale y Error	0.0001	0.0014	0.0055	8.24E-07
Max Scale y Error	0.0098	0.0033	0.0170	0.0291
Translate x RMSE	0.0886	0.1448	0.0739	0.3312
Min Trans x Error	0.0464	0.0007	0.0002	0.1945
Max Trans x Error	0.1014	0.0243	0.1866	0.8286
Translate y RMSE	0.0789	0.446	0.1505	0.4172
Min Trans y Error	5.30E-06	0.0031	0.0832	0.0954
Max Trans y Error	0.1640	1.0523	0.4361	0.9677
Shear x RMSE	N/A	0.0104	N/A	0.0106
Min Shear x Error	N/A	0.0123	N/A	0.0038
Max Shear x Error	N/A	0.0424	N/A	0.0523
Shear y RMSE	N/A	0.0117	N/A	0.0120
Min Shear y Error	N/A	0.0056	N/A	0.0204
Max Shear y Error	N/A	0.0424	N/A	0.0372

Table 4.18 Testing results for Wavelet network w/ 31 wavelons (no noise)

Source Image	Fig 4.9a w/o Shear	Fig 4.9a w/ Shear	Fig 4.9b w/o Shear	Fig 4.9b w/ Shear
Rotation RMSE	0.3342	0.4188	0.4218	0.4700
Min Rot Error	0.0727	1.76E-05	0.216	0.006
Max Rot Error	0.8168	1.3563	1.3451	0.4006
Scale x RMSE	0.0087	0.0195	0.0055	0.0161
Min Scale x Error	3.00E-04	0.0012	0.0001	0.0124
Max Scale x Error	0.0161	0.0345	0.0169	0.0372
Scale y RMSE	0.0088	0.0116	0.0065	0.0132
Min Scale y Error	0.0002	0.0019	0.0025	0.004
Max Scale y Error	0.0171	0.0247	0.0388	0.0373
Translate x RMSE	0.1004	0.1803	0.1618	0.3671
Min Trans x Error	0.0884	0.0004	0.0025	0.1043
Max Trans x Error	0.6559	0.5812	0.5911	0.7769
Translate y RMSE	0.1073	4.67E-01	0.1913	0.4337
Min Trans y Error	0.0019	0.0313	0.0921	0.0216
Max Trans y Error	0.5253	1.1204	0.6852	0.7177
Shear x RMSE	N/A	0.014	N/A	0.0148
Min Shear x Error	N/A	0.0018	N/A	0.0173
Max Shear x Error	N/A	0.0387	N/A	0.0776
Shear y RMSE	N/A	0.0150	N/A	0.0170
Min Shear y Error	N/A	0.0016	N/A	0.0039
Max Shear y Error	N/A	0.0318	N/A	0.0742

Table 4.19 Testing results for Wavelet network w/ 31 wavelons (with noise)

From the results it seems that changing the source image had no effect on the accuracy of the estimations provided by the networks. Fig 4.9b had slightly worse results than the other two images, but still gave accurate results. Also once again the wavelet networks gave more accurate rotation, scale factor, and translation estimations whereas the feedforward networks gave more accurate shear factor estimates. Furthermore the networks were once again extremely robust to noise. These accurate results further show the effectiveness of using DCT coefficients and SIFT keypoints as inputs to neural networks for image registration.

Parameter Ranges

In all the previous tests the rotation ranged from -12 to 12° , scaling ranged from 0.9 to 1.1, translation ranged from -5 to 5 pixels, and shearing ranged from -0.1 to 0.1. These

ranges are fine for most practical problems. However there might be some rare cases where there is a high amount of rotation, scaling, shearing, or translation. In this section the ranges for all the parameters are increased significantly. Rotation goes from 0 to 180°, scale goes from 0.5 to 1.5, translation goes from 0 to 100 pixels, and shear goes from -0.5 to 0.5. The image used for testing is the original brain image (Fig 3.1). One issue with large transformations is that it is harder to get as many matched keypoints. With extremely high transformation parameters, there are many cases when there are less than 50 matched keypoints. For these tests 25 matched keypoints will be used. Table 4.20 shows the testing results for a 5 layer feedforward neural network with 64, 40, 15 hidden neurons. Table 4.21 shows the testing results using the wavelet network.

Noise	No	Yes
Rotation RMSE	5.9419	6.1424
Min Rot Error	1.2836	0.8452
Max Rot Error	18.4685	22.8864
Scale x RMSE	0.1412	0.1580
Min Scale x Error	0.0332	0.0024
Max Scale x Error	0.4995	0.4059
Scale y RMSE	0.1646	0.1595
Min Scale y Error	0.0048	0.0405
Max Scale y Error	0.4758	0.4870
Translate x RMSE	5.4218	6.0411
Min Trans x Error	0.6085	1.1914
Max Trans x Error	31.773	30.0148
Translate y RMSE	6.3832	6.1400
Min Trans y Error	1.7400	2.6145
Max Trans y Error	28.2922	34.9606
Shear x RMSE	0.1202	0.1617
Min Shear x Error	0.0617	0.0081
Max Shear x Error	0.3944	0.5224
Shear y RMSE	0.1136	0.1174
Min Shear y Error	0.0283	0.0130
Max Shear y Error	0.6034	0.4159

Table 4.20 FFNN w/ 64, 40, 15 hidden neurons testing results

Noise	No	Yes
Rotation RMSE	4.112	4.3368
Min Rot Error	0.8630	1.1796
Max Rot Error	7.6085	8.9211
Scale x RMSE	0.1141	0.1393
Min Scale x Error	0.0826	0.0131
Max Scale x Error	0.2939	0.3362
Scale y RMSE	0.0932	0.1240
Min Scale y Error	0.0136	0.0208
Max Scale y Error	0.2593	0.3419
Translate x RMSE	3.6881	3.3393
Min Trans x Error	0.9533	2.6003
Max Trans x Error	17.5731	19.9412
Translate y RMSE	3.4207	5.0349
Min Trans y Error	2.2180	1.0021
Max Trans y Error	16.3959	25.9112
Shear x RMSE	0.1399	0.1531
Min Shear x Error	0.0372	0.0138
Max Shear x Error	0.3025	0.3784
Shear y RMSE	0.1311	0.1327
Min Shear y Error	0.0536	0.0434
Max Shear y Error	0.2407	0.3388

Table 4.21 Testing results for Wavelet network w/ 31 wavelons

Even though the errors look large at first, they are actually similar to the results from the networks using smaller ranges. The wavelet neural network had an average rotation error of 4.3368° which is about 2.409% error based on the range of rotation. The wavelet network in the previous case had about a 1.326% error based on the range (24°). It appears that both networks are robust to changing the parameter ranges significantly.

4.8 Recap and Comparison to Other Methods

With all the tests completed, the results are compared to other registration methods. First the best results achieved for each type of network are shown again below. Table 4.22 shows the best testing results of feedforward networks without noise. Table 4.23

shows the best results in the presence of noise. Similarly tables 4.24 and 4.25 show the best testing results of the wavelet networks with and without noise respectively.

# Hidden Layers and neurons per layer	3 layers 64,40,15 w/o Shear	3 layers 64,40,15 w/ Shear
Rotation RMSE	0.2658	0.5843
Min Rot Error	0.0629	0.133
Max Rot Error	1.5718	1.5129
Scale x RMSE	0.0105	0.0131
Min Scale x Error	0.0062	0.0078
Max Scale x Error	0.0614	0.0795
Scale y RMSE	0.0108	0.0117
Min Scale y Error	0.0094	0.0002
Max Scale y Error	0.0485	0.0725
Translate x RMSE	0.2096	0.379
Min Trans x Error	0.1001	0.0455
Max Trans x Error	1.2394	1.4506
Translate y RMSE	0.2168	0.5005
Min Trans y Error	0.0047	0.1562
Max Trans y Error	1.1134	3.3188
Shear x RMSE	N/A	0.0069
Min Shear x Error	N/A	0.0014
Max Shear x Error	N/A	0.0701
Shear y RMSE	N/A	0.0059
Min Shear y Error	N/A	0.0011
Max Shear y Error	N/A	0.0497

Table 4.22 Results for FFNN w/ 100 DCT coefficients + 50 matched points (no noise)

# Hidden Layers and neurons per layer	3 layers 64,40,15 w/o Shear	3 layers 64,40,15 w/ Shear
Rotation RMSE	0.4361	0.5744
Min Rot Error	0.0174	0.0202
Max Rot Error	3.5894	3.6563
Scale x RMSE	0.0129	0.0130
Min Scale x Error	0.0029	0.0106
Max Scale x Error	0.0901	0.1017
Scale y RMSE	0.0110	0.0108
Min Scale y Error	0.0002	0.0050
Max Scale y Error	0.0953	0.1042
Translate x RMSE	0.3188	0.4567
Min Trans x Error	0.0727	0.0189
Max Trans x Error	2.5367	1.7156
Translate y RMSE	0.5413	0.5174
Min Trans y Error	0.1453	0.076
Max Trans y Error	2.1306	2.5981
Shear x RMSE	N/A	0.0063
Min Shear x Error	N/A	0.0001
Max Shear x Error	N/A	0.0842
Shear y RMSE	N/A	0.0059
Min Shear y Error	N/A	0.0044
Max Shear y Error	N/A	0.0638

Table 4.23 FFNN w/ 100 DCT coefficients + 50 matched points (with noise)

# Wavelons	31 Wavelons w/o Shear	31 Wavelons w/ Shear
Rotation RMSE	0.1609	0.2462
Min Rot Error	0.0351	0.0079
Max Rot Error	0.3301	0.4877
Scale x RMSE	0.0089	0.0056
Min Scale x Error	0.0036	0.0007
Max Scale x Error	0.0307	0.0139
Scale y RMSE	0.0079	0.011
Min Scale y Error	0.0051	0.0014
Max Scale y Error	0.0341	0.0572
Translate x RMSE	0.0997	0.0867
Min Trans x Error	0.2188	0.0042
Max Trans x Error	0.82	0.4552
Translate y RMSE	0.1076	0.1911
Min Trans y Error	0.0375	0.0160
Max Trans y Error	0.7179	0.3993
Shear x RMSE	N/A	0.0105
Min Shear x Error	N/A	0.0047
Max Shear x Error	N/A	0.0521
Shear y RMSE	N/A	0.0097
Min Shear y Error	N/A	0.0013
Max Shear y Error	N/A	0.0597

Table 4.24 Results for WNN w/ 100 DCT coeffs + 50 matched points (no noise)

# Wavelons	31 Wavelons w/o Shear	31 Wavelons w/ Shear
Rotation RMSE	0.3039	0.3181
Min Rot Error	0.0050	0.0714
Max Rot Error	0.5350	0.4237
Scale x RMSE	0.0108	0.0139
Min Scale x Error	0.003	0.0109
Max Scale x Error	0.0653	0.0174
Scale y RMSE	0.0094	0.0149
Min Scale y Error	0.0046	0.0073
Max Scale y Error	0.0644	0.0266
Translate x RMSE	0.16064	0.2193
Min Trans x Error	0.0107	0.004
Max Trans x Error	0.6498	0.4821
Translate y RMSE	0.2332	0.3057
Min Trans y Error	0.0032	0.0629
Max Trans y Error	0.5669	0.2244
Shear x RMSE	N/A	0.0124
Min Shear x Error	N/A	0.0001
Max Shear x Error	N/A	0.0199
Shear y RMSE	N/A	0.0110
Min Shear y Error	N/A	0.0024
Max Shear y Error	N/A	0.0182

Table 4.25 Results for WNN w/ 100 DCT coeffs + 50 matched points (with noise)

The tables show that both neural network approaches are extremely strong and provide accurate registration parameters. These results can be compared to previous works as well. Even though different source-target image pairs were used, some information can be gained by the comparisons. Elhanary, Sheinfeld, Beck, Kadmon, Tal, and Trisoh used 144 DCT coefficients to train their network. [15] Their results were:

Affine parameter	Discrete range	Units	RMSE
Translation	[-5,5]	Pixels	4.4721
Rotation	[-6,6]	Degrees	5.2915
Scale	[-6,6]	Percent	5.2915

Table 4.26 144 DCT coefficient network

Since they used a standard 3 layer network and different images, an absolute comparison cannot be made. However it does appear that the results obtained in this thesis are far stronger. For example, they obtained a rotation angle rms error of 5.2915° when the range

was only 12° . In this thesis the wavelet networks obtained a rotation rmse of 0.318° when the range was 24° . Based on these results, it appears that the matched SIFT keypoints are strong features to use as inputs to the networks along with frequency information. Overall it appears that the approaches used in this thesis provide accurate results and are robust to changes in parameter ranges and noise.

Chapter 5: Conclusion

In this thesis neural networks were used to estimate the parameters needed for registration of brain images. The inputs to the networks were DCT coefficients and keypoints on both the target and source images. The keypoints were extracted using the scale invariant feature transform. SIFT keypoints were used since they are invariant to affine transformations such as rotation and scaling. Two types of neural networks were used: feed forward networks and wavelet networks. Both types were robust to noise. Even though the feed forward networks gave accurate results, the wavelet networks performed significantly better. There was some difficulty comparing the results in this study with other papers as many papers didn't list out the average errors. When the results were compared to the few papers that did list the errors; the wavelet networks performed better than most feed forward networks. Furthermore, networks that were trained with just frequency information (DFT and DCT coefficients) performed worse than both networks in this thesis. The genetic algorithm approaches did perform better than the neural network approaches on an image by image basis. However neural networks were far faster and can be used to estimate the parameters for any set of input features. As was stated in chapter 4, the genetic algorithm approach took ≈ 1394 seconds using Matlab for 200 iterations on one source-target image pair. The wavelet network approach using Matlab took ≈ 61 seconds to train with 25 image pairs and the feedforward network approach took ≈ 10 seconds to train with 300 image pairs. The genetic algorithm and wavelet networks were also run on Matlab and timed using Matlabs tic and toc functions. The feedforward approach was much faster as it used the FANN (fast artificial neural network) libraries. The feedforward approaches were run on

in Linux and timed using python's time module. All tests were run on a laptop with an Intel Core i5-2410M processor operating at 2.30 GHz.

One thing to consider in future studies is the family of wavelets used. In this thesis the Morlet wavelet was the one used. However a different mother wavelet (such as a Meyer or Mexican hat) might produce better results. Also there have been methods on using a sigmoid function after the wavelet functions and performing gradient descent after. [39] Basically the outputs of the wavelet nodes would be weighted and summed and then passed through a sigmoid function. This way the outputs are in the range 0 to 1. Next the weights are updated according to gradient descent and similarly so are the scale and translation factors. Another thing to look into in future studies is non-rigid transformations or piecewise linear transformations. Non-rigid transformations are far more complex than linear transformations as they allow for stretching of lines, dilation, and mapping lines to curves. Non-rigid transformations can be broken up into continuum mechanics models and interpolation theory models. [43] A common approach to non-rigid registration is to use basis functions to model the deformations. [18] Another way to deal with non-rigid transformations is to break up the images into parts composed of linear transformations. In this thesis neural networks were used for linear affine transformations. SIFT keypoints are robust to transformation changes and so the networks should provide strong accuracy in each piece of the non-rigid transformed images. One last thing is to perform registration for 3-D images instead of 2-D images.

Bibliography

1. A. Noble, "Finding corners," *Image and Vision Computing* 6, 1988, pp. 121–128.
2. N. Vujovic, D. Brzakovic, "Establishing the correspondence between control points in pairs of mammographic images," *IEEE Transactions on Image Processing* 6, 1997, pp 1388–1399.
3. B.S. Manjunath, C. Shekhar, R. Chellapa, "A new approach to image feature detection with applications," *Pattern Recognition* 29, 1996, pp. 627–640.
4. W.K. Pratt, "Digital Image Processing 2nd ed.", 1991.
5. D.I. Barnea, H.F. Silverman, "A class of algorithms for fast digital image registration," *IEEE Transactions on Computing* 21, 1972, pp. 179–186.
6. J. le Moigne, "Parallel registratin of multi-sensor remotely sensed imagery using wavelet coefficients," *Proceedings of the SPIE: Wavelet Applications* 2242, 1994, pp. 432–443.
7. F. Meskine, N. Taleb, "Improvement of Genetic Algorithms to Image Registration," *Journées Internationales sur l'Informatique Graphique*, 2007, pp. 69-73
8. B. Zitova, J Flusser, "Image registration methods: a survey", *Image and Vision Computing* 21, 2003, pp. 977-1000
9. Q. Dum, L. Chen, "An Image Registration Method Based on Wavelet Transformation," *International Conference on Computer, Mechatronics, Control and Electronic Engineering (CMCE)*, 2010, pp. 158-160
10. G. Stockman, S. Kopstein, S. Benett, "Matching images to models for registration and object detection via clustering," *IEEE Transactions on Pattern Analysis and Machine Intelligence* 4, 1982, pp. 229–241.

11. P.E. Anuta, "Spatial registration of multispectral and multitemporal digital imagery using Fast Fourier Transform," *IEEE Transactions on Geoscience Electronics* 8, 1970, pp. 353–368.
12. D. Veitch, "Wavelet Neural Networks and their application in the study of dynamical systems," Department of Mathematics University of York UK, 2005
<http://maths.york.ac.uk/www/sites/default/files/Veitch.pdf>
13. A.B. Abche, F. Yaacoub, A. Maalouf, E. Karam, "Image Registration based on Neural Network and Fourier Transform," *Proceedings of the 28th IEE EMBS Annual International Conference*, 2006, pp. 4803-4806
14. N. Ahmed, T. Natarajan, K. R. Rao, "Discrete Cosine Transform," *IEEE Transactions on Computers*, January 1974, pp. 90-93
15. I Elhanany, M Sheinfeld, A Beck, Y Kadmon, N Tal, D Tirosh, "Robust Image Registration Based on Feedforward Neural Networks," *IEE International Conference on Systems, Man, and Cybernetics*, 2000, pp. 1507-1511
16. R. Maini, H. Aggarwal, "Study and Comparison of Various Image Edge Detection Techniques," *International Journal of Image Processing (IJIP)*, 2009,
17. D. Lowe, "Object Recognition from Local Scale-Invariant Features," *Proc. of the International Conference on Computer Vision*, 1999, pp. 1150-1157
18. D. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," *International Journal of Computer Vision*, 2004, pp. 91-110
19. T. Hossain, S. Wei Teng, G. Lu, M. Lackmann, "An Enhancement to SIFT-Based Techniques for Image Registration," *2010 Digital Image Computing: Techniques and Applications*, pp. 166-171

20. A. Malviya, S.G. Bhirud, "Wavelet based Image Registration using Mutual Information," *International Conference on Emerging Trends in Electronic and Photonic Devices & Systems*, 2009, pp. 241-244
21. Q. Zhang, A. Benveniste, "Wavelet Networks," *IEEE Transaction on Neural Networks Vol. 3*, 1992, pp. 889-898
22. R. Moghadas, S. Gholizadeh, "A New Wavelet Back Propagation Neural Networks for Structural Dynamic Analysis," *Engineering Letters 16*, 2008, pp.12-17
23. W. Yan-hong, C. Xiang, "Wavelet Neural Network Optimization Applied to Intrusion Detection", *International Conference on Electronic & Mechanical Engineering and Information Technology*, 2011, pp. 3109-3112
24. J. Lixia, Z. Wenjun, W. Yu, "Study on Improved Algorithm for Image Edge Detection," *2nd International Conference on Computer and Automation Engineering*, 2010, pp. 476-479
25. I. A. El Rube, M. A. Sharks, A. R. Salem, "Image Registration Based on Multi-Scale SIFT for Remote Sensing Images," *3rd International Conference on Signal Processing and Communication Systems*, 2009, pp. 1-5
26. V. T. Ingole, C. N. Deshmukh, A. Joshi, D. Shete, "Medical Image Registration Using Genetic Algorithm," *Second International Conference on Emerging Trends in Engineering and Technology*, 2009, pp. 63-66
27. S. Damas, O. Cordon, J. Santamaria, "Medical Image Registration Using Evolutionary Computation: An Experimental Survey," *IEEE Computational Intelligence Magazine*, 2011, pp. 26-42

28. W. Rui, L. Minglu, "An Overview of Medical Image Registration", *Proceedings of the Fifth International Conference on Computational Intelligence and Multimedia Applications (ICCIMA '03)*, 2003, pp.385-390
29. S. A. Khayam, "The Discrete Cosine Transform (DCT): Theory and Application," March 2003
30. R. Gonzalez, R. E. Woods, "Digital Image Processing Third Edition", August 2007.
31. K. Shibata, Y. Ikeda, "Effect of number of hidden neurons on learning in large-scale layered neural networks," *ICROS-SICE International Joint Conference*, 2009, pp. 5008-5013
32. S. Wang, S. L. Faliang Chang, "Image registration Based on Neural Network," *Proceedings of the 5th International Conference on Information Technology and Application in Biomedicine*, 2008, pp. 74-77
33. K. K. Minu, M. C. Lineesh, C. Jessy John, "Wavelet Neural Networks for Nonlinear Time Series Analysis," *Applied Mathematical Sciences Vol. 4*, 2010, pp. 2485 – 2495
34. H. Sarnel, Y. Senol, D. Sagirlibas, "Accurate and Robust Image Registration Based on Radial Basis Neural Networks," *23rd International Symposium on Computer and Information Sciences*, 2008, pp. 1-5
35. 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*, 2004, pp. 4046-4048
36. X. Huang, F. Zhang, "Multi-modal Medical Image Registration Based on Gradient of Mutual Information and Hybrid Genetic Algorithm," *Third International Symposium on Intelligent Information Technology and Security Informatics*, 2010, pp. 125-128

37. C. Chow, H. Tsui, T. Lee, T. Lau, "Medical Image Registration and Model Construction using Genetic Algorithms," *Proceedings International Workshop on Medical Imaging and Augmented Reality*, 2001, pp. 174-179
38. M. Battacharya, "Multi Resolution Medical Image Registration Using Maximization of Mutual Information & Optimization By Genetic Algorithm," *IEEE Nuclear Science Symposium Conference Record*, 2007, pp. 2961-2964
39. X. Hu, B. Kong, F. Zheng, S. Wang, "Image Recognition Based on Wavelet Invariant Moments and Wavelet Neural Networks," *Proceedings of the 2007 International Conference on Information Acquisition*, 2007, pp.275-279
40. H.S. Alhichri, M. Kamel, "Virtual Circles: a new set of features for fast image registration," *Pattern Recognition Letters* 24, 2003, pp. 1181-1190
41. Y. Yamamura, "A Method for Image Registration by Maximization of Mutual Information," *SICE_ICASE International Join Conference*, 2006, pp. 1469-1472
42. D. Padfield, "Masked Object Registration in the Fourier Domain," *IEEE Transactions on Image Processing*, 2012, pp. 2706-2718
43. M. Holden, "A Review of Geometric Transformations for Nonrigid Body Registration," *IEEE Transactions on Medical Imaging*, 2008, pp. 111-128.
44. M. Brown, D. Lowe, "Invariant features from interest point groups," *British Machine Vision Conference*, 2002, pp. 656-665
45. Magnetic Resonance –Technology Information Portal. Retrieved Sept 2012, from <http://www.mr-tip.com/serv1.php>

Appendix 1: Results

# DCT Coeffs	49	64	81	100	121	144
Rotation RMSE	0.3463	0.4336	0.4070	0.4019	0.3604	0.4537
Min Rot Error	0.0049	0.0388	0.0011	0.0213	0.0562	0.0138
Max Rot Error	1.5307	1.1238	1.3745	1.3139	1.0384	1.0052
Scale x RMSE	0.00351	0.003401	0.00358	0.00339	0.00333	0.00323
Min Scale x Error	0.0002	9.00E-06	0.0011	0.0005	0.0001	0.0001
Max Scale x Error	0.1034	0.1029	0.1079	0.1084	0.1041	0.0945
Scale y RMSE	0.00357	0.003215	0.00334	0.00368	0.00338	0.00328
Min Scale y Error	0.0001	0.00003	0.0002	8.00E-06	0.0004	0.00007
Max Scale y Error	0.0953	0.1028	0.1034	0.0975	0.0949	0.1049
Translate x RMSE	0.23302	0.229956	0.22165	0.23515	0.22667	0.22791
Min Trans x Error	0.0041	0.0035	0.0017	0.0009	0.0152	0.008
Max Trans x Error	3.4121	1.1774	1.6488	1.7549	1.2488	1.86
Translate y RMSE	0.17103	0.148256	0.15875	0.16961	0.16178	0.15456
Min Trans y Error	0.0001	0.0201	0.0135	0.0239	0.0079	0.0109
Max Trans y Error	1.8451	2.1416	1.9338	1.8692	1.6957	1.8000

Table A.1 Training results for various number of DCT coefficients used as inputs

# Hidden Layers and neurons per layer	2 layers 44,25 neurons	3 layers 44,25,10 neurons	4 layers 44,25,10,5 neurons
Rotation RMSE	0.40238	0.405587	0.39329
Min Rot Error	0.0091	0.0466	0.01345
Max Rot Error	1.7867	1.8715	1.1566
Scale x RMSE	0.00319	0.003208	0.00317
Min Scale x Error	0.0003	0.0004	0.0008
Max Scale x Error	0.0956	0.0984	0.0895
Scale y RMSE	0.00299	0.002761	0.00299
Min Scale y Error	0.0006	0.0001	0.0006
Max Scale y Error	0.0994	0.1032	0.0914
Translate x RMSE	0.21155	0.218318	0.2183
Min Trans x Error	0.0169	0.0017	0.0136
Max Trans x Error	1.6451	1.798	1.1491
Translate y RMSE	0.14214	0.132591	0.14199
Min Trans y Error	0.0124	0.0007	0.011
Max Trans y Error	1.2207	1.782	1.5134

Table A.2 Training results for multi layer networks with 100 DCT coefficients inputs

Inputs (Matched Points*2)	60	80	100	120	140
Rotation RMSE	0.63171	0.618881	0.61166	0.58321	0.68439
Min Rot Error	0.0203	0.0063	0.016	0.009	0.0388
Max Rot Error	5.5438	6.3923	3.7244	3.6327	3.3585
Scale x RMSE	0.00277	0.002862	0.00271	0.00252	0.00293
Min Scale x Error	0.00004	0.00005	0.00004	0.0002	0.0001
Max Scale x Error	0.1072	0.1096	0.1063	0.1022	0.1115
Scale y RMSE	0.00318	0.003024	0.00286	0.00252	0.00296
Min Scale y Error	0.00004	0.00002	0.0001	0.0002	0.00004
Max Scale y Error	0.1196	0.1075	0.091	0.1082	0.106
Translate x RMSE	0.15064	0.155448	0.14689	0.14408	0.15079
Min Trans x Error	0.0154	0.0074	0.0025	0.0073	0.0047
Max Trans x Error	2.3262	4.7328	4.0166	3.3738	4.2494
Translate y RMSE	0.1559	0.144802	0.14364	0.18946	0.144
Min Trans y Error	0.0103	0.024	0.0023	0.0237	0.0022
Max Trans y Error	4.6081	2.6468	3.4884	3.7768	3.2754

Table A.3 Training Results for various number of matched key points as inputs

# Hidden Layers and neurons per layer	2 layers 44,25 neurons	3 layers 44,25,10 neurons	4 layers 44,25,10,5 neurons
Rotation RMSE	0.5808	0.582101	0.5862
Min Rot Error	0.0064	0.0747	0.0175
Max Rot Error	3.6845	3.1876	4.5901
Scale x RMSE	0.00215	0.002278	0.00238
Min Scale x Error	0.00002	0.0002	0.00004
Max Scale x Error	0.103	0.0932	0.1237
Scale y RMSE	0.00267	0.002472	0.0027
Min Scale y Error	0.00003	0.0005	0.0005
Max Scale y Error	0.1028	0.0961	0.1332
Translate x RMSE	0.14173	0.139784	0.12432
Min Trans x Error	0.013	0.0038	0.009
Max Trans x Error	3.6785	3.3261	4.3768
Translate y RMSE	0.12207	0.120756	0.13553
Min Trans y Error	0.0139	0.0057	0.0018
Max Trans y Error	3.8064	3.5681	5.1519

Table A. 4 Training Results for multi layer network s using 50 matched points

# Hidden Layers and neurons per layer	1 layer 44 neurons	1 layer 64 neurons	2 layers 44,25 neurons	2 layers 64,40 neurons	3 layers 44,25,10 neurons	3 layers 64,40,15 neurons	4 layers 44,25,10,5 neurons	4 layers 64,40,15,10 neurons
Rotation RMSE	0.50365	0.407522	0.37851	0.41553	0.37225	0.31729	0.32257	0.30617
Min Rot Error	0.016	0.1354	0.0047	0.0153	0.0298	0.0241	0.029	0.0044
Max Rot Error	1.1777	2.1478	2.388	2.6728	2.2978	2.0458	2.6574	2.2032
Scale x RMSE	0.00346	0.003161	0.00298	0.00327	0.00311	0.00345	0.00321	0.00312
Min Scale x Error	0.0016	0.0003	0.0007	0.0002	3.0E-06	0.00007	0.00056	5.0E-06
Max Scale x Error	0.1049	0.1057	0.1046	0.121	0.1052	0.0932	0.1402	0.1143
Scale y RMSE	0.00283	0.00253	0.00235	0.00189	0.00319	0.00296	0.00288	0.00289
Min Scale y Error	0.0003	0.0001	0.0001	0.0004	0.0001	0.00002	0.00007	0.00003
Max Scale y Error	0.1079	0.0929	0.1046	0.0962	0.0967	0.1004	0.1328	0.106
Translate x Error	0.24596	0.211979	0.20135	0.22295	0.20477	0.22494	0.21381	0.21125
Min Trans x Error	0.0058	0.0128	0.0578	0.0151	0.0071	0.0291	0.003	0.0114
Max Trans x Error	3.608	2.331	3.6316	2.0778	2.1517	2.2444	1.1239	2.3321
Translate y Error	0.12993	0.12949	0.10435	0.08334	0.15177	0.13745	0.1435	0.13396
Min Trans y Error	0.0163	0.0123	0.0322	0.0038	0.0003	0.0039	0.0033	0.0001
Max Trans y Error	2.456	3.9235	2.906	2.6755	1.7923	2.2056	2.4532	1.9981

Table A.5 Results for multi layer networks using DCT + Matched Keypoints

# Hidden Layers and neurons per layer	3 layers 64,40,15	4 layers 64,40,15,10
Rotation RMSE	0.53262	0.549909
Min Rot Error	0.0153	0.0357
Max Rot Error	2.0611	2.1608
Scale x RMSE	0.00311	0.002926
Min Scale x Error	0.00005	0.00003
Max Scale x Error	0.07104	0.0816
Scale y RMSE	0.00347	0.003416
Min Scale y Error	0.0014	0.0002
Max Scale y Error	0.0843	0.1014
Translate x RMSE	0.18651	0.00242
Min Trans x Error	0.0241	0.0051
Max Trans x Error	1.9379	2.0991
Translate y RMSE	0.1775	0.002301
Min Trans y Error	0.0107	0.0028
Max Trans y Error	2.1599	2.5454
Shear x RMSE	0.00256	0.182341
Min Shear x Error	0.0001	0.0051
Max Shear x Error	0.106	0.0923
Shear y RMSE	0.00251	0.1652
Min Shear y Error	0.00003	0.0004
Max Shear y Error	0.0805	0.0911

Table A.6 Training Results for DCT+Matched keypoints network with Shear added

# Hidden Layers and neurons per layer	1 layers 44 Neurons w/o Shear	3 layers 44,25,10 w/o Shear	1 layers 44 Neurons with Shear	3 layers 44,25,10 with Shear
Rotation RMSE	0.51299	0.474566	0.45552	0.46814
Min Rot Error	0.0058	0.0024	0.0603	0.0344
Max Rot Error	2.9566	3.6736	4.0504	3.4197
Scale x RMSE	0.00352	0.003282	0.00273	0.00318
Min Scale x Error	0.0001	0.0001	0.0001	0.0002
Max Scale x Error	0.1264	0.104	0.1046	0.1063
Scale y RMSE	0.00335	0.003085	0.00399	0.00409
Min Scale y Error	1.00E-06	0.00001	0.0003	0.00005
Max Scale y Error	0.0754	0.0924	0.1027	0.0956
Translate x RMSE	0.19451	0.177915	0.15666	0.16493
Min Trans x Error	0.0322	0.0067	0.0055	0.0117
Max Trans x Error	4.5288	3.5945	3.8598	2.7554
Translate y RMSE	0.16822	0.161794	0.13979	0.13081
Min Trans y Error	0.0322	0.0067	0.0055	0.0117
Max Trans y Error	4.4671	4.86	2.9693	3.811
Shear x RMSE	N/A	N/A	0.00312	0.00342
Min Shear x Error	N/A	N/A	0.00003	0.00008
Max Shear x Error	N/A	N/A	0.0871	1.00E-01
Shear y RMSE	N/A	N/A	0.00328	0.00326
Min Shear y Error	N/A	N/A	0.0056	0.00006
Max Shear y Error	N/A	N/A	0.1092	0.1054

Table A.7 Training Results for networks w/ 100 Wavelet Coefficients

# Hidden Layers and neurons per layer	1 layers 44 Neurons w/o Shear	3 layers 44,25,10 w/o Shear	1 layers 44 Neurons with Shear	3 layers 44,25,10 with Shear
Rotation RMSE	0.465823	0.390518	0.353873	0.335062
Min Rot Error	0.0352	0.0044	0.0853	0.0133
Max Rot Error	3.6886	3.0697	3.0318	3.4685
Scale x RMSE	0.00264	2.84E-03	0.00283	0.002263
Min Scale x Error	5.00E-04	0.00007	0.00005	0.0003
Max Scale x Error	0.1263	0.0818	0.1141	0.1032
Scale y RMSE	0.002667	2.09E-03	0.004289	0.003422
Min Scale y Error	2.00E-05	0.0007	0.0002	9.00E-05
Max Scale y Error	0.1008	9.95E-02	1.07E-01	0.0861
Translate x RMSE	0.1753	0.153819	0.115672	0.094454
Min Trans x Error	0.0204	0.0158	0.0058	0.0007
Max Trans x Error	3.2798	2.9164	3.6881	3.0503
Translate y RMSE	0.146354	1.36E-01	0.136868	0.095216
Min Trans y Error	6.60E-03	0.0007	0.0088	0.0013
Max Trans y Error	9.0226	2.3194	3.3853	2.9733
Shear x RMSE	N/A	N/A	0.003051	0.002525
Min Shear x Error	N/A	N/A	0.0003	0.0002
Max Shear x Error	N/A	N/A	0.1049	1.09E-01
Shear y RMSE	N/A	N/A	0.002835	0.002585
Min Shear y Error	N/A	N/A	0.0002	0.0004
Max Shear y Error	N/A	N/A	0.0956	0.1038

Table A.8 Training Results for networks w/ 100 coeff + 50 matched points

# Wavelons	31 Wavelons	63 Wavelons
Rotation RMSE	0.0942	1.43E-01
Min Rot Error	0.0087	0.0658
Max Rot Error	0.4149	0.5592
Scale x RMSE	0.0469	1.56E-02
Min Scale x Error	9.40E-03	0.0102
Max Scale x Error	0.0668	0.0522
Scale y RMSE	0.0351	0.0552
Min Scale y Error	9.90E-03	0.005
Max Scale y Error	0.0886	0.0613
Translate x RMSE	0.1081	1.24E-01
Min Trans x Error	0.0538	0.1018
Max Trans x Error	0.6926	0.4911
Translate y RMSE	0.091	0.1361
Min Trans y Error	0.1230	0.1275
Max Trans y Error	0.7095	0.8205

Table A.9 Training Results for WNN using Matched Keypoints

# Wavelons	31 Wavelons	63 Wavelons
Rotation RMSE	0.007416	0.006529
Min Rot Error	0.0209	0.0373
Max Rot Error	0.3386	0.2827
Scale x RMSE	0.000194	8.21E-04
Min Scale x Error	7.00E-04	0.0008
Max Scale x Error	0.0188	0.0241
Scale y RMSE	0.00051	7.33E-04
Min Scale y Error	0.0002	0.0002
Max Scale y Error	0.0189	0.0194
Translate x RMSE	0.009351	0.006584
Min Trans x Error	0.0001	0.0001
Max Trans x Error	0.2299	0.1024
Translate y RMSE	0.00475	7.34E-03
Min Trans y Error	0.0008	0.0001
Max Trans y Error	0.3718	0.1335

Table A.10 Training Results for WNN using DCT+Matched Keypoints

# Wavelons	31 Wavelons	63 Wavelons
Rotation RMSE	0.074065	0.040263
Min Rot Error	0.0007	0.0005
Max Rot Error	0.0449	0.1548
Scale x RMSE	0.000901	0.001035
Min Scale x Error	0.0001	0.0001
Max Scale x Error	0.0072	0.0048
Scale y RMSE	0.001423	0.001187
Min Scale y Error	0.0001	0.0001
Max Scale y Error	0.0056	0.0076
Translate x RMSE	0.059625	0.065552
Min Trans x Error	0.0008	0.0004
Max Trans x Error	0.1505	0.1668
Translate y RMSE	0.057019	0.053251
Min Trans y Error	0.0013	0.0024
Max Trans y Error	0.1985	0.1448
Shear x RMSE	0.003546	0.004257
Min Shear x Error	0.0011	0.0042
Max Shear x Error	0.0094	0.0144
Shear y RMSE	0.007041	0.006208
Min Shear y Error	0.001	0.0012
Max Shear y Error	0.0104	0.0096

Table A.11 Training Results for WNN using DCT+Keypoints with shearing added

# Hidden Layers and neurons per layer	3 layers 64,40,15 w/o Shear	3 layers 64,40,15 w/ Shear
Rotation RMSE	0.38655	0.453151
Min Rot Error	0.0288	0.0384
Max Rot Error	3.7659	3.7618
Scale x RMSE	0.001463	0.00127
Min Scale x Error	0.00004	0.0006
Max Scale x Error	0.0904	0.0789
Scale y RMSE	0.003281	0.003242
Min Scale y Error	0.0002	0.00003
Max Scale y Error	0.0911	0.1063
Translate x RMSE	0.140136	0.14151
Min Trans x Error	0.0171	0.0043
Max Trans x Error	2.081	2.3473
Translate y RMSE	0.121346	0.112545
Min Trans y Error	0.0051	0.06
Max Trans y Error	2.7381	2.8552
Shear x RMSE	N/A	0.002367
Min Shear x Error	N/A	0.0004
Max Shear x Error	N/A	0.0835
Shear y RMSE	N/A	0.002247
Min Shear y Error	N/A	0.00001
Max Shear y Error	N/A	0.1025

Table A.12 FFNN trained w/ 100 coeffs + 50 matched points w/ Noise

# Wavelons	31 Wavelons w/o Shear	31 Wavelons w/ Shear
Rotation RMSE	0.004187	0.006057
Min Rot Error	0.0002	0.0001
Max Rot Error	0.0293	0.0091
Scale x RMSE	0.003128	0.004361
Min Scale x Error	0.0002	0.0001
Max Scale x Error	0.0192	0.0054
Scale y RMSE	0.004468	0.003316
Min Scale y Error	0.0001	0.0001
Max Scale y Error	0.0151	0.0039
Translate x RMSE	0.018933	0.013537
Min Trans x Error	5.62E-11	0.0004
Max Trans x Error	0.0504	0.0781
Translate y RMSE	0.019421	0.010519
Min Trans y Error	0.0001	0.0002
Max Trans y Error	0.6963	0.0968
Shear x RMSE	N/A	0.001897
Min Shear x Error	N/A	0.0001
Max Shear x Error	N/A	0.0083
Shear y RMSE	N/A	0.004159
Min Shear y Error	N/A	0.0001
Max Shear y Error	N/A	0.0076

Table A.13 Wavelet networks w/ 100 coeffs+50 matched points w/ Noise

Source Image	Fig 4.9a w/o Shear	Fig 4.9a w/ Shear	Fig 4.9b w/o Shear	Fig 4.9b w/ Shear
Rotation RMSE	0.257657	0.294374	0.379634	0.376277
Min Rot Error	0.0138	0.0126	0.0081	0.0873
Max Rot Error	2.3583	2.2944	2.8866	2.5074
Scale x RMSE	0.001963	0.002001	0.002878	0.003156
Min Scale x Error	0.00008	0.0002	0.004	0.0005
Max Scale x Error	0.08	0.0816	0.1166	0.1394
Scale y RMSE	0.001987	0.001954	0.001438	0.002895
Min Scale y Error	0.0013	0.0023	0.0006	0.0003
Max Scale y Error	0.0988	0.1037	0.0832	0.1087
Translate x RMSE	0.113491	0.124354	0.198817	0.114252
Min Trans x Error	0.0155	0.0053	0.0029	0.0075
Max Trans x Error	3.1442	3.5141	3.2146	3.7109
Translate y RMSE	0.106358	0.104427	0.068027	0.167348
Min Trans y Error	0.0222	0.0046	0.0137	0.0042
Max Trans y Error	3.3849	2.8918	3.5263	3.4019
Shear x RMSE	N/A	0.002402	N/A	0.001646
Min Shear x Error	N/A	0.00001	N/A	0.00007
Max Shear x Error	N/A	0.1033	N/A	0.1083
Shear y RMSE	N/A	0.002168	N/A	0.001843
Min Shear y Error	N/A	0.0046	N/A	0.0003
Max Shear y Error	N/A	0.0954	N/A	0.1062

Table A.14 FFNN w/ 64, 40, 15 hidden neurons Training results (no noise)

Source Image	Fig 4.9a w/o Shear	Fig 4.9a w/ Shear	Fig 4.9b w/o Shear	Fig 4.9b w/ Shear
Rotation RMSE	0.391216	0.432142	0.355057	0.461151
Min Rot Error	0.0117	0.019	0.006	0.0468
Max Rot Error	3.3741	3.5312	3.4122	3.5631
Scale x RMSE	0.002328	0.00099	0.002153	0.001847
Min Scale x Error	0.00005	7.00E-06	0.0007	0.00003
Max Scale x Error	0.0917	0.074	0.1099	0.1254
Scale y RMSE	0.001419	0.003092	0.002676	0.003168
Min Scale y Error	0.00006	0.0002	0.0004	0.0013
Max Scale y Error	0.09284	0.1053	0.0803	0.1087
Translate x RMSE	0.014703	0.158813	0.174546	0.146414
Min Trans x Error	0.0072	0.0112	0.0014	0.0058
Max Trans x Error	3.6159	3.638	3.3392	3.0595
Translate y RMSE	0.070252	0.109617	0.122189	0.113335
Min Trans y Error	0.0161	0.0228	0.0142	0.001
Max Trans y Error	3.687	3.9109	3.8263	4.1039
Shear x RMSE	N/A	0.002436	N/A	0.001711
Min Shear x Error	N/A	0.0001	N/A	0.0003
Max Shear x Error	N/A	0.0919	N/A	0.108
Shear y RMSE	N/A	0.002128	N/A	0.001885
Min Shear y Error	N/A	0.0006	N/A	0.0004
Max Shear y Error	N/A	0.1036	N/A	0.0917

Table A.15 FFNN w/ 64, 40, 15 hidden neurons Training results (with noise)

Source Image	Fig 4.9a w/o Shear	Fig 4.9a w/ Shear	Fig 4.9b w/o Shear	Fig 4.9b w/ Shear
Rotation RMSE	0.143288	0.174623	0.015087	0.083877
Min Rot Error	0.0025	0.0164	0.0041	0.0862
Max Rot Error	0.2661	0.4101	0.0169	0.2699
Scale x RMSE	0.008399	0.004025	0.001607	0.0166
Min Scale x Error	0.0001	0.0014	0.0004	0.0004
Max Scale x Error	0.0165	0.0105	0.0014	0.0352
Scale y RMSE	0.008054	0.004606	0.001581	0.00841
Min Scale y Error	0.0004	3.54E-05	4.36E-05	0.0012
Max Scale y Error	0.014	0.0087	0.0015	0.0173
Translate x RMSE	0.070853	0.188551	0.057919	0.092641
Min Trans x Error	0.0009	0.0006	0.0012	0.0002
Max Trans x Error	0.1261	0.2438	0.0069	0.2211
Translate y RMSE	0.066491	0.18031	0.028329	0.095501
Min Trans y Error	0.00055	0.00758	0.0037	0.0273
Max Trans y Error	0.1327	0.3055	0.0269	0.4779
Shear x RMSE	N/A	0.007931	N/A	0.0079
Min Shear x Error	N/A	0.00011	N/A	0.0014
Max Shear x Error	N/A	0.0113	N/A	0.0301
Shear y RMSE	N/A	0.003148	N/A	0.0095
Min Shear y Error	N/A	0.0001	N/A	0.0004
Max Shear y Error	N/A	0.0147	N/A	0.0375

Table A.16 Training results for Wavelet network w/ 31 wavelons (no noise)

Source Image	Fig 4.9a w/o Shear	Fig 4.9a w/ Shear	Fig 4.9b w/o Shear	Fig 4.9b w/ Shear
Rotation RMSE	0.082091	0.100774	0.025336	0.069518
Min Rot Error	2.39E-06	0.0202	0.0009	0.0001
Max Rot Error	0.0214	0.2018	0.1964	0.0258
Scale x RMSE	0.040003	0.003316	0.025477	0.040966
Min Scale x Error	0.0001	0.0001	0.0003	0.0003
Max Scale x Error	0.0483	0.0035	0.0104	0.073
Scale y RMSE	0.003744	0.004361	0.008084	2.76E-03
Min Scale y Error	0.0002	1.80E-05	0.0002	6.78E-06
Max Scale y Error	0.0435	0.0033	0.0092	0.009
Translate x RMSE	0.035691	0.042808	0.055627	0.095658
Min Trans x Error	7.06E-11	0.0003	0.004	0.0163
Max Trans x Error	0.0049	0.0832	0.0866	0.2402
Translate y RMSE	0.067096	0.033266	0.012357	0.044181
Min Trans y Error	4.133 e-5	0.0028	0.0005	5.48E-02
Max Trans y Error	0.0182	0.0738	0.1008	0.1823
Shear x RMSE	N/A	0.001897	N/A	0.009566
Min Shear x Error	N/A	0.0001	N/A	1.38E-06
Max Shear x Error	N/A	0.0016	N/A	0.0271
Shear y RMSE	N/A	0.004159	N/A	0.008863
Min Shear y Error	N/A	8.62E-05	N/A	0.0002
Max Shear y Error	N/A	0.0033	N/A	0.0911

Table A.17 Training results for Wavelet network w/ 31 wavelons (with noise)

# Hidden Layers and neurons per layer	3 layers 44,25,10 neurons
Rotation RMSE	3.3746
Min Rot Error	0.4878
Max Rot Error	9.3178
Scale x RMSE	0.0439
Min Scale x Error	0.0145
Max Scale x Error	0.1292
Scale y RMSE	0.0383
Min Scale y Error	0.018
Max Scale y Error	0.1167
Translate x RMSE	1.476
Min Trans x Error	0.2727
Max Trans x Error	4.9343
Translate y RMSE	0.9743
Min Trans y Error	0.2381
Max Trans y Error	6.9344

Table A.18 Training Results for FFNN trained with 25 Strongest Edges

Appendix 2: Matlab and C-Code

```

function [ layers, errors, layer_output, param ] = bwnn3( inputs,
outputs, num_hidden_neurons, eta )
%Function: Wavelet Neural Network
%Author: Pramod Gadde
%Description: This function takes in the inputs and outputs, number of
%hidden neurons, and a learning rate. It outputs the average error, the
%final output, each layer with the weights, and the remaining
parameters
%(scale and translation factors)

tic
% rand('seed',1);
% randn('seed',1);
if num_hidden_neurons ~= 0
    total_layers = size(num_hidden_neurons,2) + 2;
else
    total_layers = 2;
end
for i=1:size(num_hidden_neurons,2)
    if i==1 && num_hidden_neurons ~= 0
        layers{1} = rand(num_hidden_neurons(i), size(inputs,1));
    elseif i==1 && num_hidden_neurons == 0
        layers{1} = rand(size(outputs,1),size(inputs,1));
    else
        layers{i} = rand(num_hidden_neurons(i), num_hidden_neurons(i-
1));
    end
end
if num_hidden_neurons ~= 0
    layers{i+1} = rand(size(outputs,1),num_hidden_neurons(i));
end
d1 = min(min(inputs)); d2 = max(max(inputs)); d1=0; d2=1;
for i=1:num_hidden_neurons
    if i==1
        a=rand(1);
        t(i) = (d2-d1)*a; lambda(1) = .5*(d2-d1);
    elseif mod(i,2) == 0
        t(i) = t(i/2)*rand(1); lambda(i) = .5*(t(i/2)-d1);
    elseif mod(i,2) == 1
        t(i) = d2-t((i-1)/2)*rand(1); lambda(i) = .5*(d2-t((i-1)/2));
    end
end

psi =0; rprev = 100;
for m=1:15000
    % disp({'count', m});
    if mod(m,250) == 0
        toc;
        [ rawr ] = bptest2( inputs, layers, t, lambda, psi );
        rerr = abs(rawr-outputs);
        rmerr4 = mean(rerr'); disp({'top lvl bwnn3', m}); disp(rmerr4)
    end
    for n=1:size(inputs,2)
        tin = inputs(:,n); tout = outputs(:,n);
    end
end

```



```

if num_hidden_neurons == 0
    tin2=tin;
end
for i=1:total_layers-1
    if i==1
        layer_output{1} = layers{1}*tin;
    else
        layer_output{i} = layers{i}*layer_output{i-1};
    end
    if i==1
        u3 = (layer_output{i}'-t)./lambda; u3=u3';
        psi = exp(-u3.^2/2).*cos(5*u3);
        psiDer = exp(-u3.^2/2).*(sin(5*u3)+u3.*cos(5*u3));
        gradTran2 = 1./lambda.*psiDer';
        gradSca2 = (u3'./lambda).*psiDer';
        layer_output{i} = psi;
    end
end
%work out the error for each neuron (output)
for i=1:total_layers-1
    if i==1
        errors{total_layers-i} = tout-
layer_output{total_layers-i};
        if num_hidden_neurons ~= 0
            a=layers{total_layers-
i}+eta*(layer_output{total_layers-i-1}*errors{total_layers-i}');
            b = t - eta*errors{total_layers-
i}'*layers{2}.*gradTran2;
            c = lambda - 1*errors{total_layers-
i}'*layers{2}.*gradSca2;
        end
        layers{total_layers-1} = a;
        [ rawr ] = bptest2( inputs, layers, b, lambda, psi );
        rerr = abs(rawr-outputs); rmerr4 = mean(rerr');
        if rmerr4<rprev
            t = b;
            rprev = rmerr4;
        end
        [ rawr ] = bptest2( inputs, layers, t, c, psi );
        rerr = abs(rawr-outputs); rmerr4 = mean(rerr');
        if rmerr4<rprev
            lambda = c;
            rprev = rmerr4;
        end
    end

else
    B = layer_output{total_layers-i};
    C = errors{total_layers-i+1};
    D = layers{total_layers-i+1};
    ero = ones(size(D,2),1);
    for j=1:size(D,2)
        ero(j,1) = C'*D(:,j);
    end
    ero = psiDer.*ero;
    errors{total_layers-i} = ero;
    if (total_layers-i-1 <= 0)

```

```

                                a=layers{total_layers-
i}+eta*(tin*errors{total_layers-i}');
                                else
                                a=layers{total_layers-
i}+eta*(layer_output{total_layers-i-1}*errors{total_layers-i}');
                                end
                                ltemp = layers; ltemp{total_layers-i} = a;
                                [ rawr ] = bptest2( inputs, ltemp, t, lambda, psi );
                                rerr = abs(rawr-outputs); rmerr4 = mean(rerr');
                                if rmerr4 < rprev
                                    layers{total_layers-i} = a;
                                    rprev = rmerr4;
                                end
                                % layers{total_layers-i} = a;
                                end
                                end
                                end
                                end
                                toc
                                param{1} = t; param{2} = lambda; param{3} = psi;
                                end

```

```

%Script: Create training variables
%Author: Pramod Gadde
%Description: This script creates the input variables used for the
wavelet
%network

clearvars x xd zz z R z1 z2 S1 z3 z4 S4 trueScale trueShear trueRot
clearvars tform tdct td2 outIm param transl paramTest tran tran2
clearvars trueScaleTest TrueShearTest trueRotTest origCords tranCords
clearvars inputs testinputs outputs testoutputs
% clear
clc
tic
rand('seed',1);
randn('seed',1);
% x=imread('brain4.png');
% x = imread('brain3.jpg');
x = imread('brain2.jpg');
if length(size(x))==3
    x=rgb2gray(x);
end
x=double(x);
x=x/255;
x=imnoise(x,'gaussian');
x=x*255;
cd1 = 50;
xd = dct2(x); xd = xd(1:12,1:12); xd = reshape(xd,[144,1]);
zz = [300 144 3];
for i=1:25
    %Rotation
    z= .4189*rand(1,100)-.2094;
    R = [cos(z) sin(z) 0; -sin(z) cos(z) 0; 0 0 1];
    trueRot(i) = z;
    %Scaling
    z1 = .9+.2*rand(1,1);
    z2 = .9+.2*rand(1,1);
    %    z1 = .5+rand(1,1); z2 = .5+rand(1,1);
    S1 = [z1 0 0; 0 z2 0; 0 0 1];
    trueScale = [z1 z2];
    %Shearing
    z3 = -.1+.2*rand(1,1);
    z4 = -.1+.2*rand(1,1);
    %    z3 = -.5 + rand(1,1); z4 = -.5+rand(1,1);
    S2 = [1 z3 0; z4 1 0; 0 0 1];
    trueShear = [z3 z4];
    t1 = randi(11)-6;
    t2 = randi(11)-6;
    %    t1 = randi(100); t2 = randi(100);
    S3 = [1 0 0; 0 1 0; t1 t2 1];
    %Rotation, Scaling, Shearing
    tform = maketform('affine', R*S1);
    %    tform = maketform('affine', R*S1*S2);
    [transl xdata ydata] = imtransform(x,tform, 'XYScale', 1);
    %blur
    %    h = fspecial('gaussian',25,3); blur = imfilter(transl,h);
    %    transl = blur;
    %    figure; imshow(blur,[]);

```

```

    %translation
    tform1 = maketform('affine', S3);
    [transl xdata1 ydata1] = imtransform(transl, tform1, 'XYScale',1,
'XData', [1 size(transl,2)+t1], 'YData', [1 size(transl,1)+t2]);
    disp('test bro'); disp(i);
    tdct = dct2(transl);
    tdct = tdct(1:9,1:9);
    tdct = reshape(tdct,[81,1]);
    td2 = tdct;
    temp = td2;
%     inputs(:,i) = td2;
    [num,imCord1,imCord2,imCord3]=match(x,transl);
    SS3(i) = size(imCord2,1);
    delIm = imCord2(1:cd1,:);
    delIm = reshape(delIm',[100 1]);
    temp(82:181,1) = delIm;
%     temp = delIm;
    inputs(:,i) = temp;
    outputs(:,i) = [(z+12)/24 (z1-.9)/.2 (z2-.9)/.2 (t1+5)/10
(t2+5)/10];
%     outputs(:,i) = [(z+12)/24 (z1-.9)/.2 (z2-.9)/.2 (z3+.1)/.2
(z4+.1)/.2 (t1+5)/10 (t2+5)/10];
%     outputs(:,i) = [(z)/180 (z1-.5) (z2-.5) (z3+.5) (z4+.5) (t1)/100
(t2)/100];
%     outputs(:,i) = [(z)/180 (z1-.5) (z2-.5) (z3+.5) (z4+.5) (z3+.5)
(z4+.5) (t1)/100 (t2)/100];
end

for i=1:5
%Rotation
    z= .4189*rand(1,100)-.2094;
    R = [cos(z) sin(z) 0; -sin(z) cos(z) 0; 0 0 1];
    trueRot(i) = z;
%Scaling
    z1 = .9+.2*rand(1,1);
    z2 = .9+.2*rand(1,1);
%     z1 = .5+rand(1,1); z2 = .5+rand(1,1);
    S1 = [z1 0 0; 0 z2 0; 0 0 1];
    trueScale = [z1 z2];
%Shearing
    z3 = -.1+.2*rand(1,1);
    z4 = -.1+.2*rand(1,1);
%     z3 = -.5 + rand(1,1); z4 = -.5+rand(1,1);
    S2 = [1 z3 0; z4 1 0; 0 0 1];
    trueShear = [z3 z4];
    t1 = randi(11)-6;
    t2 = randi(11)-6;
%     t1 = randi(100); t2 = randi(100);
    S3 = [1 0 0; 0 1 0; t1 t2 1];
    %Rotation, Scaling, Shearing
    tform = maketform('affine', R*S1);
%     tform = maketform('affine', R*S1*S2);
    [transl xdata ydata] = imtransform(x,tform, 'XYScale', 1);
%blur
%     h = fspecial('gaussian',25,3); blur = imfilter(transl,h);
%     transl = blur;
%     figure; imshow(blur,[]);

```

```

%translation
tform1 = maketform('affine', S3);
[transl xdata1 ydata1] = imtransform(transl, tform1, 'XYScale',1,
'XData', [1 size(transl,2)+t1], 'YData', [1 size(transl,1)+t2]);
disp('test bro'); disp(25+i);
tdct = dct2(transl);
tdct = tdct(1:9,1:9);
tdct = reshape(tdct,[81,1]);
td2 = tdct;
temp = td2;
[num,imCord1,imCord2,imCord3]=match(x,transl);
SS4(i) = size(imCord2,1);
delIm = imCord2(1:cd1,:);
delIm = reshape(delIm',[100 1]);
temp(82:181,1) = delIm;
%     temp = delIm;
testinputs(:,i) = temp;
testoutput(:,i) = [(z+12)/24 (z1-.9)/.2 (z2-.9)/.2 (t1+5)/10
(t2+5)/10];
%     testoutput(:,i) = [(z+12)/24 (z1-.9)/.2 (z2-.9)/.2 (z3+.1)/.2
(z4+.1)/.2 (t1+5)/10 (t2+5)/10];
%     outputs(:,i) = [(z)/180 (z1-.5) (z2-.5) (z3+.5) (z4+.5) (t1)/100
(t2)/100];
%     outputs(:,i) = [(z)/180 (z1-.5) (z2-.5) (z3+.5) (z4+.5) (z3+.5)
(z4+.5) (t1)/100 (t2)/100];
end
clearvars R S1 S2 S3 i t1 t2 td2 tdct tform x xd xdata ydata z z1 z2 z3
z4 zz transl
clearvars trueRot trueRotTest trueScale trueScaleTest trueShear
trueShearTest
toc

```

```

%Script: Create training files
%Author: Pramod Gadde
%Description: This script creates the input files used for training for
%FANN

clearvars x xd zz z R z1 z2 S1 z3 z4 S4 trueScale trueShear trueRot
clearvars tform tdct td2 outIm param transl paramTest
clearvars trueScaleTest TrueShearTest trueRotTest origCords tranCords
clc
tic
rand('seed',1);
randn('seed',1);
x = imread('brain2.jpg');
x=rgb2gray(x);
x=double(x);
x=x/255;
x=imnoise(x,'gaussian');
x=x*255;
sz1 = 9; cd1 = 50; prm=7;
fid = fopen('dataTrialInp407.txt', 'w');
fil{1} = 'dataTrialInp407.txt';
fidTest = fopen('dataTrialOutRot407.txt', 'w');
fil{2} = 'dataTrialOutRot407.txt';
fidPar = fopen('dataTrialPar407.txt', 'w');
fil{3} = 'dataTrialPar407.txt';
zz = [300 sz1*sz1+2*cd1 prm];
fprintf(fid, '%i ', zz);
fprintf(fid, '\n%c', ''); l=0;
for i=1:300
    %Rotation
    z= .4189*rand(1,100)-.2094;
    R = [cos(z) sin(z) 0; -sin(z) cos(z) 0; 0 0 1];
    trueRot(i) = z;
    %Scaling
    z1 = .9+.2*rand(1,1);
    z2 = .9+.2*rand(1,1);
    %    z1 = .5+rand(1,1); z2 = .5+rand(1,1);
    S1 = [z1 0 0; 0 z2 0; 0 0 1];
    trueScale = [z1 z2];
    %Shearing
    z3 = -.1+.2*rand(1,1);
    z4 = -.1+.2*rand(1,1);
    %    z3 = -.5 + rand(1,1); z4 = -.5+rand(1,1);
    S2 = [1 z3 0; z4 1 0; 0 0 1];
    trueShear = [z3 z4];
    t1 = randi(11)-6;
    t2 = randi(11)-6;
    %Rotation, Scaling, Shearing
    %    tform = maketform('affine', R*S1);
    tform = maketform('affine', R*S1*S2);
    [transl xdata ydata] = imtransform(x,tform, 'XYScale', 1);
    %translation
    tform1 = maketform('affine', S3);
    [transl xdata1 ydata1] = imtransform(transl, tform1, 'XYScale',1,
'XData', [1 size(transl,2)+t1], 'YData', [1 size(transl,1)+t2]);
    disp('test bro'); disp(i);
    [num,imCord1,imCord2,imCord3]=match(x,transl);

```

```

SS(i) = size(imCord2,1);
delIm = imCord2(1:cd1,:);
tdct = dct2(transl);
tdct = tdct(1:sz1,1:sz1); tdct = reshape(tdct,[sz1*sz1,1]);
td2 = tdct;
fprintf(fid,'%f ', td2);
fprintf(fid,'%f ', delIm');
fprintf(fid, '\n%c', '');
%   param(:,i) = [z/20 (z1-.9)/.2 (z2-.9)/.2];
%   param(:,i) = [z/20 (z1-.9)/.2 (z2-.9)/.2 t1/5 t2/5];
param(:,i) = [z/20 (z1-.9)/.2 (z2-.9)/.2 (z3+.1)/.2 (z4+.1)/.2 t1/5
t2/5];
fprintf(fid,'%f ', param(:,i));
fprintf(fid, '\n%c', '');
end

for i=1:25
    %Rotation
    z= .4189*rand(1,100)-.2094;
    R = [cos(z) sin(z) 0; -sin(z) cos(z) 0; 0 0 1];
    trueRot(i) = z;
    %Scaling
    z1 = .9+.2*rand(1,1);
    z2 = .9+.2*rand(1,1);
    %   z1 = .5+rand(1,1); z2 = .5+rand(1,1);
    S1 = [z1 0 0; 0 z2 0; 0 0 1];
    trueScale = [z1 z2];
    %Shearing
    z3 = -.1+.2*rand(1,1);
    z4 = -.1+.2*rand(1,1);
    %   z3 = -.5 + rand(1,1); z4 = -.5+rand(1,1);
    S2 = [1 z3 0; z4 1 0; 0 0 1];
    trueShear = [z3 z4];
    t1 = randi(11)-6;
    t2 = randi(11)-6;
    S3 = [1 0 0; 0 1 0; t1 t2 1];
    %Rotation, Scaling, Shearing
    %   tform = maketform('affine', R*S1);
    tform = maketform('affine', R*S1*S2);
    [transl xdata ydata] = imtransform(x,tform, 'XYScale', 1);
    %translation
    tform1 = maketform('affine', S3);
    [transl xdata1 ydata1] = imtransform(transl, tform1, 'XYScale',1,
'XData', [1 size(transl,2)+t1], 'YData', [1 size(transl,1)+t2]);
    disp('test bro'); disp(300+i);
    [num,imCord1,imCord2,imCord3]=match(x,transl);
    SS2(i) = size(imCord2,1);
    delIm = imCord2(1:cd1,:);
    tdct = dct2(transl);
    tdct = tdct(1:sz1,1:sz1); tdct = reshape(tdct,[sz1*sz1,1]);
    td2 = tdct;
    fprintf(fidTest,'%f ', td2);
    fprintf(fidTest,'%f ', delIm');
    fprintf(fidTest, '\n%c', '');
    %   paramTest(:,i) = [z/20 (z1-.9)/.2 (z2-.9)/.2];
    %   paramTest(:,i) = [z/20 (z1-.9)/.2 (z2-.9)/.2 t1/5 t2/5];

```

```

        paramTest(:,i) = [z/20 (z1-.9)/.2 (z2-.9)/.2 (z3+.1)/.2 (z4+.1)/.2
t1/5 t2/5];
        fprintf(fidPar,'%f ', paramTest(:,i));
        fprintf(fidPar, '\n%c', '');
    end
    fclose('all');
    movefile(fil{1}, '../TestFiles/');
    movefile(fil{2}, '../TestFiles/');
    movefile(fil{3}, '../TestFiles/');
    clearvars R S1 S2 fid fidPar fidTest fil i num td2 tdct tform trueRot
    clearvars trueRotTest trueScale trueScaleTest trueShear trueShearTest
    clearvars xd xdata ydata z z1 z2 z3 z4 zz
    toc

```



```

%Script: This script is the Genetic Algorithm approach for registration
%Author: Pramod Gadde
%Description: Goes through 575 iterations. The fitness function is the
mean
%pixel error between the target and source images
tic
clc
rand('seed',1);
randn('seed',1);
for countTop = 1:10
    if countTop == 3
        disp('COUNT TOP=3');
    end
    clearvars rot sca she Y xdata ydata Y Y2 temp diff diff2 diff3
    clearvars a b sd diff4 i n r tempC parent1 parent2 pind1 pind2
    clearvars crossover_point child
    %randomize the chromosomes
    AffineTransTest2p1
    cntPar(:,countTop) = PP';
    IMG{countTop} = transl;
%    par(1,:) = randi(25, [1 100])-13; %rotation
    par(1,:) = .4189*rand(1,100)-.2094;
    par(2,:) = .9+.2*rand(1,100); %scaling
    par(3,:) = .9+.2*rand(1,100); %scaling
    par(4,:) = -.1+.2*rand(1,100); %shear
    par(5,:) = -.1+.2*rand(1,100); %shear
    par(6,:) = randi(11, [1 100])-6;
    par(7,:) = randi(11, [1 100])-6;
    count = 1;
    max_iterations = 200;
    for j=1:max_iterations
        %evaluate each one
        % for i=1:size(par,2)
        for i=1:100
            clearvars tform child
%            rot = [cos(par(1,i)*pi/180) sin(par(1,i)*pi/180) 0; -
sin(par(1,i)*pi/180) cos(par(1,i)*pi/180) 0; 0 0 1];
            rot = [cos(par(1,i)) sin(par(1,i)) 0; -sin(par(1,i))
cos(par(1,i)) 0; 0 0 1];
            sca = [par(2,i) 0 0; 0 par(3,i) 0; 0 0 1];
            she = [1 par(4,i) 0; par(5,i) 1 0; 0 0 1];
            tran = [1 0 0; 0 1 0; par(6,i) par(7,i) 1];
            tform = maketform('affine', rot*sca*she);
            [Y xdata ydata] = imtransform(x, tform, 'XYScale',1);
            tform1 = maketform('affine', tran);
            [Y xdata1 ydata1] = imtransform(Y, tform1, 'XYScale',1,
'XData', [1 size(Y,2)+t1], 'YData', [1 size(Y,1)+t2]);
            %            Y2{i} = imresize(Y, [size(transl,1), size(transl,2)]);
            %            temp = Y2{i} - transl;
            %            dumdum=0;
            if size(Y,1) < size(transl,1)
                Y(size(Y,1):size(transl,1),:) = 0;
            %            dumdum=1;
        end
    end

```

```

        if size(Y,2) < size(transl,2)
            Y(:,size(Y,2):size(transl,2)) = 0;
            dumdum=1;
        end
        Y2{i} = Y;
        temp = Y(1:size(transl,1),1:size(transl,2)) - transl;
        diff(i,1) = mean(mean(abs(temp)));
        dddiff(i,j) = mean(mean(abs(temp)));
        %         tdiff(i,countTop) = mean(mean(abs(temp)));
        %         if (dumdum==1)
        %             diff(i,j) = diff(i,j) + 5;
        %         end
    end
    mut=1;
    disp(strcat(int2str(countTop), ' test bro')); disp(j);
    [a b] = find(diff(:,1) == min(diff(:,1)));
    r1 = Y2{a};
    %     r1=2;
    clearvars Y2 diff2 sd diff3 dd1 diff4
    diff2 = 1./diff(:,1);
    diff2(:,2) = 1:100;
    sd = sum(diff2(:,1));
    diff3 = diff2(:,1)/sd;
    dd1 = diff3;
    dd1(:,2) = 1:100;
    dd1 = sortrows(dd1, -1);
    diff4(1) = diff3(1);
    for i=2:100
        diff4(i) = diff3(i)+diff4(i-1);
    end
    clearvars diff2 diff3 sd
    diff4 = diff4';
    if j==5
        z0{1,countTop} = par(:,a(1)); z0{2, countTop} = r1;
    end
    if j==25
        z1{1,countTop} = par(:,a(1)); z1{2,countTop} = r1;
    end
    if j==50
        z2{1,countTop} = par(:,a(1)); z2{2,countTop} = r1;
    end
    if j==100
        z3{1,countTop} = par(:,a(1)); z3{2,countTop} = r1;
    end
    if j==200
        z4{1,countTop} = par(:,a(1)); z4{2,countTop} = r1;
        %         z4=ch{j-1}; z4=z4'; z4=sortrows(z4);
    end
    if j==250
        %         z5{1,countTop} = par(:,a(1)); z5{2,countTop} = r1;
        %         z5=ch{j-1}; z5=z5'; z5=sortrows(z5);
    end
    child(:,1:10) = par(:,dd1(1:10,2));
    for i=11:35
        n=randi(35,1);
        parent1 = par(:,dd1(n,2));
        n=randi(35,1); %orig n=randi(65,1)+35;
    end
end

```

```

parent2 = par(:,dd1(n,2));
crossover_point = randi(4,1);
child(1:crossover_point,i) = parent1(1:crossover_point,1);
child(crossover_point+1:5,i) =
parent2(crossover_point+1:5,1);
if mut == 0
    n = .3*rand(1);
else
    n=rand(1);
end
if n<.15
    nn = randi(5,1);
    z = rand(1,1)-.5; z = floor(z)+ceil(z);
    if nn==1
        child(1,i) = mod(child(1,i) + z*randi(15,1),360);
    elseif nn==2
        child(2,i) = child(2,i) + z*.3*rand(1);
    elseif nn==3
        child(3,i) = child(3,i) + z*.3*rand(1);
    elseif nn==4
        child(4,i) = child(4,i) + z*.3*(rand(1));
    elseif nn==5
        child(5,i) = child(5,i) + z*.3*(rand(1));
    end
end
end
for i=36:100
    n = rand(1);
    tempC = abs(diff4 - n(1));
    [~, b] = min(tempC);
    parent1 = par(:,b);
    %     n = randi(100,1);
    %     parent1 = par(:,n);
    %     parent1 = par(:,dd1(n,2));
    n = rand(1);
    tempC = abs(diff4 - n(1));
    [~, b] = min(tempC);
    parent2 = par(:,b);
    %     n = randi(100,1);
    %     parent2 = par(:,n);
    %     parent2 = par(:,dd1(n,2));
    crossover_point = randi(4,1);
    cp = rand(1);
    if cp > .2
        child(1:crossover_point,i) =
parent1(1:crossover_point,1);
        child(crossover_point+1:5,i) =
parent2(crossover_point+1:5,1);
    else
        child(:,i) = parent1;
    end
    %     child(1:crossover_point,i) =
parent1(1:crossover_point,i);
    %     child(crossover_point+1:5,i) =
parent2(crossover_point+1:5,i);
    if mut == 0
        n = .3*rand(1);

```

```

else
    n=rand(1);
end
if n < .15
    nn = randi(5,1);
    z = rand(1,1)-.5; z = floor(z)+ceil(z);
    if nn==1
        child(1,i) = mod(child(1,i) + randi(15,1),360);
    elseif nn==2
        child(2,i) = child(2,i) + .3*z*rand(1);
    elseif nn==3
        child(3,i) = child(3,i) + .3*z*rand(1);
    elseif nn==4
        child(4,i) = child(4,i) +.3*z*(rand(1));
    elseif nn==5
        child(5,i) = child(5,i) +.3*z*(rand(1));
    end
end
end
par = child;
%     ch{j} = child;
end
clearvars b ch count cp diff2 diff3 diff4 i j mut n nn parent1
parent2
clearvars rot sca sd she temp tempC xdata ydata a crossover_point
dd1
end
toc

```

```

%Script: This script is to perform modifications for the original file.
%Author: Pramod Gadde
%Description: This script is called by several other files. It is used
to
%basically initialize the source image

clc
% clear
% close all
x = imread('brain2.jpg');
x=rgb2gray(x);
x=double(x);
% x=x/255;
% x=imnoise(x,'gaussian');
% xnoise = imnoise(x,'gaussian');
for i=1:1
    %Rotation
    %Rotation
    z = .4189*rand(1,100)-.2094;
    R = [cos(z) sin(z) 0; -sin(z) cos(z) 0; 0 0 1];
    trueRot(i) = z;
    %Scaling
    z1 = .9+.2*rand(1,1);
    z2 = .9+.2*rand(1,1);
    %    z1 = .5+rand(1,1); z2 = .5+rand(1,1);
    S1 = [z1 0 0; 0 z2 0; 0 0 1];
    trueScale = [z1 z2];
    %Shearing
    z3 = -.1+.2*rand(1,1);
    z4 = -.1+.2*rand(1,1);
    %    z3 = -.5 + rand(1,1); z4 = -.5+rand(1,1);
    S2 = [1 z3 0; z4 1 0; 0 0 1];
    trueShear = [z3 z4];
    t1 = randi(11)-6;
    t2 = randi(11)-6;
    tt = [1 0 0; 0 1 0; t1 t2 1];
    S3 = [1 0 0; 0 1 0; t1 t2 1];
    tf = R*S1*S2;
    %Rotation, Scaling, Shearing
    tform = maketform('affine', R*S1*S2);
    [transl xdata ydata] = imtransform(x,tform, 'XYScale', 1);
    %translation
    tform1 = maketform('affine', S3);
    [transl xdata1 ydata1] = imtransform(transl, tform1, 'XYScale',1,
'XData', [1 size(transl,2)+t1], 'YData', [1 size(transl,1)+t2]);
    %    figure; imshow(transl{i}, [], 'xdata', xdata, 'ydata', ydata);
    PP = [z z1 z2 z3 z4 t1 t2];
    outIm{i} = transl;
    outPar{i} = PP;
end
clearvars z1 z2 tform xdata ydata i z z3 z4

```

```

function [ layers ] = bwnn( inputs, outputs, num_hidden_neurons, eta )
%Function: This function performs is for the regular feedforward
network
%Author: Pramod Gadde
%Description: This function outputs the weights of each layer for the
%feedforward network trained with the inputs, outputs, for a given
learning
%rate eta and number of hidden neurons in each layer. Example: for a
%standard 3 layer network with 10 hidden neurons, num_hidden_neurons =
10.
%For a 4 layer network with 10 and 5 hidden neurons,
%num_hidden_neurons=[10,5]

tic
rand('seed',1);
randn('seed',1);
if num_hidden_neurons ~= 0
    total_layers = size(num_hidden_neurons,2) + 2;
else
    total_layers = 2;
end
for i=1:size(num_hidden_neurons,2)
    if i==1 && num_hidden_neurons ~= 0
        layers{1} = rand(num_hidden_neurons(i), size(inputs,1));
    elseif i==1 && num_hidden_neurons == 0
        layers{1} = rand(size(outputs,1),size(inputs,1));
    else
        layers{i} = rand(num_hidden_neurons(i), num_hidden_neurons(i-
1));
    end
end
if num_hidden_neurons ~= 0
    layers{i+1} = rand(size(outputs,1),num_hidden_neurons(i));
end

for m=1:1000
    if mod(m,3000) == 0
        toc; outputs;
    end
    for n=1:size(inputs,2)
        tin = inputs(:,n); tout = outputs(:,n);
        if num_hidden_neurons == 0
            % tin2 = 1./(1+exp(-1*(tin)));
            tin2=tin;
        end
        for i=1:total_layers-1
            if i==1
                layer_output{1} = layers{1}*tin;
            else
                layer_output{i} = layers{i}*layer_output{i-1};
            end
            layer_output{i} = 1./(1+exp(-1*(layer_output{i})));
        end
        %work out the error for each neuron (output)
        for i=1:total_layers-1
            if i==1

```

```

        B = layer_output{total_layers-i};
        errors{total_layers-i} = (B.*(1-B)).*(tout-B);
        if num_hidden_neurons ~= 0
            a=layers{total_layers-
i}+eta*(layer_output{total_layers-i-1}*errors{total_layers-i}');
            elseif num_hidden_neurons == 0
                a=layers{total_layers-
i}+eta*(tin2*errors{total_layers-i}');
            end
            layers{total_layers-1} = a;
        else
            B = layer_output{total_layers-i};
            C = errors{total_layers-i+1};
            D = layers{total_layers-i+1};
            ero = ones(size(D,2),1);
            for j=1:size(D,2)
                ero(j,1) = C'*D(:,j);
            end
            ero = (B.*(1-B)).*ero;
            errors{total_layers-i} = ero;
            if (total_layers-i-1 <= 0)
                a=layers{total_layers-
i}+eta*(tin*errors{total_layers-i}');
            else
                a=layers{total_layers-
i}+eta*(layer_output{total_layers-i-1}*errors{total_layers-i}');
            end
            layers{total_layers-i} = a;
        end
    end
end
toc
end

```

```

function [ outruns ] = bptest2( inputs, layers, t, lambda, psi )
%Function: This function produces the output for a wavlet network with
the given
%weights, scaling, and translation parameters
%Author: Pramod Gadde
total_layers = size(layers,2)+1;
for n=1:size(inputs,2)
    tin = inputs(:,n);
    for i=1:total_layers-1
        if i==1
            layer_output{1} = layers{1}*tin;
            u3 = (layer_output{i}'-t)./lambda; u3=u3';
            psi = exp(-u3.^2/2).*cos(5*u3);
            layer_output{i} = psi;
        else
            layer_output{i} = layers{i}*layer_output{i-1};
        end
        % layer_output{i} = 1./(1+exp(-1*(layer_output{i})));
    end
    outruns(:,n) = layer_output{i};
end

end

```



```

%Script: This script is used to create input files that have the
strongest
%edges from each image. Also wavelet decomposition is performed from
which
%the DCT coefficeints and edges are extracted from
%Author: Pramod Gadde

clearvars x xd zz z R z1 z2 S1 z3 z4 S4 trueScale trueShear trueRot
clearvars tform tdct td2 outIm param transl paramTest
clearvars trueScaleTest TrueShearTest trueRotTest origCords tranCords
clc
tic
rand('seed',1);
randn('seed',1);
x = imread('brain2.jpg');
x=rgb2gray(x);
x=double(x);
x2 = x/255;

% xd = dct2(x); xd = xd(1:6,1:6); xd = reshape(xd,[36,1]);
for i=1:3
    [cA1,cH1,cV1,cD1] = dwt2(x2,'haar');
    cod_X = wcodemat(x2);
    cod_cA1 = wcodemat(cA1); x2=cod_cA1;
    cod_cH1 = wcodemat(cH1);
    cod_cV1 = wcodemat(cV1);
    cod_cD1 = wcodemat(cD1);
    dec2d = [cod_cA1,      cod_cH1;cod_cV1,      cod_cD1 ];
    tt1{i} = dec2d;
    tt2{i} = cod_cV1; tt3{i} = cod_cH1; tt4{i} = cod_cD1;
    figure; imshow(dec2d,[]);
end

% xw = cod_cA1(1:6,1:6); xw = reshape(xw, [36,1]);
xw = cod_cA1;
gx = [-1 0 1; -2 0 2; -1 0 1]; rawr1 = conv2(x,gx, 'same');
gy = [1 2 1; 0 0 0; -1 -2 -1]; rawr2 = conv2(x,gy, 'same');
rawr3 = (rawr1.^2+rawr2.^2).^5;
rawr4 = (rawr3 - min(min(rawr3)))/(max(max(rawr3))-
min(min(rawr3)))*255;
rr = [1 1 1; 1 1 1; 1 1 1];
rawr5 = conv2(rawr4,rr, 'same');
% rawr5 = rawr4;
origEdge = rawr5;
[maxRowOrig maxColOrig] = find(rawr5 == max(max(rawr5)));
rawr5(maxRowOrig, maxColOrig) = 0;
[maxRowOrig2 maxColOrig2] = find(rawr5 == max(max(rawr5)));
rawr5(maxRowOrig2, maxColOrig2) = 0;
[maxRowOrig3 maxColOrig3] = find(rawr5 == max(max(rawr5)));
% rawr5(maxRowOrig, maxColOrig) = 0;
origCords = [maxRowOrig maxColOrig maxRowOrig2 maxColOrig2 maxRowOrig3
maxColOrig3];
origCords = origCords';
clearvars rawr1 rawr2 rawr3 rawr4 rawr5 maxRowOrig maxColOrig
clearvars maxRowOrig2 maxColOrig2 maxRowOrig3 maxColOrig3
fid = fopen('dataTrialInp25.txt', 'w');

```

```

fidTest = fopen('dataTrialOutRot25.txt', 'w');
fidPar = fopen('dataTrialPar25.txt', 'w');
zz = [300 36 5];
fprintf(fid, '%i ', zz);
fprintf(fid, '\n%c', '');

for i=1:300
    %Rotation
    z=randi(360);
    R = [cos(z*pi/180) sin(z*pi/180) 0; -sin(z*pi/180) cos(z*pi/180) 0;
0 0 1];
    trueRot(i) = z;
    %Scaling
    z1 = 1+.25*randn(1,1);
    if z1 > 1.75
        z1 = 1.75;
    elseif z1 < .25
        z1 = .25;
    end
    z2 = 1+.25*randn(1,1);
    if z2 > 1.75
        z2 = 1.75;
    elseif z2 < .25
        z2 = .25;
    end
    S1 = [z1 0 0; 0 z2 0; 0 0 1];
    trueScale = [z1 z2];
    %Shearing
    z3 = 0+.5*randn(1,1);
    if z3 > .75
        z3 = .75;
    elseif z3 < -.75
        z3 = -.75;
    end
    z4 = 0+.5*randn(1,1);
    if z4 > .75
        z4 = .75;
    elseif z4 < -.75
        z4 = -.75;
    end
    S2 = [1 z3 0; z4 1 0; 0 0 1];
    trueShear = [z3 z4];
    %Rotation, Scaling, Shearing
    tform = maketform('affine', R*S1*S2);
    % tform = maketform('affine', R*S1);
    [transl xdata ydata] = imtransform(x,tform);
    rawr1 = conv2(transl, gx, 'same');
    rawr2 = conv2(transl, gy, 'same');
    rawr3 = (rawr1.^2+rawr2.^2).^5;
    rawr4 = (rawr3 - min(min(rawr3)))/(max(max(rawr3))-
min(min(rawr3)))*255;
    rawr5 = conv2(rawr4,rr, 'same');
    % rawr5 = rawr4;
    tranEdge = rawr5;
    [maxRowTran maxColTran] = find(rawr5 == max(max(rawr5)));
    rawr5(maxRowTran, maxColTran) = 0;
    [maxRowTran2 maxColTran2] = find(rawr5 == max(max(rawr5)));

```

```

rawr5(maxRowTran2, maxColTran2) = 0;
[maxRowTran3 maxColTran3] = find(rawr5 == max(max(rawr5)));
% rawr5(maxRowOrig, maxColOrig) = 0;
tranCords = [maxRowTran maxColTran maxRowTran2 maxColTran2
maxRowTran3 maxColTran3];
tranCords = tranCords';
clearvars rawr1 rawr2 rawr3 rawr4 rawr5 maxRowTran maxColOrig
clearvars maxRowTran maxColOrig2 maxRowTran maxColOrig3
tr2 = transl;
for i=1:3
    [cA1,cH1,cV1,cD1] = dwt2(tr2,'db5');
    cod_X = wcodemat(tr2);
    cod_cA1 = wcodemat(cA1); tr2=cod_cA1;
    cod_cH1 = wcodemat(cH1);
    cod_cV1 = wcodemat(cV1);
    cod_cD1 = wcodemat(cD1);
    dec2d = [cod_cA1, cod_cH1; cod_cV1, cod_cD1 ];
    figure; imshow(dec2d,[]);
end

%     tw = cod_cA1(1:6,1:6); tw = reshape(tw, [36,1]);
%     tw2 = tw-xw;
tw = cod_cA1;
fprintf(fid,'%f ', tw);
%     fprintf(fid, '%f ', origCords);
%     fprintf(fid, '%f ', tranCords);
fprintf(fid, '\n%c', '');
%     param(:,i) = [z/360 (z1-.25)/1.5 (z2-.25)/1.5];
%     param(:,i) = [z/360 (z1-.25)/1.5 (z2-.25)/1.5 (z3+.75)/1.5
(z4+.75)/1.5];
%     param(:,i) = z/360;
fprintf(fid,'%f ', param(:,i));
fprintf(fid, '\n%c', '');
outIm{i} = transl;
end

for i=1:25
    %Rotation
    z=randi(360);
    R = [cosd(z) sind(z) 0; -sind(z) cosd(z) 0; 0 0 1];
    trueRotTest(i) = z;
    %Scaling
    z1 = 1+.5*randn(1,1);
    if z1 > 1.75
        z1 = 1.75;
    elseif z1 < .25
        z1 = .25;
    end
    z2 = 1+.5*randn(1,1);
    if z2 > 1.75
        z2 = 1.75;
    elseif z2 < .25
        z2 = .25;
    end
    S1 = [z1 0 0; 0 z2 0; 0 0 1];
    trueScaleTest = [z1 z2];
end

```

```

%Shearing
z3 = 0+.5*randn(1,1);
if z3 > .75
    z3 = .75;
elseif z3 < -.75
    z3 = -.75;
end
z4 = 0+.5*randn(1,1);
if z4 > .75
    z4 = .75;
elseif z4 < -.75
    z4 = -.75;
end
S2 = [1 z3 0; z4 1 0; 0 0 1];
trueShearTest = [z3 z4];
%Rotation, Scaling, Shearing
tform = maketform('affine', R*S1*S2);
%    tform = maketform('affine', R*S1);
[transl xdata ydata] = imtransform(x,tform);
rawr1 = conv2(transl, gx, 'same');
rawr2 = conv2(transl, gy, 'same');
rawr3 = (rawr1.^2+rawr2.^2).^5;
rawr4 = (rawr3 - min(min(rawr3)))/(max(max(rawr3))-
min(min(rawr3)))*255;
rawr5 = conv2(rawr4,rr, 'same');
%    rawr5=rawr4;
tranEdge = rawr5;
[maxRowTran maxColTran] = find(rawr5 == max(max(rawr5)));
rawr5(maxRowTran, maxColTran) = 0;
[maxRowTran2 maxColTran2] = find(rawr5 == max(max(rawr5)));
rawr5(maxRowTran2, maxColTran2) = 0;
[maxRowTran3 maxColTran3] = find(rawr5 == max(max(rawr5)));
% rawr5(maxRowOrig, maxColOrig) = 0;
tranCords = [maxRowTran maxColTran maxRowTran2 maxColTran2
maxRowTran3 maxColTran3];
tranCords = tranCords';
clearvars rawr1 rawr2 rawr3 rawr4 rawr5 maxRowTran maxColOrig
clearvars maxRowTran maxColOrig2 maxRowTran maxColOrig3

tr2 = transl;
for i=1:3
    [cA1,cH1,cV1,cD1] = dwt2(tr2,'db5');
    cod_X = wcodemat(tr2);
    cod_cA1 = wcodemat(cA1); tr2=cod_cA1;
    cod_cH1 = wcodemat(cH1);
    cod_cV1 = wcodemat(cV1);
    cod_cD1 = wcodemat(cD1);
    dec2d = [cod_cA1, cod_cH1; cod_cV1, cod_cD1];
    figure; imshow(dec2d,[]);
end
tw = cod_cA1(1:6,1:6); tw = reshape(tw, [36,1]);
tw2 = tw-xw;
fprintf(fidTest,'%f ', tw);
%    fprintf(fidTest, '%f ', origCords);
%    fprintf(fidTest, '%f ', tranCords);
fprintf(fidTest, '\n%c', '');
%    paramTest(:,i) = [z/360 (z1-.25)/1.5 (z2-.25)/1.5];

```

```

        paramTest(:,i) = [z/360 (z1-.25)/1.5 (z2-.25)/1.5 (z3+.75)/1.5
(z4+.75)/1.5];
        fprintf(fidPar,'%f ', paramTest(:,i));
        fprintf(fidPar, '\n%c', '');
        outImTest{i} = transl;
end

fclose('all');
toc

```

```

function [ outruns ] = bptest( inputs, layers )
%Function: This function outputs the output of a feedforward network
with
%the given input and layer weights
%Author: Pramod Gadde
total_layers = size(layers,2)+1;
for n=1:size(inputs,2)
    tin = inputs(:,n);
    for i=1:total_layers-1
        if i==1
            layer_output{1} = layers{1}*tin;
        else
            layer_output{i} = layers{i}*layer_output{i-1};
        end
        layer_output{i} = 1./(1+exp(-1*(layer_output{i})));
    end
    outruns(:,n) = layer_output{i};
end

end

```

```

#include <stdio.h>
#include "fann.h"
#include "floatfann.h"
#include <math.h>
int main()
{
    const unsigned int num_input = 181;
    const unsigned int num_output = 7;
    const unsigned int num_layers = 5;
    const float desired_error = (const float) 0.013;
    const unsigned int max_epochs = 500000;
    const unsigned int epochs_between_reports = 1000;

    struct fann *ann =
fann_create_standard(num_layers,num_input,64,40,15,num_output);
    fann_set_learning_rate(ann, .1);

    int fin = 300;
    int ninput = num_input;
    int noutput = num_output;
    fann_set_activation_function_hidden(ann, FANN_SIGMOID_SYMMETRIC);
    fann_set_activation_function_output(ann, FANN_SIGMOID_SYMMETRIC);

    fann_train_on_file(ann, "dataTrialInp551", max_epochs,
epochs_between_reports, desired_error);

    fann_save(ann, "trial1Out.net");

    fann_destroy(ann);
    float c;
    float tarr[ninput*fin]; /*36*25 = 900 48*25 = 1200 */
    int i=0;
    FILE *file;
    /*file = fopen("dataTrialOutRot141.txt", "r");*/
    file = fopen("trainInputFile551.txt", "r");
    if(file){
        while ((fscanf(file, "%f", &c))!=EOF){
            tarr[i] = c;
            i=i+1;
        }
        fclose(file);
    }
    float truth[noutput*fin];
    FILE *file2;
    /*file2 = fopen("dataTrialPar141.txt", "r");*/
    file2 = fopen("trainOutputFile551.txt", "r");
    i = 0;
    if (file2){
        while((fscanf(file2, "%f", &c))!=EOF){
            truth[i] = c;
            printf("\n%f", truth[i]);
            i=i+1;
        }
        fclose(file2);
    }
}

```

```

struct fann *ann2 = fann_create_from_file("trial1Out.net");
float netOut[noutput*25];
int j=1;
for (j=1; j<fin+1; j++){
    fann_type *calc_out;
    fann_type input[ninput];
    int ii = 0;
    for (i=(ninput*(j-1)); i<(ninput*j); i++){
        input[ii] = tarr[i];
        ii = ii+1;
    }
    calc_out = fann_run(ann2,input);
    /* netOut[j] = calc_out[0];*/
    if (noutput ==3){
        netOut[3*(j-1)] = calc_out[0];
        netOut[3*(j-1)+1] = calc_out[1];
        netOut[3*(j-1)+2] = calc_out[2];
        printf("IR test # %d (%f, %f) -> %f %f %f\n", j,
input[0], input[ninput-1], calc_out[0], calc_out[1], calc_out[2]);
    }
    else if (noutput == 5){
        netOut[5*(j-1)] = calc_out[0];
        netOut[5*(j-1)+1] = calc_out[1];
        netOut[5*(j-1)+2] = calc_out[2];
        netOut[5*(j-1)+3] = calc_out[3];
        netOut[5*(j-1)+4] = calc_out[4];
        printf("IR test # %d (%f, %f) -> %f %f %f %f %f\n", j,
input[0], input[ninput-1], calc_out[0], calc_out[1], calc_out[2],
calc_out[3], calc_out[4]);
    }
    else if (noutput == 7){
        netOut[7*(j-1)] = calc_out[0];
        netOut[7*(j-1)+1] = calc_out[1];
        netOut[7*(j-1)+2] = calc_out[2];
        netOut[7*(j-1)+3] = calc_out[3];
        netOut[7*(j-1)+4] = calc_out[4];
        netOut[7*(j-1)+5] = calc_out[5];
        netOut[7*(j-1)+6] = calc_out[6];
        printf("IR test # %d (%f, %f) -> %f %f %f %f %f %f %f\n",
j, input[0], input[ninput-1], calc_out[0], calc_out[1], calc_out[2],
calc_out[3], calc_out[4], calc_out[5], calc_out[6]);
    }
}

float errorRot = 0; float errorSca1 = 0; float errorSca2 = 0;
float terrRot = 0; float terrSca1 = 0; float terrSca2 = 0;
float terrShe1 = 0; float terrShe2 = 0;
float errorShe1 = 0; float errorShe2 = 0;
float terrTran1 = 0; float terrTran2 = 0;
float errorTran1 = 0; float errorTran2 = 0;
float maxRot = 0; float minRot = 10;
float maxSca1 = 0; float minSca1 = 10;
float maxSca2 = 0; float minSca2 = 10;
float maxShe1 = 0; float minShe1 = 10;
float maxShe2 = 0; float minShe2 = 10;
float maxTran1 = 0; float minTran1 = 10;
float maxTran2 = 0; float minTran2 = 10;

```



```

for (i=0; i<noutput*fin; i++){
    if (noutput == 3){
        if (i%noutput ==0){
            terrRot = pow(netOut[i] - truth[i],2);
            if (terrRot < 0){
                errorRot = errorRot - terrRot;
            }
            else{
                errorRot = errorRot + terrRot;
            }
        }
        else if(i%noutput == 1){
            terrSca1 = pow(netOut[i] - truth[i],2);
            if (terrSca1 < 0){
                errorSca1 = errorSca1 - terrSca1;
            }
            else{
                errorSca1 = errorSca1 + terrSca1;
            }
        }
        else if(i%noutput == 2){
            terrSca2 = pow(netOut[i] -truth[i],2);
            if(terrSca2 < 0){
                errorSca2 = errorSca2 - terrSca2;
            }
            else{
                errorSca2 = errorSca2 + terrSca2;
            }
        }
    }
    else if (noutput==5){
        if (i%noutput ==0){
            terrRot = pow(netOut[i] - truth[i],2);
            /*printf("TT Test # %d %f\n", i, pow(terrRot,.5)); */
            if (pow(terrRot,.5) > maxRot)
                maxRot = pow(terrRot,.5);
            else if (pow(terrRot,.5) < minRot)
                minRot = pow(terrRot,.5);
            if (terrRot < 0){
                errorRot = errorRot - terrRot;
            }
            else{
                errorRot = errorRot + terrRot;
            }
        }
        else if(i%noutput == 1){
            terrSca1 = pow(netOut[i] - truth[i],2);
            if (pow(terrSca1,.5) > maxSca1)
                maxSca1 = pow(terrSca1,.5);
            else if (pow(terrSca1,.5) < minSca1)
                minSca1 = pow(terrSca1,.5);
            if (terrSca1 < 0){
                errorSca1 = errorSca1 - terrSca1;
            }
            else{
                errorSca1 = errorSca1 + terrSca1;
            }
        }
    }
}

```

```

}
else if(i%noutput == 2){
    terrSca2 = pow(netOut[i] -truth[i],2);
    if (pow(terrSca2,.5) > maxSca2)
        maxSca2 = pow(terrSca2,.5);
    else if (pow(terrSca2,.5) < minSca2)
        minSca2 = pow(terrSca2,.5);
    if(terrSca2 < 0){
        errorSca2 = errorSca2 - terrSca2;
    }
    else{
        errorSca2 = errorSca2 + terrSca2;
    }
}
else if(i%noutput == 3){
    terrShe1 = pow(netOut[i] -truth[i],2);
    if (pow(terrShe1,.5) > maxShe1)
        maxShe1 = pow(terrShe1,.5);
    else if (pow(terrShe1,.5) < minShe1)
        minShe1 = pow(terrShe1,.5);
    if(terrShe1 < 0){
        errorShe1 = errorShe1 - terrShe1;
    }
    else{
        errorShe1 = errorShe1 + terrShe1;
    }
}
else if(i%noutput == 4){
    terrShe2 = pow(netOut[i] -truth[i],2);
    if (pow(terrShe2,.5) > maxShe2)
        maxShe2 = pow(terrShe2,.5);
    else if (pow(terrShe2,.5) < minShe2)
        minShe2 = pow(terrShe2,.5);
    if(terrShe2 < 0){
        errorShe2 = errorShe2 - terrShe2;
    }
    else{
        errorShe2 = errorShe2 + terrShe2;
    }
}
}
else if (noutput==7){
    if (i%noutput ==0){
        terrRot = pow(netOut[i] - truth[i],2);
        if (pow(terrRot,.5) > maxRot)
            maxRot = pow(terrRot,.5);
        else if (pow(terrRot,.5) < minRot)
            minRot = pow(terrRot,.5);
        if (terrRot < 0){
            errorRot = errorRot - terrRot;
        }
        else{
            errorRot = errorRot + terrRot;
        }
    }
    else if(i%noutput == 1){
        terrSca1 = pow(netOut[i] - truth[i],2);

```

```

    if (pow(terrSca1,.5) > maxSca1)
        maxSca1 = pow(terrSca1,.5);
    else if (pow(terrSca1,.5) < minSca1)
        minSca1 = pow(terrSca1,.5);
    if (terrSca1 < 0){
        errorSca1 = errorSca1 - terrSca1;
    }
    else{
        errorSca1 = errorSca1 + terrSca1;
    }
}
else if(i%noutput == 2){
    terrSca2 = pow(netOut[i] -truth[i],2);
    if (pow(terrSca2,.5) > maxSca2)
        maxSca2 = pow(terrSca2,.5);
    else if (pow(terrSca2,.5) < minSca2)
        minSca2 = pow(terrSca2,.5);
    if(terrSca2 < 0){
        errorSca2 = errorSca2 - terrSca2;
    }
    else{
        errorSca2 = errorSca2 + terrSca2;
    }
}
else if(i%noutput == 3){
    terrShe1 = pow(netOut[i] -truth[i],2);
    if (pow(terrShe1,.5) > maxShe1)
        maxShe1 = pow(terrShe1,.5);
    else if (pow(terrShe1,.5) < minShe1)
        minShe1 = pow(terrShe1,.5);
    if(terrShe1 < 0){
        errorShe1 = errorShe1 - terrShe1;
    }
    else{
        errorShe1 = errorShe1 + terrShe1;
    }
}
else if(i%noutput == 4){
    terrShe2 = pow(netOut[i] -truth[i],2);
    if (pow(terrShe2,.5) > maxShe2)
        maxShe2 = pow(terrShe2,.5);
    else if (pow(terrShe2,.5) < minShe2)
        minShe2 = pow(terrShe2,.5);
    if(terrShe2 < 0){
        errorShe2 = errorShe2 - terrShe2;
    }
    else{
        errorShe2 = errorShe2 + terrShe2;
    }
}
else if(i%noutput == 5){
    terrTran1 = pow(netOut[i] -truth[i],2);
    if (pow(terrTran1,.5) > maxTran1)
        maxTran1 = pow(terrTran1,.5);
    else if (pow(terrTran1,.5) < minTran1)
        minTran1 = pow(terrTran1,.5);
    if(terrTran1 < 0){

```

```

        errorTran1 = errorTran1 - terrTran1;
    }
    else{
        errorTran1 = errorTran1 + terrTran1;
    }
}
else if(i%noutput == 6){
    terrTran2 = pow(netOut[i] -truth[i],2);
    if (pow(terrTran2,.5) > maxTran2)
        maxTran2 = pow(terrTran2,.5);
    else if (pow(terrTran2,.5) < minTran2)
        minTran2 = pow(terrTran2,.5);
    if(terrTran2 < 0){
        errorTran2 = errorTran2 - terrTran2;
    }
    else{
        errorTran2 = errorTran2 + terrTran2;
    }
}
}
}
if (noutput == 3){
    errorRot = (pow(errorRot,.5))/25; errorSca1 =
(pow(errorSca1,.5))/25; errorSca2 = (pow(errorSca2,.5))/25;
    printf("Avg Error Rotation: %f, %f\nAvg Error X Scale: %f\nAvg
Error Y Scale: %f\n", errorRot, errorRot*20, errorSca1*.1,
errorSca2*.1);
}
else if (noutput == 5){
    errorRot = (pow(errorRot,.5))/fin; errorSca1 =
(pow(errorSca1,.5))/fin; errorSca2 = (pow(errorSca2,.5))/fin;
    errorShe1 = (pow(errorShe1,.5))/fin; errorShe2 =
(pow(errorShe2,.5))/fin;
    printf("Avg Error Rotation: %f, %f\nAvg Error X Scale: %f\nAvg
Error Y Scale: %f\nAvg Error X Tran: %f\nAvg Error Y Tran: %f\n",
errorRot, errorRot*24, errorSca1*.2, errorSca2*.2, errorShe1*10,
errorShe2*10);
    /*printf("Avg Error Rotation: %f, %f\nAvg Error X Scale:
%f\nAvg Error Y Scale: %f\nAvg Error X Shear: %f\nAvg Error Y Shear:
%f\n", errorRot, errorRot*180, errorSca1*.8, errorSca2*.8, errorShe1*1,
errorShe2*1);*/
    printf("Max/Min Rotation Error: %f %f\nMax/Min x Scale Error:
%f %f\nMax/Min y Scale Error: %f %f\nMax/Min x Tran Error: %f
%f\nMax/Min y Tran Error: %f %f\n", maxRot*24, minRot*24, maxSca1*.2,
minSca1*.2, maxSca2*.2, minSca2*.2, maxShe1*10, minShe1*10, maxShe2*10,
minShe2*10);
}
else if (noutput == 7){
    errorRot = (pow(errorRot,.5))/fin; errorSca1 =
(pow(errorSca1,.5))/fin; errorSca2 = (pow(errorSca2,.5))/fin;
    errorShe1 = (pow(errorShe1,.5))/fin; errorShe2 =
(pow(errorShe2,.5))/fin;
    errorTran1 = (pow(errorTran1,.5))/fin; errorTran2 =
(pow(errorTran2,.5))/fin;
    /*    printf("RAWR: %f\n", errorSca1);*/
    printf("Avg Error Rotation: %f, %f\nAvg Error X Scale: %f\nAvg
Error Y Scale: %f\nAvg Error X Shear: %f\nAvg Error Y Shear: %f\nAvg x-

```

```

tran Error: %f\nAvg y-tran Error: %f\n", errorRot, errorRot*180,
errorScal*.2, errorSca2*.2, errorShel*.2, errorShe2*.2, errorTran1*10,
errorTran2*10);
    printf("Max/Min Rotation Error: %f %f\nMax/Min x Scale Error:
%f %f\nMax/Min y Scale Error: %f %f\nMax/Min x shear Error: %f
%f\nMax/Min y Shear Error: %f %f\nMax/Min x Tran Error: %f %f\nMax/Min
y Tran Error: %f %f\n", maxRot*24, minRot*24, maxScal*.2, minScal*.2,
maxSca2*.2, minSca2*.2, maxShel*.2, minShel*.2, maxShe2*.2, minShe2*.2,
maxTran1*10, minTran1*10, maxTran2*10, minTran2*10);
    }

    return 0;
}

```