APPLICATIONS OF ARTIFICIAL NEURAL NETWORKS TO SYNTHETIC APERTURE RADAR FOR FEATURE EXTRACTION IN NOISY ENVIRONMENTS

A Thesis Presented to the

Electrical Engineering Department Faculty of

California Polytechnic State University

San Luis Obispo, California

In Partial Fulfillment

Of the Requirements for the Degree of

Masters of Science in Electrical Engineering

By

David J. Roberts

June 2013

Committee Membership

TITLE:	Applications of Artificial Neural Networks to Synthetic Aperture Radar for Feature Extraction in Noisy Environments
AUTHOR:	David J. Roberts
DATE SUBMITTED:	June 2013
COMMITEE CHAIR:	Dr. John Saghri, Professor Electrical Engineering
COMMITTEE MEMBER:	Dr. Xiao-Hua (Helen) Yu, Professor Electrical Engineering
COMMITTEE MEMBER:	Dr. Jane Zhang, Associate Professor Electrical Engineering

Abstract

Applications of Artificial Neural Networks to Synthetic Aperture Radar for Feature Extraction in Noisy Environments

David J. Roberts

It is often that images generated from Synthetic Aperture Radar (SAR) are noisy, distorted, or incomplete pictures of a target or target region. As the goal for most SAR research pertains to automatic target recognition (ATR), extensive filtering and image processing is required in order to extract the features necessary to carry out ATR. This thesis investigates the use of Artificial Neural Networks (ANNs) in order to improve upon the feature extraction process by laying the foundation for ANN SAR ATR algorithms and programs.

The first technique investigated is that of an ANN edge detector designed to be invariant to multiplicative speckle noise. The algorithm designed uses the Back Propagation (BP) algorithm to teach a multi-layer perceptron network to detect edges. In order to do so, several parameters within a Sliding Window (SW), are calculated as the inputs to the ANN. The ANN then outputs an edge map that includes the outer edge features of the target as well as some internal edge features.

The next technique that is examined is a pattern recognition and target reconstruction algorithm based off of the associative memory ANN known as the Hopfield Network (HN). For this version of the HN, the network is trained with a collection of varying geometric shapes. The output of the network is a nearest-fit representation of the incomplete image data input. Because of the versatility of this program, it is also able to reconstruct incomplete 3D models determined from SAR data.

The final technique investigated is an automatic rotation procedure to detect the change in perspective relative to the platform. This type of detection can prove useful if used for target tracking or 3D modeling where the direction vector or relative angle of the target is a desired piece of information.

Acknowledgements

This project is a continuation of prior SAR research at Cal Poly under Dr. John Saghri with the sponsorship of Raytheon Space & Airborne Systems.

First I would like to thank Dr. Saghri for supporting me in exploring the more abstract intracicies of Neural Networks as my contribution to SAR research at Cal Poly. As such, I would also like to thank Dr. Yu for introducing me to Neural Networks. Without her class on computational intellegence, I would have never thought to apply Neural Networks to radar. And for Dr. Zhang, who's class on computer vision proved invaluable for understanding the fundamentals of automatic computer detection.

I would also like to thank my friends for their support and friendship through my time at Cal Poly. All of the fun that we had gave me a good balance of work and play to keep me focused on finishing this thesis on time. To all the great times in the Grad Lab, I thank them all. (And of course: Coffee Time!)

To the music department, especially Prof. William Johnson ("J"), Prof. Christopher Woodruff, and Dr. Andrew McMahan for giving me the wonderful opportunity to participate in the Wind Orchestra, Wind Ensemble, and Clarinet Ensembles. Without the music, my Cal Poly experience wouldn't have been complete.

Last, but certainly not least, I would especially like to thank my family for supporting me in the last six years while I worked through both my Bachelors and Masters Degrees here at Cal Poly. Their constant support and encouragement proved invaluable for my motivation to keep chugging along.

Table of Contents

List of Figures

List of Acronyms

ANN	-	Artificial Neural Network
ATR	-	Automatic Target Recognition
CSAR	-	Circular SAR
CAM	-	Content Addressable Memory
DHN	-	Discrete Hopfield Network
E-CSAR	-	Elevation Circular SAR
LV	-	Local Variance
NN	-	Neural Network
Radar	-	Radio detection and ranging
RLV	-	Real Local Variance
SW	-	Sliding Window
SAR	-	Synthetic Aperture Radar

1 Introduction

1.1 Project Scope

The goal of this project is to explore the various applications of Artificial Neural Networks (ANN) to Synthetic Aperture Radar (SAR) and lay a groundwork for more advanced projects in the field at Cal Poly.

1.2 Document Overview

This document is organized so that the reader is fully informed about the theory and background before data is presented. The first several chapters cover the basics behind Synthetic Aperture Radar, Artificial Neural Networks, as well as some methodology in images processing. The remaining chapters present the results and discussion of the outputs of the algorithms designed in this thesis. The final chapters of this thesis include conclusions and future work. A more extensive collection of data, trials and testing sets can be found in the appendices along with the used MATLAB code.

1.3 Software Platform

All of the simulations and programs written for this Thesis were written in the MATLAB® programming language and executed using the following MATLAB® programming environments versions: 7.10.0.499 (R2010a) 32-bit and 7.12.0.635 (R2011a) 32-bit. This platform was selected for the simplicity of designing and debugging image processing and matrix-based mathematical operation programs. MATLAB® has many integrated functions for viewing and analyzing 2D and 3D images as well as optimized matrix math operations that greatly accelerates development. It is to be noted that the MATLAB® Neural Network Toolbox™ was not used in the development of this thesis.

1.4 Research at Cal Poly

Previous research at Cal Poly for SAR image processing includes both 2D and 3D feature extraction and Automatic Target Recoginition (ATR) techniques. The 2D techniques include rectangular fits, peak detection, corner detection, edge detection, as well as several other analysis techniques. Some of the notable techniques inlude the use of Hausdorff distance and chamfer distance transform as applied to edges, corners, and peaks to increase accuracy of ATR.[1] Another notable project studied a method of ATR using the optimal rectangular fit of a target within a given SAR image. However, the rectangluar fit method so far explored at Cal Poly has only achieved a 40% classificiation accuracy.[2]

For 3D feature extraction, Circular SAR (CSAR), Elevation Circular SAR (E-CSAR), interferometric SAR, and several other methods have been explored. The notable original method developed at Cal Poly was dubbed the "Shadow" method. This method extracted the shadows of a target and geometrically would interpret the height of the target, using the knowlege of the platform's position relative to the target.[3]

The other research conducted at cal poly is directed at the analysis of the raw return data recorded by the SAR antenna. The eventual goal of this research is to perform ATR without the need for producing an image first.

2 Synthetic Aperture Radar (SAR)

In this section, the history and several definitions of radar and synthetic aperture radar will be given. Additionally, information about the SAR images used in this project will also be discussed.

2.1 History

Radio detection and ranging (radar) is a remote sensing technique that utilizes the phenomena that radio waves can be reflected off of certain solid objects. This phenomena of electromagnetic reflectivity was first observed by Heinrich Hertz in 1886, however it was not extensively researched until the 1920s and 1930's. The technique is based on a moving or stationary platform sending pulses of electromagnetic waves (beams) in one or many directions, then measuring the magnitude of the reflected wave(s). The larger the magnitude of the return, the more likely there is an object of interest in the direction of the beam.[4]

Synthetic Aperture Radar (SAR) is an application of conventional radar that is capable of generating high-resolution imagery of a target or target area by utilizing the relative motion of the transmitter. First proposed in the 1950's, SAR was originally built out of research intended to increase the angular resolution of radar. It wasn't until 1957 when the first focused airborne SAR image was produced.[5]

2.2 Platform and Geometry

The platform is that of a moving aircraft using a side-looking radar antenna (SLAR). In this configuration, it capable of easily imaging slow moving target objects (generally ground targets), relative to the aircraft. An example of the relationship between the platform and the target can be seen in Figure 2-1. Shown in the image is the point in time when the antenna and target are at the point of minimum distance. This point is referred to as the point of minimum slant range.

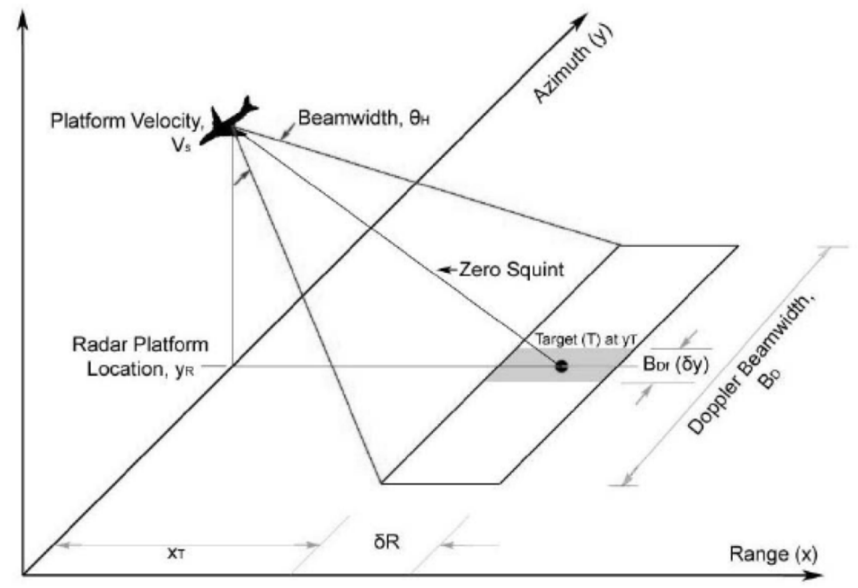

Figure 2-1: SLAR Platform-Target Geometry

The type of SAR imaging shown in this figure is known as stripmap SAR. The name is derived from the fact that the radar beam is always pointing at the same direction, relative to the platform's direction of travel. This direction is often perpendicular to the direction of travel.

For the type of SAR images used in this thesis are those of a modified method of stripmap SAR known as spotlight SAR. Spotlight SAR differs from stripmap SAR in the fact that the radar beam is always pointed towards the target or area of interest.[3]

2.3 Image Reconstruction

The desired final output of SAR is, generally, a two-dimensional representation of the radar reflectivity within a target region. For this thesis, only spotlight SAR image reconstruction is considered. In order to do so, a few variables must be defined.

The data collected by the SAR antenna is normally recorded as a function of designation $s(t,u)$, where t represents time in the range dimension (fast-time) and u represents time in the cross-range dimension (slow-time). This data is simply the magnitude and phase of the signal received, relative to the transmitted signal.

The image reconstruction of the target function, more specifically for spotlight SAR, generally involves fast-time filtering, in this case multiplying the frequency domain version of the returned signal $s(t,u)$ by a modified, phase-shifted form of the complex conjugate of the transmitted radar signal. The phase-shifting accounts for the fact that there is a constant central target position and a non-constant platform position. The two-dimensional frequency domain version of the reconstructed target function for spotlight SAR is:

$$F\left[k_x(\omega,k_u),k_y(\omega,k_u)\right] = S(\omega,k_u)\cdot P^*(\omega)\cdot \exp\left(jk_xX_c + jk_yY_c\right)$$

where $k_x(\omega,k_u)$ and $k_y(\omega,k_u)$ are domain transformations necessary for the creation of two-dimensional target function reconstructions in the desired x-y Cartesian plane. These transforms are specified by:

$$k_x(\omega,k_u) = \sqrt{4k^2 - k_u^2}$$

and

$$k_y(\omega,k_u) = k_u$$

The variables for these expressions are described in [6]. In order to reconstruct a target image as a two-dimensional (spatial) image, the transmitted signal, received (measured) signal, and the central position of the illuminated target region are required. The final spatial image is then generated by performing the 2-D Inverse Discrete Fourier Transform (or the Inverse Fast Fourier Transform in a practical scenario) of the function $F\left[k_x(\omega, k_u), k_y(\omega, k_u)\right]$. The image is plotted as a 2-D grayscale image.[3][6]

$S(\omega, k_u)$	**The measured signal returned to the SAR antenna**
$P^*(\omega)$	**The matched filter complex conjugate of the transmitted radar signal.**
k_x	**Spatial frequency domain in the range dimension.**
k_y	**Spatial frequency domain in the cross-range dimension.**
ω	**Fast-time frequency domain (frequency domain of *t*).**
k_u	**Slow-time frequency domain (frequency domain of *u*).**
X_c	**Central position of the target region in the range dimension.**
Y_c	**Central position of the target region in the cross-range dimension.**
k	**Radar Wavenumber:** $\boldsymbol{k = \frac{\omega}{c}}$

Figure 2-2: SAR Image Reconstruction Signal and Variable Descriptions

2.4 Speckle Noise

Speckle Noise is a type of granular or salt-and-pepper noise that is generally present within SAR images as well as other radar applications. This type of noise is the result of random fluctations in the reflected wave. It is a multiplicative noise and therefore more difficult to filter. Multiplicative noise does not have any direct proportional relationship to the local grey level within a SAR image.[7]

2.5 Automatic Target Recognition (ATR)

There are various applications for algorithms that are able to automatically identify a target within a given SAR image. As such, much of the SAR research done at Cal Poly has been in the development of ATR algorithms. Conventionally, a series of feature-extraction algorithms are used to find edges, corners, area, etc. The individually extracted features, or some combination thereof, are used to determine the identity of the target in question. As a method of analyzing the accuracy of the ATR algorithm, the algorithm is often performed on a known data set and a confusion matrix is used to assess the classification accuracy.[3]

2.6 SAR Image Set

The same set of images used in previous research at Cal Poly were used in this project. The images used in this project are from a data set produced by Sandia National Laboratory in 1995. The data set is called the Moving and Stationary Target Acquisition and Recognition program, more commonly referred to as MSTAR.[8] The images within the set are available at depression angles of 15°, 17°, 30° and 45°. Each image has images captured at rotation (azimuth) angles 0° to 359°. All of the targets are centered within the images.

The targets included in the data set are named 2S1, BRDM2, BTR60, D7, SLICY, T62, ZIL_131, and ZSU_23_4. The 2S1 Gvozdika is a self-propelled howitzer, formerly designed and manufactured in the Soviet Union. The BRDM2 is an armored reconnaissance vehicle, also formerly designed and manufactured in the Soviet Union. The BTR-60 is an armored personnel carrier (APC) also formerly designed and manufactured in the Soviet Union. The D7 is a medium bulldozer manufactured by the United States company Caterpillar Inc. The T62 is a main battle tank, formerly designed and manufactured in the Soviet Union. The ZIL 131 is a general-purpose army truck formerly designed and manufactured in the Soviet Union. The ZSU-23-4 is a self-propelled anti-aircraft weapon system, also formerly designed and manufactured in the Soviet Union.[9]

The only target, not an actual vehicle, is the SLICY target. This target is a radar reflective stationary structure composed of many standard radar-reflecting shapes specifically designed for radar development. This target has the additional depression angles of 16°, 29°, 31°, 43°, and 44° available.

All of the included MSTAR images have a low pixel resolution, by modern display standards, ranging from 54×54 pixels to 158×158 pixels. The perspective of each image appears as a top-down photograph as the radar images were taking from altitudes above the targets.

The MSTAR data set was collected using a radar system with a center frequency of 9.6 GHz and a 0.6 GHz bandwidth. The expected imaging resolution is 0.3047 meters/pixel (~1 foot) with pixel spacing of 0.2 meters.[3]

3 Artificial Neural Networks (ANN)

Artificial neural networks (ANNs), or more commonly referred to as neural networks (NNs), are computational networks that are modeled after the human brain. The brain, being a highly complex, nonlinear, and parallel computer[10], is well suited to solving problems that would be difficult for a normal computer. The desirability of such a computational engine has lead many mathematicians and engineers to seek a way to implement an artificial brain.

3.1 History

Through the history of development, ANNs root from the curiosity of scientists and engineers as to how the brain functions. Of the theories considered, two major ones stand out: Monotypic Models and Genotypic Models.[11] Due to the deterministic nature of the data being analyzed in this paper, the network types that have evolved from the fundamental Monotypic model will be used.

3.2 The Perceptron

The Perceptron, developed by Frank Rosenblatt, is the first computational model based off of the non-linear McCulloch-Pitts model of a neuron.[10] The McCulloch-Pitts model stated the assumption that the nervous system is a network of neurons, each containing a soma and an axon. In this network, each axon is connected to another neuron's soma via a synapse. The most significant assumption about a neuron's operation is what they dubbed "all-or-none" behavior. This is where the output of the neuron either outputs energy, or doesn't output energy.[12]

Rosenblatt used this model to formulate a computational model that described this "all-or-none" behavior, but in a way that it may be implemented in non-organic hardware. Figure 3-1 is a visualization of a single perceptron model.

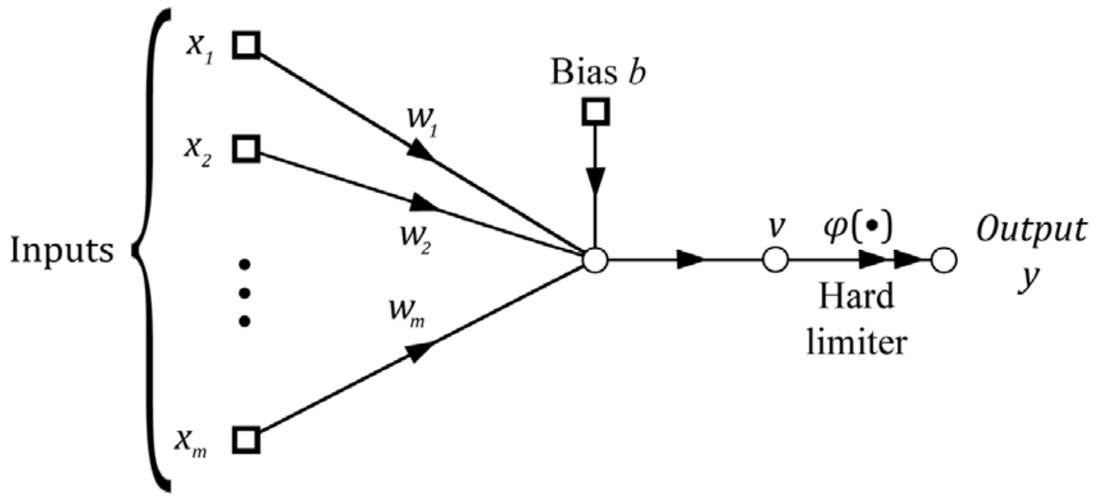

Figure 3-1: Single Perceptron Model

All of the variables in the neuron can be continuous or discrete. The only defined value range in a neuron is the output which normally ranges from -1 to +1, but may have a different range if desired. Mathematically, for a layer of m neurons, and neuron of index *j* and iteration *n*, the neuron can be defined as:

$$y_j(n) = \varphi_j\left(v_j(n)\right)$$

where y_j is the output of the current neuron, φ_j is the activation function, and v_j is defined as:

$$v_j(n) = \sum_{i=1}^{m} w_i(n) y_i(n) + b$$

where y_i is an output from a previous layer *i*, and w_i is the weight that connects the input to the neuron of index *j*.

With a single layer perceptron network, an ANN is able to solve classification problems so long as they are linearly separable by a hyperplane that is defined by:

$$\sum_{i=1}^{m} w_i x_i + b = 0$$

3.2.1 Activation Functions

Depending on the application of the ANN, different types of activation functions can be used. The ones discussed in this section are the Signum function, the Hyperbolic Tangent function, Sigmoid Function, and the Ramp function.

3.2.1.1 Signum Function

The closest defined mathematical operator that can reproduce the "all-or-nothing" behavior, as required by the McCulloch-Pitts model is that of the signum function. This function is parametrically defined as:

$$\varphi(x) = sgn(x) = \begin{cases} -1 & \text{if } x < 0 \\ 0 & \text{if } x = 0 \\ 1 & \text{if } x > 0 \end{cases}$$

for any real number *x*. A plot of the signum function can be seen in Figure 3-2.

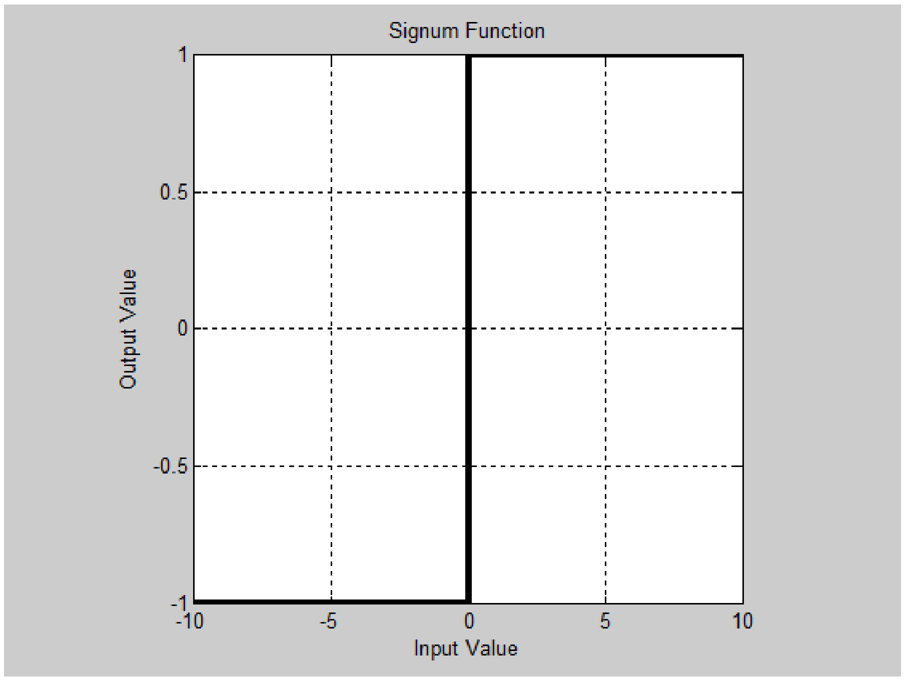

Figure 3-2: Signum Activation Function Plot

The signum function is useful when a strictly binary output is desired for the neural network.

3.2.1.2 Hyperbolic Tangent Function

The next closest "all-or-nothing" activation function is that of the hyperbolic tangent. It is mathematically defined as:

$$\varphi(x) = \tanh(x) = \frac{e^x - e^{-x}}{2}$$

for any real value of *x*. A plot of the hyperbolic tangent can be seen in Figure 3-3.

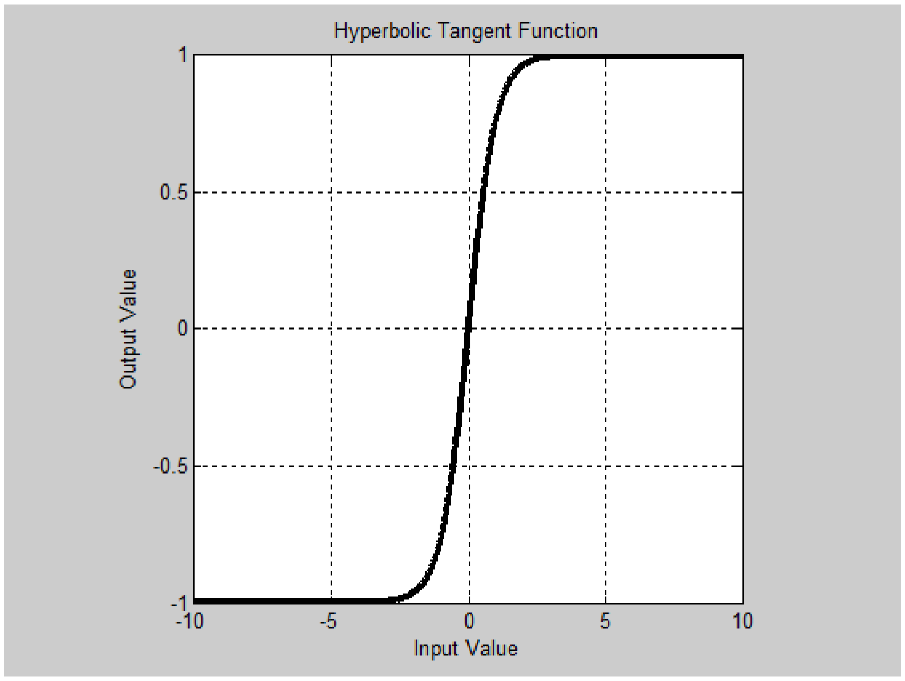

Figure 3-3: Hyperbolic Tangent Activation Function Plot

The hyperbolic tangent is often used because it is nearly "all-or-nothing", however it contains no discontinuities, as like the signum function. Depending on the computational situation, this lack of discontinuities can be desired (i.e. analog ANN applications where true step discontinuities cannot exist).

3.2.1.3 Sigmoid Function

The sigmoid function (logistic sigmoid) is a type of logistic function that, in its simplest form, is defined as:

$$\varphi(x) = P(x) = \frac{1}{1 + e^{-x}}$$

for any real value *x*. Unlike the previously stated functions, the sigmoid function output is from zero (0) to positive one (+1). This can be seen in Figure 3-4.

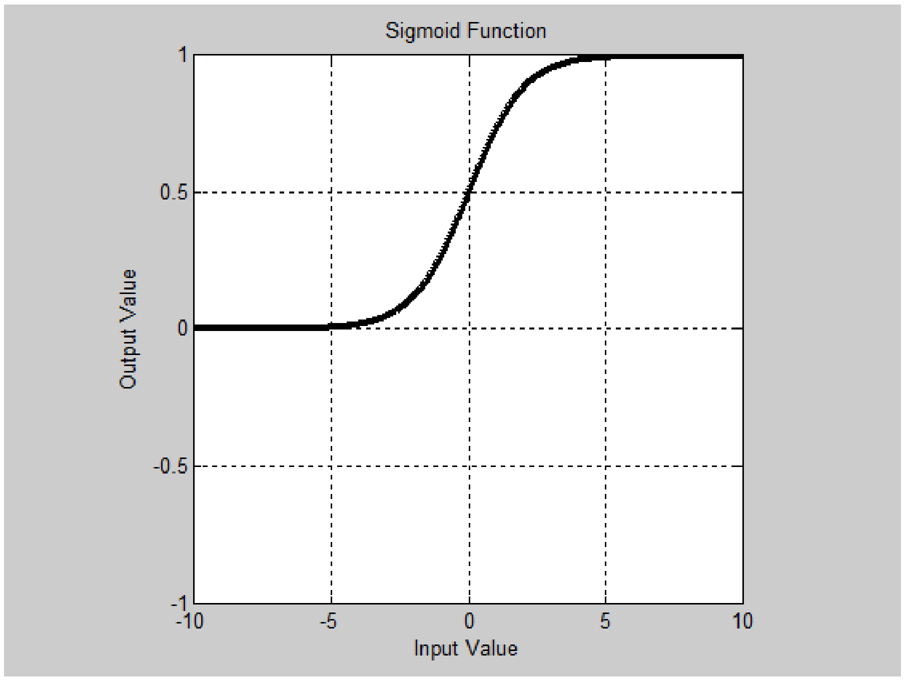

Figure 3-4: Sigmoid Activation Function Plot

This type of activation function is useful when the output range is required to be all positive or all negative, but not both. The logistic sigmoid, however, is related to the hyperbolic tangent by:

$$2P(x) = 1 + \tanh\left(\frac{x}{2}\right)$$

for real values of *x*.

3.2.1.4 Ramp Function

The last, and least used, activation function is that of a linear ramp function. This function is defined as:

$$\varphi(x) = mx$$

for a slope value of *m* and real values if *x*. A linear ramp of slope $m = \frac{1}{10}$ can be seen in Figure 3-5.

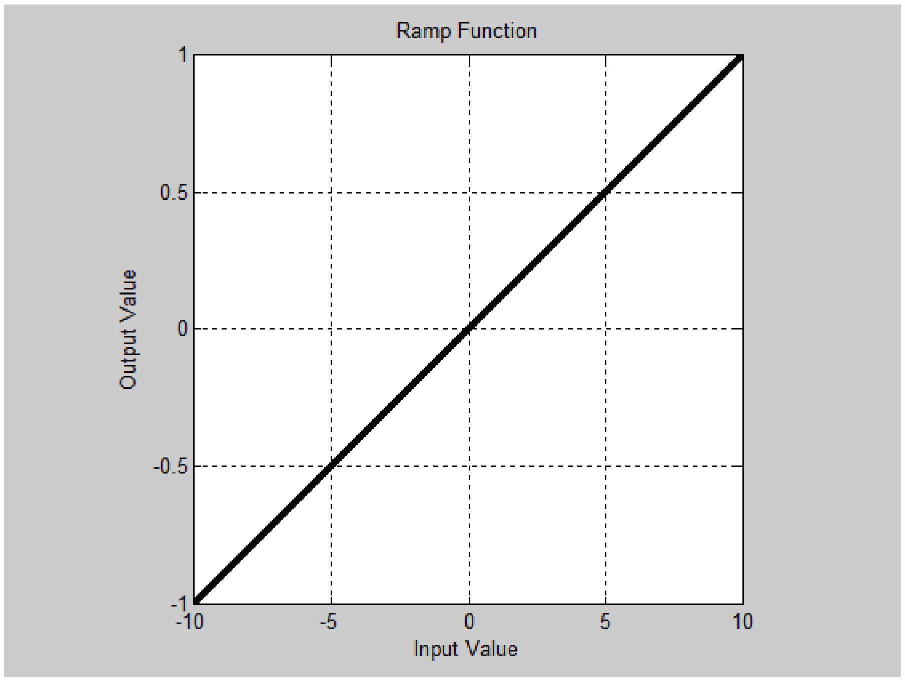

Figure 3-5: Ramp (slope of 1/10) Activation Function Plot

While not commonly used, as ANN's are desirable for non-linear problem solving, the ramp function can be used for certain modeling problems where a gradient output is desired.

3.2.1.5 Comparison

To best understand the differences between the activation functions, a plot of all four discussed are shown in Figure 3-6. From greatest to least steep slope are: signum function, hyperbolic tangent, sigmoid function, and ramp function, respectively.

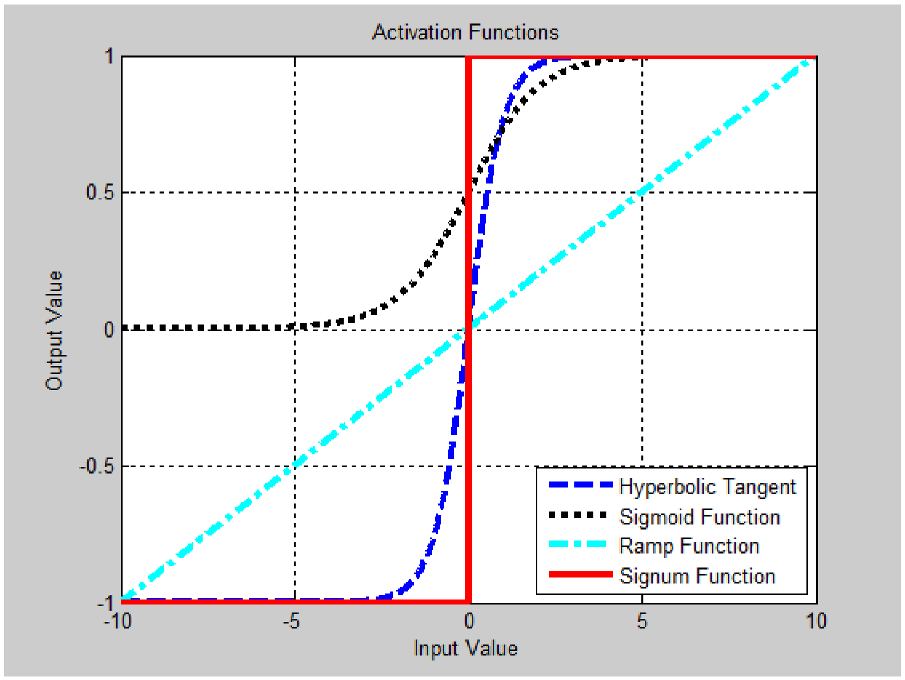

Figure 3-6: Overlay of Activation functions

Each of these functions are better for different applications of neural networks. The one that tends towards more theoretical calculations and certain types of neural network configurations is the Signum function due to its infinite slope and parametric function definition.

3.3 Multi-Layer Networks

As the name implies, a multilayer perceptron network, or more generally a multilayer neural network, is a collection of neurons that are arranged into layers with varying numbers of neurons per layer. For instance, a three layer neural network is shown in Figure 3-7. While the single perceptron is capable of performing effective pattern recognition, it is only able to solve problems that can be separated by a single hyperplane. In order to solve this problem, additional layers are added to the network, introducing the concept of hidden layers. By adding these multiple hidden layers onto the ANN, the network gains more hyperplane decision boundaries that allow it to solve more complex problems.

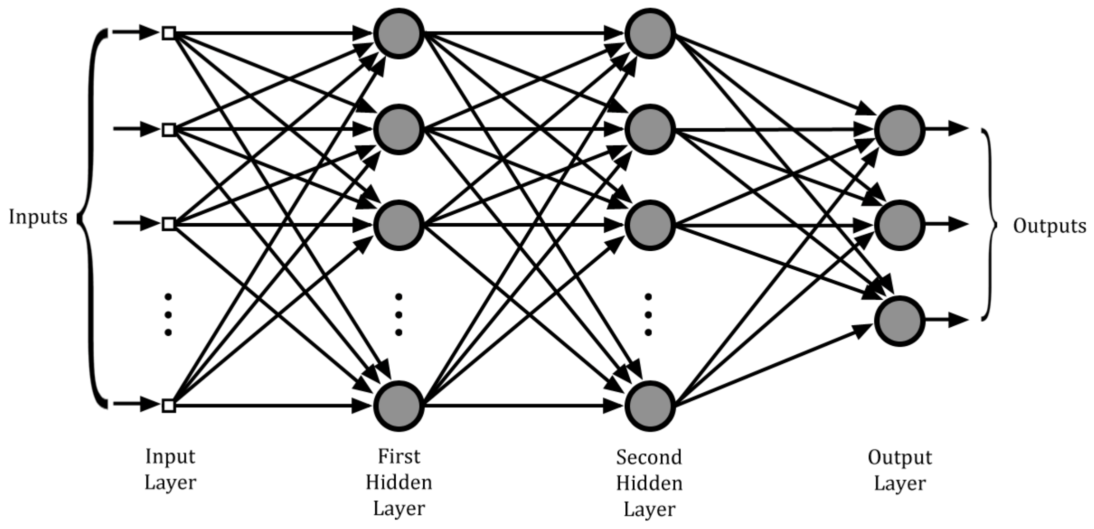

Figure 3-7: A Multilayer Perceptron Network

For clarity of notation, a "preceding" layer is a layer that is closer to the input relative to the current neuron being computed. Conversely, a "following" layer is a layer that is closer to the output relative to the current neuron being computed.

As before, the mathematics governing how a multilayer network is nearly identical to that of a single perceptron, the only modification is the neuron indexing. In general, the output of a neuron is again defined as:

$$y_j(n) = \varphi_j\left(v_j(n)\right)$$

where *j* is the current neuron being evaluated and:

$$v_j(n) = \sum_{i=1}^{m} w_{ji}(n) y_i(n) + b$$

where *i* is a neuron from the preceding layer. The standard notation for the interconnecting weight is:

$$w_{<destination\ neuron><source\ neuron>}$$

Since in this case the "destination neuron" is the neuron currently being evaluated in the *j* layer, the *j* is placed first followed by the *i*-th neuron index from the preceding layer.[10]

3.4 Network Training/Learning

The training of a neural network, more commonly referred to as learning, is a procedure in which re-organizes the interconnections of the network such that a desired output is created when given some input of known dimensions. Depending on the network configuration, different methods of network updating can be performed. The methods discussed in this section include the Back Propagation Algorithm and the Hopfield Model.

3.4.1 Back Propagation Algorithm

A method of supervised learning; the Back Propagation Algorithm (BP algorithm) is the most common and versatile method to 'learn' an ANN. The method is based around propagating the error at the output of the network back through the network. A simple logical diagram of how the BP algorithm is performed is shown in Figure 3-8. This algorithm is repeated for either 1) until the desired number of iterations is exceeded or 2) some other performance metric for overall network performance is met.

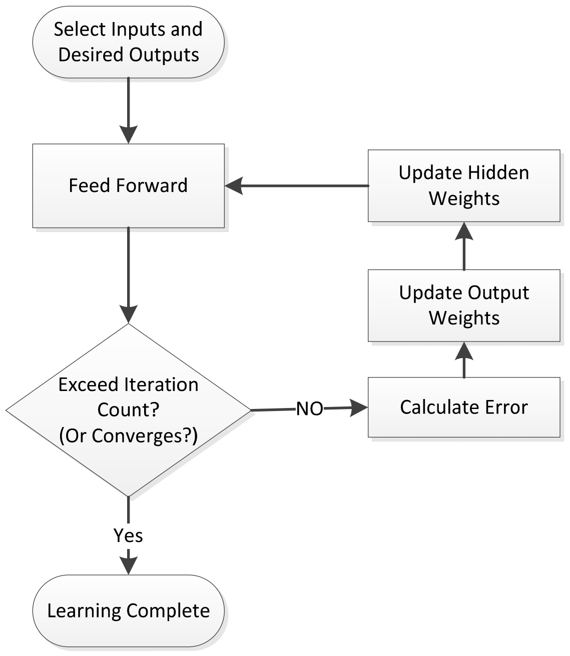

Figure 3-8: Back Propagation Algorithm Condensed Block Diagram

3.4.1.1 Feed Forward Stage

In this stage, the input is evaluated by propagating the input through the network forwards. This method is simply evaluating the equations on a neuron-by-neuron basis as defined in the background section on ANNs.

3.4.1.2 Error Calculation

Before the network is used, the user must determine what type of data they desire to have learned by the network. This information is a collection of known inputs with known associated outputs. During the error calculation phase, the network's performance is evaluated according to the known input/output relationship. The error calculation is only evaluated for the output layer. A common representation for this error is:

$$e_j(n) = d_j - y_j(n)$$

where e_j is the error for the current neuron being evaluated, d_j is the desired output for the current neuron being evaluated and y_j is the current output of neuron j for iteration n. As this is a general expression, the actual error measured may be a more elaborate algorithm depending on the network application, however the computed error must be a numerical value.

3.4.1.3 Updating Weights

In order for a network to "learn" the proper input/output relationships, the interconnecting weights must be updated. This is the back propagation portion of the Back Propagation Algorithm.

In general, the Back Propagation Algorithm is an iterative process. The weight for the current Feed Forward iteration is n and the next iteration, $n + 1$, is described by the equation:

$$w_{ji}(n+1) = w_{ji}(n) + \Delta w_{ji}(n)$$

where

$$\Delta w_{ji}(n) = \eta \delta_j y_i(n)$$

The variable η is the learning rate (value < 1) defined at program initialization and δ_j is the local gradient for neuron j. For an output neuron, the local gradient is defined as:

$$\delta_j = e_j(n)\varphi_j'\left(v_j(n)\right)$$

φ_j' is the derivative of the activation function φ. The final expression for updating an output weight is then:

$$w_{ji}(n+1) = w_{ji}(n) + \eta\left[e_j(n)\varphi_j'\left(v_j(n)\right)\right] y_i(n)$$

For hidden weights, as they have no direct contact with the outside world and are only affected by the network error that is back-propagated from the change in local gradient of the following neuron. As before, if the current neuron being computed is j, the new iteration weight is identical to that of an output weight, however the local gradient is different. The local gradient for a hidden neuron is defined as:

$$\delta_j = \varphi_j'\left(v_j(n)\right)\sum_k \delta_k(n) w_{kj}(n)$$

where k, is the index of a neuron from the following layer. Therefore the full equation for updating a hidden weight is:

$$w_{ji}(n+1) = w_{ji}(n) + \eta\left[\varphi_j'\left(v_j(n)\right)\sum_k \delta_k(n) w_{kj}(n)\right] y_i(n)$$

This can be repeated for as many layers as are desired, depending on the requisite number of separable classes needed. The number of layers, however, does not correspond to the number of separable hyperplanes.

3.5 Hopfield Model

The Hopfield Model of an ANN, unlike the previously mentioned multi-layer perceptron network, is a multiple-loop feedback system that relies on unit time-delays to calculate output. Depending on the desired function of the Hopfield Network, self-feedback may or may not be included. The example in Figure 3-9 is that of a four neuron Hopfield Network with no self feedback. While the Hopfield model can be operated in either a discrete or a continuous mode, for the purposes of this thesis, only the discrete case will be considered. Like with the previous sections, this follows the concept of operation determined by McCulloch and Pitts.[10]

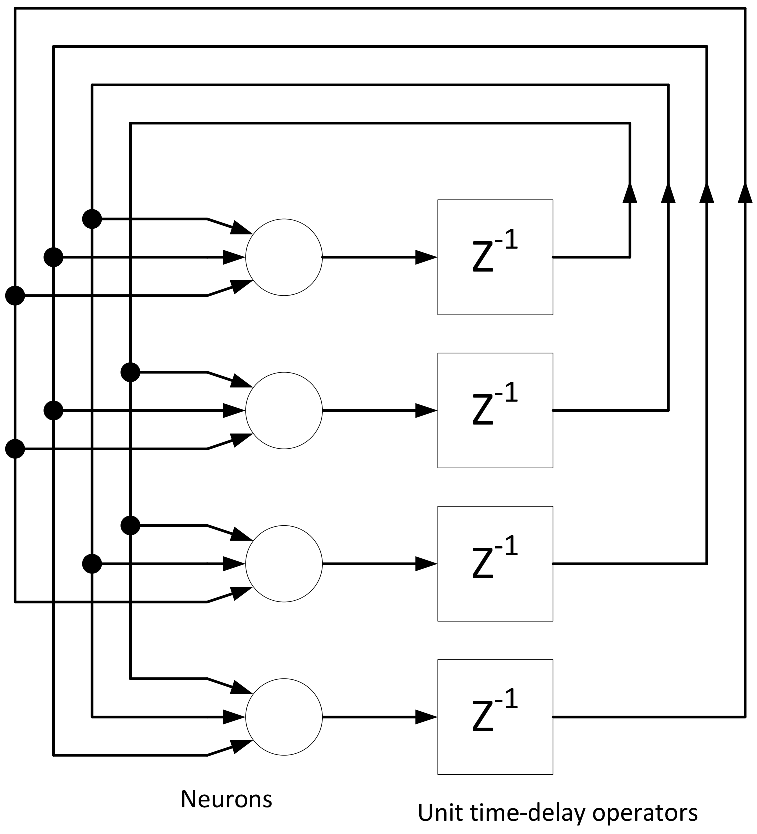

Figure 3-9: Hopfield Network Example for 4 neurons and no self-feedback.

For a discrete Hopfield network, there have to be two assumptions about the behavior of the neurons. First, the outputs of the neurons must have asymptotic values at:

$$x_j = \begin{cases} +1 & \text{for } v_j = (+) \\ -1 & \text{for } v_j = (-) \end{cases}$$

and secondly, the midpoint of the activation function of the neuron lies at the origin ($\varphi_j(0) = 0$). While each neuron may have a bias, however for the network used in this thesis, all neuron biases were set to zero.

The desired application of the Hopfield Network in this thesis is to serve the purpose of Content-Addressable Memory (CAM). The primary function of a content-addressable memory is to retrieve a pattern (item) stored in memory in response to the presentation of an incomplete or noisy version of that pattern.[10]

The goal behind how CAMs work is to map some stable fundamental memory, signified by ξ_μ, to a point x_μ in a dynamic system. Each of these fundamental memories acts as attractors that will, iteratively, stabilize the system to one of the memories.

For a Discrete Hopfield Network (DHN) to be used as a CAM, it has two phases of operation: the storage phase and the retrieval phase.[10]

3.5.1 Storage Phase

During the storage phase, the network is "trained" with *M*, *N*-dimensional vectors that constitute the fundamental memories. This can be represented as:

$$\xi_\mu \in \{\xi_1, \xi_2, \dots, \xi_M\}$$

where each ξ is a fundamental memory to be learned. Each fundamental memory must have a value of ±1 for the discrete Hopfield model. In order to build the matrix of connective weights, the outer-product of the memories is computed as follows:

$$w_{ij} = \frac{1}{N}\sum_{\mu=1}^{M} \xi_{\mu j}\xi_{\mu i}$$

where w_{ij} is the weight connecting neuron j to neuron i. For stability reasons, there is no self feedback in a Hopfield network. That is $w_{ii} = 0$ for all i. Due to the "one-shot" nature of this computation, if we allow some matrix $\boldsymbol{W}$ to be the matrix of all weights in the network, then:

$$\boldsymbol{W} = \frac{1}{N}\left[\sum_{\mu=1}^{M} \xi_{\mu}\xi_{\mu}^{T} - M\boldsymbol{I}\right]$$

where $\xi_{\mu}\xi_{\mu}^{T}$ is the outer product of vector ξ_{μ} and $\boldsymbol{I}$ is the identity matrix of dimension $M \times M$. This results in a symmetric weight matrix: $\boldsymbol{W^T} = \boldsymbol{W}$.[10]

3.5.2 Retrieval Phase

In order to retrieve an associative memory from the network, an initial state must be imposed on the system. As the network's input is a time-delayed output, the user is able to set the input to some value, of dimension *N*, to the inputs of all the neurons. This input is generally referred to as a probe and denoted by ξ_{probe}. As with the fundamental memories, the probe must also have values equal to ±1. The retrieval operation is performed by activating each neuron in a random order, or more accurately put, each neuron is activated randomly, but systematically. Network operation ceases as soon as all of the neurons have been activated and a stable state has been reached. The way to test for stability is to check if the alignment condition is satisfied. The alignment condition is defined as:

$$y_i = sgn\left(\sum_{i=1}^{N} \omega_{ji} y_i + b_i\right), \quad j = 1, 2, \dots, N$$

or in matrix form

$$\boldsymbol{Y} = sgn(\boldsymbol{WY} + \boldsymbol{B})$$

where y_i is the output of neuron i, *N* is the number of neurons, and b_i is the bias (if used) for neuron i. The matrix form of the alignment condition is mathematically identical to that of the sequential equation.[10]

4 Image Processing Techniques

This section discusses the various basic techniques that are either employed, or embellished upon within this project.

4.1 Image Thresholding

Level thresholding is a simple imaging processing operation that is commonly used when it is necessary to separate differing logical image regions. The most common type of level threshold is a binary threshold. This type of thresholding converts a grayscale image into a binary image (an image with only two discrete pixel values) by selecting a pixel intensity decision boundary (0 to 255 for 8-bit grayscale). Any value that is below the threshold decision boundary is set to a logical 0 and any value above the decision boundary is set to a logical 1. Any value that is on the decision boundary is either assigned to a logical 0 or a logical 1, as defined by the software designer or application.[3]

4.2 Sliding Window Filters (Kernel Filters)

A Sliding Window Filter (SW), also known as a Kernel Filter, is an image processing technique that is employed in many different types of image filtering algorithms. This method revolves around the use of a small window, one that is significantly smaller than that of the original image, that systematically takes spatial samples of an image. These samples of the image are then subjected to a series of mathematical operators, and the result of these operators result is written to an output image plane. The SW technique can be used either to conduct the linear convolution of the image window with a filter kernel (as used in the Canny and Roberts edge detectors) or a non-linear operation, such as the Discrete Cosine Transform Edge Detector and the proposed ANN Edge Detection Filter that will be discussed in a later chapter. The size of the SW is generally a square, odd-dimensioned matrix. The exception being that of the Roberts Edge Detector which uses two-by-two kernels.

Given some image that is of size MxN that contains information of interest. A three-by-three kernel is selected.

4.3 Edge Detection

As the name implies, Edge Detection is an algorithmic procedure to determine locations within the image that contain an "edge". Edges are defined as sub-regions of an image that contain high contrast. In most cases, edge detection is performed by linearly convolving a gradient or differentiation kernel with a SW from an image. The resulting output image is an intensity map where the higher intensity values indicate the likely presence of an edge.

$$a)\ G_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}, \qquad G_y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

$$b)\ G_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}, \qquad G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

$$c)\ K_1 = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \qquad K_2 = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

Figure 4-1: Examples of the a) Prewitt Operator, b) Sobel Operator, and c) Roberts Cross kernels used for Edge detection.[13][14][15]

More advanced edge detection algorithms perform several different image processing techniques or noise cancelation techniques in order to properly extract edges from an image. As an example, the Canny Edge Detection Filter consists of four major steps: 1) Gaussian Filtering (SW/kernel filter) for noise reduction, 2) Intensity Gradient Computation (SW/kernel filter), 3) Non-maximum suppression, and 4) Two-level thresholding, one for detecting 'weak' edges and one for 'strong' edges.[3]

There are more advanced methods of edge detection, such as the Hough transform, this method relies on a linear transform of the image into a feature space where the intersection of the lines generated in that feature space are indicative of points that are in the same line.

4.4 Noise Filtering

Depending on the type of noise present within an image, different methods of noise removal can be performed. Most of the common noise filters utilize the SW method of image processing where either a linear function is used (such as a smoothing filter) or some non-linear function (such as finding the median value of the window) is applied to reduce the noise present in the local area.[16]

5 ANN Applications to SAR

This section covers the applications of neural networks explored in this thesis. This includes both theory of operation as well as results.

5.1 Edge Detection in Noisy Environments

In one of the research papers for this project, it was shown that an ANN is able to find the edges within a SAR image before any noise removal was performed. Due to the nature of the noise present in SAR images, which is multiplicative speckle noise, most conventional noise-removal methods do not work correctly. There are, however, certain features of an image that are relatively unaffected by speckle noise, but are strong indicators of the presence of an edge. These features are Relative Local Variance (RLV), a Ratio-based Harris Edge Detector (RHED), and the Discrete Cosine Transform-based Edge Detector.

This filtering algorithm relies on the SW method of image processing, as described in the previous section.

5.1.1 Network Input

In order to judge the likelihood of an edge being present within the SW, several noise-invariant parameters must be investigated. According to research, the Relative Local Variance, Ratio-based Harris Edge Detector and Discrete Cosine Transform-based detectors are the least affected by the presence of multiplicative speckle noise.

5.1.1.1 Relative Local Variance (RLV)

The Relative Local Variance (RLV) is a statistical measure of the variance within a SW of an image in relation to the speckle noise variance within the SAR image. The RLV is computed by:

$$RLV = \delta_{ij} = \frac{\sigma_{ij}^2}{\left(\sigma_\mu^2 \bar{I}_{ij}^2\right)}$$

where σ_{ij}^2 is the SW local variance, $\bar{I}_{ij}$ is the local mean, and σ_μ^2 is the speckle variance. This value is useful to compute in edge detection as the RLV is significantly higher on SWs with edges than SWs in homogeneous regions of an image.[17] The value of the local variance and local mean are computed by using MATLAB's built in $var(\,)$ and $mean(\,)$ functions, respectively. Because the SW is *n*-by-*n*, the MATLAB function $reshape(\,)$ is first used to alter the SW to be *1*-by-n so that the $var(\,)$ and $mean(\,)$ functions properly find the true variance and mean within the SW.

5.1.1.2 Ratio-based Harris Edge Detector (RHED)

The Ratio-based Harris edge detector (RHED) is a modification to a conventional Harris edge detector by independently analyzing different quadrants of the selected SW. The output parameter for this edge detection method is:

$$R_{ij} = \min(R_1, R_2)$$

where

$$R_1 = Det(M_1) - k \cdot Tr^2(M_1)$$

and

$$R_2 = Det(M_2) - k \cdot Tr^2(M_2)$$

The value k, is an empirical constant that is defined as $k = 0.04$, $Tr(M) = \lambda_1 + \lambda_2$ (λ being the eigenvalues of M), and the matrices M_1 and M_2 are:

$$M_1 = \begin{bmatrix} I_h^2 & I_h I_V \\ I_h I_V & I_V^2 \end{bmatrix}, M_2 = \begin{bmatrix} I_{md}^2 & I_{md} I_{ad} \\ I_{md} I_{ad} & I_{ad}^2 \end{bmatrix}$$

I_h, I_V, I_{md}, I_{ad} are the ratios of means in different SW parts: left and right, top and bottom, under the main diagonal and upper the main diagonal, under the side diagonal and upper the side diagonal, respectively.[17]

5.1.1.3 Discrete Cosine Transform-based Edge Detector

The Discrete Cosine Transform (DCT) spectrum is a mathematical operation defined as:

$$y(k) = w(k) \sum_{n=1}^{N} x(n) \cos\left(\frac{\pi(2n-1)(k-1)}{2N}\right) \quad for\ k = 1,2,\dots,N$$

where x is a N dimensional input such that $x(n)$ is index n of N and where:

$$w(k) = \begin{cases} \frac{1}{\sqrt{N}} & k = 1 \\ \sqrt{\frac{2}{N}} & 2 \le k \le N \end{cases}$$

where the algorithm is indexed at 1.

For this project, the DCT of the SW is calculated using the MATLAB function $dct(\,)$. If the window is located on an edge within the image, there will be some number of DCT-coefficients in the spectrum that exceed a certain threshold. This number of coefficients that exceed the threshold is greater in regions where edges are located and fewer in homogeneous regions.[17]

5.1.2 Modified Network Input

With the first iteration of this design, there was a significant problem with network convergence. It was determined that the input parameters, as specified from research, would generate infinite values for zero-inputs. In order to correct for this, a couple modifications were made. The one parameter that was not affected by zero-inputs was the DCT and was therefore left alone.

5.1.2.1 Modified RLV

Instead of calculating the RLV, as specified by the previous section, only the value of the local variance was used in most test cases. This was done by using the $reshape(\,)$ and $var(\,)$ function in MATLAB. It was observed that this value was in fact an effective parameter to determine the presence of an edge in a noisy image.

In the several test cases that use the RLV as defined, a small value, generally the $eps(\,)$ declaration in MATLAB, was added to the calculated mean and variance values to avoid divide-by-zero errors.

5.1.2.2 Modified RHED

Following the description given in the paper of the RHED algorithm the following algorithm was implemented in MATLAB. This is not considered a conventional implementation of the RHEAD from additional reading [18] however it proved effective:

$$I_h = \frac{mean(SW_{Top\ Rows})}{mean(SW_{Bottom\ Rows})}$$

$$I_V = \frac{mean(SW_{Left\ Rows})}{mean\left(SW_{Right\ Rows}\right)}$$

$$I_{md} = \frac{mean(SW_{Values\ above\ the\ main\ diagonal})}{mean(SW_{Values\ below\ the\ main\ diagonal})}$$

and

$$I_{ad} = \frac{mean(SW_{Values\ above\ the\ side\ diagonal})}{mean(SW_{Values\ below\ the\ side\ diagonal})}$$

where SW is the 5×5 window of the input image. The final step, in order to avoid the possibility of a zero-mean SW, a pixel value of one (1) was added to each cell of the SW before computation was performed.

5.1.3 Network Configuration

For the edge detector, the BP algorithm was utilized in a multi-layer perceptron ANN. The network contained three layers: the input layer, one hidden layer, and the output layer. The output layer contains two neurons, one for "edge region found" and one for "homogeneous region found", and there are five hidden neurons. The input layer was of size three, for the three metrics stated in the previous sections.

5.1.4 Training Set Generation

The initial simulated training images used to train the ANN were generated using Adobe Photoshop. Then, to simulate the multiplicative noise, the $imnoise(\,)$ function in MATLAB was used. The three training images are named "Striped" (also referred to as the "Easy" image), "Checkerboard", and "Hard Striped". The "Hard Striped" image was named because there are smaller differences in grayscale intensity between the stripes. This, in turn, gives smaller gradients to detect edges from.

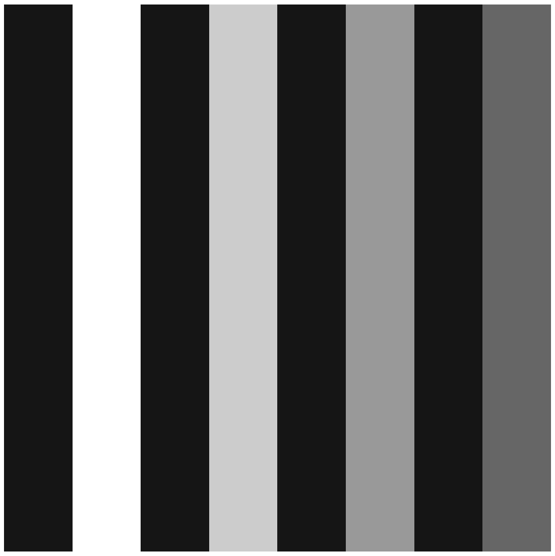

Figure 5-1: Training Set 1 "Striped" image without noise

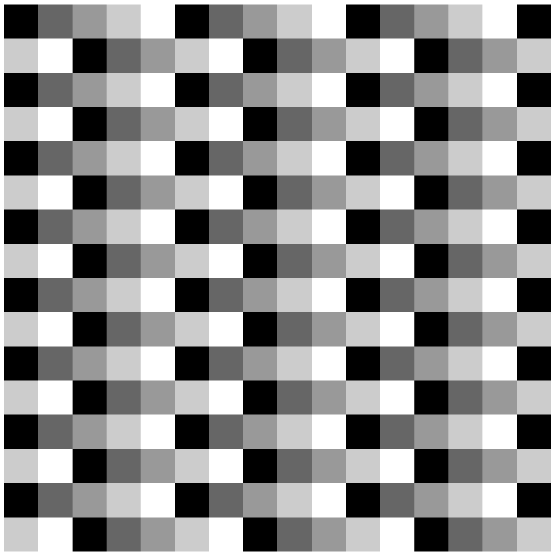

Figure 5-2: Training Set 1 "Checker Board" image without noise

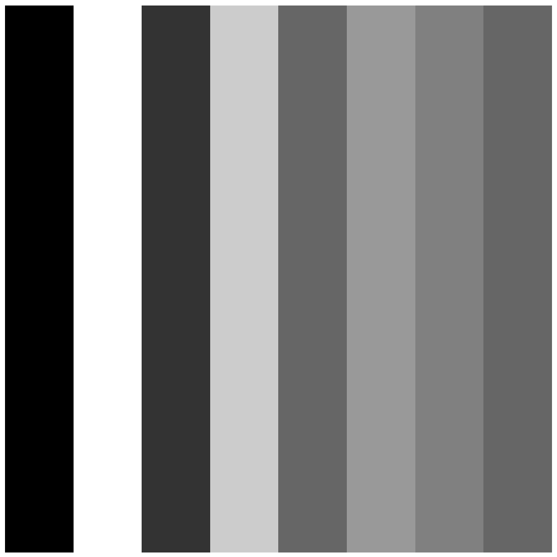
Figure 5-3: Training Set 1 "Hard Striped" image without noise

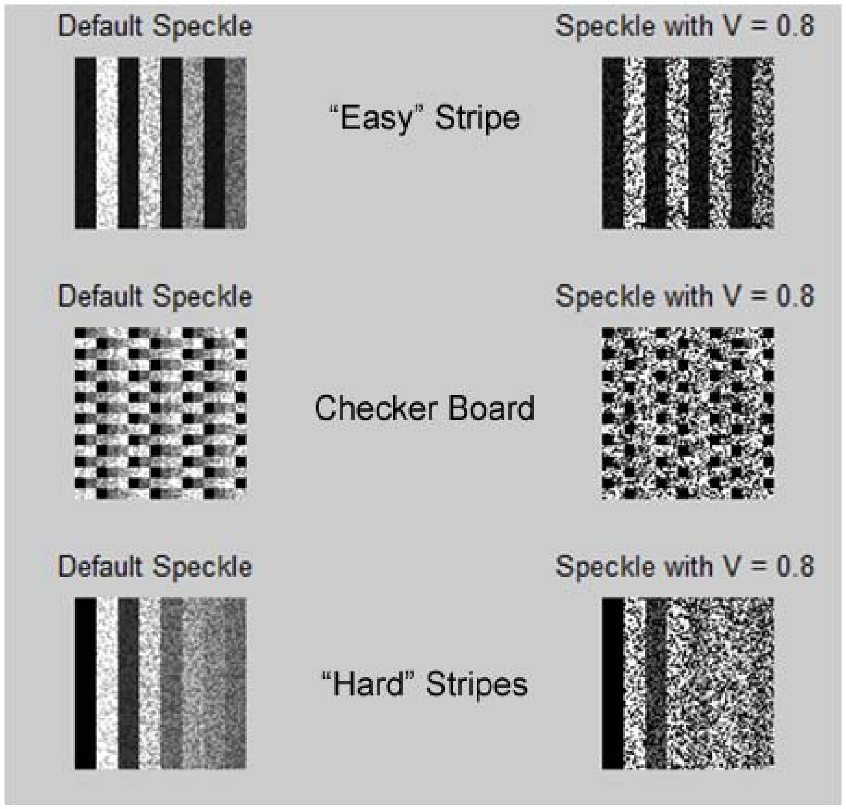

Figure 5-4: Training Set 1 Example of the introduction of speckle noise. V represents speckle variance. Default Variance is 0.04.

In order to train the network, small 5 × 5 windows of the noisy images were taken and used in the BP algorithm. These samples were collected manually, as it was intended to train the network to detect user-defined edges and not to train the program to follow any pre-defined edge detection program model. The sample library ranged from 30 images, 15 edge samples and 15 homogeneous samples, to more than 100 images with mixed quantities of edge and homogeneous samples. It is to be noted that the algorithm was using samples from the "Easy Striped" and "Checkerboard" images, not from the "Hard" image.

5.1.5 Single Run Output and Performance Analysis

With a relatively small sample set, it can be seen that the network was in fact learning to detect the given image samples. As a test of merit, the ANN detector was tested on both the simulated noisy images as well as a MSTAR image. Figure 5-5 shows the comparative results of the common Canny edge filter versus the ANN detector, and Figure 5-6 shows the same ANN detector on the MSTAR image.

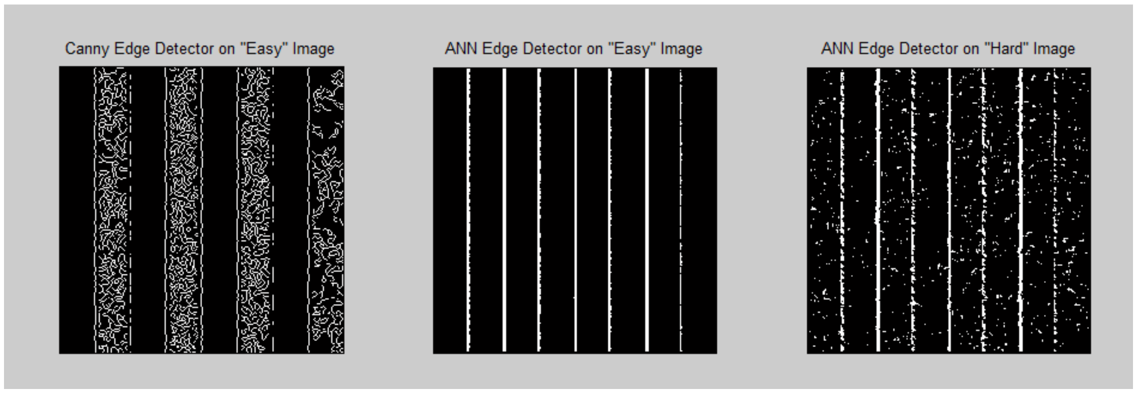
Canny Edge Detector on "Easy" Image
ANN Edge Detector on "Easy" Image
ANN Edge Detector on "Hard" Image

Figure 5-5: Output of Edge Detectors on Noisy Images. From left to right: Canny Edge Detector on "Easy" image, the ANN detector on the "Easy" Image, and the ANN detector on the "Hard" image.

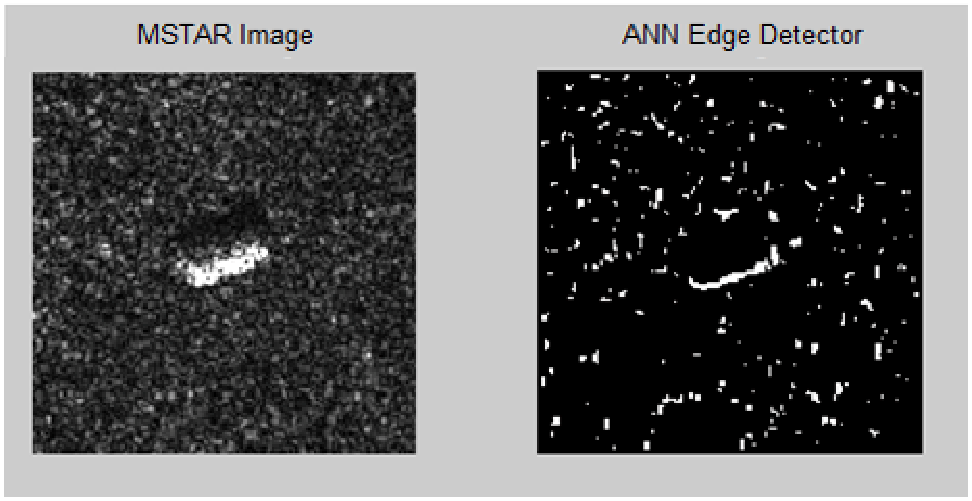

Figure 5-6: ANN Edge Detector trained with a large data set.

While in this set, it is obvious that the ANN detector isn't perfect, however the output was generated without any previous noise-removal techniques. It can be seen that the ANN program was able to extract at least half of the edges of the target object.

After several iterations and trials, it was found that a smaller, more focused training set results in significantly better performance for the edge detection versus noise. The new test set also included several edge points extracted, manually, from one MSTAR image. The noise rejection and false-positives were significantly reduced.

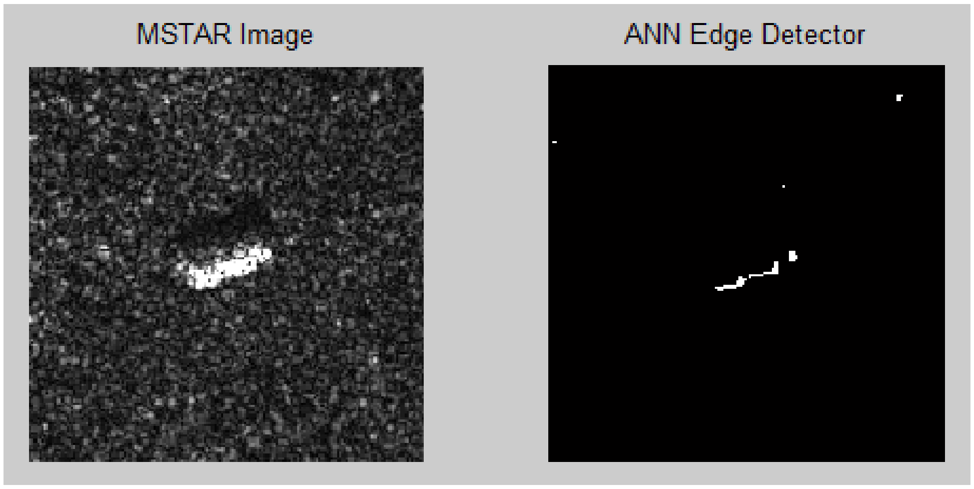

Figure 5-7: ANN Edge Detector trained with a small data set.

5.1.6 Rotational Edge Detection Method

It was found in experimentation that the algorithm, as described, would only detected half of the target's edges. More specifically, it was noted that the algorithm only detected edges on the bottom of the target within the image. It was theorized that if the image were rotated and the edge detector be run again, that perhaps the previously undetected edges would be found. It was theorized that if the image were to be rotated several times, being analyzed by the ANN each time, that perhaps the ANN would pick up the missing edges. A flowchart of the proposed method can be seen in Figure 5-8. Rotations of 90° were selected to prevent any image distortion by using smaller-angled rotations.

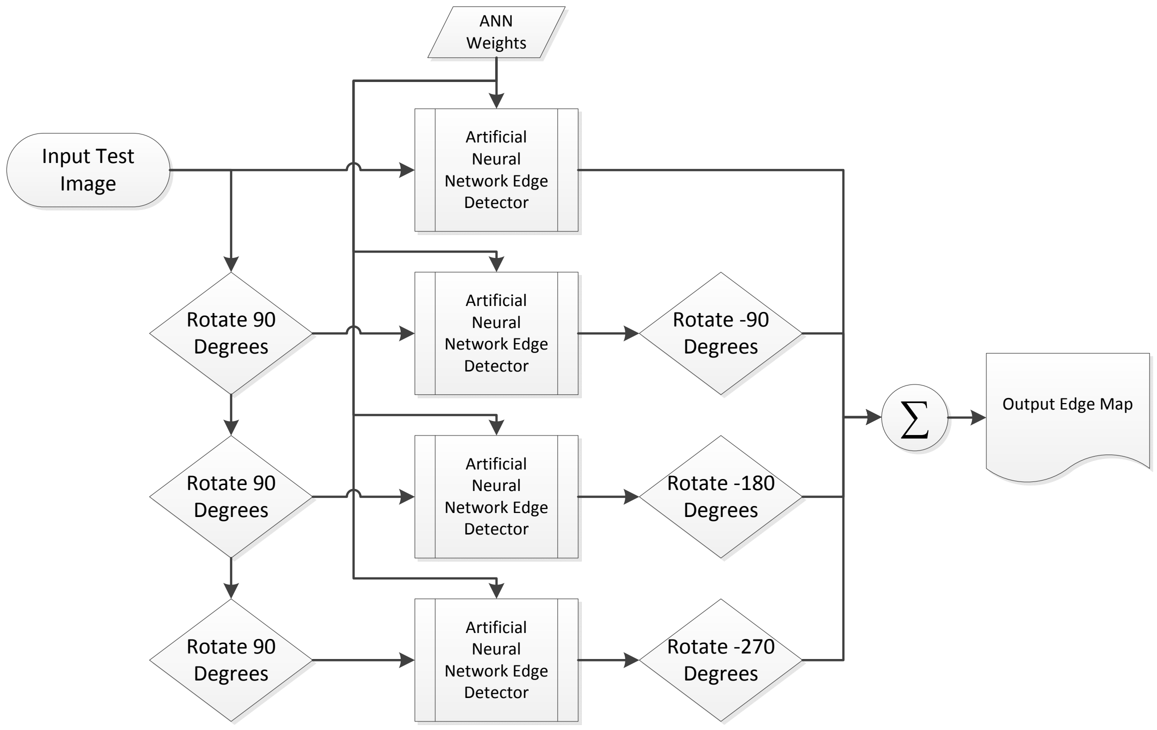
ANN Weights
Input Test Image
Artificial Neural Network Edge Detector
Rotate 90 Degrees
Artificial Neural Network Edge Detector
Rotate -90 Degrees
Rotate 90 Degrees
Artificial Neural Network Edge Detector
Rotate -180 Degrees
Rotate 90 Degrees
Artificial Neural Network Edge Detector
Rotate -270 Degrees
∑
Output Edge Map

Figure 5-8: Example of Rotational Edge Detector. (Rotation count of four in this case)

5.1.7 Rotational Edge Detector Results and Analysis

As it can be seen in Figure 5-9 and Figure 5-10. The proposed rotation method does indeed detect edges around the entire object. While the edge map is not a contiguous line, it is a substantial improvement over the results of a conventional edge detector, demonstrated using a Canny Edge Detector.

Another useful feature of this method is the detection of internal features of the target. This is especially visible in Figure 5-10 where the ANN rotational edge detector is able to find features on the top of the vehicle. In this case, it is the fact that the front of the ZSU-23-4 is lower than the turret that sits above it.

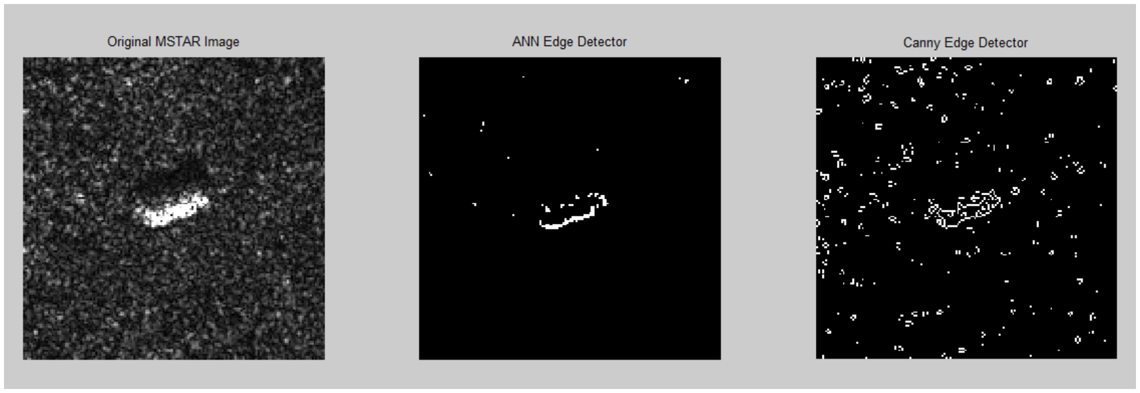

Figure 5-9: Edge Detection Comparison for a 2S1 Gvozdika

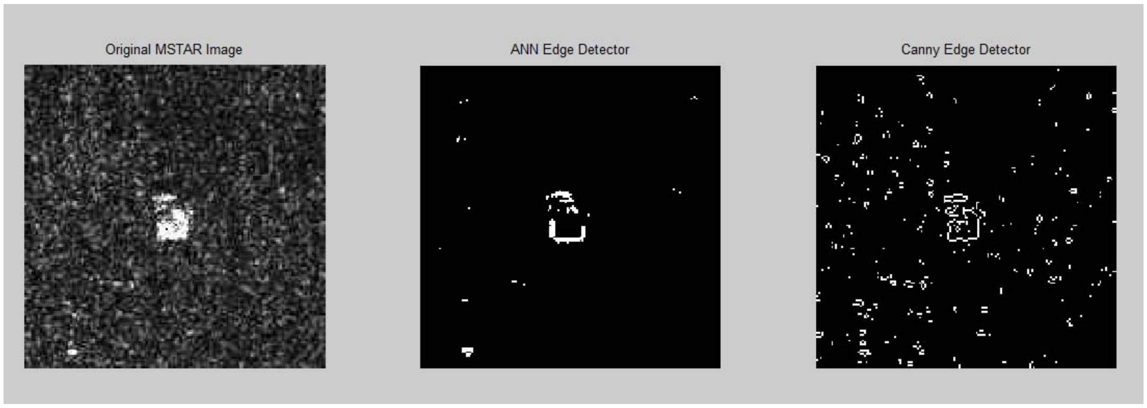

Figure 5-10: Edge Detection Comparison for a ZSU-23-4 Self Propelled Anti-Aircraft Weapon System

Most importantly it can be seen that the ANN detector has significantly fewer false-positives than a conventional edge detection filter. For a full listing of all of the edge detector outputs, please refer to Appendix C and Appendix E.

5.2 Pattern Recognition in Information-Deprived Environments

More often than not the images generated from SAR data do not fully represent all of the information within contained within the scene. Commonly, a collection of sophisticated algorithms and/or human interaction is required to reconstruct or process the information. The concept explored in this section is the use associative-memory ANNs to attempt and reconstruct the missing information or determine the nearest geometric fit of the target.

5.2.1 Data Organization

The type of ANN experimented with in this section is specifically the Discrete Hopfield Network model (DHN). As such, the inputs have to be adjusted to range, discretely, from -1 to +1. The input MSTAR images are first thresholded using MATLAB's $im2bw(\,)$ function, then all zero-valued pixels are changed to have a value of -1. This is allowable as the image data is converted from an 8-bit unsigned integer to a floating point (double) number in the MATLAB environment.

5.2.2 Simulated Results

In order to initially test the functionality of the DHN, several simulated cases were run. A small training data set of geometric shapes were created and then various noisy or incomplete version of the shapes were run through the network. The first test, seen in Figure 5-11 was that of a circle and a square. The network was trained using the shapes in the first row, the second row contains the incomplete versions of the shapes, and then the third row contains the images that were generated by the DHN.

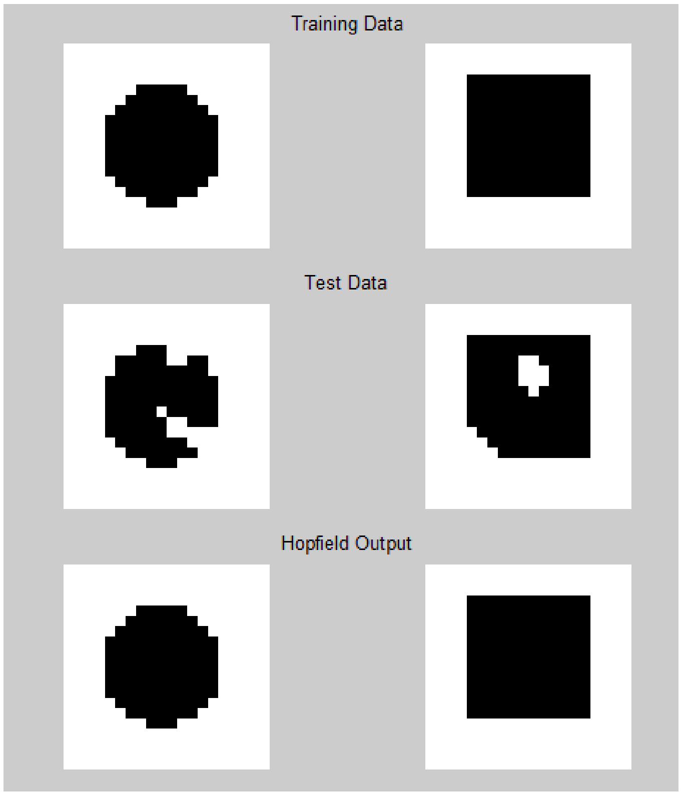

Figure 5-11: Example 1 of the DHN Process

While this example is effective in showing the effectiveness of the DHN, it is a very basic case. The next experiment on simulated data introduces noise to the image before running through the DHN.

The image series in Figure 5-12 is that of an experiment conducted on a more difficult set of images. The first task of making the images more difficult is that both shapes care commonalities in features, one being a rectangle and the other is a square. The incomplete versions of the trained shapes were purposely created such that the two shapes would, theoretically, be more difficult to distinguish from each other.

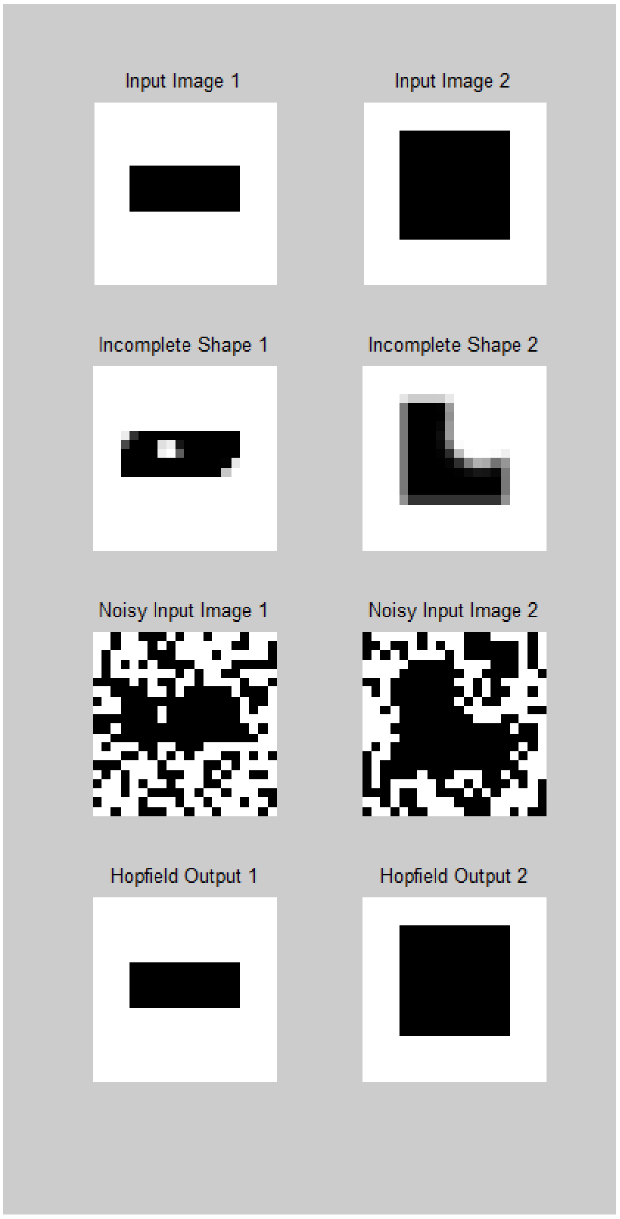

Figure 5-12: Example of Discrete Hopfield Network with speckle noise

As it can be seen, the DHN is able to distinguish the two shapes even in an exceptionally noisy environment and is able to recover the fundamental memories (shapes).

Since the simulated results definitively show that a DHN is capable of reconstructing incomplete shapes from noisy or incomplete data, the next logical step is to apply the designed DHN to actual SAR images.

5.2.3 MSTAR Results

A substantially larger data set was developed in order to do a rectangular best-fit of the available data. Because of the nature of the variability of SAR data, several different methods were developed.

The first step in applying the network is first some simple image thresholding in order to generate the binary images necessary to be inputted into the DHN. Once the images have been pre-processed, they are resized for the DHN and run through the network.

The first data set run was for a collection of pre-selected SAR images that were all "horizontally" aligned. As the first DHN used was trained for only horizontal rectangular fitting, this was a necessary manual interaction. Figure 5-13, Figure 5-14, and Figure 5-15 demonstrate the sequence of operation for the DHN.

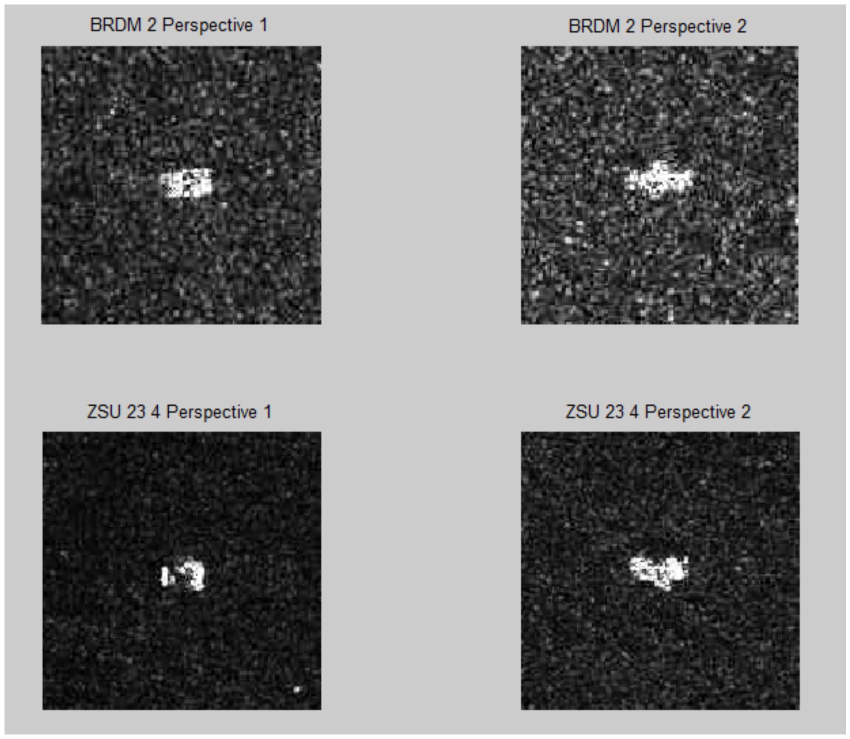

Figure 5-13: Selected SAR Target Images for DHN

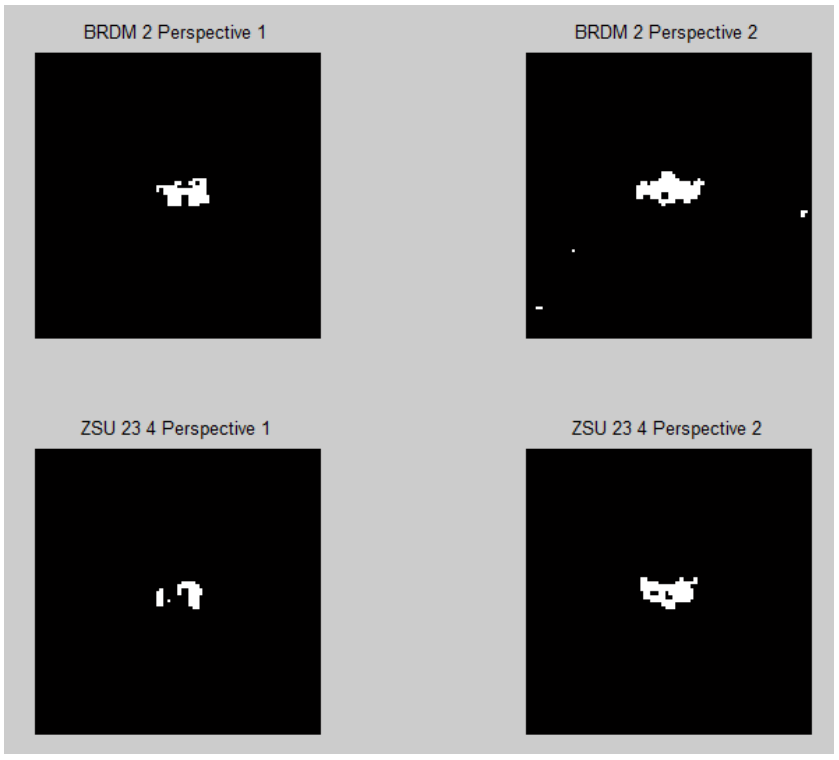

Figure 5-14: Selected SAR Target Images after Thresholding

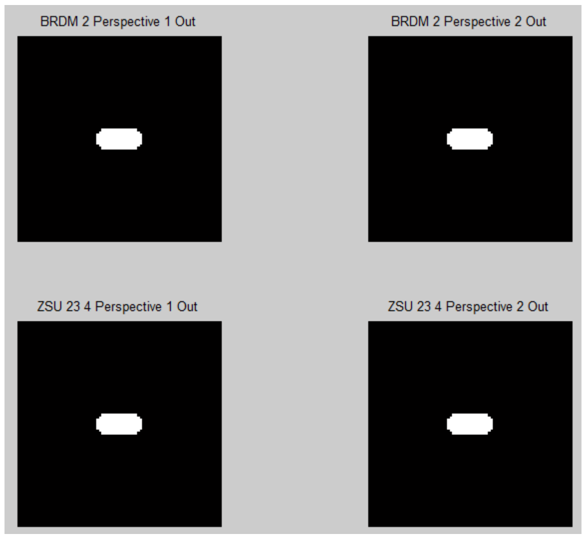

Figure 5-15: Output of Rectangular Fit Hopfield Network

The resulting output is a good approximation of the area occupied by the target. As can be seen in Figure 5-16, the targets are nearly totally encompassed by the semi-rectangular fit.

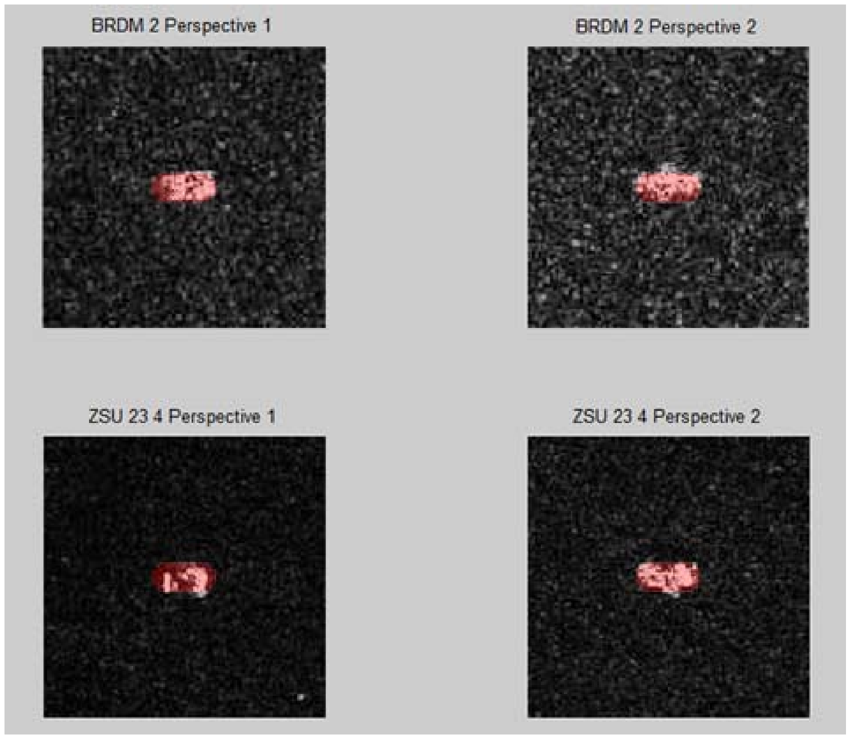

Figure 5-16: Overlay of Hopfield Output (red) over input MSTAR Images

5.3 Automatic Rotation Detection

The second program designed uses the same DHN architecture, however with a novel idea for the training set along with a new complementing algorithm developed in this thesis. The purpose of this method is to detect the relative rotation of the target within the target region.

5.3.1 Training Data and Algorithm

Instead of using solid rectangular shapes, like those that were used to train the previous network, a collection of disjoint collinear rectangles were used. An example of these collinear rectangles are seen in Figure 5-17. Full rectangles were used in the first iteration, however the network would always converge to circles for outputs due to the shared midpoints of the rectangles. (For more information on those results experiment see Appendix B.)

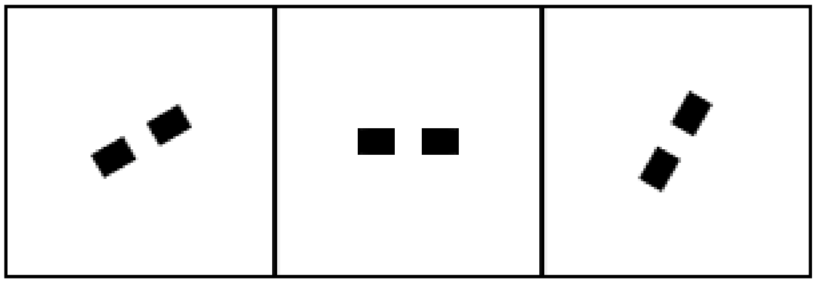

Figure 5-17: Example of Rotation Detector Training Set

As before, the MSTAR data used was pre-processed in order to generate binary input images to the network. These pre-processed images are then fed into the rotation-trained Hopfield Network. From these outputs of the Hopfield network, seen in Figure 5-21, it can be seen that there are higher concentrations of pixels that are co-linear with each other along the axis of rotation. By performing an eight-point connected component analysis, executed by using the MATLAB command $bwlabel(\,)$, the midpoints of the two largest clusters were found. The slope of the line that is formed by the midpoints is computed and used to determine the relative angle of the target with respect to the horizontal axis. A visualization of this algorithm works is demonstrated in Figure 5-18.

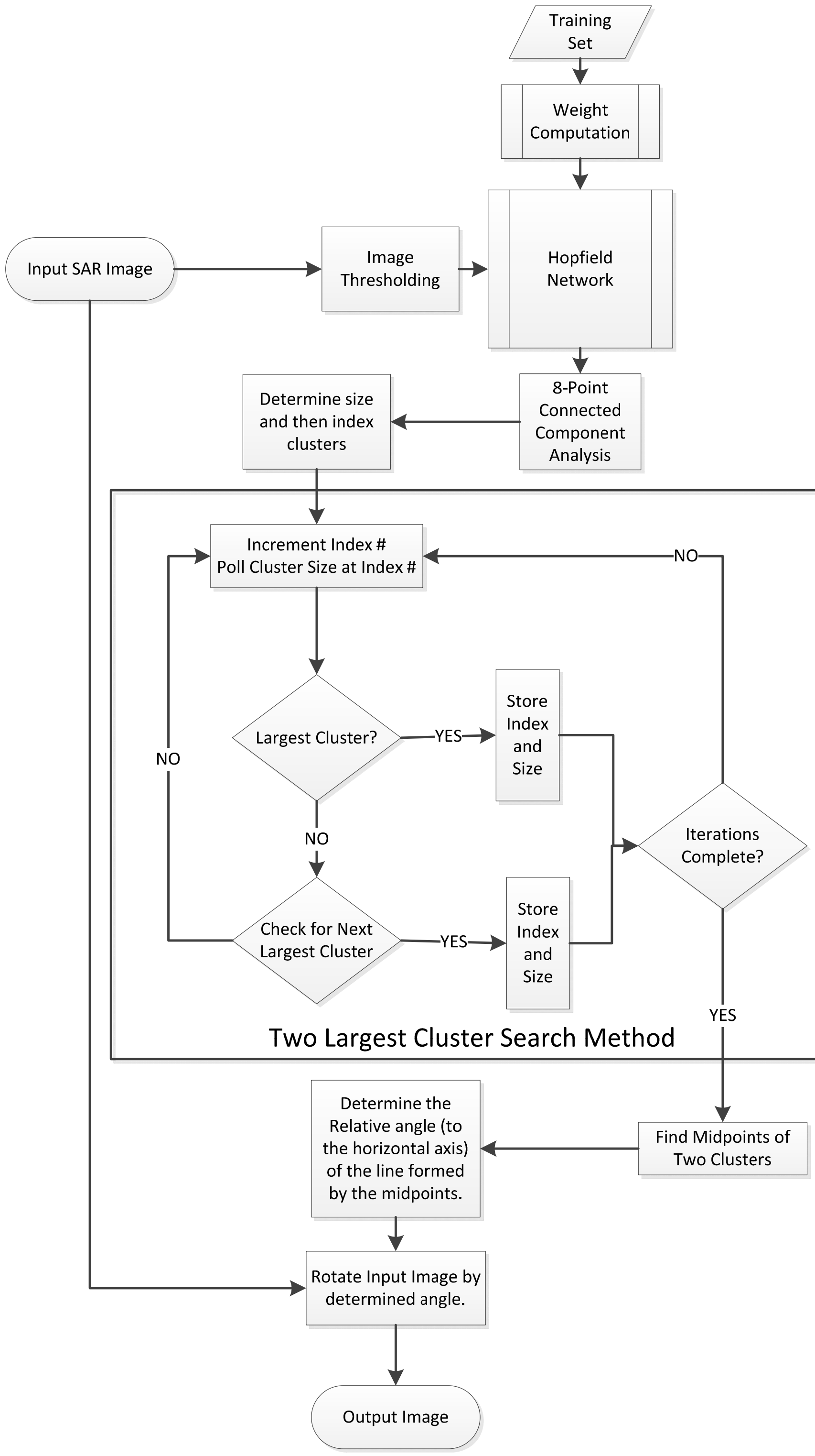
Training Set
Weight Computation
Input SAR Image
Image Thresholding
Hopfield Network
8-Point Connected Component Analysis
Determine size and then index clusters
Increment Index #
Poll Cluster Size at Index #
NO
Largest Cluster?
YES
Store Index and Size
NO
NO
Iterations Complete?
Check for Next Largest Cluster
YES
Store Index and Size
YES
Two Largest Cluster Search Method
Determine the Relative angle (to the horizontal axis) of the line formed by the midpoints.
Find Midpoints of Two Clusters
Rotate Input Image by determined angle.
Output Image

Figure 5-18: Automatic Rotation Detector Algorithm Flowchart

5.3.2 MSTAR Results

The results of the automatic rotation are presented in order of the process are presented here in the order that they were processed by the algorithm. This includes Figure 5-19, Figure 5-20, Figure 5-21, Figure 5-22, and Figure 5-23 in order.

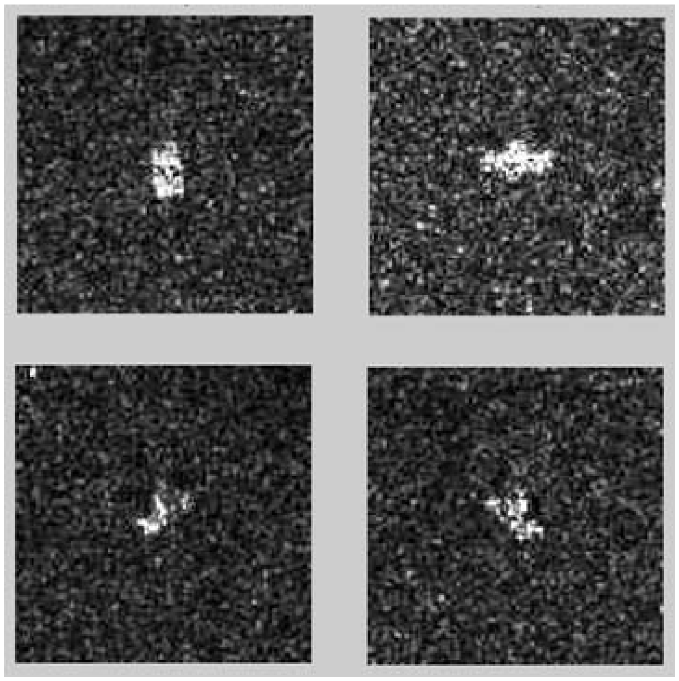

Figure 5-19: Input Four poses of the BRDM 2 MSTAR Target for Rotation Detection.

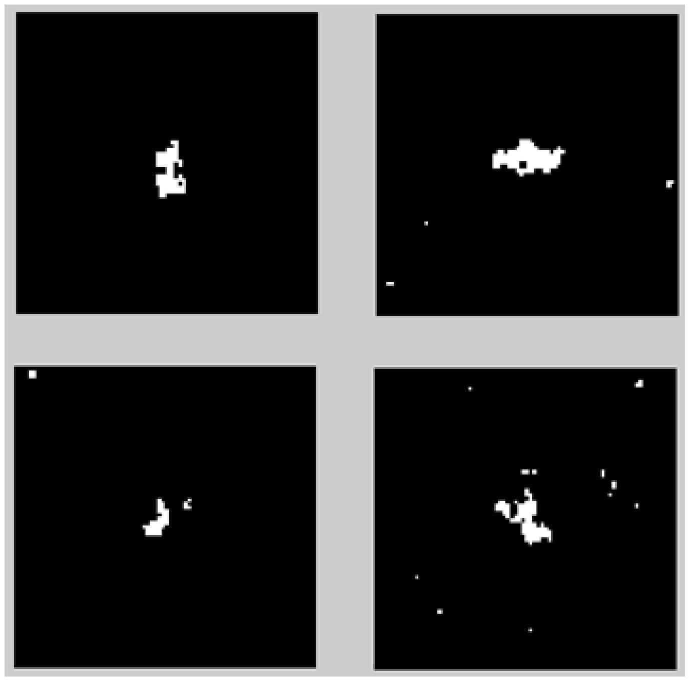

Figure 5-20: Four poses of the BRDM 2 MSTAR Target after thresholding, used as DHN Input.

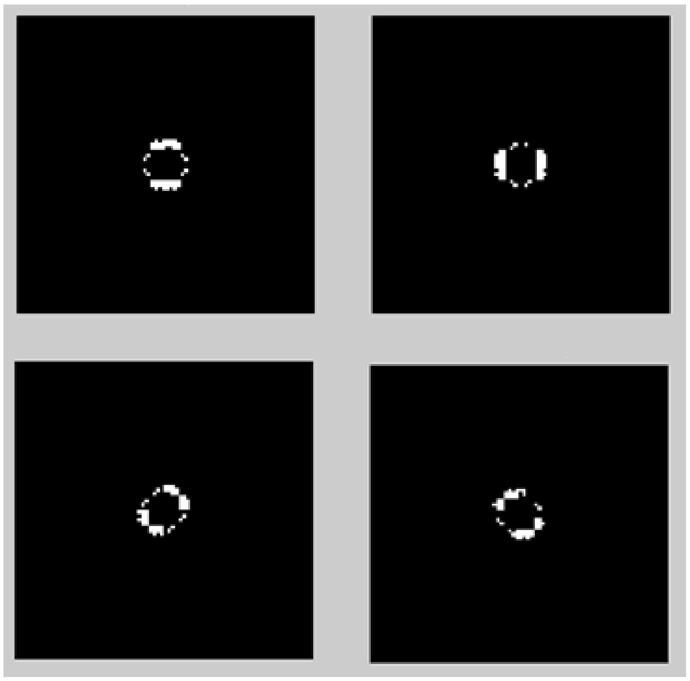

Figure 5-21: Final Stable Hopfield Network Output State

The result of this algorithm is demonstrated in a series of figures. Figure 5-19 is the set of original input target images. Figure 5-22 represents the output of the Hopfield Network and contains the colored center points indicating the two detected clusters.

Figure 5-23 shows the original images rotated such that all of the images are aligned horizontally.

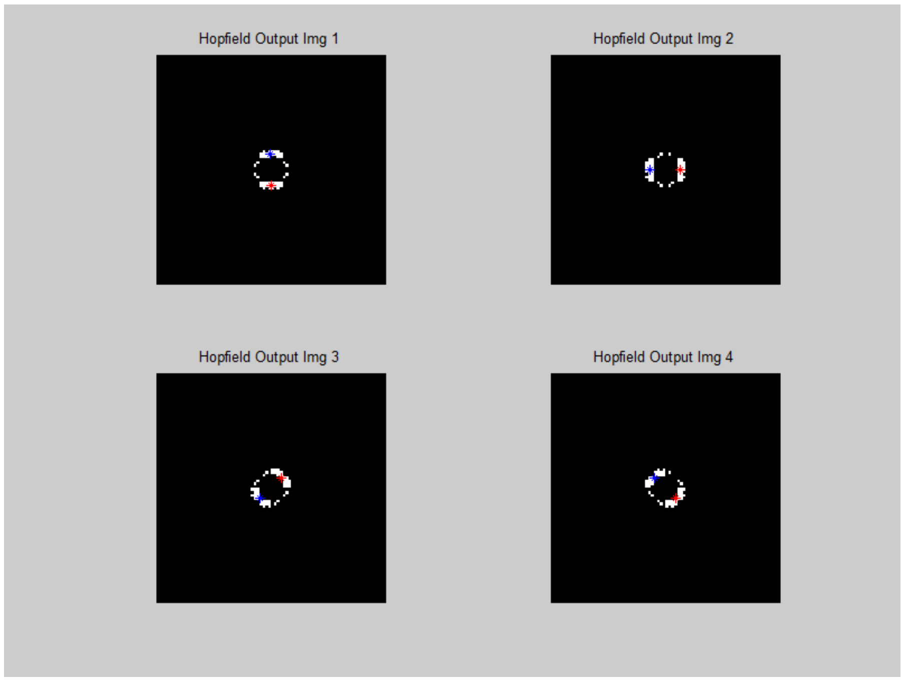

Figure 5-22: Output of the Two Largest Cluster Search Method (Indicated by Red and Blue Dots)

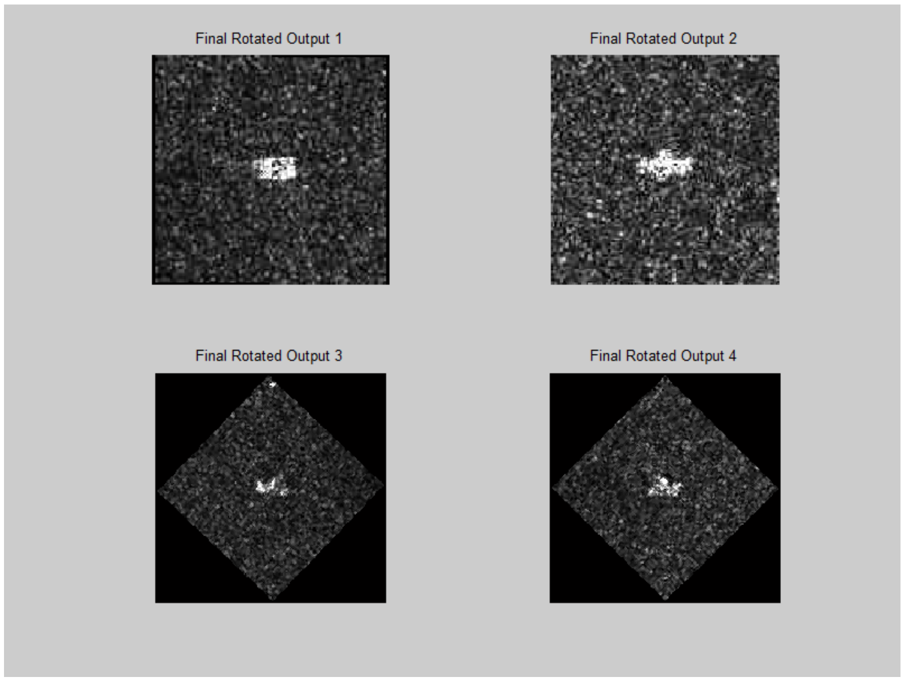

Figure 5-23: Rotated, Horizontally Aligned Input Images using the angle produced by the Rotation Detection Hopfield Network

6 Future Work Research: Deep Learning Networks

A type of ANN network configuration, that is being used by companies such as Google for complex computation, that uses multiple types of ANNs simultaneously. The concept of a Deep Learning Network (DLN) is that of a neural network that is able to read in any collection of inputs, and then return very high level feature information. For example, Google uses DLNs for voice recognition and image searching. Their DLNs are able to recognize the features within an image for their image searching engines and are able to detect language with an error rate of 7%.[19] Due to time limitations, as well as the late discovery (and availability of documentation) of this topic, this chapter is simply a brief, topographical overview of a couple published techniques and technologies.

6.1 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a class of multi-layer neural networks that are particularly designed for use on two-dimensional data. (Images and videos, for instance.) One of the first successful techniques in the DLN field, CNNs are able to achieve very robust performance with minimal data preprocessing requirements. The way a CNN works is by operating on small sections of the image (or data array) at a time. These sections are called local receptive fields (LRFs). Each of these local receptive fields are used as the inputs to the lowest layer in the CNN hierarchy. A visualization o of this type of network is displayed in Figure 6-1. The ANN/Trainable Filters are indicated by the "C#" designation and their outputted feature maps are designated by "S#". The LRF is inputted into the first trainable filter layer, C1, which in turn produces a feature map S2. This process is repeated as S2 is the input to trainable filter C3. In turn producing the final feature map that is combined and fed into a "conventional" neural network structure (a multi-layer perceptron network or similar style).[20]

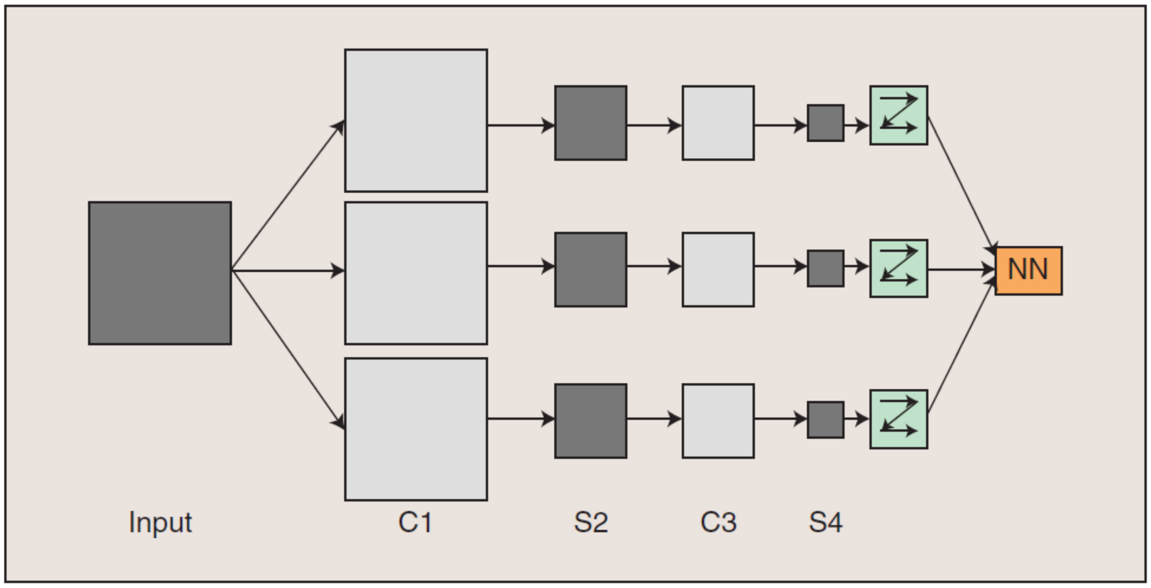

Figure 6-1: Conceptual example of a Convolutional Neural Network's function.

A CNN's ability to work with two-dimensional data inherently makes it a good candidate for working with SAR signal processing. Theoretically, such a learning machine could perform object recognition directly from raw data collected from the SAR antenna array, if properly trained.

6.2 Deep Belief Networks (DBN)

The other style of DLN explored in a top-level analysis is that of the Deep Belief Network (DBN). These types of networks are probabilistic generative models that do not behave like traditional neural networks. DBNs are composed of several layers of Restricted Boltzmann Machines. The term "Restricted" refers to the fact that the network is limited to one visible layer and one hidden layer. Also, each visible neuron must connect to a hidden neuron. (In contrast, an unrestricted Boltzmann Machine allows for connections between hidden neurons.) The remaining levels of the DBN include a collection of "top-down" generative weights and an associative memory block.

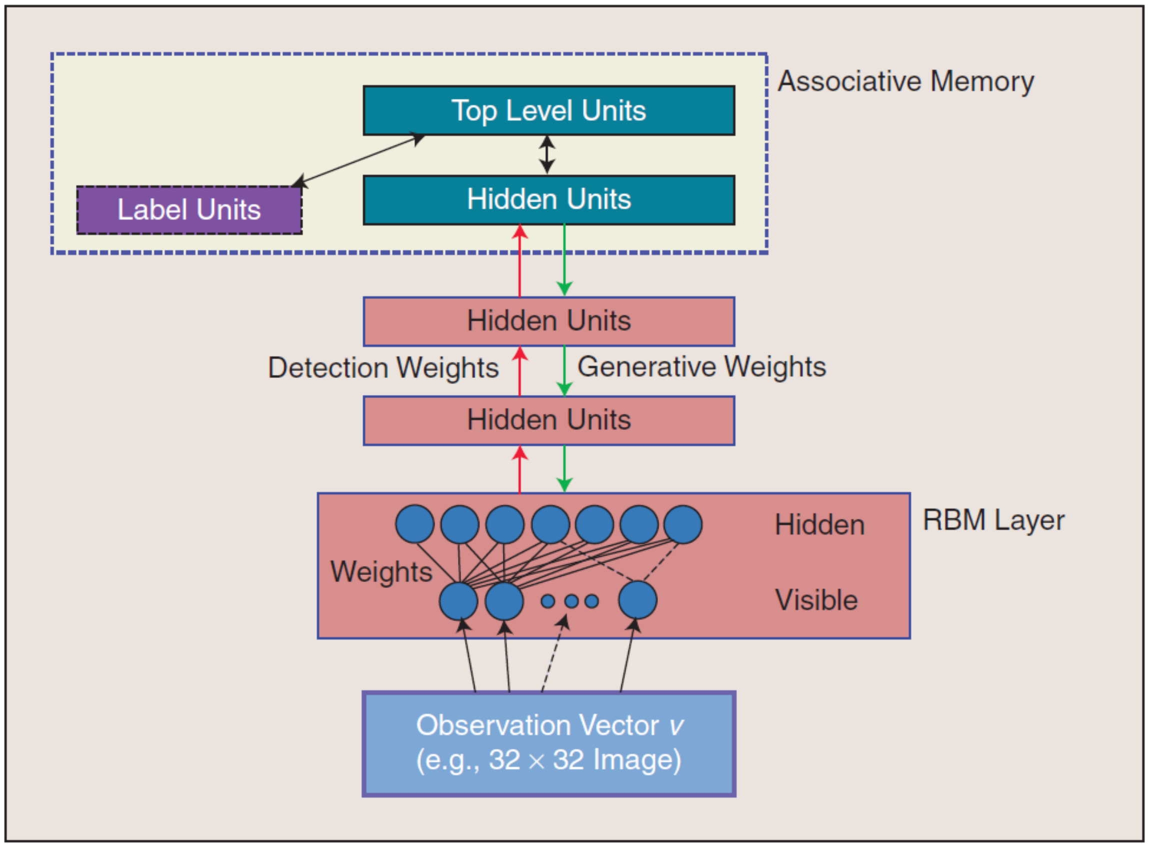

Figure 6-2: Illustration of the Deep Belief Network Configuration

7 Conclusions and Future Work

7.1 ANN Edge Detection

The rotational edge detector was shown to be successful in being able to detect the edges of the targets in question. What is even more advantageous is its ability to find internal features of the target. Since edges are a very important feature in ATR, enhancing the detection using neural networks should greatly help to improve detection accuracy. The drawback of the way that this system was implemented is the long learning times as well as the consistency of learning sessions. The consistency of learning result is always highly dependent on the selected training set as well as the initial neuron connection weights.

Future work should be done in either determining the optimal training sets to produces the desired results or finding a more efficient method of training the network. One such possibility would be to re-write the algorithm to function on a GPU processor instead of a CPU. The parallel stream-processing power of GPU's would substantially improve the speed of execution of the program.

7.2 Object Reconstruction

The explored application of the Discrete Hopfield Network model was only partially successful in accomplishing the task of incomplete information reconstruction. When used for approximating a rectangular fit for the SAR data, it did do so with a fair amount of accuracy. However it did have issues when the learning sets were too large or had too many fundamental memory vectors that shared coordinates.

The next logical step in research for performing pattern recognition and object reconstruction would be to begin exploring the Continuous Hopfield Model and the "Brain-in-a-Box" style of attractor networks. These networks may be better suited to processing SAR data than their discrete counterparts.

7.3 Automatic Rotation Detection Algorithm

The application of a Hopfield Network and the new angle detection algorithm developed in this thesis does in fact succeed at the goal of automatically recognizing the relative angle of the target within the SAR image. The specific drawback of this method is that the SAR image needs to be pre-processed before using. However, despite this issue, the algorithm is effective.

It is conceivable that if used in conjunction with the previous two algorithms developed in this thesis, it is conceivable that an efficient and accurate ATR algorithm can be developed. As such, this method could be applied to 3D modeling techniques previously explored at Cal Poly[3] where the rotation can be detected for multiple images, and then the resulting 3D models could be combined and averaged to give a more complete model.

7.4 Deep Learning Networks

This is, in the author's opinion, what should be the ultimate goal in the processing of SAR data and imagery. With a correctly trained DLN, a target can be completely identified, automatically, directly from the raw radar data. While there are systems that have been designed to perform this purpose using various filtering and image processing techniques, the DLN gives the possibility more rapid results.

A simple example of what can be done in future research would be to have several low level BP network feature detectors (edges, corners, etc.) and have all of these lead into a larger BP network that learns higher level features (closest geometric shape, number of sides, etc.). This large multi-network structure can then be fed into an associative memory unit that would then further refine the results based off of a pre-learned library of knowledge.

7.5 Summary and Final Thoughts

The material in this thesis explored the ground work for further study into the applications of Artificial Neural Networks to the study of Synthetic Aperture Radar data analysis. The methods of using Back Propagation ANNs gives a step forward in the design of efficient noise-invariant edge-detection algorithms. Because of the versatility of such a network, it can be quickly re-purposed to meet the application needs.

In an attempt to overcome one of the largest obstacles in remote sensing, i.e. incomplete information, this thesis explored the use of associative memory neural networks. The choice network explored is the Discrete Hopfield Network Model. While this network did show

There is no doubt that future research into the applications of ANNs to SAR data will result in improved results as well as more efficient networks.

8 Bibliography

[1] C. F. Guilas, "Hausdorff Probabilistic Feature Analysis in SAR Image Recognition," San Luis Obispo, 2005.

[2] D. A. Cary, "Automatic Target Recognition via Optimal Rectangular Fit," San Luis Obispo, 2007.

[3] J. R. Hupton, "Three-Dimensional Target Modeling with Synthetic Aperture Radar," San Luis Obispo, 2009.

[4] "Radar - Wikipedia, the free encyclopedia," Wikimedia, Inc., 12 May 2013. [Online]. Available: http://en.wikipedia.org/wiki/Radar. [Accessed 2013].

[5] "Synthetic aperture radar - Wikipedia, the free encyclopedia," Wikimedia, Inc., 16 May 2013. [Online]. Available: http://en.wikipedia.org/wiki/Synthetic_aperture_radar. [Accessed 2013].

[6] M. Soumekh, Synthetic Aperture Radar Signal Processing with MATLAB Algorithms, New York: John Wiley & Sons, Inc., 1999.

[7] "Speckle noise - Wikipedia, the free encyclopedia," Wikimedia, Inc., 5 October 2012. [Online]. Available: http://en.wikipedia.org/wiki/Speckle_noise. [Accessed 2013].

[8] "MSTAR Public Targets," April 2013. [Online]. Available: https://www.sdms.afrl.af.mil/index.php?collection=mstar&page=targets. [Accessed 2013].

[9] "Wikipedia, the free encyclopedia," Wikimedia Inc., 2013. [Online]. Available:

www.wikipedia.org. [Accessed 2013].

[10] S. Haykin, Neural Networks and Learning Machines, Third Edition ed., Upper Saddle River, New Jersey, 07458: Pearson Education, Inc., 2009.

[11] F. Rosenblatt, Principles of Neurodynamics, Washington D.C.: Spartan Books, 1962.

[12] W. S. McCulloch and W. H. Pitts, "A Logical Calculus of the Ideas Immanent in Nervous Activity," *Bulletin of Mathematical Biophysics,* no. 7, pp. 115-133, 1943.

[13] "Prewitt Operator - Wikipedia, the free encyclopedia," Wikimedia, Inc., 5 April 2013. [Online]. Available: http://en.wikipedia.org/wiki/Prewitt. [Accessed 2013].

[14] "Sobel Operator - Wikipedia, the free encyclopedia," Wikimedia, Inc., 28 April 2013. [Online]. Available: http://en.wikipedia.org/wiki/Sobel_operator. [Accessed 2013].

[15] "Robert Cross - Wikipedia, the free encyclopedia," Wikimedia, Inc., 28 February 2013. [Online]. Available: http://en.wikipedia.org/wiki/Roberts_Cross. [Accessed 2013].

[16] "Noise reduction - Wikipedia, the free encyclopedia," Wikimedia Inc., 28 May 2013. [Online]. Available: http://en.wikipedia.org/wiki/Noise_reduction. [Accessed 2013].

[17] A. Naumenko, V. Lukin and K. Egiazarian, "SAR-Image Edge Detection Using Artificial Neural Network," in *International Conference on Mathematical Methods in Electromagnetic Theory*, Kharkov, Ukraine, 2012.

[18] "Corner Detection - Wikipedia, the free encyclopedia," Wikimedia Inc., 1 June 2013. [Online]. Available: http://en.wikipedia.org/wiki/Corner_detection. [Accessed

2013].

[19] J. Turrell, "New Techniques from Google and Ray Kurzweil are Taking Artificial Intelligence to Another Level | MIT Technology Review," 23 April 2013. [Online]. Available: http://www.technologyreview.com/featuredstory/513696/deep-learning/. [Accessed 2013].

[20] I. Arel, D. C. Rose and T. P. Karnowski, "Deep Machine Learing - A New Frontier in Artificial Intelligence Research," *IEEE Computational Intelligence Magazine,* 2010.

[21] I. G. Cumming and F. H. Wong, Digital Processing of Synthetic Aperture Radar, Norwood, MA 02062: Artech House, INC., 2005.

[22] G. W. Stimson, Introduction to Airborne Radar, Second Edition ed., Raleigh, NC 27615: SciTech Publishing, Inc., 1998.

BP Neural Network Development and Testing

The material in this section documents the work conducted to design and verify the multi-layer artificial neural network used in this project. The theory used to design this network is detailed in Chapter 3.

The advantage of using artificial neural networks is their ability to solve non-linear problems. Therefore, the first test to determine if an ANN is functioning properly is to attempt to solve a known non-linear problem with one. The common test for non-linear problem solving is that of the exclusive-OR function (XOR). While theoretically, only one hidden layer is required to solve the XOR problem, two hidden layers were included in this simulation. The program was set to run for ten-thousand (10,000) iterations at a learning rate of 0.1. It took 14.24 seconds to train the network. Execution time of the trained weights was on the order of milliseconds.

As a debugging tool, and proof of concept, samples were taken of the interconnecting weights at each layer in order to ensure that they were in fact converging. This can be seen in Figure 0-2. As such, the absolute error of the network, seen in Figure 0-1 was recorded. As a neural network learns, the error should eventually reach zero, if enough iterations are completed. For any network to have the error not converge to zero is likely to be indicative of an unstable scenario. Such a scenario can be caused by conflicting training data. For example, if for the XOR problem, there were two outputs associated with the binary input of [1,1], then the network would never properly converge as the network would continually (and randomly) associate [1,1] to one output , then at some random training iteration later, this association would switch to be associated with the other output.

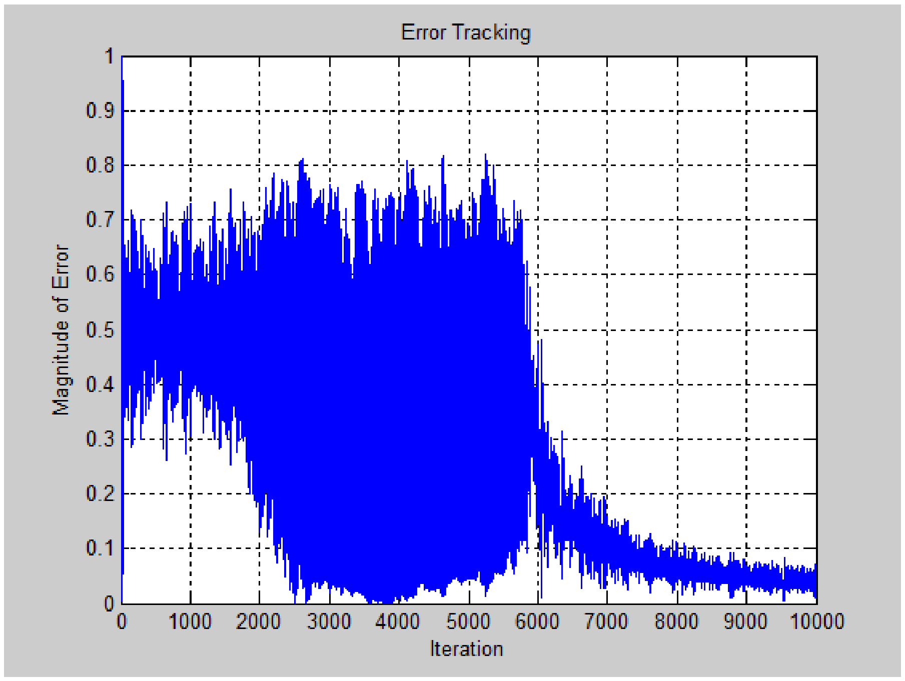

Figure 0-1: XOR Test Error Tracking

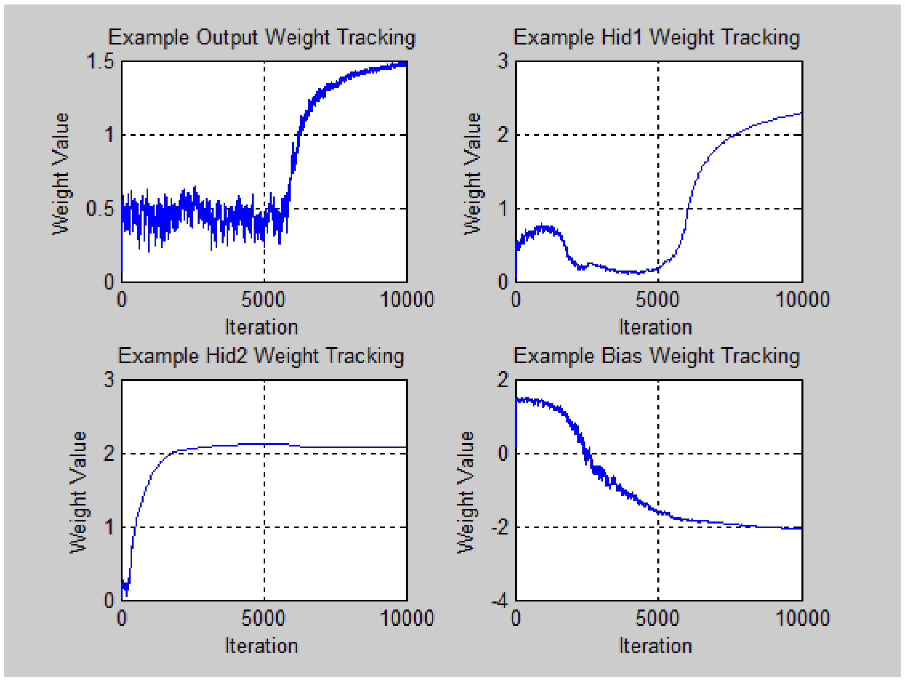

Figure 0-2: XOR Test Tracking Sample of Weight Values

As it can be clearly seen, the algorithm developed for this thesis does in fact pass the XOR test and does in fact converge to a correct solution.

Console output:

```
What learning rate? ( < 1 ) : 0.1
How many learning iterations? : 10000
--------------
Input Data:

ans =

     0
     0


ans =

     0
     1


ans =

     1
     0


ans =

     1
     1

--------------
---------------
---------------
Network Result:

netOut(:,:,1) =

    0.0244


netOut(:,:,2) =

    0.9739


netOut(:,:,3) =

    0.9722


netOut(:,:,4) =

    0.0545

---------------
```

The next experiment that was conducted was to determine if the network could be trained to recognize basic geometric shapes.

Hopfield Network Development and Testing

The material in this section documents the work conducted to design and verify the Hopfield Model of an artificial neural network used in this project. The theory used to design this network is detailed in Chapter 3.

With a very limited data set, the following data was produced. Because of memory issues in MATLAB, all images were scaled to $50 \times 50\ px$ before running them through the Hopfield network.

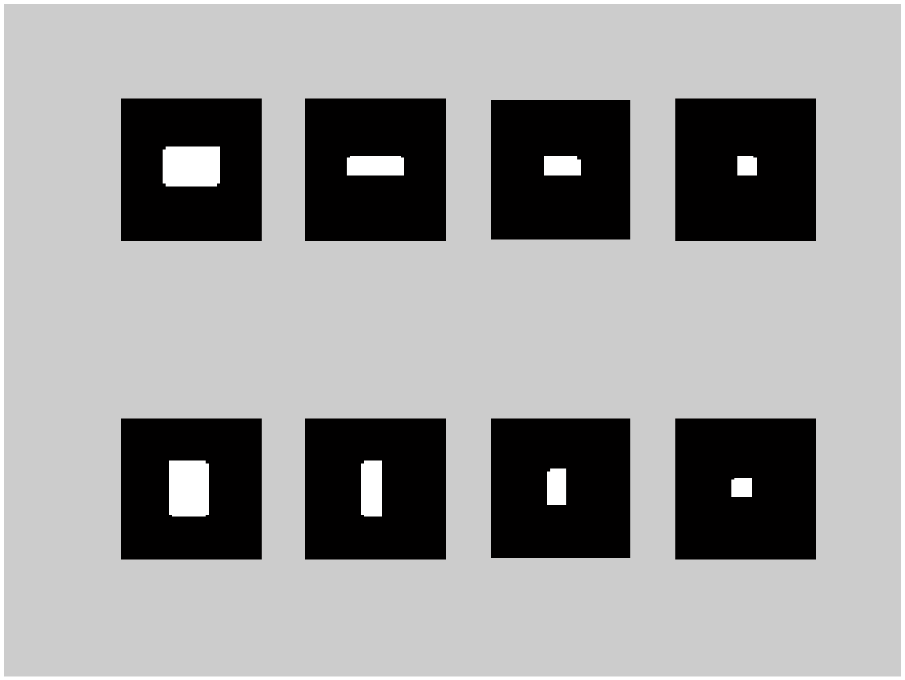

Figure 0-1: Resized Training Images

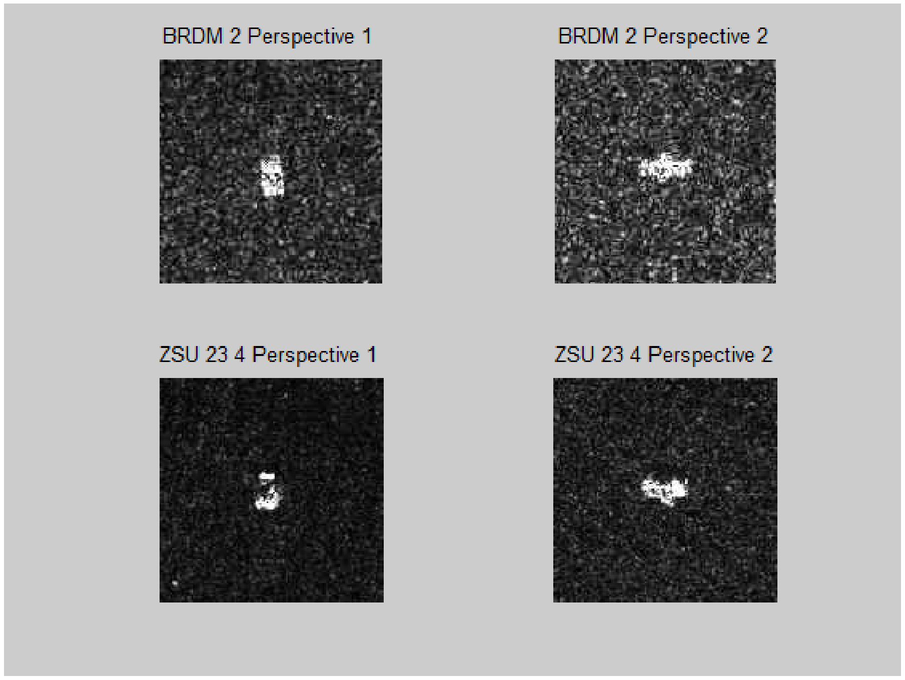

Figure 0-2: Input MSTAR Images to analyze

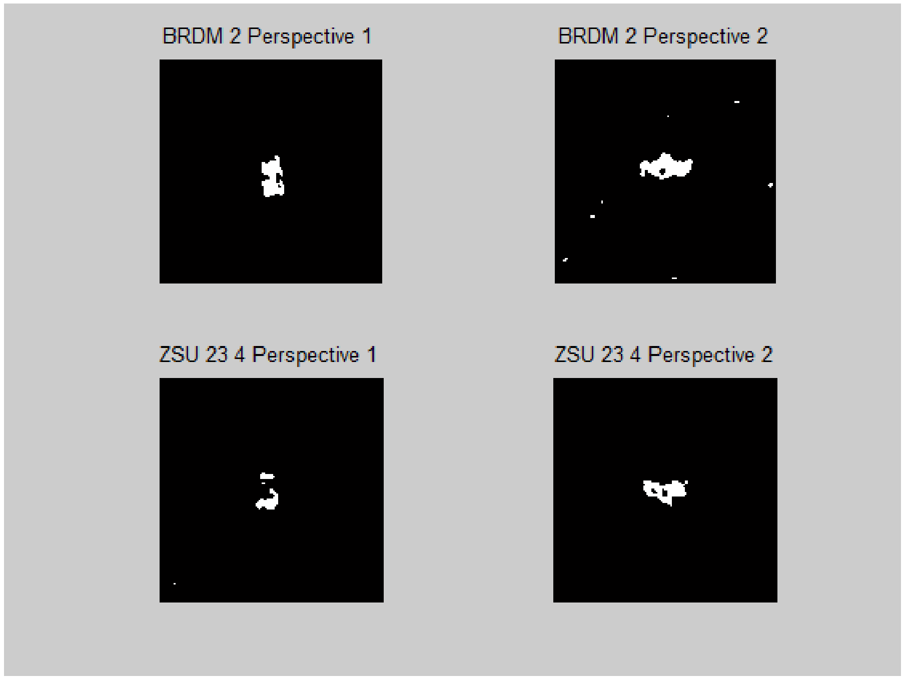

Figure 0-3: Input MSTAR Images after Thresholding

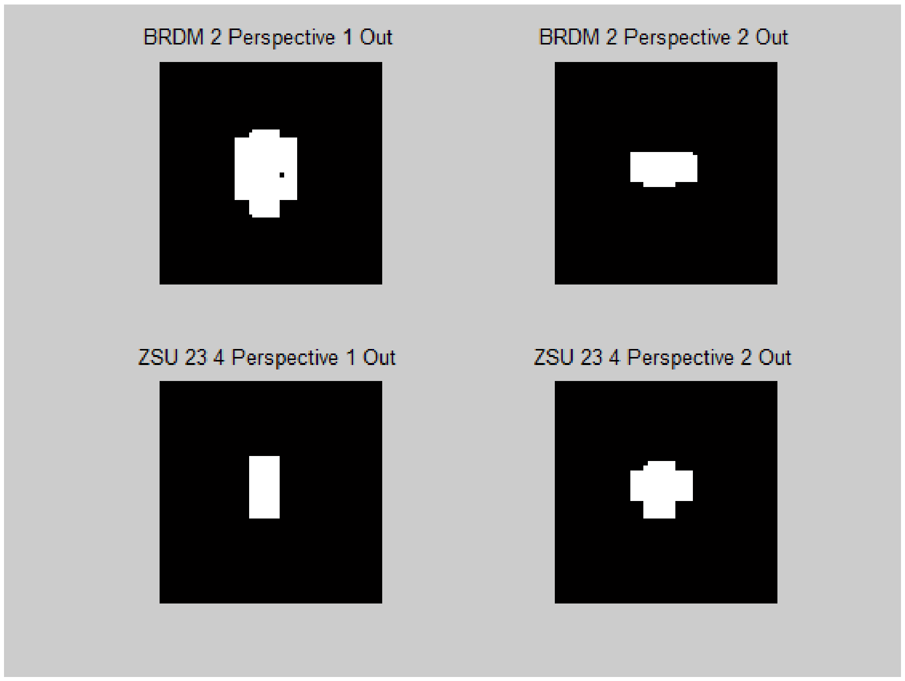

Figure 0-4: Output of Hopfield Network Trained with small set.

While it is apparent that the program is at least 50% capable of determining the correct orientation of the target, however the orthogonal fundamental memories cause conflicts with determining the approximate rectangular fit. The network would often output a cross of the similar, but orthogonal, training data.

It was then determined that the best course of action would be to only implement the rectangular fitting algorithm on images all facing the same direction. In this case, horizontally aligned targets.

For the rotational algorithm, as was mentioned in the main body of this thesis, worked better with the disjoint collinear rectangular training data, than it did with the full rectangles. To show why this was done, an example output is shown in Figure 0-5. As it can be seen that the network trained with the full rectangles tends to stabilize to a circular shape, instead of the rectangle at the proper angle.

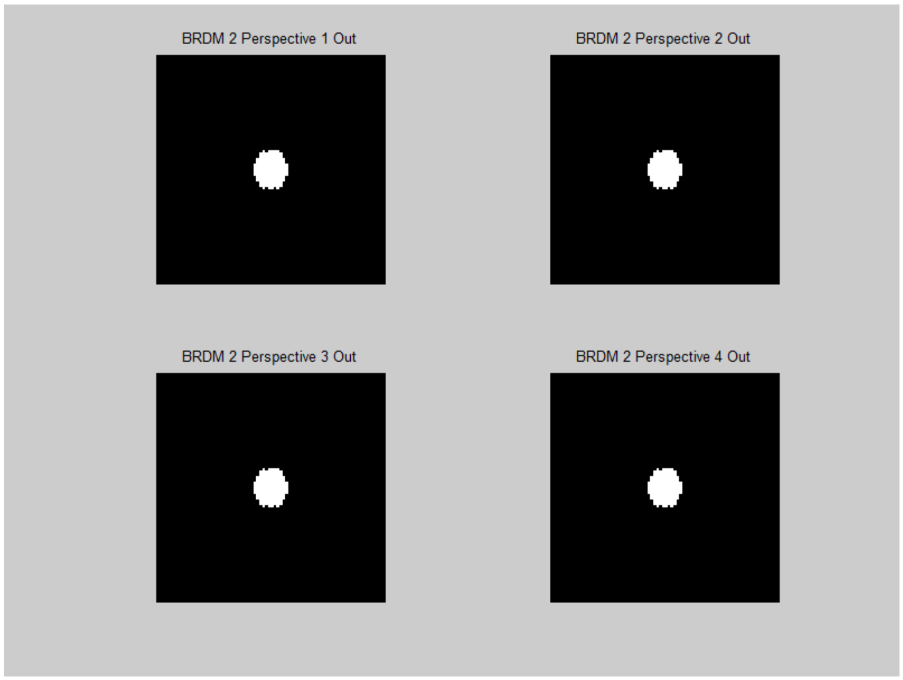

Figure 0-5: Output of Automatic Rotation Detector Using Full Rectangle Training Set

Edge Detector Training Data Sets

The data used for the ANN learning process is documented in this section. Due to the fact that some of the training sets were generated with random noise, the images provided here are only examples of what was used and may not be the exact images used.

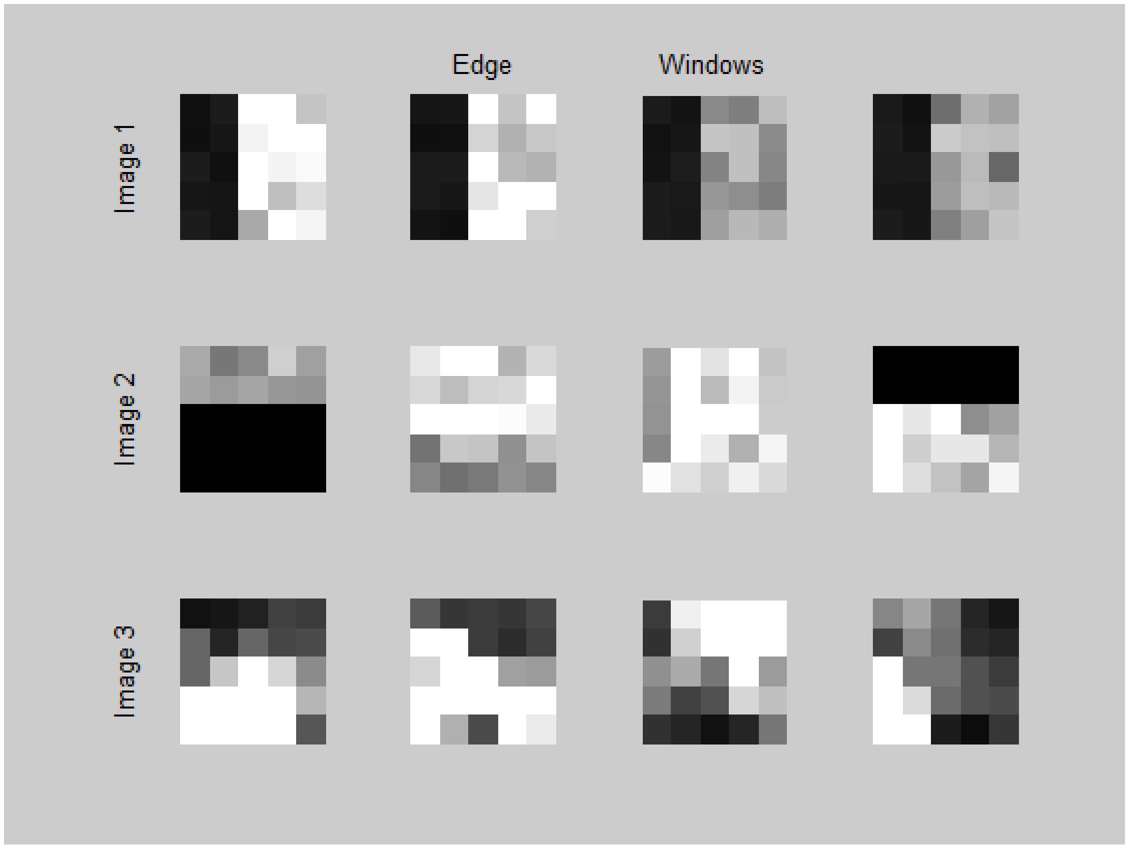

Figure 0-1: "Small" Learning Data Set of 5x5 Windows of Edge Regions

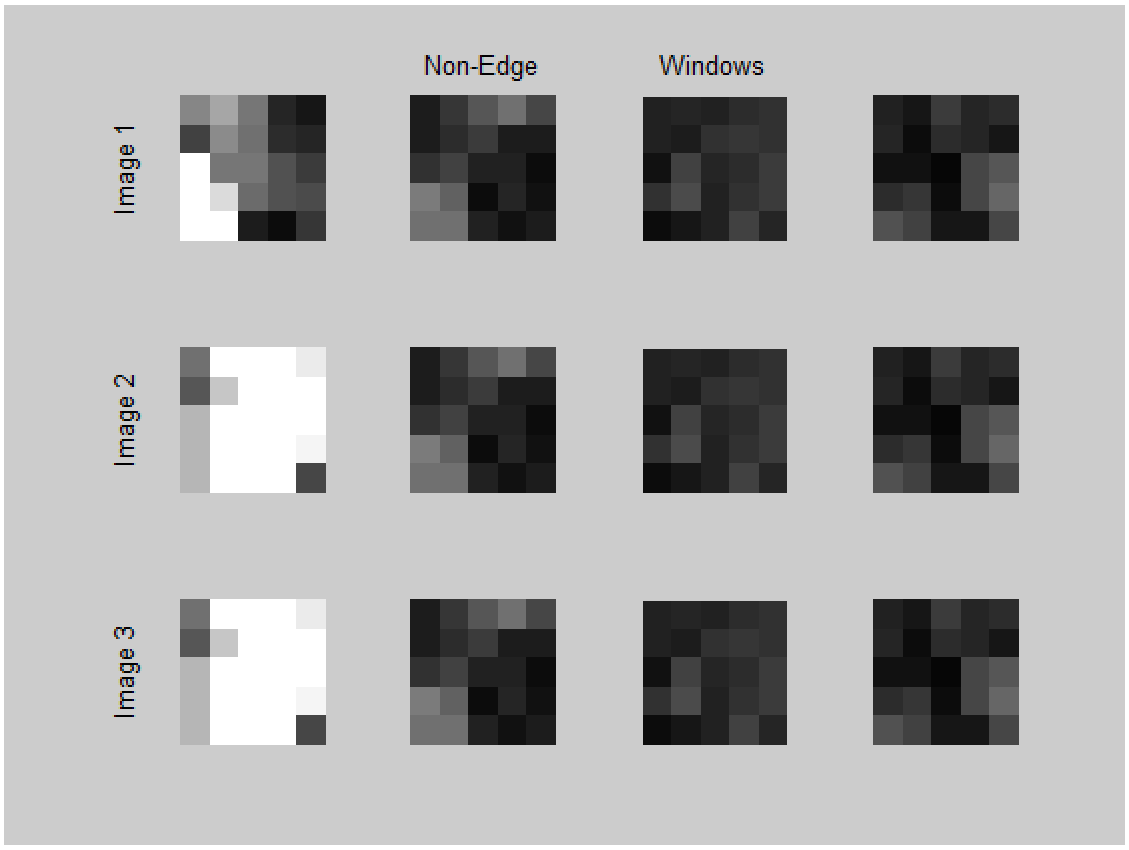

Figure 0-2: "Small" Learning Data Set of 5x5 Windows of Homogeneous Regions

Discrete Hopfield Network Model Training Data

This section contains the images used for both the DHN rectangular fit program as well as the angular measurement program. All of this data was generated manually using Adobe® Photoshop® and then processed in MATLAB.

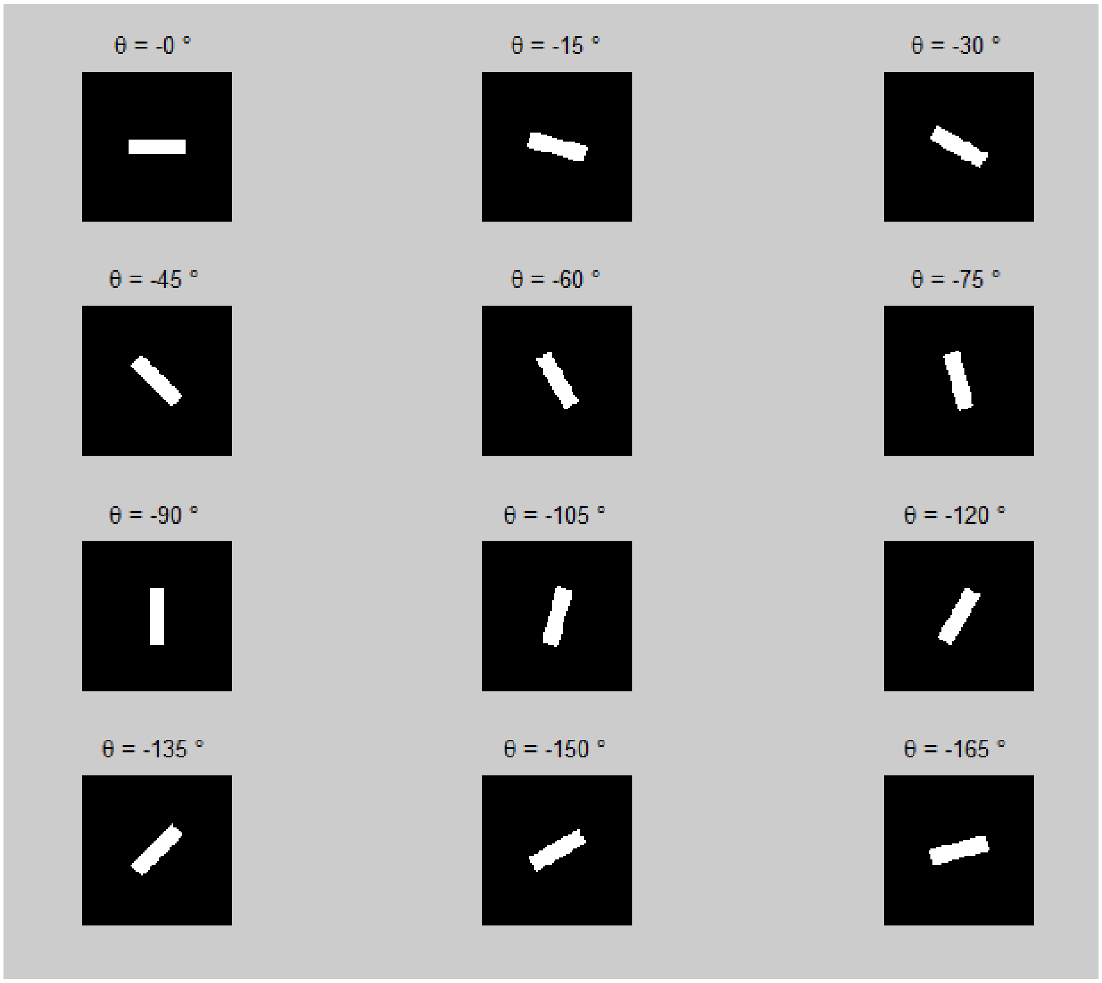

Figure 0-1: Solid Bar Rotation Detector Training Data

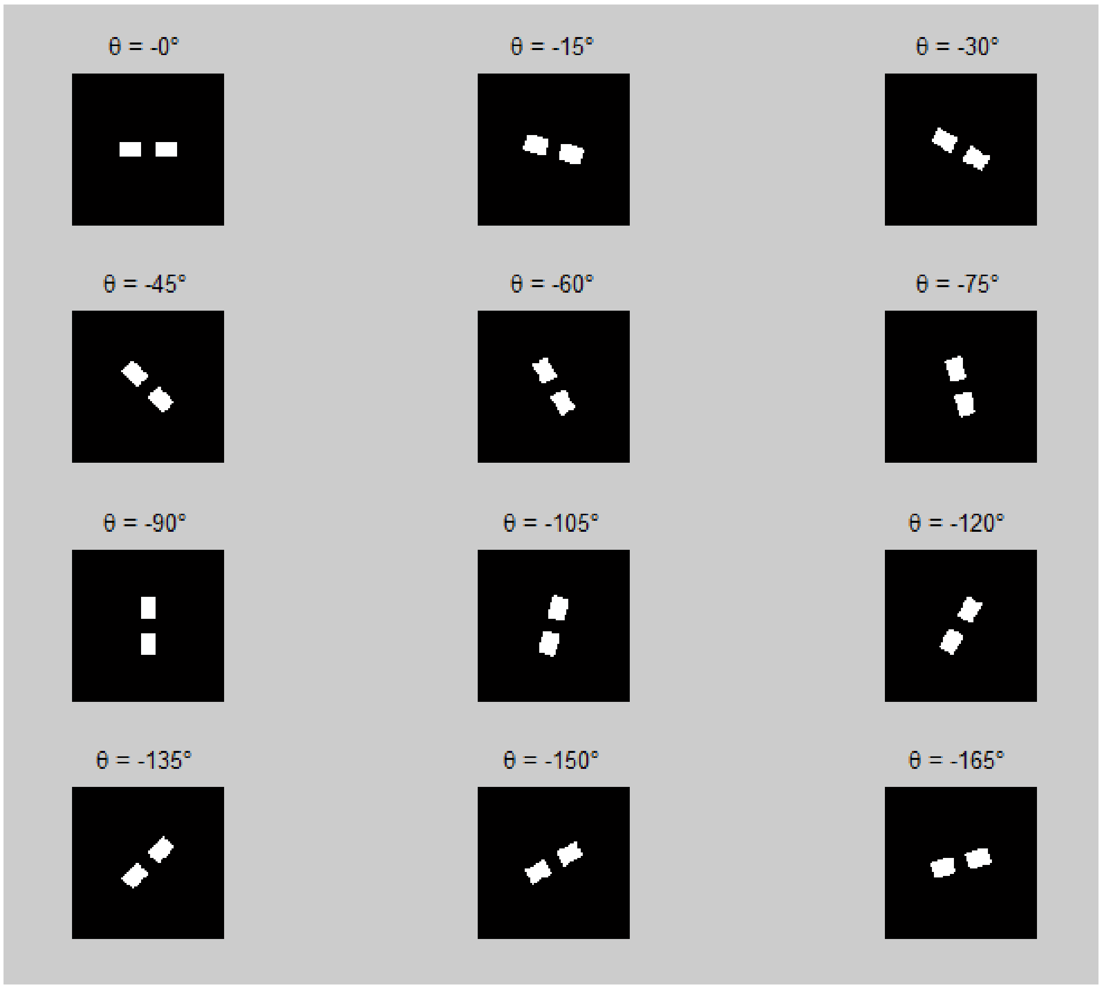

Figure 0-2: Split Bar Rotation Detector Training Data

Extended ANN Edge Detection Outputs

This section contains additional sample outputs of the Edge Detection algorithm. Each collection of images includes the final detected edge and a comparison to a Canny edge detector followed by the four composite rotated images from the rotational edge detector algorithm.

BRDM_2 45 Deg File: HB15696.JPG

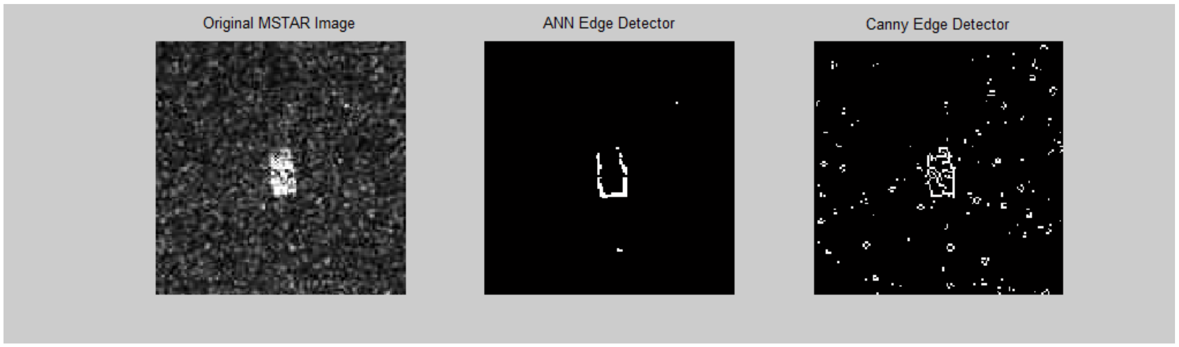

Figure 0-1: BRDM-2 45 Degree Depression Angle Composite and Comparison

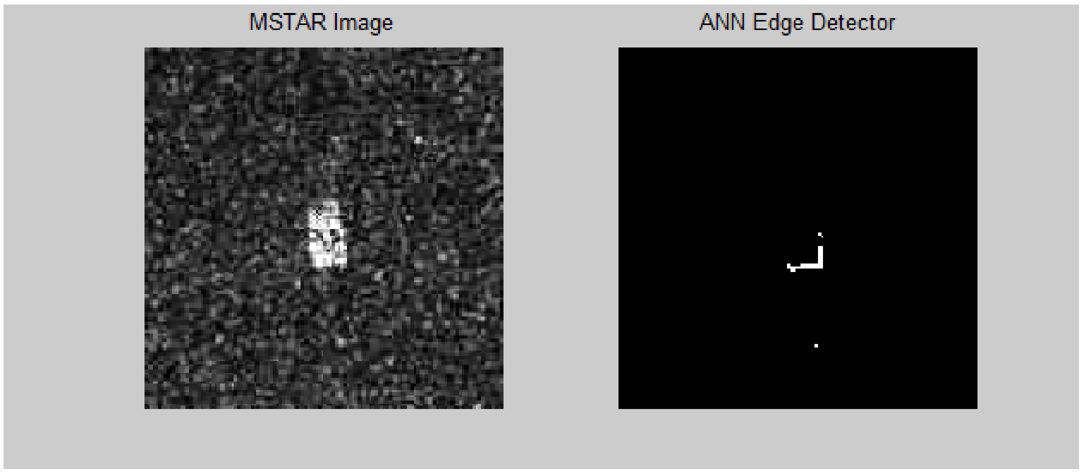

Figure 0-2: BRDM-2 45 Degree Depression Component Image 0 degree rotation.

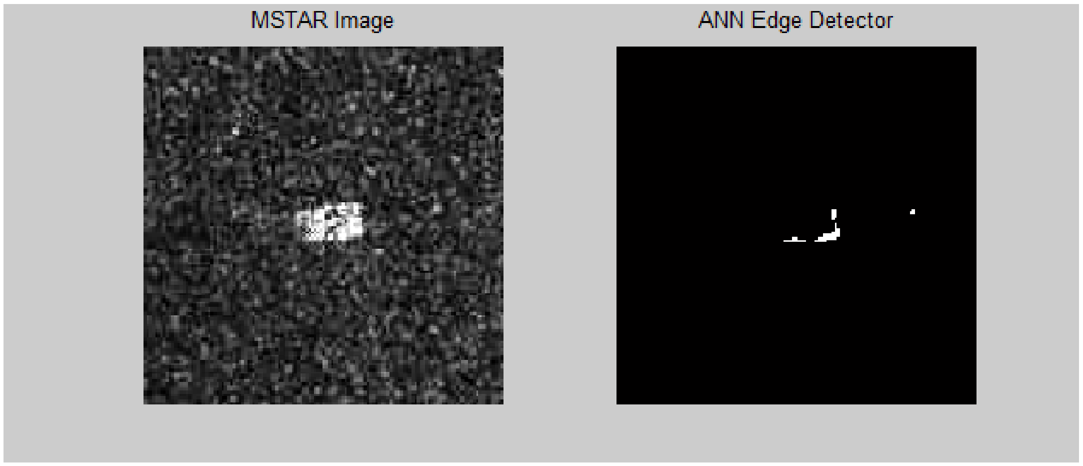

Figure 0-3: BRDM-2 45 Degree Depression Component Image 90 degree rotation.

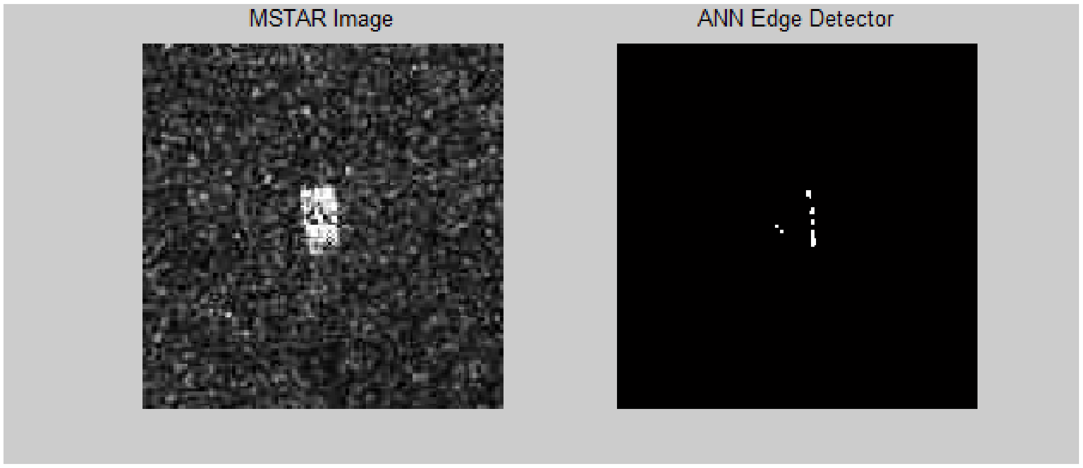

Figure 0-4: BRDM-2 45 Degree Depression Component Image 180 degree rotation.

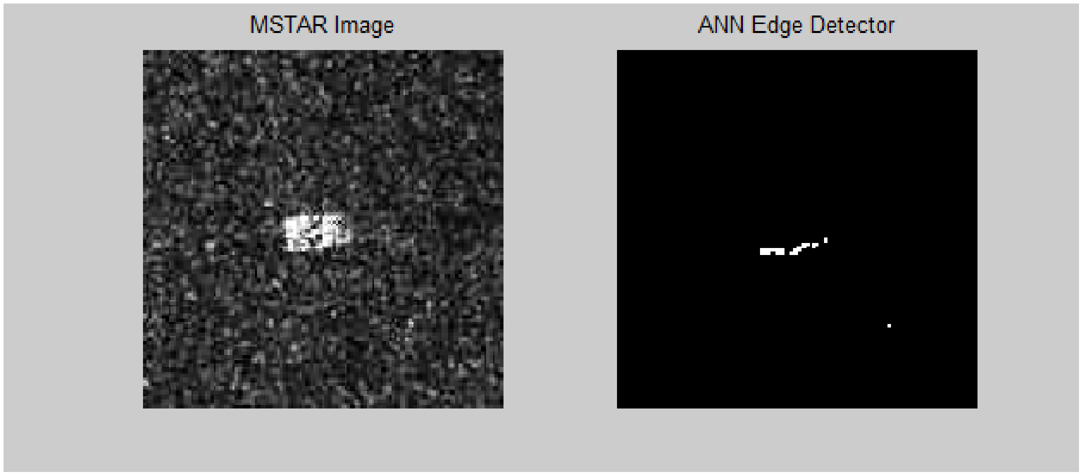

Figure 0-5: BRDM-2 45 Degree Depression Component Image 270 degree rotation.

ZSU-23-4 45 Deg File: HB16449.JPG

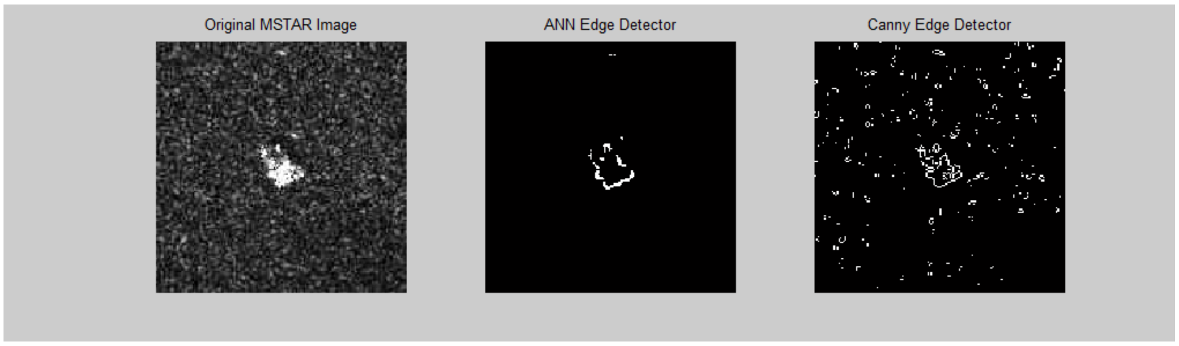

Figure 0-6: ZSU-23-4 45 Degree Depression Angle Composite and Comparison

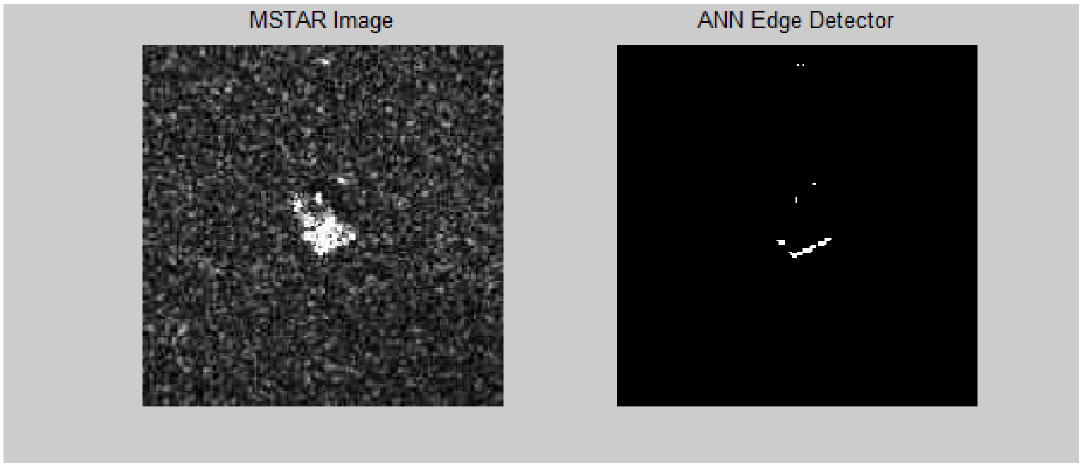

Figure 0-7: ZSU-23-4 45 Degree Depression Angle Component Image 0 degree rotation.

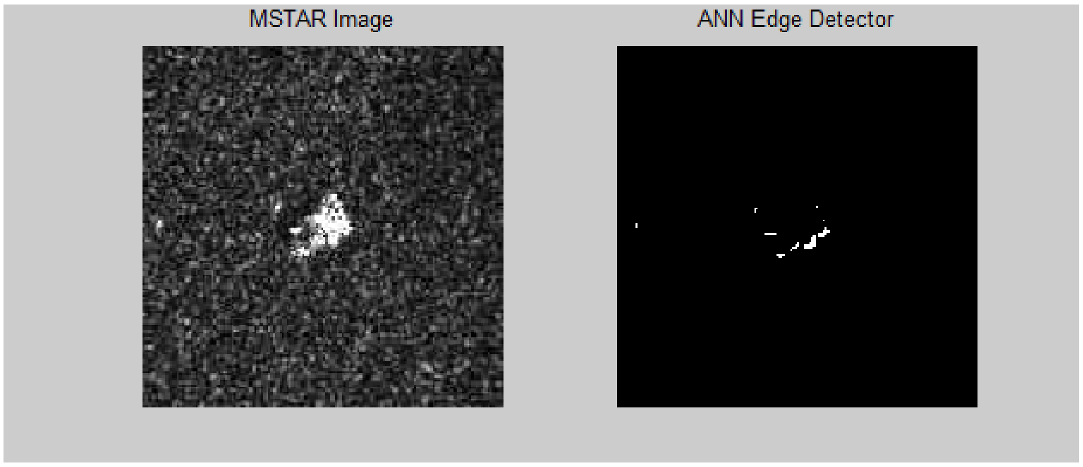

Figure 0-8: ZSU-23-4 45 Degree Depression Angle Component Image 90 degree rotation.

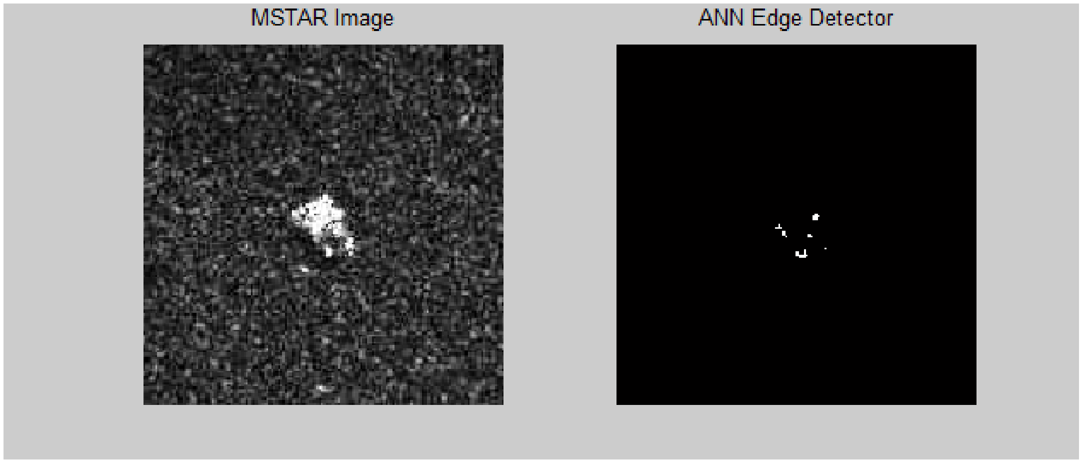

Figure 0-9: ZSU-23-4 45 Degree Depression Angle Component Image 180 degree rotation.

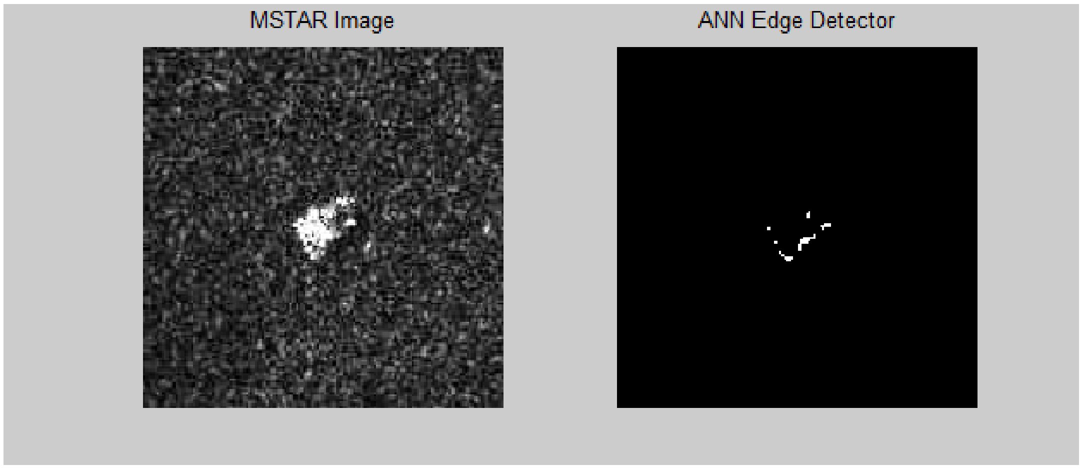

Figure 0-10: ZSU-23-4 45 Degree Depression Angle Component Image 270 degree rotation.

2S1 30 Deg File: HB15375.JPG

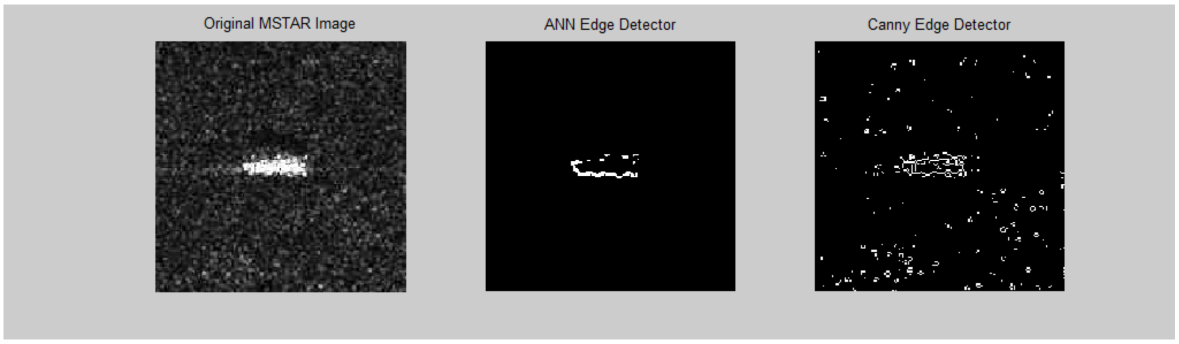

Figure 0-11: 2S1 30 Degree Depression Angle Composite and Comparison

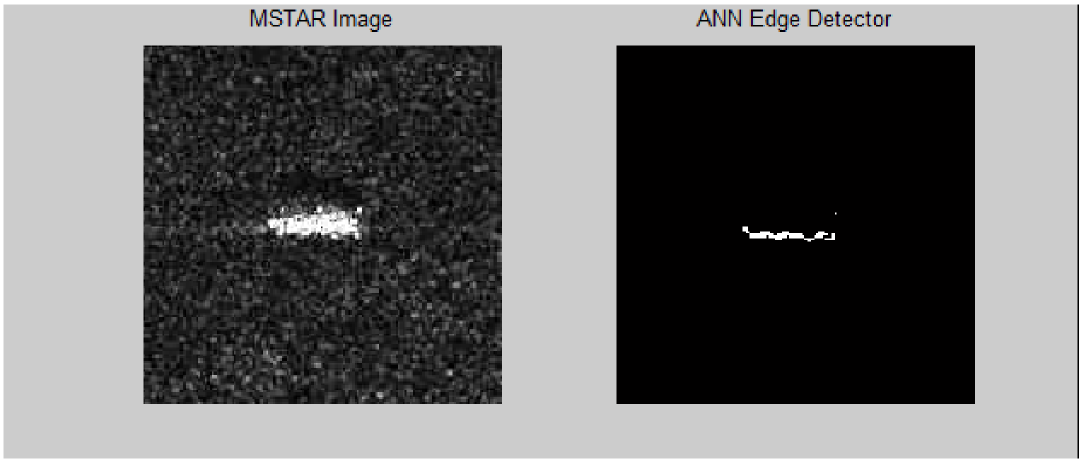

Figure 0-12: 2S1 30 Degree Depression Angle Component Image 0 degree rotation.

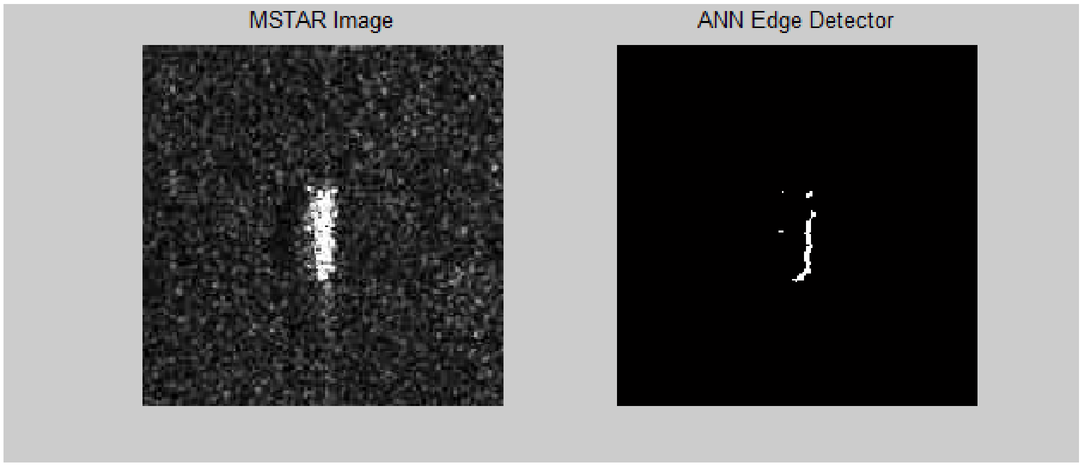

Figure 0-13: 2S1 30 Degree Depression Angle Component Image 90 degree rotation.

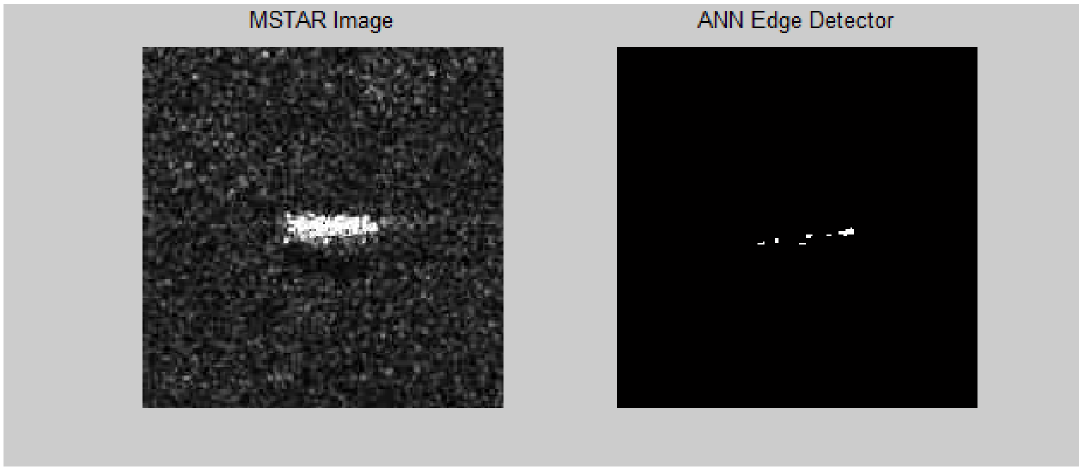

Figure 0-14: 2S1 30 Degree Depression Angle Component Image 180 degree rotation.

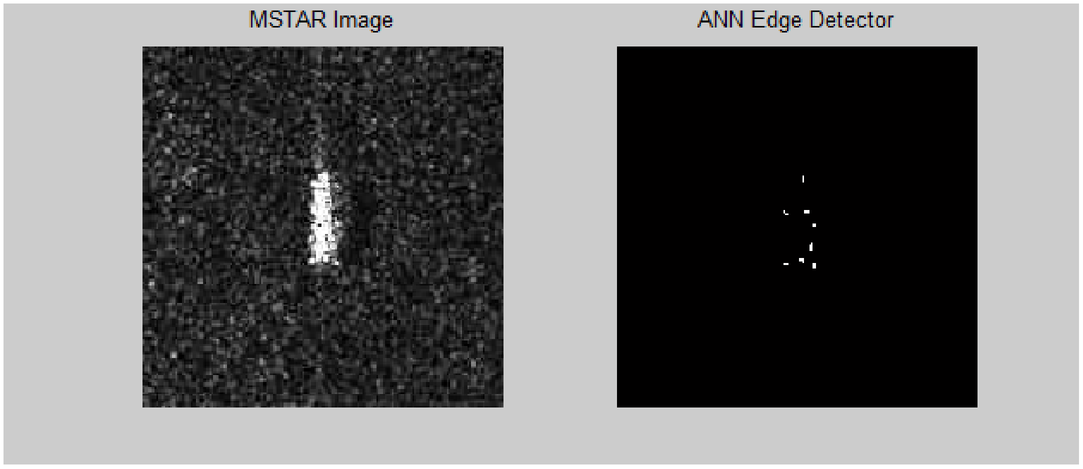

Figure 0-15: 2S1 30 Degree Depression Angle Component Image 270 degree rotation.

SLICY 15 Deg File: HB15350.JPG

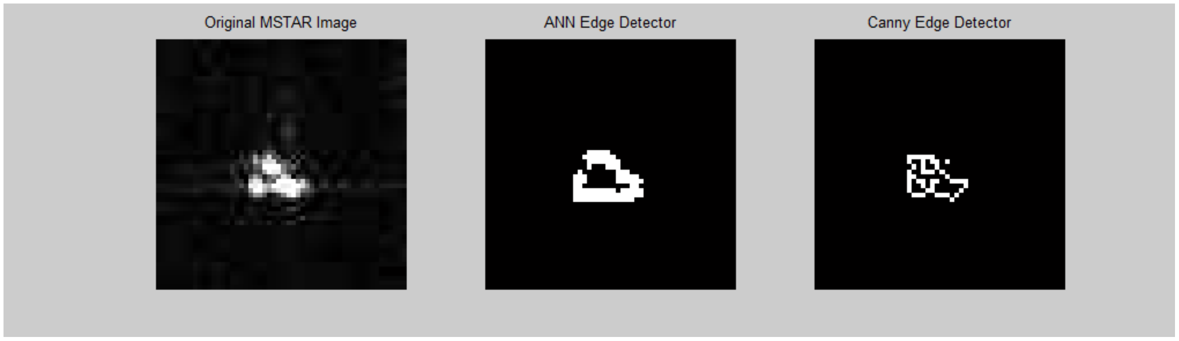

Figure 0-16: 2S1 30 Degree Depression Angle Composite and Comparison

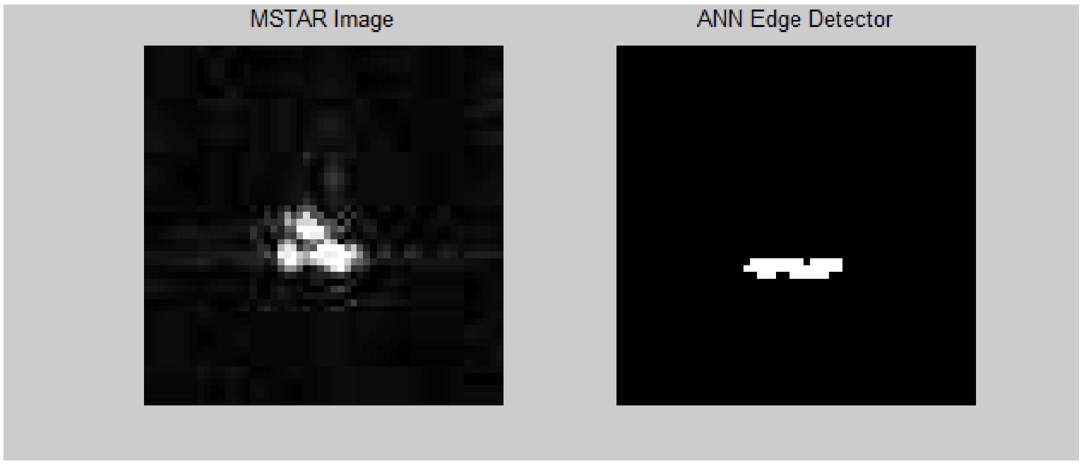

Figure 0-17: SLICY 15 Degree Depression Angle Component Image 0 degree rotation.

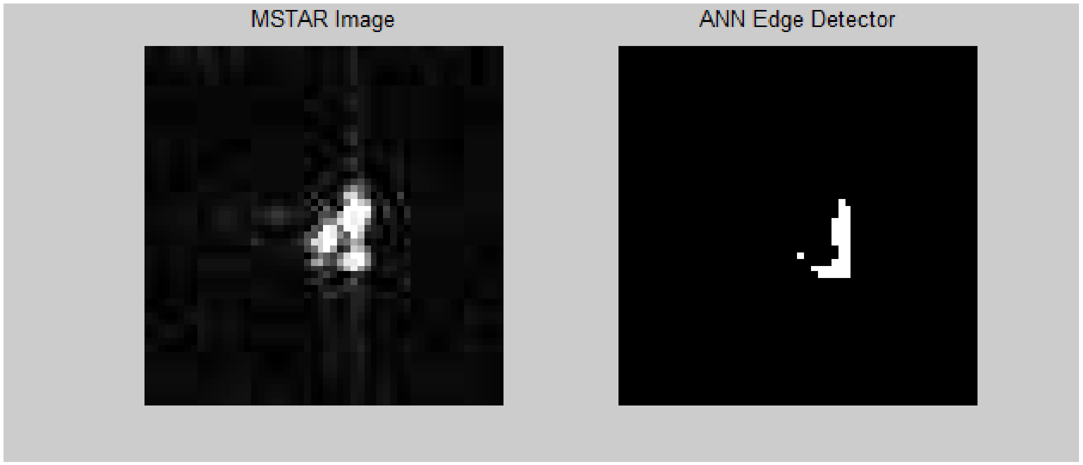

Figure 0-18: SLICY 15 Degree Depression Angle Component Image 90 degree rotation.

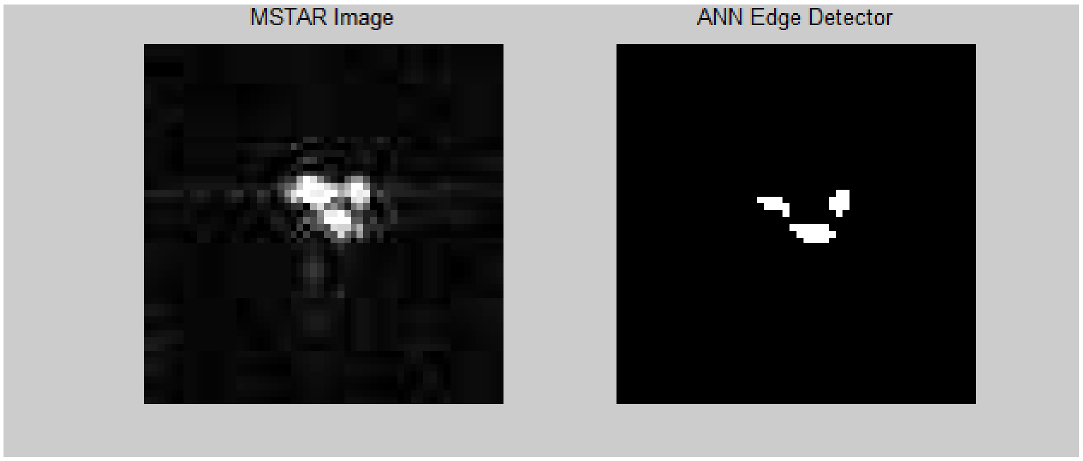

Figure 0-19: SLICY 15 Degree Depression Angle Component Image 180 degree rotation.

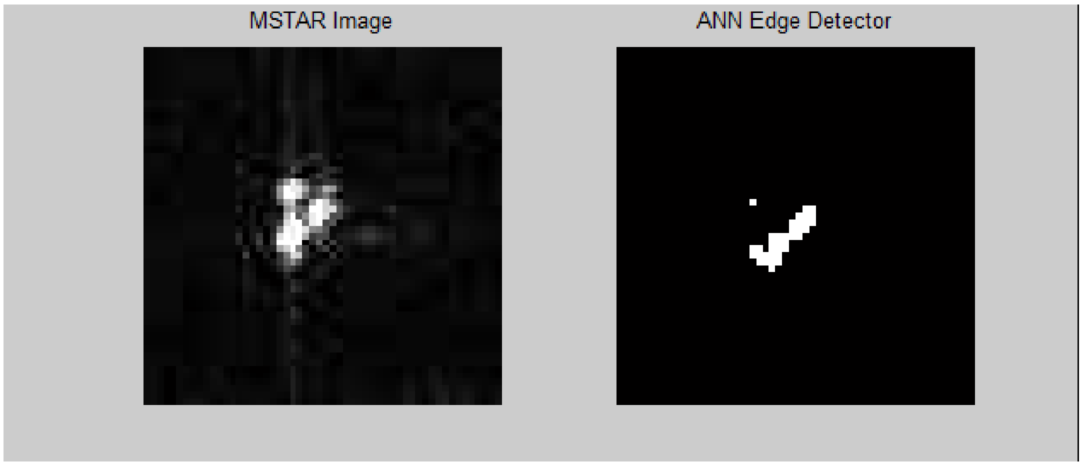

Figure 0-20: SLICY 15 Degree Depression Angle Component Image 270 degree rotation.

D7 15 Deg File HB14939.JPG

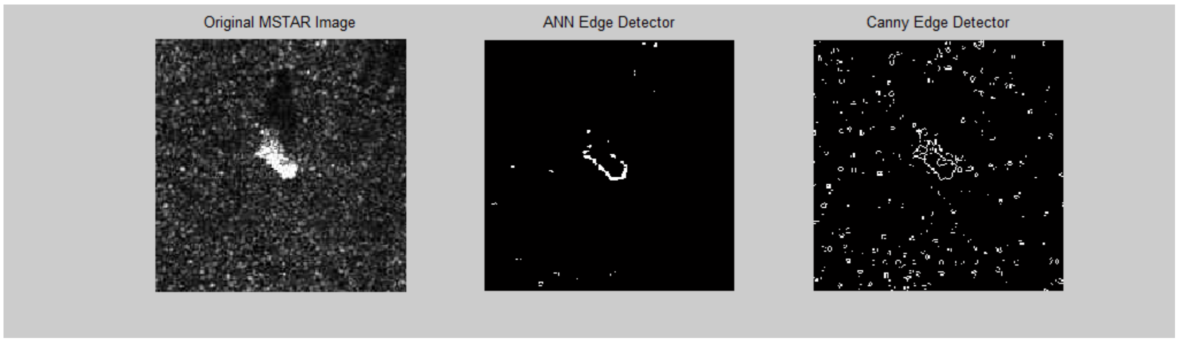

Figure 0-21: D7 15 Degree Depression Angle Composite and Comparison

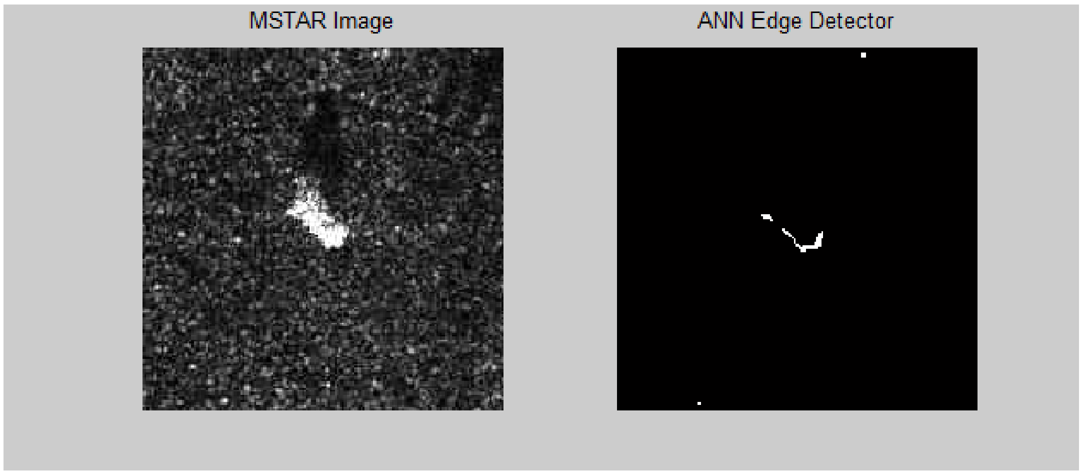

Figure 0-22: D7 15 Degree Depression Angle Component Image 0 degree rotation.

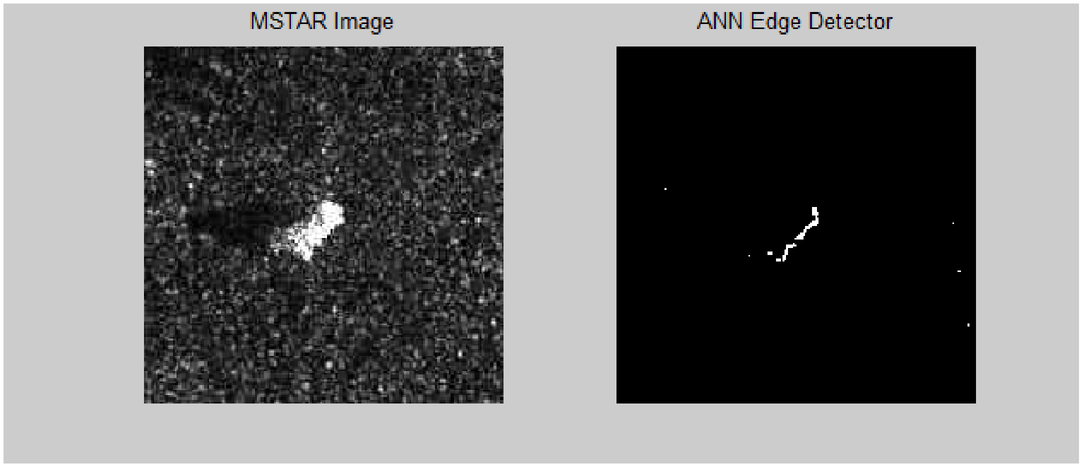

Figure 0-23: D7 15 Degree Depression Angle Component Image 90 degree rotation.

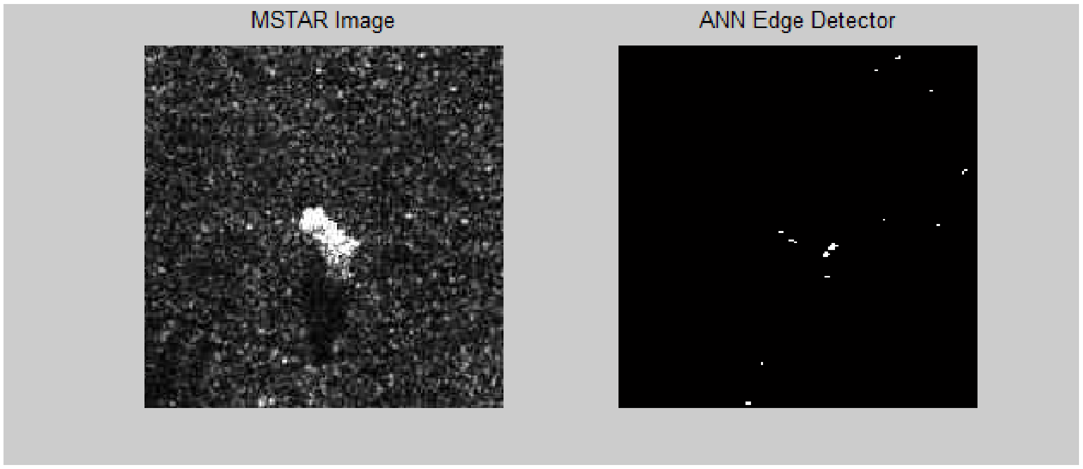

Figure 0-24: D7 15 Degree Depression Angle Component Image 180 degree rotation.

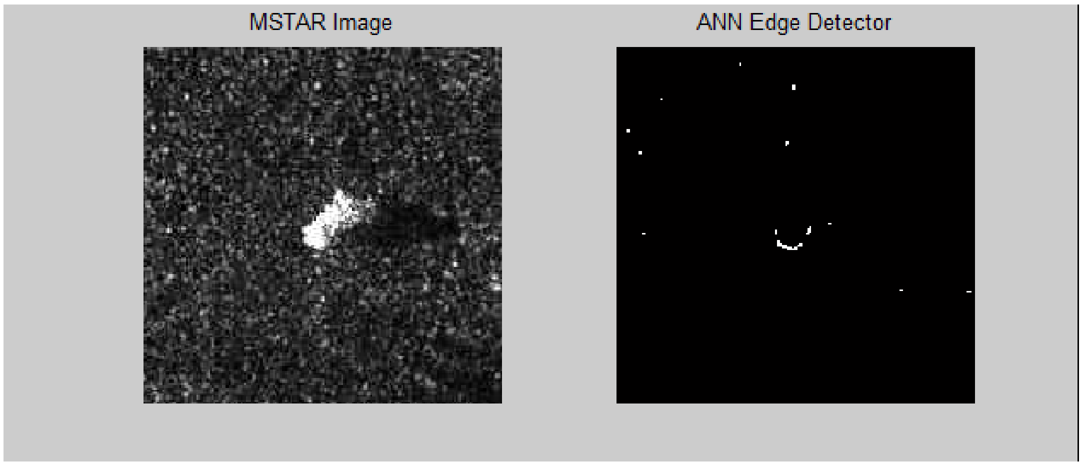

Figure 0-25: D7 15 Degree Depression Angle Component Image 270 degree rotation.

T62 15 Deg File: HB14932.JPG

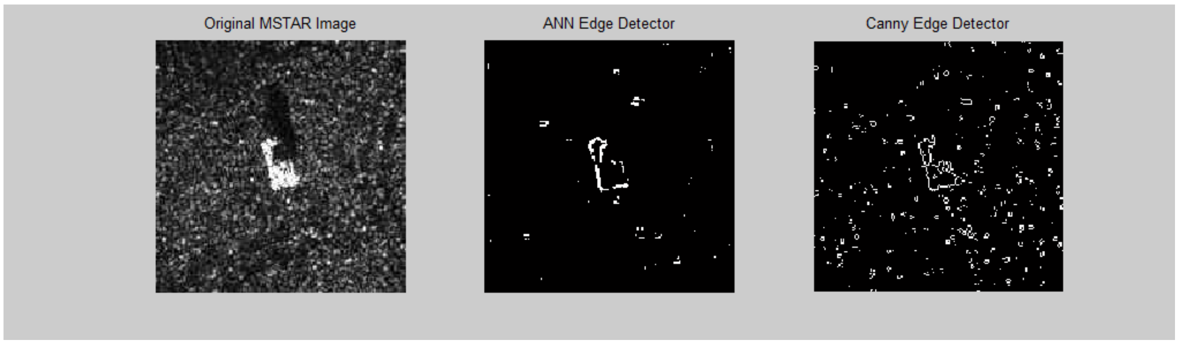

Figure 0-26: T62 15 Degree Depression Angle Composite and Comparison

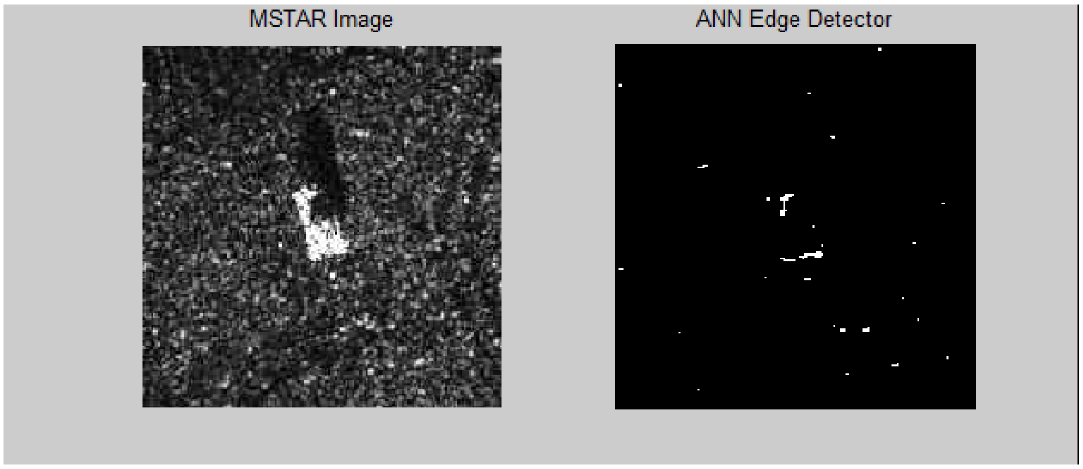

Figure 0-27: T62 15 Degree Depression Angle Component Image 0 degree rotation.

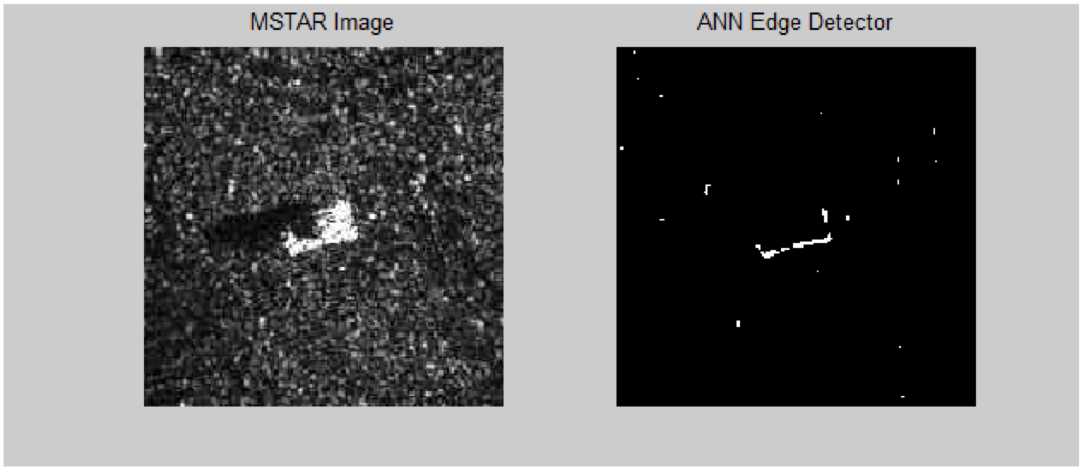

Figure 0-28: T62 15 Degree Depression Angle Component Image 90 degree rotation.

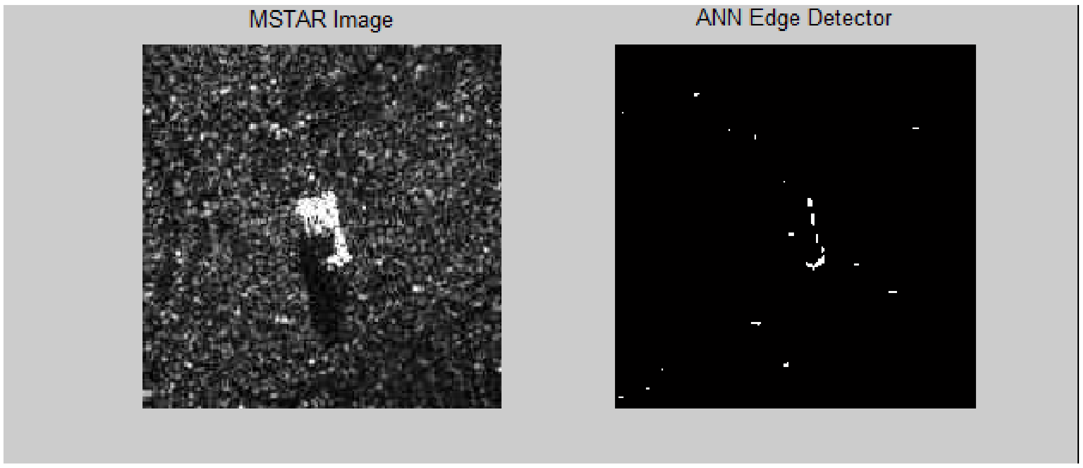

Figure 0-29: T62 15 Degree Depression Angle Component Image 180 degree rotation.

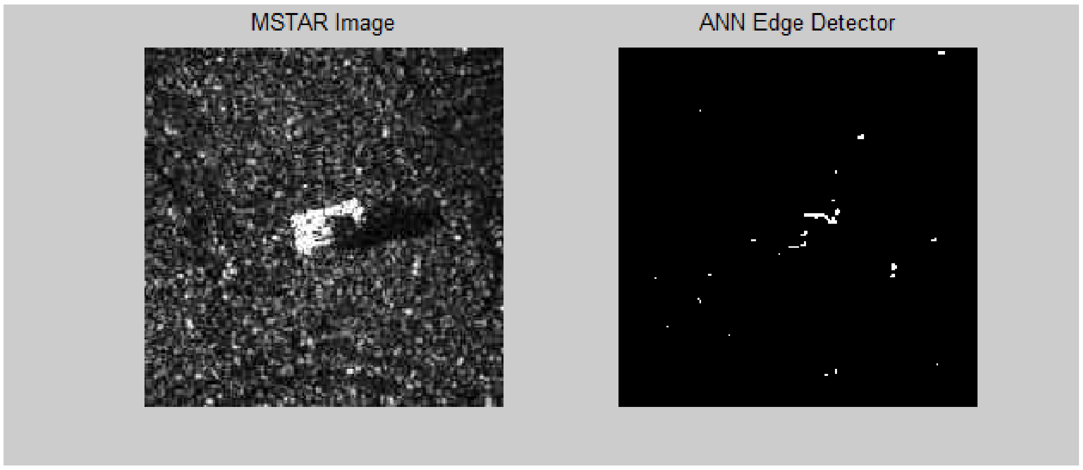

Figure 0-30: T62 15 Degree Depression Angle Component Image 270 degree rotation.

ZIL 131 15 Deg File: HB14944.JPG

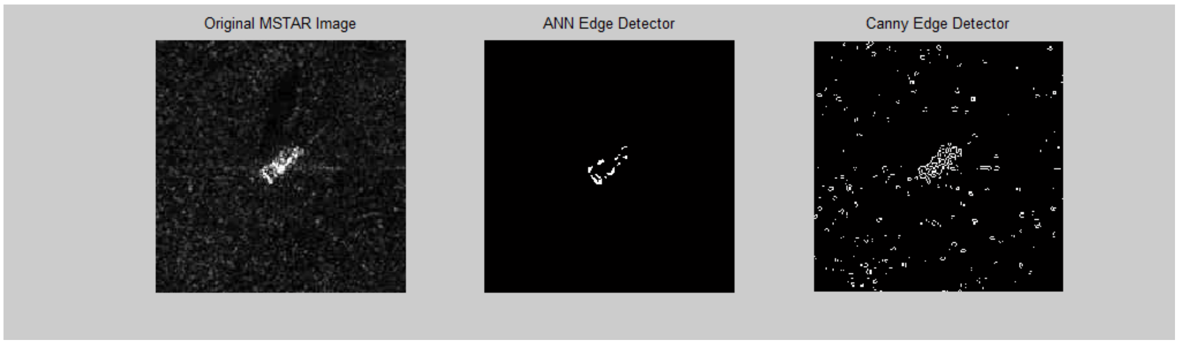

Figure 0-31: ZIL 131 15 Degree Depression Angle Composite and Comparison

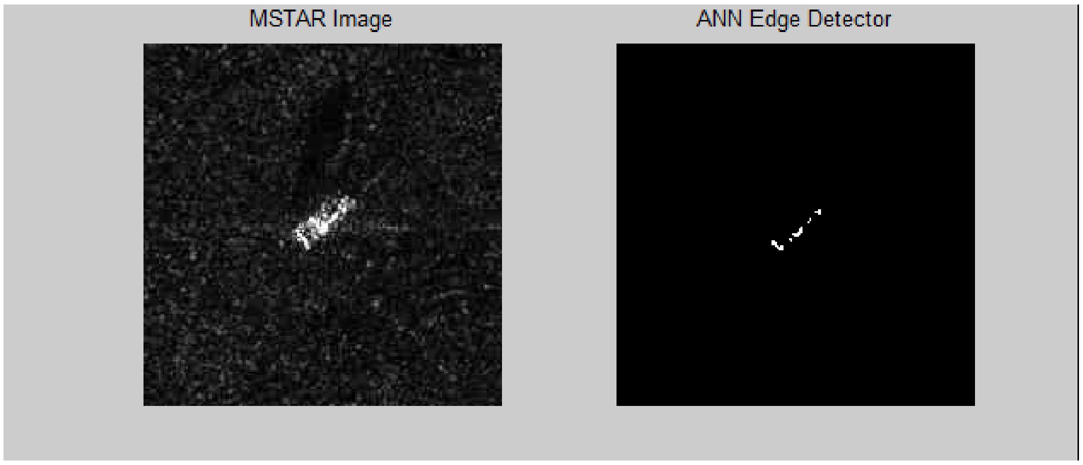

Figure 0-32: ZIL 131 15 Degree Depression Angle Component Image 0 degree rotation.

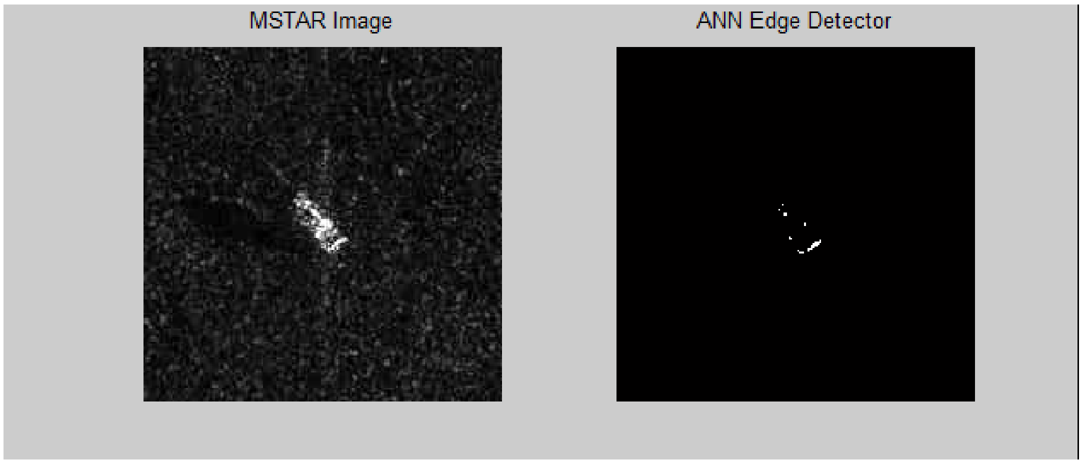

Figure 0-33: ZIL 131 15 Degree Depression Angle Component Image 90 degree rotation.

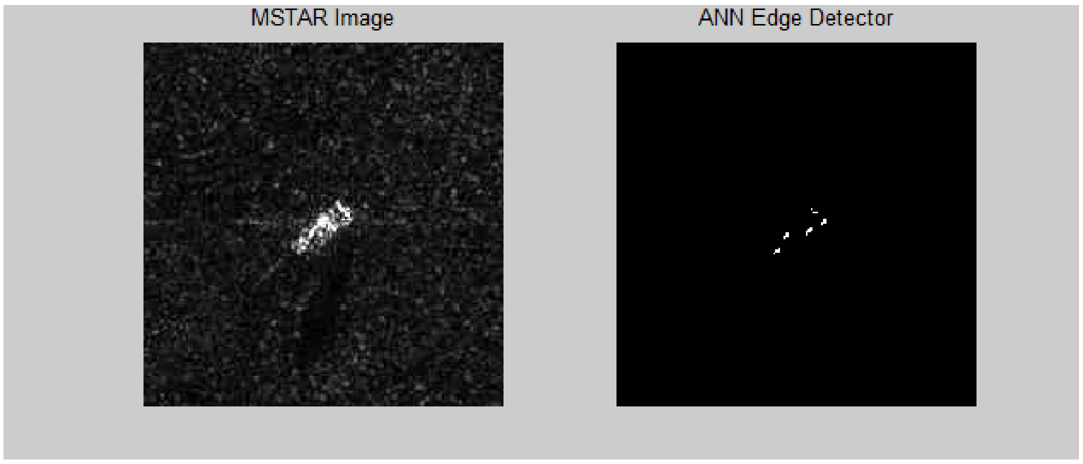

Figure 0-34: ZIL 131 15 Degree Depression Angle Component Image 180 degree rotation.

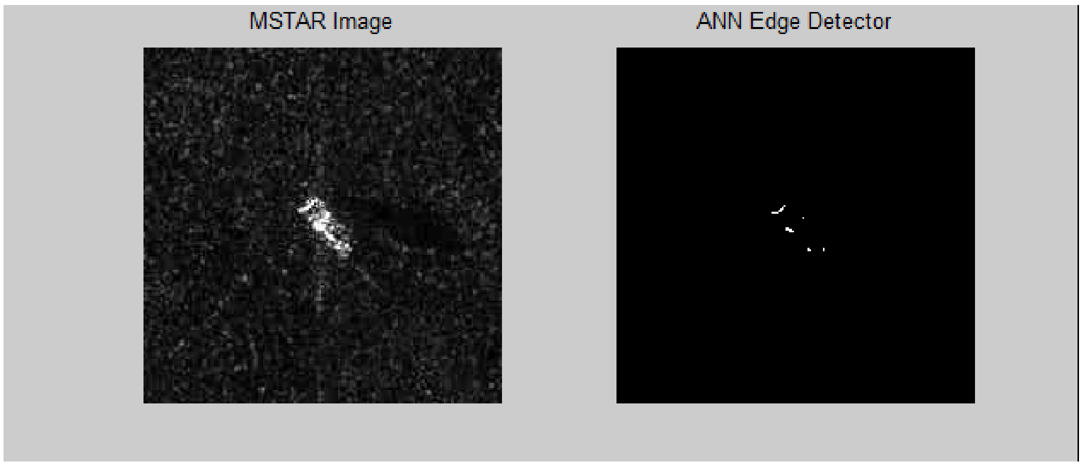

Figure 0-35: ZIL 131 15 Degree Depression Angle Component Image 270 degree rotation.

MATLAB Code

File: BP_FF.m

```
function [layerOut] = BP_FF(weights,inVals,neuronCount)
%
% [layerOut] = BP_FF(weights,inVals,outSz)
%
% weights should be n-by-m, where n = neuronCount and m =
length(inVals)
% inVals should be 1-by-m, where m = length(weights) (column Vector!)
% neuronCount should be the number of neurons in this layer
%
% Back Propagation Algorithm Feed Forward Process
%


% % % % % % TROUBLESHOOTING % % % % % %
% disp('=================Troubleshooting Data=================')
% weights
% inVals
% neuronCount
%
% size(weights)
% size(inVals)
% size(neuronCount)
%
% disp('=============END Troubleshooting Data END=============')
% % % % % END TROUBLESHOOTING % % % % %

wSz = size(weights);
layerOut = zeros(neuronCount,1);

for n = 1:neuronCount
    tempW = weights(n,:);
    tempO = tempW*inVals;

    layerOut(n) = tanh(tempO);

end

% size(layerOut)

end
```

File: BP_FF_BIAS.m

```
function [layerOut] =
BP_FF_BIAS(weights,biasWeights,biasVal,inVals,neuronCount)
%
% [layerOut] =
BP_FF_BIAS(weights,biasWeights,biasVal,inVals,neuronCount)
%
% weights should be n-by-m, where n = neuronCount and m =
length(inVals)
% inVals should be 1-by-m, where m = length(weights) (column Vector!)
% neuronCount should be the number of neurons in this layer
%
% Back Propagation Algorithm Feed Forward Process
%



%
% disp('=================Troubleshooting Data=================')
% weights
% inVals
% biasWeights
% neuronCount
% biasVal
%
% size(weights)
% size(inVals)
% size(neuronCount)
% size(biasWeights)
%
% disp('=============END Troubleshooting Data END=============')


wSz = size(weights);
layerOut = zeros(neuronCount,1);

for n = 1:neuronCount
    tempW  = weights(n,:);
    tempBW = biasWeights(n,:);
    tempO = tempW*inVals + tempBW*biasVal;

    layerOut(n) = tanh(tempO);

end

% size(layerOut)

end
```

File: BP_OutUpdate.m

```
function [newWeights, LM] = BP_OutUpdate(oldjWeights, err, learnRate,
iOuts)
%
% Output Layer Weight Updater
% Updates a single neuron at a time (all connected weights)
%
% oldjWeights should be a row vector of the connected weights to the
current
% neuron. Each column corresponds to the neuron numbers.
%
% err is the error from expected value
%
% learnRate is some scalar number for how fast learning should
progress.
%
% iOuts is the column vector of all outputs from layer i. Each row in
the
% vector is a different neuron.
%
% ineuronNum is the index number of the neuron within the layer to be
% updated. This should correspond directly to the iOuts and oldjWeights
vector.
%

% disp('=================Troubleshooting Data=================')
%
% oldjWeights
% err
% learnRate
% iOuts
%
% % oldjWeights*iOuts
%
%
% size(oldjWeights)
% size(err)
% size(learnRate)
% size(iOuts)
%
%
% disp('=============END Troubleshooting Data END=============')

LM = err*(1/((cosh(oldjWeights*iOuts)^2)));
%
deltaW = learnRate*(LM)*iOuts;
%
% newWeights = oldjWeights + transpose(deltaW);



% deltaW = learnRate*(err*(1/(cosh(oldjWeights*iOuts)^2)))*iOuts;

newWeights = oldjWeights + transpose(deltaW);

end
```

File: BP_HidUpdate.m

```
% function [newjWeights, LMHP] = BP_HidUpdate(kWeights, oldjWeights,
err, learnRate, iOuts, jOuts, LMk)
function [newjWeights, LMHP] = BP_HidUpdate(kWeights, oldjWeights,
learnRate, iOuts, LMk)

%
%                                            : Input
Output
% Weight Updater for Hidden Layer 1 (Example: Layer N....3, 2, 1,
Output Layer)
%
% LMX is the "Last Minima", this is used if more layers are created
beyond
% one hidden layer.
%1

% disp('=================Troubleshooting Data=================')
%
% kWeights
% oldjWeights
% % err
% learnRate
% iOuts
% % jOuts
% % ineuronNum
% LMk
%
% disp('=============END Troubleshooting Data END=============')

% temp = size(oldjWeights)

% deltaW =
learnRate*((1/cosh(oldjWeights*iOuts)^2)*sum(err*(1/cosh(kWeights*jOuts
)^2)*kWeights))*iOuts;
% LMHP = sum(err*(1/cosh(kWeights*jOuts)^2)*kWeights);
% LMHP = sum(err*LMk*kWeights);

LMHP = sum(LMk*kWeights);

deltaW = learnRate*((1/cosh(oldjWeights*iOuts)^2)*LMHP)*iOuts;

newjWeights = oldjWeights + transpose(deltaW);



end
```

File: BP_TopScript_XOR_V3.m

```
% Back Prop Test XOR Script 1
%
%
%

close all
clear
clc

%% Network Config and Desired Function Input

ticTimer = tic; % Program Timer


% Network Configuration as defined by weights.
layerCount = 3;

% learnRate = 0.1;
learnRate = input('What learning rate? ( < 1 ) : ');

% learningIterations = 10000;
learningIterations = input('How many learning iterations? : ');


% % % % % FF TEST WEIGHTS % % % %
% outWeights = [0.3281,0.2372,-
0.2963,0.3283,0.7171];%[1,2,3,4,5];%rand(1,5);
% hid1Weights = [-0.4179;0.7857;0.4109;-0.4180;1.2763];%rand(5,1);
% % % % % END FF TEST WEIGHTS % % % %

outWeights = rand(1,2)./10;
hid1Weights = rand(2,2)./10;
hid2Weights = rand(2,2)./10;
biasWeights = rand(2,1)./10;


biasVal = 1;

% Build input deck
%
% learnImgIN(:,:,1) =
double(imread('..\..\..\TestData\XOR_TEST_SM\XOR_TEST_00.tif'));
% learnImgIN(:,:,2) =
double(imread('..\..\..\TestData\XOR_TEST_SM\XOR_TEST_01_2.tif'));
% learnImgIN(:,:,3) =
double(imread('..\..\..\TestData\XOR_TEST_SM\XOR_TEST_10_2.tif'));
% learnImgIN(:,:,4) =
double(imread('..\..\..\TestData\XOR_TEST_SM\XOR_TEST_11.tif'));

leanrImgIN(:,:,1) = [0;0];
leanrImgIN(:,:,2) = [0;1];
leanrImgIN(:,:,3) = [1;0];
leanrImgIN(:,:,4) = [1;1];

% learnImgIN = double(learnImgIN);

% Unwrap Images for processing:
```

```
desiredOut(1) = [0];
desiredOut(2) = [1];
desiredOut(3) = [1];
desiredOut(4) = [0];

trainingSetSize = 4; % As there are four possible inputs.

disp('---------------')
disp('Input Data:')
leanrImgIN(:,:,1)
leanrImgIN(:,:,2)
leanrImgIN(:,:,3)
leanrImgIN(:,:,4)
disp('---------------')

% figure
% subplot 221
% imshow(learnImgIN(:,:,1))
% title('Logical 00')
% subplot 222
% imshow(learnImgIN(:,:,2))
% title('Logical 01')
% subplot 223
% imshow(learnImgIN(:,:,3))
% title('Logical 10')
% subplot 224
% imshow(learnImgIN(:,:,4))
% title('Logical 11')
%
% learnImg(:,:,1) = reshape(learnImgIN(:,:,1),400,1);
% learnImg(:,:,2) = reshape(learnImgIN(:,:,2),400,1);
% learnImg(:,:,3) = reshape(learnImgIN(:,:,3),400,1);
% learnImg(:,:,4) = reshape(learnImgIN(:,:,4),400,1);

learnImg = leanrImgIN;
% learnImg = learnImg./255;

%% Learning Phase

netOut = zeros(2);

h = waitbar(0,['Learning...',num2str(learningIterations),'
Iterations']);
for iteration = 1:learningIterations
    waitbar(iteration/learningIterations,h)
    tempTestIndex = randi(trainingSetSize,1);
    tempTestInput = learnImg(:,:,tempTestIndex);
    tempTestOutput = desiredOut(tempTestIndex);

    % [hid1Out] = BP_FF(hid1Weights,inValues,hid1NeuronCount);
    % For this partiuclar application, this network is to replicate
some linear
    % equation, therefore out(n) = function(n) hence the for loop

    % Input Layer (H2 Weights)

    tempSz = size(hid2Weights);
    hid2NeuronCount = tempSz(1);

    [hid2Out] = BP_FF_BIAS(hid2Weights,biasWeights,
biasVal,tempTestInput,hid2NeuronCount);

    % Hidden Layer (H1 Weights)
    tempSz = size(hid1Weights);
    hid1NeuronCount = tempSz(1);

    [hid1Out] = BP_FF(hid1Weights,hid2Out,hid1NeuronCount);

    % Output Layer (outWeights)

    tempSz = size(outWeights);
    outNeuronCount = tempSz(1);

    [netOut] = BP_FF(outWeights, hid1Out, outNeuronCount);


    % % Back Propagation % %

    % Determine Error

    err = tempTestOutput - netOut;

    tempSz = size(outWeights);
    neuronCount = tempSz(1);

    % Update Output Layer

    for n = 1:neuronCount
        [outWeights(n,:), LMO(:,n)] = BP_OutUpdate(outWeights(n,:),
err(n,:), learnRate,hid1Out);
        [biasWeights(n,:), NUSED] = BP_OutUpdate(biasWeights(n,:),
err(n,:), learnRate, biasVal);
    end

    tempSz = size(hid1Weights);
    neuronCount = tempSz(1);

    % Update Hidden Layer 1

    for n = 1:neuronCount
        [hid1Weights(n,:), LMH1(:,n)] =
BP_HidUpdate(outWeights(:,n),hid1Weights(n,:), learnRate, hid2Out,
LMO);
    end

    % Update Hidden Layer 2

    for n = 1:neuronCount
%         [hid1Weights(n,:), LMH(n,:)] =
BP_HidUpdate(outWeights,hid1Weights(n,:), err, learnRate,
tempTestInput, hid1Out, LMO);
        [hid2Weights(n,:), LMH2(:,n)] =
BP_HidUpdate(hid1Weights(:,n),hid2Weights(n,:), learnRate,
tempTestInput, LMH1);
    end




%
%     for n = 1:neuronCount
% %         [hid1Weights(n,:), LMH(n,:)] =
BP_HidUpdate(outWeights,hid1Weights(n,:), err, learnRate,
tempTestInput, hid1Out, LMO);
```

```
%          [hid1Weights(n,:), LMH1(n,:)] =
BP_HidUpdate(outWeights,hid1Weights(n,:), learnRate, tempTestInput,
LMO);
%      end
%
%      for n = 1:neuronCount
%          [hid2Weights(n,:), LMH2(n,:)] =
BP_HidUpdate(hid1Weights,hid2Weights(n,:), learnRate, hid1Out, LMH1);
%      end

    % Weight tracking for post analysis (and stability).

    WTRACK(iteration,:)  = outWeights(1,1);
    WTRACK2(iteration,:) = hid1Weights(1,1);
    WTRACK3(iteration,:) = hid2Weights(1,1);
    WTRACK4(iteration,:) = biasWeights(1,1);
    ETRACK(iteration)   = abs(err);
end
close(h) % Close Waitbar





%% Multi-Layer Feed Forward

% Output Layer

% inValues =  learnImg(:,:,tempTestIndex);

clear hid1Out hid2Out netOut


% "Incomplete" images for testing/recognition
%
% learnImgIN(:,:,1) =
double(imread('..\..\..\TestData\XOR_TEST_SM\XOR_TEST_00.tif'));
% learnImgIN(:,:,2) =
double(imread('..\..\..\TestData\XOR_TEST_SM\XOR_TEST_01_2.tif'));
% learnImgIN(:,:,3) =
double(imread('..\..\..\TestData\XOR_TEST_SM\XOR_TEST_10_2.tif'));
% learnImgIN(:,:,4) =
double(imread('..\..\..\TestData\XOR_TEST_SM\XOR_TEST_11.tif'));
%
% figure
% subplot 221
% imshow(learnImgIN(:,:,1))
% title('Logical 00')
% subplot 222
% imshow(learnImgIN(:,:,2))
% title('Logical 01')
% subplot 223
% imshow(learnImgIN(:,:,3))
% title('Logical 10')
% subplot 224
% imshow(learnImgIN(:,:,4))
% title('Logical 11')
%
% learnImg(:,:,1) = reshape(learnImgIN(:,:,1),400,1);
% learnImg(:,:,2) = reshape(learnImgIN(:,:,2),400,1);
% learnImg(:,:,3) = reshape(learnImgIN(:,:,3),400,1);
% learnImg(:,:,4) = reshape(learnImgIN(:,:,4),400,1);

% learnImg = learnImg./255;
```

```
tempSz = size(hid2Weights);
hid2NeuronCount = tempSz(1);

for n = 1:trainingSetSize

    [hid2Out(n,:)] = BP_FF_BIAS(hid2Weights,biasWeights, biasVal,
learnImg(:,:,n),hid2NeuronCount);

end

% H1

tempSz = size(hid2Weights);
hid2NeuronCount = tempSz(1);


for n = 1:trainingSetSize

    [hid1Out(n,:)] =
BP_FF(hid1Weights,transpose(hid2Out(n,:)),hid1NeuronCount);

end

% OUT

tempSz = size(outWeights);
outNeuronCount = tempSz(1);

for n = 1:trainingSetSize
    [netOut(:,:,n)] = BP_FF(outWeights, transpose(hid1Out(n,:)),
outNeuronCount);

end




% Display Output
% figure
% plot(smallRange,netOut,'-r')
% grid
% hold on
% plot(smallRange,small_funct,'-b')
% hold off

disp('---------------')
disp('---------------')
disp('Network Result:')
netOut
disp('---------------')

figure
subplot 221
plot(1:learningIterations,WTRACK)
title('Example Output Weight Tracking')
xlabel('Iteration')
ylabel('Weight Value')
grid on
subplot 222
plot(1:learningIterations,WTRACK2)
title('Example Hid1 Weight Tracking')
xlabel('Iteration')
ylabel('Weight Value')
```

```
grid on
subplot 223
plot(1:learningIterations,WTRACK3)
title('Example Hid2 Weight Tracking')
xlabel('Iteration')
ylabel('Weight Value')
grid on
subplot 224
plot(1:learningIterations,WTRACK4)
title('Example Bias Weight Tracking')
xlabel('Iteration')
ylabel('Weight Value')
grid on

figure
plot(1:learningIterations,ETRACK)
title('Error Tracking')
xlabel('Iteration')
ylabel('Magnitude of Error')
grid on


%% Timer Output

toc(ticTimer)
```

File: BP_TopScript_V4.m

```
% Back Prop Test Script 1
%
%
%

close all
clear
clc

%% Simple Neuron Output Test

weights = [1 2 3 4 5 6 7;1 2 3 4 5 6 7; 1 2 3 4 5 6 7];
inVals = [1;2;3;4;5;6;7];
neuronCount = 3;


[layerOut] = BP_FF(weights,inVals,neuronCount);


%% Network Config and Desired Function Input

% Clear data to run second test.
clear

% Network Configuration as defined by weights.
layerCount = 2;

learnRate = 0.1;
learningIterations = 2000;


% % % % % FF TEST WEIGHTS % % % %
% outWeights = [0.3281,0.2372,-
0.2963,0.3283,0.7171];%[1,2,3,4,5];%rand(1,5);
% hid1Weights = [-0.4179;0.7857;0.4109;-0.4180;1.2763];%rand(5,1);
% % % % % END FF TEST WEIGHTS % % % %

outWeights = rand(1,5);
hid1Weights = rand(5,5);
hid2Weights = rand(5,1);

% Build input deck
% (Using the example from HW 3 in EE 570)

fullRange = linspace(0,20,500);
real_funct = fullRange.*exp(-fullRange);


smallRange = linspace(0,20,50);
small_funct = smallRange.*exp(-smallRange);

figure
subplot 311
plot(fullRange,real_funct)
title('Full Resolution Plot')
grid on
subplot 312
```

```
plot(smallRange,small_funct)
title('Reduced Resolution Plot')
grid on
subplot 313
plot(fullRange,real_funct,'-b',smallRange,small_funct,'-r')
title('Plot Overlay')
legend('Full Resolution', 'Reduced Resolution')
grid on


%% Learning Phase

netOut = zeros(length(real_funct));

for iteration = 1:learningIterations
    tempTestIndex = randi(length(fullRange),1);
    tempTestInput = fullRange(tempTestIndex);
    tempTestOutput = real_funct(tempTestIndex);

    tempSz = size(hid2Weights);
    hid2NeuronCount = tempSz(1);

    % [hid1Out] = BP_FF(hid1Weights,inValues,hid1NeuronCount);
    % For this partiuclar application, this network is to replicate
some linear
    % equation, therefore out(n) = function(n) hence the for loop

    % Input Layer (H2 Weights)

    [hid2Out] = BP_FF(hid2Weights,tempTestInput,hid2NeuronCount);

%     [hid1Out] = BP_FF(hid1Weights,tempTestInput,hid1NeuronCount);

    % Hidden Layer (H1 Weights)

    hid1NeuronCount = hid2NeuronCount; % In this case, the two neuron
counts are the same.

    [hid1Out] = BP_FF(hid1Weights,hid2Out,hid1NeuronCount);

%     [hid2Out] = BP_FF(hid2Weights,hid1Out,hid2NeuronCount);

    % Output Layer (outWeights)

    tempSz = size(outWeights);
    outNeuronCount = tempSz(1);

    [netOut] = BP_FF(outWeights, hid1Out, outNeuronCount);

%     [netOut] = BP_FF(outWeights,hid2Out,outNeuronCount);


    % % Back Propagation % %

    % Determine Error

    err = tempTestOutput - netOut;

    tempSz = size(outWeights);
    neuronCount = tempSz(1);

    % Update Output Layer

    for n = 1:neuronCount
```

```
        [outWeights(n,:), LMO(:,n)] = BP_OutUpdate(outWeights(n,:),
err(n,:), learnRate, hid1Out(:,n));
    end

    tempSz = size(hid1Weights);
    neuronCount = tempSz(1);

    % Update Hidden Layer 1

    for n = 1:neuronCount
        [hid1Weights(n,:), LMH1(:,n)] =
BP_HidUpdate(outWeights(:,n),hid1Weights(n,:), learnRate, hid2Out,
LMO);
    end

    % Update Hidden Layer 2

    for n = 1:neuronCount
%          [hid1Weights(n,:), LMH(n,:)] =
BP_HidUpdate(outWeights,hid1Weights(n,:), err, learnRate,
tempTestInput, hid1Out, LMO);
        [hid2Weights(n,:), LMH2(:,n)] =
BP_HidUpdate(hid1Weights(:,n),hid2Weights(n,:), learnRate,
tempTestInput, LMH1);
    end




%
%     for n = 1:neuronCount
% %          [hid1Weights(n,:), LMH(n,:)] =
BP_HidUpdate(outWeights,hid1Weights(n,:), err, learnRate,
tempTestInput, hid1Out, LMO);
%          [hid1Weights(n,:), LMH1(n,:)] =
BP_HidUpdate(outWeights,hid1Weights(n,:), learnRate, tempTestInput,
LMO);
%     end
%
%     for n = 1:neuronCount
%          [hid2Weights(n,:), LMH2(n,:)] =
BP_HidUpdate(hid1Weights,hid2Weights(n,:), learnRate, hid1Out, LMH1);
%     end
%
    WTRACK(iteration,:) = hid2Weights;
end






%% Multi-Layer Feed Forward

% Output Layer

inValues = smallRange;

tempSz = size(hid1Weights);
hid1NeuronCount = tempSz(1);

clear hid1Out hid2Out netOut

% [hid1Out] = BP_FF(hid1Weights,inValues,hid1NeuronCount);
% For this partiuclar application, this network is to replicate some
linear
% equation, therefore out(n) = function(n) hence the for loop

% H2

for n = 1:length(inValues)

%     [hid1Out(n,:)] = BP_FF(hid1Weights,inValues(n),hid1NeuronCount);
    [hid2Out(n,:)] = BP_FF(hid2Weights,inValues(n),hid2NeuronCount);

end

% H1

tempSz = size(outWeights);
outNeuronCount = tempSz(1);

for n = 1:length(inValues)

%     [netOut(n)] =
BP_FF(outWeights,transpose(hid1Out(n,:)),outNeuronCount);
    hid1NeuronCount = hid2NeuronCount; % In this case, the two neuron
counts are the same.

    [hid1Out(n,:)] =
BP_FF(hid1Weights,transpose(hid2Out(n,:)),hid1NeuronCount);

end

% OUT

for n = 1:length(inValues)
    [netOut(n)] = BP_FF(outWeights, transpose(hid1Out(n,:)),
outNeuronCount);

end




% Display Output
figure
plot(smallRange,netOut,'-r')
grid
hold on
plot(smallRange,small_funct,'-b')
hold off
```

File: testFileBuilder.m

```
%
% This script takes the specified images and builds a larger test set
by
% adding noise to them
%

close all
clear
clc

%% Image Import

inImg1 =
imread('..\..\TestData\EdgeData\EdgeNoiseTest_IMG_1_V3.bmp','bmp') + 1;
inImg2 =
imread('..\..\TestData\EdgeData\EdgeNoiseTest_IMG_2.bmp','bmp') + 1;
inImg3 =
imread('..\..\TestData\EdgeData\EdgeNoiseTest_IMG_3.bmp','bmp') + 1;

% % DATA VERIFICATION % %
figure
subplot 311
imshow(inImg1)
title('Original Input Images')
subplot 312
imshow(inImg2)
subplot 313
imshow(inImg3)
% % END DAT VERIFICATION % %


%% Noise Processing

% specVar1 = 0.273;
specVar1 = 0.04;
specVar2 = 0.8;

img1WNoise_O = imnoise(inImg1,'speckle',specVar1);
img2WNoise_O = imnoise(inImg2,'speckle',specVar1);
img3WNoise_O = imnoise(inImg3,'speckle',specVar1);

img1WNoise_2 = imnoise(inImg1,'speckle', specVar2);
img2WNoise_2 = imnoise(inImg2,'speckle', specVar2);
img3WNoise_2 = imnoise(inImg3,'speckle', specVar2);

% % DATA VERIFICATION % %
figure
subplot 321
imshow(img1WNoise_O)
title('Default Speckle')
subplot 322
imshow(img1WNoise_2)
title(['Speckle with V = ',num2str(specVar2)])

subplot 323
imshow(img2WNoise_O)
title('Default Speckle')
```

```
subplot 324
imshow(img2WNoise_2)
title(['Speckle with V = ',num2str(specVar2)])

subplot 325
imshow(img3WNoise_O)
title('Default Speckle')
subplot 326
imshow(img3WNoise_2)
title(['Speckle with V = ',num2str(specVar2)])

% % END DAT VERIFICATION % %

fileName = '..\..\TestData\EdgeData\ADD_NOISE_EdgeNoiseTest_IMG_1';
imwrite(img1WNoise_O,[fileName,'.bmp'],'bmp');

fileName =
'..\..\TestData\EdgeData\ADD_NOISE_EdgeNoiseTest_IMG_1_HARD';
imwrite(img1WNoise_2,[fileName,'.bmp'],'bmp');


%% Save test images
% This section creates both the 3x3 and 5x5 test kernals for known
% (visually inspected) edge locations.

% Edge Coordinates

xyIMG1_EDGE_Coords = [11,33;...
                173,33;...
                175,161;...
                233,161];...
%                 242,161];
imEd1len = length(xyIMG1_EDGE_Coords)

xyIMG2_EDGE_Coords = [32,204;...
                33,153;...
                136,65;...
                160,13];...
%                 176,17];

imEd2len = length(xyIMG2_EDGE_Coords)
xyIMG3_EDGE_Coords = [11, 192;...
                102,97;...
                171,33;...
                176,161];...
%                 229,161];
imEd3len = length(xyIMG3_EDGE_Coords)

% xyIMG1_EDGE_Coords = [11,32;...
% %                       11,33;...
%                     11,64;...
% %                       11,65;...
%                     11,96;...
% %                       11,97;...
%                     11,128;...
% %                       11,129;...
%                     11,160;...
% %                       11,161;...
%                     11,192;...
% %                       11,193;...
%                     11,224];...
% %                       11,225];
%
% imEd1len = length(xyIMG1_EDGE_Coords)
```

```
%
% xyIMG2_EDGE_Coords = [16,6;...
%                        17,6;...
%                        10,16;...
%                        10,17;...
%                        16,23;...
%                        17,23;...
%                        10,32;...
%                        10,33;...
%                        16,40;...
%                        17,40;...
%                        10,48;...
%                        10,49;...
%                        16,55;...
%                        17,55;...
%                        10,64;...
%                        10,65;...
%                        16,72;...
%                        17,72;...
%                        23,80;...
%                        23,81;...
%                        16,93;...
%                        17,93;...
%                        10,80;...
%                        10,81;...
%                        193,137;
%                        192,137];
%
% imEd2len = length(xyIMG2_EDGE_Coords)
%
% xyIMG3_EDGE_Coords = [11,32;...
%                        11,33;...
%                        11,64;...
%                        11,65;...
%                        11,96;...
%                        11,97;...
%                        11,128;...
%                        11,129;...
%                        11,160;...
%                        11,161;...
%                        11,192;...
%                        11,193;...
%                        11,224;...
%                        11,225];
%
% imEd3len = length(xyIMG3_EDGE_Coords)
%
totalEdgeImages = imEd1len + imEd2len + imEd3len
%


% Homogeneous Coordinates


xyIMG1_NO_EDGE_Coords = [18,6;...
                18,42;...
                18,103;...
                18,166];...
%                 18,236];

imNOEd1len = length(xyIMG1_NO_EDGE_Coords)

xyIMG2_NO_EDGE_Coords = [152,10;...
                152,23;...
                152,40;...
                104,112];
%                 152,57;...
%                 152,76];
imNOEd2len = length(xyIMG2_NO_EDGE_Coords)

xyIMG3_NO_EDGE_Coords = [18,6;...
                18,42;...
                18,103;...
                18,166];...
%                 18,236];
imNOEd3len = length(xyIMG3_NO_EDGE_Coords)

% xyIMG1_NO_EDGE_Coords = [10,10;...
%                          10,40;...
%                          10,90;...
%                          10,108;...
%                          10,143;...
%                          10,183;...
%                          10,219;...
%                          10,246;
%                          202,4;...
%                          202,48;...
%                          204,87;...
%                          204,119;...
%                          203,241];
%
% imNOEd1len = length(xyIMG1_NO_EDGE_Coords)
%
%
% xyIMG2_NO_EDGE_Coords = [8,5;...
%                          8,25;...
%                          8,40;...
%                          8,55;...
%                          8,77;...
%                          152,4;...
%                          152,24;...
%                          152,41;...
%                          152,56;...
%                          153,75];
%
% imNOEd2len = length(xyIMG2_NO_EDGE_Coords)
%
% xyIMG3_NO_EDGE_Coords = [10,10;...
%                          10,40;...
%                          10,90;...
%                          10,108;...
%                          10,143;...
%                          10,183;...
%                          10,219;...
%                          10,246;
%                          202,4;...
%                          202,48;...
%                          204,87;...
%                          204,119;...
%                          203,241];
%
% imNOEd3len = length(xyIMG3_NO_EDGE_Coords)
%
totalNOEdgeImages = imNOEd1len + imNOEd2len + imNOEd3len
%
%
```

```
% Collect from Images
% For loop numbers are hard coded because they are known for this
% experiment.

% Edge Pixels

% tempImage = inImg1;
tempImage = img1WNoise_O;
fileName3x3 = '..\..\TestData\EdgeData\EdgeSamples\EDGEIMG_1_3x3_';
fileName5x5 = '..\..\TestData\EdgeData\EdgeSamples\EDGEIMG_1_5x5_';

for m = 1:imEd1len
    tempR = xyIMG1_EDGE_Coords(m,1);
    tempC = xyIMG1_EDGE_Coords(m,2);
    tempImg3x3 = [tempImage(tempR+1,tempC-1), tempImage(tempR+1,tempC),
tempImage(tempR+1,tempC+1);...
                  tempImage(tempR,tempC-1), tempImage(tempR,tempC),
tempImage(tempR,tempC+1);...
                  tempImage(tempR-1,tempC-1), tempImage(tempR-1,tempC),
tempImage(tempR-1,tempC+1)];

    tempImg5x5 = [tempImage(tempR+2,tempC-2), tempImage(tempR+2,tempC-
1), tempImage(tempR+2,tempC), tempImage(tempR+2,tempC+1),
tempImage(tempR+2,tempC+2);...
                  tempImage(tempR+1,tempC-2), tempImage(tempR+1,tempC-
1), tempImage(tempR+1,tempC), tempImage(tempR+1,tempC+1),
tempImage(tempR+1,tempC+2);...
                  tempImage(tempR,tempC-2), tempImage(tempR,tempC-1),
tempImage(tempR,tempC), tempImage(tempR,tempC+1),
tempImage(tempR,tempC+2);...
                  tempImage(tempR-1,tempC-2), tempImage(tempR-1,tempC-
1), tempImage(tempR-1,tempC), tempImage(tempR-1,tempC+1),
tempImage(tempR-1,tempC+2);...
                  tempImage(tempR-2,tempC-2), tempImage(tempR-2,tempC-
1), tempImage(tempR-2,tempC), tempImage(tempR-2,tempC+1),
tempImage(tempR-2,tempC+2)];

    imwrite(tempImg3x3,[fileName3x3,num2str(m),'.bmp'],'bmp');
    imwrite(tempImg5x5,[fileName5x5,num2str(m),'.bmp'],'bmp');

end



% tempImage = inImg2;
tempImage = img2WNoise_O;
fileName3x3 = '..\..\TestData\EdgeData\EdgeSamples\EDGEIMG_2_3x3_';
fileName5x5 = '..\..\TestData\EdgeData\EdgeSamples\EDGEIMG_2_5x5_';

for m = 1:imEd2len
    tempR = xyIMG2_EDGE_Coords(m,1);
    tempC = xyIMG2_EDGE_Coords(m,2);
    tempImg3x3 = [tempImage(tempR+1,tempC-1), tempImage(tempR+1,tempC),
tempImage(tempR+1,tempC+1);...
                  tempImage(tempR,tempC-1), tempImage(tempR,tempC),
tempImage(tempR,tempC+1);...
                  tempImage(tempR-1,tempC-1), tempImage(tempR-1,tempC),
tempImage(tempR-1,tempC+1)];
```

```
    tempImg5x5 = [tempImage(tempR+2,tempC-2), tempImage(tempR+2,tempC-
1), tempImage(tempR+2,tempC), tempImage(tempR+2,tempC+1),
tempImage(tempR+2,tempC+2);...
                  tempImage(tempR+1,tempC-2), tempImage(tempR+1,tempC-
1), tempImage(tempR+1,tempC), tempImage(tempR+1,tempC+1),
tempImage(tempR+1,tempC+2);...
                  tempImage(tempR,tempC-2), tempImage(tempR,tempC-1),
tempImage(tempR,tempC), tempImage(tempR,tempC+1),
tempImage(tempR,tempC+2);...
                  tempImage(tempR-1,tempC-2), tempImage(tempR-1,tempC-
1), tempImage(tempR-1,tempC), tempImage(tempR-1,tempC+1),
tempImage(tempR-1,tempC+2);...
                  tempImage(tempR-2,tempC-2), tempImage(tempR-2,tempC-
1), tempImage(tempR-2,tempC), tempImage(tempR-2,tempC+1),
tempImage(tempR-2,tempC+2)];

    imwrite(tempImg3x3,[fileName3x3,num2str(m),'.bmp'],'bmp');
    imwrite(tempImg5x5,[fileName5x5,num2str(m),'.bmp'],'bmp');

end

% tempImage = inImg3;
tempImage = img3WNoise_O;
fileName3x3 = '..\..\TestData\EdgeData\EdgeSamples\EDGEIMG_3_3x3_';
fileName5x5 = '..\..\TestData\EdgeData\EdgeSamples\EDGEIMG_3_5x5_';

for m = 1:imEd3len
    tempR = xyIMG3_EDGE_Coords(m,1);
    tempC = xyIMG3_EDGE_Coords(m,2);
    tempImg3x3 = [tempImage(tempR+1,tempC-1), tempImage(tempR+1,tempC),
tempImage(tempR+1,tempC+1);...
                  tempImage(tempR,tempC-1), tempImage(tempR,tempC),
tempImage(tempR,tempC+1);...
                  tempImage(tempR-1,tempC-1), tempImage(tempR-1,tempC),
tempImage(tempR-1,tempC+1)];

    tempImg5x5 = [tempImage(tempR+2,tempC-2), tempImage(tempR+2,tempC-
1), tempImage(tempR+2,tempC), tempImage(tempR+2,tempC+1),
tempImage(tempR+2,tempC+2);...
                  tempImage(tempR+1,tempC-2), tempImage(tempR+1,tempC-
1), tempImage(tempR+1,tempC), tempImage(tempR+1,tempC+1),
tempImage(tempR+1,tempC+2);...
                  tempImage(tempR,tempC-2), tempImage(tempR,tempC-1),
tempImage(tempR,tempC), tempImage(tempR,tempC+1),
tempImage(tempR,tempC+2);...
                  tempImage(tempR-1,tempC-2), tempImage(tempR-1,tempC-
1), tempImage(tempR-1,tempC), tempImage(tempR-1,tempC+1),
tempImage(tempR-1,tempC+2);...
                  tempImage(tempR-2,tempC-2), tempImage(tempR-2,tempC-
1), tempImage(tempR-2,tempC), tempImage(tempR-2,tempC+1),
tempImage(tempR-2,tempC+2)];

    imwrite(tempImg3x3,[fileName3x3,num2str(m),'.bmp'],'bmp');
    imwrite(tempImg5x5,[fileName5x5,num2str(m),'.bmp'],'bmp');

end

% Areas of homogeneous pixels

% tempImage = inImg1;
tempImage = img1WNoise_O;
fileName3x3 = '..\..\TestData\EdgeData\EdgeSamples\NO_EDGEIMG_1_3x3_';
fileName5x5 = '..\..\TestData\EdgeData\EdgeSamples\NO_EDGEIMG_1_5x5_';
```

```
for m = 1:imNOEd1len
    tempR = xyIMG1_NO_EDGE_Coords(m,1);
    tempC = xyIMG1_NO_EDGE_Coords(m,2);
    tempImg3x3 = [tempImage(tempR+1,tempC-1), tempImage(tempR+1,tempC),
tempImage(tempR+1,tempC+1);...
                  tempImage(tempR,tempC-1), tempImage(tempR,tempC),
tempImage(tempR,tempC+1);...
                  tempImage(tempR-1,tempC-1), tempImage(tempR-1,tempC),
tempImage(tempR-1,tempC+1)];

    tempImg5x5 = [tempImage(tempR+2,tempC-2), tempImage(tempR+2,tempC-
1), tempImage(tempR+2,tempC), tempImage(tempR+2,tempC+1),
tempImage(tempR+2,tempC+2);...
                  tempImage(tempR+1,tempC-2), tempImage(tempR+1,tempC-
1), tempImage(tempR+1,tempC), tempImage(tempR+1,tempC+1),
tempImage(tempR+1,tempC+2);...
                  tempImage(tempR,tempC-2), tempImage(tempR,tempC-1),
tempImage(tempR,tempC), tempImage(tempR,tempC+1),
tempImage(tempR,tempC+2);...
                  tempImage(tempR-1,tempC-2), tempImage(tempR-1,tempC-
1), tempImage(tempR-1,tempC), tempImage(tempR-1,tempC+1),
tempImage(tempR-1,tempC+2);...
                  tempImage(tempR-2,tempC-2), tempImage(tempR-2,tempC-
1), tempImage(tempR-2,tempC), tempImage(tempR-2,tempC+1),
tempImage(tempR-2,tempC+2)];

    imwrite(tempImg3x3,[fileName3x3,num2str(m),'.bmp'],'bmp');
    imwrite(tempImg5x5,[fileName5x5,num2str(m),'.bmp'],'bmp');

end

% tempImage = inImg2;
tempImage = img2WNoise_O;
fileName3x3 = '..\..\TestData\EdgeData\EdgeSamples\NO_EDGEIMG_2_3x3_';
fileName5x5 = '..\..\TestData\EdgeData\EdgeSamples\NO_EDGEIMG_2_5x5_';

for m = 1:imNOEd2len
    tempR = xyIMG2_NO_EDGE_Coords(m,1);
    tempC = xyIMG2_NO_EDGE_Coords(m,2);
    tempImg3x3 = [tempImage(tempR+1,tempC-1), tempImage(tempR+1,tempC),
tempImage(tempR+1,tempC+1);...
                  tempImage(tempR,tempC-1), tempImage(tempR,tempC),
tempImage(tempR,tempC+1);...
                  tempImage(tempR-1,tempC-1), tempImage(tempR-1,tempC),
tempImage(tempR-1,tempC+1)];

    tempImg5x5 = [tempImage(tempR+2,tempC-2), tempImage(tempR+2,tempC-
1), tempImage(tempR+2,tempC), tempImage(tempR+2,tempC+1),
tempImage(tempR+2,tempC+2);...
                  tempImage(tempR+1,tempC-2), tempImage(tempR+1,tempC-
1), tempImage(tempR+1,tempC), tempImage(tempR+1,tempC+1),
tempImage(tempR+1,tempC+2);...
                  tempImage(tempR,tempC-2), tempImage(tempR,tempC-1),
tempImage(tempR,tempC), tempImage(tempR,tempC+1),
tempImage(tempR,tempC+2);...
                  tempImage(tempR-1,tempC-2), tempImage(tempR-1,tempC-
1), tempImage(tempR-1,tempC), tempImage(tempR-1,tempC+1),
tempImage(tempR-1,tempC+2);...
                  tempImage(tempR-2,tempC-2), tempImage(tempR-2,tempC-
1), tempImage(tempR-2,tempC), tempImage(tempR-2,tempC+1),
tempImage(tempR-2,tempC+2)];

    imwrite(tempImg3x3,[fileName3x3,num2str(m),'.bmp'],'bmp');
    imwrite(tempImg5x5,[fileName5x5,num2str(m),'.bmp'],'bmp');
```

```
end

% tempImage = inImg3;
tempImage = img3WNoise_O;
fileName3x3 = '..\..\TestData\EdgeData\EdgeSamples\NO_EDGEIMG_3_3x3_';
fileName5x5 = '..\..\TestData\EdgeData\EdgeSamples\NO_EDGEIMG_3_5x5_';

for m = 1:imNOEd3len
    tempR = xyIMG3_NO_EDGE_Coords(m,1);
    tempC = xyIMG3_NO_EDGE_Coords(m,2);
    tempImg3x3 = [tempImage(tempR+1,tempC-1), tempImage(tempR+1,tempC),
tempImage(tempR+1,tempC+1);...
                  tempImage(tempR,tempC-1), tempImage(tempR,tempC),
tempImage(tempR,tempC+1);...
                  tempImage(tempR-1,tempC-1), tempImage(tempR-1,tempC),
tempImage(tempR-1,tempC+1)];

    tempImg5x5 = [tempImage(tempR+2,tempC-2), tempImage(tempR+2,tempC-
1), tempImage(tempR+2,tempC), tempImage(tempR+2,tempC+1),
tempImage(tempR+2,tempC+2);...
                  tempImage(tempR+1,tempC-2), tempImage(tempR+1,tempC-
1), tempImage(tempR+1,tempC), tempImage(tempR+1,tempC+1),
tempImage(tempR+1,tempC+2);...
                  tempImage(tempR,tempC-2), tempImage(tempR,tempC-1),
tempImage(tempR,tempC), tempImage(tempR,tempC+1),
tempImage(tempR,tempC+2);...
                  tempImage(tempR-1,tempC-2), tempImage(tempR-1,tempC-
1), tempImage(tempR-1,tempC), tempImage(tempR-1,tempC+1),
tempImage(tempR-1,tempC+2);...
                  tempImage(tempR-2,tempC-2), tempImage(tempR-2,tempC-
1), tempImage(tempR-2,tempC), tempImage(tempR-2,tempC+1),
tempImage(tempR-2,tempC+2)];

    imwrite(tempImg3x3,[fileName3x3,num2str(m),'.bmp'],'bmp');
    imwrite(tempImg5x5,[fileName5x5,num2str(m),'.bmp'],'bmp');

end
```

File: testFileBuilder_2.m

```
%
% This script takes the specified images and builds a larger test set
by
% adding noise to them
%

close all
clear
clc

%% Image Import

inImg1 =
imread('..\..\TestData\EdgeData\EdgeNoiseTest_IMG_1_V3.bmp','bmp') + 1;
inImg2 =
imread('..\..\TestData\EdgeData\EdgeNoiseTest_IMG_2.bmp','bmp') + 1;
% inImg3 =
imread('..\..\TestData\EdgeData\EdgeNoiseTest_IMG_3.bmp','bmp') + 1;
inImg3 = imread('..\..\TestData\EdgeData\hb15282.tif','tiff') + 1;

% % DATA VERIFICATION % %
figure
subplot 311
imshow(inImg1)
title('Original Input Images')
subplot 312
imshow(inImg2)
subplot 313
imshow(inImg3)
% % END DAT VERIFICATION % %


%% Noise Processing

% specVar1 = 0.273;
specVar1 = 0.04;
specVar2 = 0.8;

img1WNoise_O = imnoise(inImg1,'speckle',specVar1);
img2WNoise_O = imnoise(inImg2,'speckle',specVar1);
img3WNoise_O = inImg3; % imnoise(inImg3,'speckle',specVar1);

img1WNoise_2 = imnoise(inImg1,'speckle', specVar2);
img2WNoise_2 = imnoise(inImg2,'speckle', specVar2);
img3WNoise_2 = inImg3; %imnoise(inImg3,'speckle', specVar2);

% % DATA VERIFICATION % %
figure
subplot 321
```

```
imshow(img1WNoise_O)
title('Default Speckle')
subplot 322
imshow(img1WNoise_2)
title(['Speckle with V = ',num2str(specVar2)])

subplot 323
imshow(img2WNoise_O)
title('Default Speckle')
subplot 324
imshow(img2WNoise_2)
title(['Speckle with V = ',num2str(specVar2)])

subplot 325
imshow(img3WNoise_O)
title('Default Speckle')
subplot 326
imshow(img3WNoise_2)
title(['Speckle with V = ',num2str(specVar2)])

% % END DAT VERIFICATION % %

fileName = '..\..\TestData\EdgeData\ADD_NOISE_EdgeNoiseTest_IMG_1';
imwrite(img1WNoise_O,[fileName,'.bmp'],'bmp');

fileName =
'..\..\TestData\EdgeData\ADD_NOISE_EdgeNoiseTest_IMG_1_HARD';
imwrite(img1WNoise_2,[fileName,'.bmp'],'bmp');



%% Save test images
% This section creates both the 3x3 and 5x5 test kernals for known
% (visually inspected) edge locations.

% Edge Coordinates

xyIMG1_EDGE_Coords = [11,33;...
                173,33;...
                175,161;...
                233,161];...
%                 242,161];
imEd1len = length(xyIMG1_EDGE_Coords)

xyIMG2_EDGE_Coords = [32,204;...
                33,153;...
                136,65;...
                160,13];...
%                 176,17];

imEd2len = length(xyIMG2_EDGE_Coords)
% xyIMG3_EDGE_Coords = [11, 192;...
%                 102,97;...
%                 171,33;...
%                 176,161];...
% %                 229,161];
% imEd3len = length(xyIMG3_EDGE_Coords)

xyIMG3_EDGE_Coords = [84, 84;...
                      82, 89;...
                      73, 90;...
                      77, 98];

imEd3len = length(xyIMG3_EDGE_Coords)


% xyIMG1_EDGE_Coords = [11,32;...
% %                       11,33;...
%                       11,64;...
% %                       11,65;...
%                       11,96;...
% %                       11,97;...
%                       11,128;...
% %                       11,129;...
%                       11,160;...
% %                       11,161;...
%                       11,192;...
% %                       11,193;...
%                       11,224];...
% %                       11,225];
%
% imEd1len = length(xyIMG1_EDGE_Coords)
%
% xyIMG2_EDGE_Coords = [16,6;...
%                       17,6;...
%                       10,16;...
%                       10,17;...
%                       16,23;...
%                       17,23;...
%                       10,32;...
%                       10,33;...
%                       16,40;...
%                       17,40;...
%                       10,48;...
%                       10,49;...
%                       16,55;...
%                       17,55;...
%                       10,64;...
%                       10,65;...
%                       16,72;...
%                       17,72;...
%                       23,80;...
%                       23,81;...
%                       16,93;...
%                       17,93;...
%                       10,80;...
%                       10,81;...
%                       193,137;
%                       192,137];
%
% imEd2len = length(xyIMG2_EDGE_Coords)
```

```
%
% xyIMG3_EDGE_Coords = [11,32;...
%                        11,33;...
%                        11,64;...
%                        11,65;...
%                        11,96;...
%                        11,97;...
%                        11,128;...
%                        11,129;...
%                        11,160;...
%                        11,161;...
%                        11,192;...
%                        11,193;...
%                        11,224;...
%                        11,225];
%
% imEd3len = length(xyIMG3_EDGE_Coords)
%
totalEdgeImages = imEd1len + imEd2len + imEd3len
%


% Homogeneous Coordinates


xyIMG1_NO_EDGE_Coords = [18,6;...
                18,42;...
                18,103;...
                18,166];...
%                 18,236];

imNOEd1len = length(xyIMG1_NO_EDGE_Coords)

xyIMG2_NO_EDGE_Coords = [152,10;...
                152,23;...
                152,40;...
                104,112];
%                 152,57;...
%                 152,76];
imNOEd2len = length(xyIMG2_NO_EDGE_Coords)

% xyIMG3_NO_EDGE_Coords = [18,6;...
%                 18,42;...
%                 18,103;...
%                 18,166];...
% %                 18,236];
% imNOEd3len = length(xyIMG3_NO_EDGE_Coords)

xyIMG3_NO_EDGE_Coords = [13, 5;...
                         89, 5;...
                         92, 21;...
                         85, 69];

%                 18,236];
imNOEd3len = length(xyIMG3_NO_EDGE_Coords)
```

```
% xyIMG1_NO_EDGE_Coords = [10,10;...
%                          10,40;...
%                          10,90;...
%                          10,108;...
%                          10,143;...
%                          10,183;...
%                          10,219;...
%                          10,246;
%                          202,4;...
%                          202,48;...
%                          204,87;...
%                          204,119;...
%                          203,241];
%
% imNOEd1len = length(xyIMG1_NO_EDGE_Coords)
%
%
% xyIMG2_NO_EDGE_Coords = [8,5;...
%                          8,25;...
%                          8,40;...
%                          8,55;...
%                          8,77;...
%                          152,4;...
%                          152,24;...
%                          152,41;...
%                          152,56;...
%                          153,75];
%
% imNOEd2len = length(xyIMG2_NO_EDGE_Coords)
%
% xyIMG3_NO_EDGE_Coords = [10,10;...
%                          10,40;...
%                          10,90;...
%                          10,108;...
%                          10,143;...
%                          10,183;...
%                          10,219;...
%                          10,246;
%                          202,4;...
%                          202,48;...
%                          204,87;...
%                          204,119;...
%                          203,241];
%
% imNOEd3len = length(xyIMG3_NO_EDGE_Coords)
%
totalNOEdgeImages = imNOEd1len + imNOEd2len + imNOEd3len
%
%
```

```
% Collect from Images
% For loop numbers are hard coded because they are known for this
% experiment.

% Edge Pixels

% tempImage = inImg1;
tempImage = img1WNoise_O;
fileName3x3 = '..\..\TestData\EdgeData\EdgeSamples\EDGEIMG_1_3x3_';
fileName5x5 = '..\..\TestData\EdgeData\EdgeSamples\EDGEIMG_1_5x5_';

for m = 1:imEd1len
    tempR = xyIMG1_EDGE_Coords(m,1);
    tempC = xyIMG1_EDGE_Coords(m,2);
    tempImg3x3 = [tempImage(tempR+1,tempC-1), tempImage(tempR+1,tempC),
tempImage(tempR+1,tempC+1);...
                  tempImage(tempR,tempC-1), tempImage(tempR,tempC),
tempImage(tempR,tempC+1);...
                  tempImage(tempR-1,tempC-1), tempImage(tempR-1,tempC),
tempImage(tempR-1,tempC+1)];

    tempImg5x5 = [tempImage(tempR+2,tempC-2), tempImage(tempR+2,tempC-
1), tempImage(tempR+2,tempC), tempImage(tempR+2,tempC+1),
tempImage(tempR+2,tempC+2);...
                  tempImage(tempR+1,tempC-2), tempImage(tempR+1,tempC-
1), tempImage(tempR+1,tempC), tempImage(tempR+1,tempC+1),
tempImage(tempR+1,tempC+2);...
                  tempImage(tempR,tempC-2), tempImage(tempR,tempC-1),
tempImage(tempR,tempC), tempImage(tempR,tempC+1),
tempImage(tempR,tempC+2);...
                  tempImage(tempR-1,tempC-2), tempImage(tempR-1,tempC-
1), tempImage(tempR-1,tempC), tempImage(tempR-1,tempC+1),
tempImage(tempR-1,tempC+2);...
                  tempImage(tempR-2,tempC-2), tempImage(tempR-2,tempC-
1), tempImage(tempR-2,tempC), tempImage(tempR-2,tempC+1),
tempImage(tempR-2,tempC+2)];

    imwrite(tempImg3x3,[fileName3x3,num2str(m),'.bmp'],'bmp');
    imwrite(tempImg5x5,[fileName5x5,num2str(m),'.bmp'],'bmp');

end




% tempImage = inImg2;
tempImage = img2WNoise_O;
fileName3x3 = '..\..\TestData\EdgeData\EdgeSamples\EDGEIMG_2_3x3_';
fileName5x5 = '..\..\TestData\EdgeData\EdgeSamples\EDGEIMG_2_5x5_';

for m = 1:imEd2len
    tempR = xyIMG2_EDGE_Coords(m,1);
    tempC = xyIMG2_EDGE_Coords(m,2);
    tempImg3x3 = [tempImage(tempR+1,tempC-1), tempImage(tempR+1,tempC),
tempImage(tempR+1,tempC+1);...
```

```
                  tempImage(tempR,tempC-1), tempImage(tempR,tempC),
tempImage(tempR,tempC+1);...
                  tempImage(tempR-1,tempC-1), tempImage(tempR-1,tempC),
tempImage(tempR-1,tempC+1)];

    tempImg5x5 = [tempImage(tempR+2,tempC-2), tempImage(tempR+2,tempC-
1), tempImage(tempR+2,tempC), tempImage(tempR+2,tempC+1),
tempImage(tempR+2,tempC+2);...
                  tempImage(tempR+1,tempC-2), tempImage(tempR+1,tempC-
1), tempImage(tempR+1,tempC), tempImage(tempR+1,tempC+1),
tempImage(tempR+1,tempC+2);...
                  tempImage(tempR,tempC-2), tempImage(tempR,tempC-1),
tempImage(tempR,tempC), tempImage(tempR,tempC+1),
tempImage(tempR,tempC+2);...
                  tempImage(tempR-1,tempC-2), tempImage(tempR-1,tempC-
1), tempImage(tempR-1,tempC), tempImage(tempR-1,tempC+1),
tempImage(tempR-1,tempC+2);...
                  tempImage(tempR-2,tempC-2), tempImage(tempR-2,tempC-
1), tempImage(tempR-2,tempC), tempImage(tempR-2,tempC+1),
tempImage(tempR-2,tempC+2)];

    imwrite(tempImg3x3,[fileName3x3,num2str(m),'.bmp'],'bmp');
    imwrite(tempImg5x5,[fileName5x5,num2str(m),'.bmp'],'bmp');

end

% tempImage = inImg3;
tempImage = img3WNoise_O;
fileName3x3 = '..\..\TestData\EdgeData\EdgeSamples\EDGEIMG_3_3x3_';
fileName5x5 = '..\..\TestData\EdgeData\EdgeSamples\EDGEIMG_3_5x5_';

for m = 1:imEd3len
    tempR = xyIMG3_EDGE_Coords(m,1);
    tempC = xyIMG3_EDGE_Coords(m,2);
    tempImg3x3 = [tempImage(tempR+1,tempC-1), tempImage(tempR+1,tempC),
tempImage(tempR+1,tempC+1);...
                  tempImage(tempR,tempC-1), tempImage(tempR,tempC),
tempImage(tempR,tempC+1);...
                  tempImage(tempR-1,tempC-1), tempImage(tempR-1,tempC),
tempImage(tempR-1,tempC+1)];

    tempImg5x5 = [tempImage(tempR+2,tempC-2), tempImage(tempR+2,tempC-
1), tempImage(tempR+2,tempC), tempImage(tempR+2,tempC+1),
tempImage(tempR+2,tempC+2);...
                  tempImage(tempR+1,tempC-2), tempImage(tempR+1,tempC-
1), tempImage(tempR+1,tempC), tempImage(tempR+1,tempC+1),
tempImage(tempR+1,tempC+2);...
                  tempImage(tempR,tempC-2), tempImage(tempR,tempC-1),
tempImage(tempR,tempC), tempImage(tempR,tempC+1),
tempImage(tempR,tempC+2);...
                  tempImage(tempR-1,tempC-2), tempImage(tempR-1,tempC-
1), tempImage(tempR-1,tempC), tempImage(tempR-1,tempC+1),
tempImage(tempR-1,tempC+2);...
```

```
                  tempImage(tempR-2,tempC-2), tempImage(tempR-2,tempC-
1), tempImage(tempR-2,tempC), tempImage(tempR-2,tempC+1),
tempImage(tempR-2,tempC+2)];

    imwrite(tempImg3x3,[fileName3x3,num2str(m),'.bmp'],'bmp');
    imwrite(tempImg5x5,[fileName5x5,num2str(m),'.bmp'],'bmp');

end

% Areas of homogeneous pixels

% tempImage = inImg1;
tempImage = img1WNoise_O;
fileName3x3 = '..\..\TestData\EdgeData\EdgeSamples\NO_EDGEIMG_1_3x3_';
fileName5x5 = '..\..\TestData\EdgeData\EdgeSamples\NO_EDGEIMG_1_5x5_';

for m = 1:imNOEd1len
    tempR = xyIMG1_NO_EDGE_Coords(m,1);
    tempC = xyIMG1_NO_EDGE_Coords(m,2);
    tempImg3x3 = [tempImage(tempR+1,tempC-1), tempImage(tempR+1,tempC),
tempImage(tempR+1,tempC+1);...
                  tempImage(tempR,tempC-1), tempImage(tempR,tempC),
tempImage(tempR,tempC+1);...
                  tempImage(tempR-1,tempC-1), tempImage(tempR-1,tempC),
tempImage(tempR-1,tempC+1)];

    tempImg5x5 = [tempImage(tempR+2,tempC-2), tempImage(tempR+2,tempC-
1), tempImage(tempR+2,tempC), tempImage(tempR+2,tempC+1),
tempImage(tempR+2,tempC+2);...
                  tempImage(tempR+1,tempC-2), tempImage(tempR+1,tempC-
1), tempImage(tempR+1,tempC), tempImage(tempR+1,tempC+1),
tempImage(tempR+1,tempC+2);...
                  tempImage(tempR,tempC-2), tempImage(tempR,tempC-1),
tempImage(tempR,tempC), tempImage(tempR,tempC+1),
tempImage(tempR,tempC+2);...
                  tempImage(tempR-1,tempC-2), tempImage(tempR-1,tempC-
1), tempImage(tempR-1,tempC), tempImage(tempR-1,tempC+1),
tempImage(tempR-1,tempC+2);...
                  tempImage(tempR-2,tempC-2), tempImage(tempR-2,tempC-
1), tempImage(tempR-2,tempC), tempImage(tempR-2,tempC+1),
tempImage(tempR-2,tempC+2)];

    imwrite(tempImg3x3,[fileName3x3,num2str(m),'.bmp'],'bmp');
    imwrite(tempImg5x5,[fileName5x5,num2str(m),'.bmp'],'bmp');

end

% tempImage = inImg2;
tempImage = img2WNoise_O;
fileName3x3 = '..\..\TestData\EdgeData\EdgeSamples\NO_EDGEIMG_2_3x3_';
fileName5x5 = '..\..\TestData\EdgeData\EdgeSamples\NO_EDGEIMG_2_5x5_';

for m = 1:imNOEd2len
    tempR = xyIMG2_NO_EDGE_Coords(m,1);
    tempC = xyIMG2_NO_EDGE_Coords(m,2);
    tempImg3x3 = [tempImage(tempR+1,tempC-1), tempImage(tempR+1,tempC),
tempImage(tempR+1,tempC+1);...
                  tempImage(tempR,tempC-1), tempImage(tempR,tempC),
tempImage(tempR,tempC+1);...
                  tempImage(tempR-1,tempC-1), tempImage(tempR-1,tempC),
tempImage(tempR-1,tempC+1)];

    tempImg5x5 = [tempImage(tempR+2,tempC-2), tempImage(tempR+2,tempC-
1), tempImage(tempR+2,tempC), tempImage(tempR+2,tempC+1),
tempImage(tempR+2,tempC+2);...
                  tempImage(tempR+1,tempC-2), tempImage(tempR+1,tempC-
1), tempImage(tempR+1,tempC), tempImage(tempR+1,tempC+1),
tempImage(tempR+1,tempC+2);...
                  tempImage(tempR,tempC-2), tempImage(tempR,tempC-1),
tempImage(tempR,tempC), tempImage(tempR,tempC+1),
tempImage(tempR,tempC+2);...
                  tempImage(tempR-1,tempC-2), tempImage(tempR-1,tempC-
1), tempImage(tempR-1,tempC), tempImage(tempR-1,tempC+1),
tempImage(tempR-1,tempC+2);...
                  tempImage(tempR-2,tempC-2), tempImage(tempR-2,tempC-
1), tempImage(tempR-2,tempC), tempImage(tempR-2,tempC+1),
tempImage(tempR-2,tempC+2)];

    imwrite(tempImg3x3,[fileName3x3,num2str(m),'.bmp'],'bmp');
    imwrite(tempImg5x5,[fileName5x5,num2str(m),'.bmp'],'bmp');

end

% tempImage = inImg3;
tempImage = img3WNoise_O;
fileName3x3 = '..\..\TestData\EdgeData\EdgeSamples\NO_EDGEIMG_3_3x3_';
fileName5x5 = '..\..\TestData\EdgeData\EdgeSamples\NO_EDGEIMG_3_5x5_';

for m = 1:imNOEd3len
    tempR = xyIMG3_NO_EDGE_Coords(m,1);
    tempC = xyIMG3_NO_EDGE_Coords(m,2);
    tempImg3x3 = [tempImage(tempR+1,tempC-1), tempImage(tempR+1,tempC),
tempImage(tempR+1,tempC+1);...
                  tempImage(tempR,tempC-1), tempImage(tempR,tempC),
tempImage(tempR,tempC+1);...
                  tempImage(tempR-1,tempC-1), tempImage(tempR-1,tempC),
tempImage(tempR-1,tempC+1)];

    tempImg5x5 = [tempImage(tempR+2,tempC-2), tempImage(tempR+2,tempC-
1), tempImage(tempR+2,tempC), tempImage(tempR+2,tempC+1),
tempImage(tempR+2,tempC+2);...
                  tempImage(tempR+1,tempC-2), tempImage(tempR+1,tempC-
1), tempImage(tempR+1,tempC), tempImage(tempR+1,tempC+1),
tempImage(tempR+1,tempC+2);...
                  tempImage(tempR,tempC-2), tempImage(tempR,tempC-1),
tempImage(tempR,tempC), tempImage(tempR,tempC+1),
tempImage(tempR,tempC+2);...
                  tempImage(tempR-1,tempC-2), tempImage(tempR-1,tempC-
1), tempImage(tempR-1,tempC), tempImage(tempR-1,tempC+1),
tempImage(tempR-1,tempC+2);...
```

```
                  tempImage(tempR-2,tempC-2), tempImage(tempR-2,tempC-
1), tempImage(tempR-2,tempC), tempImage(tempR-2,tempC+1),
tempImage(tempR-2,tempC+2)];

    imwrite(tempImg3x3,[fileName3x3,num2str(m),'.bmp'],'bmp');
    imwrite(tempImg5x5,[fileName5x5,num2str(m),'.bmp'],'bmp');

end
```

File: NN_EdgeDetector_LEARN.m

```
%
%
% This script is used to learn the ANN
%
%
%

close all
clear
clc

programTimer = tic;

%% Input and settings

% Build input deck with kernels that have known edges in them.

% testImages = zeros(5,5,3);
imEd1len = 4;
imEd2len = 4;
imEd3len = 4;

imNOEd1len = 4;
imNOEd2len = 4;
imNOEd3len = 4;


for n = 1:imEd1len
    testImages(:,:,n) =
imread(['..\..\TestData\EdgeData\EdgeSamples\EDGEIMG_1_5x5_',num2str(n)
,'.bmp']);
end

for n = 1:imEd2len
    testImages(:,:,n+imEd1len) =
imread(['..\..\TestData\EdgeData\EdgeSamples\EDGEIMG_2_5x5_',num2str(n)
,'.bmp']);
end

for n = 1:imEd3len
    testImages(:,:,n+imEd1len+imEd2len) =
imread(['..\..\TestData\EdgeData\EdgeSamples\EDGEIMG_3_5x5_',num2str(n)
,'.bmp']);
end

edgeTotalCount = imEd1len + imEd2len + imEd3len;

% Add in areas that are homogeneous

for n = 1:imNOEd1len
    testImages(:,:,n+edgeTotalCount) =
imread(['..\..\TestData\EdgeData\EdgeSamples\NO_EDGEIMG_3_5x5_',num2str
(n),'.bmp']);
end

for n = 1:imNOEd2len
```

```
    testImages(:,:,n+edgeTotalCount+imNOEd1len) =
imread(['..\..\TestData\EdgeData\EdgeSamples\NO_EDGEIMG_3_5x5_',num2str
(n),'.bmp']);
end

for n = 1:imNOEd3len
    testImages(:,:,n+edgeTotalCount+imNOEd1len+imNOEd2len) =
imread(['..\..\TestData\EdgeData\EdgeSamples\NO_EDGEIMG_3_5x5_',num2str
(n),'.bmp']);
end

noEdgeTotalCount = imNOEd1len + imNOEd2len + imNOEd3len;

tempTrainSz = size(testImages);
trainSz = tempTrainSz(3);

testImages = double(testImages);

%% ANN Configuration

% % Weights as specified by the paper
% hidWeights = rand(5,3) - 0.5;
% outWeights = rand(2,5) - 0.5;
%
% hidBiasW = rand(5,1) - 0.5;
% outBiasW = rand(2,1) - 0.5;

% % Varied Weights not specified by paper
% hidWeights = rand(10,3) - 0.5;
% outWeights = rand(2,10) - 0.5;
%
% hidBiasW = rand(10,1) - 0.5;
% outBiasW = rand(2,1) - 0.5;

% % Fixed Starting Weights
% hidWeights = 0.1*ones(5,3);
% outWeights = 0.1*ones(2,5);
%
% hidBiasW = -0.1*ones(5,1);
% outBiasW = -0.1*ones(2,1);

% Load weights from previous run
load('WW3_MSTAR_WOnly.mat');

%% Learning Stage

biasVal = 1;

% learnN = 0.1;
learnN = input('Learning Rate? ( < 1) : ');

% iterationCount = 100000;
iterationCount = input('Number of iterations? : ');

%%

h = waitbar(0,'Edge Detection Network Learning...');
for iteration = 1:iterationCount
    waitbar(iteration/iterationCount,h)

    curTrainIndex = randi(trainSz,1,1);
    curKernel = testImages(:,:,curTrainIndex);
    curKernel = curKernel./255;
    % % FEED FORWARD % %
    % This Section was copied from the NN Edge Detector M-File

    % Determine RLV
    tempKerMean = mean(mean(curKernel));
    tempKerVar = var(var(curKernel));
    scatterVar = 0.273;

%     inVals(1) = (tempKerVar)/(scatterVar*tempKerMean + eps);
    inVals(1) = tempKerVar;
%     inVals(1)
    % Determine RHD



    inVals(2) = ratioHarrisED(curKernel,0.04);
%     inVals(2) = (tempKerVar)/(scatterVar*tempKerMean + 0.1);
%     inVals(2) = inVals(1);

    % Determine DCT

    tempDCT = dct(curKernel);

    tempDCT2 = tempDCT > 1.35 * tempKerMean;


    inVals(3) = nnz(tempDCT2);




    % Write image pixel

    tempSz = size(hidWeights);
    hidNeuronCount = tempSz(1);
    [hidLayerOut] = 
BP_FF_BIAS(hidWeights,hidBiasW,biasVal,transpose(inVals),hidNeuronCount
); % 5 Hid Neurons
%     [hidLayerOut] = 
BP_FF(hidWeights,transpose(inVals),hidNeuronCount);

    tempSz = size(outWeights);
    outNeuronCount = tempSz(1);
%     [ANNOut] = BP_FF(outWeights,hidLayerOut,outNeuronCount); % 2 
Outputs
    [ANNOut] = BP_FF_BIAS(outWeights, outBiasW, biasVal, hidLayerOut, 
outNeuronCount); % 2 Outputs

    edgeImg = ANNOut;

%     if (ANNOut(1) >= 0.5) % Edge Detected
%         edgeImg = [1;0];
%         disp('Edge')
%     elseif (ANNOut(2) > 0.5) % Homogeneous Region Detected
%         edgeImg = [0;1];
%         disp('NO Edge')
%     else
%         edgeImg = [0;1];
% 
disp('*********************************ERROR************************
*************')
%     end
%     if (ANNOut(1) >= 0.5 && ANNOut(2) < 0.5) % Edge Detected
%         edgeImg = [1;0];
%     elseif (ANNOut(2) >= 0.5 && ANNOut(1) < 0.5) % Homogeneous Region
Detected
%         edgeImg = [0;1];
%     else
%         edgeImg = ANNOut;
%
disp('****************************ERROR**********************')
%     end

    % % BACK PROPAGATION % %

    if curTrainIndex < edgeTotalCount + 1
        curTrainDesiredOut = [1;-1];
%         curTrainDesiredOut = [0.5;-0.5];
%         curTrainDesiredOut = [1;0];
%         disp('Edge')
    elseif curTrainIndex > edgeTotalCount
        curTrainDesiredOut = [-1;1];
%         curTrainDesiredOut = [-0.5;0.5];
%         curTrainDesiredOut = [0;1];
%         disp('No Edge')
    else
        curTrainDesiredOut = [-1;1];
%         curTrainDesiredOut = [-0.5;0.5];
%         curTrainDesiredOut = [0;1];

disp('*********************************ERRR************************')
    end

    err = curTrainDesiredOut - edgeImg;

    % Output Layer Update

    tempNNSz = size(outWeights);
    outNeuronCount = tempNNSz(1);

    for n = 1:outNeuronCount
        [outWeights(n,:), LMO(:,n)] = BP_OutUpdate(outWeights(n,:),
err(n,:), learnN, hidLayerOut);
        [outBiasW(n,:), NUSED] = BP_OutUpdate(outBiasW(n,:), err(n,:),
learnN, biasVal);
    end

    % Hidden Layer Update

    tempNNSz = size(hidWeights);
    hidNeuronCount = tempNNSz(1);

    for n = 1:hidNeuronCount
        [hidWeights(n,:), LMH(:,n)] =
BP_HidUpdate(outWeights(:,n),hidWeights(n,:),learnN,transpose(inVals),
LMO);
        [hidBiasW(n,:), NUSED2] =
BP_HidUpdate(outWeights(:,n),hidBiasW(n,:),learnN,biasVal, LMO);
    end

%     [hid1Out] = BP_FF_BIAS(hidWeights, inVals, hidNeuronCount);


    ERRTRACK(:,iteration) = abs(err);
    WTRACKO(iteration) = outWeights(1,1);
    WTRACKH(iteration) = hidWeights(1,1);
    WTRACKBO(iteration) = outBiasW(1,1);
    WTRACKBH(iteration) = hidBiasW(1,1);

end
close(h)

figure
subplot 221
plot(1:iterationCount,WTRACKO);
title('Output Weight Track')
grid ON
subplot 222
plot(1:iterationCount,WTRACKH);
title('Hidden Weight Track')
grid ON
subplot 223
plot(1:iterationCount,WTRACKBO);
title('Bias Output Weight Track')
grid ON
subplot 224
plot(1:iterationCount,WTRACKBH);
title('Bias Hidden Weight Track')
grid ON

figure
subplot 121
plot(1:iterationCount,ERRTRACK(1,:))
title('Error Tracking 1')
grid ON
subplot 122
plot(1:iterationCount,ERRTRACK(2,:))
title('Error Tracking 2')
grid ON

%%
totalTime = toc(programTimer);
disp(['Learning Time Elapsed: ', num2str(totalTime),'s'])

disp('Running Edge Detector...')
run('testEdgeScript')

totalTime = toc(programTimer);
disp(['Learning to Image Time Elapsed: ', num2str(totalTime),'s'])
```

File: NN_EdgeDetector_LEARN_2.m

```
%
%
% This script is used to learn the ANN
%
%
%

close all
clear
clc

programTimer = tic;

%% Input and settings

% Build input deck with kernels that have known edges in them.

% testImages = zeros(5,5,3);
imEd1len = 4;
imEd2len = 4;
imEd3len = 4;

imNOEd1len = 4;
imNOEd2len = 4;
imNOEd3len = 4;


for n = 1:imEd1len
    testImages(:,:,n) = imread(['..\..\TestData\EdgeData\EdgeSamples
Good Set 1\EDGEIMG_1_5x5_',num2str(n),'.bmp']);
end

for n = 1:imEd2len
    testImages(:,:,n+imEd1len) =
imread(['..\..\TestData\EdgeData\EdgeSamples Good Set
1\EDGEIMG_2_5x5_',num2str(n),'.bmp']);
end

for n = 1:imEd3len
    testImages(:,:,n+imEd1len+imEd2len) =
imread(['..\..\TestData\EdgeData\EdgeSamples Good Set
1\EDGEIMG_3_5x5_',num2str(n),'.bmp']);
end

edgeTotalCount = imEd1len + imEd2len + imEd3len;

% Add in areas that are homogeneous

for n = 1:imNOEd1len
    testImages(:,:,n+edgeTotalCount) =
imread(['..\..\TestData\EdgeData\EdgeSamples Good Set
1\NO_EDGEIMG_3_5x5_',num2str(n),'.bmp']);
end

for n = 1:imNOEd2len
```

```
    testImages(:,:,n+edgeTotalCount+imNOEd1len) =
imread(['..\..\TestData\EdgeData\EdgeSamples Good Set
1\NO_EDGEIMG_3_5x5_',num2str(n),'.bmp']);
end

for n = 1:imNOEd3len
    testImages(:,:,n+edgeTotalCount+imNOEd1len+imNOEd2len) =
imread(['..\..\TestData\EdgeData\EdgeSamples Good Set
1\NO_EDGEIMG_3_5x5_',num2str(n),'.bmp']);
end

noEdgeTotalCount = imNOEd1len + imNOEd2len + imNOEd3len;

tempTrainSz = size(testImages);
trainSz = tempTrainSz(3);

testImages = double(testImages);

%% ANN Configuration

% % Weights as specified by the paper
% hidWeights = rand(5,3) - 0.5;
% outWeights = rand(2,5) - 0.5;
%
% hidBiasW = rand(5,1) - 0.5;
% outBiasW = rand(2,1) - 0.5;

% % Varied Weights not specified by paper
% hidWeights = rand(10,3) - 0.5;
% outWeights = rand(2,10) - 0.5;
%
% hidBiasW = rand(10,1) - 0.5;
% outBiasW = rand(2,1) - 0.5;

% % Fixed Starting Weights
% hidWeights = 0.1*ones(5,3);
% outWeights = 0.1*ones(2,5);
%
% hidBiasW = -0.1*ones(5,1);
% outBiasW = -0.1*ones(2,1);

% Load weights from previous run
load('WW3_MSTAR_WOnly.mat');

%% Learning Stage

biasVal = 1;

% learnN = 0.1;
learnN = input('Learning Rate? ( < 1) : ');

% iterationCount = 100000;
iterationCount = input('Number of iterations? : ');

%%

h = waitbar(0,'Edge Detection Network Learning...');
for iteration = 1:iterationCount
    waitbar(iteration/iterationCount,h)

    curTrainIndex = randi(trainSz,1,1);
    curKernel = testImages(:,:,curTrainIndex);
    curKernel = curKernel./255;
    % % FEED FORWARD % %
    % This Section was copied from the NN Edge Detector M-File

    % Determine RLV
    tempKerMean = mean(mean(curKernel));
    tempKerVar = var(var(curKernel));
    scatterVar = 0.273;

%     inVals(1) = (tempKerVar)/(scatterVar*tempKerMean + eps);
    inVals(1) = tempKerVar;
%     inVals(1)
    % Determine RHD



    inVals(2) = ratioHarrisED(curKernel,0.04);
%     inVals(2) = (tempKerVar)/(scatterVar*tempKerMean + 0.1);
%     inVals(2) = inVals(1);

    % Determine DCT

    tempDCT = dct(curKernel);

    tempDCT2 = tempDCT > 1.35 * tempKerMean;


    inVals(3) = nnz(tempDCT2);




    % Write image pixel

    tempSz = size(hidWeights);
    hidNeuronCount = tempSz(1);
    [hidLayerOut] =
BP_FF_BIAS(hidWeights,hidBiasW,biasVal,transpose(inVals),hidNeuronCount
); % 5 Hid Neurons
%     [hidLayerOut] =
BP_FF(hidWeights,transpose(inVals),hidNeuronCount);

    tempSz = size(outWeights);
    outNeuronCount = tempSz(1);
%     [ANNOut] = BP_FF(outWeights,hidLayerOut,outNeuronCount); % 2
Outputs
    [ANNOut] = BP_FF_BIAS(outWeights, outBiasW, biasVal, hidLayerOut,
outNeuronCount); % 2 Outputs

    edgeImg = ANNOut;

%     if (ANNOut(1) >= 0.5) % Edge Detected
%         edgeImg = [1;0];
%         disp('Edge')
%     elseif (ANNOut(2) > 0.5) % Homogeneous Region Detected
%         edgeImg = [0;1];
%         disp('NO Edge')
%     else
%         edgeImg = [0;1];
%
disp('*********************************ERROR*************************
*************')
%     end
%     if (ANNOut(1) >= 0.5 && ANNOut(2) < 0.5) % Edge Detected
%         edgeImg = [1;0];
%     elseif (ANNOut(2) >= 0.5 && ANNOut(1) < 0.5) % Homogeneous Region
Detected
%         edgeImg = [0;1];
%     else
%         edgeImg = ANNOut;
%
disp('****************************ERROR**********************')
%     end

    % % BACK PROPAGATION % %

    if curTrainIndex < edgeTotalCount + 1
        curTrainDesiredOut = [1;-1];
%         curTrainDesiredOut = [0.5;-0.5];
%         curTrainDesiredOut = [1;0];
%         disp('Edge')
    elseif curTrainIndex > edgeTotalCount
        curTrainDesiredOut = [-1;1];
%         curTrainDesiredOut = [-0.5;0.5];
%         curTrainDesiredOut = [0;1];
%         disp('No Edge')
    else
        curTrainDesiredOut = [-1;1];
%         curTrainDesiredOut = [-0.5;0.5];
%         curTrainDesiredOut = [0;1];

disp('**********************************ERRR************************')
    end

    err = curTrainDesiredOut - edgeImg;

    % Output Layer Update

    tempNNSz = size(outWeights);
    outNeuronCount = tempNNSz(1);

    for n = 1:outNeuronCount
        [outWeights(n,:), LMO(:,n)] = BP_OutUpdate(outWeights(n,:),
err(n,:), learnN, hidLayerOut);
        [outBiasW(n,:), NUSED] = BP_OutUpdate(outBiasW(n,:), err(n,:),
learnN, biasVal);
    end

    % Hidden Layer Update

    tempNNSz = size(hidWeights);
    hidNeuronCount = tempNNSz(1);

    for n = 1:hidNeuronCount
        [hidWeights(n,:), LMH(:,n)] =
BP_HidUpdate(outWeights(:,n),hidWeights(n,:),learnN,transpose(inVals),
LMO);
        [hidBiasW(n,:), NUSED2] =
BP_HidUpdate(outWeights(:,n),hidBiasW(n,:),learnN,biasVal, LMO);
    end

%     [hid1Out] = BP_FF_BIAS(hidWeights, inVals, hidNeuronCount);


    ERRTRACK(:,iteration) = abs(err);
    WTRACKO(iteration) = outWeights(1,1);
    WTRACKH(iteration) = hidWeights(1,1);
    WTRACKBO(iteration) = outBiasW(1,1);
    WTRACKBH(iteration) = hidBiasW(1,1);

end
close(h)

figure
subplot 221
plot(1:iterationCount,WTRACKO);
title('Output Weight Track')
grid ON
subplot 222
plot(1:iterationCount,WTRACKH);
title('Hidden Weight Track')
grid ON
subplot 223
plot(1:iterationCount,WTRACKBO);
title('Bias Output Weight Track')
grid ON
subplot 224
plot(1:iterationCount,WTRACKBH);
title('Bias Hidden Weight Track')
grid ON

figure
subplot 121
plot(1:iterationCount,ERRTRACK(1,:))
title('Error Tracking 1')
grid ON
subplot 122
plot(1:iterationCount,ERRTRACK(2,:))
title('Error Tracking 2')
grid ON

%%
totalTime = toc(programTimer);
disp(['Learning Time Elapsed: ', num2str(totalTime),'s'])

disp('Running Edge Detector...')
run('testEdgeScript_2')

totalTime = toc(programTimer);
disp(['Learning to Image Time Elapsed: ', num2str(totalTime),'s'])
```

File: ratioHarrisED.m

```
function [pixelVal] = ratioHarrisED(SW,k)
% Computes the Ratio-based Harris Edge Detection method
%
%
%
% SW = SW + eps; % Prevents NaN error for zero-input matricies
SW = SW + 1; % Prevents NaN error for zero-input matricies
tranSW = transpose(SW);

topRows = [1,0;0,1;0,0;0,0;0,0];
bottomRows = [0,0;0,0;0,0;1,0;0,1];
upLDiag = [1,1,1,1,0;1,1,1,0,0;1,1,0,0,0;1,0,0,0,0;0,0,0,0,0];
bottomRDiag = [0,0,0,0,0;0,0,0,0,1;0,0,0,1,1;0,0,1,1,1;0,1,1,1,1];
upRDiag = rot90(bottomRDiag);
bottomLDiag = rot90(upLDiag);

Ih  = mean(mean(SW*topRows)) / mean(mean(SW*bottomRows));
Iv  = mean(mean(tranSW*topRows)) / mean(mean(tranSW*bottomRows));
Imd = mean(mean(SW.*upLDiag)) / mean(mean(SW.*bottomRDiag));
Iad = mean(mean(SW.*upRDiag)) / mean(mean(SW.*bottomLDiag));


M1 = [Ih^2,Ih*Iv;Ih*Iv,Iv^2];
M2 = [Imd^2,Imd*Iad;Imd*Iad,Iad^2];


[x1,lambda1] = eig(M1);
[x2,lambda2] = eig(M2);

TR1 = (sum(sum(lambda1)))^2;
TR2 = (sum(sum(lambda2)))^2;

R1 = det(M1)-k*TR1;
R2 = det(M2)-k*TR2;

pixelVal = min(R1,R2);





end
```

File: NN_EdgeDetector5x5.m

```
function [edgeImg] = NN_EdgeDetector5x5(inputImage, outWeights,
outBiasW, hidWeights, hidBiasW)
%
%
% This function is a Neural Network assisted Edge detector. By
combining
% the information from several different algorithms the network was
trained
% to find edges based off of three calculated pieces of information
about a
% 5x5 kernal of the image. Returns an edge-detected image that is
truncated
% by two pixel columns and two pixel rows (due to kernal based image
% processing.)
%
% Relative Local Variance (RLV)
%
%
% Ratio-based Harris Edge Detector (RHD)
%
%
% Discrete Cosine Transform Edge Detector (DCT)
%
%

% Set ANN static bias value
biasVal = 1;

% Determine image size
imSz = size(inputImage);

edgeImg = zeros(imSz(1),imSz(2));

inVals = zeros(3,1);


for n = 3:imSz(1)-2     % Rows
    for m = 3:imSz(2)-2 % Columns


        curKernel = [inputImage(n-2,m-2), inputImage(n-2,m-1),
inputImage(n-2,m), inputImage(n-2,m+1), inputImage(n-2,m+2);...
                     inputImage(n-1,m-2), inputImage(n-1,m-1),
inputImage(n-1,m), inputImage(n-1,m+1), inputImage(n-1,m+2);...
                     inputImage(n,m-2), inputImage(n,m-1)  ,
inputImage(n,m)  , inputImage(n,m+1), inputImage(n,m+2);...
                     inputImage(n+1,m-2), inputImage(n+1,m-1),
inputImage(n+1,m), inputImage(n+1,m+1), inputImage(n+1,m+2);
                     inputImage(n+2,m-2), inputImage(n+2,m-1),
inputImage(n+2,m), inputImage(n+2,m+1), inputImage(n+2,m+2)];

        curKernel = curKernel./255;
        % NOTES: inputImage(row,column)

        % Determine RLV

        inVals(1) = var(var(curKernel));
        % Determine RHD


        inVals(2) = ratioHarrisED(curKernel,0.04);


        % Determine DCT
        tempKerMean = mean(mean(curKernel));
        tempDCT = dct(curKernel);

        tempDCT2 = tempDCT > 1.35 * tempKerMean;


        inVals(3) = nnz(tempDCT2);

        % Write image pixel

        tempSz = size(hidWeights);
        hidNeuronCount = tempSz(1);
        [hidLayerOut] =
BP_FF_BIAS(hidWeights,hidBiasW,biasVal,inVals,hidNeuronCount); % 5 Hid
Neurons
%         [hidLayerOut] = BP_FF(hidWeights,inVals,hidNeuronCount);

        tempSz = size(outWeights);
        outNeuronCount = tempSz(1);
        [ANNOut] =
BP_FF_BIAS(outWeights,outBiasW,biasVal,hidLayerOut,outNeuronCount); % 2
Outputs
%         [ANNOut] = BP_FF(outWeights, hidLayerOut, outNeuronCount);

%         if (ANNOut(1) >= 0.5) % Edge Detected
%             edgeImg(n,m) = 255;
%         elseif (ANNOut(2) < 0.5) % Homogeneous Region Detected
%             edgeImg(n,m) = 0;
%         else
%             edgeImg(n,m) = 0;
%         end

        %%%%%%%%%%%%%%%%%%%%%%%  Normal Output
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        if (ANNOut(1) >= 0 && ANNOut(2) < 0) % Edge Detected
            edgeImgNN = [1;-1];
            edgeImg(n,m) = 255;
%             edgeImg(n,m) = 0;
        elseif (ANNOut(2) >= 0 && ANNOut(1) < 0) % Homogeneous Region
Detected
            edgeImgNN = [-1;1];
            edgeImg(n,m) = 0;
%             edgeImg(n,m) = 255;
        elseif (ANNOut(1) > ANNOut(2)) % Edge Detected
            edgeImgNN = [1;-1];
            edgeImg(n,m) = 255;
%             edgeImg(n,m) = 0;
        elseif (ANNOut(2) > ANNOut(1)) % Homogeneous Region Detected
            edgeImgNN = [-1;1];
            edgeImg(n,m) = 0;
%             edgeImg(n,m) = 255;
        else
            edgeImgNN = [-1;1];
            edgeImg(n,m) = 127;

disp('****************************ERROR***********************')
        end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%



%         if (ANNOut(1) >= 0.5 && ANNOut(2) < 0.5) % Edge Detected
%             edgeImgNN = [1;-1];
%             edgeImg(n,m) = 255;
%         elseif (ANNOut(2) >= 0.5 && ANNOut(1) < 0.5) % Homogeneous
Region Detected
%             edgeImgNN = [-1;1];
%             edgeImg(n,m) = 0;
%         else
%             edgeImgNN = [-1;1];
%             edgeImg(n,m) = 0;
%
disp('****************************ERROR**********************')
%         end


    end
end



end
```

File: testEdgeScript_2.m

```
% Test Script for NN Edge Detector
% Must have network weights already in memory to work.


% inputImage =
imread('..\..\TestData\EdgeData\EdgeNoiseTest_IMG_1_V3.bmp','bmp');
inputChecker =
imread('..\..\TestData\EdgeData\EdgeNoiseTest_IMG_2.bmp','bmp');
% inputImage =
imread('..\..\TestData\EdgeData\EdgeNoiseTest_IMG_3.bmp','bmp');
inputImageE =
imread('..\..\TestData\EdgeData\ADD_NOISE_EdgeNoiseTest_IMG_1.bmp','bmp
');
inputImageH =
imread('..\..\TestData\EdgeData\ADD_NOISE_EdgeNoiseTest_IMG_1_HARD.bmp'
,'bmp');

inputREAL   = imread('..\..\TestData\EdgeData\hb15282.tif','tiff');
% inputREAL   =
imread('..\..\TestData\MSTAR\45_DEG\ZSU_23_4\HB16389.jpg','jpg');
inputREAL2  = imrotate(inputREAL, 90);
% inputREAL2  =
imread('..\..\TestData\ObjectReconstruction\hb15282_Rotated.tif','tiff'
);
inputREAL3  = imrotate(inputREAL, 180);
inputREAL4  = imrotate(inputREAL, 270);

% figure
% imshow(inputImage)

% [checkerImgOut]    = NN_EdgeDetector5x5(double(inputChecker),
outWeights, outBiasW, hidWeights, hidBiasW);
% [outputEdgeImageE] = NN_EdgeDetector5x5(double(inputImageE),
outWeights, outBiasW, hidWeights, hidBiasW);
% [outputEdgeImageH] = NN_EdgeDetector5x5(double(inputImageH),
outWeights, outBiasW, hidWeights, hidBiasW);
[edgeREAL] = NN_EdgeDetector5x5(double(inputREAL), outWeights,
outBiasW, hidWeights, hidBiasW);
[edgeREAL2] = NN_EdgeDetector5x5(double(inputREAL2), outWeights,
outBiasW, hidWeights, hidBiasW);
[edgeREAL3] = NN_EdgeDetector5x5(double(inputREAL3), outWeights,
outBiasW, hidWeights, hidBiasW);
[edgeREAL4] = NN_EdgeDetector5x5(double(inputREAL4), outWeights,
outBiasW, hidWeights, hidBiasW);

% Rotate all of the edges back and sum them together

temp2 = imrotate(edgeREAL2,-90);
temp3 = imrotate(edgeREAL3,-180);
temp4 = imrotate(edgeREAL4,-270);

finalEdge = edgeREAL + temp2 + temp3 + temp4;

exampleCanny  = edge(double(inputImageE), 'canny');
exampleCanny2 = edge(double(inputREAL));

%% Output Data
```

```
% figure
% imshow(checkerImgOut)
% title('Checkerboard Output')

% figure
% subplot 131
% imshow(exampleCanny)
% title('Canny Edge Detector on "Easy" Image')
% subplot 132
% imshow(outputEdgeImageE)
% title('ANN Edge Detector on "Easy" Image')
% subplot 133
% imshow(outputEdgeImageH)
% title('ANN Edge Detector on "Hard" Image')


figure
subplot 121
imshow(inputREAL)
title('MSTAR Image')
subplot 122
imshow(edgeREAL)
title('ANN Edge Detector')

figure
subplot 121
imshow(inputREAL2)
title('MSTAR Image')
subplot 122
imshow(edgeREAL2)
title('ANN Edge Detector')

figure
subplot 121
imshow(inputREAL3)
title('MSTAR Image')
subplot 122
imshow(edgeREAL3)
title('ANN Edge Detector')

figure
subplot 121
imshow(inputREAL4)
title('MSTAR Image')
subplot 122
imshow(edgeREAL4)
title('ANN Edge Detector')

figure
subplot 131
imshow(inputREAL)
title('Original MSTAR Image')
subplot 132
imshow(finalEdge)
title('ANN Edge Detector')
subplot 133
imshow(exampleCanny2)
title('Canny Edge Detector')
```

File: testEdgeScript_3.m

```
% Test Script for NN Edge Detector
% Must have network weights already in memory to work.


% inputImage =
imread('..\..\TestData\EdgeData\EdgeNoiseTest_IMG_1_V3.bmp','bmp');
inputChecker =
imread('..\..\TestData\EdgeData\EdgeNoiseTest_IMG_2.bmp','bmp');
% inputImage =
imread('..\..\TestData\EdgeData\EdgeNoiseTest_IMG_3.bmp','bmp');
inputImageE =
imread('..\..\TestData\EdgeData\ADD_NOISE_EdgeNoiseTest_IMG_1.bmp','bmp
');
inputImageH =
imread('..\..\TestData\EdgeData\ADD_NOISE_EdgeNoiseTest_IMG_1_HARD.bmp'
,'bmp');

% Rotate by 90
% % inputREAL   = imread('..\..\TestData\EdgeData\hb15282.tif','tiff');
% inputREAL   =
imread('..\..\TestData\MSTAR\45_DEG\ZSU_23_4\HB16389.jpg','jpg');
% inputREAL2  = imrotate(inputREAL, 90);
% inputREAL3  = imrotate(inputREAL, 180);
% inputREAL4  = imrotate(inputREAL, 270);

% Rotate by 45 degrees
inputREAL   =
imread('..\..\TestData\MSTAR\45_DEG\ZSU_23_4\HB16389.jpg','jpg');
inputREAL2  = imrotate(inputREAL, 45, 'crop');
inputREAL3  = imrotate(inputREAL, 90, 'crop');
inputREAL4  = imrotate(inputREAL, 135, 'crop');
inputREAL5  = imrotate(inputREAL, 180, 'crop');
inputREAL6  = imrotate(inputREAL, 225, 'crop');
inputREAL7  = imrotate(inputREAL, 270, 'crop');
inputREAL8  = imrotate(inputREAL, 315, 'crop');

% figure
% imshow(inputImage)

% [checkerImgOut]    = NN_EdgeDetector5x5(double(inputChecker),
outWeights, outBiasW, hidWeights, hidBiasW);
% [outputEdgeImageE] = NN_EdgeDetector5x5(double(inputImageE),
outWeights, outBiasW, hidWeights, hidBiasW);
% [outputEdgeImageH] = NN_EdgeDetector5x5(double(inputImageH),
outWeights, outBiasW, hidWeights, hidBiasW);
[edgeREAL] = NN_EdgeDetector5x5(double(inputREAL), outWeights,
outBiasW, hidWeights, hidBiasW);
[edgeREAL2] = NN_EdgeDetector5x5(double(inputREAL2), outWeights,
outBiasW, hidWeights, hidBiasW);
[edgeREAL3] = NN_EdgeDetector5x5(double(inputREAL3), outWeights,
outBiasW, hidWeights, hidBiasW);
[edgeREAL4] = NN_EdgeDetector5x5(double(inputREAL4), outWeights,
outBiasW, hidWeights, hidBiasW);
[edgeREAL5] = NN_EdgeDetector5x5(double(inputREAL5), outWeights,
outBiasW, hidWeights, hidBiasW);
[edgeREAL6] = NN_EdgeDetector5x5(double(inputREAL6), outWeights,
outBiasW, hidWeights, hidBiasW);
```

```
[edgeREAL7] = NN_EdgeDetector5x5(double(inputREAL7), outWeights,
outBiasW, hidWeights, hidBiasW);
[edgeREAL8] = NN_EdgeDetector5x5(double(inputREAL8), outWeights,
outBiasW, hidWeights, hidBiasW);

% Rotate all of the edges back and sum them together
% 90 degrees
% temp2 = imrotate(edgeREAL2,-90);
% temp3 = imrotate(edgeREAL3,-180);
% temp4 = imrotate(edgeREAL4,-270);

% 45 degrees
temp2 = imrotate(edgeREAL2,-45, 'crop');
temp3 = imrotate(edgeREAL3,-90, 'crop');
temp4 = imrotate(edgeREAL4,-135, 'crop');
temp5 = imrotate(edgeREAL5,-180, 'crop');
temp6 = imrotate(edgeREAL6,-225, 'crop');
temp7 = imrotate(edgeREAL7,-270, 'crop');
temp8 = imrotate(edgeREAL8,-315, 'crop');

finalEdge = edgeREAL + temp2 + temp3 + temp4 + temp5 + temp6 + temp7 +
temp8;

exampleCanny  = edge(double(inputImageE), 'canny');
exampleCanny2 = edge(double(inputREAL));

%% Output Data

% figure
% imshow(checkerImgOut)
% title('Checkerboard Output')

% figure
% subplot 131
% imshow(exampleCanny)
% title('Canny Edge Detector on "Easy" Image')
% subplot 132
% imshow(outputEdgeImageE)
% title('ANN Edge Detector on "Easy" Image')
% subplot 133
% imshow(outputEdgeImageH)
% title('ANN Edge Detector on "Hard" Image')


figure
subplot 121
imshow(inputREAL)
title('MSTAR Image')
subplot 122
imshow(edgeREAL)
title('ANN Edges')

figure
subplot 121
imshow(inputREAL2)
title('MSTAR Image')
subplot 122
imshow(edgeREAL2)
title('ANN Edges')

figure
subplot 121
imshow(inputREAL3)
title('MSTAR Image')
```

```
subplot 122
imshow(edgeREAL3)
title('ANN Edges')

figure
subplot 121
imshow(inputREAL4)
title('MSTAR Image')
subplot 122
imshow(edgeREAL4)
title('ANN Edges')

figure
subplot 131
imshow(inputREAL)
title('Original MSTAR Image')
subplot 132
imshow(finalEdge)
title('ANN Edge Detector')
subplot 133
imshow(exampleCanny2)
title('Canny Edge Detector')
```

File: hopfieldRUN_V3.m

```
function [probeOutput] = hopfieldRUN_V3(weights,probeInput,testCol)
% Function takes a weight matrix that corresponds to a Hopfield
network,
% and take a input source (NxM) and will randomly select a column of
the

tempSz = size(probeInput);

% Dim of a single fundamental memory
N = tempSz(1);

% Form proper sized identity matrix
ident = eye(N);


probeOutput = zeros(tempSz(1),tempSz(2));

% Calcluate the output of neuron j
neuronCalcOut = sign(weights(testCol,:) * probeInput);


if (neuronCalcOut == 0)
    neuronCalcOut = probeInput(testCol);
    disp('Zero Found')
end

probeOutput = probeInput;

probeOutput(testCol) = neuronCalcOut;


end
```

File: hopfieldWeights.m

```
function [weightMatrix] = hopfieldWeights(trainData)
% Takes in some list of binary words that discribe the fundamental
% memeories (quantity M) for the Hopfield model and produces the N-by-N
weight matrix,
% where N is the dimension of the 1-by-N fundamental memory vectors.
%
%

tempSz = size(trainData);

% Number of memories to train
M = tempSz(3);

% Dim of a single fundamental memory
N = tempSz(1);

% Form proper sized identity matrix
ident = eye(N);

weightMatrix = zeros(N,N);

for n = 1:M
%     size(weightMatrix)
    weightMatrix = weightMatrix +
trainData(:,:,n)*transpose(trainData(:,:,n));

%     weightMatrix = weightMatrix +
trainData(:,:,n)*transpose(trainData(:,:,n)) - M*ident;
end



weightMatrix = (1/N) * (weightMatrix - M*ident);
% weightMatrix = (1/N) * (weightMatrix);



end
```

File: SCRIPT_hopfieldTest2.m

```
% Test script that calls the hopfield functions

close all
clear
clc

%% Data Inport

% Import Objects to learn.
% %
% learnDatImages(:,:,1) =
edge(im2bw(imread('..\..\TestData\ObjectReconstruction\Circle_V1.bmp'),
0.9),'canny');
% learnDatImages(:,:,2) =
edge(im2bw(imread('..\..\TestData\ObjectReconstruction\SQUARE_V1.bmp'),
0.9),'canny');
% learnDatImages(:,:,3) =
edge(im2bw(imread('..\..\TestData\ObjectReconstruction\TRIANGLE_V1.bmp'
), 0.9),'canny');
% learnDatImages(:,:,4) =
edge(im2bw(imread('..\..\TestData\ObjectReconstruction\RECTANGLE_V1.bmp
'), 0.9),'canny');

% learnDatImages(:,:,1) =
im2bw(imread('..\..\TestData\ObjectReconstruction\Circle_V1.bmp'),
0.9);
learnDatImages(:,:,2) =
im2bw(imread('..\..\TestData\ObjectReconstruction\SQUARE_V1.bmp'),
0.9);
% learnDatImages(:,:,3) =
im2bw(imread('..\..\TestData\ObjectReconstruction\TRIANGLE_V1.bmp'),
0.9);
learnDatImages(:,:,1) =
im2bw(imread('..\..\TestData\ObjectReconstruction\RECTANGLE_V1.bmp'),
0.9);

% learnDatImages(:,:,1) =
im2bw(imread('..\..\TestData\ObjectReconstruction\Circle_V1.bmp'),
0.9);
% % learnDatImages(:,:,2) =
im2bw(imread('..\..\TestData\ObjectReconstruction\SQUARE_V1.bmp'),
0.9);
% learnDatImages(:,:,2) =
im2bw(imread('..\..\TestData\ObjectReconstruction\TRIANGLE_V1.bmp'),
0.9);
% learnDatImages(:,:,3) =
im2bw(imread('..\..\TestData\ObjectReconstruction\RECTANGLE_V1.bmp'),
0.9);

% learnDatImages(:,:,1) =
rgb2gray(imread('..\..\TestData\ObjectReconstruction\Circle_V1.bmp'));
% learnDatImages(:,:,2) =
rgb2gray(imread('..\..\TestData\ObjectReconstruction\SQUARE_V1.bmp'));
% learnDatImages(:,:,3) =
rgb2gray(imread('..\..\TestData\ObjectReconstruction\TRIANGLE_V1.bmp'))
;
```

```
% learnDatImages(:,:,4) =
rgb2gray(imread('..\..\TestData\ObjectReconstruction\RECTANGLE_V1.bmp')
);

% % Data Input Verification % %

figure
subplot 221
imshow(learnDatImages(:,:,1))
title('Training Data')
subplot 222
imshow(learnDatImages(:,:,2))
% subplot 223
% imshow(learnDatImages(:,:,3))
% subplot 224
% imshow(learnDatImages(:,:,4))

% % End Data Verification % %

% Import Incomplete versions of data for testing.

% incDatImages(:,:,1) =
edge(im2bw(imread('..\..\TestData\ObjectReconstruction\INC_CIRCLE_1_V2.
bmp'), 0.9),'canny');
% incDatImages(:,:,2) =
edge(im2bw(imread('..\..\TestData\ObjectReconstruction\INC_SQUARE_1_V2.
bmp'), 0.9),'canny');
% incDatImages(:,:,3) =
edge(im2bw(imread('..\..\TestData\ObjectReconstruction\INC_TRIANGLE_1_V
1.bmp'), 0.9),'canny');
% incDatImages(:,:,4) =
edge(im2bw(imread('..\..\TestData\ObjectReconstruction\INC_RECTANGLE_1_
V2.bmp'), 0.9),'canny');
%
% incDatImages(:,:,1) =
im2bw(imread('..\..\TestData\ObjectReconstruction\INC_CIRCLE_1_V2.bmp')
, 0.9);
% incDatImages(:,:,2) =
im2bw(imread('..\..\TestData\ObjectReconstruction\INC_SQUARE_1_V2.bmp')
, 0.9);
% incDatImages(:,:,3) =
im2bw(imread('..\..\TestData\ObjectReconstruction\INC_TRIANGLE_1_V1.bmp
'), 0.9);
% incDatImages(:,:,4) =
im2bw(imread('..\..\TestData\ObjectReconstruction\INC_RECTANGLE_1_V2.bm
p'), 0.9);
%
% % incDatImages(:,:,1) =
im2bw(imread('..\..\TestData\ObjectReconstruction\INC_CIRCLE_1_V2.bmp')
, 0.9);
% incDatImages(:,:,1) =
im2bw(imread('..\..\TestData\ObjectReconstruction\INC_SQUARE_1_V2.bmp')
, 0.9);
% incDatImages(:,:,2) =
im2bw(imread('..\..\TestData\ObjectReconstruction\INC_TRIANGLE_1_V1.bmp
'), 0.9);
% incDatImages(:,:,3) =
im2bw(imread('..\..\TestData\ObjectReconstruction\INC_RECTANGLE_1_V2.bm
p'), 0.9);

% incDatImages(:,:,1) =
im2bw(imnoise(imread('..\..\TestData\ObjectReconstruction\INC_CIRCLE_1_
V2.bmp'), 'speckle'),0.5);
```

```
% incDatImages(:,:,2) =
im2bw(imnoise(imread('..\..\TestData\ObjectReconstruction\INC_SQUARE_3_
V1.bmp'), 'speckle'),0.5);
incDatImages(:,:,2) =
im2bw(imnoise(imread('..\..\TestData\ObjectReconstruction\INC_SQUARE_3_
V1.bmp'), 'speckle')+1,0.9);
% incDatImages(:,:,2) =
im2bw(imnoise(imread('..\..\TestData\ObjectReconstruction\INC_SQUARE_1_
V2.bmp'), 'speckle'),0.5);
% incDatImages(:,:,3) =
im2bw(imread('..\..\TestData\ObjectReconstruction\INC_TRIANGLE_1_V1.bmp
'), 0.9);
incDatImages(:,:,1) =
im2bw(imnoise(imread('..\..\TestData\ObjectReconstruction\INC_RECTANGLE
_1_V2.bmp'), 'speckle'), 0.9);

% incDatImages(:,:,1) =
rgb2gray(imread('..\..\TestData\ObjectReconstruction\INC_CIRCLE_1_V1.bm
p'));
% incDatImages(:,:,2) =
rgb2gray(imread('..\..\TestData\ObjectReconstruction\INC_SQUARE_1_V1.bm
p'));
% incDatImages(:,:,3) =
rgb2gray(imread('..\..\TestData\ObjectReconstruction\INC_TRIANGLE_1_V1.
bmp'));
% incDatImages(:,:,4) =
rgb2gray(imread('..\..\TestData\ObjectReconstruction\INC_RECTANGLE_1_V2
.bmp'));

% % Data Input Verification % %

figure
subplot 221
imshow(incDatImages(:,:,1))
title('Test Data')
subplot 222
imshow(incDatImages(:,:,2))
subplot 223
% imshow(incDatImages(:,:,3))
% subplot 224
% % imshow(incDatImages(:,:,4))

% % End Data Verification % %


imSz = size(learnDatImages);
lengthImSz = imSz(1)*imSz(2);

trainSetSize = imSz(3);

trainData = zeros(lengthImSz, 1, trainSetSize);
incData = zeros(lengthImSz, 1, trainSetSize);

for n = 1:trainSetSize
    trainData(:,:,n) = double(reshape(learnDatImages(:,:,n),
lengthImSz, 1));
end

tempVal = find(trainData == 0);
trainData(tempVal) = -1;

for n = 1:trainSetSize
    incData(:,:,n) = double(reshape(incDatImages(:,:,n), lengthImSz,
1));
```

```
end

% Convert 0 values to -1

tempVal = find(incData == 0);
incData(tempVal) = -1;


%% Hopfield Weight Calcluation


[weightMatrix] = hopfieldWeights(trainData);



%% Output with Incomplete Data

outData = zeros(imSz(1), imSz(2), trainSetSize);

iterationCount = 3000;

% testOut = trainData;
testOut = incData;
continueIteration = 1;

iterationTicker = 0;

% while(continueIteration)
%     iterationTicker = iterationTicker + 1;
%
%     for n = 1:trainSetSize
%         outputLast(:,:,n) = testOut(:,:,n);
%
%         testOut(:,:,n) = hopfieldRUN(weightMatrix, testOut(:,:,n));
%         testOut(:,:,n) = hopfieldRUN(weightMatrix, testOut(:,:,n));
%         testOut(:,:,n) = hopfieldRUN(weightMatrix, testOut(:,:,n));
%         testOut(:,:,n) = hopfieldRUN(weightMatrix, testOut(:,:,n));
%         testOut(:,:,n) = hopfieldRUN(weightMatrix, testOut(:,:,n));
%         testOut(:,:,n) = hopfieldRUN(weightMatrix, testOut(:,:,n));
%         testOut(:,:,n) = hopfieldRUN(weightMatrix, testOut(:,:,n));
%         testOut(:,:,n) = hopfieldRUN(weightMatrix, testOut(:,:,n));
%         testOut(:,:,n) = hopfieldRUN(weightMatrix, testOut(:,:,n));
%
%         diffImg(:,:,n) = outputLast(:,:,n) - testOut(:,:,n);
%     end
%
%     cont = max(max(abs(diffImg)));
%
%     if (cont > 0)
%         continueIteration = 1;
%     else
%         continueIteration = 0;
%     end
%
%
% end
% disp([num2str(iterationTicker), ' iterations to converge.'])


for iterations = 1:iterationCount

    for n = 1:trainSetSize

        testOut(:,:,n) = hopfieldRUN(weightMatrix, testOut(:,:,n));
    end

end


% Convert to readable image

for n = 1:trainSetSize

    outData(:,:,n) = reshape(testOut(:,:,n), imSz(1), imSz(2)).*255;

end

clear tempVal
tempVal = find(outData < 0);
outData(tempVal) = 0;

figure
subplot 221
imshow(outData(:,:,1))
title('Hopfield Output')
subplot 222
imshow(outData(:,:,2))
subplot 223
% imshow(outData(:,:,3))
% subplot 224
% % imshow(outData(:,:,4))


% end


%% Plot full set of data

temp2 =
imread('..\..\TestData\ObjectReconstruction\INC_SQUARE_3_V1.bmp');
% incDatImages(:,:,2) =
im2bw(imnoise(imread('..\..\TestData\ObjectReconstruction\INC_SQUARE_1_
V2.bmp'), 'speckle'),0.5);
% incDatImages(:,:,3) =
im2bw(imread('..\..\TestData\ObjectReconstruction\INC_TRIANGLE_1_V1.bmp
'), 0.9);
temp1 =
imread('..\..\TestData\ObjectReconstruction\INC_RECTANGLE_1_V2.bmp');


figure

subplot 421
imshow(learnDatImages(:,:,1))
title('Input Image 1')


subplot 422
imshow(learnDatImages(:,:,2))
title('Input Image 2')

subplot 423
imshow(temp1)
title('Incomplete Shape 1')

subplot 424
imshow(temp2)
```

```
title('Incomplete Shape 2')

subplot 425
imshow(incDatImages(:,:,1))
title('Noisy Input Image 1')

subplot 426
imshow(incDatImages(:,:,2))
title('Noisy Input Image 2')

subplot 427
imshow(outData(:,:,1))
title('Hopfield Output 1')
subplot 428
imshow(outData(:,:,2))
title('Hopfield Output 2')
```

File: SCRIPT_HopfieldMSTAR_V3.m

```
% This script uses the hopfield model, trained with basic geometric
shapes,
% to determine the best shape fit of a pre-processed SAR image.
%
%


close all
clear
clc

% Runtime Check
ticTimer = tic;


%% Import MSTAR images to process

disp('Importing SAR Data...')

sarImg1 =
imrotate(imread('..\..\testData\MSTAR\45_DEG\BRDM_2\HB15696.JPG','jpeg'
),90);
sarImg2 =
imread('..\..\testData\MSTAR\45_DEG\BRDM_2\HB15793.JPG','jpeg');

sarImg3 =
imrotate(imread('..\..\testData\MSTAR\45_DEG\ZSU_23_4\HB16390.JPG','jpe
g'),90);
sarImg4 =
imread('..\..\testData\MSTAR\45_DEG\ZSU_23_4\HB16401.JPG','jpeg');

% Display test Data
scrsz = get(0, 'ScreenSize');
figure_num = 1;
fig_name = ['Figure ', num2str(figure_num)];
figure('Name', fig_name, 'NumberTitle', 'off', 'Position',...
    [scrsz(3)/8 scrsz(4)/8 800 600])
subplot 221
imshow(sarImg1)
title('BRDM 2 Perspective 1')

subplot 222
imshow(sarImg2)
title('BRDM 2 Perspective 2')

subplot 223
imshow(sarImg3)
title('ZSU 23 4 Perspective 1')

subplot 224
imshow(sarImg4)
title('ZSU 23 4 Perspective 2')

%% Filter images

sarImg1filt = im2bw(sarImg1,0.6);
sarImg2filt = im2bw(sarImg2,0.6);
```

```
sarImg3filt = im2bw(sarImg3,0.6);
sarImg4filt = im2bw(sarImg4,0.6);


sarImg1filt = medfilt2(sarImg1filt);
sarImg2filt = medfilt2(sarImg2filt);
sarImg3filt = medfilt2(sarImg3filt);
sarImg4filt = medfilt2(sarImg4filt);

%
testImSz(1) = 80;
testImSz(2) = 80;
%
% testImSz(1) = 50;
% testImSz(2) = 50;
%
% testImSz(1) = 20;
% testImSz(2) = 20;

inlength = testImSz(1)*testImSz(2);

sarImg1filt = imresize(sarImg1filt,[testImSz(1) testImSz(2)]);
sarImg2filt = imresize(sarImg2filt,[testImSz(1) testImSz(2)]);
sarImg3filt = imresize(sarImg3filt,[testImSz(1) testImSz(2)]);
sarImg4filt = imresize(sarImg4filt,[testImSz(1) testImSz(2)]);

sarImg5filt = imresize(sarImg1, [testImSz(1) testImSz(2)]);

% Display test Data
scrsz = get(0, 'ScreenSize');
figure_num = 2;
fig_name = ['Figure ', num2str(figure_num)];
figure('Name', fig_name, 'NumberTitle', 'off', 'Position',...
    [scrsz(3)/8 scrsz(4)/8 800 600])
subplot 221
imshow(sarImg1filt)
title('BRDM 2 Perspective 1')

subplot 222
imshow(sarImg2filt)
title('BRDM 2 Perspective 2')

subplot 223
imshow(sarImg3filt)
title('ZSU 23 4 Perspective 1')

subplot 224
imshow(sarImg4filt)
title('ZSU 23 4 Perspective 2')

% testImSz = size(sarImg1);

% testImSz(1) = 128;
% testImSz(2) = 128;


% sarImg1filt = imresize(sarImg1filt,[testImSz(1) testImSz(2)]);
% sarImg2filt = imresize(sarImg2filt,[testImSz(1) testImSz(2)]);
% sarImg3filt = imresize(sarImg3filt,[testImSz(1) testImSz(2)]);
% sarImg4filt = imresize(sarImg4filt,[testImSz(1) testImSz(2)]);

%% Hopfield Network Learning Period

% Import and generate training data
```

```
disp('>> Importing Training Data...')
tempTimer = num2str(toc(ticTimer));
disp(['Time Elapsed: ', tempTimer, 's'])

% Import Learning Data

for n = 1:17
    learnEdge(:,:,n) =
double(imcomplement(im2bw(imresize(imread(['..\..\testData\ObjectRecons
truction\ObjSet2\edge_square_Sz', num2str(n)
,'.tif'],'tiff'),[testImSz(1) testImSz(2)]),0.9)));
end

% for n = 1:17
%     learnFilled(:,:,n) =
double(imcomplement(im2bw(imresize(imread(['..\..\testData\ObjectRecons
truction\ObjSet2\filled_square_Sz', num2str(n)
,'.tif'],'tiff'),[testImSz(1) testImSz(2)]),0.9)));
% end

for n = 1:17
    learnFilled(:,:,n) =
double(imcomplement(im2bw(imresize(imread(['..\..\testData\ObjectRecons
truction\ObjSet2\filled_rectW2_Sz', num2str(n)
,'.tif'],'tiff'),[testImSz(1) testImSz(2)]),0.9)));
end

for n = 1:17
    learnFilled(:,:,n+17) =
double(imcomplement(im2bw(imresize(imread(['..\..\testData\ObjectRecons
truction\ObjSet2\filled_rectW4_Sz', num2str(n)
,'.tif'],'tiff'),[testImSz(1) testImSz(2)]),0.9)));
end

for n = 1:17
    learnFilled(:,:,n+17+17) =
double(imcomplement(im2bw(imresize(imread(['..\..\testData\ObjectRecons
truction\ObjSet2\filled_rectW6_Sz', num2str(n)
,'.tif'],'tiff'),[testImSz(1) testImSz(2)]),0.9)));
end

for n = 1:17
    learnFilled(:,:,n+17+17+17) =
double(imcomplement(im2bw(imresize(imread(['..\..\testData\ObjectRecons
truction\ObjSet2\filled_rectW8_Sz', num2str(n)
,'.tif'],'tiff'),[testImSz(1) testImSz(2)]),0.9)));
end

for n = 1:17
    learnFilled(:,:,n+17+17+17+17) =
double(imcomplement(im2bw(imresize(imread(['..\..\testData\ObjectRecons
truction\ObjSet2\filled_rectW10_Sz', num2str(n)
,'.tif'],'tiff'),[testImSz(1) testImSz(2)]),0.9)));
end

for n = 1:17
    learnFilled(:,:,n+17+17+17+17+17) =
double(imcomplement(im2bw(imresize(imread(['..\..\testData\ObjectRecons
truction\ObjSet2\filled_rectW12_Sz', num2str(n)
,'.tif'],'tiff'),[testImSz(1) testImSz(2)]),0.9)));
end

for n = 1:17
```

```
    learnFilled(:,:,n+17+17+17+17+17+17) =
double(imcomplement(im2bw(imresize(imread(['..\..\testData\ObjectRecons
truction\ObjSet2\filled_rectW14_Sz', num2str(n)
,'.tif'],'tiff'),[testImSz(1) testImSz(2)]),0.9)));
end

for n = 1:17
    learnFilled(:,:,n+17+17+17+17+17+17+17) =
double(imcomplement(im2bw(imresize(imread(['..\..\testData\ObjectRecons
truction\ObjSet2\filled_rectW16_Sz', num2str(n)
,'.tif'],'tiff'),[testImSz(1) testImSz(2)]),0.9)));
end


% Add Rotated Images

% for n = 1:17
%     learnFilled(:,:,n+17+17+17+17) = imrotate(learnFilled(:,:,n+17),
90);
% end
%
% for n = 1:17
%     learnFilled(:,:,n+17+17+17+17+17) =
imrotate(learnFilled(:,:,n+17+17), 90);
% end
%
% for n = 1:17
%     learnFilled(:,:,n+17+17+17+17+17+17) =
imrotate(learnFilled(:,:,n+17+17+17), 90);
% end

% for n = 1:17
%     learnFilled(:,:,n+17+17) = imrotate(learnFilled(:,:,n+17), 90);
% end

% learnEdge(:,:,1) =
double(imcomplement(im2bw(imresize(imread('..\..\testData\ObjectReconst
ruction\testRect1_Edge.tif','tiff'),[testImSz(1) testImSz(2)]),0.9)));
% learnEdge(:,:,2) =
double(imcomplement(im2bw(imresize(imread('..\..\testData\ObjectReconst
ruction\testRect2_Edge.tif','tiff'),[testImSz(1) testImSz(2)]),0.9)));
% learnEdge(:,:,3) =
double(imcomplement(im2bw(imresize(imread('..\..\testData\ObjectReconst
ruction\testRect3_Edge.tif','tiff'),[testImSz(1) testImSz(2)]),0.9)));
% learnEdge(:,:,4) =
double(imcomplement(im2bw(imresize(imread('..\..\testData\ObjectReconst
ruction\testRect4_Edge.tif','tiff'),[testImSz(1) testImSz(2)]),0.9)));

%
% learnEdge(:,:,1) =
double(imcomplement(im2bw(imresize(imread('..\..\testData\ObjectReconst
ruction\testRect1_Edge_Fill.tif','tiff'),[testImSz(1)
testImSz(2)]),0.9)));
% learnEdge(:,:,2) =
double(imcomplement(im2bw(imresize(imread('..\..\testData\ObjectReconst
ruction\testRect2_Edge_Fill.tif','tiff'),[testImSz(1)
testImSz(2)]),0.9)));
% learnEdge(:,:,3) =
double(imcomplement(im2bw(imresize(imread('..\..\testData\ObjectReconst
ruction\testRect3_Edge_Fill.tif','tiff'),[testImSz(1)
testImSz(2)]),0.9)));
% learnEdge(:,:,4) =
double(imcomplement(im2bw(imresize(imread('..\..\testData\ObjectReconst
ruction\testRect4_Edge_Fill.tif','tiff'),[testImSz(1)
testImSz(2)]),0.9)));
%
% learnEdge(:,:,5) = imrotate(learnEdge(:,:,1),90);
% learnEdge(:,:,6) = imrotate(learnEdge(:,:,2),90);
% % learnEdge(:,:,8) = imrotate(learnEdge(:,:,4),90);
% learnEdge(:,:,7) = imrotate(learnEdge(:,:,3),90);


trainSz = size(learnFilled);

%
% learnEdge(find(learnEdge == 0)) = -1;
% learnFilled(find(learnFilled == 0)) = -1;
%
% figure
% subplot 241
% imshow(learnEdge(:,:,1))
% subplot 242
% imshow(learnEdge(:,:,2))
% subplot 243
% imshow(learnEdge(:,:,3))
% subplot 244
% imshow(learnEdge(:,:,4))
% subplot 245
% imshow(learnEdge(:,:,5))
% subplot 246
% imshow(learnEdge(:,:,6))
% subplot 247
% imshow(learnEdge(:,:,7))
% subplot 248
% imshow(learnEdge(:,:,8))

figure
subplot 241
imshow(learnFilled(:,:,1))
subplot 242
imshow(learnFilled(:,:,2))
subplot 243
imshow(learnFilled(:,:,3))
subplot 244
imshow(learnFilled(:,:,4))
subplot 245
imshow(learnFilled(:,:,5))
subplot 246
imshow(learnFilled(:,:,6))
subplot 247
imshow(learnFilled(:,:,7))
subplot 248
imshow(learnFilled(:,:,8))

% Reshape for linear Hopfield Network

% for n = 1:trainSz(3)
%     trainIn(:,:,n) = reshape(learnEdge(:,:,n), inlength, 1);
% end

disp('>> Reformat Input...')
tempTimer = num2str(toc(ticTimer));
disp(['Time Elapsed: ', tempTimer, 's'])

for n = 1:trainSz(3)
    trainIn(:,:,n) = reshape(learnFilled(:,:,n), inlength, 1);
end
```

```
trainIn(find(trainIn == 0)) = -1;

netIn1 = double(reshape(sarImg1filt, inlength, 1));
netIn2 = double(reshape(sarImg2filt, inlength, 1));
netIn3 = double(reshape(sarImg3filt, inlength, 1));
netIn4 = double(reshape(sarImg4filt, inlength, 1));

netIn5 = double(reshape(sarImg5filt, inlength, 1))./255;

netIn1(find(netIn1 == 0)) = -1;
netIn2(find(netIn2 == 0)) = -1;
netIn3(find(netIn3 == 0)) = -1;
netIn4(find(netIn4 == 0)) = -1;

% tester = reshape(netIn1,testImSz(1),testImSz(2));

disp('>> Network Training...')
tempTimer = num2str(toc(ticTimer));
disp(['Time Elapsed: ', tempTimer, 's'])
wMatrix = hopfieldWeights(trainIn);

iterationCount = 1;

% netOut1 = netIn;

%%

disp('>> Running Network')
tempTimer = num2str(toc(ticTimer));
disp(['Time Elapsed: ', tempTimer, 's'])

% Set up Movie Capture
scrsz = get(0, 'ScreenSize');
figure_num = 100;
fig_name = ['Figure ', num2str(figure_num)];
figure('Name', fig_name, 'NumberTitle', 'off', 'Position',...
    [scrsz(3)/8 scrsz(4)/8 800 600])

imDispreshape1 = reshape(netIn1,testImSz(1),testImSz(2));
imDispreshape2 = reshape(netIn2,testImSz(1),testImSz(2));
imDispreshape3 = reshape(netIn3,testImSz(1),testImSz(2));
imDispreshape4 = reshape(netIn4,testImSz(1),testImSz(2));
subplot 221
imshow(imDispreshape1);
title('DHN BRDM 2 Perspective 1')
subplot 222
imshow(imDispreshape2);
title('DHN BRDM 2 Perspective 2')
subplot 223
imshow(imDispreshape3);
title('DHN ZSU 23 4 Perspective 1')
subplot 224
imshow(imDispreshape4);
title('DHN ZSU 23 4 Perspective 2')

set(gca, 'NextPlot', 'replaceChildren');

% Movie writer parameters
writerObj = VideoWriter('Hopfield_Test_1');
open(writerObj)
% writeVideo(writerObj,hopIterFrames(:));
% hopIterFrames(iterationCount) = struct('cdata',[],'colormap',[]);

% h = waitbar(0, 'Working...');
disp('Working...')
for n = 1:iterationCount
%     waitbar(n/iterationCount,h)
    tempLoopSz = size(netIn1);
    tempRunList = randperm(tempLoopSz(1));

    for m = 1:tempLoopSz(1)
        netIn1 = hopfieldRUN_V3(wMatrix, netIn1, tempRunList(m));
        netIn2 = hopfieldRUN_V3(wMatrix, netIn2, tempRunList(m));
        netIn3 = hopfieldRUN_V3(wMatrix, netIn3, tempRunList(m));
        netIn4 = hopfieldRUN_V3(wMatrix, netIn4, tempRunList(m));

        imDispreshape1 = reshape(netIn1,testImSz(1),testImSz(2));
        imDispreshape2 = reshape(netIn2,testImSz(1),testImSz(2));
        imDispreshape3 = reshape(netIn3,testImSz(1),testImSz(2));
        imDispreshape4 = reshape(netIn4,testImSz(1),testImSz(2));

        subplot 221
        imshow(imDispreshape1);
        subplot 222
        imshow(imDispreshape2);
        subplot 223
        imshow(imDispreshape3);
        subplot 224
        imshow(imDispreshape4);


        hopIterFrames(:,n) = getframe;
        writeVideo(writerObj,hopIterFrames(n));
    end

%     imDispreshape1 = reshape(netIn1,testImSz(1),testImSz(2));
%     imshow(imDispreshape1);
%     hopIterFrames(n) = getframe;
    disp(['Iteration #', num2str(n)])
end

close(writerObj)

% fprintf(' \n')
% close(h)

% for n = 1:10
%     netIn1 = wMatrix*netIn1;
%     netIn2 = wMatrix*netIn2;
%     netIn3 = wMatrix*netIn3;
%     netIn4 = wMatrix*netIn4;
% end

%%

% netOut1 = wMatrix*netIn1;
% netOut1 = wMatrix*netOut1;
% netOut1 = wMatrix*netOut1;
% netOut1 = wMatrix*netOut1;
% netOut1 = wMatrix*netOut1;
netOut1(find(netIn1 > 0))  = 255;
netOut1(find(netIn1 <= 0)) = 0;

netOut2 = wMatrix*netIn2;
% test = netOut2.*255;
```

```
netOut2(find(netOut2 > 0))  = 255;
netOut2(find(netOut2 <= 0)) = 0;

netOut3 = wMatrix*netIn3;
netOut3(find(netOut3 > 0))  = 255;
netOut3(find(netOut3 <= 0)) = 0;

netOut4 = wMatrix*netIn4;
netOut4(find(netOut4 > 0))  = 255;
netOut4(find(netOut4 <= 0)) = 0;

netOut5 = wMatrix*netIn5;
% netOut5(find(netOut5 > 0))  = 255;
% netOut5(find(netOut5 <= 0)) = 0;

% netOut1 = hopfieldRUN(wMatrix,netIn);

% netOut1(find(netOut1 > 0))  = 255;
% netOut1(find(netOut1 <= 0)) = 0;

% Reshape to image

imDispreshape1 = reshape(netOut1,testImSz(1),testImSz(2));
imDispreshape2 = reshape(netOut2,testImSz(1),testImSz(2));
imDispreshape3 = reshape(netOut3,testImSz(1),testImSz(2));
imDispreshape4 = reshape(netOut4,testImSz(1),testImSz(2));

imDispreshape5 = reshape(netOut5,testImSz(1),testImSz(2));

%%
figure_num = 50;
fig_name = ['Figure ', num2str(figure_num)];
figure('Name', fig_name, 'NumberTitle', 'off', 'Position',...
    [scrsz(3)/8 scrsz(4)/8 800 600])
subplot 221
imshow(imDispreshape1)
title('BRDM 2 Perspective 1 Out')
subplot 222
imshow(imDispreshape2)
title('BRDM 2 Perspective 2 Out')
subplot 223
imshow(imDispreshape3)
title('ZSU 23 4 Perspective 1 Out')
subplot 224
imshow(imDispreshape4)
title('ZSU 23 4 Perspective 2 Out')

figure
imshow(imDispreshape5)
title('Result if used on Raw SAR Image')
%
% figure
% imshow(imDispreshape)

disp('>> Done.')
tempTimer = num2str(toc(ticTimer));
disp(['Time Elapsed: ', tempTimer, 's'])


%% Record Movie

% writerObj = VideoWriter('Hopfield_Test_1');
% open(writerObj)
% writeVideo(writerObj,hopIterFrames(:));
% close(writerObj)
%



% figure
% movie(hopIterFrames)
```

File: SCRIPT_HopfieldMSTAR_V3_Rotate.m

```
% This script uses the hopfield model, trained with basic geometric
shapes,
% to determine the best shape fit of a pre-processed SAR image.
%
%


close all
clear
clc

% Runtime Check
ticTimer = tic;


%% Import MSTAR images to process

disp('Importing SAR Data...')

sarImg1 =
imread('..\..\testData\MSTAR\45_DEG\BRDM_2\HB15696.JPG','jpeg');
sarImg2 =
imread('..\..\testData\MSTAR\45_DEG\BRDM_2\HB15793.JPG','jpeg');

sarImg3 =
imread('..\..\testData\MSTAR\45_DEG\BRDM_2\HB15744.JPG','jpeg');
sarImg4 =
imread('..\..\testData\MSTAR\45_DEG\BRDM_2\HB15779.JPG','jpeg');

% Display test Data
scrsz = get(0, 'ScreenSize');
figure_num = 1;
fig_name = ['Figure ', num2str(figure_num)];
figure('Name', fig_name, 'NumberTitle', 'off', 'Position',...
    [scrsz(3)/8 scrsz(4)/8 800 600])
subplot 221
imshow(sarImg1)
title('BRDM 2 Perspective 1')

subplot 222
imshow(sarImg2)
title('BRDM 2 Perspective 2')

subplot 223
imshow(sarImg3)
title('ZSU 23 4 Perspective 1')

subplot 224
imshow(sarImg4)
title('ZSU 23 4 Perspective 2')

%% Filter images

sarImg1filt = im2bw(sarImg1,0.6);
sarImg2filt = im2bw(sarImg2,0.6);
sarImg3filt = im2bw(sarImg3,0.5);
sarImg4filt = im2bw(sarImg4,0.4);
```

```
sarImg1filt = medfilt2(sarImg1filt);
sarImg2filt = medfilt2(sarImg2filt);
sarImg3filt = medfilt2(sarImg3filt);
sarImg4filt = medfilt2(sarImg4filt);

%
testImSz(1) = 80;
testImSz(2) = 80;
%
% testImSz(1) = 50;
% testImSz(2) = 50;
%
% testImSz(1) = 20;
% testImSz(2) = 20;

inlength = testImSz(1)*testImSz(2);

sarImg1filt = imresize(sarImg1filt,[testImSz(1) testImSz(2)]);
sarImg2filt = imresize(sarImg2filt,[testImSz(1) testImSz(2)]);
sarImg3filt = imresize(sarImg3filt,[testImSz(1) testImSz(2)]);
sarImg4filt = imresize(sarImg4filt,[testImSz(1) testImSz(2)]);

sarImg5filt = imresize(sarImg1, [testImSz(1) testImSz(2)]);

% Display test Data
scrsz = get(0, 'ScreenSize');
figure_num = 2;
fig_name = ['Figure ', num2str(figure_num)];
figure('Name', fig_name, 'NumberTitle', 'off', 'Position',...
    [scrsz(3)/8 scrsz(4)/8 800 600])
subplot 221
imshow(sarImg1filt)
title('BRDM 2 Perspective 1')

subplot 222
imshow(sarImg2filt)
title('BRDM 2 Perspective 2')

subplot 223
imshow(sarImg3filt)
title('BRDM 2 Perspective 3')

subplot 224
imshow(sarImg4filt)
title('BRDM 2 Perspective 4')

% testImSz = size(sarImg1);

% testImSz(1) = 128;
% testImSz(2) = 128;


% sarImg1filt = imresize(sarImg1filt,[testImSz(1) testImSz(2)]);
% sarImg2filt = imresize(sarImg2filt,[testImSz(1) testImSz(2)]);
% sarImg3filt = imresize(sarImg3filt,[testImSz(1) testImSz(2)]);
% sarImg4filt = imresize(sarImg4filt,[testImSz(1) testImSz(2)]);

%% Hopfield Network Learning Period

% Import and generate training data

disp('>> Importing Training Data...')
```

```
tempTimer = num2str(toc(ticTimer));
disp(['Time Elapsed: ', tempTimer, 's'])

% Import Learning Data
%
% learnFilled(:,:,1) =
double(imcomplement(im2bw(imresize(imread(['..\..\testData\ObjectRecons
truction\RotationBlocks\filled_rect_W8_R0.tif'],'tiff'),[testImSz(1)
testImSz(2)]),0.9)));
%
% for n = 1:5
%     imNum = 15*n;
%     learnFilled(:,:,n) =
double(imcomplement(im2bw(imresize(imread(['..\..\testData\ObjectRecons
truction\RotationBlocks\filled_rect_W8_R_-', num2str(imNum)
,'d.tif'],'tiff'),[testImSz(1) testImSz(2)]),0.9)));
% end

learnFilled(:,:,1) =
double(imcomplement(im2bw(imresize(imread(['..\..\testData\ObjectRecons
truction\RotationBlocks\filled_2_rect_W8_R0.tif'],'tiff'),[testImSz(1)
testImSz(2)]),0.9)));

for n = 1:11
    imNum = 15*n;
    learnFilled(:,:,1+n) =
double(imcomplement(im2bw(imresize(imread(['..\..\testData\ObjectRecons
truction\RotationBlocks\filled_2_rect_W8_R_-', num2str(imNum)
,'d.tif'],'tiff'),[testImSz(1) testImSz(2)]),0.9)));
end


trainSz = size(learnFilled);



disp('>> Reformat Input...')
tempTimer = num2str(toc(ticTimer));
disp(['Time Elapsed: ', tempTimer, 's'])

for n = 1:trainSz(3)
    trainIn(:,:,n) = reshape(learnFilled(:,:,n), inlength, 1);
end

trainIn(find(trainIn == 0)) = -1;

netIn1 = double(reshape(sarImg1filt, inlength, 1));
netIn2 = double(reshape(sarImg2filt, inlength, 1));
netIn3 = double(reshape(sarImg3filt, inlength, 1));
netIn4 = double(reshape(sarImg4filt, inlength, 1));

netIn5 = double(reshape(sarImg5filt, inlength, 1))./255;

netIn1(find(netIn1 == 0)) = -1;
netIn2(find(netIn2 == 0)) = -1;
netIn3(find(netIn3 == 0)) = -1;
netIn4(find(netIn4 == 0)) = -1;

% tester = reshape(netIn1,testImSz(1),testImSz(2));

disp('>> Network Training...')
tempTimer = num2str(toc(ticTimer));
disp(['Time Elapsed: ', tempTimer, 's'])
wMatrix = hopfieldWeights(trainIn);
```

```
iterationCount = 1;

% netOut1 = netIn;

%%

disp('>> Running Network')
tempTimer = num2str(toc(ticTimer));
disp(['Time Elapsed: ', tempTimer, 's'])

% Set up Movie Capture
scrsz = get(0, 'ScreenSize');
figure_num = 100;
fig_name = ['Figure ', num2str(figure_num)];
figure('Name', fig_name, 'NumberTitle', 'off', 'Position',...
    [scrsz(3)/8 scrsz(4)/8 800 600])

imDispreshape1 = reshape(netIn1,testImSz(1),testImSz(2));
imDispreshape2 = reshape(netIn2,testImSz(1),testImSz(2));
imDispreshape3 = reshape(netIn3,testImSz(1),testImSz(2));
imDispreshape4 = reshape(netIn4,testImSz(1),testImSz(2));
subplot 221
imshow(imDispreshape1);
title('DHN BRDM 2 Perspective 1')
subplot 222
imshow(imDispreshape2);
title('DHN BRDM 2 Perspective 2')
subplot 223
imshow(imDispreshape3);
title('DHN BRDM 2 Perspective 3')
subplot 224
imshow(imDispreshape4);
title('DHN BRDM 2 Perspective 4')

set(gca, 'NextPlot', 'replaceChildren');

% Movie writer parameters
writerObj = VideoWriter('Hopfield_Test_2');
open(writerObj)
% writeVideo(writerObj,hopIterFrames(:));


% hopIterFrames(iterationCount) = struct('cdata',[],'colormap',[]);

% h = waitbar(0, 'Working...');
disp('Working...')
for n = 1:iterationCount
%     waitbar(n/iterationCount,h)
    tempLoopSz = size(netIn1);
    tempRunList = randperm(tempLoopSz(1));

    for m = 1:tempLoopSz(1)
        netIn1 = hopfieldRUN_V3(wMatrix, netIn1, tempRunList(m));
        netIn2 = hopfieldRUN_V3(wMatrix, netIn2, tempRunList(m));
        netIn3 = hopfieldRUN_V3(wMatrix, netIn3, tempRunList(m));
        netIn4 = hopfieldRUN_V3(wMatrix, netIn4, tempRunList(m));

        imDispreshape1 = reshape(netIn1,testImSz(1),testImSz(2));
        imDispreshape2 = reshape(netIn2,testImSz(1),testImSz(2));
        imDispreshape3 = reshape(netIn3,testImSz(1),testImSz(2));
        imDispreshape4 = reshape(netIn4,testImSz(1),testImSz(2));

        subplot 221
        imshow(imDispreshape1);
        subplot 222
        imshow(imDispreshape2);
        subplot 223
        imshow(imDispreshape3);
        subplot 224
        imshow(imDispreshape4);


        hopIterFrames(n) = getframe;
        writeVideo(writerObj,hopIterFrames(n));
    end

%     imDispreshape1 = reshape(netIn1,testImSz(1),testImSz(2));
%     imshow(imDispreshape1);
%     hopIterFrames(n) = getframe;
    disp(['Iteration #', num2str(n)])
end

close(writerObj)

% fprintf(' \n')
% close(h)

% for n = 1:10
%     netIn1 = wMatrix*netIn1;
%     netIn2 = wMatrix*netIn2;
%     netIn3 = wMatrix*netIn3;
%     netIn4 = wMatrix*netIn4;
% end

%%

% netOut1 = wMatrix*netIn1;
% netOut1 = wMatrix*netOut1;
% netOut1 = wMatrix*netOut1;
% netOut1 = wMatrix*netOut1;
% netOut1 = wMatrix*netOut1;
netOut1(find(netIn1 > 0))  = 255;
netOut1(find(netIn1 <= 0)) = 0;

% netOut2 = wMatrix*netIn2;
% test = netOut2.*255;
netOut2(find(netIn2 > 0))  = 255;
netOut2(find(netIn2 <= 0)) = 0;

% netOut3 = wMatrix*netIn3;
netOut3(find(netIn3 > 0))  = 255;
netOut3(find(netIn3 <= 0)) = 0;

% netOut4 = wMatrix*netIn4;
netOut4(find(netIn4 > 0))  = 255;
netOut4(find(netIn4 <= 0)) = 0;

netOut5 = wMatrix*netIn5;
% netOut5(find(netOut5 > 0))  = 255;
% netOut5(find(netOut5 <= 0)) = 0;

% netOut1 = hopfieldRUN(wMatrix,netIn);

% netOut1(find(netOut1 > 0))  = 255;
% netOut1(find(netOut1 <= 0)) = 0;

% Reshape to image
```

```
imDispreshape1 = reshape(netOut1,testImSz(1),testImSz(2));
imDispreshape2 = reshape(netOut2,testImSz(1),testImSz(2));
imDispreshape3 = reshape(netOut3,testImSz(1),testImSz(2));
imDispreshape4 = reshape(netOut4,testImSz(1),testImSz(2));

imDispreshape5 = reshape(netOut5,testImSz(1),testImSz(2));

%%

figure
subplot 221
imshow(imDispreshape1)
title('BRDM 2 Perspective 1 Out')
subplot 222
imshow(imDispreshape2)
title('BRDM 2 Perspective 2 Out')
subplot 223
imshow(imDispreshape3)
title('BRDM 2 Perspective 3 Out')
subplot 224
imshow(imDispreshape4)
title('BRDM 2 Perspective 4 Out')

figure
imshow(imDispreshape5)
title('Result if used on Raw SAR Image')
%
% figure
% imshow(imDispreshape)

disp('>> Done.')
tempTimer = num2str(toc(ticTimer));
disp(['Time Elapsed: ', tempTimer, 's'])


%% Record Movie

% writerObj = VideoWriter('Hopfield_Test_1');
% open(writerObj)
% writeVideo(writerObj,hopIterFrames(:));
% close(writerObj)
%



% figure
% movie(hopIterFrames)
```

File: dirFinder.m

```
function [centl, centr, corAng] = dirFinder(inputHopImg)
% Finds the direction that an image, generated by the Hopfield
Direction
% Finder, that the object is pointint by using Connected Component
% Analysis.

tempImg = im2bw(inputHopImg);
imSz = size(tempImg);

[labeledIm,count] = bwlabel(tempImg);

% figure
% imshow(labeledIm.*100)

% Find Largest two connected regions
maxSize = 0;
nextMaxSize = 0;
maxIndex = 0;
secondMaxIndex = 0;

for m = 1:2
    for n = 1:count
%         n
        tempCompare = find(labeledIm == n);
        tempCount = length(tempCompare);

%          if tempCount < maxSize
%             maxSize = tempCount;
%             maxIndex = n;
%         elseif tempCount < nextMaxSize && tempCount > maxSize
%             nextMaxSize = tempCount;
%             secondMaxIndex = n;
%         elseif tempCount == maxSize && tempCount == nextMaxSize
%             disp('Error')
%         end


        if (tempCount > maxSize)
            maxSize = tempCount;
            maxIndex = n;
%             disp('Set MAX')
        elseif ((tempCount > nextMaxSize) && (n ~= maxIndex))
            nextMaxSize = tempCount;
            secondMaxIndex = n;
%             disp('Set min')
%         elseif (tempCount == maxSize && tempCount == nextMaxSize)
%             nextMaxSize = tempCount;
% %             secondMaxIndex = n;
% %             disp('Error')
        end

    end
end

% % Debugging
% maxIndex
% maxSize
```

```
%
% secondMaxIndex
% nextMaxSize

[y1,x1] = find(labeledIm == maxIndex);
[y2,x2] = find(labeledIm == secondMaxIndex);

centX1 = mean(x1);
centY1 = mean(y1);

centX2 = mean(x2);
centY2 = mean(y2);


% Straight Line Dir Approx
% t = linspace(0,imSz(2)/2);

if (centX1 > centX2)
    xr = centX1;
    yr = centY1;
    xl = centX2;
    yl = centY2;
else
    xr = centX2;
    yr = centY2;
    xl = centX1;
    yl = centY1;
end

centr(1) = xr;
centr(2) = yr;

centl(1) = xl;
centl(2) = yl;

angleCorrection = (yr-yl)/(xr-xl);

% figure
% imshow(tempImg)
% hold on
% plot(centX1,centY1,'rx')
% plot(centX2,centY2,'bx')
% % plot(1,1,'rO')
% % plot(x1,y1,'rx')
% % plot(x2,y2,'bx')
% hold off

tempAng = abs(atan(angleCorrection)*180/pi);

nanCheck = isnan(tempAng);
if (nanCheck)
    corAng = 90;
elseif (yr > yl)
    corAng = tempAng;
elseif (yr < yl)
    corAng = -1*tempAng;
else
    corAng = tempAng;
end

end
```

File: testRotateScript.m

```
% Tests the dirFinder function with pre-loaded rotation detection
material



[cl1, cr1, angR1] = dirFinder(imDispreshape1)
[cl2, cr2, angR2] = dirFinder(imDispreshape2)
[cl3, cr3, angR3] = dirFinder(imDispreshape3)
[cl4, cr4, angR4] = dirFinder(imDispreshape4)

scrsz = get(0, 'ScreenSize');
figure_num = 1;
fig_name = ['Figure ', num2str(figure_num)];
figure('Name', fig_name, 'NumberTitle', 'off', 'Position',...
    [scrsz(3)/8 scrsz(4)/8 800 600])

subplot 221
imshow(imDispreshape1)
title('Hopfield Output Img 1')
hold on
plot(cl1(1),cl1(2),'b*')
plot(cr1(1),cr1(2),'r*')
% plot(1,1,'rO')
% plot(x1,y1,'rx')
% plot(x2,y2,'bx')
hold off

subplot 222
imshow(imDispreshape2)
title('Hopfield Output Img 2')
hold on
plot(cl2(1),cl2(2),'b*')
plot(cr2(1),cr2(2),'r*')
% plot(1,1,'rO')
% plot(x1,y1,'rx')
% plot(x2,y2,'bx')
hold off

subplot 223
imshow(imDispreshape3)
title('Hopfield Output Img 3')
hold on
plot(cl3(1),cl3(2),'b*')
plot(cr3(1),cr3(2),'r*')
% plot(1,1,'rO')
% plot(x1,y1,'rx')
% plot(x2,y2,'bx')
hold off


subplot 224
imshow(imDispreshape4)
title('Hopfield Output Img 4')
hold on
plot(cl4(1),cl4(2),'b*')
plot(cr4(1),cr4(2),'r*')
% plot(1,1,'rO')
% plot(x1,y1,'rx')
% plot(x2,y2,'bx')
hold off



rotatedOut1 = imrotate(sarImg1,angR1);
rotatedOut2 = imrotate(sarImg2,angR2);
rotatedOut3 = imrotate(sarImg3,angR3);
rotatedOut4 = imrotate(sarImg4,angR4);


% scrsz = get(0, 'ScreenSize');
figure_num = 2;
fig_name = ['Figure ', num2str(figure_num)];
figure('Name', fig_name, 'NumberTitle', 'off', 'Position',...
    [scrsz(3)/8 scrsz(4)/8 800 600])


subplot 221
imshow(sarImg1)
title('Original Image 1')
subplot 222
imshow(sarImg2)
title('Original Image 2')
subplot 223
imshow(sarImg3)
title('Original Image 3')
subplot 224
imshow(sarImg4)
title('Original Image 4')

figure_num = 3;
fig_name = ['Figure ', num2str(figure_num)];
figure('Name', fig_name, 'NumberTitle', 'off', 'Position',...
    [scrsz(3)/8 scrsz(4)/8 800 600])

subplot 221
imshow(rotatedOut1)
title('Final Rotated Output 1')
subplot 222
imshow(rotatedOut2)
title('Final Rotated Output 2')
subplot 223
imshow(rotatedOut3)
title('Final Rotated Output 3')
subplot 224
imshow(rotatedOut4)
title('Final Rotated Output 4')
```