Deep Learning Networks for Target Recognition

by Roan Song

prepared for A. K. Mishra

Department of Electrical Engineering

University of Cape Town

Submitted to the Department of Electrical Engineering at the University of Cape Town in partial fulfilment of the academic requirements for a Bachelor of Science degree in Electrical and Computer Engineering.

January 2017

Blank page

Declaration

1. I know that plagiarism is wrong. Plagiarism is to use another's work and pretend that it is one's own.
2. I have used the IEEE convention for citation and referencing. Each contribution to, and quotation in, this report from the work(s) of other people has been attributed, and has been cited and referenced.
3. This report is my own work.
4. I have not allowed, and will not allow, anyone to copy my work with the intention of passing it off as their own work or part thereof.

Signature: ....

Roan Song

Date: ....

Acknowledgements

Abstract

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis aims to asses the effectiveness of deep learning techniques in the classification of radar imagery. Deep learning relies on the use of neural networks; interconnected layers of nodes sharing information and undergoing non-linear transformations that, through training and optimisation, can automatically detect and extract features from a dataset. By training a model to classify known instances, it gains predictive power, which can then be tested and used on instances not previously encountered by the model, with varying degrees of accuracy. In the specific case of this thesis, the instances are radar images from the MSTAR dataset.

1.1 Background

As processing power has become more readily available over the years, the field of machine learning has seen many innovations. Deep learning techniques were once considered only theoretically viable due to the limits of computing power when the concepts were developed, but are now being used to great effect in commercial ventures, including image classification and facial recognition applications. Their success lies in the ability of neural networks to train themselves under human supervision, extracting features from a dataset that are not always intuitive in a human sense, and in many cases providing greater accuracy in classification than humans can achieve, in a fraction of the time (cite some medical science paper here). This accuracy and speed of classification lends itself to the field of target acquisition and recognition; recognising and classifying a target based on its radar signature, to allow for human action to be taken. Naïve classification methods have already been proven somewhat effective (REALLY???), and the aim of this study is to assess the possible application of deep learning techniques in this field.

1.2 Motivation

Deep learning techniques have been proven effective in many different fields. They promote fast classification suitable for real-time applications; the time taken to train a model is much greater than the time taken to classify a specific instance. Image classification is a complex problem, and simple classification methods are not as effective on feature-rich data such as images, because the classes are typically not linearly separable.

1.3 Objectives

The goals of this thesis are to assess the effectiveness of deep learning techniques in classifying radar imagery from the MSTAR dataset. The assessment will compare and contrast different types of classifiers: K-Nearest Neighbours, Multilayer Perceptron (one hidden layer), Multilayer Perceptron (two hidden layers), and a Convolutional Neural Network.

The performance of each classifier will be assessed in terms of its accuracy, training time, and classification time.

1.4 Scope and Limitations

1.4.1 Focus

1.4.2 Scope

Within project scope

Outside project scope

1.4.3 Limitations

The MSTAR dataset is comparatively small, with 100-274 images per class.

1.5 Report Overview

Chapter 2

Literature Review

2.1 Synthetic Aperture Radar

2.1.1 Description

SAR is used to create images of objects, such as vehicles (as in this report), or landscapes. The images are constructed by sending a radar signal from a moving platform, and the time taken for the signal to return to the antenna denotes the size of the aperture. The aperture can be physical, with a large antenna, or synthetic in the case of a moving aperture. Larger apertures allow for higher image resolution. SAR images consist of magnitude and phase data, from which elevation data can be calculated. The classification of 2D SAR images, the type dealt with in this report, requires only the magnitude data to be preserved.

2.1.2 Relevance

The dataset chosen for this report is comprised of SAR imagery. Understanding the nature of the format allows the decision to strip the data of phase information and keep only magnitude data to be made.

2.2 The MSTAR Dataset

2.2.1 Description

The MSTAR public mixed targets dataset contains X-band synthetic aperture radar (SAR) image chips of multiple targets. Each image has a resolution of 1 foot, and is captured in spotlight mode. Information pertaining to each target, including its elevation, depression angle, and target type is contained in a header section of each file. The header is followed by magnitude and phase data of the SAR imagery. Because the SAR images are 2D, only the magnitude data needs to be considered. Each point represents the brightness of a pixel, reducing the complexity of this study to that of image-based target recognition. [1]. Each class of targets has between 195 and 274 instances, with class images ranging in sizes from 48x48 pixels to 198x198 pixels. The targets in each class are rotated between 0°and 360°.

2.2.2 Relevance

The MSTAR dataset was suggested for this study by A. Mishra. It provides a generic SAR image chip dataset on which any classification method can be run. The dataset is comparatively small, with 195-274 images per class, and thus is suitable for machine learning on a consumer-grade desktop computer with reasonable classification time. The rotation factor present in each image introduces complications, as Figure 2.1 shows, there is a large disparity in appearance between instances of the same class.

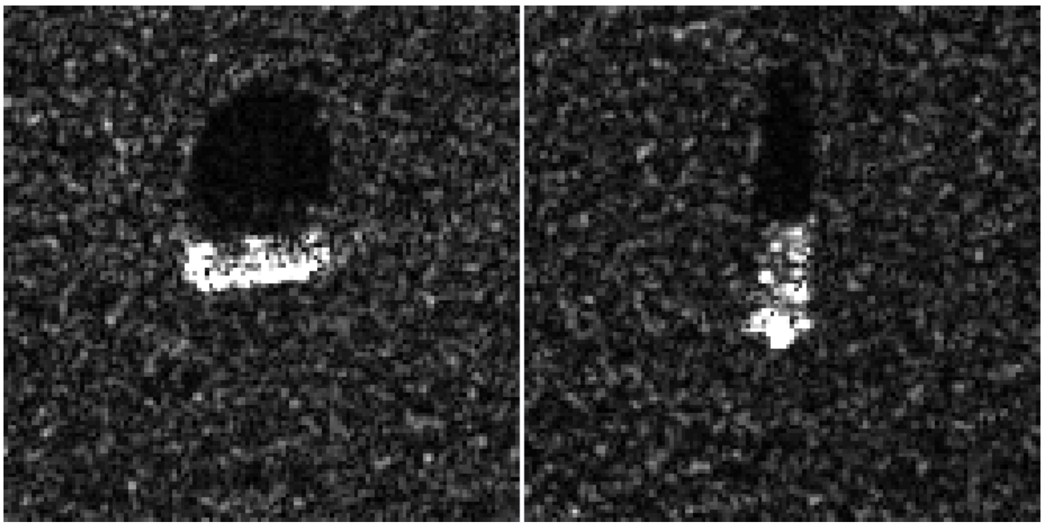

Figure 2.1: Rotational difference between two images of the same class

2.3 Classification

2.3.1 Nearest-Neighbour Classification

Description

The nearest neighbour classifier operates as follows: When given an input, the classifier compares this input to the training data set, and finds the one that is closest to the input. For example, if men and women were to be classified by their heights, a given input would be classified as either male or female based on the data point in the training data with the height closest to that of the input. This can be expanded to multiple features/dimensions by taking the Euclidean distance between the input and each instance in the training data set. For images, this amounts to comparing, pixel by pixel, each pixel value, and finding the L2 distance between them. Note that there is no need to apply the square root to the distance; it is a monotonic operation, so it will not affect the ordering of the values, and will introduce additional computational complexity. The equation for calculating the L2 distance is:

$$d_2(a, b) = (a - b)^2$$

Each pixel usually has some relation to the pixels near to it, so there is the possibility for a better definition of 'distance' between images to be made [2,3].

The nearest neighbour classifier has been shown to provide high classification rates (85+%) through sufficient image processing and classifier development. A caveat of the success, however, is the inclusion of image chip 'clutter' present in each image affecting the success of clarification. Images in the training dataset have similar clutter, and there is a non-negligible contribution to the success of the classification. [1]

Relevance

The nearest-neighbour classifier is used in this study as an example of a naïve classifier. Results of this classifier provide a good benchmark against which subsequent classifier performance can easily be measured. The success of other parties in classifying the MSTAR targets using nearest neighbour methods lays a convincing foundation for future development. There is undoubtedly room for improvement, beginning with the elimination of clutter's effect on classification.

2.4 Deep Learning

The objective of this study is to test the performance of deep learning-based classifiers on SAR image chip data. The success of naïve methods has already been proven [1], but lack the predictive power of a truly intelligent classifier. Neural networks and the application of deep learning are key to extracting features from the data to further improve classification rates.

2.4.1 Neural Networks

Description

A neural network is a system inspired by the perceived workings of the human brain; a system of neurons combine to perform tasks that exceed their individual capabilities. An input is passed through a series of neuron layers, each of which is tuned to identify characteristic features of the input between each layer, allowing for feature extraction and identification.

(From the hartford.edu site) A neural network consists of four main parts [4]:

1. Processing units{uj}, where each uj has a certain activation level aj(t) at any point in time.
2. Weighted interconnections between the various processing units which determine how the activation of one unit leads to input for another unit.
3. An activation rule which acts on the set of input signals at a unit to produce a new output signal, or activation.
4. Optionally, a learning rule that specifies how to adjust the weights for a given input/output pair.

Relevance

The motivation behind using neural networks is simple; instead of specifying basic characteristics for a system to detect, the system is given an input and a matching output and is left to develop its own perceptions of what important feature link the two. Through optimisation and iteration this can become a very successful form of classification.

2.4.2 Multilayer Perceptron

Description

A multilayer perceptron is an extension of the simple perceptron network. The initial design was an input layer, an output layer, and a hidden layer of neurons between. The basic perceptron was deemed unsuitable for complex classification tasks, as its input-output relationship could be reduced (via back-propagation) to a set of linear combinations. This means that is can only correctly classify data that is perfectly linearly separable, which limits its practical applications significantly.

The multilayer perceptron used two or more hidden neuron layers, with non-linear activation functions at each neuron. This makes a multilayer perceptron an example of a deep neural network. The depth allows for more complex feature detection, and ultimately better performance in complex classification tasks. The multilayer perceptron becomes difficult to optimise as the number of hidden layers grows, because the effect of each neuron on the output, and the effects of previous neurons on **that** neuron become progressively and recursively difficult to compute.

Relevance

Implementing a multilayer perceptron is a valuable exercise, as it provides an example of a deep neural network with a fairly straightforward implementation. Training time becomes a significant factor when using a deep neural network due to the time taken to complete back-propagation optimisation, so using a multilayer perceptron will force the development of more efficient methods of data pre-processing to speed up the training as much as possible.

2.5 Optimization and Training

A classifier is only operating efficiently when it is tuned to the data it is attempting to classify. Deep neural networks are initialised with random weights between their neurons, and at first use will perform worse on average than naïve classification methods. Through optimisation of these inter-neuron weights, however, the potential of deep neural networks can be reached, and classification results are expected to significantly improve. Tuning the classifier to the dataset is crucial, but optimising too heavily may result in *overfitting* of the data, leaving the classifier with no predictive power on unseen data.

2.5.1 Back-propagation

Description

Back-propagation is a system by which the effects of weights between neurons is adjusted through an iterative process. The base case is that of a single input, single output system. Varying the weight on the input directly effects the output. This change can be easily recognised, and the weight can be changed to more suitably link the input to the desired output. This involves developing a method of changing weights in a sensible manner. The most common form of this is through *gradient descent*, whereby the weights are adjusted corresponding to their perceived effect on the output state, and their rate of change. Back-propagation is not guaranteed to find a global minimum, and can settle on a local minimum instead, which can be somewhat alleviated through the use of random weights and multiple training rounds, before choosing the best version of the classifier that has been discovered.

One of the key issues with back-propagation is its computational complexity. With deep neural networks, the sheer number of weights and their possible combinations make discerning their impact on the output very difficult, and computationally infeasible to perfectly optimise.

Relevance

Back-propagation is a popular and successful technique, well-suited to neural networks with only a few layers. With enough time, it can help to optimise much larger networks, and potentially improve classification accuracy by a large margin. It alleviates the concern of trying to find the perfect network from the offset; it allows any network to be tuned to be better than it currently is.

2.5.2 Hyper-parameters

Description

Hyper-parameters are parameters that, when changed, modify the structure or operation of the neural network, without changing its core mechanics. Hyper-parameters under consideration in this project are:

- Image size (100px vs 1000px)
- Learning Rate
- Network Shape (hidden layer size, and number of layers)
- Choice of activation function
- Weight initialisations

Relevance

2.6 Performance Evaluation

2.6.1 Separability Index

Thornton's Separability Index (SI) is used as a metric to show how separable the data is into its different classes, i.e. the confidence level with which distinct classes can be identified with no overlap. Perfectly separable data would have an SI of 1, or 100

2.6.2 Training, Validation, and Testing

Successful classification of a dataset is divided into three distinct steps:

1. Training
2. Model Validation
3. Testing

Training is the process of fitting a classifier - it involves running multiple iterations on a given set of inputs, comparing the output of the classifier to a known target dataset, and adjusting the parameters of the classifier (typically inter-layer weights and bias) through back-propagation. To find the best iteration of the classifier model, it is periodically tested on a different set of known data; the validation dataset. Testing on this intermediate dataset is used to provide performance metrics such as the mean error and the accuracy of classification, which is useful in selecting a model that is optimised to the desired set of parameters. The data from periodically testing on this validation set is used to tune the model, or implement early-stopping procedures. For example, if the desired level of classification accuracy has been achieved or if the mean error hasn't changed significantly after a number of epochs, the training can be stopped early. On the contrary, if the training is approaching its stated limit yet still improving classification accuracy, the number of epochs or 'patience' can be increased to allow for further iterations and tuning.

Once the training is complete, having achieved the desired level of classification accuracy, the chosen model can be tested on another dataset (typically a set of real-world instances) to see how it performs, providing the testing accuracy. The classifier is no longer tuned, and can be presented with a variety of inputs to simulate its real-world performance.

A dataset is typically split into three sections. 50% training, 25% validation, and 25% testing is a reasonable starting point, and the proportions can be seen as a hyperparameter to be optimised. If the training set is too small, there is a risk that the model will not be able to successfully extract the features required for classification, and its testing accuracy will be low. If the validation or test sets are too small, the model validation and testing might not be thorough enough to cover all reasonable use cases.

2.6.3 The Confusion Matrix

Confusion matrices are useful tools for evaluating classifier performance. They can show the accuracy, misclassification rate, true/false positive rates, specificity, precision, and prevalence. They clearly show the number of correctly classified classes along the matrix's diagonal, and also shows how many of the incorrect classifications were attributed to which classes.

2.7 Hidden Layers

2.8 Logistic Regression

2.9 Dimensionality Reduction

When dealing with images, the number of pixels grows in proportion to the size of the image. Since every pixel has a weight linking it to the first layer of the neural network, and these weights in turn need to be tuned, the computation power required to train the model is proportional to the size of the input. To reduce computational overhead, it is desirable to reduce the input size, without compromising classification accuracy. This can be done using *dimensionality reduction.*

2.9.1 PCA

Chapter 3

Methodology

3.1 K-Nearest Neighbour Classification

The steps necessary to implement the NN on the MSTAR dataset are as follows:

- Convert the raw data + header MSTAR files into .tiff images
- figure out how to read and display these images in matlab
- write a pixel-by-pixel comparison method
- test the method by comparing one image to itself and others
- collate a set of mixed radar targets for testing
- extract data from each image/filename to help with seeing if the classification is correct
- TEST
- collect results!

The KNN classifier works by predicting the class of an instance based on its relative distance to nearby instances. The K nearest instances (the "neighbours" after which the classifier is named) are used, and the most common class amongst them is selected as the predicted class. K is always an odd number, to prevent ties. A measure of confidence in the prediction can be taken as the ratio of the most occurring class to the total number of neighbours under consideration. K can be optimised for each dataset

3.1.1 Implementation

The algorithm

Given a training dataset of known instances and targets, and an input instance X, the squared difference between X and each instance in the dataset is calculated. The K closest instances (neighbours) are selected, and the class predicted is the most commonly occurring class within the selection of neighbours. The time taken to predict a class is linearly proportional to the size of the training dataset ($O(n)$), and exponentially proportional to the size of each instance ($O(n^2)$). A 100x100 image has 10,000 points of comparison; 100 times more than a 10x10px image. Reducing the image size through cropping or dimensionality reduction can thus have a desirable effect on computation time.

Optimising for K

Optimal K values are found by performing leave-one-out cross-validation (LOOCV) and finding which value of k gives the greatest prediction accuracy across all cases. To find this value of k using the previous algorithm is a very time-consuming process. Because this optimisation uses only the training set, the process can be sped up significantly. A matrix containing the squared distances between each instance is constructed. An example of such a matrix is shown in Figure **??**. Each row represents the squared distances between the instance corresponding to that row, and every other instance. This leads to the matrix being mirrored along its leading diagonal, which speeds up the construction of the matrix by not having to recalculate those values. All of the diagonals of the matrix are zero; they represent the distance between an instance and itself.

To optimise for K, each row is taken in turn, sorted in ascending order, removing the first element (the zero self-term), and calculating the predicted class and confidence.

After testing the Nearest Neighbour algorithm on a set of single-target data, it is necessary to see how well the algorithm performs when other targets are added to the mix. A subset of 10 images of each target is selected, and added to the testing pool.

A system needs to be made that automates the rotation of images and testing, in order to test the Leave-One-Out Cross-Validation accuracy.

After rescaling the images, predictive accuracy dropped to just 6/70. This is because scaling the images without accounting for the effects of their size leads to smaller images having a much smaller cumulative distance. Weighting the distance result by the number of pixels in an image rectified this immediately, bringing the predictive accuracy up to 56/70!

3.2 Data Processing

The first step is to get to grips with the data - processing the data set in a way that makes sense to use, and allows different classification methods to be implemented easily on it.

The MSTAR dataset contains eight different targets. The dataset is sorted by depression angle, and by 'scene'. All of the targets have data corresponding to a 15° depression angle. Some targets have additional angles available, but the 15° set is the one that will be used.

For supervised learning, we want to have each target in the dataset labelled with corresponding information, most notably its class. This is often done using a one-hot array (i.e. [1 0 0], [0 1 0] for 1 and 2, respectively) relating to each target. The MSTAR dataset stores the information of each target in a header section of each file. This is inconvenient when reading in image files directly, so I will most likely process the data and create the identifying arrays from the header data. Since I will have defined the format in a way that is convenient for me to use, it will make my life a lot easier.

3.3 Dimensionality Reduction

An image can be viewed as a collection of pixels, comprising a feature vector. If the size of this vector can be reduced while retaining enough information for classification, we can greatly increase our training and classification speeds. Simply selecting every other pixel (or one in five) would greatly reduce the dimensionality of the data but may remove features that are key to classification, thus having a negative impact on the results.

Chapter 4

Design

4.1 Design Context

This study is within the scope of an undergraduate level approach to both radar and machine learning, with an emphasis on machine learning; the techniques applied herewith are not limited to radar imagery, but attempts should be made to tailor the design to radar applications. With development this study should be adaptable to commercial use, and be helpful to people in the radar department who need assistance with radar target classification. Thus the study should develop an easily extensible framework for radar image classification, or at least guidelines to allow others to integrate with the work covered herein.

4.2 Feasibility Study / Concept Exploration

Does the classification of radar imagery lend itself to deep learning techniques, and if so, will the performance be better or worse that naïve classification methods? This is the question that best captures the analysis of the study's feasibility.

Consideration of this question needs to include the format of the data entering the system, the ability of the system to process such data, and

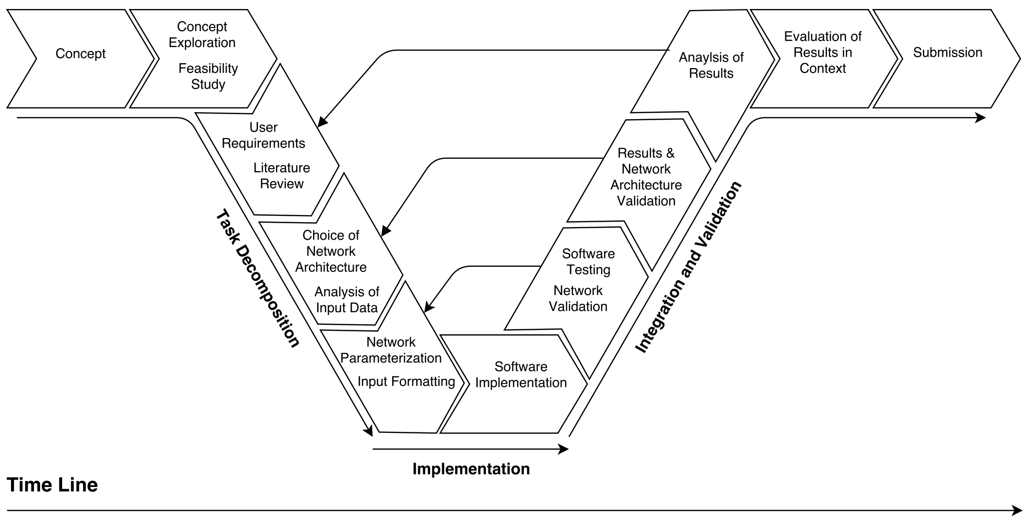

Figure 4.1: Vee Diagram

the effectiveness of the classification of the data.

The data consists of images of between 2916 (54x54) and 37054 (192x193) pixels in size. Each image contains one target, positioned at the centre of the image. This should allow for effective resizing of images to account for size discrepancies. Each pixel in an image constitutes an input. The computational cost of processing the largest image in the set versus the smallest will be at least 12.7 (37054/2916) times more expensive. Processing of such images will be made much more feasible if they can be reduced in size to match the smallest images present in the dataset, or at least be made as small as possible while retaining all of the information needed to classify each target.

Neural networks have a fixed structure; a collection of input neurons feeding their values through a series of hidden neuron layers before arriving at an output layer of neurons equal in size to the number of classes present in the data. The architecture of the network - the choice of number of hidden layers, the number of neurons in each hidden layer, and the number of neurons in the input layer are all subject to change during the development of the system. The training and operation of the system occurs through the adjustment of inter-neuron weights, with the structure of the system remaining constant.

The choice of size of the input layer is crucial; it must remain constant throughout the training and operation of the network. Since each pixel in an image forms of the input layer's neurons, all of the input images must be processed to contain the same number of pixels before any other work on the network's architecture can begin.

Once the pre-processing of the input images is complete, the structure of the neural network can be decided. This structure shall be changed and prototyped in order to try to find a good corresponding fit for the data. Having too many hidden layers will greatly increase the time taken by the back-propagation algorithm to optimise the weights of the system, and the likelihood of it settling at a local instead of global minimum increases with the complexity of the system.

The number of neurons in each hidden layer must also be chosen carefully; too many neurons in each layer will result in much longer optimization time (proportional to the increase in the number of inter-neuron weights created). Having too few neurons in a layer can result in the system being unable to extract the features key to classification, and too many neurons may lead to 'overfitting' of the training data, leaving the system with no predictive capability (an inability to classify data not present in the set of training instances).

4.3 Decomposition and Definition

This section is devoted to describing the study in terms of its requirements, operation, and implementation. An accompaniment explaining the verification and validity of each subsection will be in the next section.

4.3.1 Concept of Operations

Radar target classification is an inexact science; interpreting a radar image and comparing it to a known case is not as straightforward in all cases as one might expect. Weather conditions, environmental clutter, and image resolution all obscure the target to varying degrees, making intuitive classification ineffective. Computer-based classification through analysis of multiple targets and the application of deep learning techniques should in theory allow distorted images to be classified after the computer is trained to recognise features pertaining to each class. Naïve methods of classification lack predictive power - the ability to 'guess' effectively if the target is obscured or unrecognised. Deep learning methods are the solution that this study proposes.

4.3.2 System Requirements

4.3.3 High-Level Design

For this study, these areas of design need to be focused on:

- Image Processing/Preparation
- Further Dimensionality Reduction
- Naïve Classification (Nearest Neighbour as a benchmark)
- Deep Learning Classification (Multilayer Perceptron)

4.3.4 Detailed Design

Image Processing/Preparation

The MSTAR dataset is a compilation of image chips, all of which contain a header, as well as magnitude and phase data. The images are between 54x54 and 192x193 pixels in size, which suggests that some form of image processing should be performed to make sure that all images are the same size. The targets in each image chip are centred, suggesting that cropping each image to a size where the target (and its shadow - useful in classification) are left whole, and as much of the surrounding clutter as possible is removed.

An alternate approach is to retain the data inherent in the environmental clutter and pad the smaller images with zeros, keeping all images in the set at the size of the largest image in the set. While this preserves all of the image chip data, processing larger images leads to infeasibly long training and classification times.

To accurately preserve the data while making the image as small as possible, the images will be cropped in a rectangular box, containing the target and its shadow. The crop dimensions will be chosen to consider all of the image chips. Further processing of the images may be done, to allow for quicker classification of a larger number of images.

Dimensionality Reduction

Each pixel in an image is taken as an feature, forming a feature vector with a length equal to the total number of pixels in the image. The image cropping mentioned in section 4.3.4 is very effective at reducing the size of this feature vector. The image size is reduced to ———- (a total of —– features). This can be reduced further through the application of dimensionality reduction techniques, such as PCA, LLE, SOM and non-linear methods.

The first foray into dimensionality reduction will be conducted using PCA techniques 2.9.1(covered in the literature review)

Nearest Neighbour Design

The high-level design is fairly straightforward, and is shown in Figure 4.2. Its principles of operation are covered in section 2.3.1. To implement this classifier, the following is needed:

- Access to the dataset
- A choice of input image
- A method to calculate and sum the pixel-wise distances
- A variable storing the smallest distance and tentative classification
- A method displaying the chosen class and whether or not it is correct

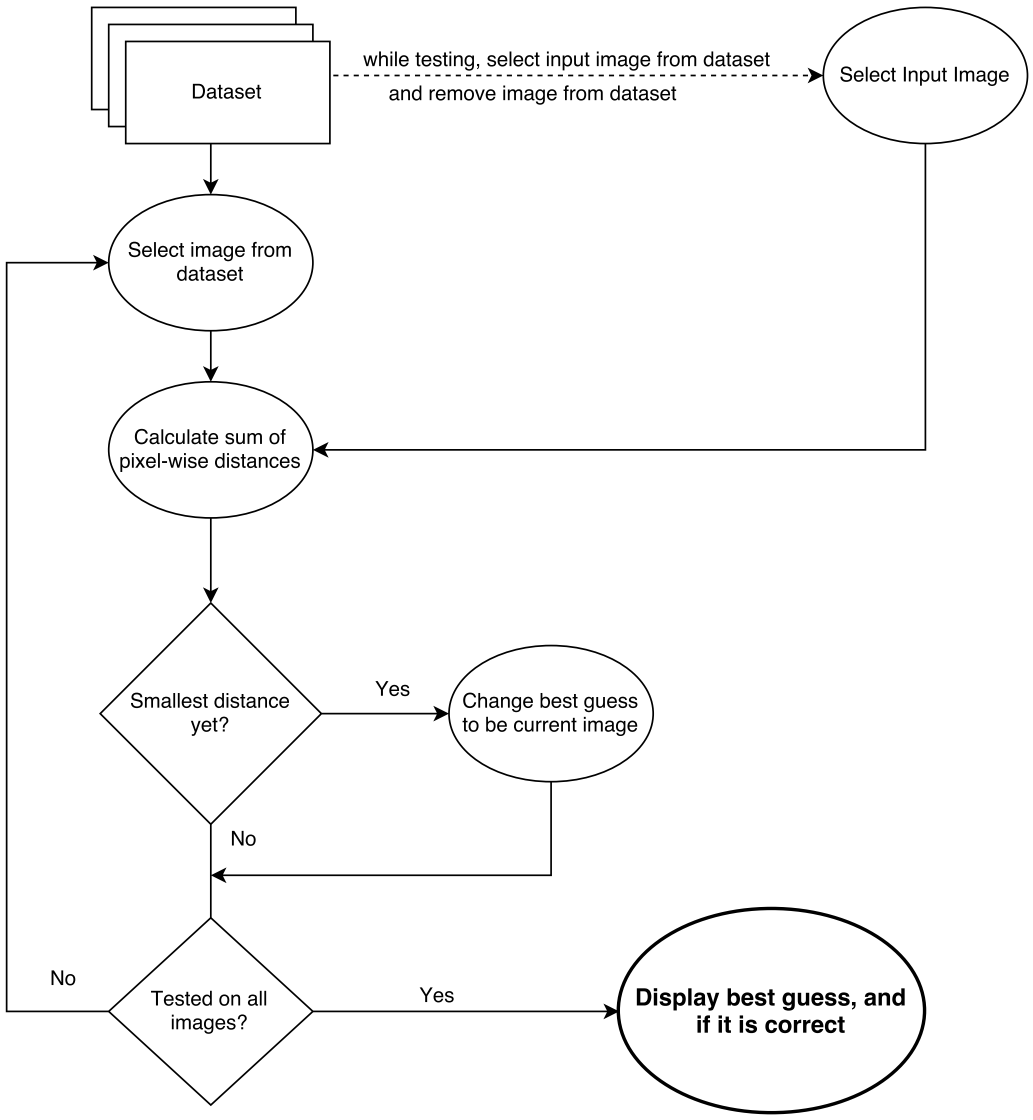
Dataset
while testing, select input image from dataset
and remove image from dataset
Select Input Image
Select image from dataset
Calculate sum of pixel-wise distances
Smallest distance yet?
Yes
Change best guess to be current image
No
No
Tested on all images?
Yes
Display best guess, and if it is correct

Figure 4.2: Nearest Neighbour Classification Flow Diagram

Multilayer Perceptron Design

Implementation of a multilayer perceptron in software can be divided into discrete sections as follows:

-

4.3.5 Software Development and Implementation

—– The software present in this study has been made available on Github

4.4 Integration and Recomposition

4.4.1 Software Development and Implementation

Inline Testing

While implementing the software for this report, I ran into some early issues that resulted from insufficient planning. After taking a step back, it was decided that more steady progress would be made by implementing rigorous inline testing, i.e. incrementally testing the code after every slight modification, instead of only testing after the addition of a major feature and then trying to iron out any latent bugs. This approach leads to much simpler debugging, as you only have to worry about a few lines of fresh code at a time.

The Multilayer Perceptron

In a process outlined in Figure 4.6, the input image is converted to an array of unsigned 8-bit integers (ranging from 0 to 255). The elements of the array are 'unitised' by subtracting the mean of the array from each and then dividing each element by the standard deviation of the array. This ensures that the processed values lie centred around zero, and mostly between -1 and

1. Unsigned integers are no longer suited to represent this data, so 32-bit floating point numbers (floats) are used.

Once each pixel has been unitised, they form the input layer of the neural network, with each pixel representing a neuron in the input layer shown in Figure 4.5.

Each of these input layer neurons has a random weight applied, and is then fed into the neurons that form the first hidden layer. Every neuron in the input layer contributes to every neuron in the next layer. At each neuron in the first hidden layer, the 'net' value is formed by summing all of the values present at its input (i.e. all the weighted values passed along from the input layer). An activation function is applied to this 'net' value to compress the range of values. There are many different activation functions that can be used, and for this study I narrowed down the choice to either the logistic function $\frac{1}{1+\exp^{-net}}$ or the hyperbolic tangent function. The logistic function produces an output between 0 and 1, while the hyperbolic tangent function's output is between -1 and 1. I decided to use the hyperbolic tangent function, because the logistic function has a tendency to incur long training times if its values lie very close to 0, while the hyperbolic tangent function tends to move towards its extreme values more quickly.

The activation function, when applied to 'net' of the neuron then becomes the output of the neuron, feeding through to every neuron in the next layer, repeating the same process until the output layer. This is shown in Figure 4.3 and more closely in Figure 4.4.

The output layer has as many neurons as there are classes. The activation function is applied to the net of each neuron as with previous layers. The neuron with the highest value is taken to be the network's prediction. An example output would look something like:

$$[-0.3, -0.2, -0.1, -0.8, 0.9]$$

There are five classes, and the fifth class has the highest value, thus the neural network has classified the input as belonging to that fifth class. Given that the maximum possible value for an output would be 1, 0.9 shows a large confidence in the classification. -1 shows strong disagreement, and something like 0.3 would show low confidence.

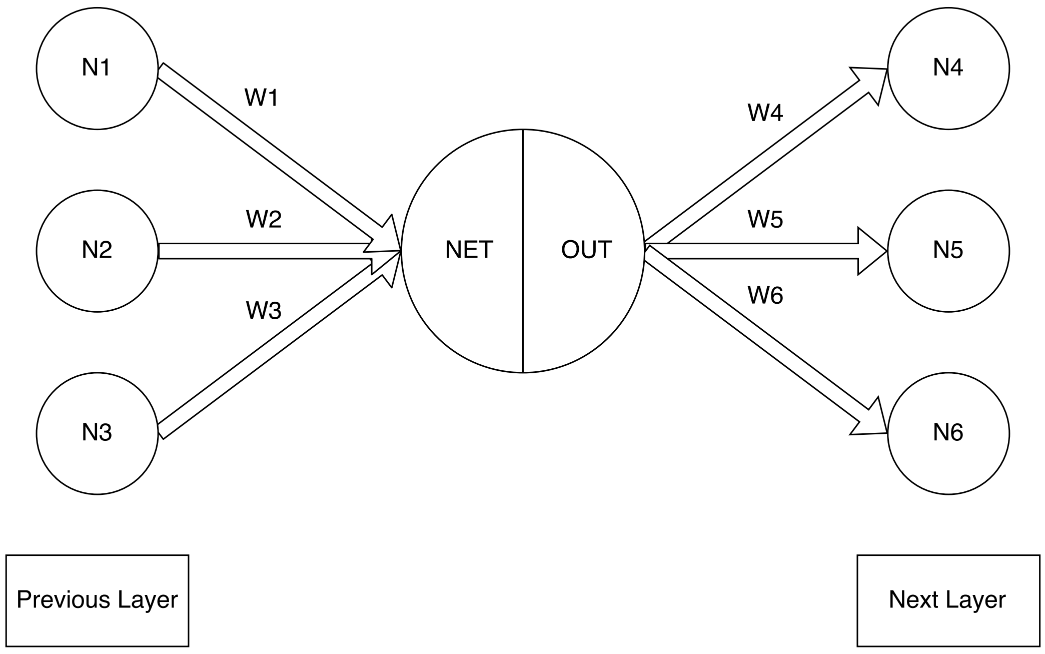

Figure 4.3: A Neuron in the Network

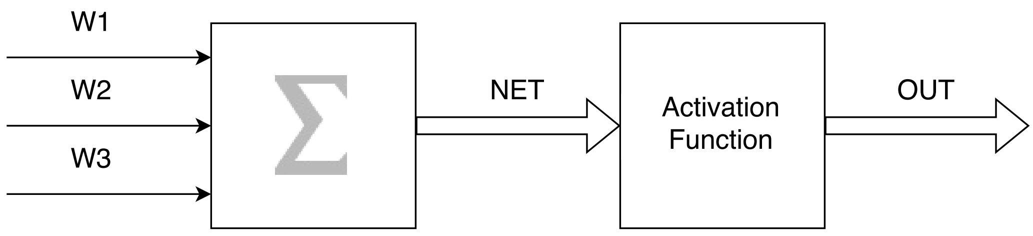

Figure 4.4: The Internal Workings of a Neuron

To train the network, the back-propagation technique is used, using gradient descent as its operational method. I found that when using this method, the results were initially less than satisfactory; the total error would decrease, but the classification rate would not mprove that much, settling at around 27/70 (asdf%). The system appeared to be minimising the total error but not promoting confident answers. To remedy this, I made it so that the output of the network would have its highest value (the value taken as the guess) clamped up to 1. This would increase the error if it was wrong, and have no error if it was right, spurring on the back-propagation algorithm and preventing it from stagnating. This change alone improved the classification rate to 36/70 (asdf%).

20/5 : 27/70, 36/70 40/10: 30/70

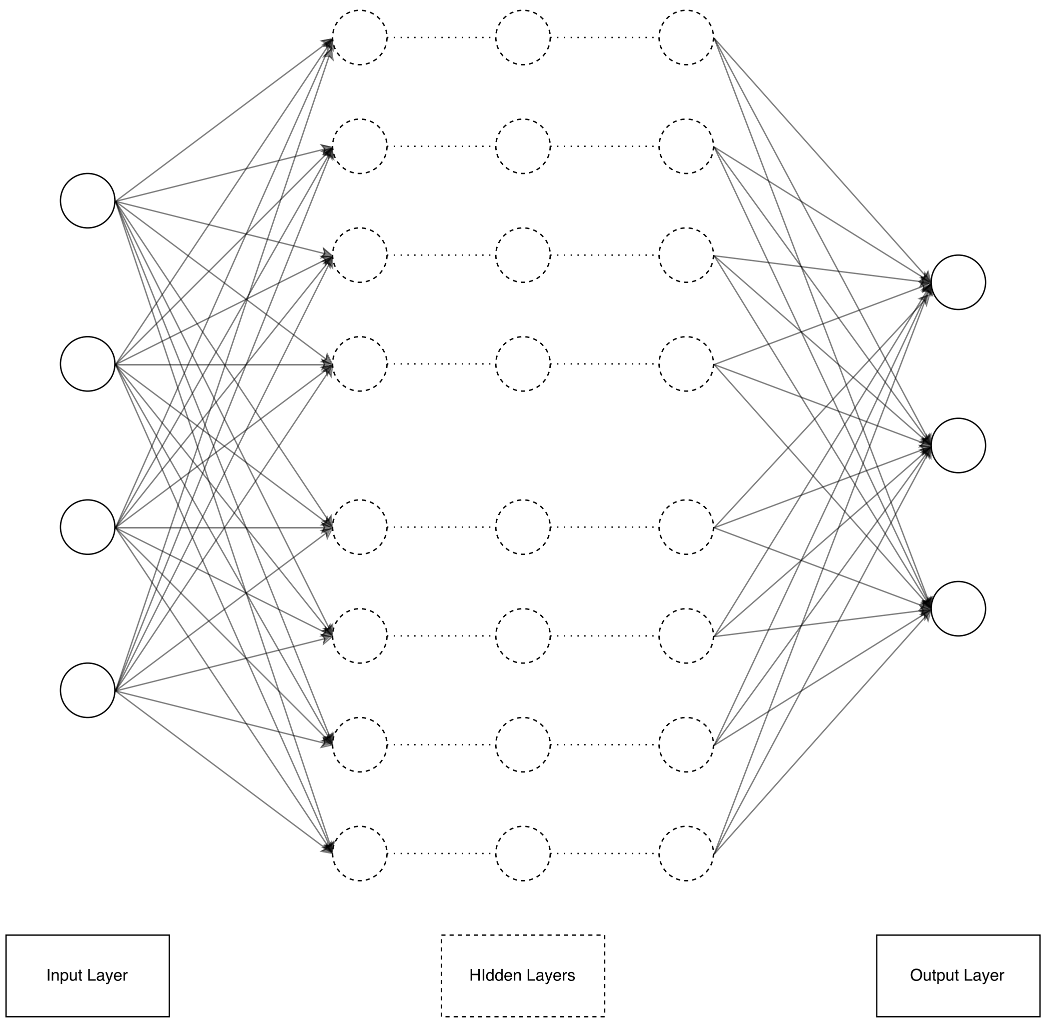

Figure 4.5: Multilayer Perceptron Overview

Algorithm Testing

For each classifier, their effectiveness on the MSTAR dataset must be tested and catalogued, beginning with the Nearest Neighbour implementation. Doing the Nearest Neighbour classification first establishes a benchmark against which further methods can be tested. The naïve nature of the Nearest

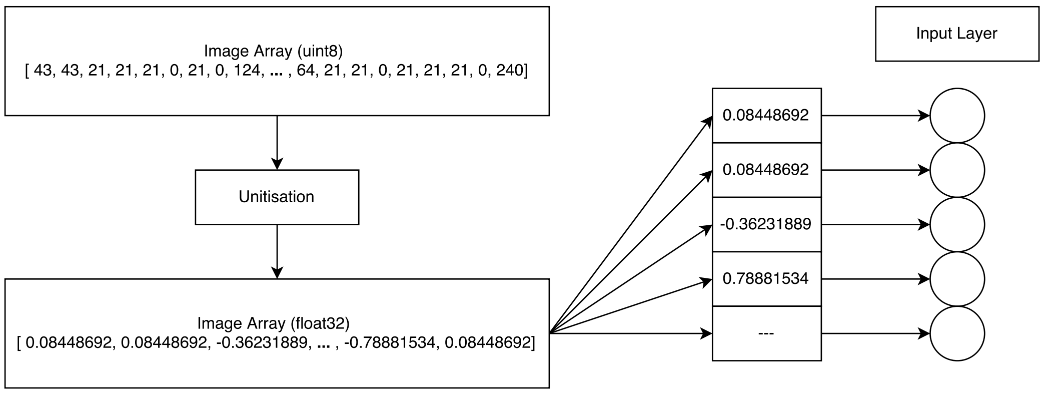

Figure 4.6: Multilayer Perceptron Input Layer

Neighbour classifier means that it is not optimised for any particular dataset. It shows the efficacy of a generic algorithm applied to the MSTAR dataset.

For the effectiveness of any algorithm to be tested, correct and incorrect outputs must be defined. The MSTAR dataset includes target labels in a header section of each file, but since the operations are conducted on files with their header and phase data stripped away it merely adds computational complexity to find the corresponding header for each file and then parse it to extract data about each target. To simplify classification, a variant of a one-hot vector denoting the classes is attached to each target. The vector consists of a series of numbers, equal in length to the number of classes in the dataset. A '1' denotes that the instance is a member of the class corresponding to that entry in the vector, and the rest of the numbers are 0, showing that the instance is not in those classes. A file is created listing all of the filenames to be tested during the run of the algorithm. An example file with ten entries and two classes would look as follows:

```
HB03333.003.tiff 1 0
HB03334.003.tiff 1 0
HB03335.003.tiff 1 0
HB03337.003.tiff 1 0
HB03338.003.tiff 1 0
HB14931.025.tiff 0 1
HB14932.025.tiff 0 1
HB14933.025.tiff 0 1
HB14934.025.tiff 0 1
HB14935.025.tiff 0 1
```

Testing During Development

The simplest way to find a classifier's efficacy is to test it on a wide variety of classes and on as many test instances as possible. To provide interim results, during the iterative phase of classifier development, only a subset of the dataset's images are used. This compromises the final accuracy of the classifier (it may perform differently on the full dataset), but bring with it the ability to test and train classifiers more quickly, due to the lower computational overheads. During the development this testing method allows for simple decisions regarding the direction of classifier implementation or optimisation to be made. In process of verifying the classifiers, the full dataset must be used to give an accurate picture of the classifier's performance and real-world implementation.

4.4.2 Subsystem Verification

The chosen method for verifying a classifier's efficacy once it is considered to be sufficiently optimised is simple; The classifier is tested on the dataset with a number of test instances; if time allows, leave-one-out cross-correlation (LOOCV) covered in section 2.6.2 will be used. This entails removing one instance from the dataset, training the classifier on the remaining points, and using the removed instance as a test case. Once this has been done, the instance is replaced, another is taken, and the process is repeated until every instance in the dataset has been tested. The system's performance is based on the number of correct classifications made during the process, and also gives a good estimate of the Separability Index of the data (section 2.6.1.

4.4.3 System Verification and Deployment

To confirm the results of each classifier, it is important to have a set of training instances on which the system can be trained. The system is then tested on another set of points whose classes are known. Once this has been tested and confirmed to have a desirable level of classification accuracy, the system will be ready for testing on previously unseen, real-world cases.

This is accomplished by dividing a set of known data points into a training set, and a testing set; for example. 90% of the data points will be used to train the system, which will then be tested on the remaining 10%. The process of cross-validation entails selecting a different training/test split each time (either systematically or at random) and performing the process again. This concept can be extended to where the system is trained on all but one of the instances and then tested against it, which is known as LOOCV (”Leave One Out” Cross-Validation). The system is tweaked until it reaches the level of classification required. Cross-Validation is an important tool for eliminating ”overfitting” of the system to the training data. Mixing up the training and test cases ensures that the classifier is left with some ability to generalise, and not just repeat what it has been shown.

4.4.4 System Validation

4.4.5 Operations and Maintenance

4.4.6 Changes and Upgrades

4.4.7 Retirement / Replacement

Chapter 5

Results

5.1 KNN Results

The KNN algorithm is simple to execute, but scales poorly with dataset size and image size. Using a dataset of 1291 images, of size 100x100, the largest bottleneck in the procedure is the generation of the squared distance matrix D2, taking 182 minutes (3 hours and 2 minutes) to complete. This was then saved to prevent the need to recalculate it in future. The time taken to calculate the optimal value of K is also proportional to the number of images, taking 379 minutes (6 hours and 19 minutes) when considering all odd values of K up to and including 1291. Plotting these K-values vs the training error obtained for each value is shown in Figure 5.1. The time taken to classify a single instance is 0.25m (15s). Testing though the full set of 1291 to obtain the training classification accuracy takes $\tilde{1}$291*0.25 minutes, resulting in a real-world time of 342 minutes (5 hours and 42 minutes).

5.2 Multilayer Perceptron Initial Results

The results in this section use an input image of 100x100px. The image is cropped around the center of the image. For some images smaller than this size (e.g. the SLICY set at 54x54px) the images are padded with zeroes to

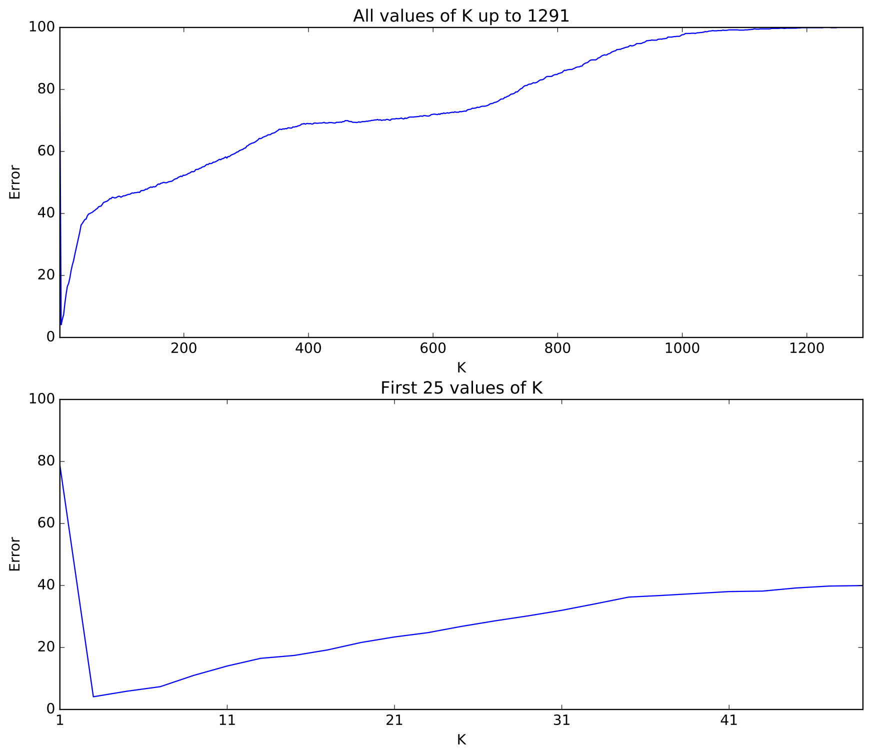

Figure 5.1: Optimising for K

ensure a consistent input image size. The multilayer perceptron has two hidden layers, of 30 and 20 neurons, respectively. The resulting structure is thus 10,000 input neurons feeding into 30, feeding into 20, feeding into the five output neurons. Back-propagation through all layers is applied after every forward run, which has effects on the output that can be seen in 5.2.1. The cost function is the square of the error. The system is trained for 10,000 iterations, and settles after approximately 300. The input images are taken in order (i.e. 195 BTR 60 images, then 274 2S1 images, etc.) per iteration. The layout of the input data is shown in Table I.

5.2.1 Analysis

The final values of the system lie at 90.6% classification accuracy (1170/1291 correct). The final confusion matrix (Table IV) shows a clear pattern; the incorrect predictions are always to the class preceding the correct class. Figure 5.2 shows the average cost decreasing over time, while Figure 5.3 shows how the actual cost sharply increases at the start of a new class and decreases swiftly, before rising again at the beginning of the next class. This appears to be due to the layout of the input data. For subsequent development I need to investigate using batches of inputs to reduce this. Reducing the number of consecutive inputs of the same class should force the network to detect underlying features and mitigate this error.

5.2.2 Performance Metrics

Class	**No. Images**
BTR 60	195
2S1	274
BRDM 2	274
D7	274
SLICY	274

Table I: Input Data

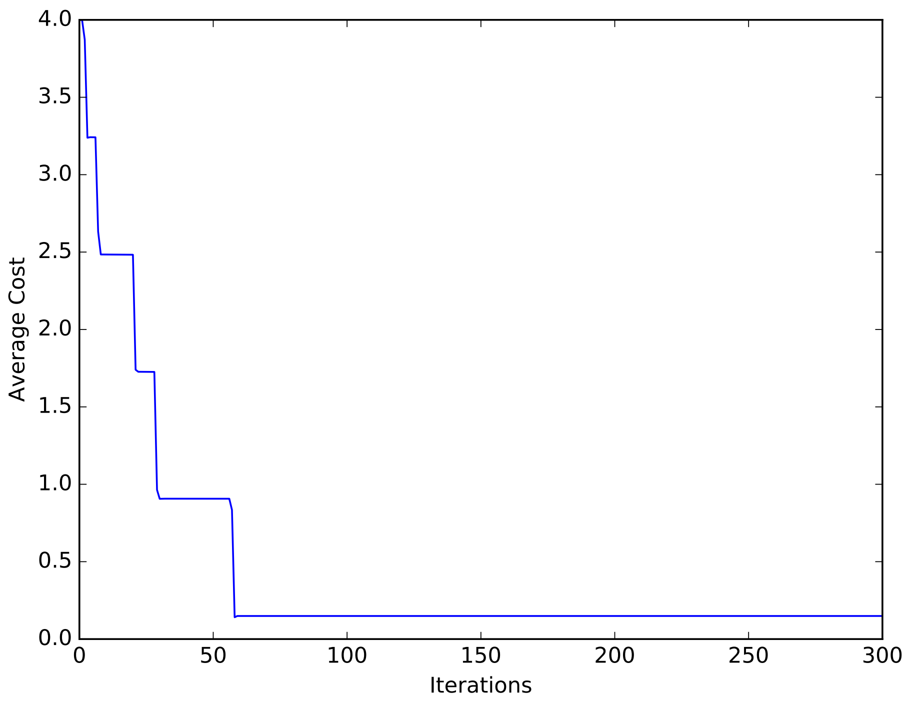

Figure 5.2: Average Cost Over Time

		Predicted Class				
		BTR 60	2S1	BRDM 2	D7	SLICY
Actual Class	BTR 60	0	195	0	0	0
	2S1	0	274	0	0	0
	BRDM 2	0	274	0	0	0
	D7	0	274	0	0	0
	SLICY	0	274	0	0	0

Table II: Initial Confusion Matrix (Before Training)

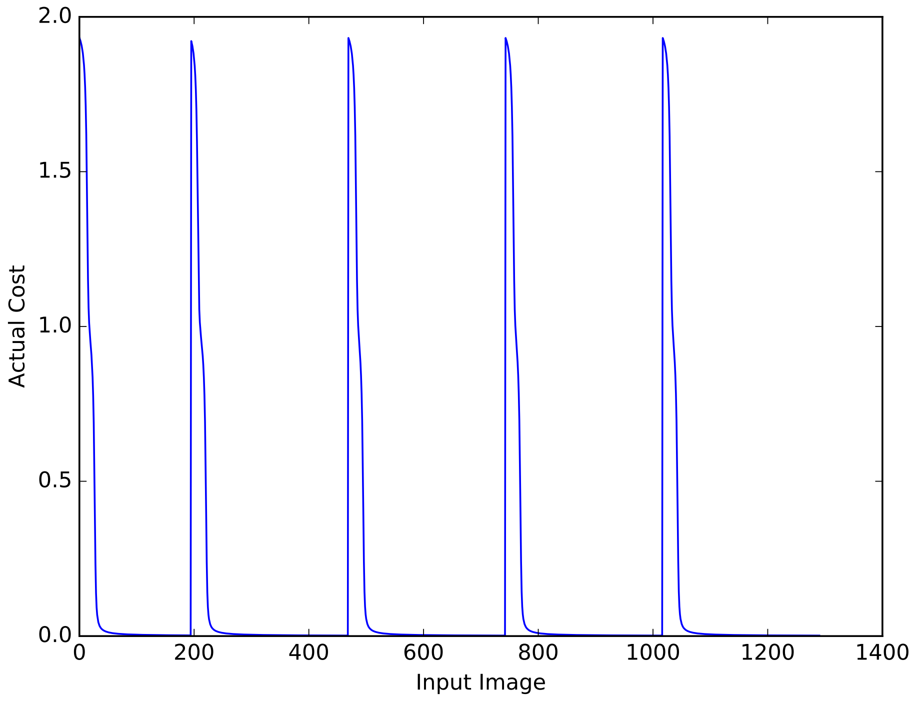

Figure 5.3: Actual Cost Within an Iteration

		Predicted Class				
		BTR 60	2S1	BRDM 2	D7	SLICY
Actual Class	BTR 60	170	0	0	0	25
	2S1	23	251	0	0	0
	BRDM 2	0	24	250	0	0
	D7	0	0	24	250	0
	SLICY	0	0	0	24	250

Table III: Transitional Confusion Matrix (Before Final)

		Predicted Class				
		BTR 60	2S1	BRDM 2	D7	SLICY
Actual Class	BTR 60	170	0	0	0	25
	2S1	24	250	0	0	0
	BRDM 2	0	24	250	0	0
	D7	0	0	24	250	0
	SLICY	0	0	0	24	250

Table IV: Final Confusion Matrix

Chapter 6

Conclusion

References

[1] R. Schumacher and K. Rosenbach, “Atr of battlefield targets by sar classification results using the public mstar dataset compared with a dataset by qinetiq, uk summary.”

[2] L. Wang, Y. Zhang, and J. Feng, “On the euclidean distance of images,” Ph.D. dissertation, School of Electronics Engineering and Computer Science, Peking University, 2005. [Online]. Available: http://www.cis.pku.edu.cn/faculty/vision/wangliwei/pdf/IMED.pdf

[3] D. Michie, D. J. Spiegelhalter, and C. Taylor, “Machine learning, neural and statistical classification,” 1994.

[4] I. Russel, “Definition of a neural network,” 1996. [Online]. Available: http://uhaweb.hartford.edu/compsci/neural-networks-definition.html

Appendix A

Progress Report

As of the 16th of September, certain progress has been made in three areas of the report, namely the Literature Review, Methodology, and Implementation.

A.1 Literature Review

The Literature Review has been somewhat written; the core topics of most relevance to the study have been covered, but the level of detail for each topic varies, and I feel it may be insufficient. This is a section that I feel needs further refinement and content, that will be added as the development of this study touches on more topics.

A.2 Implementation

I have implemented a working nearest neighbour classification system, but it is quite slow. I have implemented the first few stages of a multilayer perceptron with two hidden layers, and am currently busy with the implementation of a back-propagation algorithm to optimise the network.

A.3 Areas of Focus

The next avenue of development is to process the MSTAR image chip data to facilitate quicker classification. Until now I have been using only a subset of the MSTAR data (70 image chips), but this is not feasible in the final product, as it does not include sufficient training data. The time taken to process this subset is non-negligible, and something must be done to improve the training time. The idea is to process the images in such a way that the resolution of each image, and therefore the number of pixels to process, can be reduced while retaining all the crucial data contained in each image. This will include removing the clutter surrounding each target, and may involve clustering algorithms to preserve important areas of data.

I also need to finish my implementation of the multilayer perceptron, specifically the application of the back-propagation algorithm to optimise the network.