Pattern Recognition and the Nondeterminable Affine Parameter Problem

By Nathan Geffen

November, 1998

Submitted to the Departments of Computer Science and Geomatics in fulfilment of the requirements of the degree of Master of Science.

Declaration

I hereby declare that this thesis is my original work and has not been submitted in any form to another university.

Nathan Geffen

June, 1998

Acknowledgements

I am particularly grateful to my supervisor, Scott Mason, for his advice throughout the last two years and for the effort he put into reviewing this dissertation and its multiple drafts. I am also particularly grateful to my other supervisor, Mike Linck, for his assistance in making corrections to this thesis.

I would like to acknowledge the assistance and advice of the following people: Faizel Slamang, Val Atkinson, Kenneth Goodman, Daphne Barends, Heinz Ruther, Julian Smit, Manos Baltsavias, Malcolm Dingle, John Greene, Ben Herbst, Dirk Stallman, Hardy Hulley, Peter Wood, Sonia Berman and Edwin Blake.

I am also grateful to the Foundation for Research and Development for funding this research project.

Table of Contents

List of Figures

List of Tables

Glossary

Adaptive Least Squares Matching (ALSM)	An image matching algorithm developed, primarily, by Gruen (1985).
Artificial Neural Network (ANN)	A computational device, loosely based on the networks of neurons in biological brains, which is particularly useful for solving pattern recognition problems.
Digital Surface Model (DSM)	A spatial model on a digital computer that holds information describing a surface (e.g. a geographical landscape) using 3-dimensional co-ordinates.
Error Back-Propagation (EBP)	A learning algorithm for **Multilayered Perceptrons**, developed independently by various researchers (Werbos, 1974; Parker, 1985; LeCun, 1985; Rumelhart, Hinton and Williams, 1986).
k-Nearest Neighbour	A **classification algorithm**, which assigns previously unclassified patterns to the class of their nearest neighbour.
Learning Vector Quantization (LVQ)	A learning algorithm for a particular type of **artificial neural network** (known as a lattice structure) developed by Kohonen (1986).

Multilayered Perceptron (MLP) An **artificial neural network** consisting of at least three layers of neurons. Neurons in each layer, except the last, transmit weighted signals to the neurons in subsequent layers. The first layer is called the input layer, the last layer is called the output layer and all intermediate layers are called hidden layers. The term Perceptron was coined by Rosenblatt (1958).

Multi-photo Geometric Constraints (MPGC) An enhancement to **Adaptive Least Squares Matching** developed by Baltsavias (1991).

Patch A patch (or sub-image) is an m x n pixel sub-image within a much larger image. In this work, only 9 x 9 pixel patches are considered. Each pixel is a greyscale value in the range [0 .. 255].

Pattern Classification Model (Pattern Classification Algorithm) In this thesis, a model or algorithm which takes a pattern as input and assigns it to a class. It is one of the main components of a **pattern recognition system**. Generally, the algorithm has two stages (or modes), a training mode in which its parameters are estimated and a classification phase in which it allocates a pattern to a class. Examples of pattern classification models are **artificial neural networks** and the **k-Nearest Neighbour** algorithm.

Pattern Recognition System In this thesis, a system which consists of two main components, a set of **pre-processing operators** and a **pattern classification model**. The system takes a pattern as input, pre-processes it and assigns it to a pre-defined class.

Picture Image — In image matching this is the image on which corresponding points must be found for the images on the **template image**. There are often multiple picture images for a particular image matching problem. In this dissertation, the sub-images of the picture images are referred to as *picture patches*.

Pre-processing Operator — This is a function which transforms a pattern with the purpose of extracting, selecting or accentuating the pattern features.

Resilient-Propagation (RPROP) — A learning algorithm for **Multilayered Perceptrons** developed by Riedmiller and Braun (1993).

Template Image — In image matching this is the image for which corresponding points must be found on other images. In this research the images which must be recognised all come from template images. In practice the template image is usually divided into a set of overlapping small sub-images called patches. In this dissertation, the sub-images of the template image are referred to as *template patches*.

Abstract

This thesis reports on the process of implementing pattern recognition systems using classification models such as artificial neural networks (ANNs) and algorithms whose theoretical foundations come from statistics[1]. The issues involved in implementing several classification models and pre-processing operators — that are applied to patterns before classification takes place — are discussed and a methodology that is commonly used in developing pattern recognition systems is described. In addition, a number of pattern recognition systems for two image recognition problems that occur in the field of image matching have been developed. These image recognition problems and the issues involved in solving them are described in detail. Numerous experiments were carried out to test the accuracy and speed of the systems developed to solve these problems. These experiments and their results are also discussed.

A typical pattern recognition problem requires that a pattern drawn from a particular domain be classified into one of a finite number of classes. Examples of this include assigning a hand-written character to one of the letters in the alphabet (character recognition), assigning a speech utterance by a human speaker to a particular word in the speaker's language (speech recognition), or assigning an image to one of several categories based on the shape of the image (as in the problems in this research). A pattern recognition system is typically comprised of two main components, a set of pre-processing operators and a classification model such as an ANN or a statistical pattern recognition algorithm[2]. In order to develop such a system the following steps are usually carried out: (1) a large number of sample patterns, for which the classes to

---

[1] Both of these are known as decision-theoretic models.

[2] In order to avoid confusion, the term *pattern recognition system* is used to refer to the entire system containing both the pre-processing operators and the classification model. The term *classification model* (or *classification algorithm)* refers to the part of the pattern recognition system which is responsible for making the classification (e.g. an ANN or the k-Nearest Neighbour algorithm).

which they belong are already known are collected, (2) the sample patterns are split into training and test data sets, (3) the training data set is used to adapt the parameters of the pattern recognition system and (4) the system is tested for its accuracy on the test data set.

This first part of this thesis examines this process, each of its steps, the problems that arise in them, several pre-processing algorithms and various pattern classification models. The latter include minimum distance classifiers and the k-Nearest Neighbour (k-NN) algorithm (Cover and Hart, 1967), whose theoretical foundations lie in statistics, and ANN learning algorithms such as Error Back-propagation (Rumelhart, Hinton and Williams, 1986b) Resilient-propagation (Riedmiller and Braun, 1993) and Learning Vector Quantization (LVQ) (Kohonen, 1986).

The remainder of this thesis describes two image recognition problems, the experiments conducted to try and solve them using the algorithms described in the first part of this thesis and an analysis of the results of these experiments. The two image recognition problems provided a means for testing the researched methodology and algorithms in a practical domain. Both problems (described briefly below) come from the field of digital photogrammetry and, in particular, image matching. However, the data images for the first problem were developed artificially, while the data images for the second problem were acquired from digital photographs of real objects. The pattern recognition systems developed to recognise the artificial patterns were generally successful, with some of them achieving a recognition rate better than 90%. However, the systems developed to recognise the real data patterns were not successful, achieving results insufficiently reliable for industry.

Image matching plays a critical role in digital photogrammetry, particularly where it is necessary to construct a three-dimensional mapping of a terrain from two-dimensional images, or what is known as a Digital Surface Model (DSM). Multiple digital images from different perspectives of the surface are acquired. Then corresponding points on each image are matched. Once the matching process is completed, rigorous photogrammetry techniques are used to recover the depth dimension of the surface. However, the process of matching corresponding points is complex and not as well understood as the other steps in this process.

One of the most significant developments in digital image matching is an algorithm described by Gruen (1985) called Adaptive Least Squares Matching (ALSM) which is an area-based matching technique. Given two digital images, it — as do all area-based matching algorithms — divides one of the images, which is referred to in this thesis as the template, into small regions called patches (also referred to as sub-images in this dissertation), which may overlap and differ in size. For each template patch the approximate position of the corresponding patch on the second image, referred to in this thesis as the picture, is estimated. Using an affine transformation ALSM adapts the picture patch in order to determine its correct position more accurately. This is done by modelling the parameters of the affine transformation as a least squares observation equation and then minimising this equation.

Baltsavias (1991) enhanced this technique by showing how *a priori* geometric knowledge can be used to determine the first approximation of the picture patch and by modifying the algorithm to work for multiple (i.e. more than two) images. Critically, from the point of view of this work, Baltsavias also pointed out that for various types of patches the texture of the patch does not support an affine transformation. In these cases the patch is said to have nondeterminable affine parameters.

The aim of the pattern recognition systems developed in this work is to recognise image patches with nondeterminable affine parameters. Successfully developing such a system would enable the future development of modifications to the matching process that would improve the handling of these sub-images. For instance the pattern recognition system could act as an interest operator, selecting points that are likely to be matched before the matching process actually starts, thereby saving processing time and improving the accuracy of the matching process. ANNs such as multilayered perceptrons trained with Error Back-propagation and Kohonen's LVQ model, as well classification algorithms such as the k-NN have been used successfully as solutions to multitudes of real-world pattern recognition problems. It was therefore decided to use them as part of the recognition systems that attempt to recognise images with nondeterminable affine parameters.

The nondeterminable affine parameter problem has been recast as two problems in this thesis: an artificial problem (using artificially generated images) and a real-world

problem using data extracted from digital photographs. Reasons for having done this include:

- It was clear from an early stage of this research that, based on poor results in informally conducted experiments, achieving successful results using the real-world data was unlikely. It is one of the aims of this research to learn how various pattern recognition systems work and it would not be feasible to do so on a problem which cannot be solved. The idealisation of the images comprising the artificial data set implies that they are not likely to suffer as much from problems associated with real data (such as too much noise and errors resulting from data collection methods). It is also easier to determine what sort of image processing techniques are likely to be usefully applied to the idealised artificial images as opposed to real images, since the former are well understood. Developing a system to recognise artificial images is, therefore, an easier task than developing one to recognise real images and therefore afforded a better opportunity to study and compare different classifiers and image processing techniques.

- The patterns in the artificial data set share many of the characteristics of popular toy problems used as benchmark tests in ANN literature, such as detecting parity and separating two intertwining spirals: generally, they are easily recognised by humans (and therefore there is little ambiguity in their category assignments), clearly defined and easily reproduced.

That the pattern recognition systems performed well on the artificial problem but poorly on the real-world problem demonstrates the difference in complexity between solving real-world pattern recognition problems and artificial problems. The primary reasons for the poor results with the real-world image patches are: (1) the samples collected do not adequately represent the extremely large pattern space of the nondeterminable affine parameter problem; (2) many of the samples might have been incorrectly classified in the process of collecting them; (3) the implemented pre-processing operators are not sophisticated enough to select and extract the essential features for partitioning the patterns into classes and; (4) the patch size used in this thesis is too small (each patch is a 9x9 grid of greyscale pixels), resulting in patches that do not contain enough texture.

The results of this research indicate that using standard decision-theoretic pattern recognition methods to recognise real-world patches with nondeterminable affine parameters is complex and might not be possible. However, suggestions for improving the methods used in this research are discussed in the last chapter. In addition, other methods, such as those described in Baltsavias (1991) (briefly summarised in this dissertation), should also be investigated further.

In contrast to the real-world images[3], the images in the artificially created data set were, for the most part, recognised successfully. A positive result of this research is that this data set can be used to benchmark pattern recognition systems. It has been made available for public access on the Internet[4].

---

[3]The real-world images were comprised from photographs of a propeller, a rhinoceros footprint, a prehistoric hominoid footprint, a cheetah paw and two rock images from mines, which constitutes a diverse range of short-range photogrammetry domains. The set might be improved by using images taken using aerial photography, such as images of towns.

[4] The artificial data set can be downloaded from ftp://foxbat.sur.uct.za.za/pub/data/art.txt.gz.

1. Introduction

1.1. Objectives

The primary objectives of this research project are to:

- Demonstrate an understanding of the process of implementing a pattern recognition system.
- Demonstrate an understanding of several ANN and statistical pattern classification models, in particular the issues that must be overcome in order to implement and test them.
- Implement, or acquire implementations of, these recognition models and, by means of setting up experiments, use them to solve the two nondeterminable affine parameter image recognition problems.
- Analyse the results of these experiments.

Secondary objectives of this research project are to:

- Demonstrate an understanding of some of the issues regarding pattern recognition of image data.
- Compare the performance of some of the more popular ANN models (such as Learning Vector Quantization and multilayered perceptrons (MLPs) trained with Error Back-propagation) to the performance of popular nonparametric statistical pattern classification techniques (such as the k-NN algorithm).
- Demonstrate an understanding of the fundamentals of ALSM.
- Demonstrate an understanding of the nondeterminable affine parameter problem.
- Develop pattern classification techniques and software for application in the Department of Geomatics at the University of Cape Town (UCT).

The nondeterminable affine parameter problem has been used to provide a means for testing the implemented pattern recognition systems. Implementing a fit-for-production solution to it is not an aim of this thesis.

1.2. Experimental Procedure

Based on the above objectives, the following experimental procedure has been carried out:

- Measurements against which the implemented classifiers can be compared and evaluated have been identified.
- Data pre-processing operations, useful for solving the nondeterminable affine parameter problems, have been implemented by the author.
- Several pattern classification models have been implemented by the author or obtained via the Internet.
- Two data sets, one containing sub-images taken from digital photographs and the other containing artificially constructed sub-images, have been constructed.
- Image patches in the data sets have been classified using pattern recognition systems. In this thesis these systems consist of a set of data pre-processing operations and a pattern classification model.
- The performance of the systems has been analysed in terms of the measurement criteria.

This dissertation describes in detail each of the steps in this experimental procedure.

1.3. Scope and Limitations

The scope of this research has been limited in a number of ways in order to reduce the time needed to complete this research, as well as its complexity.

- Only 9x9 greyscale sub-images (or patches) have been used. In practice the size of the template patches can vary considerably, from 7x7 to 31x31 greyscale patches. The decision to use 9x9 greyscale patches is not an

arbitrary one, since this size is frequently used in UCT Department of Geomatics image matching applications.

- No effort has been made to integrate one of the implemented pattern recognition systems into an implementation of ALSM.

- Biological issues concerning ANNs, although briefly mentioned in the text, are not considered in any depth, even though they are an area of active investigation in ANN literature.

- Certain practical hardware constraints were placed on this research as a result of financial limitations. All experiments have been conducted on a Pentium Processor with 92 MB of RAM running at 166MHz under the Linux Operating System. No specialised ANN architecture has been used. In addition it has been impossible to conduct all the experiments without any system load, since the machine often has to be used by other students at the same time. Therefore the timing measurements of the experiments have not been particularly accurate. Owing to software limitations (much of the software was supplied by third parties), it has not been possible to overcome this by measuring CPU cycles.

- A significant problem in this research has been to create an unbiased large data set of pre-classified sub-images for training and testing the systems. A program written by Julian Smit of the University of Cape Town's Department of Geomatics, as part of his PhD thesis, has been the primary tool used by the author to generate real-world sub-images. However, it is not the primary aim of Smit's program (referred to as MatchProg in the remainder of this dissertation) to generate data sets of images with nondeterminable affine parameters. Thus it is probable that there were a significant number of incorrectly classified images in the real sub-image data set. Towards the completion of this thesis a program developed by ETH in Zurich, called PVD, was brought to the author's attention. It might be a better tool for generating correctly pre-classified sub-images, but a substantial amount of work by the authors of PVD would be

necessary for it to automatically generate large numbers of sub-images. It was therefore decided not to use PVD for this purpose.

- There is a bias in this research towards issues dealing with ANNs as opposed to traditional statistical pattern classification models. This is because the author's original focus was almost exclusively on ANNs. Only comparatively recently has the author investigated statistical pattern classification models. This bias is a result, primarily, of the author's computer science background. The author, however, has no intention of implying that ANNs are a better, or more important, model of pattern classification than statistical methods.

- There are numerous sub-fields within pattern recognition and many pattern recognition algorithms and pre-processing operators. For practical purposes only a few have been researched in detail. Syntactic pattern recognition models are not explored. This is not because the methods of syntactic pattern recognition do not offer potential solutions to the nondeterminable affine parameter problem, but because of time and space constraints. A number of important models, such as Radial Basis Function Networks and Parzen Window classifiers, have been left out because of time constraints. Many informal preliminary experiments were conducted with a number of ANN models, as well as an algorithm developed by the author. Only models that performed well on these initial tests were considered[5]. In addition, some experiments noted in Appendix A were conducted with the Scaled Conjugate Gradient learning algorithm for MLPs (Moller, 1993), but are not discussed in the text. The selected models, and the reasons for their selection, follow:

---

[5] The author's algorithm was eliminated on this basis, as well as a number of other ANN learning algorithms, such as Quickprop (Fahlmann, 1988) and Cascade-correlation (Fahlman. and Lebiere, 1990). It is quite probable that the latter algorithms would have performed better if more effort had been put into tweaking their parameters.

- *Minimum distance classifiers:* These are among the simplest pattern recognition algorithms to understand and implement. They are also very fast and, as the results of tests on the artificial data set indicate, they sometimes provide acceptable solutions.

- *K-NN Rule:* This is a very popular classification technique with a solid theoretical basis in statistics. Cover and Hart (1967) demonstrated that this algorithm performs with an accuracy at most twice the Bayes' error (the meaning of this is discussed in Chapter 2). It is also common in ANN literature to compare the performance of ANNs against the performance of the k-NN algorithm and its variations (e.g. Weidemann et. al., 1995 and Holmstrom et. al., 1997).

- *Multilayered Perceptrons (MLPs):* This class of ANNs is probably the most studied and most widely used in practice. The well-understood Error Back-propagation algorithm (Rumelhart *et al.*, 1986a and Rumelhart *et al.*, 1986b) and various enhancements to it (described in Sarkar, 1995) are discussed. The Error Back-propagation model has been selected because it has a strong claim to be the standard ANN model, (it is usually the only ANN model discussed in popular journals, such as *Scientific American*, e.g. Hinton, 1992). In addition to the Error Back-propagation algorithms, the Resilient-propagation algorithm (Riedmiller and Braun, 1993) is considered because it produced promising results during informal preliminary experiments on the nondeterminable affine parameter problem.

- *ANN trained with LVQ:* This algorithm was developed by Kohonen (1986). It is appealing for a number of reasons: it is simple to understand (in contrast to the Error Back-propagation model, this is a linear model), it is popular in ANN literature and a free, well-documented and excellent implementation of this algorithm developed by Kohonen *et al.* (1992) is available on the Internet.

- Baltsavias (1991) suggests a number of methods for identifying the determinability of the affine parameters. These are briefly mentioned in this dissertation but not explored in detail, since they are specific to the nondeterminable affine parameter problem and are unrelated to decision-theoretic pattern recognition, with which this research is concerned.

- In correspondence from Baltsavias he comments that the determinability of the affine parameters can be effected by factors other than the image content. He also suggests that these factors should be investigated as part of developing a successful pattern recognition system for the nondeterminable affine parameter problem. However, this would necessitate research outside of pattern recognition and has not been included in this research.

1.4. Organisation of this Dissertation

Chapter Two describes the fundamentals of pattern recognition, including a commonly employed methodology for developing pattern recognition systems and Bayes' Rule, which is critical to statistical decision theory. It also discusses the fundamentals of ANNs and issues that need to be considered when implementing them (with emphasis on those issues relevant to the problems investigated in this research). In addition criteria for evaluating pattern recognition systems are discussed. Chapter Three examines the pre-processing operators (feature extractors and selectors) used in this research. Chapter Four describes the pattern classification models implemented in this research. Chapter Five describes the nondeterminable affine parameter problem. Many of the issues discussed in Chapter Two are examined in the context of this problem. Baltsavias's suggestions for solving the nondeterminable affine parameter problem are briefly outlined and some papers which deal with similar image recognition problems are discussed. The chapter also describes how the artificial and real sub-image data have been collected. Chapter Six discusses the experiments conducted and what their results imply regarding the performance of the recognition systems. Reasons for the large difference in success between the artificial and real image recognition problems are also discussed. Conclusions are given in Chapter Seven. Appendix A contains details regarding the experiments conducted.

2. Pattern Recognition Fundamentals

2.1. Introduction

This chapter describes pattern recognition and the process of developing a pattern recognition system. A very brief historical overview is given of ANNs, which are then formally defined and described. Some common problems and issues that occur in pattern recognition are discussed[6]. Measures with which the usefulness, reliability and efficiency of pattern recognition systems can be evaluated are also described.

Pattern classification is traditionally divided into two types, decision-theoretic methods (which include statistical algorithms and ANNs) and syntactical methods. A combination of both methods can be used to solve many practical problems, although there are some extreme problems which can be solved using only one or the other of these approaches (Bow, 1984). Syntactic pattern recognition models divide patterns into sub-patterns (pattern primitives). The relationships of the pattern primitives to each other is described as a hierarchy analogous to the syntactic structure of language[7]. Typical applications of syntactic pattern recognition methods include computer-language parsing, natural-language parsing and chromosome recognition. However, syntactic methods can also be used for image processing problems (Gonzalez and Thomasan, 1978, Tou and Gonzalez, 1974). In the decision-theoretic approach a set of measurements, called features, are extracted from patterns. 'The assignment of each pattern to a class is made by partitioning the feature space' (Fu, 1976, p. 2).

Within both these fields, there are a multitude of classification models. The mathematics, journals and researchers involved in these fields are also generally

---

[6]Not all of the some important problems that occur in pattern recognition are mentioned, only those particularly relevant to the nondeterminable affine parameter problem.

[7] The parsing stage of a compiler represents a familiar example of syntactic pattern recognition to computer scientists. The language elements are represented using BNF and a program is classified either as syntactically correct or incorrect by a top-down (e.g. recursive descent) or bottom-up (such as the table driven methods used by yacc) parser.

different. Since, the author was primarily interested in ANNs — which falls into the decision-theoretic category — only models in the decision-theoretic category have been considered. Besides, it is arguable that most image recognition tasks for which there is little *a priori* knowledge, such as the nondeterminable affine parameter problem, are best tackled using decision-theoretic models, because syntactic methods often require the definition of pattern primitives which are, to some extent, problem specific.

2.2. Decision-theoretic Methods and Bayes' Rule

The following definitions and explanations are based on Fukunaga (1990), Bow (1984), Tou and Gonzalez (1974), Duda and Hart (1974), Fu (1976) and Meisel (1972).

A pattern can be expressed as a feature vector $\mathbf{x}$ of dimension n, such that $\mathbf{x} = (x_1, x_2, ..., x_n)^T$, where x_i is a real-valued measurement of feature i.

In a pattern recognition problem each pattern in the pattern space belongs to one class, ω. Given M classes, $\omega_1 \ldots \omega_M$, and a pattern, $\mathbf{x}$, a pattern recognition system allocates $\mathbf{x}$ to one of the M classes. Generally, a pattern recognition system consists of pre-processing operators and a classification model; the former transform the pattern vectors, with the intention of accentuating the features of the patterns that determine to which class they belong, while the latter takes a transformed pattern vector as input and allocates it to one of the classes.

In decision-theoretic models the classification model consists of a set of M decision functions, $d_1(\mathbf{x}), d_2(\mathbf{x}), ..., d_M(\mathbf{x})$ — one for each class — such that if pattern $\mathbf{x}$ belongs to class ω_i, then

$$d_i(\mathbf{x}) > d_j(\mathbf{x}), \quad j = 1,2, ..., M, \; j \neq i. \qquad \textbf{Eq. 2-1}$$

In this work the decision functions are real, single-valued and non-linear.

It can be shown that applying Bayes' Rule is the statistically optimal classifier. By optimal it is meant that the class chosen by this classifier is the most probable. In the long run it will perform more accurately than any other classifier. However, in order to apply Bayes' Rule, the probability of occurrence of each class, $P(\omega_i)$, and the

probability density function of the patterns, $p(\mathbf{x}/\omega_i)$, must be known. $P(\omega_i)$ and $p(\mathbf{x}/\omega_i)$ are known as the *a priori* information.

Assuming that no loss is incurred for a correct decision and that the loss for all incorrect decisions is the same[8] the following set of decision functions is obtained:

$$d_i(\mathbf{x}) = p(\mathbf{x}/\omega_i)p(\omega_i),\ i = 1, 2, ..., M. \qquad \textbf{Eq. 2-2}$$

Bayes' formula for the distribution of the classes, given **x**, (the *a posteriori* distribution) is

$$p(\omega_i/\mathbf{x}) = \frac{p(\omega_i)p(\mathbf{x}/\omega_i)}{p(\mathbf{x})} \qquad \textbf{Eq. 2-3}$$

Substituting Eq. 2-3 into Eq. 2-2, the set of decision rules

$$d_i(\mathbf{x}) = p(\omega_i/\mathbf{x})p(\mathbf{x}),\ i = 1, 2, ..., M \qquad \textbf{Eq. 2-4}$$

is obtained.

Since $p(\mathbf{x})$ does not depend on i, equation Eq. 2-4 can be modified so that,

$$d_i(\mathbf{x}) = p(\omega_i/\mathbf{x}). \qquad \textbf{Eq. 2-5}$$

Bayes' Rule consists of selecting the class associated with the maximum decision function from the set defined by Eq. 2-5, for a particular pattern, **x** (i.e. Eq. 2-1 is applied to Eq. 2-5).

In practice the *a priori* information can seldom be determined unless there are very few features in the pattern vector or the features of a specific problem have a known distribution (e.g. Bayes' Rule is used in speech recognition in Massaro and Stork, 1998). Therefore, most pattern classifiers use nonparametric techniques (i.e. they make no assumptions about the distributions of the classes or the pattern vectors) to estimate Eq. 2-5. The accuracy of a good nonparametric classifier should approach the accuracy of Bayes' Classifier as the number of samples increases (Fukunaga, 1990). The classifiers implemented in this research, ANNs, minimum distance classifiers and k-NN, are nonparametric methods.

---

[8] This assumption simplifies the mathematical derivation of Bayes' rule. In addition it is a reasonable assumption to apply to the nondeterminable affine parameter problem.

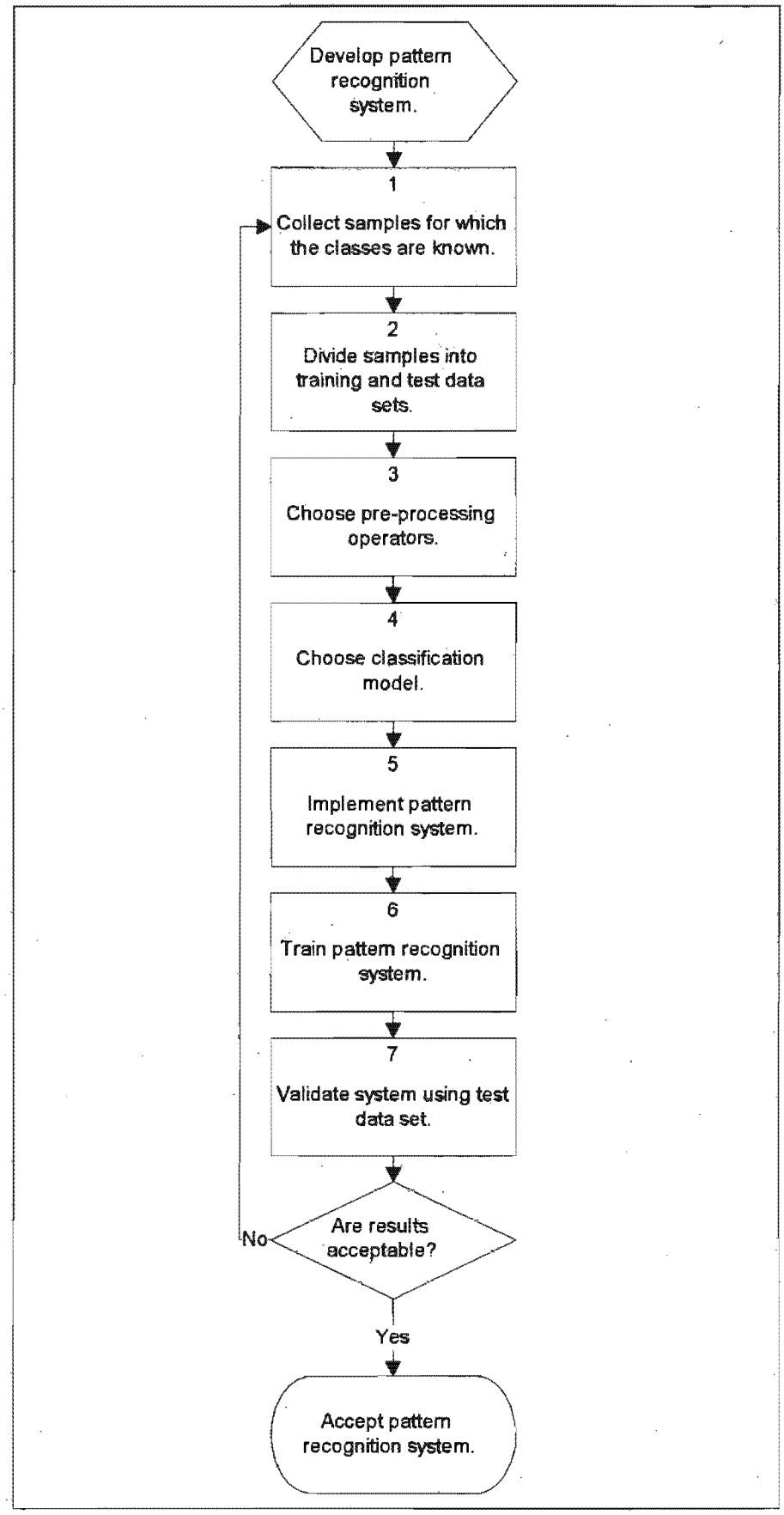


Figure 2-1: Process for developing nonparametric pattern recognition system.

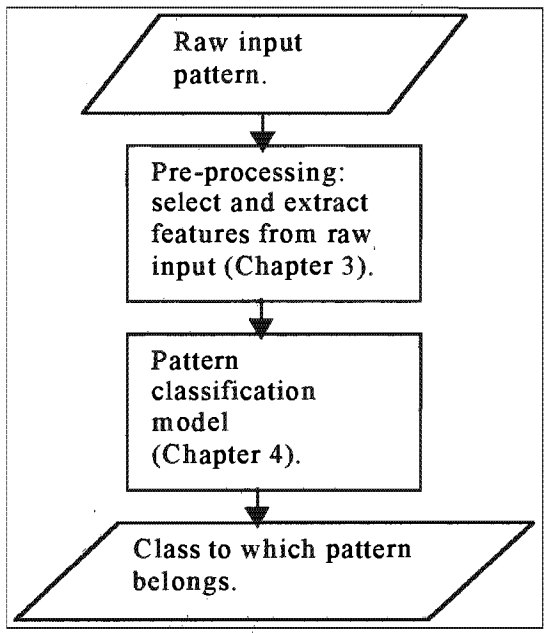


Figure 2-2: Typical architecture of a pattern recognition system.

2.3. Process for Developing a Pattern Recognition System

Figure 2-1 shows a high-level flowchart depicting a commonly employed process for developing a nonparametric pattern recognition system. Firstly, measurements are made of a large number of raw sample patterns for which the classes are already known (Process 1). It is critical that these patterns are representative of the pattern space. Then this set of patterns is divided into a training data set and a validation (or test) data set (Process 2). The features of the patterns that will be useful for ascertaining the class of the patterns must be determined and appropriate pre-processing operators must be chosen to extract and select these features (Process 3). Next, a nonparametric pattern classification model must be chosen (Process 4). The pre-processing operators and the pattern classification model are then implemented and integrated (Process 5). Then the parameters of the pattern recognition system are established using the training data (Process 6). Finally the entire system (see Figure 2-2 for a depiction of a pattern recognition system) is tested on the validation set (Process 7). It is critical that this data set is not used in the training process. If the results are acceptable the pattern recognition system can be considered successfully developed, else the development process must be restarted.

This is not a universally applied approach. For instance some pattern classification techniques (such as those derived from Kalman filters) work by trying to determine the *a priori* information and then applying Bayes' Rule. Also, if it is impossible to collect a large set of samples, alternative methods to splitting the data into a training and validation set can be used (such as the Leave-k-out method).

2.4. Historical Overview of Artificial Neural Networks

Ramon y Cajal (1911) pioneered research on the physical workings of the brain and developed the idea of neurons being its fundamental constituents. McColloch and Pitts (1943) and Hebb (1949) produced the modern seminal work, formally describing models for neural networks and rules of how learning takes place, respectively. Then came the development of *artificial* neural networks: Rosenblatt (1958) applied the growing theory of neural networks to pattern recognition and described the Perceptron, a machine that learns classes of patterns. It consists of a set of input measuring devices, each sending a weighted value of its input to a summation device. The summation device is also fed a threshold value. The output of the summation device is the sum of all the values fed to the summation device less the threshold. This value is applied to a hard limiter, resulting in a final output of -1 or 1. The machine makes a decision between two classes, one class represented by an output of -1, the other by an output of 1[9]. Rosenblatt proved that the learning algorithm of the Perceptron always converges. Widrow and Hoff (1960) developed the Adaline (adaptive linear element) which could be trained to differentiate between different pattern classes using the Least Mean Squares algorithm. There is little difference between the Adaline and the Perceptron except in their learning algorithm.

Minsky and Papert (1969) showed that the Perceptron is severely limited in what it can classify. They formally demonstrated that Rosenblatt's Perceptron can only differentiate between linearly separable classes. In particular, the Perceptron cannot solve a simple pattern classification problem, such as determining whether a binary string has an odd or even number of zeroes (known as the parity problem). Minsky

---

[9] The Perceptron is very similar to the maximum-likelihood Gaussian classifier, a classical linear statistical pattern recognition technique.

and Papert also suggested that multilayered perceptrons (i.e. with the weighted inputs being applied to hidden functions which then apply their outputs to a more sophisticated output device than the summation device) would not be of much practical use: no training algorithms had been developed for them and it was unclear whether or not they suffered from the same limitations as Rosenblatt's perceptron. This turned out to be untrue, but research into ANNs slowed down after their paper was published.

Kohonen (1982a and 1982b), von der Malsburg (1973) and Wilshaw (1976, with von der Malsburg) conducted work on self-organising maps leading to completely different types of ANNs to the Perceptron. This research mainly culminated in unsupervised learning algorithms for ANNs (not covered in this research, since they are usually used in applications where little is known about the classification categories), but it also resulted in the LVQ learning algorithm developed by Kohonen (1986), which is researched here. Also, Hopfield (1982) described a new type of ANN which learns to separate classes based on ideas borrowed from physics.

A major breakthrough in the development of MLPs was the Error Back-propagation algorithm proposed independently by Parker (1985), LeCun (1985), and Rumelhart, Hinton and Williams (1986). It had also been discovered by Werbos (1974), but remained largely unknown until Rumelhart *et al.* published their paper in *Nature*. This algorithm seems to be the first published learning mechanism developed for multilayered perceptrons. The objections to multilayered perceptrons suggested by Minsky and Papert are overcome by the Error Back-propagation algorithm (and many other learning algorithms that have been suggested since then). Rumelhart and McClelland (1986) published extensive research on the Error Back-propagation algorithm and suggested enhancements which improved its efficiency substantially.

An important application of ANNs is the nettalk ANN developed by Sejnowski and Rosenberg (1987), which can pronounce written English text. It took almost two weeks of CPU time on a VAX to train, and achieved a performance level of over 90%. This application did much to popularise ANNs and generated interest in researching them.

2.5. Definition and Description of Artificial Neural Networks

The following definition is based on Muller and Reinhardt (1991, p. 12). It is not the only definition of ANNs found in the literature. Also there are some ANN models which do not fit precisely into the following definition and description.

An Artificial Neural Network Model is a directed graph with the following properties:

- A state variable n_i is associated with each node i.
- A real-valued weight w_{ik} is associated with each link (ik) between two nodes i and k. The direction of the link is from k to i.
- A real-valued bias θ_i is associated with node i.
- A transfer (or firing) function, $f_i[n_k, w_{ik}, \theta_i, (k \neq i)]$, generally non-linear, is defined for each node i. The function calculates the state variable n_i.

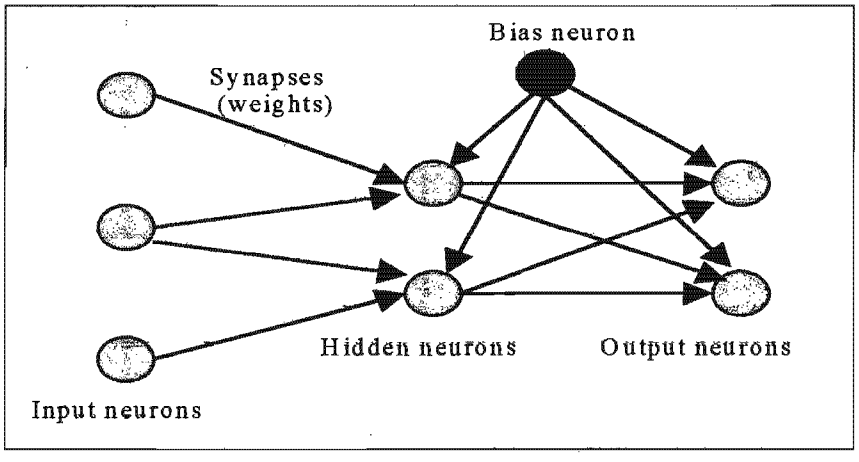


Figure 2-3: Artificial neural network depicted as a directed graph. The graph nodes represent neurons and the graph connections represent synapses.

The following terms are used in ANN literature and in this dissertation: The graph's nodes are called *neurons* and the links are called *synapses*. Nodes without links towards them are called *input* neurons and nodes without links leading away from them are called *output* neurons. Nodes which have a synapse leading to and from them are called *hidden* neurons. The transfer function is usually bounded between zero and one for classification problems. When a neuron's transfer function has been calculated it is said to have *fired*. The values calculated by the output neurons are

collectively referred to as the classifier's *response* (this term is used for any classifier, not just ANNs). It is usual in classification problems for each output neuron to correspond with one classification class. An ANN is called *feed-forward* if the graph topology has no closed paths. In this thesis only multilayered feed-forward architectures are considered, except for LVQ, which is a learning algorithm for an ANN whose structure is referred to as a lattice (Haykin, 1994). In particular, the ANN models considered (except LVQ) are referred to as *multilayered perceptrons* (MLPs). These have at least three layers, one input layer, at least one hidden layer and one output layer. Each layer usually only has connections to its adjacent layers. In general, the input to a neuron in a hidden or output layer is calculated by taking the sum of the product of the neurons and synapses feeding it and subtracting the bias value. However, as a mathematically equivalent alternative to associating each hidden and output neuron with a bias value, it is common to implement an extra node called a *bias neuron* which always fires a value of 1. The bias neuron is commonly connected to all the hidden and output neurons via synapses. The weights of these synapses correspond to the bias value θ_i. Formally, for a (non-input) neuron attached to a set of s weights $(w_1, w_2 \ldots w_s)$ connected to the bias unit and neurons in the previous layer with outputs $(o_1, o_2 \ldots o_s)$, its input, n, is defined as

$$n = \sum_{i}^{s} w_i o_i \,. \qquad \textbf{Eq. 2-6}$$

Figure 2-3 shows an ANN depicted as a directed graph, while Figure 2-4 shows an example of an MLP that executes the exclusive-or function. Rosenblatt's Perceptron cannot execute this function.

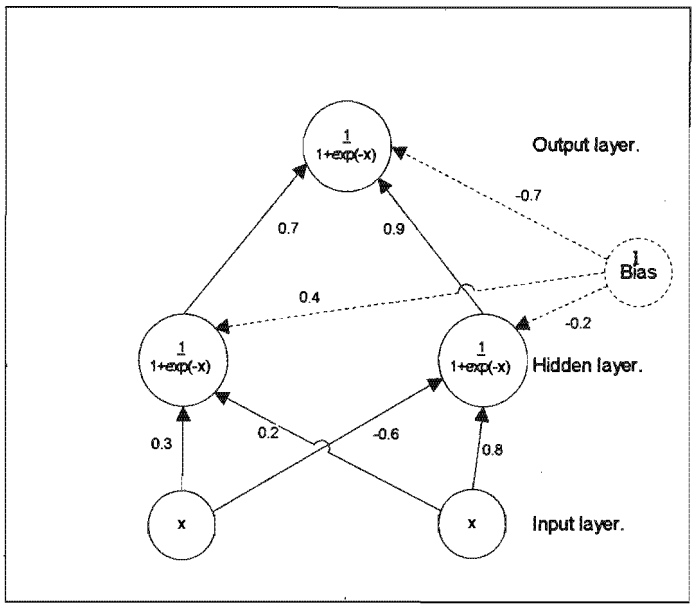


Figure 2-4: Example of a multilayered perceptron.
The circles represent neurons. Inside each circle is the firing function that the neuron applies to its input, x. The bias neuron outputs 1. The directed lines represent the network's weights and the values next to the lines are the values of the weights. This MLP receives binary values at its input neurons and executes the exclusive-or function.

An ANN runs in two modes, *training* and *classification*. In training mode the ANN parameters are modified so that it correctly classifies the training patterns (i.e. by producing the correct values at the output units). This is an iterative process. Initially, the synapse weights and biases of the ANN are set to random values. On each iteration these parameters are modified slightly so that the ANN fits the training data better. This is known as *supervised learning*. In *classification* mode the ANN's parameters are not modifiable. It receives a pattern as input and produces a response, thereby classifying the pattern. As such an ANN is said to *learn* a function. This is akin to the term *estimation* used in statistical literature (Sarle, 1994). In addition, all ANN models have associated algorithms which modify the parameters (synapse weights and biases) of the model during training mode. These are called *learning* (or *training*) *algorithms*. For instance, the Error Back-propagation algorithm is one particular learning algorithm for MLPs. The patterns used to train the ANN come from the *training set* (in statistical literature this is often called the *design set*) and the patterns used to test (or *validate*) the interpolation ability of the ANN constitute the *test (*or *validation*) *set*.

The following explanation of the rationale an MLP is based on Sarle (1997): Given a hidden or output neuron with N inputs, which define an N-dimensional space, the neuron constructs a hyperplane through that space with a different decision

represented by each side of the hyperplane. Thus by increasing the number of hidden (and output) neurons in a network, one creates a larger hyperspace for different decisions. The larger the hyperspace, the greater will be the discriminative ability of the ANN (but not necessarily its accuracy). The purpose of the bias units is best described by Sarle (1997), 'The weights determine where this hyperplane lies in the input space. Without a bias input, this separating hyperplane is constrained to pass through the origin of the space defined by the inputs. If you have many units in a layer, they share the same input space and without bias would all be constrained to pass through the origin.' This constraint would render the ANN incapable of approximating some functions.

2.6. Issues in Pattern Classification

There are numerous issues which need to be considered in the design of pattern recognition systems. Some of these are discussed here, particularly those relevant to the nondeterminable affine parameter problem which are typical of those encountered in image recognition. The issues are discussed here in general terms. Chapter 5 re-examines some of these in the context of the nondeterminable affine parameter problem.

2.6.1. Learning Invariance

Most image recognition problems require that the pattern recognition system be robust in respect of the following:

- *They must be tolerant of noise in the images.*

  In practical problems, small fluctuations in the values of some of the features always occur. This is caused by many factors (e.g. images are often slightly blurred or speckled).

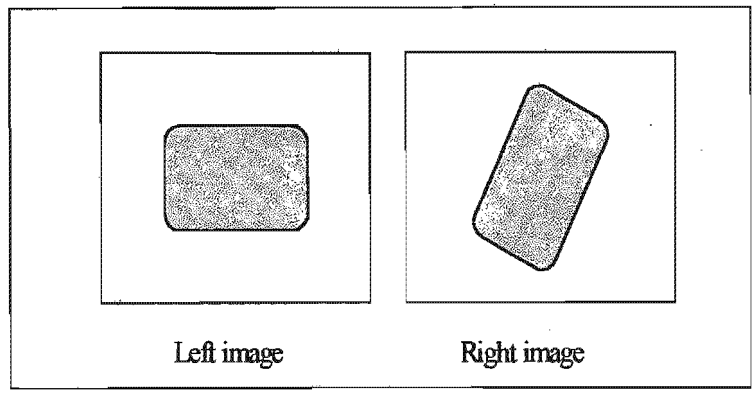


Figure 2-5: Identically shaped images with different rotations.

- *Images that differ in only their rotation must be classified in the same class (referred to as rotational invariance, see Figure 2-5).*

  For example, a pattern recognition system that recognises a person's face should still recognise the face if it is rotated, no matter what the angle of rotation[10].

  The problem is usually overcome in one of two ways. Either features that are invariant to rotation must be extracted, or the training data must be supplemented with samples rotated at various angles (Haykin, 1994). Supplementing the training data is not guaranteed to teach the classifier rotational invariance, since it is often impractical to supplement an already large data set with all the patterns comprising it rotated by arbitrarily chosen angles. It is, however, a simple method of introducing rotational invariance and is the method that is used in this research.

  A number of ANNs have been developed which have rotational invariance built into them (e.g. Lin and Wang, 1996). However, they have been developed very recently and have not received much coverage in ANN literature. In addition implementations of these ANNs could not

[10] It often happens that rotational invariance must definitely not be learnt. For instance an upside down "M" should not be classified as an "M" by a letter recognition system. Nevertheless, most classifiers, even letter recognition systems, should be invariant to small rotations.

be obtained. They were therefore considered too experimental, at this time, to use in this research.

- *Images that differ only in their scale must be classified in the same class (referred to as scale invariance, see Figure 2-6).*

Teaching a classifier to learn scale invariance presents similar problems to teaching a classifier to learn rotational invariance. It is also solved in the same way (i.e. extracting scale invariant features or supplementing the training data). As with rotational invariance, some ANN models are designed to be resistant to scale changes (again, see Lin and Wang, 1996).

In contrast to rotational invariance, it is difficult to define an automatic procedure for supplementing the training data with scaled patterns. To supplement a data set with rotated patterns, one simply rotates the images in the training data by various angles (e.g. 45, 90, 180 and 270 degrees). However, it is possible to scale an image by too large a factor, such that the image texture crosses the image frame. It is also possible to reduce the image by too much, thereby reducing a meaningful image to white space. As such, scaling often has to be done manually.

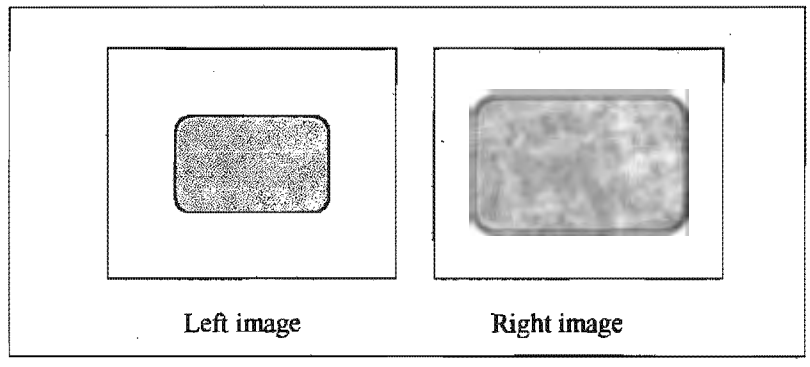


Figure 2-6: Identically shaped images with different scales.

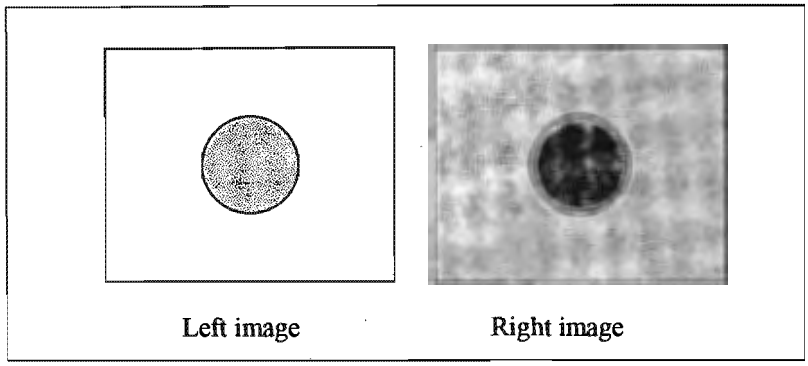


Figure 2-7: Identically shaped images with different greyscale amplitudes (a). All the greyscale values in the right image are equal to the left image greyscale values plus a constant.

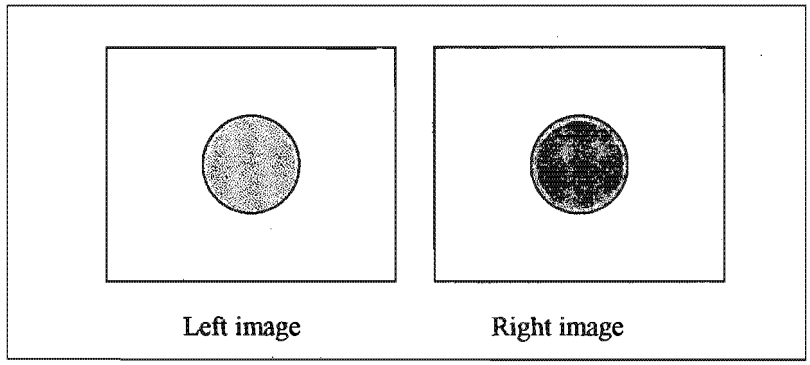


Figure 2-8: Identically shaped images with different greyscale amplitudes (b). Unlike Figure 2-7, the difference in the amplitudes is not just scale. The background of the right image is identical to the background of the left image, but the foreground is darker.

- *Images whose greyscale values differ, but have the same shape, must be classified in the same class (referred to as amplitude invariance, see Figure 2-7 and Figure 2-8).*

  Amplitude invariance is handled in two ways in this research. Firstly the training data is supplemented with identical images with differing greyscale values. Secondly a number of pre-processing operators assist, to some extent, in introducing amplitude invariance to the images. Some extract the gradient in the image (also called edge detection) and another normalises the greyscale values, by determining the lowest greyscale value in the image and then subtracting this value from all the greyscale values in the image. These pre-processing operators are useful for removing variance in the *scale* of the greyscale values (Figure 2-7), but

they do not cater for the more complex situation where only part of the image has a vastly different amplitude (Figure 2-8). This more complex problem can be overcome to some extent by applying a threshold to the greyscale values (a pre-processing operator, also used in this research). Each greyscale value is divided by the threshold value, thereby reducing the range of the greyscale spectrum (the remainder in the division operation is discarded). The ideal threshold value will eliminate the problem depicted in Figure 2-8 without removing too much texture. Of course, the ideal threshold is usually a moving target, differing across images.

- *Images, which differ only in their position in the image frame, must be classified in the same class (referred to as shift invariance, see Figure 2-9).*

  As with rotational, scale and amplitude invariance, this can be handled by supplementing the training data set with images which differ only in their shift, or by extracting features invariant to shift. The former approach has been used in this research.

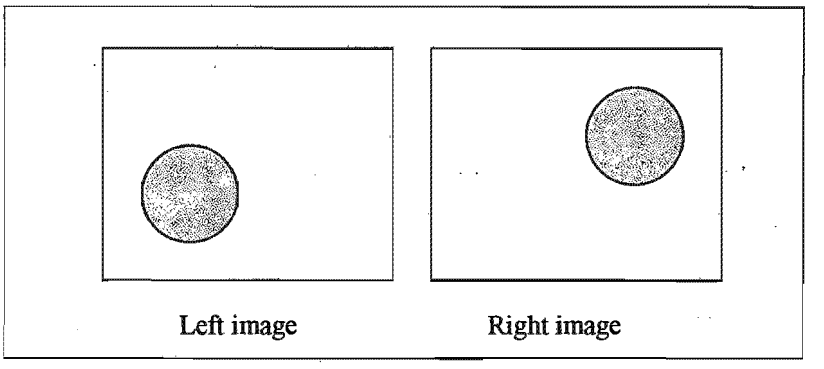


Figure 2-9: Identically shaped images with different shifts.

Established techniques for extracting features invariant to rotation, scale and shift from two dimensional images exist[11], but only if the object in the image frame can be

[11] For example, calculating a set of moments based on the centre of gravity of the object in the image (see Murtagh, 1996, for a description).

isolated. It seems that this would be extremely difficult to do in the case of the nondeterminable affine parameter problem. However, it is worth researching this issue further than has been done here. Interestingly, although a successful pattern recognition system for the nondeterminable affine parameter problem requires invariance to noise, rotation, scale and amplitude and shift, there are additional complicating factors involved. These complicating factors are discussed in Chapter 5. Pre-processing operators implemented to deal with the above issues are discussed in Chapter 3.

2.6.2. Assigning a Degree of Classification

Often patterns do not fit precisely into a particular category. Factors such as noise can render them ambiguous. It is often necessary to express the degree, confidence or probability with which a pattern belongs to a class. At the outset of this research it was indicated that it would be useful if, for each particular image examined by a pattern recognition system, a confidence measure could be assigned to its classification. That is, a classification model which specifies the degree to which a pattern belongs to a class would have an advantage over one that does not. All the models used in this work, except for ANNs trained with LVQ, have this characteristic.

However it is important to be cautious of the confidence measures expressed by the pattern recognition systems developed in this research. The graded responses of the classifiers do not correspond to probability measures. For example, if an ANN's class A neuron outputs 0.9, this does not imply that the pattern belongs to class A with a probability of 90%.

Measuring statistical confidence levels for the systems developed in this thesis would be extremely complex because the class distributions are unknown and there are multiple classes. Confidence levels have therefore not been calculated.

2.6.3. Dynamic Learning of New Data

Often pattern recognition systems exist in a dynamic environment where it would be useful to fine-tune their parameters (i.e. continue training the system) based on newly acquired patterns that were not part of the original training data set. For example it would be advantageous if a system that recognises patches with nondeterminable

parameters could improve its accuracy continuously by incorporating information regarding newly acquired patches in its decisions.

The problem of adapting ANNs to new data has received some attention in ANN literature. Adaptability to new patterns occurring in a real-time environment has been explored extensively with Adaptive Resonance Theory (ART) ANN models (Carpenter *et al.*, 1991). However, ART models, which are quite controversial with some statisticians (e.g. Sarle, 1997), are not considered in this work. Orr (1996) also describes how Radial Basis Function ANNs can be used to dynamically learn new patterns.

2.6.4. Artificial Neural Network Issues[12]

Certain issues have to be overcome (or accepted) in the development of ANNs. For the problems of this research, the most important of these are:

- *The ANN must not overfit the training data (see Figure 2-10).*

  If an ANN learns the irrelevant idiosyncrasies of the patterns in the training data to such an extent that it performs poorly when classifying images in the validation set, then it is said to be overfitting. This problem occurs with other pattern recognition techniques, but characteristics of the architecture of ANNs (and in particular MLPs) make them particularly susceptible to this problem. Overfitting can result from having too many unrepresentative patterns in the training set, having too many free parameters in the ANN (too many hidden neurons in the case of MLPs) or if the weights of the network become too large.

  The reason why overfitting occurs when the weights of an ANN become too large is that small changes in the input values of the patterns result in large changes in the calculations made by the ANN, ultimately resulting in large changes in the response of the ANN. Even though the ANN might learn the correct response to the patterns in the training data, slight

[12] The notes in this section are based, primarily, on Masters (1993).

deviations from these patterns result in incorrect decisions. Some learning algorithms combat this problem by introducing a concept called weight decay.[13] An example of this is given in the explanation of the Resilient-propagation ANN in Chapter 5.

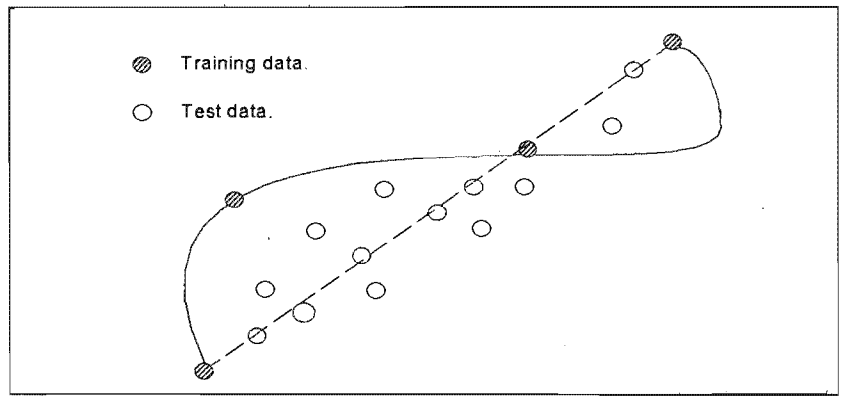


Figure 2-10: Overfitting depicted graphically.
The curved solid line represents the classification function learnt by a system that is overfitting the training data. The straight dashed line represents a much better fit.

- *The number of hidden neurons and hidden layers must be determined.*

  Determining the number of hidden layers and the ideal number of neurons to put into the hidden layers of an MLP are two of the most important tasks in implementing these models. Some theoretical and empirical results need to be taken into account when making decisions in this regard.

  Firstly, any continuous function whose domain variables have definite bounds can be learnt by an MLP with one hidden layer (Cybenko, 1988; Funahashi, 1989; Hornik, et al, 1989; Hornik, 1991). In addition, an MLP with two hidden layers is a universal approximator (Hornick *et al.*, 1989; Kurkova, 1992) — i.e. it can approximate any partial-recursive function. The variables of greyscale images are bounded and continuous, therefore one hidden layer should be sufficient to solve the nondeterminable affine parameter problem in theory, if it is a function. However, there is

[13] This is ridge regression in statistical terminology (Sarle, 1997).

empirical evidence that many practical problems are better solved with multiple hidden layers (e.g. the handwriting recognition devices of LeCun *et al.*, 1989a, 1989b, 1990a have multiple hidden layers).

If an ANN has too few hidden neurons it will fail to learn the function sufficiently well. If it has too many it will overfit the training data, as mentioned above. The optimal number of hidden neurons per layer has to be determined empirically by comparing interpolation results between different implementations[14]. According to Masters (1993) the geometric pyramid rule provides a rough but reasonable first approximation of the ideal number of neurons per layer. This rule states that the number of neurons in each hidden layer follows a geometric progression. Let i be the number of input units, o be the number of output units and l be the number of layers. Then the number of hidden neurons in hidden layer H_k (where k denotes the layer number) is

$$H_k = or^{l-k}, \quad k=2,3,\ldots,l-1 \qquad \textbf{Eq. 2-7}$$

where

$$r = \sqrt[l-1]{\frac{i}{o}}. \qquad \textbf{Eq. 2-8}$$

An alternative is to use either a hidden neuron pruning or growth learning algorithm. Examples of the former are the Optimal Brain Damage and Optimal Brain Surgeon techniques of LeCun *et al.* (1990b) and Hassibi *et al.* (1993), respectively. A popular example of the latter is the Cascade-

---

[14] Note that in this instance, the validation set should only be used to benchmark different implementations. As such, the validation set is not being used for training (which would be unacceptable). It would also be unacceptable to train the ANNs (with differing numbers of neurons) multiple times and, after each training session, compare their performances against the validation set until an ANN is derived that has achieved an acceptable level of accuracy.

correlation algorithm (Fahlman and Lebiere, 1990). Pruning and growth techniques have not been explored in this research[15].

- *When to stop training the ANN must be determined.*

  Determining the number of training iterations for an ANN is difficult. In practical problems the training set seldom represents the pattern space perfectly. As a result of this, if training is continued for too long (i.e. too many training iterations are carried out), the ANN overfits the training patterns and its interpolation performance on the validation set will actually degrade. An often tried, but incorrect, solution to this problem is to train the ANN while regularly checking its performance on the validation set. When the performance on the validation set begins to degrade training is stopped. As Masters (1993) points out, this process is unsound because the validation set is being used to influence training. Better solutions are to either set a limit on the number of training iterations or to train the ANN until the improvement on the training set data is negligible.

- *Local minima must be avoided.*

  ANNs often get trapped in local minima. A particular random initialisation of weights might result in a substantially less than optimal final state. As such, Masters (1993) recommends initialising many training runs for each ANN, selecting the one that performs best on the training set to use in the validation procedure.

- *The optimal values of the ANN learning parameters must be determined.*

  Most ANN models have a number of parameters that have to be set before the ANN can be trained. While some of the statistical classification

---

[15] A third-party implementation of Cascade-correlation was tried during the informal experimentation period of this research, without any success. This could be attributed to the poor documentation of the third-party package or the failure on the author's part to determine the ideal parameters for the system.

models in this research also have parameters which have to be pre-set, the parameters in the ANN models are generally much harder to determine. The Error Back-propagation algorithm used in this research has two user-defined parameters, learning and momentum, which have to be determined. Resilient Propagation has three. Generally, ANN parameters have to be determined empirically. This, in addition to having to determine the number of hidden neurons and layers, significantly influences the time it takes to develop a successful ANN.

- *The output neuron threshold values have to be determined.*

Determining the classification threshold of an output neuron of an MLP is also complex. The values of the output neurons of the MLP classifiers in this work are calculated using a sigmoid function bounded between zero and one. The value between zero and one separating the classification decision of an output neuron is called the threshold. The optimal threshold values for the output neurons are those that minimise the total cost, C, of the classifier, defined as follows (Masters, 1993)

$$C = \sum_{i}^{m} q_i p_i c_i \,. \qquad \textbf{Eq. 2-9}$$

where q_i is the prior probability of the classifier encountering a member of class i, p_i is the probability of failing to detect a member of class i, c_i is the cost of failing to detect a member of class i and m is the number of classes. The cost of a misclassification in many image recognition problems, including the nondeterminable affine parameter problem, is difficult to quantify. For simplicity all misclassifications are assumed to have an equal cost. Therefore, the term c_i is dropped from equation Eq. 2-9, giving

$$C = \sum_{i}^{M} q_i p_i \,. \qquad \textbf{Eq. 2-10}$$

This discussion implies that it is important to minimise equation Eq. 2-10 in the implementation of the MLPs in this work. However, Chapter 5 makes it clear that values of q_i and p_i differ across applications within the nondeterminable affine parameter problem. Therefore, for the MLPs in

this research, the simple *winner takes all* method is used, whereby a pattern is classified in the class corresponding to the output neuron with the largest output.

- *The intractability of an ANN must be resolved (or accepted).*

  Intractability is a problem frequently discussed in ANN literature. It is difficult to determine how an ANN with only a few hidden neurons reaches a decision, let alone more practical ANNs which often have hundreds of neurons. There are visualisation techniques, such as Hinton diagrams (Rumelhart and McClelland, 1986) and Bond diagrams (Wejchert and Tesauro, 1991) which attempt to render the derivation of an ANNs output more understandable, but their practical value is limited. Intractability is of particular concern in critical applications where failure is expensive and must be predictable. ANN models, for the most part, simply do not have the well formulated theoretical foundations of statistical methods and are often unsuitable for these critical applications. Nevertheless, ANNs are particularly useful for real-time pattern recognition problems, which cannot be solved off-line using traditional statistical methods (Sarle, 1994).

There are other issues which effect ANNs, such as the credit assignment problem (outlined by Minsky, 1961), a problem found throughout the field of artificial intelligence (Haykin, 1994). With respect to ANN theory the *structural* credit assignment problem is concerned with determining which synapses and neurons are primarily responsible for a particular decision (Minsky also describes *temporal* credit assignment, but this is less relevant to ANNs). However credit assignment is, primarily, of concern to researchers developing new learning algorithms, which is not an objective of this research.

2.7. Evaluation Criteria

One of the objectives of this dissertation is to compare the performance of the recognition systems. The following sections discuss these in order of importance with respect to this thesis.

2.7.1. Measuring Accuracy (Interpolation)

The most important measurement of a pattern recognition system's performance is its ability to interpolate to patterns that have not previously been seen. This is the primary purpose of building these systems and the reason why the validation set must be created. Interpolation can be measured in a number of ways. One possibility is to use Eq. 2-10, the cost equation. Alternatively, assuming the densities of the classes in the validation set closely approximate the densities over the entire pattern space, a simple measurement of accuracy, A, is then

$$A = \frac{G}{P}, \qquad \text{Eq. 2-11}$$

where G is the number of correct classifications and P is the number of patterns in the validation set. In fact, in pattern recognition problems, Eq. 2-11 is usually the most sensible. As such it is the measurement of accuracy used in this research.

For MLPs, which assign an approximation between zero and one to a classification, it is also useful to examine the mean squared error, *MSE*, over the validation set

$$MSE = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{m}(d_{ij} - a_{ij})^2 \qquad \text{Eq. 2-12}$$

where n is the number of patterns, m is the number of output classes, d_{ij} is the desired response of the classifier for class j of pattern i and a_{ij} is the classifier's actual response. This measurement is not as informative as the two previous equations, but it does give an indication of how well the MLP has converged to the classification function it is trying to learn.

It is important to differentiate between the accuracy with which a pattern recognition system classifies patterns in the training set and the accuracy with which it classifies patterns in the validation set (or other unseen data). Obviously it is the accuracy of the latter which determines the success of the system.

2.7.2. Measuring Efficiency (Execution Speed of Classification)

The classification speed is critical in applications where numerous patterns have to be classified, as is the case with the nondeterminable affine parameter problem, or where classification must take place in real-time, such as face recognition for an access system. For example, (using the nondeterminable affine parameter problem) a typical

image matching problem contains hundreds of thousands of template patches. Patches must be classified quickly, so as not to render the matching process impracticably slow. More importantly, a useful system must make a decision much faster than it takes for the ALSM program to fail to converge. (The ALSM program can be viewed as a pattern recognition system as well. If it successfully matches a patch the affine parameters are determinable, else not.)

Classification speed includes the time taken to perform pre-processing and the time taken for the classification model to make a decision. The simplest and most obvious measure of classification speed, execution time in seconds, is problematic. This is affected by factors unrelated to the system's efficiency, such as computer speed, computer load, development environment, implementation programming language and the quality of the implementation. Although execution times are used in this dissertation they should be treated with some scepticism.

Because execution speed is unreliable, analytical methods are perhaps the best way to calculate efficiency. However this is often extremely difficult, if not practically impossible. Complex factors such as parallelization and pipelining[16] are difficult to take into account.

2.7.3. Measuring Training Speed

Training speed is only relevant to some classification models but it is particularly relevant to ANNs. For instance, the standard k-NN algorithm does not have a training phase[17], while the minimum distance classifiers implemented in this research need negligible time for training. Much attention is devoted to improving training speed in ANN literature (see Sarkar, 1995). It is important that training occurs in a practically reasonable amount of time (approximately 48 hours was specified as the upper limit for this research).

---

[16] Pipelining is a recent technique implemented in modern computer processing units. It speeds up execution by executing instructions in parallel.

[17] This might seem like an advantage, but it is offset by its slow classification speed. Also, the system containing the k-NN model must pre-process the training data. The time taken to do this is considered part of the training time.

The problems discussed with regard to measuring efficiency apply here as well. The same measuring techniques, with the same associated problems, can be used: time and analytical methods (time is used in this work). With regard to ANNs it is interesting to compare the number of training iterations but this is deceptive, since the time taken for an iteration is dependent on factors such as the learning algorithm and the number of neurons in the network.

It is important to realise, however, that training occurs once in most applications, while classification is an on-going process. Therefore the classification execution speed is far more important for judging the usefulness of a pattern recognition system (except, of course, where the training speed is exceptionally slow).

Some classification models can adapt to new data: they can be continuously trained as new training data is obtained. Therefore, it is of interest to measure the speed of the initial training procedure and the speed of subsequent training on newly obtained data for these classifiers. An excellent example of this is Orr (1996), which contains an example of analytical methods being used to calculate the initial training time and adaptive training time for Radial Basis Function Neural Networks.

2.7.4. Other Comparative Criteria (Less Relevant to this Research)

Other criteria which are of interest when comparing pattern recognition systems are:

- *Amount of storage space needed by the classifier.*

  The computer storage space occupied by a system can be a factor in selecting it for practical use. For instance, in this research the third-party ANN implementations often ran out of storage space during the training phase. This was, however, related to the limited hardware used and the shortcomings of the *implementations* of the classification algorithms, rather than the algorithms themselves.

- *Viability of implementing classification algorithm in parallel hardware.*

  Classification models that are designed so that they can be implemented in parallel hardware are of interest, since their speed would increase substantially. Haykin (1994) discusses issues regarding parallelism of

ANNs in detail. In order for an ANN to be efficiently implemented in parallel hardware its components must act locally. This means that the behaviour of its synapses and neurons should only be affected by neighbouring neurons and synapses. The k-NN model has also been implemented in parallel hardware (this is discussed further in Chapter 4). There seems little need to implement minimum distance classifiers in parallel, since their design is well-suited to the standard von Neumann architecture of modern computers and they are fast, even with serial execution.

- *Similarity of the system to biological pattern recognition methods.*

Since pattern recognition is generally a stochastic, imprecisely achieved task that is performed well by human beings (and animals), there has been substantial interest in comparing computer and animal methods of pattern recognition, particularly in the field of vision. Hebb's work (1949), which describes how the brain's neurons and synapses can learn patterns, is the basis for many learning rules in ANN research. Peretto (1992) discusses biological neural network modelling in detail. Haykin (1994) and Aleksander and Morton (1990) also discuss biological models, with particular emphasis on Hebb's work and that of Hubel and Wiesel (1962, 1977). An interesting biologically based neural network is proposed by Yadid-Pecht and Gur (1996). In this thesis ANNs trained with LVQ are of some biological interest, but this aspect of them is not explored here.

It is also interesting to note that many well-known researchers in the neural network field are examining how neural networks can help towards building a model of human consciousness, e.g. Taylor and Freeman (1997) and Aleksander (1992). However it should be noted that this research is at an exploratory stage. Biological pattern recognition issues are not yet well understood and this dissertation views classification from an engineering perspective. Therefore, no metrics have been developed to assess the biological validity of the implemented classifiers.

- *Fault tolerance of the classifier.*

Some research has gone into developing specialised ANN hardware that is fault tolerant (e.g. Aleksander *et al.*, 1984; Haykin, 1994 – Chapter 15 – describes hardware issues). Fault tolerance implies that the ANN's performance degrades gradually under adverse operating conditions. That is, some weights or neurons can be eliminated without drastically effecting accuracy. Since no specialised ANN hardware has been used in this work no metrics have been examined regarding fault tolerance.

3. Pre-processing Operators

3.1. Introduction

This chapter describes the pre-processing operators that were implemented (or obtained) by the author. For a digital pattern recognition system to work, the patterns must be represented as vectors of — usually floating point — numbers. The pre-processing operators are applied to these raw data vectors, thereby accentuating features which can be used by the classification models to partition the pattern space. Many pre-processing operators are referred to as feature extractors or feature selectors. A feature selector reduces the dimension of the problem by selecting a subset of the variables of a pattern vector for processing by the classification model and a feature extractor maps useful information content in the patterns to a lower dimension (Kittler, 1986).

There are a number of uses of feature extraction and selection. The engineering complexity, computing space and computing time of a classification model grows with the number of dimensions in the pattern space. By reducing the dimensionality of the pattern space the complexity of these is reduced and classification decisions are based only on essential discriminatory information (Kittler, 1986). In addition, feature selectors and extractors are used to build invariance into pattern recognition systems.

In any particular recognition system implemented in this work only a few of the pre-processing operators presented in this chapter are applied. The order in which they are applied is usually also significant (i.e. most of these operations are not commutative). In the experiments conducted (described in Chapter 6) many variations, on the pre-processing operators used and the order in which they are applied, have been tried. Thus the optimal selection and processing order of the pre-processing operators for solving the nondeterminable affine parameter problem have been determined empirically.

The pre-processing operators described in the remaining sections of this chapter have been selected by the author because of their potential for contributing to a solution of the nondeterminable affine parameter problem. Thus there is an emphasis in this

selection on pre-processing operators suited for greyscale image recognition. Each section formally describes how an operator works and its purpose. All the pre-processing operators were implemented in the C programming language by the author[18].

The pre-processing operators described include:

- Subtracting the minimum value of a pattern vector from all the features in the vector.
- Sobel edge detection.
- Maximum gradient edge detection.
- Dividing the elements of a pattern vector by a constant (thresholding).
- Applying a ceiling (maximum value) to the elements of a pattern vector.
- Re-scaling the elements of a pattern vector using linear mapping.
- Applying z-axis normalisation to the elements of a pattern vector.
- Standardising the features of a pattern vector.

The author decided not to implement a noise filtering pre-processing operation, other than thresholding. The reasons for this are related to the properties of the nondeterminable affine parameter problem, and are, therefore, explained in Section 5.2 (p. 73).

3.2. Reducing the Pattern Elements by the Minimum Value

For each pattern vector the minimum feature, x, is determined. Then x is subtracted from every feature in the pattern vector. This results in a new vector which has at least one feature having a value of zero.

---

[18] The code for z-axis normalisation is based on code in Masters (1993).

Formally, given a pattern vector $\mathbf{x}$, such that $\mathbf{x} = (x_1, x_2, x_3, ..., x_n)^T$ with all its variables ≥ 0, there exists an $x_i \in \mathbf{x}$, such that $x_i \leq x_j$ for all j such that $1 \leq j \leq n$. This operator subtracts x_i from $\mathbf{x}$ giving a new vector $\mathbf{x}'$. That is,

$$\mathbf{x}' = \mathbf{x} - x_i . \qquad \textbf{Eq. 3-1}$$

The purpose of this operator is to assist the classification model with learning amplitude invariance, as explained in Section 2.6.1. By re-examining Figure 2-7, one can see that by applying this operator, the Euclidean distance (and most other measures of distance) between patterns that have similar shapes, but occupying different parts of the greyscale range, would be reduced.

3.3. Gradient Measurement (Edge Detection) Operators[19]

Measuring the gradients (or determining the edges[20]) of an image extracts information regarding its shape and, therefore, these operators usually play a critical role in image recognition problems. Two gradient measurement operators were implemented, the Sobel and Maximum Gradient edge detection algorithms. These operators were chosen, as opposed to more complex edge detection operators such the Canny operator, because of pre-processing speed considerations and their simplicity. Besides extracting important features (i.e. the edges of the image texture), these operators also reduce the dimensionality of the pattern vectors from $n \times n$ space to $(n - 2) \times (n - 2)$ space, where n is the dimensionality of the original pattern. In this respect these operators compare favourably with the Roberts edge detection algorithm, which reduces the dimension of the pattern vectors from $n \times n$ space to $(n - 1) \times (n - 1)$ space.

G_1	G_2	G_3
G_4	G_5	G_6
G_7	G_8	G_9

Figure 3-1: 3x3 subsection of a greyscale image.

---

[19] The edge detection operator descriptions are based on Boyle and Thomas (1988), Ekstrom (1984), Gonzalez (1987) and Smit (1997).

[20] Determining the edges of an image is the same as determining its gradients (or gradient image), because the edges of an image correspond to changes in gradient.

The Sobel operator works as follows: For each 3x3 subsection of a greyscale image (see Figure 3-1) the gradient g is calculated as follows:

$$\delta = (G_3 + 2G_6 + G_9 - G_1 - 2G_4 - G_7)^2 + (G_1 + 2G_2 + G_3 - G_7 - 2G_8 - G_9)^2 \quad \text{Eq. 3-2}$$

if $\delta < \alpha$, where α is a predefined constant, then $g = \delta$, else $g = 0$.

The Maximum Gradient operator works as follows: For each 3x3 subsection of a grey-scale image (see Figure 3-1)

$$\delta = \frac{\max(|G1 - G9|, |G2 - G8|, |G3 - G7|, |G4 - G6|)}{2}. \quad \text{Eq. 3-3}$$

if $\delta > \rho$, where ρ is a predefined constant, then $g = \delta$, else $g = 0$.

Both operators transforms the original greyscale ($n \times n$) image into an (n - 2) × (n - 2) gradient image.

3.4. Thresholding

Thresholding maps the features of a pattern within particular ranges to one value. For instance, binary thresholding transforms all greyscale values less than 128 to 0 and all other greyscale values to 1 (assuming a greyscale range of 0 to 255). In general

$$g' = \text{int}(g/\rho) \quad \text{Eq. 3-4}$$

for every feature g in the original pattern. ρ is a user-defined parameter specifying the number of partitions (ρ = 128 in the case of binary thresholding for greyscale images).

This operator is used to eliminate small, noisy fluctuations in the pattern values. As mentioned in Section 2.6.1 it also reduces variation in amplitude. If small values of ρ are used the difference in similar patterns in the same class is likely to be reduced. With large values of ρ (such as 128) too much information is lost and patterns from different classes become too similar for the classification algorithms to differentiate between them.

3.5. Setting a Maximum Value (Ceiling)

All features with a value higher than a user-defined parameter, ρ, are set to ρ. Formally, for every feature with a value g, the transformed value g′ is defined as

$$g' = \max(g, \rho). \qquad \textbf{Eq. 3-5}$$

This operator is only useful in this thesis when used in conjunction with one of the edge detection operators (i.e. it is applied to the gradient image). Intuitively it seems that, above a minimum value, the strength of an edge (i.e. the gradient across a subsection of grey values) is not likely to be a factor in determining to which class an image belongs. As such, this operator also assists in reducing amplitude variation. As with the operator discussed in Section 3.2, this operator should reduce the Euclidean distance (and most other measures of distance) between images in the same class. Determining the optimal value of ρ is a major consideration when applying this operator. In most problems it has to be determined empirically.

3.6. Re-scaling Using Linear Mapping

This operator maps the features of a pattern from one range (0 to 255 for greyscale images) to another (typically -1 to 1). Formally, for every feature, its value, g, is transformed from the range $[U .. L]$ to g', a value in the range $[u .. l]$, the lower and upper bounds respectively of the new range, using the following formulae (Sarle, 1997)

$$r = u - l, \text{ and} \qquad \textbf{Eq. 3-6}$$

$$g' = \left(\frac{g}{(U-L)}\right) r + l + u. \qquad \textbf{Eq. 3-7}$$

As discussed in Section 2.6.4 overfitting in an MLP can result from large weights. Large input values have the same effect (i.e. they can result in large input values to the hidden neurons, resulting in the MLP being too susceptible to minor fluctuations in the input values). By reducing the range from [0 .. 255] to [-1 .. 1] this problem should be averted (Sarle, 1997).

In addition, it is necessary to pre-process the input to an ANN trained with LVQ so that the Euclidean Norm of the patterns is a constant. This operation is used in conjunction with another operation described below (z-axis Normalisation) in order to achieve this.

3.7. Z-axis Normalisation[21]

This operation modifies the patterns so that they all have a Euclidean norm of one. This is achieved by pre-processing the data with the linear re-scaling operation described above *(l = -1, u = 1)* and then applying the following equations to a pattern $\mathbf{x}$, such that $\mathbf{x}^T = (x_1, x_2, ..., x_i, ..., x_n)$, in order to get a new pattern $\mathbf{x}'^T=(x'_1, x'_2, ..., x'_i, ..., x'_n, x'_{n+1})$

$$l = \sqrt{\sum_{i=1}^{n} x_i^2}\,, \qquad \textbf{Eq. 3-8}$$

$$\alpha = \frac{1}{\sqrt{n}}, \qquad \textbf{Eq. 3-9}$$

$$x'_i = \alpha x_i \text{ and} \qquad \textbf{Eq. 3-10}$$

$$x'_{n+1} = \alpha\sqrt{n - l^2}\,. \qquad \textbf{Eq. 3-11}$$

Z-axis normalisation has the advantage over simpler normalisation techniques (such as dividing each re-scaled vector element by *l*) of preserving absolute magnitude information. Note that the transformed pattern space has n+1 dimensions. x'_{n+1} is a scaling factor and is necessary in order to ensure that the Euclidean norm of the vector adds up to 1. Normalisation has to be applied to the input vectors of ANNs trained with LVQ. The justification for this can be found in Section 4.6.

3.8. Standardising the Data

This well-known operator transforms the data with the intention of making the means of the features 0 and their standard deviations 1. The mean and standard deviation must be calculated using the training data only[22]. Formally, for a feature x_i of pattern $\mathbf{x}$, the transformed feature x'_i is calculated as follows

---

[21] This description is based on Masters (1993).

[22] Clearly it is incorrect to calculate the mean and standard deviation using the test data because the construction of the pattern recognition system will then be influenced by the test data, rendering this data useless for validation purposes.

$$x'_i = \frac{x_i - \mu}{\sigma}, \qquad \text{Eq. 3-12}$$

where μ and σ, the mean and standard deviation respectively, are **estimated** from a sample of size n of the i_{th} feature of the pattern space, using the following formulae:

$$\mu = \frac{\sum_{i=1}^{n} x_i}{n} \text{ and} \qquad \text{Eq. 3-13}$$

$$\hat{\sigma} = \sqrt{\frac{1}{n-1}\sum_{k=1}^{n}(x_{ik} - \hat{\mu})^2}\,. \qquad \text{Eq. 3-14}$$

$\hat{\mu}$ and $\hat{\sigma}$ are estimates of μ and σ, respectively. Note that Eq. 3-13 and Eq. 3-14 are calculated using the training data, but Eq. 3-12 is applied to every pattern.

It is likely that the means and deviations of the pattern variables vary significantly from each other. It is possible that by standardising them so that they have the same mean and standard deviation, the classification models will not give undue precedence to some input variables over others, which is why this operator is used. However, this operator assumes that the input variables have an approximately Gaussian distribution. If poor results are achieved by pattern recognition systems that use this operator it could be a result of the input variables of the pattern space having complex, non-Gaussian distributions.

3.9. Feature Selector using a Genetic Algorithm

A feature selector reduces the dimension of the problem by selecting a subset of the variables (features or measurements) of a pattern vector for processing by a classification algorithm. Determining the optimal subset of features is a combinatorial problem (Kittler, 1986). For patterns with n features, there are 2^n possible feature sets. It is clearly impractical to examine every possible feature set for even moderate values of n. [23] Good feature selection algorithms have to operate within a reasonable

[23] The patterns in the data sets in this research have 81 features. If the edge detection pre-processing operators are applied (these offer the greatest reduction in dimension of the implemented operators) the dimension is reduced to 49 features. No computer in the world could examine all 2^{49} feature sets in a reasonable amount of time.

length of time (which is quite a subjective factor) and select a good set of features for partitioning the pattern space.

One feature selection algorithm, developed by Siedlecki and Sklansky (1988) and Brill et. al. (1992), was researched in some detail for this dissertation. The algorithm uses a genetic algorithm (Holland, 1975) to attempt to select a subset of features close to the optimal subset of features[24].

Genetic algorithms are typically used to find good solutions to optimisation problems with a large number of variables. The standard genetic algorithm was developed primarily by Holland and is discussed in detail in Holland (1975). Its development was inspired by genetic theory, natural selection and evolutionary theory. The description of the genetic algorithm used in this research is based on Goldberg (1989), Beasley et. al. (1993a and 1993b) and Koza (1992). The description of its integration with feature selection is based primarily on Brill *et al.* (1992).

A genetic algorithm (GA) optimises a given function (referred to in GA literature as the fitness function) by generating a *population* of multiple instantiations of the variables of the fitness function. Various operators, metaphorically akin to the processes of evolution found in life on earth, are applied to the population with the intention of optimising the fitness function.

A GA can be formally defined as

$$\mathrm{GA} = (P_0, p, g, m, S, M, R, F), \qquad \textbf{Eq. 3-15}$$

where P_0 is the initial randomly generated population of binary strings, p is the population size, g is the maximum number of generations, m is the mutation rate, S is the selection operator, M is the mutation operator, R is the recombination operator and F is the fitness function that the GA is optimising. (See Figure 3-2).

---

[24] This feature selector was chosen for two reasons: (1) the author had previously studied genetic algorithms and (2) the article by Brill *et al.* (1992) claimed the genetic feature selection algorithm to be very fast, which was definitely not the experience of this author (see Chapter 6 for the results of experiments conducted with this operator).

```
Generate the initial population of size p
Repeat for g generations, or until an acceptable solution is found
        Calculate the fitness of each individual (operator F)
        Repeat until the new population is replenished
                Select individuals for recombination (operator S)
                Recombine the selected individuals (operator R)
                Mutate the recombined individuals with probability m (operator M)
                Put the recombined individuals into the new population
        End Loop
End Loop
```

Figure 3-2: High-level pseudo-code of a typical genetic algorithm.

The following descriptions of the operators are specific to the GA implemented here.

The binary strings of the population represent the instantiations of the variables of the fitness function. Part of the process of calculating the fitness function involves converting the binary string of an individual instantiation to the appropriate fitness function variables. Usually the binary string is converted to a vector of real numbers, but here each binary value is a boolean corresponding to the on or off state of each feature. The length of the binary string is usually a function of the number of variables and the accuracy required For example, a pattern with 81 features, such as the raw images used in this dissertation, will be represented by a binary string of length 81. Only 1 bit is required for each feature because it is either represented as on or off in the binary string.

The fitness function is the only domain specific part of the standard GA. Here it measures the usefulness of a feature set by examining the ratio of the success of a feature set over its size. For each pattern, its nearest neighbour is determined using a measurement such as Euclidean distance, using only the selected features. The fitness of a feature set, f, is calculated as

$$f = \frac{c}{s}, \qquad \textbf{Eq. 3-16}$$

where c is the proportion of patterns whose nearest neighbours are in the same class as they are and s is the number of features in the feature set. This is almost identical to the method discussed in Brill *et al.* (1992).

The selection operator S selects two individuals (binary strings) to recombine (or mate, according to the evolutionary metaphor). The method considered here sorts the population strings (individuals or genotypes, according to the evolutionary metaphor)

in order of ascending fitness. A random number is then generated between 1 and the sum of all integers less than or equal to p, the population size. Each individual is allocated a portion of the range of the random number, according to its fitness. The fittest individual occupies the range from $p(p + 1) / 2$ to $p(p + 1) / 2 - p$, while the least fit individual occupies only the range from 0 to 1. Therefore, fitter individuals are more likely to be selected than less fit individuals. To be precise

$$\text{Probability}(\textit{individual is selected}) = \frac{i}{p(p+1)/2}, \qquad \textbf{Eq. 3-17}$$

where i is the position in the fitness list of a particular individual. (A more commonly employed method is Roulette Wheel Selection with Linear Scaling discussed in Goldberg, 1989.)

The recombination operator R used is called single-point crossover. At a randomly chosen position the two selected binary strings are spliced into two segments, a head and a tail segment. The tails of the two strings are interchanged with each other, producing two new individuals (see an example of this in Figure 3-3).

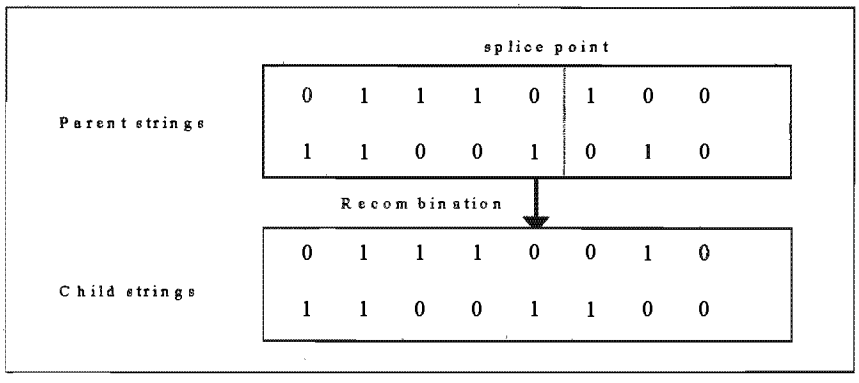


Figure 3-3: Depiction of single-point crossover.
In this example, two eight bit *parent* strings are spliced after their fifth bit. Two *child* strings are formed, each containing the first five bits of one parent and the last three bits of the other parent.

The mutation operator M randomly changes the digits of the newly generated binary strings with probability m, the mutation rate.

The values of the user-defined parameters of a GA need to be determined empirically. Typical values found in GA literature are: 50 to 100 for the population size, 10 to 100

for the number of generations and 0.0001 to 0.00001 for the mutation rate. However, these ranges are arbitrary and optimal parameter values are domain-specific.

Holland's schema theorem explains how GAs optimise their fitness functions (Holland, 1975). The details of this are beyond the scope of this research.

4. Classification Models

4.1. Introduction

This chapter describes the classification models used in this research. These are the sub-systems within a pattern recognition system corresponding to the second process (second rectangular block) in Figure 2-2. They are responsible for partitioning the pattern space based on the features they receive as input from the pre-processing sub-system within the pattern recognition system.

The classification models described in the following sections include:

- The Euclidean minimum distance classifier (statistical method).
- The Mahalanobis minimum distance classifier (statistical method).
- A minimum distance classifier that incorporates data clustering (statistical method).
- The k-NN algorithm (statistical method).
- MLP trained with Error Back-propagation.
- MLP trained with Resilient-propagation.
- ANN trained with LVQ.

The remaining sections in this chapter describe how each of these classification models work and some of their important characteristics.

Figure 4-1 depicts an example of a pattern recognition system. It shows how the pre-processing operators and the classification model are combined to form a complete pattern recognition system.

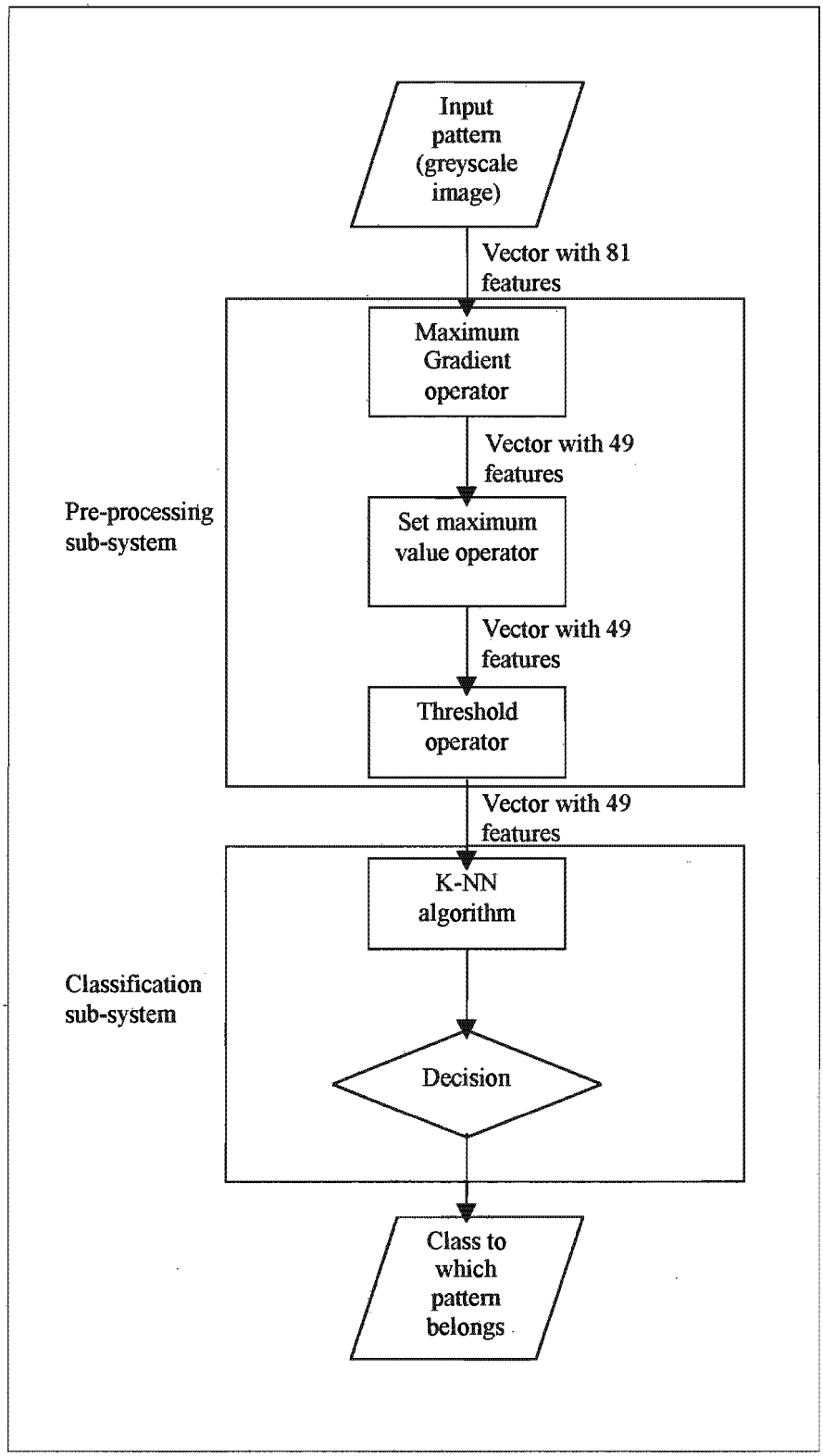


Figure 4-1: Example of a pattern recognition system.

4.2. Minimum Distance Classifiers[25]

The three minimum distance classifiers described here are appealing because of the simplicity in how they work. In addition, their classification speed is very fast. The first classifier examined, a minimum distance classifier using Euclidean distance, calculates a mean vector for each class using a training data set. Unseen patterns are assigned to the class whose mean vector has the nearest Euclidean distance. The second algorithm is an enhancement of the first, in so far as it uses the Mahalanobis distance, which takes the covariance of the pattern vector elements into account. The descriptions of the first two algorithms are based on Duda (1997). The third algorithm is a synthesis of the second algorithm and the k-Means Clustering algorithm. Instead of using a mean vector for each class, a small set of prototype vectors is determined using the k-Means algorithm.

4.2.1. Euclidean Minimum Distance Classifiers

The training phase for this algorithm requires that mean vectors be calculated for each of the different classes. For a set of training pattern vectors, $\mathbf{u}_1$ to $\mathbf{u}_n$, in class φ_i, the mean vector $\mathbf{m}$, such that $\mathbf{m}=(m_1, m_2, m_3, ..., m_d)^T$, is calculated as

$$\mathbf{m} = \frac{\sum_{j=1}^{n} \mathbf{u}_j}{n}. \qquad \text{Eq. 4-1}$$

Thus for c classes, a set of mean vectors, $\mathbf{m}_1$ .. $\mathbf{m}_i$ .. $\mathbf{m}_c$, is established (one mean vector per class). The classification stage of this algorithm assigns unclassified patterns to the class associated with the nearest mean vector. The nearest vector is established using the Euclidean distance metric. An unclassified pattern vector $\mathbf{u}$ (with d features) is assigned to a class i, using the following rule

$$\mathbf{u} = \min_i [\sqrt{\sum_{j=1}^{d} (u_j - m_{ij})^2}]. \qquad \text{Eq. 4-2}$$

Duda (1997) points out a number of potential problems that may arise with the Euclidean minimum distance algorithm:

---

[25] All the minimum distance classifiers discussed here were implemented by the author using the C programming language.

- A Euclidean minimum distance classifier will give poor results if two (or more) features are highly correlated with different scales. For instance, consider the set of patterns with their associated classes in Table 4-1.

Pattern No.	Feature 1	Feature 2	Class
1.	5	12	0
2.	19	39	0
3.	44	81	0
4.	8	7	1
5.	10	11	1
6.	44	51	1

Table 4-1: Example of correlated features.

It is easy for a human observer to notice that an approximate correlation ratio of 2:1 exists between the features for patterns in Class 0 and an approximate correlation ratio of 1:1 exists between the features for patterns in Class 1. However, the Euclidean distance between Pattern 4 and Pattern 1 is smaller than the Euclidean distance between Pattern 4 and any Pattern in Class 1. As such, the Euclidean minimum distance classifier would incorrectly classify Pattern 4. The Mahalanobis minimum distance classifier, discussed in the next section, overcomes this problem, since the Mahalanobis distance is invariant to linear transformations in the input variables, while the Euclidean distance is not.

- Figure 4-2 depicts a pattern recognition problem that requires a curved decision boundary, as opposed to a linear one for which a Euclidean minimum distance classifier is better suited. The solid straight line in the diagram corresponds to the partitioning of the pattern space that would be created by the Euclidean minimum distance classifier. All patterns below the line would be classified as *o*'s and all the patterns above the line would be classified as *x*'s. As such, the algorithm will incorrectly classify many of the patterns from both classes.

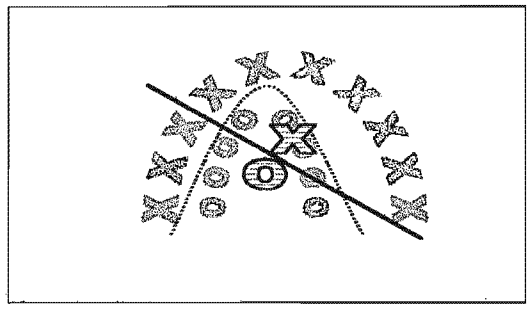


Figure 4-2: Classification problem requiring a curved decision boundary.
To separate the *x*'s from the *o*'s, a curved decision boundary is required (dotted line). The solid line corresponds to the partition an Euclidean minimum distance classifier would create. The patterned *x* and *o* correspond to the mean vectors of the *x*'s and *o*'s, respectively.

- Figure 4-3 depicts the situation where there are sub-classes of patterns clustered in different parts of the pattern space. In this example the mean pattern vector for the patterns in class *x* will not be representative of either of the *x* clusters and it is much closer to the *o*'s. As such the Euclidean minimum distance classifier will classify some *x*'s as *o*'s and vice-versa. The minimum distance classification algorithm with clustering discussed in Section 4.2.3 attempts to address this issue.

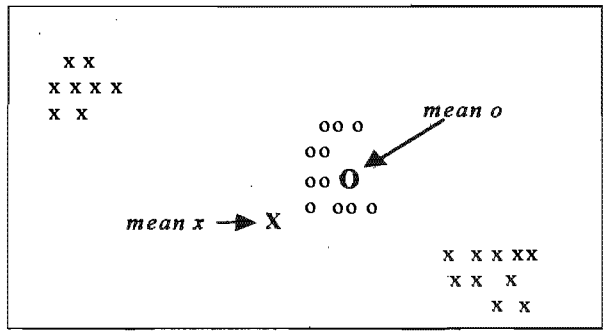


Figure 4-3: Sub-classes existing within a class.
The mean pattern vector for the *x*'s is actually far away from the *x*'s, but very close to the *o*'s.

- The feature space might be complex and it may only be possible to separate the classes using a highly non-linear pattern classification algorithm. In this case, ANNs or the k-NN algorithm will almost always perform better than the minimum distance classifiers.

4.2.2. Mahalanobis Minimum Distance Classifiers

The Mahalanobis minimum distance algorithm generally achieves better classification accuracy than the Euclidean minimum distance algorithm for a number of reasons. It is invariant to rotation, scaling and other linear transformations of the data; the correlation problem between variables is overcome; and it is capable of curved, as well as linear decision boundaries (Duda, 1997). However, this method is more expensive than the Euclidean minimum distance classifier in both training and classification time[26], because in the training phase, a covariance matrix (and its inverse) must be calculated for each class and, in the classification phase, the inverse of the covariance matrix for each class is incorporated into the distance calculation.

As with the Euclidean minimum distance algorithm, the training phase for this algorithm consists of calculating the mean vector, $\mathbf{m}$, for each class (see Eq. 4-1). In addition, for a set of n training pattern vectors — $\mathbf{u}_1$ to $\mathbf{u}_n$, with d features, such that $\mathbf{u}_i=(u_{i1},u_{i2}...u_{id})^T$ — in class φ, the symmetrical d by d covariance matrix, Σ, must be calculated. For two variables (or features), i and j ($i <= d, j <= d$), there covariance matrix entry $\Sigma(i,j)$ is defined as

$$\Sigma(i,j)=\frac{\sum_{k=1}^{n}(u_{ki}-m_i)(u_{kj}-m_j)}{n-1}. \qquad \textbf{Eq. 4-3}$$

Note that if i tends to increase with j, $\Sigma(i,j)$ is positive. If i decreases as j increases, $\Sigma(i,j)$ is negative. If they are independent, $\Sigma(i,j)$ is zero. One covariance matrix is calculated per class. (Duda, 1997)

The classification stage is identical to the Euclidean minimum distance algorithm, except that the Mahalanobis distance metric is used. An unclassified pattern vector $\mathbf{u}$ (of dimension d) is assigned to class i, using the following equation:

$$\mathbf{u} = \min_i[(\mathbf{u}-\mathbf{m}_i)^T \Sigma_i^{-1}(\mathbf{u}-\mathbf{m}_i)],\ i=1...c. \qquad \textbf{Eq. 4-4}$$

---

[26] It is still much faster than the other classification algorithms used in this research.

4.2.3. Clustering with Minimum Distance Classification

Input:
1.) A set of patterns $\mathbf{p}_1$ .. $\mathbf{p}_n$, each with an associated class indicator, c_1 .. c_n, assigning the pattern to one of m classes φ_1.. φ_m.
2.) *K*, the number of clusters.

Find a set of K cluster centres (means).

Data:
1.) The cluster centres, ($\mathbf{w}_1$..$\mathbf{w}_k$), all vectors with the same dimension as the patterns.

Procedure:
Initialise the cluster centres ($\mathbf{w}_1$..$\mathbf{w}_k$) to the first *K* cluster centres. (There are variations on this.)
Repeat
    Assign patterns to their closest cluster centre.
    For i := 1 to n
        Assign $\mathbf{p}_i$ to the nearest cluster centre (using Euclidean distance)
    End For

    Compute the new centres.
    For j := 1 to k
        $\mathbf{w}_j$:= the mean of the patterns assigned to it.
    End For
Until there is no change in the cluster assignments between iterations.

Figure 4-4: K-Means Clustering algorithm.
(Based on Hush and Horne, 1993.)

If subclasses exist within the classes of the pattern domain (see Figure 4-3), a clustering algorithm, such as k-Means Clustering (see Figure 4-4), can be combined with the classifiers just described. Instead of determining one mean vector per class, one mean vector per cluster is calculated. In some cases this might overcome the problem of the patterns in a class being distributed in different areas of the pattern space. Therefore, two modifications to the training stage of the Mahalanobis minimum distance algorithm have been implemented. The first modification uses k-Means Clustering to find clusters in the classes. For the second modification, means are calculated for each of the clusters, not the classes. Using clustering to find sub-classes is a frequently used idea in pattern recognition. For example, Hush and Horne (1993) summarise how clustering is used in the training stage of a Radial Basis Function Network, an ANN model designed by Broomhead and Lowe (1988). Another example is an ANN developed by Kia and Coghill (1992) which uses a method that they call dynamic clustering to adapt the weights of the network.

Employing clustering in conjunction with minimum distance classifiers is not guaranteed to eliminate the sub-class problem. In almost all practical problems it is not known in advance how many sub-classes there are, but most clustering algorithms (such as the k-Means Clustering algorithm) require that an arbitrary number of clusters per class be specified in advance. In addition, even if the number of sub-classes matches the number of pre-specified clusters, there is no guarantee that the clusters found by the clustering algorithm will correspond precisely to the optimal set of sub-classes for overcoming the *sub-classes within a class* problem. Choosing a representative training data set is critical for ensuring that all sub-classes are represented.

4.2.4. Time Complexity of the Minimum Distance Classifiers

For a Euclidean minimum distance classifier to classify one pattern, the Euclidean distance must be calculated between that pattern and the class means. For m classes, classification of one pattern is $O(m)$ [27]. Since m is usually very small and the Euclidean distance calculation requires little computation, classification is extremely fast. Practically, this algorithm executes in negligible time, even for large validation data sets.

The Mahalanobis minimum distance classifier has the same time complexity as the Euclidean minimum distance classifier for training and classification: O(n) and O(m), respectively. However, the basic operations on each iteration involve far more computation than the Euclidean Minimum Distance Classifier (compare Eq. 4-1 to Eq. 4-3 for training and Eq. 4-2 to Eq. 4-4 for classification). It is therefore linearly slower than the Euclidean minimum distance classifier.

Introducing clustering further slows down both training and classification. The k-Means Clustering algorithm significantly slows down training, since the algorithm

---

[27] The time and space complexity of algorithms is a topic of undergraduate computer-science and knowledge of its notation and terms are assumed here. See Baase (1988) for a description. In this discussion, a time complexity of $O(n)$ implies that a basic operation - where the term *basic operation* is specifically defined for each algorithm - executes n (or a linear function of n) times.

executes a number of iterations, depending on the training set, over the patterns. Assuming there are k clusters per class, classification complexity is $O(km)$. [28]

4.3. K-Nearest Neighbour Algorithm[29]

The k-NN rule is a nonparametric decision-theoretic algorithm. This means that it assigns patterns to categories without making any assumption about the class densities (see Section 2.2 and Section 2.3). Cover and Hart (1967) studied the nearest neighbour rule as a tool for pattern recognition. They determined lower and upper bounds for the error rates of the algorithm. The lower bound is Bayes' probability of error (see Section 2.2). More importantly, they determined that the upper bound is at most twice the Bayes' Error. The implication of this is that 'half of all the available information in an infinite training sample set is contained in the nearest neighbour' (Dasarathy, 1991, p. 2).

The Nearest Neighbour (NN) rule assigns an unclassified pattern to the same class as its nearest neighbour in the training set. There are a number of ways of defining the nearest neighbour. These definitions are referred to as distance measurements. The most common distance measurement is the Euclidean distance. However there are many others (see Table 4-2 for examples). The k-NN rule (k-NN) is a slight extension to the NN one. The k — where k is a pre-specified positive integer — nearest neighbours are determined. The unclassified pattern is assigned to the class in which most of the neighbours are classified. In the case of a tie between competing classes, either the pattern remains unassigned, or an arbitrary tie breaking procedure is invoked (such as randomly selecting one of the winning classes). Therefore, the NN rule is simply the k-NN rule with k set to one.

Unlike the other classification models studied here, there is no estimation (or training) stage for k-NN, only a classification stage. It is, of course, necessary to collect a training data set of pre-classified patterns so that the neighbourhood of an unclassified

---

[28] This is easily demonstrated: a pattern must be compared against the k cluster mean vectors of each of the m classes.

[29] The standard k-NN algorithm was implemented by the author using the C programming language.

pattern exists. The larger this set, the better the accuracy of the algorithm — assuming it is unbiased — but the slower its execution time, as discussed in the next section.

As Dasarathy (1991), points out as larger values of k are used, the less the individual samples, which could be outliers, influence the decision process. However, if k is set too large (relative to the training set size), the 'neighborhood under review is no longer ... the immediate neighborhood of the sample and the very basis of the nearest neighbor principle is sacrificed' (Dasarathy, 1991, pp. 5). Therefore, k is usually in the range $1 < k << sample\ size$.

Many advances have been made to the standard k-NN algorithm. Dasarathy (1991) and Devijer (1980 and 1982) contain seminal papers and reviews of these advancements. Of note is that the k-NN algorithm has been implemented as an ANN (Montgomery and Vijaya Kumar, 1986a and 1986b). Figure 4-5 contains a high-level pseudo-code description of the k-NN algorithm.

Distance Measure Name	Mathematical Definition
City block	$D(\mathbf{u}, \mathbf{v}) = \sum_{i=1}^{n} \lvert u_i - v_i \rvert$
Chebyshev	$D(\mathbf{u}, \mathbf{v}) = \max_i(\lvert u_i - v_i \rvert), i=1..n$
Euclidean	$D(\mathbf{u}, \mathbf{v}) = \sqrt{\sum_{i=1}^{n} (u_i - v_i)^2}$
Mahalanobis	$D(\mathbf{u}, \mathbf{v}) = (\mathbf{u} - \mathbf{v})^{\mathrm{T}} \Sigma^{-1} (\mathbf{u} - \mathbf{v})$, where Σ is the covariance matrix for the class to which u belongs.
Non-linear	$D(\mathbf{u}, \mathbf{v}) = \begin{cases} \text{Constant}, & \delta(\mathbf{u}, \mathbf{v}) > \text{threshold} \\ 0 & \delta(\mathbf{u}, \mathbf{v}) \leq \text{threshold} \end{cases}$, where δ is a non-linear function.
Quadratic	$D(\mathbf{u}, \mathbf{v}) = (\mathbf{u} - \mathbf{v})^T Q (\mathbf{u} - \mathbf{v})$, where Q is a symmetric positive definite matrix.

Table 4-2: Examples of Distance Measures.
***n* is the vector dimension, u is a vector drawn from the sample, for which the classification is known, and v is the vector being classified. (Sources: Kittler, 1986 and Duda, 1997.)**

Input:
1.) A set of patterns $\mathbf{p}_1 .. \mathbf{p}_n$, each with an associated class indicator, $\mathbf{c}_1 .. \mathbf{c}_n$, assigning the pattern to one of m classes $\varphi_1 .. \varphi_m$
2.) k (a user defined positive integer.)

Algorithm to classify a single unclassified pattern **u**.
Data:
1.) An array (**B**), $B_1 .. B_k$, of class indicators.
2.) An array (**C**), $C_1 .. C_n$, of class indicators.
3.) An array (**D**), $D_1 .. D_n$, for storing distances.
Procedure:
For $i = 1$ to n
    Measure the distance, D_i, between $\mathbf{p}_i$ and **u**.
    Store the class, $\mathbf{c}_i$, of $\mathbf{p}_i$ into C_i.
End For
Find the classes of the k smallest distances and assign them to $\mathbf{B}_1 .. \mathbf{B}_k$.
Assign **u** to the φ that appears most often in the **B** array.
(In the case of a tie, either indicate that the class is unknown or invoke a further algorithm to break ties, e.g. random selection.)

Figure 4-5: High-level pseudo-code of the k-Nearest Neighbour algorithm. (Unoptimised for the sake of clarity.)

4.3.1. Improving the K-Nearest Neighbour algorithm

One of the most important drawbacks of the standard k-NN algorithm is its classification execution time. Unlike the other algorithms presented in this thesis, no learning stage occurs in this one. Therefore, all information processing takes place in the classification stage. A serially implemented, unoptimised, version of the k-NN algorithm performs the distance calculation for every pattern in the sample set in order to classify one pattern. This part of the algorithm is by far the most expensive and has a time complexity of $O(n)$ where n is the number of samples in the sample set. If the distance calculations were to be made in parallel, the time complexity — which would be $O(1)$ — improvement would be countered by the increase in space complexity, since all the patterns would have to be stored in random access memory at the same time.

There have been two major approaches to optimising k-NN classifiers. The first looks at ways to reduce the size of the training data set, so that n is reduced without seriously impairing classification accuracy (referred to as editing the training set), while the second looks at ways to improve the algorithm itself. Both these approaches usually involve introducing a pre-processing stage akin to training. Some examples of each of these approaches are examined very briefly.

- *Editing the training set.*

Hart (1968) proposed the following method to reduce the training set. A new training subset composed of randomly selected, but representative, examples, is chosen from each class. The training patterns not included in the subset are classified according to the k-NN rule. If they are classified incorrectly they are added to the subset. The process is repeated until no more patterns are added to the subset. This method ensures consistency because no patterns in the original training set would be incorrectly classified using the new training set, but it is theoretically possible, though unlikely in practice, for the new training set to be equivalent to the old one. This method can be computationally expensive, but it is akin to an estimation (training) phase and should result in a substantial reduction in processing time for classification.

An interesting editing rule was proposed by Wilson (1972). He showed that for a number of problems, Bayes' probability of error is approached more quickly using his rule than the standard k-NN rule alone. This rule requires that each pattern in the sample set be classified using the k-NN rule (Wilson set k to 3). The pattern itself is not considered when it is being classified, since it will obviously always be its own nearest neighbour. The class to which it is assigned using the k-NN rule is then compared to the pattern's pre-classified class. If they do not match, then the pattern is marked for deletion. Once this process has been applied to all the sample set patterns, those marked for deletion are removed from the sample set.

Note that the minimum distance classifiers, discussed in the previous section, can be viewed as k-NN algorithms with severe editing applied to their training sets. The training set is reduced to one prototype pattern per class (or a few prototypes when clustering is applied).

- *Algorithm modifications.*

Branch and bound (BAB) algorithms have provided a successful line of research into improving the execution speed of k-NN algorithms. A BAB

algorithm reduces the number of distance calculations that need to be made in the standard k-NN algorithm. It uses a tree structure to store the patterns in the sample set. The process of determining the ordering of the patterns in the tree is called a hierarchical decomposition. Instead of computing the distance between an unclassified pattern and all the patterns in the sample set, only the distances between the unclassified pattern and certain nodes in the tree are calculated (a node contains one pattern). Using certain rules, it can be determined whether a node is a nearest neighbour or not. If it is not, the entire sub-tree to which the node belongs can be eliminated, thereby eliminating a subset of patterns from consideration. The efficiency of BAB methods depends on the hierarchical decomposition employed and the rules for determining nearest neighbours (Niemann and Goppert, 1988). Detailed BAB algorithms for the k-NN rule have been proposed by Fukunaga and Narendra (1975), Kamgar-Parsi and Kanal (1985) and Niemann and Goppert (1988), among others.

Additional, implementation orientated ways of speeding up the k-NN algorithm are to use a distance measurement such as the City Block distance as opposed to the Euclidean distance, since the squaring and square root operations are eliminated in the former. Using integers to represent the patterns instead of floating point numbers (perhaps sacrificing some accuracy in the process) would also significantly speed up the algorithm. Other methods involved in speeding up k-NN classification involve using ANN concepts (Montgomery and Kumar, 1986a and 1986b) and special purpose hardware (Loizou and Maybank, 1987).

4.4. MLP Trained with Error Back-propagation

This section describes the Error Back-propagation (EBP) algorithm as formulated by Rumelhart *et al.* (1986). The aim of training an MLP in pattern recognition problems is to find a set of weights such that for each input vector, the output response is equal or very close to, the category to which the pattern belongs. EBP attempts to achieve this aim by minimising an error function which computes the mean squared difference

between the desired response and actual response of the ANN. It is typically applied to an MLP with the following characteristics:

- There is a one-to-one correspondence between the elements of the input pattern vector and the input neurons. Each input neuron fires with the same value as its corresponding input pattern vector feature (i.e. the input neurons execute the identity function).

- The total input, x_j, to a hidden or output neuron j, is a linear function of the outputs, y_i, (i =1 to n), of the neurons connected to x_j by weights w_{ji}. Typically,

$$x_j = \sum_{i=1}^{n} y_i w_{ji} . \qquad \textbf{Eq. 4-5}$$

- Every hidden and output neuron has a threshold associated with it. This is usually implemented by connecting weights from a *bias* neuron to all the hidden and output neurons. The bias neuron has a fixed output of 1.

- Every hidden and output neuron produces a real-valued output, o, which is a non-linear, continuous, differentiable, function of its total input x. Typically, the logistic function, which has a range of (0 .. 1), is used

$$o = \frac{1}{1+e^{-x}} . \qquad \textbf{Eq. 4-6}$$

- The weights of the network are initialised to random values (typically in the range [–1 .. 1]).

The error function is the mean squared error between the actual response and the desired response. Given a training set of p patterns, m output neurons (or classes), a desired response d and an actual response o, the total mean squared error, E, is defined as

$$E = \frac{1}{2} \sum_{i}^{p} \sum_{j}^{m} (o_{ji} - d_{ji})^2 . \qquad \textbf{Eq. 4-7}$$

EBP minimises E using gradient descent. The algorithm's name comes from the fact that it propagates derivatives from the output layer back to the input layer. By calculating the partial derivative of the error with respect to each weight, the direction

in which that weight must be moved in order to reduce the error can be determined. For the output layer, the change in the error E with respect to each weight w_{ji}, connected from neuron i in the last hidden layer to neuron j in the output layer, is the partial derivative

$$\frac{\partial E}{\partial w_{ji}} = -o_i \frac{\partial o_j}{\partial x}(d_j - o_j). \qquad \textbf{Eq. 4-8}$$

For the logistic function the derivative is

$$\frac{\partial o}{\partial x} = o(1-o). \qquad \textbf{Eq. 4-9}$$

For the output layer, the algorithm modifies each weight by Δw_{ji}, which is a negative of a constant, ε, multiplied by Eq. 4-8. Therefore,

$$\Delta w_{ji} = -\varepsilon\, o_i o_j (1-o_j)(d_j - o_j). \qquad \textbf{Eq. 4-10}$$

The constant, ε, is called the *learning* parameter.

The changes that must be applied to weights in the hidden layers are more complex. Rumelhart *et al.* (1986) developed the idea of propagating the error in the output layer neurons back to the hidden layer neurons. It is useful to recursively define a new term, δ. For a neuron, j, in the output layer

$$\delta_j = \frac{\partial o_j}{\partial x}(d_j - o_j), \qquad \textbf{Eq. 4-11}$$

and for a neuron i, in a hidden layer

$$\delta_i = \frac{\partial o_i}{\partial x}\sum_j \delta_j w_{ji}, \qquad \textbf{Eq. 4-12}$$

where j is an index over the neurons that neuron i is connected to in the subsequent layer.

Using this recursive definition of δ, a general equation for the change of all the weights in the MLP can be formulated. For weight w_{ji}, connecting neuron i to neuron j in the subsequent layer, the EBP algorithm changes the weight by Δw_{ji}, which is defined as

$$\Delta w_{ji} = -\varepsilon o_i \delta_j. \qquad \textbf{Eq. 4-13}$$

The standard EBP algorithm is susceptible to a number of critical problems: (1) getting trapped in local minima, (2) extremely slow convergence, (3) sensitivity to initial conditions and (4) oscillating weights (Sarker, 1995). A slight modification to the definition of Δw_{ji}, suggested in Rumelhart *et al.* (1986) is to add — what is referred to in ANN literature as — *momentum*. The change in weight on an iteration of the algorithm is taken into account in the next iteration. Therefore,

$$\Delta w_{ji}(t+1) = -\varepsilon o_i \delta_j + \eta \Delta w_{ji}(t), \qquad \textbf{Eq. 4-14}$$

where t is an index over the iterations of the algorithm and η is a constant, usually set between 0 and 1. There are two intuitive purposes of this enhancement. Firstly it speeds up convergence because the weight changes are larger if they are in the same direction on consecutive iterations. Secondly, should the network become trapped in a local minimum the momentum term can assist it in climbing over the ridge in which it is trapped. The better performance of EBP with a momentum term is backed up by empirical evidence (e.g. Sarker, 1995) and it is almost always included in any explanation of the standard EBP algorithm.

It is this version of the EBP (i.e. using the logistic activation function and the momentum term) that is used in this research[30]. In addition, the weights can be updated after each training pattern is shown to the network or after the entire set of training patterns is shown to the network. The former is called on-line training, while the latter, which is used in this research, is referred to as batch training. In fact, the error definition given in Eq. 4-7 is for batch training[31].

The ideal values for the two constants, ε and η, have been the subject of some research. Empirical evidence gathered by Dai and Macbeth (1997) has shown that the

---

[30] The algorithm was executed in this research using the Stuttgart Neural Network Simulator (SNNS – Mache, 1997).

[31] The average weight update for batch and on-line training is slightly different. On-line training requires less storage, but the batch method measures the error gradient more accurately (Haykin, 1994). There are practical situations where one method will be more suitable than the other. For instance, real-time learning requires on-line training. For the applications in this research, the accuracy of batch learning makes it more appropriate.

values of the parameters effect the speed of convergence, but do not effect the overall performance or structure of the ANN. In this research the learning and momentum parameters have been set to 0.8 and 0.2, respectively.

Many other improvements on the original algorithm have been suggested, mostly with the aim of speeding up convergence. Sarker (1995) contains an excellent summary of these. Some simple suggestions include: (1) using different learning and momentum parameters for every weight, (2) using alternatives to the logistic function for neuron activation and (3) using momentum terms that are a function of all previous iterations, not just the last one. These improvements have some empirical support.

4.5. MLP Trained with Resilient-propagation

Resilient-propagation (RPROP), developed by Riedmiller and Braun (1993), is a variation on EBP developed with the aim of eliminating 'the harmful influence of the size of the partial derivative (Eq. 4-8) on the weight step' (Mache, 1997[32]). The algorithm is similar to EBP, except that the weights are updated by a value determined by the **sign of the gradient** of the error with respect to the weight only, **not by its size**. The change in weight, Δw_{ji}, for a weight connected from neuron i to neuron j is

$$\Delta w_{ji} = \begin{cases} -\Delta_{ji}, & \text{if } \frac{\partial E}{\partial w_{ji}} > 0 \\ +\Delta_{ji}, & \text{if } \frac{\partial E}{\partial w_{ji}} < 0 \\ 0, & \text{else} \end{cases}, \qquad \textbf{Eq. 4-15}$$

where Δ_{ji} is the value with which the weight is updated. This value is modified between iterations. If the sign of the partial derivative changes, the update value is decreased, else it is increased. Formally,

---

[32] http://www.informatik.uni-stuttgart.de/ipvr/bv/projekte/snns/UserManual/node152.html.

$$\Delta^{(t)}{}_{ji} = \begin{cases} \eta^+ . - \Delta^{(t-1)}{}_{ji} , & \text{if } \frac{\partial E}{\partial w_{ji}}^{(t)} . \frac{\partial E}{\partial w_{ji}}^{(t-1)} > 0 \\ \eta^- . \Delta^{(t-1)}{}_{ji} , & \text{if } \frac{\partial E}{\partial w_{ji}}^{(t)} . \frac{\partial E}{\partial w_{ji}}^{(t-1)} < 0 \\ \Delta^{(t-1)}{}_{ji} , & \text{else} \end{cases} , \qquad \textbf{Eq. 4-16}$$

where (η^+) and (η^-) are the increase and decrease factors, respectively, and t is an index over the iterations of the algorithm.

The RPROP algorithm was executed in this research, using the Stuttgart Neural Network Simulator (SNNS – Mache, 1997). The implementation sets η^+ to 1.2 and η^- to 0.5. The definition of the error is slightly different to the one used in Eq. 4-7, in so far as an extra term is added. A weight decay term is introduced to prevent the weights from becoming too large, which can result in the overfitting problem discussed in Section 2.6.4. The equation for the error is

$$E = \frac{1}{2}\sum_{i}^{p}\sum_{j}^{m}(o_{ji} - d_{ji})^2 + 10^{-\alpha}\sum_{k}^{W} w_k^2 . \qquad \textbf{Eq. 4-17}$$

where k is an index over the number of weights W, w is a weight and α is a user-defined constant. Note that as the weights get larger, the extra term gets larger, thereby increasing the size of the error.

In this research, the value of α has been set to 4 and the weight update values have been initialised to 0.2 (i.e. the Δ^0's). In addition a ceiling of 50 on the size of the weight update values has been set, since the network would behave unpredictably if the weights were allowed to change by very large amounts. In contrast, in order to try and prevent the network from becoming stuck in local minima, an update value floor of 10^{-6} has been set.

4.6. Learning Vector Quantization[33]

Kohonen (1986) has developed a simple, but effective, ANN learning algorithm primarily used for pattern classification, called Learning Vector Quantization (LVQ). The ANN model, for which LVQ was developed, consists of a set of output layer neurons, fully connected with weights to a layer of input neurons, with each input neuron executing the identity function on a corresponding feature in the pattern. Each output neuron has a pre-determined class associated with it. There are generally more output neurons than classes, and multiple output neurons are assigned to one class. The transfer (firing) function of each output neuron, o, is the inner product of the weights attached to it (represented by a vector, $\mathbf{w}$) and the input pattern vector, $\mathbf{x}$, such that

$$o(\mathbf{x}) = \mathbf{w}^T \mathbf{x} . \qquad \textbf{Eq. 4-18}$$

The input pattern is assigned to the class of the best-matching, or winning, neuron, which is defined to be the output neuron which fires the highest value. If the Euclidean norms of all the input vectors are equal to the same number (using an operation such as z-axis normalisation), then the same result (as taking the inner product) is achieved by selecting the output neuron whose weight vector and input pattern have the minimum Euclidean distance between them. Formally,

$$\textit{Index of winning output neuron} = \arg_j \min \| \mathbf{x} - \mathbf{w}_j \|, \qquad \textbf{Eq. 4-19}$$

where j is an index over the output neurons.

As such, the network consists of a set of m weight vectors, one for each output neuron, consisting of n variables, the number of features in the input patterns. Kohonen *et al.* (1992) refer to these as codebook vectors. LVQ is a stochastic, iterative process. An iteration consists of randomly selecting a pattern vector from the training set. The closest codebook vector to the pattern vector is found. If the class of the pattern vector matches the class of the codebook vector, the codebook vector is modified so that the Euclidean distance between the two vectors is reduced. If the class of the pattern vector does not match the class of the codebook vector, the

---

[33] In this research, the implementation of Kohonen *et al.* (1992) was used.

codebook vector is modified so that the Euclidean distance between the two vectors is increased. Formally, if $\omega_x = \omega_w$, then

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \alpha_t[\mathbf{x} - \mathbf{w}(t)], \qquad \text{Eq. 4-20}$$

where $0 < \alpha_t < 1$, and t is an index over the iterations of the algorithm.

If $\omega_x \neq \omega_w$, then

$$\mathbf{w}(t+1) = \mathbf{w}(t) - \alpha_t[\mathbf{x} - \mathbf{w}(t)]. \qquad \text{Eq. 4-21}$$

The learning parameter, α, can be a constant or decrease monotonically with the number of iterations. It is typically initialised to a value smaller than 0.1. In Kohonen's implementation, which is used in this research, the value decreases linearly with each iteration of the algorithm. Figure 4-6 is an example of the ANN for which LVQ is designed. Note that LVQ is an on-line learning process.

The input data must be scaled, using an operator such as z-axis normalisation, so that the Euclidean norms of all the patterns are identical in order to prevent patterns with features with large values having a greater effect on the weight adaptation process, described by the above equations, than patterns with features with small values.

Two important issues that must be resolved when developing this model are (1) the initialisation of the codebook vectors (or the synapse weights) and (2) determining the number of codebook vectors (or output neurons) to use. The former issue is resolved in the Kohonen *et al.* (1992) implementation by initialising the vectors to patterns in the training set. An alternative, discussed in Haykin (1994), is to initialise the vectors using another model developed by Kohonen, the Self-organising Feature Map (SOFM), which is not discussed here, since it has not been used in this research.

The number of codebook vectors to use was determined empirically in this research. A small number of vectors was used to start off. The number was gradually increased until the accuracy of the ANN, with respect to the training set, stopped increasing significantly.

It is interesting to note the similarities between the k-NN algorithm with k set to 1 and LVQ. The classification phases of these algorithms actually work identically. However, the standard k-NN rule uses the entire training set to find the nearest

neighbour to the unclassified pattern, while LVQ uses the set of codebook vectors. Therefore, LVQ can be viewed as a method of editing the training data set for the k-NN algorithm. As a result of this the time complexity of its classification phase compares favourably with k-NN.

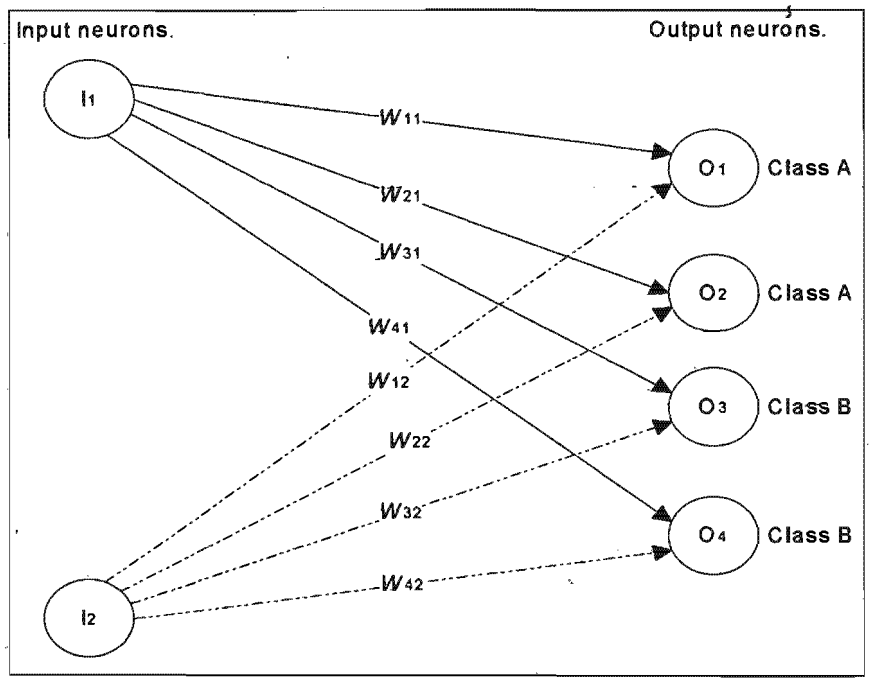


Figure 4-6: Example of the ANN model for which LVQ was designed. The ANN has two input neurons connected to four output neurons representing two classes. Multiple output neurons are used per class in order to represent sub-classes. The directed lines represent the weights from the input neurons to the output neurons. There are four codevectors, c_1, c_2, c_3 and c_4 equal to (w_{11}, w_{12}), (w_{21}, w_{22}), (w_{31}, w_{32}) and (w_{41}, w_{42}), respectively. If, for example, an input pattern (i_1, i_2) is closest to c_2 then the pattern is assigned to class A.

5. Image Recognition Problems

5.1. Introduction

This chapter describes the nondeterminable affine parameter problem and briefly mentions solutions to it suggested by Baltsavias (1991). As mentioned in the Abstract, in this thesis the problem has been reformulated as an artificial problem and a real-world problem. For the former, a data set of images was artificially constructed based on an idealisation of the problem and, for the latter, a data set of digital photographic sub-images was collected in order to solve the nondeterminable affine parameter problem as it occurs in reality. The construction of these data sets is described. Minimum requirements in terms of accuracy and speed are then given for a pattern recognition system to be considered to have successfully solved the two problems.

5.2. Nondeterminable Affine Parameter Problem

One of the aims of Digital Photogrammetry is to recover the elevation of a terrain from two or more two-dimensional images, thereby leading to a digital surface model (DSM). Image matching is a critical part of the process of generating a DSM. Given multiple photographs of a surface[34], it is the aim of an image matching algorithm to find the corresponding points on each photograph. Once this has been accomplished, the depth co-ordinates of the surface can be determined using rigorous photogrammetric techniques.

Many image matching techniques have been developed over the last sixty years. These are traditionally divided into area based matching techniques and feature based matching techniques. Heipke (1996) and Baltsavias (1991) contain summaries of the major algorithms. Feature based techniques involve detecting interesting features in all the images and then determining their correspondence, while area based techniques match corresponding points on images by examining the area surrounding the points.

---

[34] These surfaces can be any type of object. Typically it is a landscape, but rocks, pipes, car doors, sunken ships and propellers are also objects for which DSMs have been constructed.

An important characteristic of these techniques is that they are capable of achieving sub-pixel accuracy. Adaptive Least Squares Matching (ALSM), a popular and successful area based matching technique, is the algorithm on which this work is based. ALSM was developed by Gruen (1985). Baltsavias (1991) has enhanced the original algorithm, by exploiting *a priori* known geometric information and catering for multiple photographs. This extended algorithm is called Multiphoto Geometrically Constrained Matching (MPGC). It has been used extensively in practice with considerable success. For instance, the UCT Geomatics Department has used MPGC to reconstruct DSMs of informal settlements in Cape Town and the Laetoli Footprints, an archaeological site in Tanzania.

The following description (and the mathematical details which follow) of ALSM is based on Gruen (1985), Van Der Merwe (1995) and Heipke (1996).

Assume two digital photographs have to be matched. One photograph is designated as the template and the other is designated as the picture. The template is divided into a set of sub-images (or patches). For each template patch, it is necessary to find the corresponding patch on the picture. The patches can be described by discrete two-dimensional greyscale functions: $f(x,y)$ for the template patch and $g(x,y)$ for the picture patch. The ideal aim of the matching process is to correlate the two functions such that

$$f(x,y) = g(x,y) \qquad \textbf{Eq. 5-1}$$

where x and y are the greyscale co-ordinates of the patches.

In realistic situations, images disturbances such as noise, occlusions or differing radiometric factors between the photographs occur, and a perfect match is impossible. Therefore an error vector, $e(x,y)$, is introduced into the equation such that

$$f(x,y) - e(x,y) = g(x,y). \qquad \textbf{Eq. 5-2}$$

Using *a priori* geometric knowledge (described in Baltsavias, 1991), an estimate is made of the position of the template patch. The algorithm proceeds iteratively by calculating the squared difference[35] between the template and picture patch greyscale

---

[35] The squared difference between two vectors u, v of dimension n is $\frac{1}{n}\sum_{i=1}^{n}(u_i - v_i)^2$.

values and then shifting the picture patch. The picture patch is also scaled and rotated in order to account for image disturbances. The shifting, scaling and rotation (or shear) of the patch can be represented by an affine transformation with six parameters, two for shift, two for scale and two for shear. The goal of the algorithm is to find a picture patch such that the squared difference is minimised (Figure 5-1 and Figure 5-2 depict the matching process graphically). This has to be done iteratively using a least-squares approximation.

Least squares models with intrinsically linear parameters are described in introductory statistics textbooks (e.g. Rice, 1988). However, minimising Eq. 5-2 requires a least squares transformation that is *intrinsically* non-linear[36] in its parameters and more complex than the linear case. Draper and Smith (1981) contains a description of non-linear estimation, while Gruen (1985) and Baltsavias (1991) contain thorough descriptions of the least squares solution to the image matching problem. The following description leaves out some details (such as a parameter for radiometric correction) since this thesis is more concerned with the pattern recognition problem that arises in ALSM, than its algorithmic details.

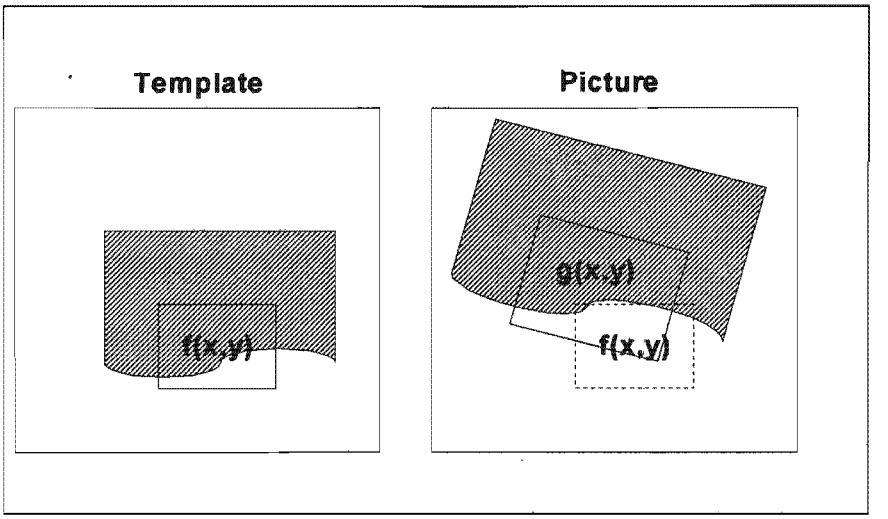


Figure 5-1: Idealised graphical depiction of ALSM.
The f(x,y) window is the template patch. ALSM determines the corresponding patch, g(x,y), on the picture, by determining the parameters of an affine transformation of f(x,y) (based on a diagram in Van Der Merwe, 1995).

---

[36] It is not possible to make an intrinsically nonlinear least squares observation equation linear, other than by using approximation methods, such as a Taylor Series, which iteratively solves closer approximations of the observation equation (Draper and Smith, 1981).

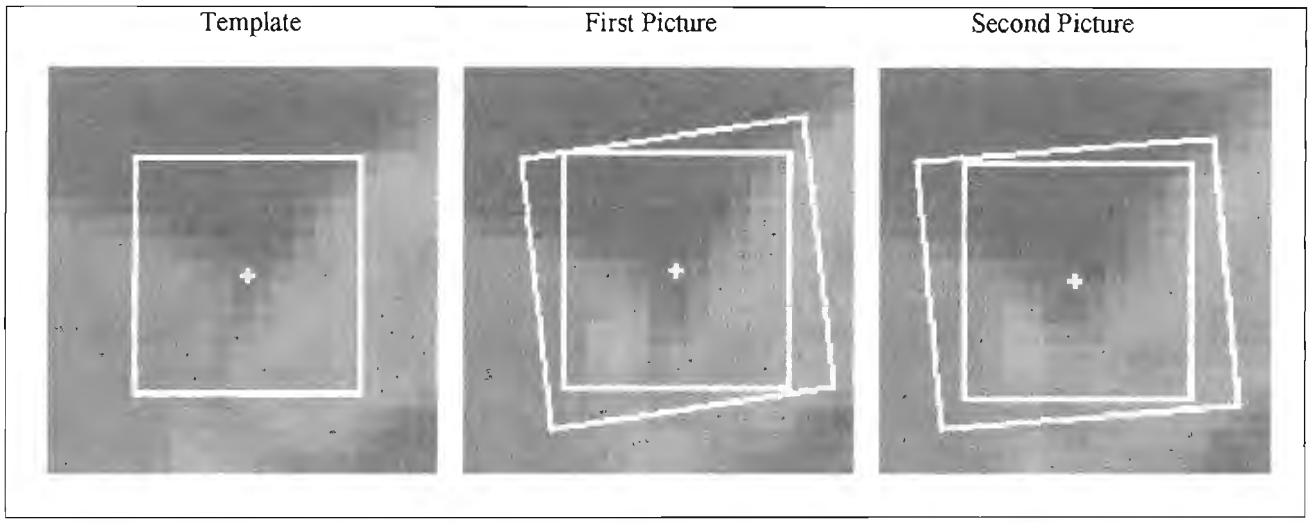


Figure 5-2: Computer generated example of MPGC.
The squares in the centres of the picture images are the initial approximations of the template patch. The rhombuses are the more accurate estimates of the template patch after the algorithm has affine-shaped. The crosses represent the particular point being matched. (Created using PVD, a MPGC matching program developed by ETH.)

Since a least square minimisation of Eq. 5-2 is non-linear in its parameters, it is approximated using the Taylor Expansion which is linear:

$$f(x,y) - e(x,y) = g^0(x,y) + (\frac{\delta g(x,y)}{\delta x})dx + (\frac{\delta g(x,y)}{\delta y})dy \qquad \textbf{Eq. 5-3}$$

where $g^0(x,y)$ is the initial estimation of the picture patch. Eq. 5-3 is a least squares observation equation. For notational simplicity, let

$$g_x = (\frac{\delta g(x,y)}{\delta x})dx \qquad \textbf{Eq. 5-4}$$

and

$$g_y = (\frac{\delta g(x,y)}{\delta y})dy\,. \qquad \textbf{Eq. 5-5}$$

The derivatives, g_x and g_y are calculated using edge detection techniques, such as the Sobel operator (discussed in Section 3.3).

The affine transformation to calculate the new co-ordinates x' and y' of each x and y co-ordinate on $g(x,y)$ is

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} u_{11} \\ v_{11} \end{pmatrix} + \begin{pmatrix} u_{12} & u_{21} \\ v_{12} & v_{21} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}, \qquad \textbf{Eq. 5-6}$$

where u_{11} and v_{11} are the shift parameters, u_{12} and v_{21} are the scale parameters and u_{21} and v_{12} are the shear parameters. This implies that

$$x' = u_{11} + u_{12}x + u_{21}y \qquad \textbf{Eq. 5-7}$$

and

$$y' = v_{11} + v_{12}x + v_{21}y\,. \qquad \textbf{Eq. 5-8}$$

The initial values of u_{11}, v_{11}, u_{21} and v_{21} are zero, while u_{12} and v_{12} are set to one. Differentiating for x' and y' we obtain

$$dx' = du_{11} + du_{12}x + du_{21}y \qquad \textbf{Eq. 5-9}$$

and

$$dy' = dv_{11} + dv_{12}x + dv_{21}y\,. \qquad \textbf{Eq. 5-10}$$

By substituting equations Eq. 5-9 and Eq. 5-10 back into Eq. 5-3, the following equation is obtained:

$$f(x,y) - e(x,y) = \qquad \textbf{Eq. 5-11}$$
$$g^0(x,y) + g_x du_{11} + g_x x du_{12} + g_x y du_{21} + g_y dv_{11} + g_y x dv_{12} + g_y y dv_{21}$$

This equation is now in the format of a linear least squares observation equation, except that an approximating linear expansion of the model has been employed. Therefore, the solution must be calculated iteratively. The least squares model determines a new set of affine parameters. The picture template $g^0(x,y)$ is then re-sampled. The new picture template and the new affine parameters are then inserted back into Eq. 5-11, replacing the old parameters. The process is repeated until an acceptable error is achieved or the number of iterations exceeds a pre-specified maximum. The least squares solution to Eq. 5-11 is

$$\overrightarrow{X} = (A^T A)^{-1} A^T l\,. \qquad \textbf{Eq. 5-12}$$

where for each grey-value i

$$X_i^T = (du_{11} \quad du_{12} \quad du_{21} \quad dv_{11} \quad dv_{12} \quad dv_{21}), \qquad \textbf{Eq. 5-13}$$

$$l_i = f(x,y) - g^0(x,y) \qquad \textbf{Eq. 5-14}$$

and

$$A_i = (g_{xi} \quad g_{xi}x \quad g_{xi}y \quad g_{yi} \quad g_{yi}x \quad g_{yi}y)\,. \qquad \textbf{Eq. 5-15}$$

Figure 5-3 contains a description of the above process in pseudo-code.

```
For each template patch f(x,y):
        Determine g^0(x,y), the approximate position of the picture patch.
        Calculate the difference between f(x,y) and g^0(x,y).
        Repeat until the difference is acceptably small or the maximum number of iterations
        has been exceeded:
                Calculate the gradient of g^0(x,y).
                Setup the parameters of Eq. 5-12, using Eq. 5-13, Eq. 5-14 and Eq. 5-15.
                Calculate the new affine parameters using Eq. 5-12.
                Resample g^0(x,y) (usually a bilinear interpolation is used).
                Calculate the difference between f(x,y) and g^0(x,y).
        end.
end.
```

Figure 5-3: High-level pseudo-code description of ALSM.

An important problem with ALSM is that it may converge very slowly, or even diverge, resulting in failed or incorrect matches. Baltsavias (1991) points out a number of reasons for this. One of these is that there are classes of sub-images with textures such that some or all of the affine parameters cannot be determined. This is the cause of non-convergence researched here; however, poor matching can also result from ambiguous patches (i.e. there are multiple areas on the picture which match the template patch) and occlusions, among other causes. In all these cases, either an inaccurate match is made or the algorithm fails to converge. In addition the matching process will execute a large number of iterations attempting to match the problematic patch, thereby wasting computing time.

A number of advantages would result if patches with nondeterminable affine parameters could be identified automatically by a pattern recognition system. The template patches could be examined in advance and classified according to which parameters are nondeterminable. When the matching process is executed the problematic affine parameters can be ignored when the affine transformation is being calculated. For instance, if a shear parameter is nondeterminable, it can be set to zero. Alternatively, if a scale parameter is nondeterminable it can be set to one (thereby keeping Eq. 5-6 meaningful for the shift parameters) (Baltsavias, 1991). If too many of the affine parameters cannot be determined (matching can take place with just the shift parameters), the faulty template patch can be excluded from the matching process, thereby saving computing time. Figure 5-4 depicts a possible method for incorporating a pattern recognition system into the MPGC process.

arguments apply to images 2, 3 and 4. This issue has been circumvented in the reformulation of the problem.

As mentioned in Chapter 3, no noise filtering pre-processing operation was implemented in this research, except for thresholding (which is very simple as far as noise reduction pre-processing operators go). The reasons for this are:

- The images used to construct the data set of real images for the nondeterminable affine parameter problem had been cleaned of noise and radiometric distortions in advance.

- Unlike most other pattern recognition problems, the noise in a sub-image might actually be useful to the matching process by providing just enough texture for a match to be applied. By applying a noise filter to the patches[37], the author suspected that essential information for deducing affine parameter determinability might be lost. However, with hindsight, it might have been better to implement more sophisticated noise filters and to determine their usefulness empirically.

---

[37] Noise filters had been applied to the entire images. By applying a noise filter at the patch level, even more information might be removed from the images.

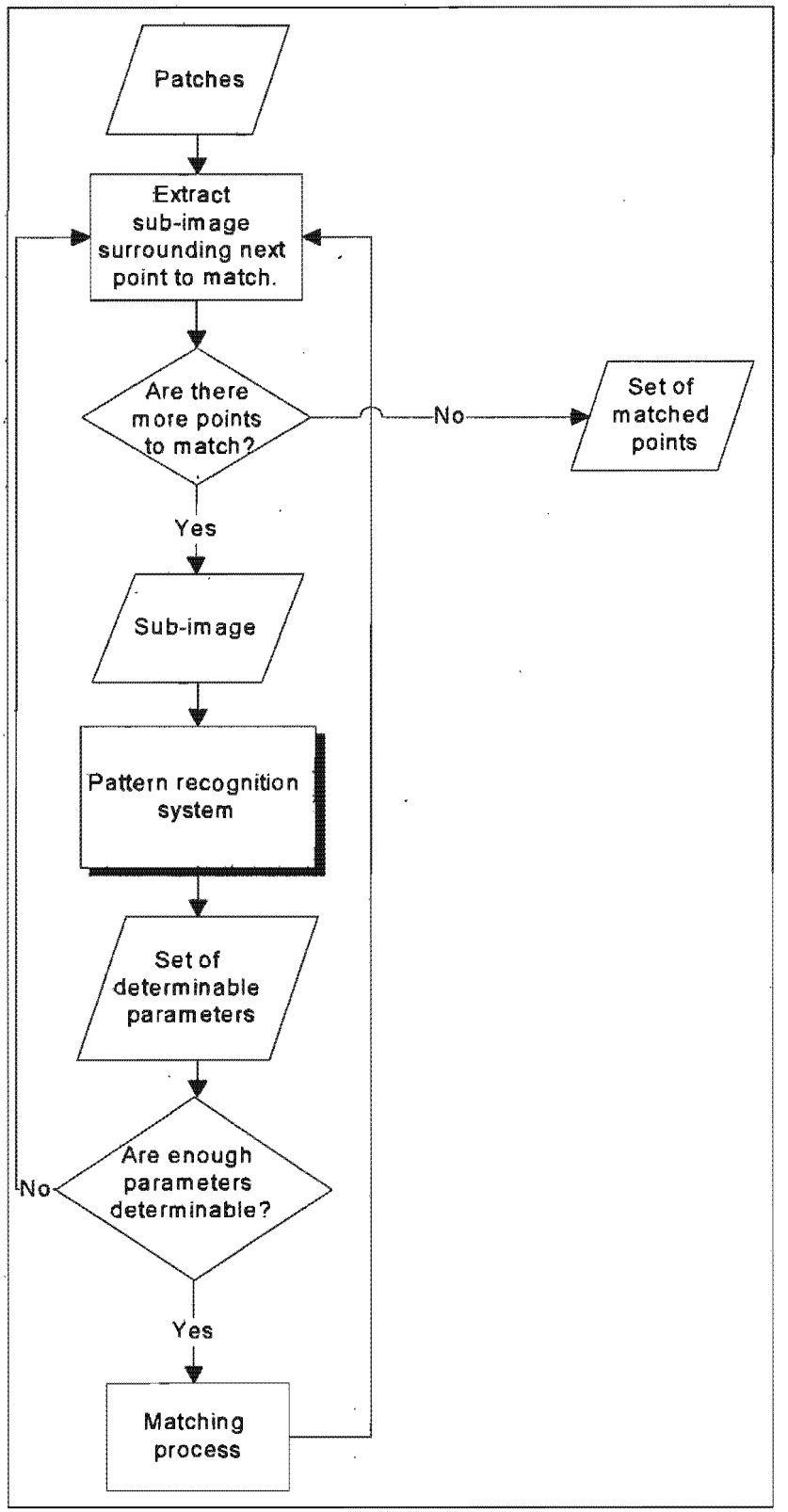


Figure 5-4: Hypothetical image matching system.
A pattern recognition system (the process shown with a shadow) has been incorporated into the MPGC process.

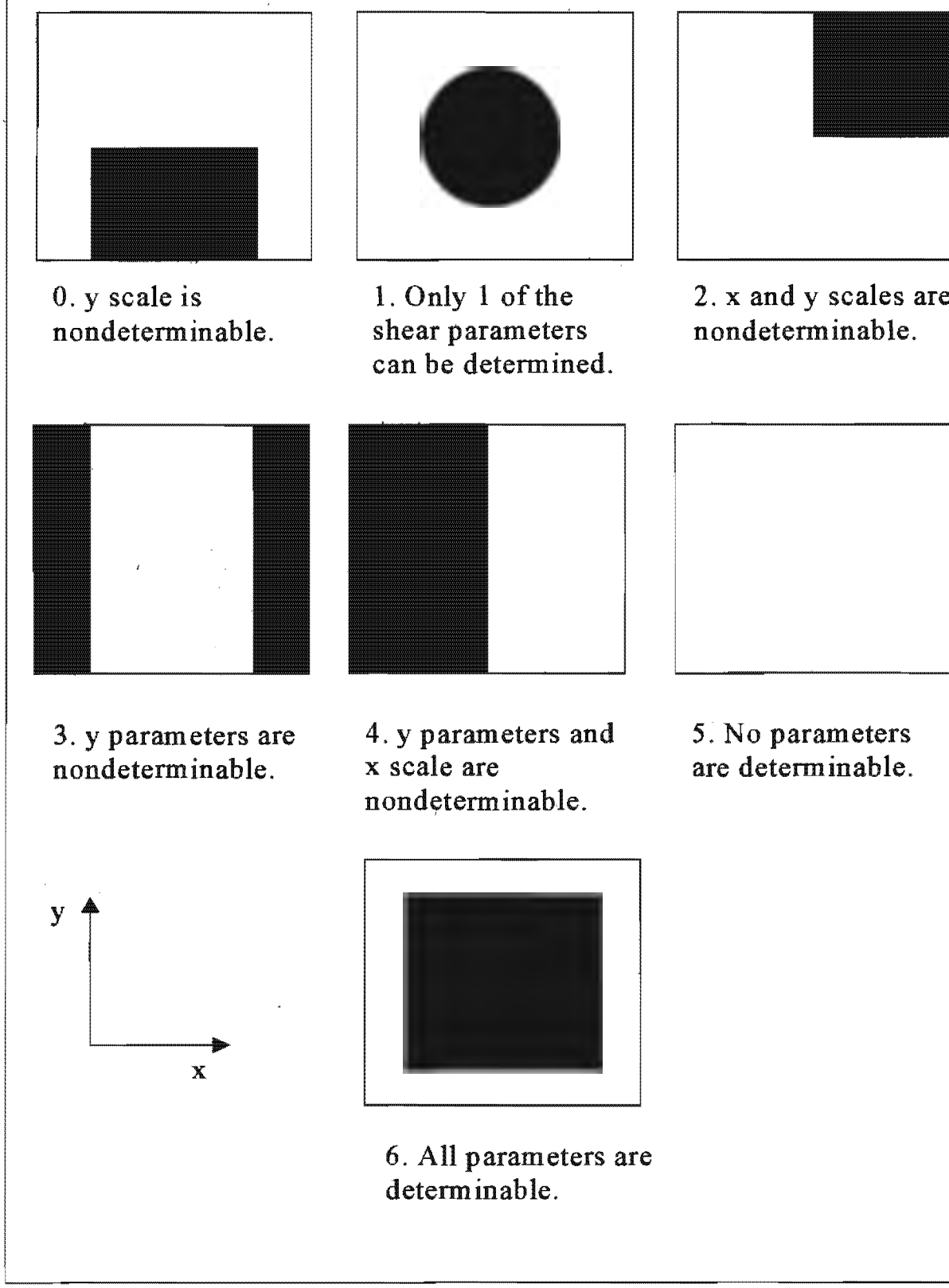


Figure 5-5: Idealised cases of nondeterminable affine parameters. (taken from Baltsavias, 1991).

5.3. Solutions Suggested by Baltsavias

Baltsavias (1991) suggests a number of techniques to identify patches with nondeterminable affine parameters. These are mentioned very briefly.

- Using a statistical test procedure called data snooping, the shaping parameters are checked at each iteration to see if they lie in a known range. If they do not they are excluded from subsequent iterations. Knowledge of the range of the shaping parameters is required for this method to work successfully.

- Using the $(A^TA)^{-1}$ matrix in Eq. 5-12, which is derived from the greyscale values of the template patch, an ellipse can be fitted over the patch. An algorithm has been developed that determines which parameters should be used based on the properties of the ellipse.

- Tests can be conducted on the correlation between the parameters. For instance, if two parameters are highly correlated they can be excluded. (One parameter, at least, should be turned off, since they are not simultaneously determinable.) This test is only conducted after a match has taken place and is used to determine the success of a match.

- After each iteration, the eigenvalue of each affine parameter can be calculated, using the (A^TA) matrix in Eq. 5-12. Baltsavias has formulated a number of criteria for including and excluding affine parameters based on the eigenvalues.

- Baltsavias has also developed a number of techniques which modify the patch size, according to the amount of signal in the area content in the patch area or according to the properties of the object surface. By enlarging the patch more texture is usually introduced, thereby increasing the chance of having determinable parameters. However, larger patches result in increased processing time. In addition large fluctuations in the depth across a terrain can lead to reduced matching accuracy.

These methods have not been considered in this research. The primary objective of this thesis is to examine pattern recognition techniques. In the loosest definition, the above methods can be considered pattern recognition techniques, but they are domain specific and unrelated to the class of domain-independent classification models considered in this research.

5.4. Related Work in Pattern Recognition

A search of the literature has not uncovered any attempts to solve the nondeterminable affine parameter problem using traditional pattern recognition techniques. However, there has been much research involving the recognition of patterns in greyscale images, particularly using ANNs, which is essentially the problem being addressed by this research.

LeCun *et al.* (1989a, 1989b, 1990a) have conducted a number of experiments with a neural network chip they have developed for recognising hand-written digits. Their work is among the most cited with regard to image processing tasks in ANN literature. Greyscale 20x20 pixel images are passed directly to an ANN with 400 input units. The network has four hidden layers and one output layer with ten units, one for each digit. The size of the output values is used as a confidence measure. The difference between the two highest values computed by the output units represents a measure of ambiguity. Small differences imply an ambiguous digit. If all output values are low an unclear digit is implied. Particularly interesting is the processing done by the hidden layers. Each layer performs an important feature extraction task suggesting that successful neural networks often need to have *a priori* task specific information built into them.

Kepuska and Mason (1995) used 35x35 pixel greyscale patches as input to the same ANN model as that used with LVQ in this thesis, except that it was used to generate a Self Organising Feature Map (SOFM) (Kohonen, 1982a), which was used as input to an ANN without hidden layers. They cleaned the images using Wallis Filtering, Median Filtering, Histogram Transformation and Low-Pass Filtering. A success rate of 74% on unseen patches was achieved. Their work is relevant to this work for a number of reasons: (1) true greyscale patches were used, as opposed to binary images, (2) their application is also in photogrammetry and (3) the ANN model they employed for the SOFM is used in this thesis with the LVQ algorithm.

The k-NN algorithm is also often used in image recognition tasks. It is also often used as a benchmark with which to compare ANN models (e.g. Weidemann *et al.*, 1995 and Holmstrom *et al.*, 1997).

The data sets used to conduct the experiments in this thesis are now described.

5.5. Artificial Reformulation of the Problem[38]

The artificial problem requires that a pattern recognition system be developed to differentiate between the different images in Figure 5-5. Moreover, the images must be recognised if they have been rotated, or scaled, or had noise added to them, or if their greyscale values have been changed, or any combination of these transformations have been applied to them.

The artificial data set was constructed based on the first six idealised images of Figure 5-5 [39]. The data set was created using the following procedure:

- Firstly, the six idealised images were created using 9x9 greyscale patches. Two greyscale values, 0 and 255, were used to construct the background and foreground of the images, respectively.

- Variations on the scale of the images created using the previous step were manually constructed by the author. The data set was then augmented with these images. The purpose of this step was to introduce enough variation to teach the pattern recognition systems scale invariance.

- Various distortions of the images created using the above steps, such as modifying the symmetry of the images, were manually constructed. The data set was then augmented with these images. The purpose of this was

---

[38] The raw (i.e. without any pre-processing applied to it) artificial data set can be downloaded from ftp://foxbat.sur.uct.ac.za/pub/data/ngeffen.

[39] Image 7 was ignored, since its affine parameters are fully determinable and it represents an extremely small part of the pattern space with fully determinable affine parameters (i.e. it is too idealised).

to teach the pattern recognition systems invariance to shift and other minor distortions. Figure 5-6 shows examples of these distortions.

- A utility, written by the author, was used to reproduce the images created using the above steps, but using different greyscale values. The images were reproduced a number of times using different greyscale values on each occasion. The purpose of this was to introduce enough information to teach the pattern recognition systems amplitude invariance.

- Since the images created using the above steps have the same properties when rotated at certain angles, a utility written by the author was used to augment the data set with the above images rotated by 90, 180 and 270 degrees. The purpose of this is to introduce enough information to teach the pattern recognition systems rotational invariance.

- In order to increase the complexity of the problem, thereby posing a more difficult pattern recognition problem, the data set was augmented with the above images with Gaussian noise added to them. A standard deviation of three greyscale units was used.

Each image patch was classified into one of six classes, each class corresponding to each of the six images with nondeterminable affine parameters in Figure 5-5. Some implications of this are that the Euclidean and Mahalanobis minimum distance classifiers require the construction of six mean vectors during their training stages and the implemented MLPs require six output neurons, one per class. Also, an ANN trained with LVQ requires at least six codevectors (but in practice a lot more). Table 5-1 lists the classes and the number of images per class in the artificial data set.

Class	Number of Images	Density
0 *(Image 0 in* Figure 5-5*)*	480	0.22
1 *(Image 1 in* Figure 5-5*)*	288	0.13
2 *(Image 2 in* Figure 5-5*)*	528	0.24
3 *(Image 3 in* Figure 5-5*)*	527	0.24
4 *(Image 4 in* Figure 5-5*)*	336	0.15
5 *(Image 5 in* Figure 5-5*)*	48	0.02
TOTAL	**2207**	**1**

Table 5-1: Number of images per class in the artificial data set.

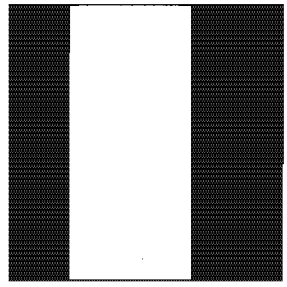


Image 3 of Figure 5-5 with the left and right dark bars set to different widths.

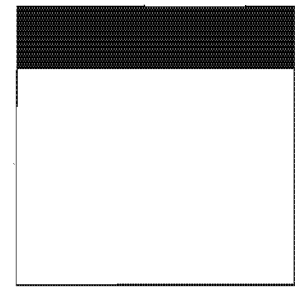


Image 4 of Figure 5-5 rotated with the width of the dark bar decreased.

Image 2 of Figure 5-5 rotated, scaled and the colours of the foreground and background switched.

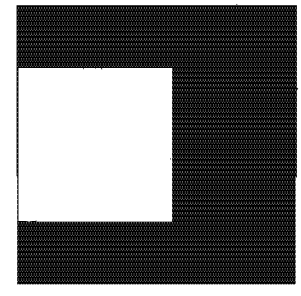


Image 0 of Figure 5-5 rotated, scaled and the colours of the foreground and background switched.

Figure 5-6: Examples of distorted idealised images with nondeterminable affine parameters.

There are limitations to the artificial data set:

- It contains images with only nondeterminable affine parameters. This was done so as to simplify the problem, and also because there is no idealised good image. The set of good patches is the complement of the set of patches which do not have fully determinable affine parameters. Image seven of Figure 5-5 demonstrates only one of the many possible types of good images. No easy method was found of collecting artificial good images akin to the method for collecting poor images. In addition, an ANN that has been trained to only recognise images with nondeterminable affine parameters can be considered to have recognised a good image, if all its output neurons give a low response. Though, it must be pointed out, this is not a guaranteed characteristic of the ANNs used in this research (Sarle, 1997)[40].

- Classifiers developed for the artificial data do not solve the nondeterminable affine parameter problem for real data. Therefore, the data set is of academic interest only.

5.6. Real Image Problem

A number of data sets, comprising real sub-images, were generated with the aid of an ALSM with geometrical constraints program, MatchProg[41], written by Julian Smit as part of his PhD thesis (Smit, 1997). However, only the one described here was deemed suitable for the formally conducted experiments. Image matching was performed on photographs of a number of objects, a propeller (PROP), a rhinoceros footprint (RHINO), a prehistoric hominoid footprint (FOOT), a cheetah paw (CAT)

---

[40]Parzen's classification method (Parzen, 1962 and Cacoullos, 1966), which has been recast in the form Probabilistic Neural Networks (Specht, 1990), does have the property of giving low output responses to unfamiliar patterns (unlike most ANN models). However, the classification time of this technique is probably impractical for this research.

[41] MatchProg is actually a number of programs, since there are multiple versions of Smit's ALSM program, a number of which were specifically written in order to produce the test data for this research.

and two rock images from mines (ROCK 1 and ROCK 2). The templates photograph of the propeller image is shown in Figure 5-7.

MatchProg designates one of the images as the template and divides it into overlapping patches. It then attempts to find the corresponding picture patch for each template patch. Once the matching process is complete for a patch, the program uses various techniques discussed in Baltsavias (1991) to check the quality of the match. In addition, the affine parameters of MatchProg can be turned off. Therefore, by running the program multiple times with the same input, but with different variations of the on/off state of the affine parameters, the results of each run can be compared. If a patch is matched successfully with an affine parameter switched off, but not when that parameter is switched on, then that affine parameter is considered nondeterminable for that patch.

Since there are six parameters, there are 64 combinations of the affine parameter states with which MatchProg can be run[42]. Six data images have been used, therefore the program would have to be run 378 times to generate the final data set[43]. Since each program execution for a particular image takes a few hours, this would be completely impractical. In addition, the process of tabulating which affine parameters were successful in matching a particular patch would be very complicated. Therefore, a compromise was necessary between producing a data set and the information output by the pattern recognition systems.

A successful match is impossible if the shift parameters are switched off. By varying the states of the remaining four parameters, the number of program execution combinations is reduced to 96 (i.e. $2^4 \times 6$), which is still impractical. In communication with Smit, the author of MatchProg, he confirmed that a more limited pattern recognition system, which recognises only whether the shear or scaling parameters are determinable (but not which particular shear or scaling parameter),

---

[42]Two states and six parameters implies that there are 2^6 affine parameter states.

[43]Since there are 378 states and six images, the program must be run ($378 \times 6 - 6$) times. Clearly, it is not necessary to run the program with all the parameters switched off, which is why six is subtracted (one for each data image).

would still be useful, especially since, in many cases the x and y components of the shear and scaling parameters are often either both determinable or both nondeterminable, particularly where aerial photography has been used.

Based on the above discussion, MatchProg was executed for each data image using the following four affine parameter states:

- All the affine parameters switched on
- Only the shear (rotation) parameters switched off
- Only the scaling parameters switched off
- Both the scaling and shear parameters switched off.

Using this method, only 24 executions of the program were needed[44]. Patches which Matchprog excluded for reasons unrelated to affine determinability were excluded from the data set. New patches were generated by flipping all the generated patches by 90, 180 and 270 degrees and concatenated to the data set, in order to teach the classifiers rotational invariance. The algorithm presented in Figure 5-8 was used to classify each image patch into one of five classes. Some implications of this are that the Euclidean and Mahalanobis minimum distance classifiers require the construction of five mean vectors during their training stages, and the implemented MLPs require five output neurons, one per class. Also, an ANN trained with LVQ requires at least five codevectors (but in practice a lot more). Table 5-2 describes the composition of the data set. Table 5-3 describes the class densities of the data set.

---

[44]With this method there were effectively only four variations of the affine parameters. Since each variation had to be run for each of the six images, MatchProg had to be executed 24 times.

Figure 5-7: Template photograph of a propeller used to construct the real data set.

If the patch was matched with all the affine parameters on
    assign to **class 0** *(category for good patches)*
else if the patch was matched with only the shear parameters switched off
    assign to **class 1** *(category for patches with shear nondeterminable)*
else if the patch was matched with only the scale parameters switched off
    assign to **class 2** *(category for patches with scale nondeterminable)*
else if the patch was matched with both the scale and shear parameters switched off
    assign to **class 3** *(category for patches with only shift determinable)*
else (the patch could not be matched at all)
    assign to **class 4** *(category for patches with no parameters determinable).*

Figure 5-8: Algorithm for assigning patterns to classes in the real data set. Geometric constraints were used for the generation of the classes.

Image Name	Class 0 (good patches)	Class 1 (Shears not determinable)	Class 2 (Scales not determinable)	Class 3 (Only shifts determinable)	Class 4 (No determinable parameters)	Total
PROP	3260	780	664	388	124	***5216***
RHINO	3596	900	280	200	24	***5000***
FOOT	2672	1092	652	380	404	***5200***
CAT	2976	668	320	196	184	***4344***
ROCK 1	1264	448	120	72	60	***1964***
ROCK 2	7372	3472	800	488	932	***13064***
Total	***21140***	***7360***	***2836***	***1724***	***1728***	***34788***

Table 5-2: Contributions of each image to the real data set.
The table contains the number of images obtained from each photograph for each of the classes.

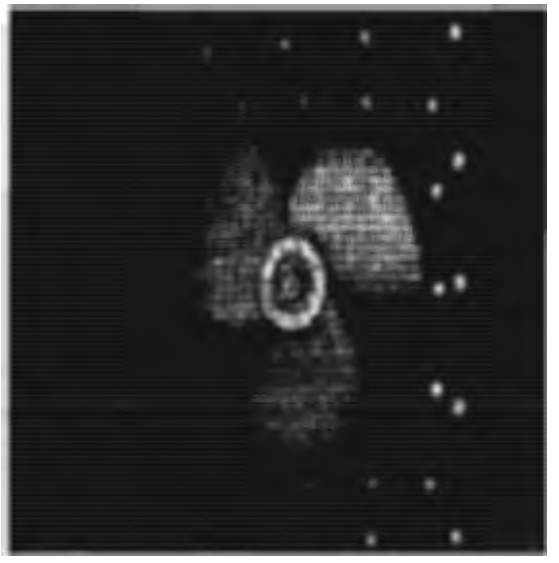


Figure 5-7: Template photograph of a propeller used to construct the real data set.

If the patch was matched with all the affine parameters on
assign to **class 0** *(category for good patches)*
else if the patch was matched with only the shear parameters switched off
assign to **class 1** *(category for patches with shear nondeterminable)*
else if the patch was matched with only the scale parameters switched off
assign to **class 2** *(category for patches with scale nondeterminable)*
else if the patch was matched with both the scale and shear parameters switched off
assign to **class 3** *(category for patches with only shift determinable)*
else (the patch could not be matched at all)
assign to **class 4** *(category for patches with no parameters determinable).*

Figure 5-8: Algorithm for assigning patterns to classes in the real data set. Geometric constraints were used for the generation of the classes.

Image Name	Class 0 (good patches)	Class 1 (Shears not determinable)	Class 2 (Scales not determinable)	Class 3 (Only shifts determinable)	Class 4 (No determinable parameters)	Total
PROP	3260	780	664	388	124	***5216***
RHINO	3596	900	280	200	24	***5000***
FOOT	2672	1092	652	380	404	***5200***
CAT	2976	668	320	196	184	***4344***
ROCK 1	1264	448	120	72	60	***1964***
ROCK 2	7372	3472	800	488	932	***13064***
Total	***21140***	***7360***	***2836***	***1724***	***1728***	***34788***

Table 5-2: Contributions of each image to the real data set.
The table contains the number of images obtained from each photograph for each of the classes.

Image	Class 0	Class 1	Class 2	Class 3	Class 4
PROP	0.63	0.15	0.13	0.07	0.02
RHINO	0.72	0.18	0.06	0.04	0.00
FOOT	0.51	0.21	0.13	0.07	0.08
CAT	0.69	0.15	0.07	0.05	0.04
ROCK 1	0.64	0.23	0.06	0.04	0.03
ROCK 2	0.56	0.27	0.06	0.04	0.07
Total	***0.61***	***0.21***	***0.08***	***0.05***	***0.05***

Table 5-3: Class densities (per image) of the real data set.

5.7. Minimum Requirements of the Systems

Given M classes a pattern recognition system that assigns patterns to a class at random would have an average accuracy A_r, such that

$$A_r = 1/M \,. \qquad \textbf{Eq. 5-16}$$

Moreover, if the class densities are known, a system that assigns all the patterns to the class with the highest density, would have an accuracy A_d such that

$$A_d = D \,. \qquad \textbf{Eq. 5-17}$$

Note that A_d has to be bigger than or equal to A_r. Even if A_d is not known, but the class with the highest density of patterns is known, the second scheme will perform better than random classification.

A useful system should have a better accuracy than A_r and, if the class densities are known, it should also have a better accuracy than A_d, otherwise one of the above simple assignment schemes might just as well be used.

Since the class densities of the artificial data set have been artificially constructed and have no intended correlation with the class densities that occur in reality, it does not make sense to consider A_d for the artificial data set. Therefore, the minimum accuracy expected of a recognition system for the artificial data set for it to be considered successful must be substantially larger than $^1/_6$ since there are 6 categories in the data set.

A_d cannot be known with much certainty for the data set of real images either. Table 5-3 shows that the class densities differ significantly across the different images.

However, the density of the category of good images (Class 0) ranges from 0.51 to 0.72. Since the data set is large, it seems reasonable to take the average density of Class 0 across the entire data set and to set A_d to 0.61. Therefore, significantly more than 61% of images should be classified correctly by a successful pattern recognition system of the real data.

A minimum useful classification speed can also be defined for the real data set. The patterns in this set were all classified in the process of generating data for it by running MatchProg four times on the images that comprise it, once for each variation of the affine parameters. The same process could be used to classify the patches of any image. Therefore, a useful system should classify a pattern faster than the time it takes MatchProg to process that pattern four times. The processing time of MatchProg for a patch differed across images. In addition, the program was run many times under different system loads. An approximate average of 0.8 seconds processing time per patch per execution was calculated. Since each patch was processed four times, this comes to approximately 3.2 seconds per patch. As such, it is reasonable to demand that a useful pattern recognition system classify a patch in substantially less than 3.2 seconds.

6. Experiments and Results

6.1. Introduction

A number of pattern recognition systems were implemented comprising the pre-processing algorithms discussed in Chapter 3 and the classification models discussed in Chapter 4. These were tested on the data sets described in Chapter 5. This chapter examines the results of these experiments and discusses their implications. The results of all the formally conducted experiments can be found in Appendix A.

The following technical points concerning the experiments should be noted:

- Both the artificial data set and the real sub-image data set were randomly divided into training and validation sets, comprising 80% and 20% of the patterns, respectively.

- The number of possible pattern recognition systems that can be constructed using the algorithms developed in Chapter 3 and Chapter 4 is very large, since a system can consist of any number of pre-processing operations in any order, followed by a pattern classification algorithm. The number of systems implemented was limited by knowledge of the algorithms and experimenting with various combinations of the pre-processing operations (and variations of their parameters) in conjunction with the minimum distance classifiers, which are the quickest classifiers to train and execute. The more sophisticated classification algorithms (ANNs and k-NN) were then combined with those pre-processing operations which produced reasonable results with the minimum distance classifiers. Regarding the MLPs, an initial estimate was made of the number of hidden neurons to use. Then additional experiments were conducted, adding one neuron per experiment. About 15 to 20 variations on the number of hidden neurons were experimented for each ANN on the artificial data set. Fewer ANN systems were implemented for the real data set, since each experiment took a long time to execute and it became clear that results would improve fractionally, at best, by varying the number of hidden neurons (or codevectors in the case of LVQ).

- For the MLPs, patterns were assigned to the class of the output neuron with the highest response, irrespective of the size of this response, or the size of the difference between the highest response and the second-highest response. This is called the *Winner Takes All* method (terminology used in Mache, 1997).

- Some of the pre-processing operators (such as thresholding) transformed the training data, rendering two or more patterns equal in the process. An operator was therefore developed to remove duplicate patterns from the training data sets. Sometimes results improved marginally (but almost always less than a percentage point) by removing the duplicates. This operator was never applied to the validation set, so as not to introduce bias.

- Owing to the dependency of an ANN's results on the random initialisation of its weights, each ANN implementation was run three times. Of the three runs, the network with the highest accuracy (or lowest mean squared error) in respect of the *training data* was selected for validation.

In addition, experimental results with the feature selector using the genetic algorithm described in Section 3.9 were unsatisfactory in so far as the implemented algorithm executed so slowly that it had to be discarded. A preliminary, highly optimised version of the algorithm was implemented and executed on an experimental data set approximately 40% smaller in size than the real data set. The Maximum Gradient edge detection operator was first applied to the images in the data set resulting in an input data set to the genetic algorithm consisting of patterns with 49 features (i.e. a 9x9 pixel image transformed into a 7x7 gradient image). This implies that one out of 2^{49} possible feature sets had to be selected by the algorithm. The population size was set to 30 and the program was executed for 10 generations. These are unusually low numbers for these parameters, but the aim of this preliminary test was to determine if the algorithm could be executed successfully in a reasonable amount of time. The program failed to terminate within 48 hours. It was decided that once the population size and number of generations were set to higher values, the program would take an impracticably long time to complete. Besides using an optimising C compiler (GNU C under Unix with optimisation set to the highest level), the following optimisations

were implemented: (1) all floating point numbers in the data set were converted to integers, and (2) the City block distance measurement (see Table 4-2), was used instead of the Euclidean distance measurement, thereby eliminating the floating point operations associated with the Euclidean distance measurement.

Given that the algorithm is somewhat speculative (research into genetic algorithms is relatively new and there are very few papers that deal with using a genetic algorithm in the context of feature selection), it was decided that further attempts to improve it would not be cost-effective. It is conceivable that this feature selection algorithm could be of practical use if the nearest neighbour part of the algorithm were to be further optimised and the size of the training data set were to be significantly reduced. Techniques for doing this are discussed in detail in Dasarathy (1991) (also see Section 4.3.1).

6.2. Artificial Data Set: Results and Discussion

6.2.1. Artificial Data Set: Accuracy

Classification algorithms (discussed in Chapter 4).	**Pre-processing algorithms (discussed in Chapter 3).**	**Overall accuracy (%).**
k-NN (k = 1) [4.3].	Sobel edge detection [3.3], set the maximum value to 48 [3.5], threshold (ρ=4) [3.4], standardise [3.8].	94
ANN trained with LVQ (400 codevectors) [4.6].	Sobel edge detection [3.3], set the maximum value to 48 [3.5], threshold (ρ=4) [3.4], standardise [3.8]. z-axis normalisation [3.7].	91.5
ANNs trained with Error Back-propagation (24, 25 or 26 hidden neurons in one hidden layer or 4 layered ANN with 24 neurons in first hidden layer and 12 neurons in second hidden layer give similar results. The input layers of the ANNs contain 49 and the output layers contain 6 neurons [4.4].	Sobel edge detection [3.3], set the maximum value to 48 [3.5], threshold (ρ=4) [3.4], standardise [3.8].	88 – 90
ANN trained with Resilient-propagation (23 hidden neurons in 1 hidden layer) [4.5].	Sobel edge detection [3.3], set the maximum value to 48 [3.5], threshold (ρ=4) [3.4], standardise [3.8].	87.5
Mahalanobis minimum distance [4.2.2].	Reduce all pattern features by feature with minimum value [3.2].	77
Mahalanobis minimum distance with clustering (5 clusters per class) [4.2.3].	Reduce all pattern features by feature with minimum value [3.2].	64
Euclidean minimum distance [4.2.1].	Sobel edge detection [3.3], set the maximum value to 48 [3.5], threshold (ρ=4) [3.4], standardise [3.8].	40

Table 6-1: Accuracy of pattern recognition systems for artificial validation data set. The table shows the most accurate system developed for each of the seven classification models discussed in Chapter 4. Each pattern recognition system consists of the pre-processing operators in the second column combined with the classification model in the first column. The pre-processing operators for each system are listed in order of application. The third column is the percentage of patterns in the validation set correctly classified. Next to each algorithm is a cross-reference in brackets to the section where it is described. Results are rounded to 0.5%.

Pre-processing Operators			**Classification Algorithm**			
Sobel edge detection [3.3], set the maximum value to 48 [3.5], threshold (ρ=4) [3.4], standardise [3.8].			k-NN, with k = 1			
Overall Accuracy	**Class 0**	**Class 1**	**Class 2**	**Class 3**	**Class 4**	**Class 5**
94%	99%	91%	98%	100%	82%	100%

Table 6-2: Complete results of the most accurate system for the artificial validation data set.

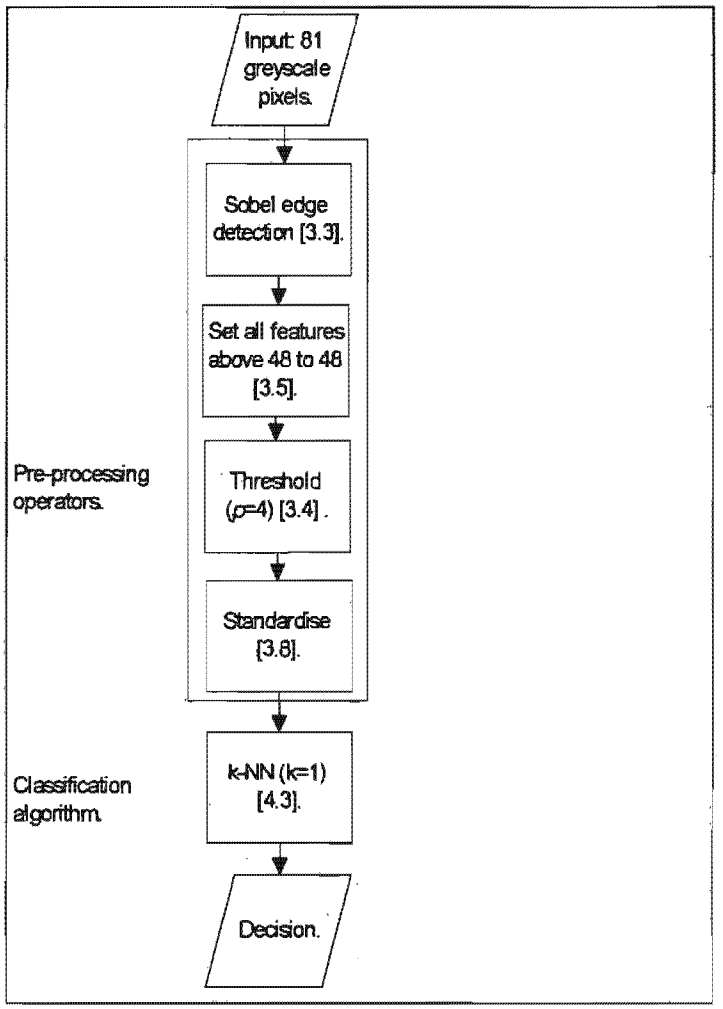


Figure 6-1: Diagram of the most accurate system for the artificial validation data set. The numbers in brackets are cross-references to the sections where the algorithms are described.

Table 6-1 lists the most successful set of pre-processing operators and parameter values for each of the classification models discussed in Chapter 4. The first two columns of each entry in the table comprise a description of a complete pattern recognition system. The systems are listed in order of accuracy. The following are some of the most important results — extracted from Table 6-1 and Appendix A — of the experiments conducted on the artificial data set[45]:

[45] The results are in respect of this problem only and are not general statements about the quality of the classification models of Chapter 4 or the pre-processing algorithms of Chapter 3.

- All the pattern classification algorithms, if paired with a suitable set of pre-processing operators, exceed the minimum accuracy requirement of 0.167 (or $^1/_6$) for the artificial data set described in the Section 5.7.

- K-NN is the most accurate of the classification models, followed by LVQ, Error Back-propagation, Resilient-propagation, Mahalanobis minimum distance classifier, Mahalanobis minimum distance classifier with clustering and the Euclidean minimum distance classifier. These results are not surprising, since k-NN frequently performs slightly better than most ANN models (e.g. Weidemann *et al.*, 1995), albeit with a higher cost in system resources, such as execution time. The most accurate pattern recognition system is depicted in Figure 6-1. Its complete results, with respect to accuracy, are listed in Table 6-2.

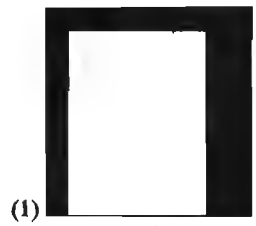
- For the k-NN algorithm, as k increases the accuracy decreases (albeit slowly). This is unexpected and probably implies that the classes are quite intertwined in the pattern space. This could be because the pre-processing operators have not separated the classes enough.

- Better results were achieved using the Sobel edge detector than the Maximum Gradient edge detector. This is not surprising, since the Sobel operator incorporates information regarding all the greyscale values into the gradient image, as opposed to the Maximum Gradient edge detector which ignores all but the strongest gradient in each 3x3 pixel area.

- Patterns in Class 5 presented the greatest difficulty to the pattern recognition systems. Those systems that managed to separate these patterns from the rest of the pattern space (such as the top four in Table 6-1), achieved the best results. Class 5 patterns are difficult to recognise for two reasons (1) there are large fluctuations in the greyscale amplitudes of these patterns and (2) they are often confused with images which have been scaled down in other classes. The first problem can be overcome with the edge detection algorithms, since they remove the amplitude variations. Failure due to the second reason could only be overcome by

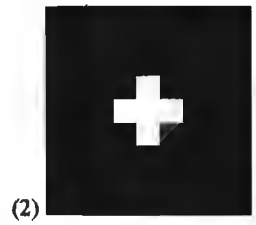
the more powerful class separation abilities of the ANN and k-NN models.

- The number of hidden neurons in MLPs with one hidden layer was a critical factor with respect to accuracy. Results ranged from as low as 76% (12 hidden neurons) to almost 90% percent (24 to 26 hidden neurons). A similar situation applied to LVQ, with accuracy ranging from approximately 73% (69 codevectors) to over 90% (more than 300 codevectors). An ANN with 2 hidden layers was implemented with good results as well (approximately 89%).



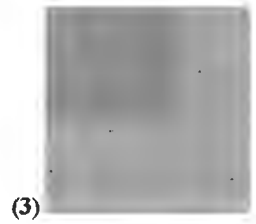
(1) Class 0.

```
6   2   8   2   0   0   0   1   0
1   251 255 255 255 255 255 1   0
2   255 255 255 255 253 255 0   0
0   252 255 255 255 255 255 0   0
0   254 253 252 253 255 255 0   0
1   255 249 254 255 255 255 1   0
0   255 253 255 254 255 255 0   1
5   253 250 255 255 255 251 1   0
0   255 255 255 255 250 255 3   5
```



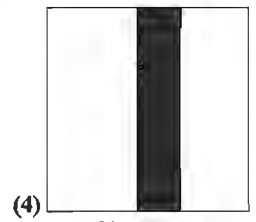
(2) Class 1.

```
0   0   0   0   6   0   4   0   0
0   0   0   0   1   0   0   0   0
2   0   0   0   0   1   1   0   2
0   0   0   6   255 2   0   0   0
4   0   5   252 255 255 0   0   1
0   0   0   3   255 1   3   0   2
2   0   1   0   0   0   3   3   6
0   0   0   2   0   1   0   0   0
0   0   1   0   0   1   2   2   0
```



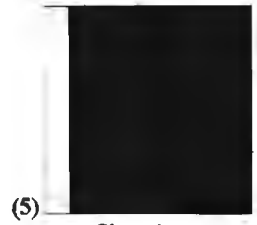
(3) Class 2.

```
112 112 112 112 112 112 138 138 138
112 112 112 112 112 112 138 138 138
112 112 112 112 112 112 138 138 138
112 112 112 112 112 112 138 138 138
112 112 112 112 112 112 138 138 138
138 138 138 138 138 138 138 138 138
138 138 138 138 138 138 138 138 138
138 138 138 138 138 138 138 138 138
138 138 138 138 138 138 138 138 138
```



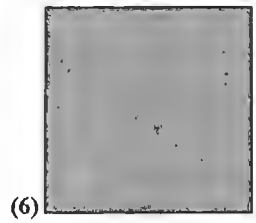
(4) Class 3.

```
254 255 252 250 1   5   252 255 255
254 251 255 255 0   3   255 254 253
255 255 255 255 0   0   250 254 252
255 255 250 255 2   1   253 255 252
251 253 252 255 2   0   253 255 255
255 255 255 255 4   3   254 248 249
252 247 255 254 3   0   254 252 253
255 255 252 253 0   0   255 255 253
255 255 252 255 0   0   251 255 255
```



(5) Class 4.

```
255 2   0   6   2   0   0   1   0
255 0   0   1   0   0   2   2   0
252 4   0   0   0   5   0   3   0
255 0   1   1   0   6   0   0   0
255 0   0   2   0   2   0   1   0
254 2   0   0   3   0   1   2   4
255 0   0   1   1   0   1   2   0
255 0   2   0   0   0   1   0   0
249 0   2   0   2   2   4   0   4
```



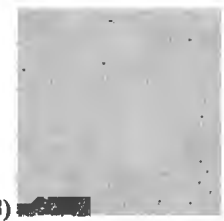
(6) Class 5.

```
140 133 135 130 133 129 138 133 131
138 135 132 131 129 137 132 132 131
133 129 131 133 133 132 134 129 132
133 132 131 131 129 132 131 138 131
133 138 138 133 131 133 134 131 137
135 133 133 135 133 133 135 137 133
134 142 130 133 136 137 131 137 132
135 140 129 129 133 130 134 133 133
133 133 133 128 130 133 133 133 130
```

Figure 6-2: Examples of correctly classified patches in the artificial validation data set. The patches cannot be printed accurately. Therefore, the greyscale bitmap values of the images are shown alongside the patches.

(7)
Class 0 incorrectly classified as Class 5

```
180 185 180 180 180 180 180 180 180
180 185 180 180 180 180 180 180 180
180 185 180 180 180 180 180 180 180
180 185 180 180 180 180 180 180 180
180 180 180 180 180 180 180 180 180
180 180 180 180 180 180 180 180 180
180 180 180 180 180 180 180 180 180
180 180 180 180 180 180 180 180 180
180 180 180 180 180 180 180 180 180
```



(8)
Class 1 incorrectly classified as Class 3.

```
179 175 187 184 186 180 181 181 187
183 181 181 185 188 186 180 182 176
181 181 187 186 185 182 190 184 181
185 184 187 187 187 190 187 188 183
187 185 187 189 186 192 185 185 187
177 186 187 187 186 189 189 181 180
184 181 185 186 188 192 186 181 179
181 183 179 191 178 182 182 182 181
179 179 176 180 185 181 182 189 177
```

(9)
Class 2 incorrectly classified as Class 3.

```
187 186 191 189 185 189 190 191 189
190 192 189 189 190 175 182 183 182
185 189 191 190 189 178 180 189 179
191 194 190 187 185 176 182 179 178
185 190 186 187 190 179 180 183 175
194 190 190 191 185 180 181 182 179
195 193 190 191 189 182 187 183 177
189 191 189 189 194 177 186 181 178
189 189 191 187 190 177 179 177 179
```

(10)
Class 4 incorrectly classified as Class 2.

```
43  46  40  36  41  34  39  37  32
42  42  46  41  43  41  39  35  37
43  40  39  42  42  37  35  36  40
46  42  41  41  40  35  41  36  38
42  44  42  45  42  34  33  35  38
47  41  37  41  43  32  36  37  40
42  45  41  42  42  39  36  39  41
41  40  45  41  41  37  40  39  37
42  42  36  46  44  37  39  35  38
```

Figure 6-3: Examples of incorrectly classified patches.
The patches cannot be printed accurately. Therefore, the greyscale bitmap values of the images are shown alongside the patches.

Figure 6-2 contains an example from each class of a correctly classified image, while Figure 6-3 contains examples of incorrectly classified patterns by the most accurate pattern recognition system. The cases in Figure 6-3 sit on the borders between classes but they have been generated by the data collection process described in Section 5.5. Explanations of why failure occurred with these patches are:

- *Patch 7, Figure 6-3.*

  The system fails to notice the thin foreground strip in the top left corner (the strip is signalled by the greyscale values of 185) because the difference in foreground and background amplitudes is too small.

- *Patch 8, Figure 6-3.*

  As with Patch 7, the difference in foreground and background is too small. In addition, the noise has distorted the original sub-image by changing the circular texture in the middle (the critical characteristic of Class 1 patches) to texture resembling a bar through the sub-image (the critical characteristic of Class 3 patches).

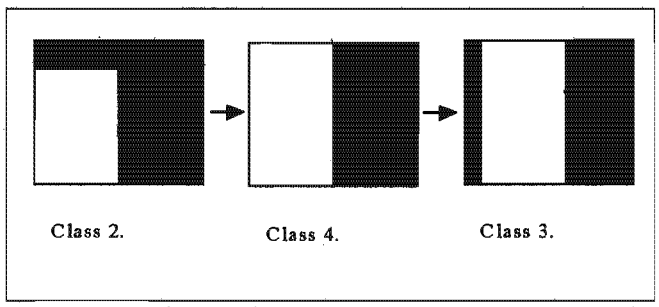


Figure 6-4: Depiction of why Patch 9 is incorrectly classified.

- *Patch 9, Figure 6-3.*

  Figure 6-4 is an aid to understanding this error. This pattern sits on the border between Class 2 and Class 3. The corner-placed rectangle which is found in all Class 2 images, has been subtended one line short of the image bisecting bar (common to patches in Class 4). In addition, noise in the leftmost line has made the foreground look like the background,

thereby making the patch look similar to patches in Class 3. The foreground and background in the topmost and leftmost lines have been swamped by the noise in these lines, thus confusing the system.

- *Patch 10, Figure 6-3.*

  By examining the greyscale values of this patch it can be seen that it is composed of a light and dark bar, thus making it a Class 4 patch. However, the difference in foreground and background of the sub-image is very small. This difference is made smaller by the noise in the patch, thereby confusing the system.

The examples of incorrectly classified patterns indicate that the pattern recognition systems find patches that have small differences in the greyscale values of the foreground and background texture hardest to classify. The Gaussian noise further complicates these images by occasionally reducing the differences in foreground and background even more.

Based on these results, the most accurate systems have learnt rotational and scale invariance. They have also learnt amplitude invariance, except in borderline cases where the difference between the foreground and background amplitudes are very small.

6.2.2. Artificial Data Set: Classification Speed

Classification algorithms (discussed in Chapter 4).	**Approximate speed to classify validation set (444 patterns).**
Euclidean minimum distance	< 1 second
Mahalanobis minimum distance	< 10 seconds
MLPs (speed independent of training algorithm)	< 10 seconds
ANN trained with LVQ	< 10 seconds
Mahalanobis minimum distance with clustering	< 30 seconds
k-NN	< 30 seconds

Table 6-3: Execution times of classification models for artificial validation data set. The times are approximate, and based on classification speeds when the system load was low. Pre-processing operations seldom add more than a few seconds to the execution times. As such, the entire execution process of a system seldom exceeds 35 seconds. The number of neurons in the MLP does not significantly effect the speed. The table entries are listed in order of execution speed. The speed refers to elapsed time, not CPU time.

Table 6-3 lists approximate execution times of the classification models on the validation set (pre-processing time was, generally, negligible, i.e. less than five seconds). The implication of this is that the execution time to process one pattern for the slowest system is less than 0.1 seconds. Note that k-NN is, as would be expected, the slowest. The LVQ ANN and the MLPs are only slightly less accurate than the k-NN algorithm, but classify much faster. This is consistent with other comparisons between ANNs and k-NN algorithms (e.g. Weidemann et. al., 1995 and Holmstrom et. al., 1997).

6.2.3. Artificial Data Set: Training Speed

Classification algorithms (discussed in Chapter 4).	**Approximate training speed.**
k-NN	0 seconds
Euclidean minimum distance	< 2 seconds
Mahalanobis minimum distance	< 10 seconds
Mahalanobis minimum distance with clustering	< 30 seconds
ANN trained with LVQ	< 40 seconds
MLP trained with Error Back-propagation	< 2 minutes 30 seconds for 150 iterations
MLP trained with Resilient-propagation	< 2 minutes 30 seconds for 150 iterations

Table 6-4: Training times of classification models for the artificial validation data set. Pre-processing time seldom adds more than a few seconds onto the overall training time. The number of neurons in the MLPs does not significantly effect training speed. The speed refers to elapsed time, not CPU time.

Table 6-4 lists the training times for the classification algorithms. No training occurs for the standard k-NN algorithm. The most successful MLPs improve their training set accuracy negligibly after 150 training iterations (i.e. the mean squared error decreases insignificantly).

6.3. Real Data Set: Results and Discussion

6.3.1. Real Data Set: Accuracy

Classification algorithms (discussed in Chapter 4).	Pre-processing algorithms (discussed in Chapter 3).	Overall accuracy (%).
k-Nearest Neighbour (k = 1) [4.3].	Sobel edge detector [3.3], threshold (ρ=28) [3.4].	61
ANN trained with LVQ (90 codevectors) [4.6].	Sobel edge detector [3.3], threshold (ρ=28) [3.4].	61
ANNs trained with Error Back-propagation (15 hidden neurons) [4.4].	Sobel edge detector [3.3], threshold (ρ=28) [3.4].	61
ANN trained with Resilient-propagation (23 hidden neurons in 1 hidden layer) [4.5].	Reduce all features in pattern by feature with minimum value [3.2].	60
Mahalanobis minimum distance with clustering (5 clusters per class) [4.2.2].	Sobel edge detector [3.3], threshold (ρ=28) [3.4].	47
Mahalanobis minimum distance [4.2.2].	Sobel edge detector [3.3], threshold (ρ=28) [3.4], standardise [3.8].	46.5
Euclidean minimum distance [4.2.1].	Reduce all features in pattern by feature with minimum value [3.2].	19

Table 6-5: Accuracy of pattern recognition systems for real sub-image validation data set. The table shows the most accurate system developed for each of the seven classification models discussed in Chapter 4. Each pattern recognition system consists of the pre-processing operators in the second column combined with the classification model in the first column. The pre-processing operators for each system are listed in order of application. The third column is the percentage of patterns in the validation set correctly classified. Next to each algorithm is a cross-reference in brackets to the section where it is described. Results are rounded to 0.5%.

Table 6-5 lists the most successful set of pre-processing operators and parameter values for each of the classification models discussed in Chapter 4. The first two columns of each entry in the table comprise a description of a complete pattern recognition system. The systems are listed in order of accuracy. The following are some of the most important results — extracted from Table 6-5 and Appendix A — of the experiments conducted on the data set containing real sub-images:

- None of the pattern recognition systems achieved better accuracy than the 61% minimum requirement for success calculated in Section 5.7. In fact the top three systems listed in Table 6-5 learnt the rule discussed in Section 5.7 to calculate A_d (which is 61%), precisely. They assigned all patterns to the class with the highest density, thereby achieving 100% accuracy on the images with fully determinable parameters and 0% on the other four classes. Obviously these pattern recognition systems have failed to separate the classes in the pattern space. The reasons for this probably lie with the data collection method employed and not with the

pattern recognition systems themselves. The training set seems to be inadequate for training the systems to recognise images with nondeterminable parameters. The following problems could have contributed to the creation of an inadequate training data set:

- Affine determinability depends only upon the actual patch texture itself. However, the MatchProg program might have classified some patches based on their position in the image (and the surrounding texture) from which they come from, not only on the determinability of their affine parameters. An example of how this can happen is when the initial approximation of the position of the picture patch is poor. The patch will then be, possibly incorrectly, classified by MatchProg as having no determinable parameters (Class 4).

- The pattern space is extremely large (there are 256^{81} possible patches). Despite the large size of the training data set (27 850 patches) it is probable that it is still not representative of the pattern space. The pre-processing operators transform the original pattern space to a lower dimension but it is probable that the transformations eliminate critical signal for determining the classes of the patches.

  Most of the patterns in the pattern space are probably unlikely to occur in practice and are therefore irrelevant to the nondeterminable affine parameter problem. Determining the subset of patterns in the pattern space that is relevant to the problem is extremely difficult, but worth investigating.

- MatchProg executes a user specified maximum number of iterations of the MPGC algorithm. If the difference between the template patch and the best matching picture patch is not below a user specified minimum, the patch is considered unmatched. It is possible that for a particular state of the affine parameters in one of the executions of MatchProg, some patches are matched on an

iteration very close to the maximum. For another execution, the same patches might just fail to match, but would match if a few more iterations were allowed. These patches would then be pre-classified in the wrong class.

- MatchProg has a number of blunder detection heuristics built in, which detect if a patch has been incorrectly matched. However, the heuristics are not fail-proof. Many incorrect matches are not detected. Such patches are quite likely to be incorrectly pre-classified.

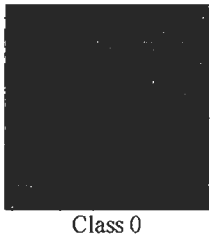
If the training data set has a large number of incorrectly classified patches in it, no pattern recognition system is likely to successfully partition the pattern space. However, it is also conceivable that the pre-processing algorithms are inadequate for extracting (or accentuating) the appropriate features necessary for separating the classes in the pattern space.

- The systems that achieved an accuracy of A_d used the Sobel edge detector pre-processing operator coupled with threshholding. However, systems that simply reduced the features in a pattern vector by the feature with the minimum value performed almost as well.

Pre-processing Operators			**Classification Algorithm**		
Sobel edge detector, followed by thresholding (ρ= *28 to 30*) (but no standardisation, unlike Table 6-5).			Mahalanobis classifier		
Overall Accuracy	**Class 0**	**Class 1**	**Class 2**	**Class 3**	**Class 4**
41%	61%	0%	30%	7%	22%

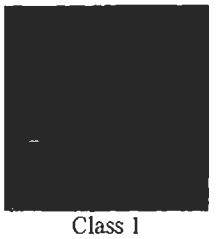
Table 6-6: Potentially promising results of one pattern recognition system.

- One of the pattern recognition systems (and a few others based on it), described in Table 6-6, has produced results that are potentially promising. Although its accuracy overall is only 41%, it is one of the only classifiers to produce a score for a class that is significantly higher than that class's density (i.e. for Class 2 and Class 4). The reasons for this are unclear. It occurs with the Mahalanobis classifier perhaps, because the density of the classes has less bearing on decisions than a classification model such as k-NN.



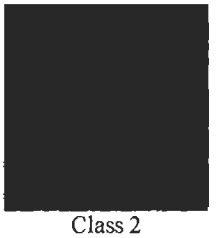
Class 0

98 97 97 95 95 97 96 98 102
95 98 99 100 104 105 105 107 104
97 101 103 105 111 113 114 113 109
101 101 102 104 109 111 112 107 106
104 100 98 98 100 101 102 97 93
103 97 94 92 91 91 90 85 80
100 96 92 87 85 84 82 77 75
100 97 89 83 82 81 80 77 79
100 93 87 82 81 79 81 78 80



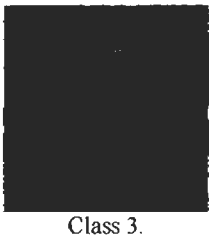
Class 1

97 98 96 94 89 81 75 72 68
100 101 102 99 93 83 77 72 67
101 103 103 100 95 85 76 71 66
103 104 102 98 91 83 77 69 65
105 105 101 96 85 77 72 70 67
104 104 99 92 82 76 71 71 69
101 98 96 89 82 77 72 70 69
103 98 93 89 83 80 76 74 73
103 99 95 93 89 84 84 80 78



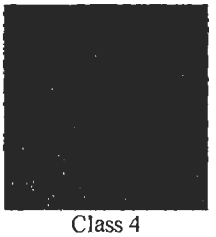
Class 2

74 75 77 77 76 75 72 73 74
72 74 76 77 75 73 72 72 73
72 75 76 78 77 75 73 74 77
76 77 79 82 80 80 80 78 81
78 81 85 86 87 86 87 86 87
81 85 89 90 93 93 92 90 91
83 86 91 95 97 97 94 90 89
87 90 93 98 99 99 95 91 85
98 98 98 98 99 98 95 90 83



Class 3.

99 96 96 96 96 97 96 93 89
100 97 96 97 97 100 98 97 91
98 98 96 96 98 100 98 97 89
97 96 94 93 94 95 93 93 85
94 92 90 86 85 87 88 88 85
90 87 84 81 78 83 85 84 85
88 85 82 80 79 81 84 84 87
86 85 82 81 83 83 86 87 87
85 86 84 83 85 85 87 87 84



Class 4

96 95 93 89 88 90 94 95 96
99 97 94 91 91 94 97 99 97
102 98 100 95 95 97 99 99 96
107 102 104 100 99 99 100 98 96
109 108 107 106 104 102 104 100 98
112 112 112 113 113 110 109 103 101
115 117 117 117 116 114 112 109 109
116 120 119 121 118 113 110 113 110
115 122 126 125 121 118 116 114 113

Figure 6-5: Examples of some of the images from the real sub-image validation data set. The patches cannot be printed accurately. Therefore the greyscale bitmap values of the images are shown alongside the patches. However, unlike the artificial images, it is hard to see differences between the images at all, even in a bitmap editor. The classes of these images were determined by the algorithm in Figure 5-8.

- By examining Figure 6-5, one of the central difficulties with this problem can be seen. Unlike the artificial sub-images the real ones, generally, do not have significant texture. There is very little difference between the foreground and background of these patches. Even a human observer cannot differentiate between patterns in different classes, even if the sub-

images are viewed in a bitmap editor. Clearly not enough textural changes occur in the average 9x9 pixel patch. Better results would probably be achieved with larger patches because these would, on average, contain more textural variation.

6.3.2. Real Data Set: Classification Speed

Classification algorithms (discussed in Chapter 4).	**Approximate speed to classify validation set (6938 patterns).**
Euclidean minimum distance	< 15 seconds
MLPs (speed independent of training algorithm)	< 30 seconds
ANN trained with LVQ	< 30 seconds
Mahalanobis minimum distance	< 30 seconds
Mahalanobis minimum distance with clustering	< 30 seconds
k-NN	150 – 170 minutes

Table 6-7: Execution times of classification models for real sub-image validation data set. Pre-processing time seldom exceeds 30 seconds. As such, its effect on whether or not the classification systems meet the minimum time requirements outlined in Section 5.7 is negligible. K-NN is extremely slow because of the large training set.

All the systems meet the minimum time requirements specified in Section 5.7. K-NN is by far the most expensive (see Table 6-7), but using the most conservative estimate of the performance of the systems in which it is used, one pattern is processed in 1.5 seconds, which is less than half the minimum requirement of 3.2 seconds. All the other pattern recognition systems processed one pattern in negligible time (<< 0.1 seconds). This is an important result because it implies that future attempts to solve the nondeterminable affine parameter problem can spend a lot more processing time in the pre-processing phase in order to accentuate the critical features of this problem.

6.3.3. Real Data Set: Training Speed

Classification algorithms (discussed in Chapter 4).	Approximate training speed.
k-NN	0 seconds
Euclidean minimum distance	< 5 seconds
Mahalanobis minimum distance	< 3 minutes
ANN trained with LVQ	< 10 minutes
MLP trained with Error Back-propagation	< 2 hours
MLP trained with Resilient-propagation	< 2 hours
Mahalanobis minimum distance with clustering	< 4 hours

Table 6-8: Training times of classification models for real sub-image data set. Pre-processing time seldom adds more than a few seconds onto the overall training time. The table is sorted from least training time to most training time.

The approximate training times of the classification models are listed in Table 6-8. No training occurs for the standard k-NN algorithm. ANNs trained with LVQ are much faster than the MLPs, since the weight update rules and neuron firing functions for LVQ are linear. One surprising result is the slow training time of the Mahalanobis minimum distance classifier with clustering. (compare Table 6-8 to Table 6-4). The reason for its slow training time is that clustering the data of the real data set requires more iterations of the k-Means algorithm than for the artificial data set. In addition each iteration takes longer to execute, because of the large size of the real sub-image training set.

7. Conclusions

7.1. Introduction

This thesis has examined the process of developing a pattern recognition system. The main components in this process, data collection, pre-processing of the data and pattern classification models have been discussed. Pattern recognition systems for two applications derived from the nondeterminable affine parameter problem have been implemented and tested. The primary results of these experiments are that successful pattern recognition systems have been developed to solve a problem involving artificial patterns. However, poor results have been achieved using patterns derived from digital photographs. It has been argued that this has primarily been a result of (1) the problems associated with the data collection process, (2) the large dimensionality of the pattern space, (3) the limitations of the implemented pre-processing operators and (4) the small patch sizes. As such, one of the aims of this research, to implement a pattern recognition system that can assist in image matching applications in digital photogrammetry, has not been accomplished.

The main contributions of this thesis are:

- A thorough analysis has been performed on the issues involved in developing a decision-theoretic pattern recognition system to solve the nondeterminable affine parameter problem. Should there be further attempts in the future to solve this problem using decision-theoretic methods, this thesis could be a useful source of information.

- The artificial sub-image problem has been defined. The data set containing the artificial sub-images developed for this problem has been made available on the Internet so that other researchers can use it to benchmark pattern recognition systems. In addition, this thesis has shown that pattern recognition systems can be developed which separate the classes in this data set with a high degree of accuracy.

The remainder of this chapter examines how the systems developed in this research can be improved. Suggestions for future research are also made.

7.2. Improving the Pattern Recognition Systems

By examining Figure 2-1 (p. 10) and the results of the experiments discussed in the previous chapter, it is clear that the most problematic components, regarding the development of a successful pattern recognition system to solve the nondeterminable affine parameter problem regarding real-world patches, are the processes of data collection and pre-processing (including feature extraction and selection). For both the artificial problem and the real sub-image problem, the variation in accuracy between systems is more a function of these aspects of the pattern recognition process than the pattern classification models. The ANNs and the k-NN algorithm (all of which outperformed the minimum distance classifiers) achieve similar results (within a few percentage points) when they process the same input data, but accuracy differs widely when different pre-processing operations are used. Therefore, future research on this problem should concentrate on these problematic aspects of the pattern recognition process as opposed to the selection of classification models. Much research effort in pattern recognition theory is put into tweaking classification models so as to improve accuracy by a few percentage points (e.g. Wettschereck and Dietterich, 1992 benchmark a number of ANNs with a variation of 10 percentage points in performance). This is useful, but in practical problems it is often the case that more effort has to go into the process of collecting the data and determining what pre-processing has to be applied to it. Understanding the problem domain and how training samples can be extracted from it is often the most critical part of the process of developing a pattern recognition system.

A number of approaches could be taken in order to attempt to develop sufficiently accurate recognition systems for the nondeterminable affine parameter problem:

- Baltsavias's methods, discussed in Section 5.3, should be investigated further. These might be more successful than the general decision-theoretic methods researched in this thesis, since these are intended to be specific solutions to the nondeterminable affine parameter problem.

- Methods to improve the collection of the training set samples must be examined. Possibilities include: (1) using a more sophisticated program than MatchProg, such as ETH's PVD program (but it will need to be

modified) to collect the training samples, (2) manually examining the sub-images collected for affine determinability (although this will probably result in bias creeping into the data set) and (3) using Baltsavias's methods to assist in the data collection process. This research would be very time consuming and difficult. It should be asked whether the potential benefit to the MPGC algorithm is worth the effort.

- More sophisticated pre-processing operators must be tested. Noise reduction methods must be examined. More sophisticated edge detection techniques, such as the Canny operator, should also be investigated. A more thorough analysis of the images than has been done in this research should be performed. It would also be worthwhile to invest more time into tweaking the parameters of some of the pre-processing operators used in this thesis, such as thresholding. Developing pre-processing operators that are invariant to rotation and scaling should also be considered, as opposed to incorporating invariance via the training data.

- Using larger patch sizes (such as 31x31 pixels) will probably also achieve better results. As pointed out in the previous chapter, there is simply not enough texture on the 9x9 pixel patches in the real sub-image data set. However, using larger patches would bring extra complexity to the problem by increasing the dimensionality of the patterns. Methods to reduce the number of features would have to be investigated. In addition, larger patches have repercussions for MPGC because the accuracy of this algorithm is often compromised when larger patch sizes are used.

- Investigating simpler versions of the nondeterminable affine parameter problem, such as trying to solve the problem for a particular domain (e.g. images of houses only, or images of footprints only) might produce better results. By doing this, the size of the relevant part of the pattern space would be substantially reduced, and it might be easier to collect a more representative training data set, than the one collected in this thesis.

There is also scope for research into developing a decision-theoretic pattern recognition system which identify **precisely** which affine parameters are

nondeterminable, unlike the systems in this thesis which attempt only to identify pairs of nondeterminable parameters.

7.3. Research Suggestions

The success of the pattern recognition systems with respect to recognising the artificial patterns is a positive result of this research. The complexity of the patterns in the artificial data set should not be underestimated, since they test the ability of systems to learn amplitude, scale, rotational and noise invariance. As such, this data set is useful for benchmarking pattern recognition systems and it has been made available for public access via the Internet[46]. The patterns could be made more challenging by adding more Gaussian noise to them, as well as other types of noise.

The accuracy of the systems for the artificial problem diminishes when there is little difference in the background and foreground greyscale amplitudes. It would be worth investigating ways of overcoming this, thereby increasing the accuracy of the systems to almost 100%. It seems reasonable that the key to achieving this would be via more sophisticated pre-processing operators that provide greater contrast between the foreground and background of the images.

Potentially interesting areas of research with respect to both the artificial and real sub-image problem include: (1) implementing invariance to rotation, scale and amplitude via pre-processing operations as opposed to using the training data set, (2) developing statistically sound confidence measurements of the accuracy of the systems and (3) implementing systems that are adaptable to newly acquired training data (unlike those used in this work).

---

[46] The data set can be obtained via anonymous ftp at ftp://foxbat.sur.uct.za.za/pub/data/art.txt.gz. The author can be contacted for advice regarding the use of this file via email at ngeffen@yahoo.com. Various, freely available, utilities have been developed for manipulating files in this format by the author, which might be of use to pattern recognition system developers.

Appendix A.

This appendix contains two tables listing the overall accuracy of the formally tested pattern recognition systems on the artificial validation data set and the real sub-image validation data set. Only the experiments conducted with the most successful pre-processing parameter values are listed. Duplicate removal in the training set has also not been indicated, since the effect of this on accuracy is generally less than a percentage point.

Pre-processing Operators *(Discussed in Chapter 3)*	**Classification Algorithm** *(Discussed in Chapter 4)*	**Overall Accuracy (%).**
Sobel edge detection [3.3], set the maximum value to 48 [3.5], threshold (ρ=4) [3.4].	k-NN (k=1) (k=3)	94 91
Sobel edge detection [3.3], set the maximum value to 48 [3.5], threshold (ρ=4) [3.4], standardise [3.8]. z-axis normalisation [3.7].	ANN trained with LVQ. The number of codevectors used ranged from 69 to 400. Peak occurred at 400. Worst performance occurred with 69.	91.5 (peak). 71 to 91.5 (range).
Sobel edge detection [3.3], set the maximum value to 48 [3.5], threshold (ρ=4) [3.4], standardise [3.8].	k-NN (k=1) (k=3)	90 88
Sobel edge detection [3.3], set the maximum value to 48 [3.5], threshold (ρ=4) [3.4], standardise [3.8].	ANN trained with Error Back-propagation. ANNs with 12 to 30 hidden neurons were tested. The peak occurred with 26 neurons. The worst performance occurred with 12 neurons.	90 (peak) 76.1 to 90 (range)
Sobel edge detection [3.3], set the maximum value to 48 [3.5], threshold (ρ=4) [3.4], standardise [3.8].	ANN trained with Error Back-propagation: two hidden layers were used, 24 neurons in the first, 12 in the second.	89
Sobel edge detection [3.3], set the maximum value to 48 [3.5], threshold (ρ=4) [3.4], standardise [3.8].	ANN trained with Resilient-propagation. ANNs with 12 to 24 hidden neurons were tested. The peak occurred with 24 neurons. The worst performance occurred with 15 neurons.	87.5 (peak). 78.6 to 87.5 (range).

Table continued on next page ...

Table continued from previous page ...

Sobel edge detection [3.3], set the maximum value to 48 [3.5], threshold (ρ=4) [3.4], standardise [3.8].	ANN trained with Scaled Conjugate Gradient Method[47]. ANNs with 12 to 24 hidden neurons were tested. The peak occurred with 24 neurons. The worst performance occurred with 12 neurons.	87 (peak). 74.3 to 87 (range).
Reduce all pattern features by feature with minimum value [3.2].	k-NN (k=1) (k=3) (k=5) (k=7).	83 75 71 70
Reduce all pattern features by feature with minimum value [3.2].	Mahalanobis minimum distance.	77.5
Sobel edge detection [3.3], set the maximum value to 48 [3.5], threshold (ρ=4) [3.4].	Mahalanobis minimum distance.	75
(none)	Mahalanobis minimum distance.	71
Maximum Gradient edge detection [3.3], set the maximum value to 12 [3.5], threshold (ρ=4) [3.4].	Mahalanobis minimum distance.	70
Sobel edge detector [3.3].	Mahalanobis minimum distance.	65
Maximum Gradient edge detection [3.3], set the maximum value to 12 [3.5].	Mahalanobis minimum distance.	64
Reduce all pattern features by feature with minimum value [3.2].	Mahalanobis minimum distance with clustering (5 clusters per class).	64
Maximum Gradient edge detection [3.3].	Mahalanobis minimum distance.	62
Maximum Gradient edge detection [3.3], threshold (ρ=10) [3.4].	Mahalanobis minimum distance.	56
(none)	Euclidean minimum distance.	16

Table A-1: Accuracy of systems for artificial validation data set.
The table is sorted from most accurate to least accurate. Next to the pre-processing operators are cross-references to their descriptions.

[47] This MLP learning algorithm has not been discussed in this dissertation, but the author has recently tested it out of interest.

Pre-processing Operators *(Discussed in Chapter 3)*	**Classification Algorithm** *(Discussed in Chapter 4)*	**Overall Accuracy (%).**
Sobel edge detector [3.3], threshold (ρ=28) [3.4].	MLP trained with Error Back-propagation with 15 hidden neurons: 49 (input) – 15 (hidden) – 5 (output).	61
Sobel edge detector [3.3], threshold (ρ=28) [3.4].	k-NN (k=1).	61
Sobel edge detector [3.3], threshold (ρ=28) [3.4], z-axis normalisation [3.7].	MLP trained with LVQ (28, 58, 90 or 100 codevectors).	61
Reduce all features in pattern by feature with minimum value [3.2].	MLP trained with Resilient-propagation with 24 hidden neurons: 81 (input) – 24 (hidden) – 5 (output).	60
Reduce all features in pattern by feature with minimum value [3.2].	k-NN (k=1) (k=3) (k=5) (k=7).	53 51 52 55
Maximum Gradient edge detection [3.3], set the maximum value to 12 [3.5], threshold (ρ=4) [3.4].	k-NN (k=1) (k=3).	49 47
Sobel edge detection [3.3], set the maximum value to 48 [3.5], threshold (ρ=4) [3.4].	k-NN (k=1) (k=3).	49 47
Sobel edge detector [3.3], threshold (ρ=28) [3.4].	Mahalanobis minimum distance with clustering (5 clusters per class) [4.2.2].	47
Sobel edge detector [3.3], threshold (ρ=28) [3.4], standardise [3.8].	Mahalanobis minimum distance.	46.4
Sobel edge detector [3.3], thresholding (ρ=28) [3.4].	Mahalanobis minimum distance.	41.6
Reduce all features in pattern by feature with minimum value [3.2].	k-NN (k=1) (k=3).	40 29
Sobel edge detection [3.3], set the maximum value to 48 [3.5], threshold (ρ=4) [3.4].	Mahalanobis minimum distance with clustering (5 clusters per class).	34
Reduce all features in pattern by feature with minimum value [3.2].	Euclidean minimum distance.	19
Reduce all features in pattern by feature with minimum value [3.2].	Mahalanobis minimum distance.	19

Table A-2: Accuracy of systems for real sub-image validation data set. The table is sorted from most accurate to least accurate. Next to the pre-processing operators are cross-references to their descriptions.

References

Aleksander, I., 1992. *Capturing consciousness in neural systems.* Neural Networks, **2**, pp. 17-22.

Aleksander I. and Morton, H., 1990. *An Introduction to Neural Computing.* Chapman & Hall, London.

Baase, S., 1988. *Computer Algorithms: Introduction to Design and Analysis.* Addison-Wesley, Massachusetts.

Baltsavias, E., 1991. *Multiphoto Geometrically Constrained Matching.* Dissertation 49, Institute of Geodesy and Photogrammetry, ETH, Zurich.

Beasley, D., Bull, D. R. and Martin, R. R., 1993a. *An Overview of Genetic Algorithms: Part 1: Fundamentals.* University Computing, **15**, pp. 58-69.

Beasley, D., Bull, D. R. and Martin, R. R., 1993b. *An Overview of Genetic Algorithms: Part 2: Research Topics.* University Computing, **15**, pp. 170-181.

Bow, S., 1984. *Pattern Recognition: Applications to Large Data-set Problems.* Marcel Dekker, Inc., New York.

Boyle, R.D. and Thomas, R.C., 1988. *Computer Vision: a First Course.* Blackwell Scientific Publications., Boston.

Brill, F. Z., Brown, D. E. and Martin, W. N., 1992. *Fast genetic selection of features for neural network classifiers.* IEEE Transactions on Neural Networks, **3**, pp. 324-328.

Broomhead, D. S. and Lowe, D., 1988. *Multivariable functional interpolation and adaptive networks.* Complex Systems. **2**, pp. 321-355.

Cacoullos, T., 1966. *Estimation of multivariate density.* Annals of the Institute of Statistical Mathematics. **18**, pp. 179-189.

Carpenter, G., Grossberg, S. and Reynolds, J. H., 1991. *ARTMAP: Supervised real-time learning and classification of nonstationary data by a self-organizing neural network.* Neural Networks, **4**, pp. 565-568.

Cover, T. M. and Hart, P. E. 1967. *Nearest neighbor pattern classification.* IEEE Transactions on Information Theory, **IT-13** (1), pp. 21-27.

Cybenko, G., 1989. *Approximation by superpositions of a sigmoidal function.* Mathematics of Control, Signals and Systems, **2**, pp. 303-314.

Dasarathy, B. V., 1991. *Nearest Neighbor(NN) Norms: NN pattern classification techniques.* IEEE Computer Society Press, California.

Dayhoff, J. E., 1990. *Neural Network Architectures: An Introduction.* Von Nostrand Reinhold, New York.

Devijver, P. A., 1980. *An overview of asymptotic properties of nearest neighbor rules.* Pattern Recognition in Practice, (E. S. Gelsema and L. N. Kanal – eds.), pp. 343-350, North-Holland, Amsterdam.

Devijver, P. A., 1982. *Advances in nonparametric techniques of statistical pattern classification.* Pattern Recognition Theory and Applications, (J. Kittler, K. S. Fu and L. F. Pau - eds.), pp. 3-18, D. Reidel Publishing Company, Dordrecht, Holland.

Draper, N. and Smith, H., 1981. *Applied Regression Analysis.* (2nd ed.), John Wiley & Sons, Inc., New York.

Duda, R. O., 1997. Online. *Pattern Recognition for HCI.* http://www.engr.sjsu.edu/~knapp/HCIRODPR/PR_home.htm.

Duda, R. O. and Hart, P. E., 1974. *Pattern Classification and Scene Analysis.* John Wiley & Sons, Inc., New York.

Ekstrom, M.P. (ed.), 1984. *Digital Image Processing Techniques, Volume 2.* Academic Press, Inc., Orlando.

Fahlman, S. E., 1988. *Faster-learning variations on Error Back-propagation: An empirical study*. Connectionist Models Summer School. (T.J. Sejnowski, G. E. Hinton and D. S. Touretzky – eds.), Morgan Kaufmann, San Mateo.

Fahlman, S. E. and Lebiere, C., 1990. *The Cascade-correlation learning architecture.* Advances in Neural Information Processing Systems 2. (D. S. Touretsky – ed.), pp. 524-532, Morgan Kaufmann, San Mateo.

Fu, K. S. (ed.), 1976. *Digital Pattern Recognition.* Springer-Verlag, Berlin.

Fukunaga, K. and Narendra, P. M., 1975. *A branch and bound algorithm for computing k-Nearest Neighbours.* IEEE Transactions on Computing, **24**, pp. 750-753.

Fukunaga, K., 1990. *Introduction to Statistical Pattern Recognition.* (2nd ed.), Academic Press, Inc., San Diego.

Goldberg, D. E., 1989. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, Massachusetts.

Gonzalez, R. C. and Thomason, M. G., 1978. *Syntactic Pattern Recognition: An Introduction.* Addison-Wesley, Massachusetts.

Gonzalez, R. C. and Wintz, P., 1987. *Digital Image Processing.* Addison-Wesley, Massachusetts.

Gruen, A., 1985. *Adaptive least squares correlation: A powerful image matching technique.* South African Journal of Photogrammetry, Remote Sensing and Cartography, **14**, pp. 175-187.

Hart, P. E., 1968. *The condensed nearest neighbor rule.* IEEE Transactions on Information Theory, **14**, pp 515-516.

Haykin, S., 1994. *Neural Networks: A Comprehensive Foundation*. Macmillan, New York.

Hassibi, B., Stork, D. G. and Wolff, G. J., 1993. *Optimal brain surgeon and general network pruning*. IEEE International Conference on Neural Networks, **1**, pp. 293-299, San Francisco.

Hebb, D.O., 1949. *The Organization of Behaviour: A Neuropsychological Theory*. Wiley, New York.

Heipke, C., 1996. *Overview of image matching techniques*. Proceedings of the OEEPE Workshop on Application of Digital Photogrammetric Workstations, (O. Koelbl – ed.), **33**, pp. 173-189. European Organization for Experimental Photogrammetric Research, Frankfurt.

Hinton, G. E., 1993. *How neural networks learn from experience*. Scientific American, **267**, pp. 104-109.

Holland, J. H., 1975. *Adaptation in Natural and Artificial Systems*. MIT Press, Massachusetts.

Holmstrom, L., Koistinen, P., Laaksonen, J. and Oja, E., 1997. *Neural and statistical classifiers - taxonomy and two case studies*. IEEE Transactions on Neural Networks, **8**, pp. 5-17.

Hopfield, J.J., 1982. *Neural networks and physical systems with emergent collective computational abilities*. Proceedings of the National Academy of Sciences of the U.S.A., **81**, pp. 3088-3092.

Hornick, K., Stithcombe, M. and White, H. 1989. *Multilayer feedforward networks are universal approximators*. Neural Networks, **2**, pp. 359-366.

Hornick, K., 1991. *Approximation capabilities of mutlilayer feedforward networks*. Neural Networks, **4**, pp. 251-257.

Hubel, D. H. and Wiesel, T.N., 1962. *Receptive fields, binocular interaction and functional architecture in the cat's visual cortex.* Journal of Physiology (London), **160**, pp. 106-154.

Hubel, D. H. and Wiesel, T. N., 1977. *Functional architecture of macaque visual cortex.* Proceedings of the Royal Society of London, Series B, **198**, pp. 1-59.

Hush, D. R. and Horne, B. G., 1993. *Progress in supervised neural networks: What's new since Lippmann?*. IEEE Signal Processing Magazine, **10**, pp. 8-39.

Kamgar-Parsi, B. and Kanal, L. N., 1985. *An improved branch and bound algorithm for computing k-Nearest Neighbours.* Pattern Recognition Letters, **3**, pp. 7-12.

Kepuska, V. Z., Mason, S. O., 1995. *A hierarchical neural network system for signalized point recognition in aerial photographs.* Photogrammetric Engineering & Remote Sensing, **61**, pp. 917-925.

Kia, S.J. and Coghill, C. G., 1992. *High performance clustering using dynamic competitive learning.* Electronic Letters, **28**, pp. 1753-1755.

Kittler, J., 1986. *Feature selection and extraction.* Handbook of Pattern Recognition and Image Processing, (T. Y. Young and K. Fu – eds.), pp. 59-83. Academic Press, San Diego.

Kohonen, T. , 1982a. *Self-organised formation of topologically correct feature maps.* Biological Cybernetics, **43**, pp. 59-69.

Kohonen, T. , 1982b. *Clustering, taxonomy and topological maps of patterns.* Proceedings of the 6th International Conference on Pattern Recognition. pp. 114-128, Munich.

Kohonen, T., 1986. *Learning vector quantization for pattern recognition.* Technical Report TKK-F-A601. Helsinki University of Technology, Helsinki.

Kohonen, T., Kangas, J., Laaksonen, J. and Torkkola, K., 1992. *LVQ-PAK: The learning vector quantization Program Package.* Helsinki University of Technology, Helsinki.

Koza, J., 1992. *Genetic Programming: On the programming of computers by means of natural selection.* MIT Press, Massachusetts.

LeCun, Y., 1985. *Une procedure d'apprentissage pour reseau a seuil assymetrique.* Cognitiva, **85**, pp. 599-604.

LeCun, Y., Boser, B., Denker, J., Henderson, D., Howard, R. E., Hubbard, W. and Jackel, L. D., 1989a. *Backpropagation applied to hand-written zip code recognition.* Neural Computation, **1**, pp. 541-555.

LeCun, Y., Jackel, L. D., Boser, B., Denker, J. S., Graf, H. P., Guyon, I., Henderson, D., Howard, R. E. and Hubbard, W., 1989b. *Hand-written digit recognition: Applications of neural network chips and automatic learning.* IEEE Communications Magazine, **27**, pp. 41-46.

LeCun, Y., Boser, B., Denker, J.S., Henderson, R.E., Howard, R.E., Hubbard, W. and Jackel, L.D., 1990a. *Hand-written digit recognition with a Error Back-propagation network.* Advances in Neural Information Processing Systems 2, (D. S. Touretsky - ed.), pp. 396-404, Morgan Kaufmann, San Mateo.

LeCun, Y., Denker, J. S. and Solla, S. A., 1990b. *Optimal brain damage.* Advances in Neural Information Processing Systems 2, (D. S. Touretsky - ed.), pp. 598-605, Morgan Kaufmann, San Mateo.

Lin, W. and Wang, S., 1996. *A new model for invariant pattern recognition.* Neural Networks, **9**, pp. 899-913.

Mache, N., 1997. On-line. *SNNS: Stuttgart Neural Network Simulator.* http://www.informatik.uni-stuttgart.de/ipvr/bv/projekte/snns/snns.html. University of Stuttgart: Dept. Applied Computer Science and Image Understanding. (Developed by

the Institute of Parallel and Distributed High-performance Systems at Stuttgart University.)

Massaro, D. W, and Stork, D. G., 1998. *Speech recognition and sensory integration.* American Scientist, **86**, pp. 236-244.

McCullogh, W. S. and Pitts, W., 1943. *A logical calculus of the ideas immanent in nervous activity.* Bulletin of Mathematical Biophysics, **5**, pp. 115-133.

Meisel, W. S., 1972. *Computer-Orientated Approaches to Pattern Recognition.* Academic Press, New York.

Minsky, M. L., 1961. *Steps towards artificial intelligence.* Proceedings of the Institute of Radio Engineers. **49**, pp. 8-30.

Minsky, M. L. and Papert, S., 1969. *Perceptrons.* MIT Press, Massachusetts.

Moller, M. F., 1993. *A scaled conjugate gradient algorithm for fast supervised learning.* Neural Networks, **6**, pp. 525-533.

Montgomery, B. L. and Vijaya Kumar, B. V. K., 1986a. *Nearest neighbor non-iterative error correcting optical associative memory processor.* Proceedings of the SPIE: Hybrid Image Processing, SPIE, Bellingham, Washington, **638**, pp. 83-90.

Montgomery, B. L. and Vijaya Kumar, B. V. K., 1986b. *Evaluation of the use of the Hopfield neural network model as a nearest neighbor algorithm.* Applied Optics, **25**, pp. 3759-3766.

Muller, B. and Reinhardt, J., 1990. *Neural Networks: An Introduction.* Springer-Verlag, Berlin.

Murtagh, F., 1996. On-line. *Two-dimensional Invariant Moments for Planar Shape Recognition.* http://www.infm.ulst.ac.uk/research/ac460/ip8a/node36.html.

Niemann, H. and Goppert, R. 1988. *An efficient branch-and-bound nearest neighbour classifier.* Pattern Recognition Letters, 7, pp. 63-72.

Orr, M. J. L., 1996. On-line. *Introduction to Radial Basis Function Networks.* http://www.cns.ed.ac.uk/people/mark/intro/intro.html.

Parker, D.B., 1987. *Learning-logic: Casting the cortex of the human brain in silicon.* Technical Report TR-47. Center for Computational Research in Economics and Management Science, MIT, Massachusetts.

Parzen, E., 1962. *On estimation of probability density function and mode.* Annals of Mathematical Statistics, **33,** pp. 1065-1076.

Peretto, P., 1992. *An Introduction to the modelling of neural networks.* Cambridge University Press, Cambridge.

Ramon y Cajal, S., 1911, *Histologie du systéme neveux de l'homme et des vertébrés.* Paris: Maloine; Edition Francaise Revue: Tome I, 1952; Tome II, 1955; Madrid; Consejo Superior de Investigaciones Cienteficas.

Rice, J. A., 1988. *Mathematical Statistics and Data Analysis.* Wadsworth, Inc., California.

Riedmiller, M. and Braun, H., 1993. *A direct adaptive algorithm for faster backpropagation learning: The RPROP algorithm.* Proceedings of the IEEE Conference of Neural Networks, 1993 (ICNN 93).

Rosenblatt, F., 1958. *The Perceptron: A probabilistic model for information storage and organization of the brain.* Psychological Review, **65**, pp. 386-408.

Rumelhart, D. E. and McClelland J.L., eds., 1986a. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1.* MIT Press, Massachusetts.

Rumelhart, D. E., Hinton, G. E. and Williams, R. J., 1986b. *Learning representation by back-propagating errors.* Nature, **323**, pp. 533-536.

Sarkar, D. 1995. *Methods to speed up error Error Back-propagation learning algorithm.* ACM Computing Surveys, **27**, pp. 519-541.

Sarle, W. S., 1994. *Neural networks and statistical models.* Proceedings of the Nineteenth Annual SAS Users Group International Conference.

Sarle, W. S., 1997. Online. *Neural Networks FAQ.* ftp://ftp.sas.com/pub/neural/FAQ.html.

Sejnowski, T. and Rosenberg, C. 1987. *Parallel networks that learn to pronounce English text.* Complex Systems, **1**, pp. 145-168.

Siedlecki, W. and Sklansky, J., 1988. *On automatic feature selection.* Int. Journal of Pattern Recognition and Artificial Intelligence, **2**, pp. 197-220.

Smit, J. L., 1997. *The Three Dimensional Measurement of Textured Surfaces Using Digital Photogrammetric Techniques.* PhD. Dissertation, Department of Geomatics, University of Cape Town, Cape Town.

Specht, D., 1990. *Probabilistic neural networks.* Neural Networks, **3**, pp. 109-118.

Taylor, J. G. and Freeman, W., eds., 1997. *Neural networks for consciousness.* Neural Networks, **10** (Special Issue).

Tou, J. T. and Gonzalez, R. C., 1974. *Pattern Recognition Principles.* Addison-Wesley, Massachusetts.

Van Der Merwe, N., 1995. *Development of an Image Matching Scheme Using Feature- and Area Based Matching Techniques.*, PhD thesis, University of Cape Town, Cape Town.

Yadid-Pecht, O. and Gur, M. 1996. *Efficient biologically based pattern-recognizing networks.* Neural Networks, **9**, pp. 1061-1070.

Von der Malsburg, C., 1973. *Self-organisation of orientation sensitive cells in the striate cortex.* Kybernetik, **14**, pp. 85-100.

Weidemann, W., Manry, M., Yau, H. and Gong, W., 1995. *Comparison of a neural network and a nearest-neighbor classifier via the numeric handprint recognition problem.* IEEE Transactions on Neural Networks, **6**, pp. 1524-1530.

Wejchert, J. and Tesauro, G., 1991. *Visualising processes in neural networks.* IBM Journal of Research and Development, **35**, pp. 244-253.

Werbos, P. J., 1974. *Beyond regression: New tools for prediction and analysis in the behavioural sciences.* Ph.D. Thesis, Harvard University, Massachusetts.

Wettschereck, D. and Dietterich, T., 1992. *Improving the performance of radial basis function networks by learning centre locations.* Advances in Neural Information Processing Systems. (Moody, J.E., Hanson, S.J. and Lippman, R.P. eds.) pp. 1133-1140. San Mateo, Morgan Kaufmann.

Willshaw, D. J. and Von der Malsburg, C., 1976. *How patterned neural connections can be set up by self-organisation.* Proceedings of the Royal Society of London, Series B, **194**, pp 431-445.

Widrow, B. and Hoff, Jr., M. E., 1960. *Adaptive switching circuits.* IRE WESCON Convention Record, pp. 96-104.

Wilson, D. L. *Asymptotic properties of nearest neighbor rules using edited data.* IEEE Transactions on Systems, Man, and Cybernetics, **2**, pp. 408-421.