Connectionist-Based Intelligent Information Systems for Image Analysis and Knowledge Engineering: Applications in Horticulture

Brendon James Woodford

A thesis submitted for the degree of Doctor of Philosophy at the University of Otago, Dunedin, New Zealand.

December 11, 2003
Abstract

New Zealand’s main export earnings come from the primary production area including agriculture, horticulture, and viticulture. One of the major contributors in this area of horticulture is the production of quality export grade fruit; specifically apples. In order to maintain a competitive advantage, the systems and methods used to grow the fruit are constantly being refined and are increasingly based on data collected and analysed by both the orchardist who grows the produce and also researchers who refine the methods used to determine high levels of fruit quality.

To support the task of data analysis and the resulting decision-making process requires efficient and reliable tools. This thesis attempts to address these issues by applying the techniques of Connectionist-Based Intelligent Information Systems (CBIIS) for Image Analysis and Knowledge Discovery. Using advanced neurocomputing techniques and a novel knowledge engineering methodology, this thesis attempts to seek some solutions to a set of specific problems that exist within the horticultural domain.

In particular it describes a methodology based on previous research into neuro-fuzzy systems for knowledge acquisition, manipulation, and extraction and furthers this area by introducing a novel and innovative knowledge-based architecture for knowledge-discovery using an online/real-time incremental learning system based on the Evolving Connectionist System (ECOS) paradigm known as the Evolving Fuzzy Neural Network (EFuNN).

The emphases of this work highlights knowledge discovery from these data sets using a novel rule insertion and rule extraction method. The advantage of this method is that it can operate on data sets of limited sizes. This method can be used to validate the results produced by the EFuNN and also allow for greater insight into what aspects of the collected data contribute to the development of high quality produce.
Acknowledgments

This thesis is the culmination of nigh on five contiguous years of research. When I began the initial study that lead to the formation of the thesis topic, I had no idea that this research would take me just about all over New Zealand to parts of the country that I had never been to before. And in doing so I was educated in the diverse range of tasks and methods employed in the horticultural industry. It was an eye-opener for me as I hadn’t realised that not only the intelligent systems technology had not been applied to this discipline but also how complex growing high quality produce actually was!

It has been a difficult task as the problem domain was so novel and posed many challenges but I think I have successfully addressed them. So I would like to thank some special people in my life, both in the social and academic contexts that contributed to this work.

My intial supervisor Professor Nikola Kasabov, for his enthusiasm, guidance, and patience in this project. He aided me in transforming a small seed of an idea into a fully fledged original contribution to the area of neuro-fuzzy systems. I thank you as a mentor and as a friend.

Dr Da Deng and Professor George Benwell for making sure that I continued the research in order to address the final touches to the thesis.

The members of the Crown Research Institute (CRI) HortResearch - Dr. C. Howard Wearing at the Clyde Research Centre, and Dr. Jim Walker and Andrew Hodson of the Havelock North Research Centre. They all helped me to understand the complexities of growing fruit, an activity that I thought was a trivial task. Without their support and knowhow for this project would not have come to fruition.

Alistair Mowat at the Kiwifruit Marketing Board (Zespri). He provided the interface between the techniques I developed and their appropriate application to solve many of the problems encountered in his research.

ENZAFruit New Zealand Limited (ENZA) who supplied the various data sets required for the latter experiments and implementations described in this thesis. Such real-world data was essential to validate the developed methods.

My parents for their long distance encouragement and support. They have now supported me
through two main bodies of research. Being so far away from me, it was comforting to know that they were only a phone call away.

The past and present members of the Knowledge Engineering Laboratory (KEL) based at the Department of Information Science, University of Otago. Especially Mark Laws, Richard Kilgour, Mike Watts, Akbar Gholakhlou, Silvia Raykova, James Calvert, Matt Futschik, and Carl Leichter. They supported me throughout this academic endeavour with a sounding board to bounce ideas off, good mates to share a quiet drink with but ultimately these people made KEL an amazing place to work in and discover what we were capable of in our respective research interests.

I would also like to extend my thanks to members of the Information Science Department, both past and present: Professor Philip Sallis, Professor Steve MacDonell, and Professor George Benwell for their leadership and head of department. Louise Vink, Heather Te Koeti, and Leanne Vink for providing administrative support. The technical support group of John Kaiser, Brendon Sly, Graeme Roxburgh, and Peter George who made sure I have the necessary and appropriate software and hardware required to undertake this research. And finally some past and present members of staff: Peter Whigham, Nigel Stanger, Dax Roberts, Kaaren Dooher, Kirsty Richards, William Wong, Karen Peterson, and Simon Crowther.

The plethora of good friends who have shared in this academic exercise: Vernon Trainor, Harry Harding, Felicity Cotton, James and Leeann Mowat, Andrew and Anna, Stephen Hall-Jones, Sophie Barker, Kerry McPhail, Andrew Stephen, the “Bennu Crew”, and the usual suspects at the Inch Bar.

Finally this research would not have been possible unless funding was made available from the New Zealand Government. This research was part of a research programme funded by the New Zealand Foundation for Research Science and Technology. The grants UOO808, HRO809, and UOXX0016 aided me in all aspects of my research and I thank the government for investing in and supporting innovative research within New Zealand.
**Contents**

Abstract ii

Acknowledgments iii

Contents v

List of Tables xi

List of Figures xiii

**Chapter 1 Introduction**

1.1 Goal of the thesis ......................................................... 1
1.2 Research Questions ......................................................... 2
1.3 Background ................................................................. 2
1.4 Connectionist-Based Intelligent Information Systems .................. 6
1.5 Original Contribution ....................................................... 6
1.6 Outline of the thesis ....................................................... 7

**Chapter 2 Generic Techniques for Image Processing: A Review**

2.1 Introduction ............................................................... 10
2.2 Data Transformations ..................................................... 11
2.3 Data normalisation ....................................................... 12
2.4 Feature extraction ....................................................... 13
  2.4.1 Why is feature extraction so important? .......................... 13
  2.4.2 The Discrete Fourier Transformation .............................. 14
  2.4.3 Gabor Filters ........................................................ 15


v
Chapter 4  Connectionist Based Methodologies for Adaptive Learning and Knowledge Engineering

4.1 Introduction ............................................................................. 71
4.2 On-line, knowledge-based, and adaptive methodologies ........ 72
4.3 The ECOS Principle ................................................................. 77
4.4 The EFuNN ............................................................................. 79
4.5 Previous Rule Insertion and Rule Extraction Methods ............ 83
   4.5.1 NeuroRule and NeuroLinear ............................................. 84
4.6 Rule Insertion and Rule Extraction in EFuNN ......................... 86
4.7 Empirical/Benchmark study ..................................................... 91
   4.7.1 Gas-Furnace data set ....................................................... 92
   4.7.2 Iris Data set ..................................................................... 96
4.8 Analysis of the method ............................................................ 101
4.9 Conclusion ............................................................................. 104

Chapter 5  CBIIS for Image Analysis and Recognition: An application in horticulture

5.1 Introduction ............................................................................. 105
5.2 Analysis of images of pest damage to apples ....................... 107
   5.2.1 Image database generation ............................................ 109
   5.2.2 Image querying .............................................................. 111
   5.2.3 Using an EFuNN to identify pest damage to apples .......... 114
   5.2.4 Training and testing of the EFuNN .................................. 119
   5.2.5 Comparison between the EFuNN model and other classification models 122
   5.2.6 Discussion ..................................................................... 124
5.3 Classifying temporal image data using an EFuNN ................. 125
   5.3.1 Method ........................................................................... 127
   5.3.2 Training and testing of the EFuNN .................................. 128
   5.3.3 Discussion ..................................................................... 131
5.4 Summary and concluding remarks ......................................... 132
Chapter 6  CBIIS for NIR data classification: A case study in horticulture  133

6.1 Introduction ................................................................. 133
6.2 NIR Data Recognition .................................................... 134
6.3 Classification of Persimmon Fruit ...................................... 136
  6.3.1 Background .......................................................... 136
  6.3.2 Method ................................................................. 137
  6.3.3 Training and testing of the EFuNN ................................ 138
  6.3.4 Training and testing of the FuNN ................................. 141
  6.3.5 Training and testing of the MLP .................................. 142
  6.3.6 Training and testing of the SVM ................................. 143
  6.3.7 Discussion ............................................................ 144
6.4 Classification of pine-tree phenotypes .............................. 144
  6.4.1 Problem ............................................................... 144
  6.4.2 Method ................................................................. 145
  6.4.3 Training and testing of the EFuNN ............................... 146
  6.4.4 Training and testing of the FuNN ............................... 149
  6.4.5 Training and testing of the MLP ............................... 150
  6.4.6 Training and testing of the SVM ............................... 152
  6.4.7 Discussion ............................................................ 153
6.5 Summary and concluding remarks .................................... 153

Chapter 7  CBIIS for decision support: Adaptive Expert System design and implementation  154

7.1 Introduction ................................................................. 154
7.2 Traditional Expert Systems ............................................ 155
7.3 The Adaptive Generic Expert System (AGES) ...................... 156
7.4 Rule Extraction in AGES ................................................ 158
  7.4.1 Crisp Rule Extraction .............................................. 159
  7.4.2 Fuzzy Rule Extraction .............................................. 163
Chapter 7 An adaptive expert systems for horticulture: The IPMES

8.1 Introduction ........................................ 174
8.2 Sources of Information ............................ 176
8.3 The Knowledge Base for the IPMES .......... 178
8.4 The Data Base for the IPMES .................. 179
8.5 The Functionality of the IPMES ............... 179
8.6 Architecture of the IPMES ...................... 180
8.7 Description and Functionality of the Modules for the IPMES .............................. 181
  8.7.1 Data Visualisation Modules ................. 182
  8.7.2 Grower Advice Modules .................. 185
  8.7.3 An example of the IPMES in action ....... 188
  8.7.4 Validation of the advice provided by the IPMES .................................. 190
  8.7.5 Threshold Improvement Module .......... 191
8.8 Summary and Conclusion ....................... 194

Chapter 9 Conclusion and Future Directions

9.1 Concluding Remarks ............................ 195
9.2 Future Directions ............................... 196

References ........................................... 199

Appendix A Test Case Scenarios for the IPMES Inference Engine

A.1 Introduction ..................................... 229
A.2 Description of the data set ................................................. 229
A.3 Test Case Scenarios – Inference Engine V1 ......................... 230
   A.3.1 Orchard D1164: Pest Trapping Information .................. 230
   A.3.2 Orchard D1164: Pest monitoring ............................... 232
A.4 Test Case Scenarios – Inference Engine V2 ......................... 236
   A.4.1 Orchard D1164: Pest monitoring ............................... 237

Appendix B  Timings for the experiments conducted in this thesis  240
   B.1 FWT versus DCT versus FFT ...................................... 240
   B.2 Iris benchmark data set ............................................. 241
   B.3 Timings of training for the different classifiers ................. 242
List of Tables

3.1 Four frequently used kernel functions ................................................. 46
4.1 The 29 rules extracted from the EFuNN ............................................. 94
4.2 Results from ten runs of the EFuNN on the Iris data set ....................... 97
4.3 EFuNN when tested on entire Iris data set after three rules inserted .......... 98
4.4 EFuNN when tested on 45 instances of Iris data set after completion of Step 2 . 98
4.5 The 14 rules extracted from the EFuNN ............................................. 98
4.6 EFuNN when tested using the 14 extracted rules .................................. 99
4.7 Results from the other three classifiers .......................................... 100
5.1 Distribution of pest damage examples .............................................. 118
5.2 Results from ten runs of the EFuNN ................................................. 121
5.3 Analysis of the rule set from the best performing EFuNN run ................... 121
5.4 Confusion matrix from run 6 ............................................................... 122
5.5 Results from the other three classifiers .......................................... 124
5.6 A subset of the MRI image data set .................................................... 127
5.7 Distribution of MRI image examples ................................................. 128
5.8 Results from ten runs of the EFuNN over the MRI image data set ............. 129
5.9 The 19 rules extracted from the EFuNN ............................................. 129
5.10 Results from the other three classifiers ......................................... 131
6.1 Results from ten runs of the EFuNN on the Persimmon data set ............... 138
6.2 EFuNN when tested on 40 instances of the persimmon data set ............... 139
6.3 The 17 rules extracted from the EFuNN ............................................. 139
6.4 Results from ten runs of the FuNN on the Persimmon data set ............... 141
6.5 FuNN when tested on 40 instances of the persimmon data set ........................................ 142
6.6 Results from ten runs of the MLP on the Persimmon data set ........................................ 142
6.7 MLP when tested on 40 instances of the persimmon data set ........................................ 143
6.8 Results from ten runs of the SVM on the persimmon data set ........................................ 143
6.9 SVM when tested on 40 instances of the persimmon data set ........................................ 144
6.10 Results from ten runs of the EFuNN on the pine needle data set ................................... 146
6.11 The 16 rules extracted from the EFuNN .......................................................... 147
6.12 EFuNN when tested on 50 instances of the pine needle data set ................................ 149
6.13 Results from ten runs of the FuNN on the pine needle data set ................................... 150
6.14 FuNN when tested on 50 instances of the pine needle data set ................................... 150
6.15 Results from ten runs of the MLP on the pine needle data set .................................... 151
6.16 MLP when tested on 50 instances of the pine needle data set .................................... 151
6.17 Results from ten runs of the SVM on the pine needle data set .................................... 152
6.18 SVM when tested on 50 instances of the pine needle data set .................................... 152
7.1 The 17 rules generated from the CRE algorithm ....................................................... 161
7.2 Table detailing output of classifier implemented in CLIPS based on rules derived using CRE 163
7.3 The 18 rules generated from the FRE algorithm ....................................................... 165
7.4 Table detailing output of classifier implemented in MATLAB based on rules derived using FRE 166
7.5 The 13 rules extracted from the trained EFuNN with two rules initially inserted .......... 169
7.6 EFuNN when tested using the 13 extracted rules ...................................................... 169
7.7 The 13 rules extracted from the trained EFuNN with no initial rules .............................. 170
7.8 EFuNN when tested using the 13 extracted rules ...................................................... 171
8.1 The major insect and mite pest complex attacking New Zealand apples ....................... 175
A.1 Data used for the test case scenarios ........................................................................ 230
B.1 Average timings to process all 96 images ..................................................................... 240
List of Figures

2.1 The different structures of a sine wave and a wavelet adapted and reproduced from Misiti et al. (2000, p. 1–9) .......................... 17
2.2 An image of pest damage inflicted on an apple .................................................. 19
2.3 Comparison between compression and decompression of an image using a wavelet transformation and a discrete cosine transformation ................................................. 19
2.4 Comparison between the number of coefficients required to store the image .................... 20
2.5 Comparison between the number of coefficients required to store the image .................... 21
2.6 Comparison between Haar and Daubechies wavelet transformation of an image .................. 23
2.7 The original Lena image (128x128 pixels) .............................................................. 26
2.8 Comparison of the Lena image rescaled using three different rescaling method to 150% of the original image size .......................................................... 27

3.1 Architecture of a perceptron adapted and modified from Pudi (2003) .......................... 34
3.2 Architecture of a single-layer neural network ......................................................... 35
3.3 A structure of a multi-layer perceptron with three layers ......................................... 36
3.4 Two examples of sigmoidal transfer functions used in feedforward neural networks ............ 37
3.5 Architecture of the ART model ........................................................................... 42
3.6 Architecture of the SVM model. Reproduced from (Cortes and Vapnik; 1995) .............. 47
3.7 Typical MF’s of linguistic values “low”, “medium”, and “high”. Adapted from Bezdek (1993, p. 4) 52
3.8 Example of how we can model the concept of “tall” using fuzzy sets. Adapted from Jang (2001, pp. 2-9) ............................................................ 53
3.9 Modelling the risk of cancer. Reproduced from Kasabov (1996a, p. 186) ................. 56
3.10 The “tipping problem” modelled as a Mamdani-type fuzzy inference system. Reproduced from Jang (2001, p. 2–27) .......................................................... 58
3.11 The “tipping problem” modelled as a Sugeno-type fuzzy inference system. Reproduced from Jang (2001, p. 2-73) ........................................... 59
3.12 A NEFCLASS structure of 2 crisp inputs (input variables), 5 rule nodes, and 2 output classes (output variables). Reproduced from Nauck (1997, p. 1049) ....................... 63
3.13 A FuNN structure of 2 inputs (input variables), 3 fuzzy linguistic terms for each variable (3 membership functions). The number of the rule (case) nodes can vary. Three output membership functions are used for the single output variable. Reproduced from (Kasabov; 1996a, p. 321) ... 65
3.14 A simplified version of a FuNN fuzzy neural network. Reproduced from (Kasabov; 1996a, p. 321) 66

4.1 The architecture of the EFuNN. Reproduced from Koprinska and Kasabov (1999, p. 96) ........ 79
4.2 Rule insertion into an EFuNN ................................................. 87
4.3 Initial testing of trained EFuNN when first half of the gas-furnace data set is used .................. 93
4.4 Testing of trained EFuNN when the entire gas-furnace data set is used after one pass of training and after a second pass of training on the entire data set ................................. 93
4.5 Testing of EFuNN with 29 rules inserted into its structure (no further training) ....................... 95
4.6 PCA projection of the first two principal components of the Iris data set ......................... 96

5.1 Examples of codling moth damage ............................................ 108
5.2 Examples of appleleaf curling midge damage .................................. 108
5.3 Examples of leafroller damage .................................................. 108
5.4 Query image and retrieved images of codling moth damage ...................... 111
5.5 Query image and retrieved images of leafroller damage ............................ 113
5.6 Original and segmented fruit damage .......................................... 115
5.7 Original and segmented leaf damage ......................................... 116
5.8 The process of applying a wavelet transformation .................................. 116
5.9 Output of the process in Figure 5.8 .............................................. 117
5.10 PCA projection of pest data projected onto the first two eigenvectors ................. 118
5.11 The image recognition process .................................................. 120
5.12 Subset of images used to test the classification system ............................. 127
5.13 PCA projection of MRI data projected onto first three eigenvectors ............... 128
6.1 PCA projection of persimmon training data projected onto first two eigenvectors . . . . . . . . 137
6.2 PCA projection of persimmon testing data projected onto first two eigenvectors . . . . . . . . 140
6.3 PCA projection of rule nodes projected onto the testing data for run 9 . . . . . . . . . . . . . . 140
6.4 PCA projection of pine training data projected onto first two eigenvectors . . . . . . . . . . . 145
6.5 PCA projection of Pine testing data projected onto first two eigenvectors . . . . . . . . . . . 148
6.6 PCA projection of rule nodes projected onto the testing data for run 9 . . . . . . . . . . . . . . 148

7.1 A diagram of a traditional expert system. Reproduced from (Kasabov; 1996a, p. 128) . . . . . 156
7.2 A block diagram of AGES, an on-line adaptive expert system. Reproduced from Kasabov and Woodford (1999, p. 1410) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 158
7.3 The two rules inserted into the untrained EFuNN . . . . . . . . . . . . . . . . . . . . . . . . 170

8.1 Information flow of the pest management data . . . . . . . . . . . . . . . . . . . . . . . . . . . 177
8.2 Logical architecture of the IPMES . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 180
8.3 Physical architecture of the IPMES . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 181
8.4 Making a query on trap data contained in the database . . . . . . . . . . . . . . . . . . . . . . 183
8.5 The resulting output of the query specified in Figure 8.4 . . . . . . . . . . . . . . . . . . . . . 183
8.6 The data input screen for the IPMES . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 185
8.7 Data supplied to the IPMES resulting in no spray recommendation . . . . . . . . . . . . . . 188
8.8 Output of IPMES with no spray recommendation . . . . . . . . . . . . . . . . . . . . . . . . . 189
8.9 Output of IPMES with spray recommendation . . . . . . . . . . . . . . . . . . . . . . . . . . . 190
8.10 The user interface for the threshold adjustment module . . . . . . . . . . . . . . . . . . . . . 191
8.11 Screenshot for the insertion of rules . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 192
8.12 Screenshot for the adjustment of rules . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 193
8.13 Screenshot for the extraction of rules . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 193
8.14 Screenshot of the rules extraction from threshold modification . . . . . . . . . . . . . . . . 193
## List of Algorithms

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The data preprocessing and fuzzification algorithm for the EFuNN</td>
<td>81</td>
</tr>
<tr>
<td>2</td>
<td>The EFuNN learning algorithm</td>
<td>82</td>
</tr>
<tr>
<td>3</td>
<td>The EFuNN Rule Extraction Algorithm</td>
<td>89</td>
</tr>
<tr>
<td>4</td>
<td>Algorithm to generate the wavelet coefficients. Adapted from Wang et al. (1998)</td>
<td>110</td>
</tr>
<tr>
<td>5</td>
<td>Algorithm to query the image database. Adapted from Wang et al. (1998)</td>
<td>112</td>
</tr>
<tr>
<td>6</td>
<td>Algorithm for crisp rule extraction from a data set</td>
<td>160</td>
</tr>
<tr>
<td>7</td>
<td>Algorithm for fuzzy rule extraction from a data set</td>
<td>164</td>
</tr>
<tr>
<td>8</td>
<td>Algorithm for linguistic rule insertion into an EFuNN</td>
<td>168</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Goal of the thesis

New Zealand’s main export earnings come from the primary production area including dairying, viticulture, and horticulture. One of the major contributors in the area of horticulture is the production of quality export grade fruit. In order to maintain a competitive advantage, the systems and methods used to grow the fruit are constantly being refined and directly involve two main groups:

1. The grower of the produce who is increasingly required to collect and analyse the data from the orchard in order to satisfy the systems and methods used to grow the fruit.

2. The research institutions who support the research into the development of these systems and methods along with their own research into many aspects of the production of high quality fruit.

To support the process of analysis of the data and the resulting decision-making process, the aim of the thesis is to describe the methods and the techniques of Connectionist-Based Intelligent Information Systems (CBIIS) for Image Analysis and Knowledge Engineering that have been developed with specific applications in horticulture.
1.2 Research Questions

Underlying the development of these tools and techniques has been four major research questions that are addressed throughout this thesis:

1. What type of information is required for decision-making in horticulture and how is this real-world knowledge represented?

2. How do we create appropriate computational models to represent this knowledge?

3. How do we manage and extract this knowledge from these computational models?

4. How can we best apply these developed models for the process of decision-making in horticulture?

1.3 Background

The advent of computer-based information systems has in the past been directly applied to traditional business models and practice supporting the way in which a business conducts its activities. There are many examples of this including Caron et al. (1994), Hess and Kemerer (1994), and Kraemer et al. (2000).

Recent innovations in the area of artificial intelligence, specifically learning systems and data mining techniques have led to the application of these methods and technologies to enhance the traditional information systems. Diverse areas such as agriculture (Witten et al.; 1993; McQueen et al.; 1995), the dairy industry (Bright et al.; 2000), and the international horticultural industry (Lu et al.; 1993) have benefited from such approaches with a view to providing a tangible market advantage in the world marketplace.

Although the advantages of such information systems and their application to the primary producing industry has been well documented, most of them have relied on traditional data processing and storage requirements with which to supply the relevant information to the user. With the amount of data now being collected on the processes with which to develop quality produce
raises issues on what implicit information embedded in these large data sets may exist that could potentially benefit the producer. Furthermore most traditional information systems require robust and accurate data collection mechanisms to function effectively and provide timely and relevant information to the producer.

Such data acquisition, storage, and processing requirements are critical for growing produce to supply the international export market and is especially important to those producers involved in the horticultural industry, for example, those growers involved in the production of quality export grade apples.

In this example, the main driver for the accurate acquisition and storage of data collected on orchards growing apples for the export market has been the introduction of the Integrated Fruit Production (IFP) programme (Walker et al.; 1997); an initiative by HortResearch New Zealand and The New Zealand Apple and Pear Marketing Board (ENZA) to standardise the method by which the fruit are grown. This programme and related document outlines to the grower key information in the maintenance of their orchard over the growing season including:

- Prescribed pesticide applications throughout the growing season.
- Defining thresholds for specific pests found in the orchard. Exceeding these thresholds would action a pesticide application.
- Indication of what pesticide should be applied when a threshold is exceeded.
- Pest phenology and spray timing.
- Orchard disease control based on number of trees/shoots/apples infected from the previous season.

The IFP objective was to maximise the use of biological control through pest monitoring and strategic use of selective pesticides when justified. This required the grower to:

- Monitor for pest and diseases.
- Respond when thresholds are exceeded.
• Use the safest control methods possible.

• Document justifications and responses.

• Supply records for auditing as required.

Unless these data collection activities can directly benefit the group of New Zealand apple producers, then they will not “buy-in” to this new method of apple production. There has to be some output from this programme other than a better quality apple. Compounding this issue is the fact that since the year 2000, all orchardists who intend to grow apples for the international market must strictly adhere to these guidelines in order to have their crop accepted for export.

Traditional information systems can offer a part solution to this problem by supplementing the decision support and knowledge management of the grower. But as apple production is a complex activity and requires direct application of the knowledge of experienced growers in order to produce a good quality crop, then new methods and techniques need to be developed to support this process.

One area of information science could be of use in this context. With the advent of expert systems design and implementations such as MYCIN (Gasching et al.; 1983), artificial neural networks (Kohonen; 1997; Kasabov; 1998a), more powerful data visualisation techniques (Ma; 2000), and data mining methods (Pazzani; 2000), a combination of these technologies could be applied to complement the growing process. However, as yet, there has been little progress in the application of these techniques to the horticultural area.

This lack of technology uptake in the area of horticulture could be explained by the current state of these tools and techniques. Until recently, most of these knowledge engineering methods had been developed on the assumption that the data is a static picture of a process, accurate, and with little uncertainty built into the data itself. In other words there is little capability for these methods to adapt to changes in the environment from which the data were acquired.

One could further hypothesise that the reason for this is a lack of a methodology for the development of such intelligent information systems and their direct application to these and other problems in the horticultural area. For example growing apples is a task that is in real-time and
requires techniques and technologies that also operate in this fashion. Most conventional knowledge engineering techniques do not fit comfortably into the way in which an orchard develops over time.

Computer based learning systems, primarily connectionist-based architectures such as artificial neural networks (Rumelhart et al.; 1986b; Carpenter and Grossberg; 1991; Kohonen; 1990) possess this functionality and can model dynamical processes over time. But there are still deficiencies to this approach. They still operate in an off-line fashion and are unable to learn directly from incoming data in a real-time and on-line environment.

In order to successfully apply these types of soft computing models to the horticultural area it requires an extension of the current mechanisms by which these structures learn into an on-line real-time environment that supports incremental learning of a process. But this approach still requires a framework for the development of these tools to be of any benefit to the grower. In addition, as a connectionist system is to be employed, there are several issues about how this knowledge would be stored in an appropriate neural network architecture.

A corollary to this argument would be to propose a methodology for developing intelligent on-line information systems to benefit the horticultural industry in New Zealand but within a specified domain such as the process of growing apples or innovative methods of pattern recognition of horticultural data. Contained within this system would then be a set of tools and technologies that could then be directly applied with a view to:

- Enhancing presentation of the data collected on the development of the produce.
- Supplementing the decision making process of the grower of the produce.
- Engineering information from the data collected on the produce that is embedded within the data set.
- Applying real-time, on-line learning systems to the process of complementing the processes in which the process of growing the produce is achieved.
- Creating robust pattern classification mechanisms, specifically for image and Near InfraRed spectroscopy (NIR) data, to identify specific characteristics of the collected data...
The methodology is also extensible and adaptable towards the development of other on-line and real time adaptive learning techniques and tools for knowledge engineering in the horticultural domain.

1.4 Connectionist-Based Intelligent Information Systems

This thesis extends the area of Connectionist-Based Intelligent Information Systems (CBIIS) (Kasabov, Purvis and Sallis; 1996). These are emerging computer systems and at their core are connectionist-based computing modules such as artificial neural networks. CBIIS include a variety of other artificial intelligence methods, such as fuzzy inference mechanisms, genetic algorithms, evolutionary computations, agent-based distributed computations, artificial life, as well as standard mathematical and statistical methods. CBIIS are characterised by features such as learning from data, generalisation, adaptation, knowledge acquisition, dealing with inconsistency, incompleteness and noise in data, and handling intelligently large quantities of data from multiple sources. These features can be summarised as computational intelligence. With the explosive increase of electronic information, CBIIS has been shown to be a leading role in medicine (Buchanan and Bilkey; 1997), manufacturing (Kasabov; 1998d), business (Kasabov and Fedrizzi; 1999), science (Song and Hegg; 1999), government (Deng and Kasabov; 1999), agriculture (Kim et al.; 1999) and telecommunications (Iliev and Kasabov; 2000) for solving complex tasks such as classification, decision-making, prediction, speech and language processing, image recognition, planning, modelling and process control. In this thesis we apply the same principles to the challenging issues surrounding decision-support and knowledge-engineering in the domain of horticulture.

1.5 Original Contribution

The original contribution of this thesis is in five main areas:
1. The identification, acquisition, and representation of information required for applying CBIIS to decision-making in horticulture.

2. The development of an incremental, on-line, adaptive, real-time learning CBIIS model to aid in the classification of horticultural data.

3. Generic knowledge management and acquisition methods within this CBIIS model using a new rule extraction and rule insertion algorithm.

4. An architecture for an on-line, adaptive, agent-based expert system that employs the CBIIS model.

5. The application of this new expert-system architecture for decision-support in apple orchard management.

1.6 Outline of the thesis

In this chapter the formulation of the thesis has been presented. Starting with a brief introduction to standard information, several key issues have been raised as to their application within the horticultural domain. Although previous systems have existed in the past whose objective was to complement the decision-making process of the grower, those techniques and technologies have failed to address the real-time nature of the complex interactions of how the dynamics of an orchard change over time as the growing season progresses.

As growing produce for the export market is now a much more regulated and methodical task, information systems that model these processes need to also be adaptive, work in real-time and also provide information to the grower that goes beyond that of what is explicitly represented in the data sets that is collected on these processes. To address these requirements, a new methodology needs to be devised along with a set of techniques, tools, and technologies to implement it.

Therefore in order to address these issues, this thesis describes in detail this methodology and its associated techniques and tools and applies it in a series of case studies dealing with the
horticultural industry in New Zealand. To add context to the subsequent flow of the thesis, a brief description of each subsequent chapter is presented below.

Chapter 2 and Chapter 3 presents the fundamental generic methods for the construction of adaptive, on-line decision-making and recognition systems for knowledge engineering. Here we describe these techniques and methods and also address how they have been applied in the past. Based on the previously conducted work, a set of requirements for the development of methods to be directly applied to the horticultural industry for knowledge engineering is derived.

In Chapter 4 a selection of the methodologies that apply the techniques and methods discussed in Chapters 2 and 3 is presented with a view to developing the underlying framework of a methodology for the design and development of connectionist-based intelligent information systems. This methodology is then extended by introducing the concept of incremental learning for on-line intelligent systems and also real-time pattern recognition. This new methodology is then applied to knowledge engineering in horticulture.

Chapter 5 details one case study in image analysis and recognition that draws on the methods, techniques, and methodologies described in Chapter 4. Using a combination of wavelet-based transformations and an incremental on-line learning system, a robust image recognition engine was developed to identify pest damage to apples. The advantage of this system over traditional image recognition methods is that it is able to work with data sets where the size is limited.

A similar experiment is described in Chapter 6 by applying this methodology to create a classifier to determine the phenotype of different pine needles and Persimmon fruit using Near InfraRed (NIR) data. Comparison of the on-line incremental learning mechanism against statistically based classifiers and neural network based models are presented along with what knowledge can be elicited about these different phenotypes.

And as the horticultural area requires decision-making processes that are governed by the real-time dynamics of an orchard as it progresses over time, a computer-based decision support system should also adapt to the growers’ needs. In Chapter 7 an architecture for an Adaptive Generic Expert System (AGES) is presented to address this issue.

Chapter 8 applies the AGES framework in the form of an adaptive, on-line, real-time, incremental learning expert system called the Integrated Pest Management Expert System (IPMES).
In this chapter the development and architecture of the IPMES data visualisation, decision support, and knowledge extraction modules are described along with the justification for selecting the programming language that it was implemented in.

Finally Chapter 9 concludes the thesis by summarising the work undertaken, emphasising its original contribution, and then examines how this methodology can be extended and applied in the future.
Chapter 2

Generic Techniques for Image Processing: A Review

2.1 Introduction

The building blocks of most intelligent systems are grounded in the areas of data transformations, feature extraction, and data dimensionality reduction. Here an appropriate technique is applied to a raw data set with the objective of eliciting some characteristic of the data set with the objective of then using these meaningful features to create a model whose purpose may be for classification or prediction.

In this chapter we cover a subset of these techniques which are candidates for the purpose of selecting the most relevant features from the various horticultural data sets used in this thesis. Some of the former described techniques relate to feature selection, whilst the latter methods are used as the basis for visualisation of the knowledge elicited from the proposed classification model.

More specifically Section 2.2 provides the criteria for the selection of the techniques used in this thesis. As all the data sets described in this thesis required some form of pre-processing, Section 2.3 reviews a set of simple data normalisation methods. The more complex techniques are then described in detail. Section 2.4 investigates the general area of feature selection with a view to describing the main feature selection method used in this thesis. Since some of the data sets used in the experiments are images, Section 2.5 looks at the task of image transformations.
Section 2.6 presents a review of dimensionality reduction techniques for one way of visualising the output of the proposed classification model, and finally Section 2.7 highlights an area where most of these techniques have been combined to form a solution to the problem of accessing repositories of images. This section has been included for completeness as this topic relates to some of the material and experiments reported in Chapter 5.

2.2 Data Transformations

Governing the decision-making process for the orchardist is the data they acquire throughout the growing season. To identify and represent this data as information in a CBIIS requires that the data be transformed through specific and relevant data transformation methods or techniques. These data transformations lend themselves to more effective analysis of the data for two reasons. Firstly, the direct output of a data transformation may immediately provide some insights into some underlying characteristics of the data set and secondly, the resulting output may be used as the input to a classification or prediction model.

Sometimes the representation of the data set may not be in a format for those who wish to effectively analyse the data, and many techniques, discussed in this thesis, involve some degree of pre-processing of the data or a translation of the data set into a different data domain. Driving the choice of what data transformation to apply is based on the main aim of what the data transformation does to the data. In this thesis we cover four criteria which we consider to be key influences on the choice of data transformations employed in the methodology.

The first criterion is how effective the data dimension reduction process is where meaningful features are extracted from the data. These features normally correspond to a particular attribute or attributes that describe some important aspect of the data set. In this thesis we argue that feature selection using the Wavelet Transformation (WT) (Chakrabarti et al.; 2001) which is described in Section 2.4.4 produces a set of relevant compact features crucial for the efficient online learning ability of adaptive artificial neural networks for both image and signal recognition.

The second criterion is the level of quality or clarity of the acquired data via noise suppression or image enhancement. Data acquired from a real-world environment is prone to noise and
effective analysis of data only occurs if the data is as free from this noise as possible. By either reducing this noise in the raw data or enhancing certain characteristics of the data we want to analyse, much higher quality data is then obtained. For example, recent approaches to noise suppression can be found in Cohen and Berdugo (2001), Hamza and Krim (2001), and Stahl et al. (2000) whilst Tang et al. (2001), Lemeshevsky (2002), and Rahman et al. (2002) propose new solutions to image enhancement.

The third criterion is how beneficial the data transformation would be for knowledge discovery and better understanding of the processes and events that are to be modelled. Certain data transformations allow for much more detailed analysis of the process from which the data was acquired. Implicit information that is not obviously present in the raw non-transformed data may then be elicited using these transformations. We cover these techniques in Section 2.4 and argue that the output of a WT serves as the basis for extraction of the underlying knowledge that is contained in the raw data set.

Finally, the fourth criterion is how appropriate the data transformation is for finding similarities and analogies between processes, events, and phenomena. Sometimes more complex data transformations can identify causal relationships between data sets acquired from the same domain. The analysis of macroeconomic data, for example, is one case where such analysis is conducted. Stock and Watson (2001) look at the use of vector autoregression for this purpose.

This is especially important when developing a holistic picture of a complex system; for example, the interactions between pest numbers and pesticide applications in an apple orchard over the course of the growing season.

Therefore in Section 2.3 to Section 2.6 we define, describe, and justify our selection of specific data transformations for the pre-processing and subsequent analysis of horticultural data based on the four criteria described above.

### 2.3 Data normalisation

This simple transformation is widely used to rescale values in a raw data set into values generally between [-1,1] or [0,1] (Kasabov; 1996a). Equation 2.1 is an example of a linear normalisation
of a data set where \( x_{ij} \) is a particular value in a data set, \( x_{i,min} \) is the smallest value in the data set, and \( x_{i,max} \) is the largest value in the data set.

\[
x_{ij,norm} = \frac{x_{ij} - x_{i,min}}{x_{i,max} - x_{i,min}}
\]  

(2.1)

Linear normalisation has been previously used to normalise microarray databases (Sherlock et al.; 2001), and has been extensively used by Michie et al. (1994) to pre-process data for presentation to various classification models.

Another variation on linear data normalisation is logarithmic normalisation where the data is transformed using the logarithmic scale resulting in values that have a small dynamic range across the entire data set. This type of normalisation is generally used on seismic data (Jeffrey et al.; 1999) and more recently on microarray data (Yang et al.; 2002) when the original data values span a wide range of values. In both studies the natural logarithm was used to define the logarithmic scale. Equation 2.2 is an example of logarithmic normalisation where \( x_{ij} \) is a particular value in a data set, and \( m = min(x_i) \).

\[
x_{ij,norm} = \ln(x_{ij} - m + 1)
\]  

(2.2)

Although the nature of some data sets reported in this thesis did exhibit a wide range of values, it was decided that linear normalisation was best suited to transforming them since all the learning models required that the data be normalised between the range of 0 and 1.

## 2.4 Feature extraction

### 2.4.1 Why is feature extraction so important?

Before a classification model can be properly designed or effectively used however, it is necessary to consider the problems of feature extraction and data reduction.

Consider multivariate data, \( \mathbb{R}^N \), that occupies \( N \) dimensions. One assumption that can be made is that the underlying structure of data in \( \mathbb{R}^N \) is almost always of a dimension lower than \( N \) (Scott; 1992). In other words although \( N \) may be very large, the relevant structure of the solution
data space may be in a dimension lower than \( N \) (Scott; 1992). Hence it is possible to partition the input space into subspaces of the signal or image we are interested in with the residual subspace possibly considered to be noise. Due to the imprecise nature of the partitioning process, the objective is to eliminate as many dimensions of the data to obtain an efficient representation of the underlying structure. In the context of this thesis we define feature extraction as the process of choosing those features which are most effective for image and signal class separability.

This is perhaps the most challenging task in classification; determining a feature set which accommodates the difficulties in the selection or extraction process, and at the same time results in acceptable performance. Because of this fact feature extraction plays a central role in image and signal recognition.

The class of feature extraction methods we chose to examine in this thesis are based on some statistics derived from the data set. Statistical feature extraction has received a great deal of attention. This is because statistical methods lend themselves to direct mathematical description and machine implementation. The significant contributors to statistical feature extraction are the orthogonal transformation methods such as the Discrete Fourier Transformation (DFT) (Brigham; 1988) and the WT. We examine both these approaches as they have been widely used in the area of signal and image processing.

### 2.4.2 The Discrete Fourier Transformation

The analysis of two-dimensional (2D) signals such as images sometimes require that a data reduction or feature extraction method be applied in order to isolate the regions of the image that need attention. This is because we may only be interested in parts of the two-dimensional signal and require a method of isolating them. One commonly applied data transformation is the Discrete Fourier Transform (DFT) (Winograd; 1976), expressed in Equation 2.3, that multiplies the signal \( a_k \) by an exponential \( 2\pi i \) where \( i^2 = -1 \). The resulting Fourier coefficients, \( A_j \), can then be used to compute the sinusoidal components of the original signal by multiplying the coefficients by a sinusoid of frequency \( 2\pi i \) where \( i^2 = -1 \).
\begin{equation}
A_j = \sum_{k=0}^{n-1} w^{kj} a_k; \quad j = 0 \ldots, n - 1, \quad w = e^{2\pi i/n}
\end{equation}

Yet there are problems with the standard DFT:

“However, most interesting signals contain numerous nonstationary or transitory characteristics: drift, trends, abrupt changes, and beginnings and ends of events. These characteristics are often the most important part of the signal, yet Fourier analysis is not suited to detecting them.” (Misiti et al.; 2000, p. 1–5)

Because the DFT assumes that the signal is piecewise smooth, these small discontinuities within the signal may not be detected (Vetterli; 2001). This is especially important with image data as the images of developing apples change over time and it is necessary to know what characteristics of the apple have changed; for example if an apple has been infected with a disease or damage has been inflicted on it by a pest then these characteristics may be appear as discontinuities within the signal.

A DFT analysis of 2D signal data like images requires a more localised approach. Although the Short-Time Fourier Transform (STFT) (Weldon et al.; 1996b) can offer a partial solution by shifting a window of a particular size over the signal. However the size of the window governs the accuracy of the Fourier coefficients obtained from this transform. Many signals, especially image signals, require a more flexible approach where the window size can vary to more accurately determine the frequency information.

### 2.4.3 Gabor Filters

An alternative to Fourier analysis is to use Gabor filters. These filters provide a compromise between spatial and frequency resolution and are well suited to texture region extraction where the localisation of the texture is important.

Weldon and Higgins (1996b) describe the Gabor filter as:

“… a harmonic oscillator, composed of a sinusoidal plane wave of a particular frequency and orientation within a Gaussian envelope. The frequency (\(\omega\)), bandwidth (\(\sigma\)), location
The Gabor function $G$ is defined as follows:

\[
G(x, y, \sigma, \omega, \phi) = J(x, y, \sigma, \omega, \phi) \exp \left( \frac{(x - x_o)^2 + (y - y_o)^2}{-2\sigma^2} \right)
\] (2.4)

where the harmonic oscillator $J(x, y, \sigma, \omega, \phi)$ is defined as:

\[
J(x, y, \sigma, \omega, \phi) = \sin(\omega [x \cos \theta - y \sin \theta] + \phi)
\] (2.5)

The centre of the Gaussian is specified by $x_o$ and $y_o$ and the standard deviation along both axes is specified by $\sigma$. $\omega$, $\phi$, and $\theta$ determine the frequency of the sinusoidal plane wave, the angle of orientation, and the phase of the plane wave respectively.

Texture segmentation by Gabor filters are realised by tuning these filters to the dominant spectral information contained in the image. Thus the selection of parameters for bandwidth, orientation, and frequency needs to be carefully considered. One overriding criteria for this selection is the type of image to be segmented.

In Weldon and Higgins (1996a); Weldon et al. (1996a) Gabor filters have been successfully used for texture segmentation but use banks of Gabor filters where the parameters of the constituent filters are not restricted thus increasing the number of textures that could be segmented.

However for a practical application, Gabor filters have their drawbacks. If there is a marked variation between the images then this would increase the number of filters as selection of the filters is dependent on the image. Furthermore as Gabor filters are not orthogonal this increases the computational complexity and duration to generate the filter coefficients.

### 2.4.4 The Wavelet Transformation

Effective signal processing is governed by the ability to describe the local parameters of a selected basis function in both frequency and time. Two types of horticultural data fit into this category. The first are 2D colour images, for which the time domain is the spatial location of certain colour pixels and the frequency domain is the intensity variation around a pixel which
is normally represented as a value between 0 and 255. Localised variances in either the spatial location or frequency domain may indicate areas of interest within the images that require further investigation. The second are NearInfraRed (NIR) data extracted using a Fourier Transform Infrared Spectrometer. These data normally contains up to 2000 spectra and only subtle changes in the signal indicate the important characteristics of the signal.

Wavelets (Daubechies; 1990) exhibit these properties that can analyse these two types of signals. They are a waveform with two main characteristics; that they have a specified length and the values that the values of the wavelet are composed of average out to zero. They tend to be regular and asymmetric and because of these characteristics are better at analysing signals with sharp changes due to their irregularity (Misiti et al.; 2000). Figure 2.1 depicts the difference between a sine-wave and a wavelet. As can be seen, the sine wave, because of its inherent structure may not have the scale to detect subtleties within a signal.

Figure 2.1: The different structures of a sine wave and a wavelet adapted and reproduced from Misiti et al. (2000, p. 1–9)

Mathematically a wavelet, $\psi$, can be defined as in Equation 2.6 where $t$ denotes time.

$$\int_{-\infty}^{+\infty} \psi(t) dt = 0 \quad (2.6)$$

which is dilated with a scale parameter $s$, and translated by $u$ as in Equation 2.7

$$\psi_{u,s}(t) = \frac{1}{\sqrt{s}} \psi \left( \frac{t - u}{s} \right) \quad (2.7)$$
Conceptually, the process of wavelet transformation occurs by shifting the wavelet across the signal and generating a correlation coefficient that describes how closely the wavelet resembles the signal. The wavelet is then scaled by a factor that stretches it to cover more of the signal and it is then shifted across the signal again. Misiti et al. (2000, p. 1–13) concisely describes this procedure as a five step process:

1. Compare a selected wavelet to a section at the beginning of the original section.
2. Determine how much the wavelet resembles that section of the signal by calculating a number, $C$. The higher the value of $C$, the closer the wavelet correlates to that section of the signal.
3. The wavelet is shifted to the right and Steps 1 and 2 are then repeated until the wavelet covers the entire signal.
4. The wavelet is returned to the start of the signal, scaled (stretched) by a scaling factor and then Steps 1 to 3 are repeated.
5. Steps 1 to 4 are then repeated for all scaling factors.

The result of this process returns a set of coefficients, $C$, that are an effective regression of the original signal carried out over the wavelets.

The nature of wavelets and their scaling quality makes for analysis of different data types that other signal processing techniques may overlook. Important facets of the data, such as their trend, points where the signal breaks down, a possible discontinuity in the signal, or parts of the same signal which are similar (Misiti et al.; 2000, p. 1–8). Recent applications of wavelets include signal de-noising (Durand and Froment; 2001), image compression (Hilton et al.; 1994), image smoothing (von Sachs and Schneider; 1996), and fractal analysis (Degaudenzi and Arizmendi; 2000). All these applications indicate that wavelets offer significant advantages over the traditional techniques due to the way that this paradigm views the data.

**Wavelet transformations versus the DFT**

To illustrate the difference between a conventional Discrete Fourier transformation and the wavelet transformation, the image in Figure 2.2 was compressed using both a Discrete Wavelet Transformation (DWT) where a Coiflet wavelet with 2 bases was used for the DWT and a Discrete Cosine Transformation (DCT). 5% of the coefficients from each respective transformation used to reconstruct the image.
Figure 2.2: An image of pest damage inflicted on an apple

Figure 2.3(a) is the result of the reconstruction of the original image from 5% of its wavelet coefficients. Note the clarity of the image is still present. The image in Figure 2.3 is the result of the reconstruction of the original image from 5% of its DCT coefficients. As can be seen, the DWT transformation has retained better features to describe the image than the DCT transformation.

(a) DWT transformation  (b) DCT transformation

Figure 2.3: Comparison between compression and decompression of an image using a wavelet transformation and a discrete cosine transformation
The number of coefficients produced by each respective transformation, 65535 in total occupying 524288 bytes, is also important. In Figure 2.4, the graph shows that fewer wavelet coefficients are required to obtain the same error between the reconstructed image and the original image.

![Wavelet Compression vs. DCT Compression](image)

**Figure 2.4:** Comparison between the number of coefficients required to store the image

This point is further illustrated by Figure 2.5 where the errors between the original image and the two different transformations are compared. If there were no errors present between the original image and the reconstructed image then the image would be completely white. In this figure, important information corresponding to the pest damage itself is lost in the DCT compression/reconstruction of the image. This large error is depicted as black pixels on the image. The error between the original image and the DWT compression/reconstruction of the image is far less and of less magnitude.
Which wavelet to choose?

The wide applicability of wavelets for signal analysis and recognition is due to the many different flavours of wavelets that could be applied and selection of the most appropriate wavelet basis function is critical to the analysis of the signal or image. Mojsilovic et al. (2000) argues that certain families of wavelets, namely orthonormal wavelets, are more appropriate for analysis of images where the main artifact is a texture. This is also true for apples as a change in texture on the apples surface would indicate pest damage to the apple.

In order to determine the best wavelet for a signal recognition task, work done by the University of Washington (Jacobs et al.; 1995) became the basis for our investigation. Here the authors demonstrated that the Haar wavelet was a suitable candidate for multiresolution image querying. In instances where the query was a hand sketch of a low-quality image, the authors reported that their algorithm was much faster but just as accurate as the traditional algorithms.

However, the nature of the Haar wavelet is that it is incapable of effectively analysing images where there are frequent sharp changes in colour. Furthermore in Wang et al. (1997), the authors do not consider the Haar wavelet as a good candidate for the analysis of natural images or signals since this wavelet creates more residual noise when analysing natural signals compared to more advanced families of wavelets. The reason is based on the architecture of the Haar wavelet. The base functions are discontinuous step functions and are not well suited to the analysis of continuous functions with continuous derivatives (Wang et al.; 1997). When one regards images
to be 2D continuous surfaces where the spatial location of a certain colour pixels represents the
time domain and the colour variation around the pixel itself represents the frequency domain
then the properties of the Haar wavelet are deemed inappropriate for image analysis.

Therefore in Wang et al. (1996, 1997), the authors proposed Daubechies’ wavelets (Daubechies;
1990, 1988) as an alternative to the Haar wavelet. This family of wavelets offer better representa-
tion of the image semantics, namely object configuration and local colour variation, both of
which can be represented by Daubechies’ wavelet coefficients.

In Wang et al. (1997) they describe the basis for Daubechies’ wavelets. Given an integer \( r \),
the Daubechies’ orthonormal basis for \( L^2(\mathbb{R}) \) is defined as

\[
\phi_{r,j,k}(x) = 2^{j/2} \phi_r(2^j x - k), \quad j, k \in \mathbb{Z}
\]  

(2.8)

This assumes that the function \( \phi_r(x) \) in \( L^2(\mathbb{R}) \) possesses the property that \( \{ \phi_r(x - k) | k \in \mathbb{Z} \} \)
is an orthonormal sequence in \( L^2(\mathbb{R}) \). For each function \( f \in L^2(\mathbb{R}) \), the trend \( f_j \) at scale \( 2^{-j} \) is
defined as

\[
f_j(x) = \sum <f, \phi_{r,j,k} > \phi_{r,j,k}(x)
\]  

(2.9)

where the “details” or “fluctuations” are defined by

\[
d_j(x) = f_{j+1}(x) - f_j(x)
\]  

(2.10)

Wang et al. (1997) modified the original orthonormal basis \( \phi_r(x) \) by introducing an orthonor-
mal basis \( \psi_r(x) \) that has similar properties to that of the Daubechies’ orthonormal basis. For
wavelet analysis they defined two prototype functions \( \psi_r(x) \) and \( \phi_r(x) \) which are referred to as
the father wavelet and mother wavelet respectively. Wavelets like those defined in Equation 2.8
can then be generated from the father or mother wavelet by altering the scale and translation in
space for the purpose of image processing.

Favourable results were reported in Wang et al. (1997) for the purpose of image analysis
using Daubechies’ orthonormal basis as they identified three suitable properties of this family of
wavelets:

- The support interval for \( \psi_r \) was \([0, 2^r + 1]\);
• There are about \( r/5 \) continuous derivatives for \( \psi_r \);

\[
\int_{-\infty}^{\infty} \psi_r(x) \, dx = \ldots = \int_{-\infty}^{\infty} x^r \psi_r(x) \, dx = 0
\]

In this thesis we adopted Daubechies’ wavelets as a more appropriate wavelet to use for image and signal analysis due to the above properties. This is because functions with more continuous derivatives analyse continuous functions more efficiently and avoid the generation of edge artifacts. A Daubechies’ wavelet transform is more like a weighted average which better preserves the trend information stored in the signals only if the low-pass filter information is considered. Various experiments and studies have shown Daubechies’ wavelets are better for dealing with general-purpose images including Wang et al. (1998) and Mojsilovic et al. (2000).

We can illustrate the superior properties of Daubechies’ wavelets over Haar wavelets by applying a 2D 3 layer DWT transformation over the image depicted in Figure 2.2.

![Haar wavelet decomposition of image](image1.png)

Three layer Haar wavelet decomposition of image

![Daubechies wavelet decomposition of image](image2.png)

Three layer Daubechies wavelet decomposition of image

(a) Haar transformation  
(b) Daubechies transformation

Figure 2.6: Comparison between Haar and Daubechies wavelet transformation of an image

In both Figures 2.6(a) and 2.6(b), the black dots indicate non-zero coefficients. The resulting output from the Haar wavelet transformation indicate that more than just the texture of the pest damage was extracted. The leaves in the top right-hand corner of the image were also detected. This is in contrast to the resulting output using Daubechies wavelets where the damage in the

23
second image from the top left-hand corner of the wavelet decomposition in Figure 2.6(b) is more obvious than the corresponding image in Figure 2.6(a).

This wavelet has also been applied in the area of wavelet-based image processing techniques and neural networks to develop a method of on-line identification of pest damage in pipfruit orchards (Woodford, Kasabov and Wearing; 1999; Kasabov, Israel and Woodford; 2000a). In these studies Daubechies’ wavelets were used as they were found to perform better extraction of the relevant features for the identification of images of damaged apples because of its ability to better represent the colour variation and local characteristics of the image. For NIR data, the Daubechies’ wavelet transform has also been successfully used to classify different clones of pine trees (Woodford; 2001).

2.4.5 Why use wavelet transformations for feature extraction?

There are many other methods we could have used for feature extraction from images. Techniques such as image segmentation (Meila and Shi; 2001; Goldenberg et al.; 2002; Deng and Manjunath; 2001) and independent component analysis (Hoyer and Hyvärinen; 2000), could have been used as the feature extraction scheme however three main reasons for not using these techniques are listed below:

1. The problem is not an image segmentation task: Image segmentation is normally used when the task is to measure some characteristic of the object or objects in the image. In the experiments reported in Chapter 5 we merely wanted to detect changes within the image which may indicate the presence of damage.

2. On-line/real-time processing and classification: The wavelet transformation is directly applicable for on-line processing of the image and subsequent real-time classification of the image. Section B.1 of Appendix B contains the results of performing a DWT, DCT, and FFT on a set of 96 images used for the experiments conducted in Chapter 5. Although the DWT transformation is slower than the FFT and the DCT. It still takes, on average, less than 1/100 of a second to transform an image.
3. **Knowledge discovery**: The coefficients generated from a wavelet transformation combined with a classification mechanism that allows for rule extraction, can produce rules that are directly interpretable. These rules can then provide insight into what specific wavelet coefficients contribute to the classification of an image.

## 2.5 Image Transformations

Complementing the methods for wavelet analysis of images are the techniques related to image transformations. Such transformations are required when pre-processing the image data. For example images such as the ones used in the study contained in Chapter 5 were required to be resized before the wavelet analysis was applied. The choice of rescaling algorithm normally incurs a tradeoff between the speed of processing and the degree to which the quality and clarity of the image is preserved (Bourke; 2001).

Since the emphasis was on on-line and real-time analysis of the images, a suitable image rescaling method needed to be included that found a balance between time to process the image and the quality of the resulting resized image. There were two reasons for this. The first was to resize the images to a standard 128x128 to better suite the wavelet transformation since matrices that were of the power of two could be processed faster. And the second was that the original images were of varying sizes ranging from 296x199 pixels to 300x220 pixels.

Consider the image in Figure 2.7. This is the frequently used Lena image (Munson Jr; 1996) used to demonstrate the effects of different image transformation methods on a chosen image. The complexity of the image presents challenges to many different image transformation algorithms.
One simple method to resize the image could be based on the nearest neighbour algorithm (Unser et al.; 1995). Given a pixel \((i, j)\) in the original image of dimensions \(w\) and \(h\), the closest corresponding pixel \((i', j')\) is found in the resized image of dimensions \(w'\) and \(h'\) where the point \((i', j')\) is computed by

\[
i' = \frac{i w'}{w} \\
j' = \frac{j h'}{h}
\]

where the division above is integer and any remainder is ignored. Although this form of interpolation is the fastest to compute, Bourke (2001) identified that result of this interpolation method, whether for increasing or decreasing the size of the image, resulted in blocky “stair-stepped” diagonal lines in the image.

Bilinear interpolation uses the weighted average of the nearest four pixels to the output pixel and reduces the “stair-stepped” effect of the nearest neighbour method resulting in a smoother image. But two disadvantages of this method identified by Jervis (2002) are that the original data is altered and the contrast of the image is reduced through taking the weighted average. In addition it is computationally more expensive than the nearest neighbour approach.

The standard approach, bicubic interpolation, evaluates a block of 16 nearest pixels therefore not strictly a linear interpolation. The mean and variance of the output distribution match the input distribution however the data values themselves may be altered. This method both sharpens the image and smooths out any noise but is the most computationally expensive image scaling operation.

To illustrate the differences between these three methods, the image depicted in Figure 2.7
was scaled up to 150% its original size using the three different methods described above. In Figure 2.8 it is clear to see that bicubic interpolation is the best method out of the three with which image rescaling can occur without too much loss of detail.

Figure 2.8: Comparison of the Lena image rescaled using three different rescaling method to 150% of the original image size

2.6 Dimensionality reduction methods

Related to the problem of feature extraction is the task of dimensionality reduction which determines the best subset or combination of features for the purpose of classification. Reducing the dimensionality of the problem simplifies the task of the classifier, and alleviates the generalisation problems due to the curse of dimensionality.

There are many different dimensionality reduction techniques that can be used including Multidimensional Scaling (MDS), Sammon mapping, and Principal Component Analysis (PCA). We describe these in the context of explaining the figures for visualising the extracted rules in Chapter 6.

2.6.1 MDS

MDS covers a wide variety of multivariate data analysis techniques originally developed in mathematical psychology but now applied to common problem areas such as molecular biology and
linguistics (Hughes and Lowe; 2003).

Its main objective is to realise proximity relations of objects by distances between points in a low-dimensional Euclidean space. The proximity values are represented as dissimilarity values. Mathematically, the dissimilarity of object $i$ to object $j$ is defined as a real number $\delta_{ij}$. The MDS algorithm determines a spatial representation of the objects where each object is represented by coordinates $x_i \in \mathbb{R}^M$ in a $M$-dimensional space. The distance between two points $x_i$ and $x_j$ of $X$ is usually measured by Euclidean distance (Klock and Buhmann; 1996).

The main problem with this dimension reduction technique is that this classical technique has an inherent batch character to it. For every new data item that is added, the program has to recompute the proximity values as adding data will modify the embedding of the old data as well. Therefore MDS is not suited to the real-time visualisation of the rule sets.

### 2.6.2 Sammon projection

One popular non-linear method for data reduction has been Sammon Mapping (Sammon; 1969), sometimes referred to as non-linear mapping. Sammon Mapping is determined by the optimisation of an error, or ‘STRESS’, measure which attempts to preserve all interpoint distances under the projection. The Sammon STRESS is defined as

$$E_{ss} = \frac{1}{\sum_i \sum_{j<i} d_{ij}^* \sum_i \sum_{j<i} \left[ \frac{d_{ij}^* - d_{ij}}{d_{ij}} \right]^2}$$

(2.11)

where $d_{ij}^*$ is the distance $\|x_i - x_j\|$ between points $i$ and $j$ in the input space $\mathbb{R}^p$, and $d_{ij}$ is the distance $\|y_i - y_j\|$ between the images in the map, or feature space $\mathbb{R}^q$. These distance measures are generally Euclidean but need not strictly be so. Finally $\left[ \frac{d_{ij}^* - d_{ij}}{d_{ij}} \right]$ is a measure of the deviation between corresponding distances, and the Sammon STRESS.

Although Sammon Mapping was originally applied in the engineering field and was designed as a computational tool for data structure analysis and for visualisation, its use is still popular in many domains including the visualisation of gene expression data (Ewing and Cherry; 2001).

The simple and intuitive nature of Sammon Mapping would make this technique a good candidate for the data reduction but it does suffer from some important drawbacks which have
been identified by Tipping (1996). The most significant is that the mapping is generated through an iterative process that may result in the process being caught in sub-optimal local minima. When Sammon Mapping is performed on large data sets, the computational requirements scale square with the number of data points making it unfeasible to use it on only small data sets with low dimensionality. And finally, the map is generated as a ‘look-up’ table. There is no easy way to project new data without re-generating the entire map when new data points are added. Although this issue has been addressed by Mao and Jain (1995), it still does not occur in real time. Their approach still requires a Multi Layer Perceptron (MLP) to be trained on the Sammon Mapping which takes time and still limits the real-time visualisation of the rule-nodes.

In this thesis some rule sets sets have 768 attributes to them. More importantly, our proposed learning mechanism is an on-line learning system, new data examples can be presented at any time and to regenerate a new visualisation of the rule set using Sammon Mapping would take more time. To generate a visualisation using this method would not seem appropriate for real time visualisation of the data sets.

2.6.3 Principal Component Analysis

PCA is a canonical and widely used method for dimensionality reduction of multivariate data. Applications include exploratory data analysis (Berkhin; 2002), clustering of gene expression data (Yeung and Ruzzo; 2001), and face recognition (Turk and Pentland; 1991).

PCA determines an optimal linear transformation

\[ y = Wx \]  

(2.12)

of a real-values \( n \)-dimension random data pattern \( x \) into another \( m \)-dimensional \((m \leq n)\) transformed vector \( y \). The \( m \times n \) fixed linear transformation matrix \( W \) is designed optimal by exploring statistical correlations among elements of the original patterns and finding possibly reduced compact data representation retaining maximum nonredundant and uncorrelated intrinsic information of the original data. PCA can be effectively used for dimensionality reduction. Instead of a whole \( n \)-dimensional original data pattern \( x \) one can form the \( m \)-dimensional \((m \leq n)\)
feature vector \( y = [y_1, y_2, \ldots, y_m]^T \) containing only the first \( m \) most dominant eigenvalues \( \lambda_1, \lambda_2, \ldots, \lambda_m \) of the original data pattern covariance matrix. From this we can infer that the \( m \) principal components \( y_i \) are the most expressive features of the data set. Using the first two of these components we can then visualise a large and high dimensional data set in two-dimensions making it a suitable candidate for rule set visualisation.

### 2.7 Content-based image retrieval

Advancements in processing technology have led to more types of data being supported by applications. Included in this group are image and video data which requires the development of new methods to store, query, and retrieve this data based on their content. Such Content Based Image Retrieval (CBIR) systems (Park et al.; 2002; Newsam et al.; 2001) use previously developed feature extraction and pattern recognition methods to determine the similarity in the content of the query and contents of the database.

Because image and video databases differ in many aspects from traditional relational databases containing textual information, the main focus of previous research in CBIR has been to develop methods that can transform the image into a set of features that are used to generate indices to store the image in the database. Satisfying a user query involves extracting the necessary features from this image data and then have the CBIR system match it against the same type of features to the indexed elements of the database.

With the number of methods developed for querying images that employ the use of specific features transformed from image data, more attention needs to be paid on adjusting the parameters for these methods to ensure their success. Previous object recognition techniques relied on the expected geometry, shape, texture, and colour of the object are known beforehand. But due to the fact that the images used in this thesis are of varying quality, size, colour, and complexity, the nature of these images prevents us from a fixed and prior decided set of features. Several approaches have been proposed to address this issue and Antani et al. (2002) provides a comprehensive survey of the many different techniques used in CBIR.

In this thesis we have used wavelet coefficients as the indices for indexing an image data base
from the horticultural domain for the purpose of CBIR. This provides the starting point for the experiments conducted in Chapter in 5 as it has influenced our subsequent use of the wavelet coefficients as input to a fuzzy neural-network based classifier for image and signal recognition.

2.8 Summary and Conclusion

Beginning with simple techniques for data normalisation and progressing to the topic of CBIR, this chapter has investigated the individual building-blocks that constitute the data and image analysis techniques used in this thesis.

Initially simple linear and non-linear data normalisation techniques were presented as most of the data sets required some translation before being processed by the classification models.

Next, the main focus of the chapter was a discussion on the task of feature extraction where wavelets were proposed as an effective solution to this problem. Out of the many types of wavelets that could have been used, we argued for the use of Daubechies wavelets as the exhibited properties that suited the analysis of the image and signal data used in the experiments reported in the latter chapters.

A section on image transformations was included to justify the selection of the rescaling method used on the image data sets. This is important issue to address as the majority of detail contained in the image should still be present. Bicubic interpolation was proposed as a solution to this problem as it provided the best method of resizing an image object with little loss of definition.

The issue of dimension reduction techniques were presented as PCA is used as a means of visualising the data sets throughout Chapter 5 and Chapter 6 because of the high dimensionality of the rule sets. And as image retrieval retrieval is relevant to the work in Chapter 5, a brief review of CBIR was presented.
Chapter 3

Generic Techniques for Connectionist Systems: A Review

3.1 Introduction

Chapter 2 covered generic methods and techniques for the normalisation, transformation, feature extraction, and dimensionality reduction of specific data sets. Once these features have been selected, it is then necessary to generate a model from this data that allows for subsequent classification of new instances from the same data domain that the original data set was sourced from. More importantly, knowledge should be able to be extracted from this model to both support the decisions made by the user and also to justify the model’s output.

Systems that learn from data are firmly grounded in the area of Machine Learning (ML) (Michie et al.; 1994; Mitchell; 1997). However selecting an appropriate ML model from the plethora of those available is not an easy task. Criteria such as the amount of available data, the dimensionality of the data set, and the characteristics of the extracted feature set from the data set all contribute to this choice.

In this chapter we describe two distinct groups of ML techniques for the modelling of horticultural data. One of these techniques, neural networks, firmly positions itself in the area of Connectionist Systems (CS). The other, statistically-based classifiers, offers an alternative approach to this problem. The Support Vector Machine (SVM) is an example of the current state-of-the art in statistically-based classifiers so this method is covered in detail.
However deriving a ML model from horticultural data is only half the solution to the problem. This thesis also concerns itself with knowledge representation from the learning model. There are many ways in which knowledge can be represented and in this thesis we concentrate on rules since they offer an elegant and human-readable form. Therefore Section 3.4 covers a brief account of how rules model knowledge and their application to rule-based or expert systems.

And because growing produce is a process built on real-world knowledge, an effective means of reasoning about within an ML model is required. Therefore we introduce the discipline of fuzzy logic and fuzzy reasoning (Zadeh; 1965) to create a richer means of representing this real-world knowledge when we generate our ML model.

Finally, in this chapter we draw some conclusions about the disadvantages of applying neural networks and fuzzy logic in isolation from each other. By selecting the advantages of each paradigm, we can combine these into a more powerful ML system. Neuro-Fuzzy Systems (NFS) are the result and provide a better learning mechanism for the acquisition of knowledge from data driven ML systems and in Section 3.6 we cover this fusion of techniques.

### 3.2 Neural Networks

Creating appropriate models to represent the knowledge required to facilitate a task in the horticultural area is a complex problem. There are many different sources of information that need to be fused together and then modelled using some method or technique. As the underlying nature of any horticultural process is dynamic, it would be logical to choose a paradigm that learns over time to model these dynamics. One potential candidate would be an artificial neural network based on the connectionist paradigm.

#### 3.2.1 The architecture of a neural network

The basic building elements of artificial neural networks, influenced by many disciplines including mathematics, psychology, and neuroscience, emerged around the 1940s with the combined development on theories of human learning (Hebb; 1949) and the calculus of the McCulloch and
Pitts (1943) to form Rosenblatt’s Perceptron (Rosenblatt; 1958) in the latter part of the 1950s. Pudi (2003) provides a good definition of a perceptron and Figure 3.1 depicts this architecture.

- The inputs are denoted as \( x_1, x_2, \ldots, x_n \) and are normally represented either as real numbers or Boolean values; the choice of which are governed by the type of problem.
- The output \( y \) is represented as a Boolean value.
- The weights of the edges \( w_1, w_2, \ldots, w_n \) are represented as real values.
- The threshold \( T \) takes values of type real.

\[
\text{Figure 3.1: Architecture of a perceptron adapted and modified from Pudi (2003)}
\]

Since the output of a perceptron is either 0 or 1, calculating this value is governed by the net input \( w_1 x_1 + w_2 x_2 + \ldots + w_n x_n \). When this net value exceeds the threshold \( T \) then the output is 1 otherwise the output is 0.

In the case of the perceptron, it had an input layer of input units where data vectors are fed into the perceptron, and an output unit that contains the output of the perceptron.

Single-layered neural networks alternatively known as first-order networks were then constructed from groups of perceptrons in order to solve simple problems whose architecture is depicted in Figure 3.2.
However, the size of the network and time required for its training both increase significantly with order. For example a second-order neural (two-layer) network with $n_o$ output units which is fed by $n_i$ input variables consists of $n_o \times n_i^2$ synaptic weights compared to $n_o \times n_i$ synaptic weights of the corresponding first-order neural network.

Early research in neural networks focused on simple neural networks with one layer on linear or nonlinear output units. Unfortunately this limited the applications of these network architectures to elementary problems. The current network architecture needed to become more complicated in order to solve more complex real-world problems.

This architecture was then extended to create a multi-layer feed-forward network also known as the Multi-Layer Perceptron (MLP) (Rumelhart et al.; 1986b). This network is composed of several layers of simple processing units. The state of a unit at any given time is represented by its activation, which is a real-valued number, typically in the range $[0, 1]$ or in the range $[-1, 1]$. The input layer of a feed-forward contains units whose activations represent values for the features of the problem domain in which the network is being applied. In most cases, a real-valued feature is represented by a single input unit, and a discrete feature with $n$ possible values is represented by $n$ input units. The units of the output layer of a MLP represent the decisions.
made by the network. The units of a feed-forward network are related by weighted connections. To address the work of Minsky and Papert (1969) who had criticised previous perceptron-based neural networks since they could only learn linearly separable classes, the introduction of one or more hidden layers in the MLP and a new learning algorithm resulted in the activation functions in the hidden nodes becoming non-linear and able to learn non-linearly separable classes.

Figure 3.3: A structure of a multi-layer perceptron with three layers

Computation in a feed-forward network proceeds by setting the activation values of the input units to represent a particular instance in the problem domain. The activation of the inputs feeds forward through the weighted connections to the units at the hidden layers and then to units at the output layer. The answer provided by the network is determined by the resultant activations on the output units.

For a particular example presented to the feed-forward network, the net input to a unit in a hidden or output layer is given by:

\[
net_i = \sum_j w_{ij} a_j + \theta_i
\]  

(3.1)

where the weight from unit \(j\) to unit \(i\) is denoted as \(w_{ij}\), \(a_j\) is the activation of unit \(j\) in response to the example and \(\theta_i\) is the bias for unit \(i\). The bias of a unit, which is an adjustable parameter can be regarded as the unit’s predisposition to have a high (or possibly low) activation before it received any activation from other units it is connected to. The activation of a hidden or an output
unit is determined by passing its net input through a transfer function or activation function. One commonly used transfer function is the logistic function.

\[ a_i = \frac{1}{1 + e^{-net_i}} \]

This function has the effect of “squashing” the unit’s net input to an activation value in the range [0, 1]. A similar transfer function is the hyperbolic tangent function which squashes a unit’s net input into an activation value in the range [-1, 1]:

\[ a_i = \frac{e^{net_i} - e^{-net_i}}{e^{net_i} + e^{-net_i}} \]

Both the logistic function and the hyperbolic tangent function, depicted in Figure 3.4 are examples of sigmoidal functions and are continuous approximations of a threshold function.

![Figure 3.4: Two examples of sigmoidal transfer functions used in feedforward neural networks](image)

3.2.2 Learning in Neural Networks

The underlying ability for a neural network is to learn data and then generalise to unseen or new data is the most attractive characteristic of a neural network. Learning enables the modification of behaviour in response to the environment. A neural network is trained so that an application of a set \( X \) of input vectors produces the desired set of output vectors \( Y \). The learning ability
of a neural network is achieved by applying a learning algorithm and this function is normally classified into two main groups:

- **Supervised:** The training examples are comprised of input vectors $X$ and the desired output vectors $Y$. Training is performed until the neural network “learns” to associate each input vector $X$ to its corresponding and desired output vectors $Y$. The most popular learning algorithm is the backpropagation learning algorithm.

- **Unsupervised:** Only input vectors $X$ are supplied and the neural network learn some internal features of the whole set of all the input vectors presented to it. Such methods are touted as more biologically plausible algorithms.

The backpropagation learning algorithm

The addition of the non-linear or “hidden” layer could potentially allow the MLP to learn linearly non-separable classes but it wasn’t until the development of the backpropagation of errors supervised learning algorithm or just backpropagation (Rumelhart et al.; 1986a; Werbos; 1988, 1990) that this particular neural network was put into practice. Backpropagation learning in a neural network involved modifying the weights and biases in order to minimise a cost function. The cost function always includes an error term that is a measure of how close the networks’ predictions are to the class labels for the examples in the training set. An appropriate cost function for classification problems is the cross-entropy function (Hinton; 1989):

$$C = - \sum_i \sum_j [t_j \ln(a_j) + (1 - t_j) \ln(1 - a_j)]$$

Here $i$ ranges over the examples in the training set, $j$ ranges over the output units of the network, $t_j$ is the target value for the $j$th output unit for a given example, and $a_j$ is the activation of the $j$th output unit in response to the example. The target value for an output unit is the activation value that it should have for a given example.

In almost all neural network methods, the function implemented by the network is continuous and differentiable. Therefore, the cost function can be minimised by calculating its partial derivatives with respect to each of the network’s parameters, and making changes to the parameters as
follows:

\[ \Delta \vec{w} \propto -\nabla_w C \]

where \( \vec{w} \) represents the vector of weights and biases in the network and \( C \) is the cost function.

Various optimisation algorithms can be used to minimise the cost function, the most popular of them being on-line backpropagation. This method can be thought of as a stochastic form of gradient descent in that weight changes do not follow the gradient of the cost function for the entire training set, but instead they follow the gradient for a single training example (White; 1989a,b). In order to avoid large oscillations, these weight changes normally incorporate a momentum term (Rumelhart et al.; 1986b), which is a time-decaying average of previous weight changes. Other optimisation methods can be used to minimise the cost function. Standard gradient descent augmented with a momentum term is sometimes used, as is the conjugate-gradient method (Kramer and Sangiovanni-Vincentelli; 1989).

More often than not, network training is stopped before a local minimum in the cost function is reached. The motivation underlying this technique of early stopping is that over-fitting may occur if the network is trained to fit the training data too closely.

**Hebbian Learning**

The other set of learning algorithms relates to those that implement unsupervised learning. Here many of the initial unsupervised learning algorithms have their foundations based on a biologically inspired learning rule proposed by (Hebb; 1949). The presentation of the Hebb rule begins the single unit shown in Figure 3.1. Using the assumption the unit is linear, the output of this unit is given in terms of the input vector \( \mathbf{x}^* = [x_1, x_2, \ldots, x_{ni}] \) and the vector of the synaptic weights \( \mathbf{w}^* = [w_1, w_2, \ldots, w_{ni}] \) by

\[
\hat{y} = \bar{y} = \mathbf{w}^* \mathbf{x} = \mathbf{x}^* \mathbf{w}
\]

According to the Hebb rule, the change of each synaptic weight \( w_i \) is proportional to the product of its input \( x_i \) and its output \( \hat{y} \). To study the behaviour of the system in time, a variable \( k \) is introduced. If the weight vector \( \mathbf{w} = \mathbf{w}_{k-1} \) maps the input \( \mathbf{x} = \mathbf{x}_k \) into the output \( \hat{y} = \hat{y}_k \), the
Hebb rule provides
\[ \Delta w_{i,k-1} = w_{i,k} - w_{i,k-1} = \alpha \hat{y}_k x_{i,k} \] (3.3)

The update equation provided by the Hebb rule may also be obtained by applying the gradient descent method to maximise \( \hat{y}^2 \) or, equivalently to minimise \( J(w) = -\frac{1}{2} \hat{y}^2 \). Therefore,
\[ \Delta w_{k-1} = -\alpha \frac{\partial J(w)}{\partial w} \bigg|_{w=w_{k-1}} = \alpha \hat{y}_k x_k \] (3.4)

To address the inherent instability of the fixed points for the update equation, work conducted by Oja (1989), Linsker (1988), and Sanger (1989) have proposed improvements and variations to the original Hebb learning rule. Recent applications using the Hebbian learning for invariant object recognition have been proposed by Wang (2001) and Wersing and Körner (2003).

**Competitive Learning**

An alternative view of unsupervised learning is the concept that the system adapts itself to the environment with no involvement of an external teacher. The ability of a system to self-organise is the basis of competitive learning algorithms. Here the competition between neurons is formulated through learning rules. In a competitive learning scheme all the inputs nodes receive identical input but compete to get the maximum share of resources.

The Self-Organising Feature Map (SOFM) or SOM (Kohonen; 1990) is the most representative neural system that uses the concepts of competitive learning. Unlike the previously described feed-forward architecture, the SOM is a two-dimensional neural network, where each unit is attuned to the input patterns through an unsupervised learning process.

The two-dimensional map consists of cells which form a hexagonal or rectangular grid. The map receives as input the vector \( x^* = [x_1, x_2, \ldots, x_n] \) which is connected in parallel to all the cells. A weight vector is assigned to each one of the cells. Let \( \mathbf{m}_i^* = [m_{i1}, m_{i2}, \ldots, m_{in}] \) be the vector which corresponds to the \( i^{th} \) cell of the map. The weights \( m_{ij} \) which form each weight vector \( \mathbf{m}_i \) are the adaptive parameters of the system. Each weight vector may be interpreted as an “image” that will be compared to the input \( x^* = [x_1, x_2, \ldots, x_n] \). The similarity between the input vector \( x \) and each of the weight vectors is usually measured by their Euclidean distance.
Kohonen proposed an adaptive process which determines the weight vector of each cell in such a way that every cell becomes sensitive to a particular subset of input signals in a regular order which results in a spatial ordering of the output vectors. During this ordering process, the cells of the map are not updated independently but as topologically related subsets. The selection of the subset of cells to be updated at each learning step required the definition of a centre cell whose weight vector $m_c$ is the closest to the input vector $x$ normally specified by its Euclidean distance.

The subset of cells whose weight vectors are updated at a given learning step is determined by defining a neighbourhood $N_c(t)$ around the centre cell. The weight vectors of the cell can be updated according to the following rule:

$$m_i(t + 1) = \begin{cases} m_i(t) + \alpha(t) \left[ x(t) - m_i(t) \right] & \text{if } i \in N_c(t) \\ m_i(t) & \text{if } i \notin N_c(t) \end{cases} \quad (3.5)$$

where $\alpha(t)$ is the adaptation gain which satisfies $0 < \alpha(t) < 1$ and $c$ is the label of centre cell of the neighbourhood $N$.

Since its inception there have been many variants of the original SOM learning algorithm including those proposed by Blackmore and Miikkulainen (1993), Ahrns et al. (1995), and Vesanto and Alhoniemi (2000).

This method of competitive learning using the SOM has been widely applied to many problem domains including detecting skin in images of people (Brown et al.; 2001), analysis and visualisation of gene expression data (Nikkilä et al.; 2002), and remote sensing image analysis (Villmann et al.; 2003).

In a similar vein Grossberg (1980) and Carpenter and Grossberg (1987), influenced by principles derived from biological systems, proposed a self-organising neural network architecture known as Adaptive Resonance Theory (ART). Their intention was to solve the stability-plasticity dilemma where autonomous system have to adapt to new events or learn new events whilst retaining its old knowledge.

Learning in ART is based on using a layer that is a competitive network of neurons operating on the “winner-take-all” principle. The objective is that top-down expectations fuse with bottom-
up information to prevent loss of already learnt knowledge and include new knowledge in a globally self-consistent fashion. ART provides a computational mechanism by providing an architecture to satisfy these objectives.

The general architecture of the ART model is shown in Figure 3.5. Let \((x_1, \ldots, x_n)\) represent the input pattern, \(u_{ij}\) represent the bottom-up weights from \(F_1\) to \(F_2\), and \(d_{ij}\) represent the top-down weights.

![Figure 3.5: Architecture of the ART model](image)

The input pattern \((x_1, \ldots, x_n)\) activates layer \(F_1\). Corresponding to this input pattern, the winning node in layer \(F_2\) of \(y_j\) receives the largest total signal from \(F_1\). \(F_2\) then in turn sends its learned expectation to \(F_1\) and top-down expectations and input patterns are matched. A mismatch turns off the node \(y_j\) and the process is repeated without \(y_j\). This process is repeated until one of the following three conditions occurs.

- An output node (\(F_2\)) with an approximate match to the input pattern is found.
- A new node with no previously assigned weight is found.
- The capacity of the network is exhausted.

An approximate match between the input pattern and top-down expectation results in a consensus or fusion between top-down expectation and input and both the top-down and the bottom-up
weights are updated. ART maintains the plasticity required to learn new patterns, while preventing the modifications of patterns that been previously learned.

**Reinforcement Learning**

A combination of supervised learning with the field of dynamic programming, whose application was traditionally used to solve problems in optimisation and control is known as Reinforcement Learning (RL) (Bakker; 2002). Here the model is goal driven where learning is conducted through a series of trial-and-error interactions with a dynamic environment (Kaelbling et al.; 1996). The interactions between the RL model and the environment are normally manifested as sensor readings, symbolic descriptions, or possible “mental” situations (Harmon and Harmon; 1996).

The most elementary form of RL learning algorithms employ mechanisms which increase the RL system’s experience interacting with the environment, and improving the RL system’s decision-making policy over time. One definition of a policy is “… a mapping from states to actions, or to probability distributions over actions.” (Sutton; 1999, p. 3). This policy is represented in a manner that allows for the fast generation of appropriate responses where the RL system has encountered an unforeseen state (Sutton; 1999).

The simplest RL algorithm uses a state table to manage this task and is known as tabular 1-step Q-learning (Watkins and Dayan; 1992). Here a table $Q$ has entries $Q(s, a)$, where $s$ denotes a state and $a$ denotes the action. Updating an element of the table is performed by the algorithm

$$ Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[ r_{t+1} + \gamma \max_a Q(s_{t+1}, a_t) - Q(s_t, a_t) \right] $$

(3.6)

where $r_{t+1}$ is the reward for executing an action $a_t$ when the transition from $s_t; s_{t+1}$, and $\alpha$ is a positive step-size parameter (Sutton; 1999). Other methods have also represented $Q$ as a function within a neural network in which generalisation between states can occur resulting in faster learning time (Humphrys; 1997).
Learning modes in neural networks

One other distinct aspect of learning in neural networks relates to the learning mode. In on-line (incremental) learning weight changes are applied to the network after each training pattern. In offline (batch) learning the weight changes are accumulated for all patterns in the training file and the sum of all changes is applied after one full cycle through the training pattern file.

This choice of learning mode governs the effectiveness of learning in neural networks. For applications that are required to operate in real-time, on-line learning is the most appropriate choice and a suitable neural network architecture found to implement this. In Section 4.2 of Chapter 4 we explore this issue in more detail.

To date many articles have been published on their impressive performance in certain problem domains compared to their counterpart models derived mainly from rigorously defined statistical theory (Blue et al.; 1994). For example some of these models were more biologically plausible and allowed researchers to understand brain function and behaviour (Cruces et al.; 2000) while others were developed to serve a specific purpose such as mammography classification (Hojjatoleslami et al.; 1997) or prediction of noisy time series data (Giles et al.; 2001).

However, as opposed to the well defined mathematical models, two major criticisms directed at a majority of these connectionist architectures have been identified:

1. **The lack of many of the conventional connectionist structures to learn and to model real-time data.** Most of the learning algorithms that underlie the good performance of the conventional neural network structures require that the data to be passed through these models many times before the neural network has satisfactorily learnt the input data and then be able to then generalise to new data that is being presented to the network. Although there have been attempts at creating real-time learning algorithms such as motor controllers for robots (van der Smagt and Krose; 1991), these approaches are normally relegated to a specific domain and are unable to be generalised to other problems.

2. **Encoding and extracting high-level or symbolic knowledge within these connectionist-based structures.** Although neural networks have shown very good performance in many
application domains, one of their main drawbacks lies in their inability to provide an explanation for the underlying reasoning mechanisms. There was no method of interrogating the learnt artificial neural network to find out how it functioned and therefore such a structure could not easily be validated or new knowledge derived from them. Efforts by Shavlik (1992); Gallant (1993); Towell and Shavlik (1993); Cristea et al. (1997); Kasabov, Kim, Watts and Gray (1996); Craven and Shavlik (1997); Fu (1999); Garcez et al. (2001); Andrews and Geva (2002); Remm and Alexandre (2002); Castellano et al. (2003); Iyatomi and Hagiwara (2003), and Jin and Sendhoff (2003) have attempted to create a mapping between the knowledge contained in the connectionist structure and representing it in a symbolic way. We explore this issue further in Chapter 4.

3.3 Support Vector Machines

Traditionally in supervised learning there is normally a set of examples of input vectors \( \{x_n\}_{n=1}^N \) along with corresponding targets \( \{t_n\}_{n=1}^N \), the latter of which might be real values for a regression problem or class labels for a classification problem. By using this “training set” a model can be learnt to make where the presentation of previously unseen values of \( x \) can result in more accurate predictions of \( t \). The challenge in developing this model is to avoid the problem of ‘over-fitting’ of the training set, which is either characterised by the presence of noise in regression problems or class overlap in classification problems, thus creating a learning model that has high generalisation.

Cortes and Vapnik (1995) addressed this problem by proposing a new type of learning machine whose grounding is in the area of statistical learning theory. The Support Vector Machine (SVM) maps the input vectors into some high dimensional feature space \( Z \) through some nonlinear mapping chosen apriori. In this space a linear decision surface is constructed with special properties that ensure high generalisation ability of the network.

The SVM is a very successful approach to supervised learning that differs from the way in which a neural network learns. It makes predictions based on a function of the form as in
Equation 3.7 where \( \{w_n\} \) are the model “weights” and \( K(\cdot, \cdot) \) is the kernel function.

\[
y(x) = \sum_{n=1}^{N} w_n K(x, x_n) + w_0
\]  

(3.7)

There are many different kernel functions that could be employed in the SVM. Table 3.1 contains four common kernel functions.

<table>
<thead>
<tr>
<th>Kernel Function</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gaussian Radial Basis Function (RBF)</td>
<td>( k(x, y) = \exp \left( -\frac{|x-y|^2}{c} \right) )</td>
</tr>
<tr>
<td>Polynomial</td>
<td>( ((x \cdot y) + \theta)^d )</td>
</tr>
<tr>
<td>Sigmoidal</td>
<td>( \tanh(\kappa(x \cdot y) + \theta) )</td>
</tr>
<tr>
<td>Inverse multiquadratic</td>
<td>( \frac{1}{\sqrt{|x-y|^2 + c^2}} )</td>
</tr>
</tbody>
</table>

For classification problems, the target function of the SVM, bounded by the feature space defined by the kernel function, attempts performs a simultaneous task of maximising the ‘distance’ between two classes within the feature space whilst attempting to reduce the number of errors made on the training set. Good generalisation of the model can be obtained as this method avoids the problem of over-fitting. Tipping (1999) identified that the reason for this good generalisation was that a sparse model was created dependent only on a subset of kernel functions which accounted for training examples \( x_n \) that may either appear on the margin between two classes or possibly be located near the boundary of a different class.

To achieve this the SVM nonlinearly projects the linearly non-separable input data into a high dimensional feature space where there exists a linear hyperplane. According to Moghaddam and Yang (2000) this method may not always result in a linear separable solution. It, however, can normally be achieved if the process of computing the best hyperplane is regarded as a constrained optimisation problem and solved using quadratic programming techniques.

The process of classification by a SVM of an unknown pattern occurs by first transforming the pattern into some high-dimensional feature space. An optimal hyperplane constructed in this feature space then determines the output. Figure 3.6 realises this process as two-layer perceptron.
In the areas of sex classification on thumbnail images (Moghaddam and Yang; 2001), clustering of the iris data set (Ben-Hur et al.; 2001), and more accurate regression models (Tipping; 1999), state-of-the-art results had been reported. However there are some problems with SVM. For classification problems the SVM generates a binary decision and for regression problems a point estimate is produced. There is no method to determine the probability of what the SVM predicts is the correct output. Tipping (1999) identified that the uncertainty of the prediction could be estimated if there was a conditional distribution $p(t|x)$ where $x$ is the data instance and $t$ is the target class.

Furthermore the original SVM model was designed to separate two classes. In other words there wasn’t originally a SVM that was designed for multiple class separation. Recently Vazquez and Walter (2003) have extended the learning algorithm to include Multi-Input Multi-Output (MIMO) systems as well. But more importantly the SVM model generally requires large data sets when training. When such data is not readily available, the generalisation ability of the SVM is decreased. This is an important issue to highlight due to the size of the data sets used in this thesis.
3.4 Rule-based Systems

In any business, as it functions, staff involved at the management level need to make timely decisions in order to benefit the overall operations of the enterprise. Growing quality pipfruit for the export market is no different and is a task that requires important decisions to be made over the growing season. Until the IFP programme (Walker et al.; 1997) was implemented in 1997, this task was unstructured and based on ad-hoc decisions made by the orchardist. Today growing pipfruit is a more regulated process that requires the orchardist to consult a manual which governs their choice action to take based on the state of the orchard. This more structured approach makes the whole process a good candidate for the use of rule-based systems to represent and manipulate the knowledge used to grow the produce.

Rule-Based Systems (RBS) are an extension of propositional logic and the transformation rules or productions that are applied to an initial string of propositions to create a new string of propositions. These strings of propositions are normally in the form \( IF \) (condition - compound proposition) \( THEN \) (conclusion - proposition) and are referred to as production rules.

The (condition - compound proposition) is known as the antecedent part that describes the facts or conditions that must exist for the rule to fire whilst the (conclusion - proposition) is known as the consequent part and describes the action taken as the result of the rule being true. These rules are normally written in the form \( IF \) (conditions) \( THEN \) (action).

Production rules alone are not enough for a rule-based system. They are normally combined with a memory for the initial facts of the problem to be solved and an inference mechanism matches facts from working memory with the left-hand side of the productions. There are two inference mechanisms that can be applied. Forward chaining inference starts with a set of initial facts or data thus the output of the system is based on the set of initial facts presented to it. Backward-chaining is a goal driven mechanism. The inference process starts after a goal is identified. Then a search for a production rule which has this goal in its antecedent part is performed, the data or facts that satisfy all the conditions for this rule are sought in the memory of initial facts.

There have been many examples at these inference mechanisms including the OPS5 engine.
(Forgy; 1981) to more recent methods such as the Protege-2000 (Grosso et al.; 1999).

RBS have also formed the basis of Expert Systems (ES) (Buchanan and Shortliffe; 1984). Here the emphasis is on providing expertise, similar to those of experts in a restricted application area through the use of knowledge-based systems. The Knowledge-Base itself is normally stored as production rules. The current facts or past data are contained in a Database Module that is the working memory of the production languages. In some ES architectures this module, containing the past data can be used as a source of knowledge in addition to the knowledge base. With regard to the Inference engine of an ES, it can either adopt forward- or backward-chaining mechanisms. In some cases a combination of forward and backward chaining has been used by Fedra and Winkelbauer (2002).

More importantly, what sets ES apart from RBS is the addition of an Explanation Module to trace the execution of the ES and accumulate information about the course of the reasoning process. This allows the system to explain its behaviour about HOW or WHY the conclusion was inferred. Similarly, with the User Interface Module, the ES can communicate with the environment to interact with the user in an user friendly yet sophisticated way. This User Interface Module has evolved from textual interaction using natural language queries, to more sophisticated Graphical User Interfaces, and even speech recognition and synthesis to process the user’s commands.

Over the course of the past 20 years, these expert systems have been applied to provide decision-support in many areas of business and science. For example PEARL (DeJesus and Callan; 1985) and DAS/Logic (Birmingham and Kim; 1985) were developed as design assistants for Computer Aided Architecture (CAD) design. IDDEX (Iwai et al.; 1994) was created to support the design of new drugs, and Bridger (Reich; 1995) to aid in the design of bridges. In addition to these examples that have been applied to a specific problem, expert system shells such as the C Language Inference Production System (CLIPS) (see (Kasabov; 1996a) for examples of the use of CLIPS) or the Java Expert System Shell (JESS) (Friedman-Hill; 1997) were developed offering all the functionality of a standard forward-chaining expert system where the production-rules can represent many different problem domains.

However the traditional architecture of conventional rule-based systems had a fixed structure
of modules and automatic rule adaptation could not easily occur during the rule-based systems operation. Although they were successful in very specific areas, this particular architecture resulted in little or no flexibility for the system to adapt to the changes required by the user or the environment in which the rule-based system operated. This imposed rigidity of conventional rule-based systems is one of the major disadvantages of maintaining the rules which are at the core of its knowledge. Furthermore, traditional expert systems have only minor abilities for the automatic acquisition of new knowledge that may be the result of existing inferences made by the rule-based system.

This is an extremely important issue when growing fruit using the IFP programme. Modifications to the IFP manual are made at the end of the growing season based on a combination of the feedback on how good the quality of the apples that were produced, and the results of field trials undertaken by HortResearch using the current IFP programme manual. Yet, in order to provide better advice to the grower over the course of the season, an expert system should be able to adapt the parameters of its rules in real-time order to better model the dynamics of the orchard. This is a function that conventional expert systems have minimal support for.

### 3.5 Fuzzy Logic

The real world does not often deal in absolutes and this is certainly the case when working with horticultural data. There may be missing, noisy, or erroneous data collected that then affects the development of the produce. In addition, the knowledge to be modelled when growing the produce is based on more real-world concepts often in the shades of grey. These problems make fuzzy logic (Zadeh; 1965) a good candidate for representing and modelling the information and knowledge decision-making processes in horticulture. Fuzzy logic dispenses with the need for absolutes, instead representing problems that are found in the real world. Fuzzy logic can thus be a powerful method for building knowledge-based systems that must not only deal with the real world, but also be comprehensible to users.
3.5.1 Definition of a fuzzy set

Consider a collect of objects $X$ where $x$ is a generic element of $X$. Classical set theory would define this collection as a set $A$ where $A \subseteq X$ and our element or object $x \in X$ such that each element $(x)$ either belongs to the set $A$ or not. To indicate the absence or presence of the element $x$ in $X$ a classical set $A$ can represented as a set of ordered pairs $(x,0)$ to indicate $x \notin X$ and $(x,1)$ to indicate $x \in X$ through the use of a characteristic function (Bezdek; 1993).

Fuzzy sets on the other hand represent the degree to which an element belongs to a set. Therefore the membership function of a fuzzy set can have values that range between 0 and 1 to indicate the degree of membership an element has to the set. A more formal definition of a fuzzy set is as follows:

"Definition 1. Fuzzy sets and membership functions. If $X$ is a collection of objects denoted generically by $x$, then a fuzzy set $A$ in $X$ is defined as a set of ordered pairs $A = \{(x, \mu_A(x)) | x \in X\}$ where $\mu_A(x)$ is called the membership function (or MF for short) for the fuzzy set $A$. The MF maps each element of $X$ to a membership grade (or membership value) between 0 and 1 (included)." (Bezdek; 1993, pp. 4)

Fuzzy sets extend classical or crisp sets so that the values of the characteristic function are allowed to be between 0 and 1. By restricting the membership function to only take values of 0 or 1 reduces a fuzzy set to a classical set.

To complete the definition of a fuzzy set, $X$ is defined as the universe of discourse (Jang; 2001) but more commonly known as the universe which can either be composed of discrete objects or represent a continuous space.

In this thesis we consider the universe as a continuous space. We can illustrate this point by considering the how we define the modelling of pest numbers. For example, assume that $X = \text{"pest numbers"}$ where $X$ is the number of pests found within an orchard. Fuzzy sets “low”, “medium”, and “high” are then characterised by MFs that are defined to represent these different values. “pest numbers” then becomes a linguistic variable that can be represented by the linguistic values “low”, “medium”, and “high”. Therefore if “pest numbers” adopts the value of “high” then the expression becomes “pest numbers are high”. Figure 3.7 demonstrates how these three MFs are able to fully cover the universe of $X$. 

51
Jang (2001) describes another example of how fuzzy sets can represent a linguistic concept such as tall and illustrates how classical sets have problems representing the same concept. To make this example more relevant to apple orchards we have modified and applied this example to the problem of modelling a set of tall apple trees.

Consider the universe of discourse to be all potential heights where the height ranges from 3 feet to 9 feet. The word “tall” then corresponds to a curve that indicates the degree to which any tree is tall. In classical set theory, if the boundary for the set of tall trees is set at six feet, would then infer that all trees above that height are tall trees. Therefore a tree with a height of 5.9 feet would be considered short and a tree at 6.1 feet would be considered tall. Such a small margin of difference does not clearly distinguish between a short and a tall tree. The solution is to model the set of tall trees using a smoothly varying curve that ranges from not-tall to tall with the output axis denoting the membership value of the height to the set of tall trees as in Figure 3.8. In this way two trees can be tall to some degree but one is going to be less tall than the other.

Figure 3.7: Typical MF’s of linguistic values “low”, “medium”, and “high”. Adapted from Bezdek (1993, p. 4)
3.5.2 Determining the Membership Functions

Kantrowitz et al. (2003) break down methods for determining MFs into the four following categories:

Firstly one could use subjective evaluation and elicitation. Since the intention of fuzzy sets is to model a human’s natural form of reasoning, simple to more sophisticated elicitation procedures can be applied to determine this. With the simplest method, experts in the problem area could define a host of membership curves to model the problem. To constrain the number of membership curves produced, the expert could be provided with a set of candidate curves from which the would select the most appropriate one. More complex methods may involve the users being tested using psychological methods.

Although there is a huge (infinite) array of possible membership function forms, most are actually sampled from a much smaller set of different curves. Simple forms of fuzzy numbers (Tran and Duckstein; 2002) are described as an attractive method to simplify this problem where one chooses the middle value and the gradient on either side of it.

Converted frequencies or probabilities is also an option. It is possible to extract information from probability curves or frequency histograms to construct a membership function but
each method has its own advantages and disadvantages. Bilgic and Turkseven (1999) investigate a number of these. It, however, should always be noted that membership functions are NOT (necessarily) probabilities so these conversions are not easily interchangeable.

A fourth and final option is to resort to physical measurement which has been used in many applications of fuzzy logic. The membership functions are generated through a process of transforming the measured values into a set of distinct membership grades as it is not easily possible to measure the membership grade directly from the measured values.

3.5.3 Fuzzification

Fuzzification is where the membership functions defined on the input variables are applied to their actual crisp values, to determine the degree of truth for each rule premise in a fuzzy expert system.

For example if we have fuzzy rules of the type IF $x_1$ is $A_1$ and $x_2$ is $A_2$ THEN $y$ is $B$ then the process of fuzzification will be finding the membership degrees $\mu_{A_1}(\hat{x}_1)$ and $\mu_{A_2}(\hat{x}_2)$ to which $\hat{x}_1$ and $\hat{x}_2$ belong to the fuzzy sets $A_1$ and $A_2$ in the antecedent part of a fuzzy rule. Through fuzzification the degrees to which input data match the condition elements in a rule are calculated.

3.5.4 Defuzzification

This is the process of calculating a single-output numerical value for a fuzzy output variable on the basis of the inferred resulting membership function for this variable. Two methods for defuzzification are widely used. In the The centre-of-gravity-method (COG) (Faravelli and Yao; 1995) computes the geometrical centre $\hat{y}$ in the universe $V$ of an output variable $y$, which centre “balances” the inferred membership function $\hat{B}$ as a fuzzy value for $y$. In COG the following formula is used

$$\hat{y} = \frac{\sum \mu_{\hat{B}} v \mu_{\hat{B}}}{\sum \mu_{\hat{B}}} \quad (3.8)$$

Alternatively the mean-of-maxima method (MOM) (Mendel; 1995) method finds the value $\hat{y}$ for the output variable $y$ which has maximum membership degree according to the fuzzy member-
ship function \( \hat{B} \); if there is more than one value which has maximum degree, then the mean of them is taken.

While there is merit in investigating the fuzzy subsets that are the result of the fuzzification process, more commonly defuzzification is used to calculate a single (crisp) number from the fuzzy value.

### 3.5.5 Inference Methods in Fuzzy Systems

Jang (2001, pp. 2-20) defines fuzzy inference as “...the process of formulating the mapping from a given input to an output using fuzzy logic”. The result of this mapping creates the foundation for decision making or pattern recognition.

Two main types of fuzzy inference systems have been developed in the past. Mamdani-type (Mamdani and Assilian; 1975) and Sugeno-type (Sugeno; 1985; Jang and Sun; 1997) both of which combine the building blocks of membership functions, fuzzy logic operators, and if-then rules but they differ in the way the outputs of the fuzzy inference system are determined (Jang; 2001).

Since 1975 this has been one of the most common fuzzy methodologies. Motivated by a 1973 paper by Lotfi Zadeh on fuzzy algorithms for complex systems and decision processes (Zadeh; 1973) Mamdani was one of the first to apply fuzzy set theory to build control systems. Using a set of linguistic control rules elicited from experts operators, Mamdani was able to control a steam engine and boiler combination.

Fuzzy inference systems have been successfully applied in fields such as automatic control (Wang; 1993), data classification (García-Pérez et al.; 2000), decision analysis (Zhou et al.; 1999), expert systems (Woodford, Wearing, Walker and Kasabov; 1999), and computer vision (Walker; 1998). Fuzzy inference systems are referred to by many names depending on the domain in which they are applied. Fuzzy-rule-based systems (Jang; 1993), fuzzy expert systems (Kandel; 1992), fuzzy modelling (Baraldi and Blonda; 1999), fuzzy associative memory (Nauck and Kruse; 1999b), fuzzy logic controllers (Bonarini; 1996), or just fuzzy systems (Nauck and Kruse; 1999) extend conventional methods and techniques by introducing the concept of fuzzy
The reasoning strategies over fuzzy rules are many and most of them either use the *generalised modus ponens* rule (Mendel; 1995). The generalised modus ponens inference law applied over a simple fuzzy rule can be expressed as follows: (IF \(x\) is \(A\), THEN \(y\) is \(B\)) and (\(x\) is \(A'\)), then (\(y\) is \(B'\)) should be inferred (Kasabov; 1996a). More specifically, we can rewrite the generalised modus ponens rule as

\[
B' = A' \circ (A \rightarrow B) = A' \circ Rab
\]

(3.9)

where \(\circ\) is a *compositional* operator, and \(Rab\) is a fuzzy relational matrix representing the *implication relation* between the fuzzy concepts \(A\) and \(B\).

To illustrate the relationship between fuzzy concepts \(A\) and \(B\), consider the “Smoker” example taken from Kasabov (1996a, p. 186) where the relationship between the number of cigarettes smoked in a day, \(A\), implies the level of risk of cancer, \(B\). In Figure 3.9(a) two fuzzy sets are modelled: *Smoker* by two membership functions and *Risk of Cancer* by one membership function. Alternatively the same relationship between the fuzzy sets *Smoker* and *Risk of Cancer* can also be modelled as a fuzzy relational matrix, \(R\), as depicted in Figure 3.9(b). In both cases the implication is the same but modelled in a different way.

**Figure 3.9:** Modelling the risk of cancer. Reproduced from Kasabov (1996a, p. 186)
In Mamdani-type inference there is an expectation that the output membership functions are fuzzy sets (Jang; 2001). For each output variable there is a fuzzy set that need to be defuzzified. Such a process normally incurs a computational overhead since this operation needs to be applied to the entire fuzzy set.

An alternative is to use a pre-defuzzified fuzzy set which is referred to as a singleton output membership function (Jang; 2001) that is computed using the weighted average of a few data points. This model is also adopted by Sugeno-type systems where the output membership functions of the inference system are either linear or constant (Jang; 2001).

To illustrate the difference between Mamdani-type inference and Sugeno-type inference systems Jang (2001) describes a fuzzy inference system to model a “tipping problem” where the level of service and food determines the percentage of a tip. Two variables are used as input to the fuzzy inference system. The first, service, is modelled using three linguistic values: “poor”, “good”, and “excellent” and has a crisp input range between 0 and 10. The second, food, is modelled using two linguistic values: “rancid” and “delicious” and also has a crisp input range between 0 and 10. The single output of the fuzzy inference system is the tip, which is modelled using three linguistic variables: “cheap”, “average”, and “generous” and has a crisp output range between 0% and 25%. Three rules were then constructed that model the level of service and food to the amount of the tip.

1. If service is “poor” or the food is “rancid”, then tip is “cheap”

2. If service is “good”, then tip is “average”

3. If service is “excellent” or food is “delicious”, then tip is “generous”
Figure 3.10: The “tipping problem” modelled as a Mamdani-type fuzzy inference system. Reproduced from Jang (2001, p. 2–27)

Figure 3.10 is the solution to the problem realised as a Mamdani-type fuzzy inference system. The linguistic values for the input variable service are modelled as Gaussian membership functions and the linguist values for the input variable food are modelled as trapezoidal membership functions. The single output variable, tip, has its linguistic values represented as triangular membership functions.

For the crisp inputs of service=3 and food=8 all three rules have fired and their separate outputs have been aggregated into a single fuzzy set from which the crisp value of 16.7% is produced using the COG or MOM method.
In contrast, although the Sugeno-type inference system solution to this problem depicted in Figure 3.11 has the same structure for the inputs to the fuzzy inference system, it differs in terms of how the output and resulting crisp value is calculated. Singleton output functions are used instead to model the linguistic values of the output variable. Like the Mamdani-type fuzzy inference system, the outputs and aggregated but in order to determine the crisp value, a weighted average of the outputs are calculated. This results in a slightly different crisp value of 16.3%.

For fuzzy reasoning in the horticultural domain, we require a method that can accommodate both time-series analysis and classification tasks. Mamdani-type rules, because of their greater applicability to these tasks, offer a better solution to these tasks. Sugeno-type systems only offer a part solution to the problem. Most of the problems described in this thesis are of the multi-input/multi-output type. Sugeno-type systems typically only solve problems that are of the
multi-input/single-output type.

### 3.6 Neuro-Fuzzy Systems

In conjunction with fuzzy systems, the learning capabilities of neural networks make them a prime target for automating the process of developing a robust fuzzy system for a given task. The first so-called neuro-fuzzy approaches were considered mainly in the domain of fuzzy control but today the approach is more general and such systems have been applied in various domains as control, data analysis, and decision support.

The general properties of a neuro-fuzzy system are that they are trained by a modified learning algorithm that is normally derived from neural network theory. The learning procedure operates on local information and causes only local modification in the underlying fuzzy system. In many neuro-fuzzy architectures the learning process is not knowledge-based but data driven.

Neuro-fuzzy systems can also be viewed as a special case of a $n$-layer feedforward neural network with at least three layers. Fuzzy states are encoded as (fuzzy) connection weights. This view of a fuzzy system illustrates the data flow within the system and its parallel nature. However using the model of a neural network to explain the learning procedure serves as one way in which we can realise the architecture of a neural-fuzzy system.

Before, during, and after learning one can also treat a neuro-fuzzy system as a system of fuzzy rules. It is possible to create the system out of training from scratch, and to initialise it with prior knowledge in the form of fuzzy rules.

Three neuro-fuzzy systems are now introduced that cover previous attempts in the fusion of fuzzy logic and neural networks. They are the Adaptive Fuzzy Inference System (ANFIS) (Jang; 1993), NEFCLASS (Nauck and Kruse; 1995a), and the Fuzzy Neural Network (FuNN) (Kasabov, Kim, Watts and Gray; 1996). This review seeks to describe their relevant features and also identify some deficiencies in their architectures that would not make them suitable candidates for knowledge engineering from the types of horticultural data sets described later on in this thesis.
3.6.1 ANFIS

There have been several examples of neuro-fuzzy systems in the past that have exhibited those general properties described in the previous section. ANFIS is a fuzzy inference system implemented in the framework of adaptive networks, uses a hybrid learning procedure that can construct an input-output mapping based both on human knowledge (in the form of fuzzy if-then rules), and stipulated input-output pairs based on Takagi-Sugeno Type-III (Takagi and Sugeno; 1983) rules like:

\[ R_1: \text{If } x \text{ is } A_1 \text{ and } y \text{ is } B_1, \text{ then } f_1 = p_1 x + q_1 y + r_1. \]
\[ R_2: \text{If } x \text{ is } A_2 \text{ and } y \text{ is } B_2, \text{ then } f_2 = p_2 x + q_2 y + r_2. \]

where \( x \) and \( y \) are the input variables, \( A \) and \( B \) are the membership functions, \( f \) is the output variable, and \( p, q, \) and \( r \) are the consequent parameters.

Learning occurs through a gradient descent method, where the optimal parameters of the adaptive network are found. However this method is generally slow and likely to become trapped in local minima. Hence one feature of ANFIS is that it combines the gradient descent method and the Least Squares Estimate (LSE) to identify parameters of the network. LSE is used in the forward pass when learning to minimise the error between the actual state of the adaptive network and desired state of the adaptive network. In the backward pass, the error rates (the derivative of the error measure with respect to each node output) propagate from the output end toward the input end, and the parameters are updated by the gradient descent method.

Two major benefits arise from using this hybrid learning rule. Firstly the dimensionality of the gradient descent search space is drastically reduced, and secondly, it results in substantially lower convergence times.

There are other minor benefits of this architecture and learning algorithm. By employing a hybrid learning procedure, the proposed architecture can refine fuzzy if-then rules obtained by human experts to describe the input-output behaviour of a complex system. Conversely if human expertise is not available, it can still set up intuitively reasonable initial membership functions and start the learning process to generate a set of fuzzy if-then rules to approximate a desired data set. Furthermore, it does not require the choice of the number of hidden nodes. These are
taken from the number of input vectors. Finally there are a host of membership functions that could be used for the nodes thus increasing its applicability to many types of data sets.

However there are some disadvantages of ANFIS that need to be highlighted. There is only one output from an ANFIS. This is due to the nature of the format of the fuzzy rules it is attempting to represent. Thus ANFIS can only be applied to prediction tasks or the approximation of non-linear function where there is only one output. With regard to the membership functions, prior choice is a critical issue when generating the ANFIS system as the membership functions associated with each input and output node cannot be adjusted, only the values of the rules can be. Finally there is a major restriction to the ability for ANFIS to learn as there are no current variations of the hybrid learning rule that ANFIS employs.

### 3.6.2 NEFCLASS

NEFCLASS (Nauck and Kruse; 1995b) is a specialised instance of an architecture for a neuro-fuzzy system. The goal of NEFCLASS is to derive fuzzy rules from a set of data that can be separated in different crisp classes. NEFCLASS attempts to generate fuzzy rules describing the data of the form:

\[
\text{If } x_1 \text{ is } \mu_1 \text{ and } x_2 \text{ is } \mu_2 \ldots \text{ and } x_n \text{ is } \mu_n \text{ then the pattern } (x_1, x_2, \ldots, x_n) \text{ belongs to class } i, \text{ where } x_1, \ldots, x_n \text{ are the input variables and } \mu_1, \ldots, \mu_n \text{ are the fuzzy sets.}
\]

The task of the NEFCLASS model is to discover these rules and learn the shape of the membership functions. The architecture of NEFCLASS uses a three layer fuzzy perceptron where each node in the neural network models fuzzy values instead of crisp values. Figure 3.12 depicts a NEFCLASS structure of two inputs, five rules, and two output classes where \( x_1 \) & \( x_2 \) are the two crisp inputs, \( R_1, \ldots, R_5 \) are the five rule nodes, \( c_1 \) & \( c_2 \) are the two output classes, and \( \mu_1, \ldots, \mu_n \) are fuzzy sets which describe the pattern’s features.
Figure 3.12: A NEFCLASS structure of 2 crisp inputs (input variables), 5 rule nodes, and 2 output classes (output variables). Reproduced from Nauck (1997, p. 1049)

This fuzzy perceptron can be viewed as a usual 3-layer perceptron that is “fuzzified to a certain extent”. Only the weights, the net inputs, and the activations of the output units are modelled as fuzzy sets. The advantage lies with the interpretation of its structure in the form of linguistic rules, because the fuzzy weights can be associated with linguistic terms.

A NEFCLASS system can be built from partial knowledge about the patterns, and can be then refined by learning, or it can be created from scratch by learning. A user has to define the number of initial fuzzy sets partitioning the domains of the input features, and must specify the largest number of rule nodes that may be created in the hidden layer. A NEFCLASS system that is created from scratch starts with no hidden units at all. Nodes in NEFCLASS are then created during a first run by presenting it with a set of examples from the overall data set. A rule is created by finding, for a given input pattern $P$, the combination of fuzzy sets where each yields the highest degree of membership for the respective input feature. If this combination is not identical to the antecedents of an already existing rule, a new rule node is created.
Once the data has been processed, the best $K$ rules are selected, and all the other rule nodes are deleted. The best rules are found by another run through the data set, and accumulating the individual rule error values weighted by the rule activation values. After the rule base is created, the learning algorithm will adapt the membership functions of the antecedents. Normally triangular membership functions are used.

However there is one major problem associated with NEFCLASS and that is based on its learning algorithm. The learning algorithm also has a set of constraints that are required to be satisfied during learning that are mainly defined to keep the shape of the membership function and do not leave the domain of the respective input variable. But if this domain changes then it is questionable how NEFCLASS would adapt to the new input.

Also NEFCLASS has been specifically designed to be applied to classification tasks. The learning algorithm assumes that there is a known class that the parameters within the network are tuned to produce the correct class output and this limits the applicability of NEFCLASS for prediction, non-linear function approximation, or control tasks.

### 3.6.3 The FuNN

In contrast to ANFIS and NEFCLASS, FuNN is a fuzzy neural network introduced in Kasabov (1996a) and developed as FuNN/2 by Kasabov, Kim, Watts and Gray (1996). It is a feed-forward architecture with five layers of neurons and four layers of connections. In the ***input layer*** each node represents the input variable as crisp values. In the ***condition elements layer*** each node represents a fuzzy predicate of an input variable. The activation values of the node represent the membership degrees of the input variables and this is where the values are fuzzified. A different summation function, $S_C$, activation function $a_C$, and output function $O_C$ can be used in neurons of this layer. In the ***rule layer:*** each node represents either an existing rule or a rule anticipated after training and it represents the degree to which input data matches the antecedent component of an associated fuzzy rule. The inference method performed by this layer can be characterised by a summation function, $S_R$, activation function $a_R$, and a neuronal output function, $O_R$. In the ***action elements layer*** each node represents one fuzzy predicate (label) in the “action” (con-
sequent) elements of the rules. The activation of the node represents the degree to which this membership function is supported by the current data used for recall. Three functions are defined for these nodes, a summation function $S_A$, activation function $a_A$, and neuronal output function $O_A$. Finally in the output variable layer each node represents the defuzzified output variables of the system. It is defined by three functions: summation function $S_O$, activation function $a_O$, and output $O_O$.

Connection weights from the input layer to the condition element layer take are in the range $[0 \ldots 1]$ so it is assumed that the data set has been normalised to this range. Although the connection weights in the rule layer are in the range $[-1 \ldots 1]$ the sigmoidal logistic function used as the activation function for, $a_R$, in the rule layer results in activation values in the range $[0 \ldots 1]$. Finally connection weights in the output variable layer also employ a sigmoidal logistic function for $a_O$, resulting in outputs between the range $[0 \ldots 1]$.

A simple FuNN structure is shown in Figure 3.13.

![Figure 3.13: A FuNN structure of 2 inputs (input variables), 3 fuzzy linguistic terms for each variable (3 membership functions). The number of the rule (case) nodes can vary. Three output membership functions are used for the single output variable. Reproduced from (Kasabov; 1996a, p. 321)](image)

A FuNN combines the features of both a fuzzy inference engine and a neural network. The
number of neurons in each of the layers can potentially change during operation by growing or shrinking. Zeroing, pruning, applying learning with forgetting can also modify the number of connections present in the FuNN (Kasabov; 1996a; Kasabov, Kim, Watts and Gray; 1996).

In Figure 3.14 a simplified version of a FuNN is represented. There are two rules embedded in this system and are of the form:

\[ R_1: \text{IF } x_1 \text{ is } A_1(DI_{1,1}) \text{ AND } x_2 \text{ is } B_1(DI_{2,1}) \text{ THEN } y \text{ is } C_1(CF_1) \]
\[ R_2: \text{IF } x_1 \text{ is } A_2(DI_{1,2}) \text{ AND } x_2 \text{ is } B_2(DI_{2,2}) \text{ THEN } y \text{ is } C_2(CF_2) \]

where \( x_1 \) and \( x_2 \) are the input data \( A_1, A_2, B_1, \) and \( B_2 \) and the condition elements, \( DI_{1,1}, DI_{2,1}, DI_{1,2}, \) and \( DI_{2,2} \) are the degrees to which the input data matches the condition elements, \( y \) is the output data, \( C_1 \) and \( C_2 \) are the action elements, and \( CF_1 \) and \( CF_2 \) are the degrees to which the output data matches the action element.

![Figure 3.14: A simplified version of a FuNN fuzzy neural network. Reproduced from (Kasabov; 1996a, p. 321)](image)

Triangular membership functions (MF) are used in the FuNN to represent fuzzy values. The weights for the connections in the FuNN are attached to the centers of the triangles. Several training algorithms have been developed for FuNN (Kasabov; 1996a; Kasabov, Kim, Watts and Gray; 1996) where learning modifies the MFs through changing the centers and the widths of the triangles (Kasabov and Woodford; 1999).
The first modifies the existing back-propagation (BP) algorithm so that does not alter the input and the output connections representing MFs. An extension to this algorithm is the addition of a structural learning with forgetting ingredient based on the work of Ishikawa (1996) and Kozma (1996), e.g. $10^{-5}$, that is employed when the connection weights are updated. The effect of introducing this parameter decreases connection weights within the FuNN that do not contribute to a low training error. This results in a structure of the FuNN from which extraction of rules can be made possible.

The second utilises the first algorithm to calculate separately the derivatives for the two parts of the triangular MF to update the inner connection and the membership layers. This computation forms a non-monotonic activation function for the neurons in the condition element layer (Watts and Kasabov; 1998).

Genetic algorithms may also be utilised as well for training of the FuNN. Many different selection strategies and crossover points can be defined. It is also possible to combine these methods to provide a specific learning method for the FuNN.

Several algorithms for rule extraction from FuNNs have been developed and applied by Kasabov (1996a,b); Kasabov, Kim, Watts and Gray (1996) with each algorithm representing each rule node of a trained FuNN as an IF-THEN fuzzy rule.

FuNNs have several advantages when compared with the traditional connectionist systems, or with fuzzy systems (Kasabov; 1996a). FuNNs are both statistical and knowledge engineering tools. As statistical tools they can perform clustering of the input space (Kasabov; 1997) and as knowledge engineering tools rules can be extracted from a trained FuNN (Kasabov, Kim, Watts and Gray; 1996). Their learning mechanism is robust to catastrophic forgetting (Kasabov and Kozma; 1998). Where there are regions where data is sparse they can interpolate and extrapolate well. Finally both real data and fuzzy data represented as singletons can be accepted as input to the FuNN (Kasabov et al.; 1998).

The above features of FuNNs make them universal statistical and knowledge engineering tools. Many applications of FuNNs have been developed and explored so far: pattern recognition and classification; dynamical systems identification and control; modelling chaotic time series and extracting the underlying rules that can explain these chaotic processes, prediction and
decision making (Kasabov; 1996a).

But the FuNN does have some drawbacks. Determining the number of MFs for the condition and action layers is still an issue and left to much experimentation. Prior knowledge of number of rule nodes to create an optimal FuNN structure is also an issue. Although a rudimentary structural learning with forgetting ingredient is used to inhibit rule nodes that don’t contribute to a low training error, this assumes that there were already sufficient nodes in the rule layer to begin with. Penalty functions like those proposed by Setiono (1997); LeCun et al. (1990) or constructivist methods such as those proposed by Costa et al. (2002); Herrero et al. (2001) have been offered as alternative approaches to address this problem. Finally, with respect to the time required to train a FuNN, this process normally takes longer due to the complexity of the architecture of the FuNN and to compound this fact it is using a more complex version of the back-propagation algorithm to train the system.

3.7 Conclusion

In this chapter we have sought to identify and describe the generic techniques for learning systems for knowledge engineering that have been previously developed. In light of the questions posed at the beginning of Chapter 1 do these generic techniques satisfy the requirements?

The resulting coverage of these methods and techniques would indicate any of the aforementioned neuro-fuzzy models would be good candidates, but decision-making in horticulture is a very complex task and there needs to be a refinement of these methods in order for them to be applied to this problem domain. The main reason, as stated in Section 3.2, is that in order to satisfactorily model the evolving dynamics of the orchard, the learning model is required to/needs to be adaptive and learn in real-time. Moreover, methods for the acquisition and management of horticultural knowledge using such models also needs to be developed. Finally, the lack of data sets is also an issue that needs to be addressed.

The failure/inaibility of current approaches in handling these requirements is a possible reason as to the lack of applications of connectionist-based models to the horticulture industry. This is not to say that traditional symbolic artificial intelligence models such as expert systems
haven’t been applied. In fact Gerevini et al. (1992) have developed very robust expert systems and computational models to aid in measures of fruit quality and determining fruit growth patterns. However there are some disadvantages to applying such traditional techniques to the New Zealand context:

- The rule-base used in the IFP manual (Walker et al.; 1997), whilst static for a growing season, would more useful if the rule-base could be automatically modified using the data collected over the growing season in order to refine the knowledge used to generate the rules.

- It is not possible to extract new knowledge from a conventional expert system other than an “audit trail” of its reasoning to how it arrived at its decision. No real new knowledge can be acquired that may aid the orchardist in making future decisions.

- Such expert systems developed for the horticulture industry suffer from the same problems common to expert systems design. The is the process of eliciting enough knowledge with which to generate the rule-base (Liou; 1992). The initial rule-set used in the IFP manual is a combination of the results from field trials and heuristics about growing the produce. A desired outcome of collecting the data about the dynamics of the orchard means that through the use of knowledge engineering techniques, these rules can be validated.

- Developing computational models for this task normally require a reasonable amount of data with which to satisfactorily model a particular process. The IFP programme has only collected data since 1997 and using a minor subset of the potential growers of the produce. This lack of data is a major restriction in developing appropriate tools to be applied in this area.

Therefore a list of the desirable properties of the ML model for horticulture would be:

- Allows for adaptive incremental learning as data come in from the environment in real-time.

- Be robust to uncertain, noisy or corrupt data by the use of a fuzzy inference mechanism.
• Possess the ability to represent high-level knowledge fundamental to growing fruit within the model and also to extract this knowledge as rules as the model adapts to the external environment.

• Able to satisfactorily model the problem given a small data set.

The Evolving Connectionist System (ECOS) (Kasabov; 1998a) is one such framework that satisfies all five criteria as described in Section 4.3 of Chapter 4, and its embodiment in the form of an algorithm is through the Evolving Fuzzy Neural Network (EFuNN) (Kasabov; 1998b). This framework and implementation is described in Chapter 4 and compared to other ML systems that exhibit these on-line, adaptive learning abilities.
Chapter 4

Connectionist Based Methodologies for Adaptive Learning and Knowledge Engineering

4.1 Introduction

Chapter 2 reviewed a series of generic methods for the transformation and visualisation of data. Here the wavelet transformation was touted as an appropriate technique for feature extraction from the data sets used in this thesis. Similarly Chapter 3 extensively covered the field of connectionist-based systems in terms of both their architecture and learning algorithms. In the chapter several deficiencies of the current techniques were highlighted. This is especially true when their intended application is to model real-world problems whose characteristics involve real-time decision-making. The domain of horticulture certainly falls into this category.

Tasks like these require flexible learning and dynamically adaptive Intelligent Systems (IS) that have “open structures and are able to process both data and knowledge” (Kasabov; 1998a, p. 1232). Such IS offer a better solution than the traditional learning models. From this assumption fuzzy neural network models such as those proposed by Jang (1993) and Kasabov, Kim, Watts and Gray (1996) have been designed and applied to domains that require knowledge acquisition and management. However these models only address part of the solution. The notion of the “open structure” has only recently been addressed and the resulting technologies created are
the Evolving Fuzzy Neural Network (EFuNN) (Kasabov; 1998a), the Dynamic Evolving Fuzzy Inference System (DENFIS) (Kasabov and Song; 2002), and the Evolving Self-Organising Map (Deng and Kasabov; 2000); a realisation of the ECOS principle (Kasabov; 1998a).

These novel learning systems have emerged from the field of learning systems that evolve and grow to model a specific problem. Only within the past ten years have researchers begun to propose learning systems that base themselves on an open and adaptive structure instead of having a rigid structure such as the MLP. Therefore the purpose of this chapter is to introduce and detail the ECOS and the EFuNN technologies as a further extension of the current self-adaptive learning systems that have been proposed in the past. In Section 4.3 and Section 4.4 we present an overview of the ECOS and EFuNN. The extensions to these technologies constitute part of our original contribution and are detailed in Section 4.6. Several publications have been written based on this original contribution which include image classification (Woodford, Kasabov and Wearing; 1999; Woodford; 1999; Kasabov, Israel and Woodford; 2000b,a; Woodford and Kasabov; 2001a; Woodford; 2001), rule insertion and rule extraction from the EFuNN (Kasabov and Woodford; 1999), comparative analysis of these techniques against state-of-the-art learning mechanisms (Woodford; 2001), an extension of the EFuNN architecture (Woodford and Kasabov; 2001b), and an adaptive agent-based expert system (Woodford, Wearing, Walker and Kasabov; 1999).

### 4.2 On-line, knowledge-based, and adaptive methodologies

Before we introduce the ECOS and EFuNN, it is prudent to cover what other works have been conducted in the field of connectionist-based, on-line, incremental, and knowledge-based learning systems and then emphasise how our approach is both a fusion and extension of some previous works.

It is also important to create a context for the analysis of the previous work. In the case of this chapter, the following criteria are considered:

- Can these models adapt themselves to the ever changing environment from which the data
is being sourced?

- Do these models learn in an incremental, on-line, and real-time fashion?
- Do these models satisfactorily manage, manipulate, and acquire this knowledge for the important decision-making tasks normally undertaken by growers of produce in the horticulture domain?

In connectionist-based models, learning plays an essential role. It is the mechanism by which a neural network adapts itself to its environment. The result of this adaptation process, in both natural and artificial systems, is that the network obtains a representation of its environment. This representation is normally encoded as weights within the nodes of the neural network. The function of a connectionist-based structure can be described in terms of its input-output relation which in turn is fully determined by the architecture of the connectionist-based structure and by the learning rule. The representation that the network has learning of the environment enables the network to perform its function in a way that is “optimally” suited for the environment on which it is taught. Only the relevant features of the environment that help to maintain a strong input-output relation by the connectionist-based structure are retained.

This “environment” has traditionally been a data set of a predetermined size that is presented several times to the neural network model in order to progress the learning process. Unlike traditional static data sets, data sourced from many processes involved in horticulture is dynamic and frequently updated throughout a growing season. The dynamic characteristics of the data set pose a significant problem for traditional connectionist-based models both in terms of having an appropriate structure to model the process and also implementing a learning algorithm that can learn in real-time. For these reasons the traditional neural network learning algorithms such as the backpropagation learning rule are not suitable candidates.

In previous years, there has been research conducted to address this problem by developing on-line architectures and on-line learning algorithms. Saad (1998) gave a comprehensive investigation of the area. Other examples of on-line learning algorithms were proposed by Heskes and Kappen (1993) who transformed the learning process into a continuous-time class of a master
equation that can approximate the overall distribution of the examples drawn from the environment that is used to teach it. DRAMA (Billard and Hayes; 1999) developed an on-line learning model for the dynamic control and learning of autonomous robots. In this work a time-delay recurrent neural network, using the Hebb rule was used. Finally, Jordan and Jacobs (1994) used a tree structure architecture and an on-line learning algorithm in which the parameters are updated incrementally.

The nature of the learning algorithms, however, used in the above examples, rely on either sampling from a known distribution of a large data set or are limited to a small set of encoded features that are presented to the connectionist-based systems. The nature of the data sets used in this thesis do not conform to either assumption as they lack the large size and low dimensionality which would make these on-line architectures suitable candidates for selection.

There is also the issue of representing knowledge using these traditional connectionist structures. Unless knowledge about a particular process can either be represented or elicited from a neural network then such learning mechanisms offer little in aiding a person in discovering what the neural network has learnt. Concepts learnt by standard neural networks are normally incomprehensible to humans. Trained neural networks, such as the MLP, cannot easily explain its decisions, so it is difficult to convince experts of the reliability of these systems conclusions. Such criticisms of the “black-box” characteristic of neural networks reduced their use and acceptance.

Over the past decade, research had been conducted to address these issues. These type of connectionist structures are referred to as Knowledge-Based Neural Networks (KBNN). The main benefit of these connectionist systems is that knowledge can be inserted and extracted from these models thus making the “black-box” nature of these models less opaque. The knowledge represented in a KBNN is mainly in the form of different types of IF-THEN depending on the knowledge domain to be modelled, including simple propositional rules (Gallant; 1993), propositional rules with certainty factors (Fu; 1989), Zadeh-Mamdani rules (Mamdani and Assilian; 1975), and Takagi-Sugeno rules, (Takagi and Sugeno; 1983) or variations of them.

The rule extraction methods used are also dependent on the connectionist model used to represent the type of rule. A rule may be generated by observing the activations of the KBNN
when an (interesting) input is presented to it. By examining the connections of a KBNN, a rule may be derived using an appropriate algorithm. Tickle et al. (1998) and Andrews et al. (1995) compared some of the current methods for knowledge extraction from conventional neural networks while a more comprehensive study of neuro-fuzzy rule generation was conducted by Mitra and Hayashi (2000).

Alternatively a separate inference machine such as a fuzzy expert system may interpret the extracted rules. There have been many examples of this. KBANN (Towell and Shavlik; 1994) represents knowledge using propositional logic that is mapped to a standard MLP neural network. The network is then trained using a modified version of the backpropagation learning algorithm, and the modified rules extracted. RAPTURE (Mahoney and Mooney; 1993) combines connectionist and symbolic methods to both revise and structure a certainty factor rulebase (Buchanan and Shortliffe; 1984). The certainty factors become the initial weights for the neural network, the network is trained using a modified version of the backpropagation algorithm, and then revised rules are read directly off the network. Taha and Ghosh (1995) developed HIA that is a combination of connectionist and statistical to revise the initial domain knowledge for controlling water reservoirs.

Finally, the ability for a learning system to adapt itself to better model the incoming data is not just based on its learning algorithm. The actual architecture of the learning system also contributes to its performance. Learning is achieved through adaptation and normally influenced by the external environment and maybe to adjust a learning systems internal state. In connectionist-based systems this on-line learning method is known as incremental learning and refers to the ability of a neural network to learn new data without destroying the learned patterns that were previously presented to it.

This issue is known as the stability/plasticity problem has been well documented in the literature and addressed using architectures such as ART (Carpenter and Grossberg; 1991), FuzzyARTMap (Carpenter et al.; 1992), and FANNC (Zhou et al.; 2000) have been developed to support incremental learning. But this only addresses part of the problem. As the environment changes, the system must be able to maintain learning throughout its existence. This may require that the structure of the neural network be adjusted as well as its learning mechanism.
Initial approaches to this problem altered the structure of conventional MLP-based neural networks. Optimal brain damage (LeCun et al.; 1990), pruning (Miller et al.; 1996), and structural learning with forgetting, (Ishikawa; 1996), take an existing MLP-based neural network and begin to selectively reduce it down to an optimal architecture by pruning useless connections that do not contribute to the good performance of the system. Although performance gains can be attained by employing this method, such procedures assume a static data set. If the values for the data set dramatically alter over time then the structure of the system may need to be adapted to better model the data. This could result in more connections being required to create an optimal architecture yet the aforementioned algorithms do not have this ability.

An alternative approach was to grow the neural network structure as it learns. Growing Cell Structures (Fritzke; 1993) and Growing Neural Gas (Fritzke; 1995) fall into the category of constructionist approaches that start with a minimalist connectionist structure that grows as it learns. In these approaches both supervised and unsupervised learning can be applied and the resulting structures model the problem space more accurately than standard neural network structures. Recent extensions of this work have been proposed by Hamker (2001) who extends this concept of growing cell structures and Marlsand et al. (2002) in which they have addressed the problem of adapting the structure of the network to dynamic data sets.

The criteria for selecting the optimal size of the structure is of concern here. If the structure of the neural network is small it is more likely to model the global mean, rather than be tuned to local patterns and hence under-learning will occur. And if the structure is too large, too many variances will be introduced resulting in over-training, and poor generalisation. In order to avoid this problem, the learning process should dynamically adjust the neural network structure during the learning process to better represent the patterns in the data from a changing environment. This is even more important when considering the dynamic nature of the data set that the structure is attempting to accommodate.

Other incremental learning models include some form of incremental Expectation Maximisation (EM) (Moon; 1996). Starting from some initial guess this statistically-based algorithm iteratively estimates the parameters of a model. Each iteration is comprised of two steps. The Expectation (E) step, uses the known values for the observed variables and the current estimate
of the parameters to determine a distribution for the unobserved variables. Assuming that the
distribution found in the E step is correct, the Maximisation (M) step re-estimates the parameters
to be those with maximum likelihood. Neal and Hinton (1998) and Ng and McLachlan (2003)
investigated the use of incremental EM on fitting of normal mixtures.

4.3 The ECOS Principle

In the previous section we investigated the problems associated with current learning algorithms,
explored how a ML model could represent knowledge, and also looked at the concepts sur-
rounding how an incremental learning system could adapt itself to the environment. However
real-world problems such as phoneme-based speech recognition (Kasabov; 1998c), wastewater
flow time series prediction (Kasabov et al.; 2001), or financial time series prediction (Kasabov,
Deng, Erzegovesi, Fedrizzi and Beber; 2000) require Intelligent Systems (IS) that can cope with
their ever-changing complexities, dynamics, and also the tools and methods with which they can
be developed. Such systems should be able to grow as they operate, to update their knowledge
and refine the model through interaction with the environment. This is especially true in the area
of modelling the dynamics of orchard development over time. The criteria for the framework of
such IS are outlined in Kasabov (1998a):

1. IS should learn fast from a large amount of data (using fast training, e.g. one-pass training).

2. IS should be able to adapt incrementally in both real time, and in an on-line mode, where
new data is accommodated as they become available. The system should tolerate and
accommodate imprecise and uncertain facts or knowledge in order to refine its knowledge.

3. IS should have an open structure where new features (relevant to the task) can be intro-
duced at a later stage of the system’s operation. IS should dynamically create new mod-
ules, new inputs and outputs, new connections and nodes. That should occur either in a
supervised, or in an unsupervised mode, using one modality or another, accommodating
data, heuristic rules, text, images, etc.
4. IS should be memory-based, i.e. they should keep a reasonable track of information that has been used in the past and be able to retrieve some of it for the purpose of inner refinement, or for answering an external query.

5. IS should improve continuously (possibly in a life-long mode) through active interaction with other IS and with the environment they operate in.

Section 4.2 covered work conducted that partially fulfils these criteria but these approaches fall short of expectation especially in the on-line incremental approaches. Therefore one has to modify their thinking on how to develop the tools and methods that can applied to create IS that solve real world problems.

The Evolving Connectionist System (ECOS) framework has been proposed by Kasabov (1998b) to address all five criteria. ECOS is a blueprint for creating IS that evolve in time through interaction with the environment. They initially start with some pre-defined knowledge but they also learn and adapt as they operate. They evolve or develop through incremental learning mechanisms through changing their structure in order to better represent the data. They learn in an on-line and a knowledge-based mode accommodating any new data from a data stream. The learning process itself can be expressed as a process of rule manipulation.

The main parts of the ECOS are described below. For a more detailed explanation refer to Kasabov (2001a):

1. **Feature selection part:** It performs filtering of the input information, feature extraction and forming the input vectors. The number of inputs (features) can vary from example to example from the input data stream fed to the ECOS.

2. **Presentation and representation (memory) part:** Where information (patterns) are stored. It is a multi-modular, evolving structure of a Neural Network Module (NNM) organised in spatially distributed groups; for example one NNM can represent the phonemes in a spoken language (one NN representing one class phoneme).

3. **Higher-level decision part:** This part consists of several modules, each taking decision on a particular problem (e.g., phoneme, word, concept). The modules receive feedback from
the environment and make decisions about the functioning and the adaptation of the whole ECOS.

4. **Action parts:** Take the output from the decision modules and pass output information to the environment.

5. **Self-analysis, and rule extraction modules:** This part extracts compressed abstract information from the representation modules and from the decision modules in different forms of rules or abstract symbolic associations.

### 4.4 The EFuNN

EFuNNs are an extension of the FuNN structure that evolve according to the ECOS principles (Kasabov; 1998a, 1999a,b, 1998b). EFuNNs adopt some known techniques from Kohonen (1990); Amari and Kasabov (1998); Carpenter and Grossberg (1991); Kohonen (1997) but they also introduce new NN techniques, e.g. all nodes in an EFuNN are created during (possibly one-pass) learning. They have a five layer structure that is depicted in Figure 4.1.

![Diagram of the EFuNN architecture.](image)

Figure 4.1: The architecture of the EFuNN. Reproduced from Koprinska and Kasabov (1999, p. 96)
The input layer represents input variables. The second layer of nodes (fuzzy input neurons) represent fuzzy quantification of each input variable space using membership functions. The third layer contains rule nodes that evolve through learning according to the ECOS principles. Initially there are no rule nodes in this layer. Rule nodes are either evolved when the EFuNN is training or they may initially be inserted into the EFuNN prior to training. When the EFuNN is trained rule nodes are added and adapted based on the data in the data set. The rule nodes represent prototypes of data mapping between the fuzzy input and fuzzy output spaces. Each rule node is defined by two vectors of connection weights: $W_1$ and $W_2$. The former is adjusted via unsupervised learning based on the similarity between the fuzzy input vector and the prototypes already stored. $W_2$ is updated by the least mean squared algorithm to minimise the output fuzzy error. Neurons in the fourth layer are called fuzzy output neurons and represent the fuzzy quantisation for the output variables. Finally the fifth layer contains output nodes that represent the real values for the output variables.

The EFuNN evolving algorithm is given as a procedure of consecutive steps (Kasabov; 1998a,b,d, 1999a, 2001b). Although there are several options for growing of the EFuNN we restrict the learning algorithm to the 1-of-n algorithm which is split between the definitions in Algorithm 1 and Algorithm 2.

Algorithm 1 preprocesses the dataset by normalising the data values between 0 and 1 then fuzzifying the normalised data using triangular membership functions similar to those used in the FuNN.

Algorithm 2 is the actual EFuNN learning algorithm that uses the preprocessed data set from Algorithm 1 and generates a trained EFuNN. Of note are several key steps in this algorithm that require explanation.

Step 6 of the algorithm calculates the normalised fuzzy local distance, $D$, between the current fuzzy input vector $inpF_i$ and the already stored prototypes in the rule nodes $r_j, j = 1 \ldots R$, where $R$ is the current number of rule nodes. These distances are then used in Step 7 to calculate the activation $A_{1r_j}$ of the rule nodes $r_j, j = 1 \ldots R$. The rule node with the highest activation, $r_j^*$, is then compared to the sensitivity threshold $sThr$ as in Step 8. An activation lower than sensitivity threshold $sThr$ would indicate that the current rule nodes are not similar to the current fuzzy
Algorithm 1: The data preprocessing and fuzzification algorithm for the EFuNN

**Input:**
- A set of training examples, $X_1, X_2, \ldots, X_n$ where $N$ is the number of training examples
- $mf$ which is the number of membership functions to fuzzify both the input and output variables in the training examples
- $numInputs$ is the number of input vector features
- $numOutputs$ is the number of output vector features
- $Y$ is the number of classes in the data set where $Y = 1$ for prediction problems and $Y > 1$ for classification problems

**Output:**
- Two normalised and fuzzified matrices $\text{inpF}$ and $\text{targetF}$ along with the $W_3$ defuzzification matrix

1. $\text{inp} = X(1:\text{numInputs})$
2. $\text{target} = X(1:\text{numOutputs})$
3. For $i = 1$ to $N$ do
   4. Normalise the data instance between 0 and 1;
   5. $\text{inp}_i = \text{Normalise}(\text{inp}_i)$;
   6. Fuzzify the input and output so that each fuzzy input and fuzzy target vector has $X \cdot mf$ elements;
   7. $\text{inpF}_i = \text{Fuzzify}(\text{inp}_i)$;
   8. $\text{targetF}_i = \text{Fuzzify}(\text{target}_i)$;
4. For $i = 1$ to $Y$ do
   11. $W_3_y = \frac{y}{mf-1}$
13. end

input vector so a new rule node is created.

A similar approach is taken in Steps 13 to 18 to calculate the error between the rule node layer and the fuzzy output layer. The activation value of $r_j^*$ is propagated throughout the EFuNN to produce activations, $A2$, for the action neurons. The action node $k_*$ with highest activation from $A2$ is then found. If the output activation error $k_*$ is greater than the error threshold, $errThr$, then this means the fuzzy output for the current fuzzy input vector is not similar to the current action neurons of the EFuNN so a new rule node is created.

In Step 23 and Step 24 the input and output connections of rule node, $k_*$, are then updated where $D(\text{inpF}_i, r_j)$ is the normalised fuzzy distance between two fuzzy vectors as defined by Step 5. Computing $Err_{k^*} \cdot A1_{j^*}$ has the effect of “shifting” the rule node to the center of the input space rule node $k^*$ accommodates. All updates to the $W1$ and $W2$ weights always result in non-negative values in the range $[0 \ldots 1]$.  

81
Algorithm 2: The EFuNN learning algorithm

Input:
A set of fuzzified training examples comprising of matrices inpF and targetF
The W3 defuzzification matrix

Output:
The trained EFuNN comprising of matrices W1, W2, and W3

1. Create the first rule node \( r_1 \) to represent the first example;
2. \( W_{11} = \text{inp}^F_1, W_{21} = \text{target}^F_1 \);
3. \( i = 1 \);
4. while \( i < N \) do
   5. \( i = i + 1 \);
   6. \[ D(\text{inp}^F_i, r_j) = \frac{\sum_{j=1}^{m} |\text{inp}^F_i - W_{1j}^i|}{\sum_{j=1}^{m} W_{1j}^i} \]
   7. \( A_{1j} = 1 - \frac{D(\text{inp}^F_i, r_j)}{D(\text{inp}^F_i, r_j)} \);
   8. if \( A_{1j} < s\text{Thr} \) then
      9. create a new rule node;
     10. \( W_1 = \text{inp}^F_i, W_2 = \text{target}^F_i \);
     11. \( j = j + 1 \);
   else
      12. find the action node \( r_a \) with highest activation \( A_1 \);
      13. Propagate the activation of \( r_a \) to the action neurons;
      14. \( A_2 = A_{1j} \cdot W_{2a} \);
      15. Calculate the fuzzy output error;
      16. \( \text{Err} = |A_2 - \text{target}^F_i| \);
      17. find the action node \( k_a \) with highest activation \( A_2 \);
      18. if \( (\text{Err}(k_a) > \text{errThr}) \) then
         19. create a new rule node::;
        20. \( W_1 = \text{inp}^F_i, W_2 = \text{target}^F_i \);
        21. \( j = j + 1 \);
      else
        22. Update the input and output connections of rule node \( k^* \);
        23. \( W_{1k}^{i+1} = W_{1k}^i + lr_1 \cdot D(\text{inp}^F_i, r_j) \);
        24. \( W_{2k}^{i+1} = W_{2k}^i + lr_2 \cdot \text{Err}(k_a) \cdot A_{1j} \);
      end
   end
   if \( (i == i\text{Ag}) \) then
      29. aggregate the rule nodes;
      30. \[ D_{\text{eucl}}(W_{1r_a}, W_{1r_a}) = \sqrt{\sum_{j=1}^{m} (W_{1j}^r - W_{1j}^a)^2} / \sqrt{m} < w1\text{Thr} \]
      31. \[ D_{\text{eucl}}(W_{2r_a}, W_{2r_a}) = \sqrt{\sum_{j=1}^{m} (W_{2j}^r - W_{2j}^a)^2} / \sqrt{m} < w2\text{Thr} \]
      32. where \( m \) and \( l \) are the number of conditional nodes and action nodes, respectively;
      33. Merge the nodes \( r_a, a = 1 \ldots A \) and update \( W_{1r_j} \) and \( W_{2r_j} \) as follows;
      34. \( W_{1r_j} = \frac{\sum_{a=1}^{A} (W_{1r_a})}{A}, W_{2r_j} = \frac{\sum_{a=1}^{A} (W_{2r_a})}{A} \);
      35. Delete \( r_a, a = 1 \ldots A \);
   end
end
Finally Steps 29 to 35 is when rule node aggregation may occur. For each rule node $R_j, j = 1 \ldots R$ finds the subset of rule nodes $r_a, a = 1 \ldots A, A < R$ for which the normalised euclidean distances $D_{euc}(W1_{rj}, W1_{ra})$ and $D_{euc}(W2_{rj}, W2_{ra})$ are below the thresholds $w1Thr$ and $w2Thr$, respectively. This subset of the rule nodes, $r_a$, are then merged by creating a new rule node that is the average of the $W1$ and $W2$ weights in the subset of rule nodes, $r_a$.

For classification problems, an example that has not been seen during the learning phase, is first normalised then propagated through the EFuNN. The propagation from rule nodes to output layer is restricted only for the winning rule node. The example is classified as an instance of the class $y$, where $y$ is the index of the output neuron with the highest activation value.

Similarly, for prediction problems, an example that has not been seen during the learning phase, is first normalised then propagated through the EFuNN. The propagation from rule nodes to output layer is restricted only for the winning rule node. The output from the winning rule node is therefore the output of the EFuNN and normally compared against the desired (known) output for the unseen example to calculate the error between the output of the EFuNN (actual) and the desired output.

### 4.5 Previous Rule Insertion and Rule Extraction Methods

Given that an appropriate model can be chosen to represent the knowledge required to make a decision, additional issues also need to be addressed in how this knowledge is managed and extracted using incremental learning techniques such as the EFuNN.

As the knowledge used to grow fruit using the IFP programme is inherently rule-based then this representation would logically be the best choice. Furthermore with the FuNN and EFuNN models being the realisation of a fuzzy rule-based expert system embedded in a neural network, the most appropriate way to handle this knowledge would be through rule insertion and rule extraction.

There already have been several attempts in the past for rule extraction in traditional neural network structures such as the MLP. Towell and Shavlik (1993) use a partitioning algorithm to produce subsets of nodes in the neural network that can then be thought of as rules. (Craven and
Shavlik; 1997) view the problem of extracting comprehensible hypotheses from a trained neural network as an inductive learning task. In this learning task, the target concept is the function represented by the network, and the hypothesis produced by the learning algorithm is a decision tree that approximates the network. Finally Omlin and Giles (1996) used a recurrent neural network to classify strings of a regular language. Rules defining the learned grammar could be extracted from the trained neural network in the form of deterministic finite-state automata.

Similarly, previous works on rule insertion methods have been conducted to “prime” the neural network with prior knowledge and boost its ability to successfully learn the data set. The Knowledge-Based Artificial Neural Network (KBANN) developed by Shavlik (1992) produces neural networks whose topological structure matches the dependency structure of the rules in an approximately-correct “domain theory” (a collection of inference rules about the current task). Frasconi et al. (1995) represents the rules as Discrete Finite Automata (DFA) with the states of the DFA being represented as activations in the nodes of the neural network. More recently Tan (1997) has used a fuzzy ARTmap to model rules within the system. The recognition nodes in the \( F_2 \) competitive layer are treated as rules that links the antecedent to the consequent. Lee et al. (1997) used a rule insertion algorithm to insert refined fuzzy rules into a fuzzy neural network model to improve its learning performance. Tan (2001) integrated IF-THEN rules into a hybrid neural network based on ART for the purpose of text categorisation. Raouzaiou et al. (2002) embedded rules in a fuzzy system for the purpose of facial expression recognition. Finally Prentzas et al. (2002) describes a method by which specified domain knowledge can be transformed into hybrid rules that are destined for a neural network.

### 4.5.1 NeuroRule and NeuroLinear

Two rule extraction algorithms of direct relevance to this thesis are the work of Setiono and Liu (1996, 1997). NeuroRule (Setiono and Liu; 1996) and NeuroLinear (Setiono and Liu; 1997) were developed to extract human-interpretable rules from a trained neural network which can explain its decision process.

Both algorithms consist of three main stages:
1. The process first starts out by training a three layer neural network. This network is a standard structure consisting of three layers: the input layer, a hidden layer, and an output layer. Both the input layer and output layer corresponds to the dimensionality of the inputs of the data set and number of output classes respectively. The number of units in the hidden layer is initially set at a high value and all the weights of the neural network are initially populated with random values between [-1.0 ... 1.0].

To train the network a modified version of the standard backpropagation algorithm that also employs a weight decay function prunes redundant units from both the input and hidden layers while at the same time preserving the accuracy of the network. This algorithm is detailed in Setiono (1997). The key feature of this algorithm is that it inhibits the network weights from obtaining overly high values whilst decreasing weights or irrelevent connections to zero. This has the benefit is removing input units that do not contribute to the classification and allow for the production of a simple set of rules to explain behaviour of the neural network.

The neural network is trained and pruned for a number of epochs until it falls below a predefined level of classification accuracy. Once this step is completed, an optimal neural network will be produced that only contains a minimal number hidden nodes and connections between the input and output nodes that enabled it to maintain a high classification accuracy.

2. Because the hidden unit activations are values within the range between [-1.0 ... 1.0], they are not in a representation that allows for the extraction of rules. Grouping these activation values into a small set of clusters whilst conserving the initial accuracy of the network is required. (Setiono and Liu; 1997) discretise the hidden unit activation values using the Chi2 algorithm (Liu and Setiono; 1995) which iteratively merges continuous-valued attributes into smaller and smaller subintervals from which unique discrete values are produced that account for each subinterval. This process continues until the subintervals cannot be merged any further. A value, Chi-0, determines the point at which merging the subintervals terminates.
3. To generate the rules involves two phases. In the first phase rules that relate the discretised hidden unit activation values to the classification are produced. In this phase the X2R rule generation algorithm (Liu and Tan; 1995) is used to generate a minimal set of rules that cover all the data. Generation of rules terminates when the error rate exceeds the inconsistency rate present in the data. In the second phase rules that relate the original input attributes of the data set to the discretised hidden unit activation values are generated. This is possible because another result of applying the Chi2 algorithm also produces the boundaries of the subintervals of the hidden unit activation values.

Therefore any activation of an input pattern will be discretised into one of the clusters bounded by a subinterval and these clusters map directly to one of the classifications produced by the X2R algorithm. This means that weights of the network connections from the input units to the hidden units specify the rule antecedents that determine the subintervals of the discretised hidden unit activation values.

Hence two separate mappings are created: 1) between the input attributes and the hidden unit activations and 2) between the hidden unit activations and the classification. These rule “components” are then combined which results in decision boundaries in terms of the original attributes that classify the patterns of the data set.

NeuroRule and NeuroLinear generate rules that are both minimal in terms of the number of rules and the number of antecedents per rule as well as being very human comprehensible. Very high classification rates have been reported using the rules generated from these two algorithms. Although NeuroLinear had initially been designed to produce rules for classifiers (Baesens et al.; 2003), more recently the same algorithm had been used to generate rules for regression problems (Setiono and Thong; 2004).

4.6 Rule Insertion and Rule Extraction in EFuNN

Yet none of these approaches fit comfortably with the underlying concept of ECOS and the EFuNN. Since this architecture is based on an incremental learning algorithm where the nodes
within the EFuNN grown, aggregated, and pruned then rules within this neural network need to be handled in a different way. One approach by Kasabov and Woodford (1999) has addressed the problem of rule insertion and rule extraction in the EFuNN.

An EFuNN in particular can be adequately represented by a set of fuzzy rules through rule extraction techniques (Kasabov; 1996a,b, 1998d, 2001b). The fuzzy inference is embodied in the connectionist structure. In this respect an EFuNN can be considered as an evolving fuzzy system. The rules that represent the rule nodes need to be aggregated in clusters of rules. The degree of aggregation can vary depending on the level of granularity needed. Sometimes, for explanation purposes, the number of rules needed, could be even less than the number of the fuzzy output values. At any time (phase) of the evolving (learning) process, fuzzy, or exact rules can be inserted and extracted. This is the original contribution of rule extraction from an EFuNN and rule insertion into an EFuNN developed by the author of this thesis.

Insertion of fuzzy rules in the EFuNN is achieved through setting a new rule node for each new rule, such as the connection weights $W_1$ and $W_2$ of the rule node represent the fuzzy or the exact rule (Kasabov and Woodford; 1999) and can be implemented using the description detailed in Figure 4.2.

**Example 1:** The fuzzy rule IF $x_1$ is Small AND $x_2$ is Small THEN $y$ is Small, can be inserted into an EFuNN structure by setting the connection weights of a new rule node to the fuzzy condition nodes $x_1 – Small$ and $x_2 – Small$ to 0.5 each, and the connection weights to the output fuzzy node $y – Small$ to a value of 1. The rest of the connections are set to zero.

**Example 2:** The exact rule IF $x_1$ is 3.4 AND $x_2$ is 6.7 THEN $y$ is 9.5 can be inserted in the same way as in Example 1, but here the membership degrees to which the input values $x_1 = 3.4$ and $x_2 = 6.7$ belong to the corresponding fuzzy values are calculated and attached to the connection weights instead of values of 1.0.

The same procedure is applied for the fuzzy output connection weights. Changing a MF during operation may be needed for a re-fined performance after a certain time of the system operation. For example, instead of three MFs the system can change to five MFs. In traditional fuzzy neural networks this change is not possible, but in EFuNNs it is possible, because an EFuNN stores in its $W_1$ and $W_2$ connections fuzzy exemplars. These exemplars, if necessary, can be defuzzified at any time of the operation of the whole system, then used to evolve a new EFuNN structure.

**Figure 4.2:** Rule insertion into an EFuNN
Guiding the development of the EFuNN rule extraction algorithm were the criteria proposed by Craven (1996) when extracting symbolic representations from neural networks:

1. **Comprehensibility:** The algorithm should generate symbolic representations that are comprehensible by experts in the domain.

2. **Fidelity:** The algorithm should produce symbolic representations that accurately model the network from which they were extracted.

3. **Scalability:** The algorithm should be scalable to networks that have large input spaces and large numbers of units and weighted connections.

4. **Generality:** The algorithm should require neither special training regimes, nor restrictions on the network architecture.

Algorithm 3 presents the steps that comprise the EFuNN rule extraction algorithm for rule extraction from a trained EFuNN, where $W_1$ represents the weight matrix of the connections between the fuzzy inputs layer and the rule layer, and $W_2$ represents the weight matrix of the connections between the rule layer and the fuzzy output layer (Kasabov and Woodford; 1999).

The values $[W_1(t_{ij})]$ are interpreted as fuzzy coordinates at the clusters represented in the rules $R_j$ and the values $[W_2(t_{ij})]$ are interpreted as certainty factors that this cluster belongs to the fuzzy output class.

The rules that have the same condition and action parts, but differ in fuzzy co-ordinate value and certainty factors, are aggregated by taking the average of the values across the fuzzy co-ordinates as in Step 30 of the algorithm and the maximum value for the certainty degrees as in Step 31 of the algorithm. Taking an average value for the fuzzy co-ordinates is equivalent to finding the geometrical centre at the cluster that aggregates several rule nodes into one.

The rules that are extracted are in the form:

$$IF \ In_1 \ is \ MF\text{(degree)} \ldots \ In_n \ is \ MF\text{(degree)} \ THEN \ Output \ is \ MF\text{(degree)} \ or \ Output \ is \ Class\text{(degree)}$$

where:

- $In_n$ is the $n$th attribute of the input vector. Where there are four or less input attributes, the labels are changed to their linguistic realisation.
**Algorithm 3**: The EFuNN Rule Extraction Algorithm

**Input**:
The number of membership functions, \( m_f \)
An evolved/trained EFuNN comprising of matrix \( W_1 \) and matrix \( W_2 \)
Threshold \( T_{hr1} \) for the \( W_1 \) matrix which has the value 0...1
Threshold \( T_{hr2} \) for the \( W_2 \) matrix which has the value 0...1

**Output**:
A set of \( R \) rules

1. Set number of extracted rules, \( r_n \), to zero;
2. \( r_n=0; \)
3. Find the number of inputs and outputs;
4. \( \left[ \text{numInputs}, \text{numRows} \right]=\text{Size} \left( W_1 \right)/m_f; \)
5. \( \left[ \text{numOutputs}, \text{numRows} \right]=\text{Size} \left( W_2 \right)/m_f; \)
6. Eliminate the weights in both \( W_1 \) and \( W_2 \) that are below \( T_{hr1} \) and \( T_{hr2} \) respectively;
7. \( \text{for } i=1 \text{ to } \text{numRows} \text{ do} \)
8. \( \text{for } j=1 \text{ to } \text{numInputs} \cdot m_f \text{ do} \)
9. \( \text{if } W_1(i,j) > T_{hr1} \text{ then} \)
10. \( \quad W_1(i,j) = W_1(i,j); \)
11. \( \text{else} \)
12. \( \quad W_1(i,j) = 0; \)
13. \( \text{end} \)
14. \( \text{end} \)
15. \( \text{for } j=1 \text{ to } \text{numOutputs} \cdot m_f \text{ do} \)
16. \( \text{if } W_1(i,j) > T_{hr2} \text{ then} \)
17. \( \quad W_2(i,j) = W_1(i,j); \)
18. \( \text{else} \)
19. \( \quad W_2(i,j) = 0; \)
20. \( \text{end} \)
21. \( \text{end} \)
22. \( \text{end} \)
23. Remove all rows in \( W_1_t \) and \( W_2_t \) where all elements in the row are equal to zero;
24. \( W_1_t=\text{Remove} \left( W_1_t==0; \right); \)
25. \( W_2_t=\text{Remove} \left( W_2_t==0; \right); \)
26. Aggregate the remaining rows;
27. \( \text{while } W_1_t \neq \{ \} \text{ and } W_2_t \neq \{ \} \text{ do} \)
28. \( \text{for } i=1 \text{ to } m_f \text{ do} \)
29. \( \text{forall elements of } W_1_t \text{ and } W_2_t \text{ where each element has the } m_f \text{ value of } i \text{ do} \)
30. \( R_{r_n} = \text{Average} \left( W_1_t; \right); \)
31. \( R_{r_n} = \text{Maximum} \left( W_2_t; \right); \)
32. \( r_n=r_n+1 \)
33. \( \text{end} \)
34. \( \text{end} \)
35. \( \text{end} \)
• **MF** is the linguistic realisation of the Membership Function (MF) that models either the input attribute or the output attribute. In this version of the EFuNN the input vectors are fuzzified using triangular membership functions. For the experiments described in this thesis no more than five MFs were used to model the input and output vectors thus the linguistic realisation of these MFs were:

  - 2MFs = \{low, high\}.
  - 3MFs = \{low, medium, high\}
  - 5MFs = \{low, low-medium, medium, medium-high, high\}

• **degree** is a confidence value between 0.0 and 1.0 where 0.0 indicates no confidence and 1.0 indicates full confidence.

• For the consequent of the rules in the case of prediction tasks, the format is **Output is MF(degree)** where:

  - **MF** is the linguistic realisation of the MFs modelling the output of the rule
  - **degree** is a confidence value between 0.0 and 1.0 where 0.0 indicates no confidence and 1.0 indicates full confidence.

• In terms of classification tasks, the format for the consequent part of the rule is **Output is Class(degree)** where:

  - **Class** is a nominal value that indicates the one of the classes that exist in the data set.
  - **degree** is a confidence value between 0.0 and 1.0 where 0.0 indicates no confidence and 1.0 indicates full confidence.

In both the cases of prediction tasks or classification tasks all the extracted rules must be evaluated to draw an inference.
4.7 Empirical/Benchmark study

To illustrate our method, two experiments were conducted to test EFuNN rule insertion and rule extraction algorithms. The first experiment is a benchmark problem of adaptive time-series prediction (approximation) illustrated on the gas-furnace data. The second experiment is of a classification problem illustrated on the benchmark Iris data set.

In order to justify our results on the classification example in the chapters and those in the rest of the thesis, some method of cross-validation was necessary. Under normal conditions where there is a data set of a reasonable size then such methods as ten-fold cross validation or leave-one-out (Kohavi; 1995) can be applied. But this assumption does not hold with the data sets used in this thesis.

The composition of most of the data sets is such that there is an varied number of data examples per class. Therefore it is possible that applying such a method of ten-fold cross validation will result in training examples from the full data set that do not contain data instances from all the classes. Conversely there may be a case that the classifier is not tested on data instances from all the classes.

To account for this situation we have developed a method that ensures there are enough data examples from each class with which to train and test the classifiers used in this thesis.

By using a function from the MATLAB PRTOOLS Toolbox (Duin; 2000), data examples from the full data set were randomly selected and allocated to either a training data set or testing data set. This function guaranteed that there would be data examples from all the classes in the two respective data sets. The classifier was trained on the training data set and tested on the testing set. This process was repeated ten times. We call this method the multiple-run method.

The choice of learning parameters for the EFuNN were based on the fact that the two experiments also involved the rule insertion and rule extraction algorithms. The sensitivity threshold, and error threshold parameters were selected to generate a reasonably large number of rule nodes compared to the total number of training examples. This was deemed important as our rule extraction technique would expect to aggregate these initial set of rules into a smaller set of rules. The parameters were selected so that there would not be a 1-1 mapping between the number of...
data examples and the number of rule nodes thus increasing the generalisation capability of the EFuNN.

4.7.1 Gas-Furnace data set

The gas-furnace data has been used by many researchers in the area of neuro-fuzzy engineering Jang (1993); Kohonen (1990). The data set consists of 292 consecutive values of methane at a time moment \((t - 4)\), and the carbon dioxide \(CO_2\) produced in a furnace at a time moment \((t - 1)\) as input variables, with the produced \(CO_2\) at the moment \(t\) as an output variable. The following steps were taken:

1. Evolve an EFuNN on half the data set for one pass and then test on the whole data set.
2. Retrain the EFuNN on the entire data set initially for one pass, repeat for another pass of training, and then test on the entire data set.
3. Extract rules from the EFuNN and then insert into a brand new empty FuNN structure and then re-test on the entire data set.

For the above task an EFuNN was set up with an architecture of

- 2(inputs)-5(inputMFs per input)-0(initial ruleNodes)-5(outputMFs per output)-1(output)

The sensitivity threshold \(Sthr=0.1\), error threshold \(Errthr=0.05\), and learning rate for both the first and second layer \(lr=0.5\). After the completion of Step 1 the number of rule nodes generated was 107 and the Root Mean Squared (RMS) error of the tested EFuNN was 0.008. The result in Figure 4.3 shows that the EFuNN has accurately learnt the first half of the data set and generalised well to the other half of the data set just after one pass of learning.
After Step 2 the EFuNN generated 119 rule nodes with a RMS of 0.015 and 219 rule nodes with a RMS of 0.010 respectively. Figure 4.4 shows the results of testing the entire data set on the retrained EFuNN after both passes of the data set. In this case the EFuNN has retained the memory of the first half of the data set whilst achieving a better function approximation of the entire data set. The EFuNN further improved after the second pass of training but there was a dramatic increase in the number of rule nodes.
In the final part of the experiment a set of rules were extracted from the EFuNN in Figure 4.3. Where there was a case in which rules that had the same condition and action part existed, the average of the condition values and the maximum of the action values were taken. This has been found to generate the best results as performing this operation was a form of aggregating the rules. This resulted in 29 rules extracted using the algorithm for Thr1=0.1 and Thr2=0.8.

The rules extracted from the EFuNN are listed below in Table 4.1:

<table>
<thead>
<tr>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF CO2(<em>{(t-4)}) is high(0.80) AND CO2(</em>{(t-1)}) is low-medium(0.82) THEN CO2(_{(t)}) is low-medium(0.72)</td>
</tr>
<tr>
<td>IF CO2(<em>{(t-4)}) is high(0.75) AND CO2(</em>{(t-1)}) is low(0.95) THEN CO2(_{(t)}) is low(1.00)</td>
</tr>
<tr>
<td>IF CO2(<em>{(t-4)}) is medium-high(0.83) AND CO2(</em>{(t-1)}) is low(0.65) THEN CO2(_{(t)}) is low-medium(0.59)</td>
</tr>
<tr>
<td>IF CO2(<em>{(t-4)}) is medium-high(0.58) AND CO2(</em>{(t-1)}) is low-medium(0.62) THEN CO2(_{(t)}) is low(0.57)</td>
</tr>
<tr>
<td>IF CO2(<em>{(t-4)}) is medium-high(0.73) AND CO2(</em>{(t-1)}) is low(0.57) THEN CO2(_{(t)}) is low(0.57)</td>
</tr>
<tr>
<td>IF CO2(<em>{(t-4)}) is medium(0.66) AND CO2(</em>{(t-1)}) is low(0.57) THEN CO2(_{(t)}) is low-medium(0.67)</td>
</tr>
<tr>
<td>IF CO2(<em>{(t-4)}) is low(0.68) AND CO2(</em>{(t-1)}) is medium(0.58) THEN CO2(_{(t)}) is medium-high(0.90)</td>
</tr>
<tr>
<td>IF CO2(<em>{(t-4)}) is low(0.95) AND CO2(</em>{(t-1)}) is medium-high(0.67) THEN CO2(_{(t)}) is high(0.70)</td>
</tr>
<tr>
<td>IF CO2(<em>{(t-4)}) is low-medium(0.56) AND CO2(</em>{(t-1)}) is low-medium(0.66) THEN CO2(_{(t)}) is medium(0.66)</td>
</tr>
<tr>
<td>IF CO2(<em>{(t-4)}) is medium-high(0.58) AND CO2(</em>{(t-1)}) is medium(0.56) THEN CO2(_{(t)}) is low-medium(0.55)</td>
</tr>
<tr>
<td>IF CO2(<em>{(t-4)}) is low(0.52) AND CO2(</em>{(t-1)}) is high(0.97) THEN CO2(_{(t)}) is high(1.00)</td>
</tr>
<tr>
<td>IF CO2(<em>{(t-4)}) is medium(0.55) AND CO2(</em>{(t-1)}) is high(0.79) THEN CO2(_{(t)}) is high(0.60)</td>
</tr>
<tr>
<td>IF CO2(<em>{(t-4)}) is medium(0.86) AND CO2(</em>{(t-1)}) is high(0.60) THEN CO2(_{(t)}) is medium-high(0.78)</td>
</tr>
<tr>
<td>IF CO2(<em>{(t-4)}) is low(0.50) AND CO2(</em>{(t-1)}) is medium-high(0.89) THEN CO2(_{(t)}) is medium-high(0.72)</td>
</tr>
<tr>
<td>IF CO2(<em>{(t-4)}) is low-medium(0.66) AND CO2(</em>{(t-1)}) is medium-high(0.51) THEN CO2(_{(t)}) is high(0.54)</td>
</tr>
<tr>
<td>IF CO2(<em>{(t-4)}) is low-medium(0.98) AND CO2(</em>{(t-1)}) is high(0.54) THEN CO2(_{(t)}) is high(0.54)</td>
</tr>
<tr>
<td>IF CO2(<em>{(t-4)}) is low-medium(0.78) AND CO2(</em>{(t-1)}) is high(0.54) THEN CO2(_{(t)}) is medium-high(0.51)</td>
</tr>
<tr>
<td>IF CO2(<em>{(t-4)}) is medium(0.53) AND CO2(</em>{(t-1)}) is medium-high(0.58) THEN CO2(_{(t)}) is medium(0.74)</td>
</tr>
<tr>
<td>IF CO2(<em>{(t-4)}) is medium-high(0.64) AND CO2(</em>{(t-1)}) is medium(0.61) THEN CO2(_{(t)}) is medium(0.61)</td>
</tr>
<tr>
<td>IF CO2(<em>{(t-4)}) is medium-high(0.55) AND CO2(</em>{(t-1)}) is low-medium(0.71) THEN CO2(_{(t)}) is low-medium(0.82)</td>
</tr>
<tr>
<td>IF CO2(<em>{(t-4)}) is medium(0.97) AND CO2(</em>{(t-1)}) is low-medium(0.82) THEN CO2(_{(t)}) is low-medium(0.82)</td>
</tr>
<tr>
<td>IF CO2(<em>{(t-4)}) is medium-high(0.62) AND CO2(</em>{(t-1)}) is medium(0.74) THEN CO2(_{(t)}) is medium-high(0.55)</td>
</tr>
</tbody>
</table>

(continued on next page)
A new EFuNN was initialised and the 29 rules inserted into it. It was then tested on the entire gas-furnace data set and the results shown in Figure 4.5. The results show that only 29 clusters (new rule nodes) were created to aggregate the previous 219 and the obtained RMS error of this EFuNN was 0.08.

![Figure 4.5: Testing of EFuNN with 29 rules inserted into its structure (no further training)](image)

The result of this experiment suggests that using a much smaller ruleset gave an acceptable performance for the EFuNN. This would indicate that there are a lot of duplicate or redundant rules within the EFuNN before rule extraction. By deleting these rules, a more compact EFuNN structure has been produced. Where the EFuNN did not perform well was in instances of the data set where the current output value altered dramatically from the previous one.
4.7.2 Iris Data set

The realisation of rule extraction and insertion was used on the benchmark Iris Classification data set (Fisher; 1936; Dasarathy; 1980). This data set consists of 150 instances of four measured attributes: Sepal-Length, Sepal-Width, Petal-Length, and Petal-Width of three different varieties of the Iris flower: *Iris-Setosa*, *Iris-Versicolour*, and *Iris-Virginica*. By performing a Principal Component Analysis (PCA) on the data set and projecting the first two principal components onto a 2D plot, Figure 4.6 clearly shows that the main problem that this data set presents to any classifier is that it needs to accurately differentiate between the *Iris-Versicolour* class from the *Iris-Virginica* class.

![Figure 4.6: PCA projection of the first two principal components of the Iris data set](image)

Hence we would expect the EFuNN to generate a smaller number of rules to classify the *Iris-Setosa* class and a much larger number of rules associated with classifying the *Iris-Versicolour* class from the *Iris-Virginica* classes.

The following steps were taken:

1. Insert three initial rules into an EFuNN structure and test on the entire data set. The linguistic realisation of the rules are:
   - If Petal-Length is Short and Petal-Width is Short then class is *Iris-Setosa*.
   - If Petal-Length is Medium and Petal-Width is Medium then class is *Iris-Versicolour*.
• If Petal-Length is Long and Petal-Width is Long then class is Iris-Virginica.

2. Train the EFuNN using a single pass on 105 random instances of the data set and test it on 45 random instances.

3. Extract rules from the EFuNN, insert into a brand new empty EFuNN structure, then re-test on the 45 instances

For the above task an EFuNN was initialised with an architecture of:

- 4(inputs)-3(inputMFs per input)-3(initial ruleNodes)-3(outputMFs per output)-3(outputs)
and then populated with the three rules,

The parameters for this EFuNN were Sensitivity threshold \( S_{thr} = 0.95 \) and an Error threshold \( Err = 0.03 \). The Learning Rate (\( L_r \)) for both the first and second layers of the EFuNN was \( L_r = 0.01 \).

A rule node was deemed \( Old \) at 90 data examples and pruned after 90 epochs.

This experiment was repeated and the results of these runs are displayed in Table 4.2.

<table>
<thead>
<tr>
<th>Run</th>
<th>RMS (Training)</th>
<th>No. Rule Nodes</th>
<th>Accuracy</th>
<th>Rules Extracted</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.0005</td>
<td>86</td>
<td>100.0%</td>
<td>15</td>
<td>95.56%</td>
</tr>
<tr>
<td>2</td>
<td>0.0007</td>
<td>87</td>
<td>91.11%</td>
<td>16</td>
<td>93.33%</td>
</tr>
<tr>
<td>3</td>
<td>0.0007</td>
<td>80</td>
<td>91.11%</td>
<td>13</td>
<td>91.11%</td>
</tr>
<tr>
<td>4</td>
<td>0.0004</td>
<td>83</td>
<td>93.33%</td>
<td>15</td>
<td>97.78%</td>
</tr>
<tr>
<td>5</td>
<td>0.0007</td>
<td>86</td>
<td>95.56%</td>
<td>17</td>
<td>91.11%</td>
</tr>
<tr>
<td>6</td>
<td>0.0005</td>
<td>85</td>
<td>97.78%</td>
<td>18</td>
<td>91.11%</td>
</tr>
<tr>
<td>7</td>
<td>0.0006</td>
<td>85</td>
<td>97.78%</td>
<td>13</td>
<td>93.33%</td>
</tr>
<tr>
<td>8</td>
<td>0.0004</td>
<td>84</td>
<td>95.56%</td>
<td>14</td>
<td>100.0%</td>
</tr>
<tr>
<td>9</td>
<td>0.0005</td>
<td>81</td>
<td>93.33%</td>
<td>16</td>
<td>86.67%</td>
</tr>
<tr>
<td>10</td>
<td>0.0006</td>
<td>84</td>
<td>97.78%</td>
<td>18</td>
<td>97.78%</td>
</tr>
<tr>
<td>AVG</td>
<td></td>
<td></td>
<td>95.33%</td>
<td></td>
<td>93.80%</td>
</tr>
<tr>
<td>STD</td>
<td></td>
<td></td>
<td>3.04%</td>
<td></td>
<td>4.03%</td>
</tr>
</tbody>
</table>

Table 4.2: Results from ten runs of the EFuNN on the Iris data set

Both the number of rule nodes before rule extraction and after rule extraction are shown along with their respective recall rates. For easier comparison the average performance (AVG) and standard deviation (STD) have been included.
Run 8 was the superior in terms of increasing the recall rate of the EFuNN using a smaller ruleset. The overall classification rate of the this run after Step 1 was completed and can be found in Table 4.3.

Table 4.3: EFuNN when tested on entire Iris data set after three rules inserted

| EFuNN Parameters: numMF=3, Sthr=0.95, Lr=0.01, Err=0.03, Old=90, Prune=90 |
| Classified as   | Iris Setosa | Iris Versicolour | Iris Virginica | Total Correct |
| Iris Setosa     | 15          | 0                | 0              |               |
| Iris Versicolour| 0           | 15               | 0              |               |
| Iris Virginica  | 0           | 8                | 7              |               |

37/45

Once Step 2 had been completed 84 rules were generated by the EFuNN and the overall classification rate of the EFuNN after this step can be found in Table 4.4.

Table 4.4: EFuNN when tested on 45 instances of Iris data set after completion of Step 2

| EFuNN Parameters: numMF=3, Sthr=0.95, Lr=0.01, Err=0.03, Old=90, Prune=90 |
| Classified as   | Iris Setosa | Iris Versicolour | Iris Virginica | Total Correct |
| Iris Setosa     | 15          | 0                | 0              |               |
| Iris Versicolour| 0           | 14               | 1              |               |
| Iris Virginica  | 0           | 1                | 14             |               |

43/45

Using the rule extraction algorithm 14 rules were extracted and are listed below in Table 4.5:

Table 4.5: The 14 rules extracted from the EFuNN

IF Sepal-Length is Medium(0.78) AND Sepal-Width is Long(1.00) AND Petal-Width is Short(0.86) AND Petal-Length is Short(0.75) THEN Iris-Setosa(1.00)
IF Sepal-Length is Medium(0.67) AND Sepal-Width is Medium(0.82) AND Petal-Width is Short(0.93) AND Petal-Length is Short(0.92) THEN Iris-Setosa(1.00)
IF Sepal-Length is Short(0.94) AND Sepal-Width is Medium(0.64) AND Petal-Width is Short(0.89) AND Petal-Length is Short(0.92) THEN Iris-Setosa(1.00)
IF Sepal-Length is Short(0.56) AND Sepal-Width is Short(0.73) AND Petal-Width is Medium(0.68) AND Petal-Length is Medium(0.83) THEN Iris-Versicolour(1.00)
IF Sepal-Length is Medium(0.72) AND Sepal-Width is Short(0.55) AND Petal-Width is Medium(0.89) AND Petal-Length is Medium(1.00) THEN Iris-Versicolour(1.00)

(continued on next page)
An analysis of the extracted rules revealed that only three rules were required to successfully classify *Iris-Setosa*. Whereas the same number of rules were required to classify *Iris-Versicolour*. Finally the other eight rules were used to classify *Iris-Virginica*. The results of the rule set generated from the rule extraction algorithm was consistent with the original hypothesis proposed at the beginning of the experiment.

When Step 3 had completed the classification rate of the EFuNN after this step can be found in Table 4.6.

**Table 4.6:** EFuNN when tested using the 14 extracted rules

| EFuNN Parameters: numMF=3, thr=0.95, Lr=0.01, Err=0.03, Old=90, Prune=90 |
|---------------------|---------------------|---------------------|---------------------|---------------------|
| **Classified as**   | Iris Setosa         | Iris Versicolour    | Iris Virginica      | **Total Correct**   |
| Iris Setosa         | 15                  | 0                   | 0                   | 15                  |
| Iris Versicolour    | 0                   | 15                  | 0                   | 15                  |
| Iris Virginica      | 0                   | 0                   | 15                  | 45                  |

The result of 45/45 (100%) is a good classification rate given the small number of rules. However, on average accuracy rate from the extracted rule sets was 93% and when compared
to the work of Setiono and Liu (1996) who achieved a classification accuracy of 97.33% on the same data set using 3 rules, the EFuNN did not compare well either in classification accuracy or the number of rules extracted.

To justify the results obtained from our study, we compared the results obtained from the EFuNN when rule extraction was employed against three other classifiers. The $k$-means classifier as it is a well documented traditional statistical classifier, the MLP as it is also neural network, and the SVM since it is the current state-of-the-art in classifiers. The PRTools Pattern Recognition Toolbox for MATLAB (Duin; 2000) was used as the implementation for these three classifiers.

The parameters used for each classifier were

- **$k$-means classifier**: 3 clusters

- **MLP**: 10 hidden nodes, error goal = 0.02, momentum = 0.95, learning rate = 0.01, trained for 50 epochs

- **SVM**: A polynomial kernel of degree 4.

and again our multiple-run method was used. The results of the classifiers can be found in Table 4.7. For easier comparison the average performance (AVG) and standard deviation (STD) have been included.

<table>
<thead>
<tr>
<th>Run</th>
<th>EFuNN</th>
<th>K-means</th>
<th>MLP</th>
<th>SVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>95.56%</td>
<td>95.56%</td>
<td>100.00%</td>
<td>95.56%</td>
</tr>
<tr>
<td>2</td>
<td>93.33%</td>
<td>93.33%</td>
<td>93.33%</td>
<td>95.56%</td>
</tr>
<tr>
<td>3</td>
<td>91.11%</td>
<td>91.11%</td>
<td>95.56%</td>
<td>91.11%</td>
</tr>
<tr>
<td>4</td>
<td>97.78%</td>
<td>91.11%</td>
<td>93.33%</td>
<td>88.89%</td>
</tr>
<tr>
<td>5</td>
<td>91.11%</td>
<td>97.78%</td>
<td>100.0%</td>
<td>97.78%</td>
</tr>
<tr>
<td>6</td>
<td>91.11%</td>
<td>95.56%</td>
<td>93.33%</td>
<td>95.56%</td>
</tr>
<tr>
<td>7</td>
<td>93.33%</td>
<td>95.56%</td>
<td>95.56%</td>
<td>95.56%</td>
</tr>
<tr>
<td>8</td>
<td>100.00%</td>
<td>97.78%</td>
<td>97.78%</td>
<td>97.78%</td>
</tr>
<tr>
<td>9</td>
<td>86.67%</td>
<td>95.56%</td>
<td>95.56%</td>
<td>95.56%</td>
</tr>
<tr>
<td>10</td>
<td>97.78%</td>
<td>97.78%</td>
<td>97.78%</td>
<td>97.78%</td>
</tr>
<tr>
<td>AVG</td>
<td>93.80%</td>
<td>95.11%</td>
<td>96.22%</td>
<td>95.11%</td>
</tr>
<tr>
<td>STD</td>
<td>4.03%</td>
<td>2.52%</td>
<td>2.57%</td>
<td>2.92%</td>
</tr>
</tbody>
</table>
The results obtained indicated that the EFuNN performed at a comparable level to that of the other classifiers. In terms of overall classification accuracy, the MLP was the best approach (96.22%) with $k$-means and SVM exhibiting the same average performance (95.11%). However there were minor variation in the standard deviations of the results from the three other classifiers.

4.8 Analysis of the method

Based on the two experiments conducted it appears that this rule extraction and rule insertion method described works well. The advantage of using this rule extraction and rule insertion method on an EFuNN allows:

- For a compact EFuNN structure to be maintained.
- To realise rules within the EFuNN to aid in the validation of this connectionist-based model.
- To use natural language constructs to formulate the initial rule set for the EFuNN.

However there are two main identifiable disadvantages to this method:

- Rule aggregation can only occur when the EFuNN has finished training. It would be better if aggregation of the ruleset occurred during the training of the EFuNN. Thus the EFuNN reverts to an off-line learning model.
- Although the EFuNN has the facility for multi-pass presentation of the training data examples, this EFuNN may not necessarily reduce the RMS error of the EFuNN or increase its classification accuracy.

Finally, since Setiono and Liu (1996, 1997) reported that their work performs very well at extracting rules that achieve high classification accuracy and are very human comprehensible, it would be prudent to compare and contrast the different rule extraction algorithms and type of rules that are generated by the EFuNN and their work.
In the case of the NeuroRule and NeuroLinear rule extraction algorithms, the process starts out by training one or more three layer neural networks on the data which are all initialised with random values between \([-1.0 \ldots 1.0]\). Each neural network is trained and pruned for a number of epochs until it falls below a level of classification accuracy. The criteria for pruning nodes is based on the magnitude of their weights. Once this step has been completed, one of the trained neural networks will then be selected as an optimal neural network that only contains a minimal number hidden nodes and connections between the input and output nodes that enabled it to maintain a high classification accuracy.

The training algorithm of the EFuNN used in this thesis, however, only trains one network using a single pass of the data set. The criteria for pruning rule nodes is based on how recently the rule node has been activated and not based on its activation value. No attempt is made to delete connections between the input layer, rule layer, and output layer. This is because of the complexity of the nodes in the fuzzy input layer and the fuzzy output layer, each of which may contain up to five membership functions attached to each input node or output node.

The process of rule generation also differs between the work of Setiono and Liu (1996, 1997) and the EFuNN rule extraction algorithm. In their work the activation values of the hidden nodes are discretised using the Chi2 algorithm and those discretised activation values are used to generate rule components that describe the output of the neural network. The other part of the rule components describe the discretised activation values of the neural network in terms of the inputs patterns of the data set. These two types of rule components are then combined together to form the final set of rules.

On the other hand, the EFuNN rule extraction algorithm generates rules which are governed by two thresholds that specify, for each rule, a level of degree of confidence for those rules that will initially be candidates for extraction. When rule aggregation occurs, clusters of similar rules are merged together to form a rule that accounts for a region within the feature space defined by the data set.

Depending on the values for these thresholds, a different number of rules will be extracted. Low values for the thresholds will result in more rules being extracted in which each rule only accommodates a small region of the feature space of the data set. Conversely high values for the
thresholds will result in fewer rules in which each rule accommodates a much larger region of the feature space of the data set.

Finally there is one major difference between the type of rules that are extracted. The work of Setiono and Liu (1996, 1997) extracts crisp rules which contain only the attributes from the training data set that are relevant to classifying a particular class. In Setiono and Liu (1996) the authors reported that NeuroRule generated only three rules to classify the Iris data set to an accuracy of 97.33% when tested on new data and these rules are presented below.

- **Rule 1:** If Petal-length $\leq$ 1.9, then *Iris setosa*

- **Rule 2:** If Petal-length $\leq$ 4.9 and Petal-width $\leq$ 1.6, then *Iris versicolor*

- **Default Rule:** *Iris virginica.*

The EFuNN rule extraction algorithm, however, generates fuzzy rules in which all attributes from the original data set are present in each rule. The rule presented below, for example, is one of the rules used used to classify instances of Iris setosa from Subsection 4.7.2. This rule represents a region that accommodates a cluster of data instances of the same class.

- **IF** Sepal-Length is Medium(0.78) AND Sepal-Width is Long(1.00) AND Petal-Width is Short(0.86) AND Petal-Length is Short(0.75) **THEN** Iris-Setosa(1.00)

These two examples highlight the difference between the comprehensiblity of the two different types of rules. Because NeuroRule and NeuroLinear extract crisp rules, each rule can be easily mapped back to instances of the original data set. Rules extracted from the EFuNN, however, are fuzzy rules and require that the fuzzy rules be fired by a fuzzy inference engine in parallel to classify specific instances of the original data set. Furthermore the number of membership functions for each antecedent and consequent of the rule must also known in advance to make sense of each rule.
4.9 Conclusion

The objective of this chapter is to describe connectionist-based methodologies for knowledge engineering. Three main areas were discussed: on-line learning, knowledge based methods, and adaptive learning mechanisms. In light of the identified deficiencies of these methods, the ECOS framework has been proposed to extend these connectionist-based methods for the production of CBIIS.

To manage existing knowledge and facilitate the acquisition of new knowledge in ECOS, a rule insertion and rule extraction scheme has been proposed and applied to a benchmark prediction task and a benchmark classification task. In both cases, their performance does favourably well compared to other prediction and classification models.

With respect to the rule extraction algorithm, it is fast and able to elicit a compact set of rules. And as the EFuNN learns new instances of data, the extracted rules easily adapt to the new state of the EFuNN. Rule insertion into an EFuNN has the added benefit of having an EFuNN populated with either crisp or fuzzy rules depending on the problem to be modelled.

In order to fully test the rule insertion and rule extraction algorithms, Chapter 5 and Chapter 6 apply these techniques to the problem of classification of data sets sourced from the horticultural domain. Each chapter both seeks to justify the use of EFuNN as the classification mechanism and also compare its performance against other connectionist and statistically based classifiers.
Chapter 5

CBIIS for Image Analysis and Recognition: An application in horticulture

5.1 Introduction

The domain of image recognition involves many tasks that seek to produce robust machine-based classifiers which can identify some characteristic of an image. Many techniques have been developed to facilitate this including noise reduction, image enhancement, image resizing, or feature extraction (Gonzalez and Woods; 1993).

This traditional process normally starts with sourcing a set of images that are scanned in by imaging software, more recently taken by a digital camera, or fed directly into a computer in real time by a digital video camera. The image recognition system then processes these images and the resulting output of the system identifies the presence or absence of an object within the image that the system has been taught to identify.

But one area where these particular image recognition systems have infrequently been applied is in horticulture. More specifically we consider those tasks that support an orchardist’s decision-making process when they are growing apples destined for the international market.

There are two problem areas where an image recognition system could be applied:

- The task of identifying the type of damage that has been inflicted in apple orchards. Many pests have the potential to damage both the fruit and leaves on the apple tree. Each insect or insect group features specific characteristics that allow it to be identified through the
damage it does to the fruit and/or leaves as by-products of its activities e.g. feeding. Once the insect has been successfully identified, the appropriate insecticide can be applied to eradicate the pest.

- Determining the rate at which browning occurs within Braeburn apples when they have been harvested and in storage. Storage of apples is via Controlled Atmosphere (CA) storage. CA storage does not improve fruit quality, but it can slow down the loss of quality after harvest. Successful CA storage begins by harvesting fruit at its proper maturity. The ability to detect the rate of loss of quality governs the decision concerning whether the apples are to remain in CA storage or immediately processed for export.

There could be many reasons why CBIIS systems for image recognition have not been adopted in the past as a solution to these problems. One explanation could be that there is a lack of technology that was small and mobile enough to be taken around an orchard or into a coolstore. Only recent advances in digital cameras and laptop computers have paved the way for orchardists and coolstore managers to utilise this technology.

For example, a web-based system decision-support to complement the task of pest identification is called BugKEY (Wearing; 1999) and was developed by HortResearch New Zealand. Unfortunately, this tool is only of use to those orchardists that have on-line access to this facility.

The robustness and accuracy of the image recognition system is also of concern. If the system incorrectly identifies the pest damage then a wrong choice of pesticide to apply may be made. This will result in targeting the wrong pest and causing an imbalance to the natural ecosystem that exists within the orchard, and apples of poorer quality may develop since the pest was not killed off. Therefore a desirable level of accuracy would need to be very high for a system to be of satisfactory performance and instill some confidence within the orchardist about the recognition system.

But the overriding problem is the lack of adequately sized data sets for the construction of an image recognition system. As stated in Chapter 1, only within the past few years has data been collected on pest damage both in terms of images of the pest damage itself and other related data such as the number of pests present in the orchard. If there is limited data then this drastically
limits our choice of what model one would use to construct a robust image recognition system because traditional machine learning, neural network, or statistically-based classifiers are built on the assumption that there is enough data with which to train, validate, and test the model.

This chapter addresses these issues by describing one solution to these problems by proposing a novel image recognition architecture based on an EFuNN that incrementally learns on-line and in real-time, is adaptive, and employs many of the ECOS principles as described in Chapter 4 but differs from these principles in that this version of the EFuNN does not create new inputs or new outputs nor has any unsupervised learning capacity. It does, however, have supervised learning where new connections and rule nodes are created.

Comparison of this architecture against other connectionist and statistical classification models is also included to validate the choice of EFuNN for this difficult task.

The rest of the chapter is composed of two main sections. Section 5.2 describes an experiment in using this architecture for the purpose of analysis and identification of pest damage to apples. Two experiments are conducted. One applies CBIR for the purpose of accessing similar images of pest damage from an image database. The objective of this experiment is to justify our choice of wavelets for our feature representation. We then use the same wavelet features in a novel image recognition model for pest identification.

Section 5.3 then uses the same model to determine the rate at which internal browning of Braeburn apples in CA storage occurs. Here we also use a small data set but highlight the benefits of using such a model to determine what week most of the browning occurs. This is only possible through the rule extraction method described in Section 4.6 of Chapter 4.

5.2 Analysis of images of pest damage to apples

Image data of pest damage collected as a pilot study by HortResearch New Zealand was used as the basis for research into three pests that are prevalent in Central Otago orchards namely the leafroller, codling moth, and appleleaf curling midge. There were a total of 96 images, all in colour, taken at different orientations, lighting conditions, and sometime contained clusters of fruit on the tree rather than individual fruit. Furthermore the damage to the fruit itself was
of varying size and shape. Figure 5.1, Figure 5.2, and Figure 5.3 show examples of the type of images that were taken. There were 19 images taken of appleleaf curling midge damage, 34 images of codling moth damage, and 43 images of leafroller damage. Appleleaf curling midge and leafroller damage can be caused to both the fruit and the leaves of the tree and both exhibit similar but subtly different damage. In the case of codling moth damage only the fruit is damaged.

![Figure 5.1: Examples of codling moth damage](image1)

![Figure 5.2: Examples of appleleaf curling midge damage](image2)

![Figure 5.3: Examples of leafroller damage](image3)

Recognising this range of images is a complex task and requires that a set of important features be extracted that can then be used for classification of the damage. In Chapter 2, wavelets were discussed as a suitable data transformation, analysis, and feature extraction technique for complex images.
As an image analysis technique, wavelets offer some advantages over the traditional filtering or analysis methods that have been developed in the past such as the Gabor filter (Weldon and Higgins; 1994) or the DFT (Misiti et al.; 2000). Because they can be easily adapted to analyse most 1-Dimensional (1D) or 2-Dimensional (2D) signals and due to their multiresolution characteristics, important local information about the signal can either be extracted or examined.

Wang et al. (1996) successfully used Daubechies wavelets (Daubechies; 1988, 1990) for image analysis and comparison of natural images and applied them the problem of finding similar images in a large image databases by using a combination of wavelet coefficients extracted from the images and a novel image indexing algorithm (Wang et al.; 1997). Here the authors stated that the reason why wavelets were used was that they could characterise the colour variations over the spatial extent of the image that can provide semantically meaningful image analysis. The technique used by these authors is split into two main parts: 1) Image database generation and 2) Image querying.

### 5.2.1 Image database generation

To create the database Algorithm 4 generates the set of wavelet coefficients extracted from each image. Coefficients resulting from the wavelet transformation of each colour component are contained in the upper-left 8x8 corner of each transform matrix, $W_C,(1:16,1:16)$. They represent the lowest frequency band of the image which corresponds to the spatial location of objects within the image. The higher frequency band represents texture and variation of the colour within the image.

This results in feature vectors of dimensionality:

- Three 16x16 matrices that contain the wavelet coefficients as a result of the 4 layer 2D transform over the R/G/B space of the image.
- Three 1x1 matrices that contain the standard deviations for each 16x16 matrix.
- Three 8x8 matrices that contain the wavelet coefficients as a result of the 5 layer 2D transform over the R/G/B space of the image.
Algorithm 4: Algorithm to generate the wavelet coefficients. Adapted from Wang et al. (1998)

Input:
A set of images, $I_1, I_2, \ldots, I_n$ where $N$ is the number of images

Output:
A set of $N$ wavelet coefficients and standard deviations, $WC$, resulting from the wavelet transformation

1. $max = 255$ for the 24 bit colour images we are working with. Wang et al. (1998) reported that computing this colour space for the image has the effect of reducing the sensitivity of contrast variations within the image;

for $j=1$ to $N$ do

1. Resize each image to 128x128 preserving the aspect ratio to reduce the amount of distortion of the image;
2. $I_i$ = Resize($I_i$, 128x128);

3. Convert the image into three 128x128 colour space matrices denoted as $C_1$, $C_2$, and $C_3$ respectively based on the Red/Green/Blue (R/G/B) components of the image using the following formulae;
4. $[R,G,B]$ = ConvertToRGB ($I_i$);
5. $C_1$ = ($R+G+B)/3$;
6. $C_2$ = ($R+ (max-B))/2$;
7. $C_3$ = ($R+2*(max-G)+B)/4$;

8. Apply a 2D 4-Layer wavelet transform using a Daubechies wavelet to each of the three colour space matrices;
9. for $i = 1$ to $3$ do
10. $W_{C_i}(1:128,1:128)$ = Daubechies ($4,C_i$); end

11. Extract a submatrix of size 16 x 16 from each $W_{C_i}$ matrix;
12. for $i = 1$ to $3$ do
13. $W_{16x16C_i}(1:16,1:16)$ = $W_{C_i}(1:16,1:16)$; end

14. Computer the standard deviations of each $W_{C_i}(1:8,1:8)$ matrix;
15. for $i = 1$ to $3$ do
16. $\sigma_{C_i}$ = StDev ($W_{C_i}$); end

17. Apply a 2D 5-Layer wavelet transform using a Daubechies wavelet to each of the three colour space matrices;
18. for $i = 1$ to $3$ do
19. $W_{C_i}(1:128,1:128)$ = Daubechies ($5,C_i$); end

20. Extract and store a submatrix of size 8x8 from the upper-left corner of each resulting matrix from the wavelet transform;
21. for $i = 1$ to $3$ do
22. $W_{8X8C_i}(1:8,1:8)$ = $W_{C_i}(1:8,1:8)$; end

23. Store these wavelet coefficients and standard deviations for each image in the database;
24. $WC_j$ = [$W_{16x16C_{1j}},\sigma_{C_{1j}},W_{8X8C_{1j}}$]; end
5.2.2 Image querying

To retrieve the related images, Algorithm 5 is employed.

The whole search is based on the image semantics, thus images of a similar type will be selected and displayed. For example, given a query image of a wind surfer, other images of wind surfers will be selected and displayed. Results from their work also suggested that images of similar damage caused by a particular pest could also be retrieved using this method.

We then coded the algorithm in the product MATLAB (http://www.mathworks.com) using a combination of the Image Processing Toolbox (Thompson and Shure; 2001) for the image rescaling and the UVI 300 Wavelet Toolbox (Sánchez et al.; 1996) for the wavelet transformation of the scaled images.

After all 96 images had been examined Figure 5.4(a) shows the query image while Figure 5.4(b) shows the retrieved images in the database. Each image displayed has the Euclidean distance from the query image displayed above it. As a second example Figure 5.5 shows a second query submitted to the database on leafroller damage.

![Query Image](image1.png)

(a) Query Image

![Retrieved Images](image2.png)

(b) Retrieved Images

Figure 5.4: Query image and retrieved images of codling moth damage
Algorithm 5: Algorithm to query the image database. Adapted from Wang et al. (1998)

Input:
A database of images and their wavelet coefficients, \( I_1', I_2', \ldots, I_n' \) where \( n \) is the number of images
A query image \( I \)

Output:
A set of six images from the image database that are similar to the query image \( I \)

1. \( iCandidates = \{ \} \);
2. \( iCandidateCounter = 0 \);
3. Preprocess the query image using Algorithm 4;
4. \([W16x16C1,3, σ, W8X8C1,3] = PreProcess (I)\);
5. Select the wavelet coefficients used for matching the query image;
6. for \( i = 1 \) to \( n \) do
7. \( W_{C,i,1} = W16x16C_i(1 : 8, 1 : 8); \)
8. \( W_{C,i,2} = W16x16C_i(1 : 8, 9 : 16); \)
9. \( W_{C,i,3} = W16x16C_i(9 : 16, 1 : 8); \)
10. \( W_{C,i,4} = W16x16C_i(9 : 16, 9 : 16); \)
11. end
12. Compare the standard deviations of the query image with all those of the images in the database;
13. for \( i = 1 \) to \( n \) do
14. \( [σ_{C,i,1}] = GetStandardDeviations (I'_i); \)
15. if \( ((σ_{c1} > (100/(100 - percent) * σ_{c1})) \) or \( (percent < 100 * (σ_{c1} - σ'_{c1})/σ_{c1})) \) or \( ((σ_{c2} > (100/(100 - percent) * σ_{c2})) \) or \( (percent < 100 * (σ_{c2} - σ'_{c2})/σ_{c2})) \) and \( ((σ_{c3} > (100/(100 - percent) * σ_{c3})) \) or \( (percent < 100 * (σ_{c3} - σ'_{c3})/σ_{c3})) \) then
16. Select the image wavelet coefficients from the database as a potential candidate;
17. \( iCandidates ← iCandidates + I'_i; \)
18. \( iCandidateCounter = iCandidateCounter + 1; \)
19. end
20. end
21. for \( j = 1 \) to \( iCandidateCounter \) do
22. Select the same wavelet coefficients for each image in the image database;
23. \([W16x16C,j,3, σ, W8X8C,j,3] = SelectCoefficients (iCandidates_j); \)
24. for \( i = 1 \) to \( 3 \) do
25. \( W_{C,i,1} = W16x16C_i(1 : 8, 1 : 8); \)
26. \( W_{C,i,2} = W16x16C_i(1 : 8, 9 : 16); \)
27. \( W_{C,i,3} = W16x16C_i(9 : 16, 1 : 8); \)
28. \( W_{C,i,4} = W16x16C_i(9 : 16, 9 : 16); \)
29. end
30. Compare the wavelet coefficients of the candidate images with the wavelet coefficients of the query image using a weighted distance function \( \text{Euclid} \) where \( \sum_{i=1}^{3} ((W_{C,i} - W'_{C,i})^2) \) is the Euclidean distance between vectors \( W_{C,i} \) and \( W'_{C,i}; \)
31. \( w_{Dist_i} = w_{1,1} \sum_{i=1}^{3} (w_{c_i} \text{Euclid}(W_{C,i,1} - W'_{C,i,1})) + w_{1,2} \sum_{i=1}^{3} (w_{c_i} \text{Euclid}(W_{C,i,2} - W'_{C,i,2})) + \)
32. \( w_{2,1} \sum_{i=1}^{3} (w_{c_i} \text{Euclid}(W_{C,i,2} - W'_{C,i,2})) + w_{2,2} \sum_{i=1}^{3} (w_{c_i} \text{Euclid}(W_{C,i,2} - W'_{C,i,2})); \)
33. Sort \( w_{Dist} \) in ascending order based on their Euclidean distance;
34. \( w_{DistSorted} ← \text{Sort} (w_{Dist}); \)
35. Display the images that correspond to the first six values in \( w_{DistSorted}; \)
Results obtained from the method described by (Wang et al.; 1996, 1997) for comparing these images worked well, but there were some instances where their approach failed. For example when invoking a query on leafroller damage as in Figure 5.5, not all the resulting images were of leafroller damage. This is because the Euclidean distance measure does not take into account the subtle differences between the different types of damage, especially when it comes to comparing appleleaf curling midge damage and leafroller damage which exhibit similar characteristics. As a consolation, this approach serves as an attractive method for better fruit image management. CBIR systems such as these can assist in image analysis, and data management for horticulture research.

![Query Image](image1.png)

![Retrieved Images](image2.png)

**Figure 5.5:** Query image and retrieved images of leafroller damage
5.2.3 Using an EFuNN to identify pest damage to apples

Section 5.2 demonstrated that wavelets could be used as an appropriate form of feature representation for the analysis of natural images. The only problem was the method by which the wavelet coefficients generated from each image were compared. The next logical step in this investigation would be to see if an appropriate classification model could be constructed to identify the images themselves and indicate what pest caused the damage. Using the same wavelet coefficients, a model could be trained and subsequently used as an image recognition engine for fast real-time image classification. Furthermore, if the same model could have knowledge extracted from it then this would provide useful information as to which pest damage was more difficult to detect.

Traditional classification models such as the MLP or SVM would be likely contenders as a solution to this problem. However, models such as these offer no facility for knowledge extraction. The FuNN and NEFCLASS possess the ability for knowledge extraction through rules, but these models are constructed based on the assumption that there is adequate data with which to train and test them. If this assumption does not hold then the viability of using these models is reduced.

This is exactly the case with the problem at hand. As stated in Section 5.2 only 96 images were available for analysis. With such a limited data set, it would be difficult to build a classification model that exhibited good generalisation and performance. Furthermore the aforementioned models also do not learn in an on-line and real-time fashion as images are acquired and stored.

There is also the issue of the development of the apple as it grows over time. This characteristic is crucial when identifying the damage caused by the pest as the damage changes over time. Therefore an appropriate classifier is one that would be able to adapt to the external environment from which the images are being sourced. Using an EFuNN with its incremental learning algorithm would be more suited to this task since acquired images could be continuously submitted to the model to increase its classification accuracy as the apples develop or as the damage to the apple becomes more widespread.

Images like those in Figures 5.1, 5.2, and 5.3, although having one apple or leaf in each
shot, contain many examples of the pest damage on it. Because we are interested in the damage without too much regard to the apple itself or leaf itself, it would be logical to extract the pest damage from the image and use these examples to train the EFuNN. This increases the number of training examples that the EFuNN can learn from. Testing of the EFuNN proceeds by comparing the trained EFuNN against complete images of damaged apples.

Our approach to solving the problem combines image segmentation and wavelet transformation to produce a set of feature vectors used to train our classifier. The colour image segmentation method devised by Ma and Manjunath (1997); Deng and Manjunath (2001) is used to segment regions which correspond to the damage on the apple. Once the images have been segmented, the identified regions are then manually extracted from the image. Figure 5.6(a) displays an apple with stem damage whilst Figure 5.6(b) shows the output of the segmentation algorithm. Here the algorithm has segmented the inner part of the stem. This is the damaged part of the apple.

![Original Image](image1.jpg) ![Segmented Image](image2.jpg)

(a) Original Image  (b) Segmented Image

**Figure 5.6:** Original and segmented fruit damage

When the same image segmentation algorithm is applied to leaf damage, the algorithm conveniently segments the damaged region from the rest of the leaf. Figure 5.7(a) is an example of leafroller damage and Figure 5.7(b) indicates the segmented regions of this image. The top segment is the damage on the leaf. This segment will be used as the training example for our model.
Each colour image of the pest damage is resized to a 32x32 image and separated into three matrices corresponding to the original images’ Red, Green, and Blue colour components. A 4-Layer 2D Daubechies wavelet transformation (DWT) (Daubechies; 1990) is then applied to each colour component resulting in a 32x32 matrix. A sub-matrix of 16x16 is then extracted from the top left hand quartile of each colour component matrix resulting in 256 wavelet coefficients. The three 16x16 colour component sub-matrices are then combined to form a feature vector of 768 coefficients. Figure 5.8 outlines this process.

For example Figure 5.9 is an example of the output of this process. The leftmost image
is that of codling moth stem damage. This is the segmented damage taken from the image in Figure 5.6(b). After transforming this image using the method described in Figure 5.8 three sets of 32x32 wavelet coefficients, one for each of the Red, Green, and Blue colour components are generated. The middle image depicts the 32x32 set of wavelet coefficients for the Red colour component. The rightmost image is the 16x16 submatrix extracted from this 32x32 matrix and forms part of the feature vector the classifier is trained on.

![Red component](image1.png) ![32x32 coefficients](image2.png) ![16x16 sub-matrix](image3.png)

**Figure 5.9:** Output of the process in Figure 5.8

The justification for using the wavelet transformation is related to its multi-resolution analysis of an image, which in turn allows for the detection of the discontinuities resulting from the presence of pest damage on the apple or leaf.

The output vector for each data example consisted of six different vectors:

- **[0 0 0 0 0]** if there was no damage on the image.
- **[1 0 0 0 0]** if the image was fruit damage inflicted by an appleleaf curling midge.
- **[0 1 0 0 0]** if the image was leaf damage inflicted by an appleleaf curling midge..
- **[0 0 1 0 0]** if the image was fruit damage inflicted by a codling moth.
- **[0 0 0 1 0]** if the image was fruit damage inflicted by a leafroller
- **[0 0 0 0 1]** if the image was leaf damage inflicted by a leafroller.

Using the previously described segmentation method produced 160 segments of pest damage. Table 5.1 is a break down of the distribution of these image segments.
Table 5.1: Distribution of pest damage examples

<table>
<thead>
<tr>
<th>Pest</th>
<th>Fruit</th>
<th>Leaf</th>
</tr>
</thead>
<tbody>
<tr>
<td>Appleleaf-curling midge</td>
<td>10</td>
<td>33</td>
</tr>
<tr>
<td>Codling Moth</td>
<td>44</td>
<td></td>
</tr>
<tr>
<td>Leafroller</td>
<td>29</td>
<td>44</td>
</tr>
</tbody>
</table>

To gain some appreciation of the complexity of the problem. PCA was performed on the data and the first two eigenvectors used as X and Y coordinates for the purpose of visualising the data set to identify any clusters that may correspond to similar pest damage. The results are displayed in Figure 5.10. From this visualisation we can conclude that there appear to be some clusters within this data set. For example appleleaf curling midge damage and leafroller leaf damage appear to exist within the same data space. Likewise codling moth fruit and leafroller fruit damage also share similar data space. However the distance between clusters is very small and there is a lot of overlap between the clusters. Therefore this data set provides a challenge to any classification model.

![PCA projection of the pest image data set onto the first 2 eigenvectors](image_url)

Figure 5.10: PCA projection of pest data projected onto the first two eigenvectors
5.2.4 Training and testing of the EFuNN

To train and test our EFuNN we applied the multiple-run method for this experiment. To train the EFuNN 140 image segments were randomly selected from the total pool of 160. Using this method guaranteed, for each of the five classes, the same number of training examples were selected from the data set. This method also was used to test the generalisation of the EFuNN to damage it had not been trained on. The multiple-run method was applied ten times.

Selection of the parameters for the EFuNN were based on the assumption that there was a need to maximise the number of rule nodes generated by the EFuNN so as to retain as many unique image templates that would give the best performance for the EFuNN. Therefore the parameters for the EFuNN were: Number of membership functions = 3, sensitivity threshold (SThr) = 0.95, learning rate (lr) = 0.25, error threshold (Err) = 0.01. The single-pass learning mode was used to train the EFuNN. The version of the EFuNN used in this experiment was the version described in Chapter 4 and implemented in MATLAB. The architecture of the EFuNN was:

- 768(inputs)-3(inputMFs per input)-0(initial ruleNodes)-3(outputMFs per output)-5(outputs)

Once the EFuNN was trained, it was tested on all 96 full sized images.

Because of the limited size of the training set, a new method for testing the EFuNN on the original set of 96 images needed to be developed. The solution involved combining traditional image filtering techniques with a classifier creating an adaptive image filter for the purpose of image recognition.

Here use a 32x32 window to perform a scan over the original image. The window is initially positioned at the top leftmost position of a 128x128 image and the output of each 32x32 window of the image is then converted to a set of wavelets using the scheme outlined in Section 5.2.1 that generates a set of 768 coefficients which are then passed to the classifier. The window is then shifted along by a predefined number of pixels and another set of wavelet coefficients generated. This process is repeated until the 32x32 window has completely covered the entire image.

The choice of size of the window was governed by finding an appropriate compromise between speed and time to process the entire image. Having a small window would incur large time
delays before the system responded with an output as the window would need to be shifted many
times over the image to completely cover it. If the window was too large, then subtle information
related to pest damage would be lost. We choose to have a 32x32 window that was shifted 16
pixels along the image. This meant the system would have to process 64 separate segments for
each image. By creating an overlapping window where we offset the shift by 24 pixels both in
the horizontal and vertical directions we were able to decrease the number of segments to 49
resulting in a marked speedup. The coefficients generated from each window became the input
to our classification model which can then identify the presence of a specific type of damage on
the apple or leaf and in Figure 5.11 we outline this process.

\[ \begin{array}{c|c|c|c}
128 & 32 & 32 & 128 \\
32 & 32 & 32 & \\
128 & & & \\
\end{array} \]

Figure 5.11: The image recognition process

Once the EFuNN was trained, it was tested on all 96 full sized images using this method.
As the 32x32 subwindow was passed over the image, the contents of the subwindow were trans-
formed into a set of 768 wavelet coefficients and passed to the EFuNN. If there was pest damage
present within this sub-window, the EFuNN would output a high value close to 1.0. And if there
was no pest damage the EFuNN would report a low value close to 0. The overall selection of the
pest damaged caused to the image would be taken from the highest output for a particular class
of insect damage.

For example if an image of codling moth damage was being scanned by this method, the
output of the EFuNN would be \([0 \ 0 \ 0.95 \ 0 \ 0]\) where there was a presence of this type of damage
and \([0 \ 0 \ 0.01 \ 0 \ 0]\) where there was not. The system would store the outputs of the EFuNN until
the system had scanned the entire image. The vector with the highest output would then be
compared to a known output vector for that entire image.

Table 5.2 shows the result of the testing of the EFuNN on all ten runs. With an average recognition rate of 74%, this indicated that slightly less than one in four testing examples were incorrectly classified.

Table 5.2: Results from ten runs of the EFuNN

<table>
<thead>
<tr>
<th>Run</th>
<th>RMS (Training)</th>
<th>No. Rule Nodes</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.0045</td>
<td>84</td>
<td>73.95%</td>
</tr>
<tr>
<td>2</td>
<td>0.0023</td>
<td>68</td>
<td>75.00%</td>
</tr>
<tr>
<td>3</td>
<td>0.0029</td>
<td>84</td>
<td>72.91%</td>
</tr>
<tr>
<td>4</td>
<td>0.0028</td>
<td>86</td>
<td>74.79%</td>
</tr>
<tr>
<td>5</td>
<td>0.0045</td>
<td>83</td>
<td>71.85%</td>
</tr>
<tr>
<td>6</td>
<td>0.0023</td>
<td>68</td>
<td>76.04%</td>
</tr>
<tr>
<td>7</td>
<td>0.0034</td>
<td>80</td>
<td>75.00%</td>
</tr>
<tr>
<td>8</td>
<td>0.0012</td>
<td>68</td>
<td>72.91%</td>
</tr>
<tr>
<td>9</td>
<td>0.0013</td>
<td>82</td>
<td>76.04%</td>
</tr>
<tr>
<td>10</td>
<td>0.0045</td>
<td>86</td>
<td>73.95%</td>
</tr>
<tr>
<td>AVG</td>
<td></td>
<td></td>
<td>74.23%</td>
</tr>
<tr>
<td>STD</td>
<td></td>
<td></td>
<td>1.390%</td>
</tr>
</tbody>
</table>

To determine why the level of accuracy was on average 74% a detailed investigation of the extracted ruleset from run 6 indicated that 1 rule was used to classify appleleaf curling midge fruit damage, 10 rules to classify appleleaf curling midge leaf damage, 17 rules to classify codling moth damage, 15 rules to classify leafroller fruit damage, and 25 rules to classify leafroller leaf damage. We can conclude that in order to classify leafroller leaf damage is a more difficult task hence the large number of rules used to accommodate these training examples.

And codling moth damage, with 17 rules, the easier to identify. The results of the extracted rules are summarised in Table 5.3.

Table 5.3: Analysis of the rule set from the best performing EFuNN run

<table>
<thead>
<tr>
<th>Pest Damage</th>
<th>No. Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>Appleleaf curling midge fruit damage</td>
<td>1</td>
</tr>
<tr>
<td>Appleleaf curling midge leaf damage</td>
<td>10</td>
</tr>
<tr>
<td>Codling moth damage</td>
<td>17</td>
</tr>
<tr>
<td>Leafroller fruit damage</td>
<td>15</td>
</tr>
<tr>
<td>Leafroller leaf damage</td>
<td>25</td>
</tr>
</tbody>
</table>

The results obtained from this study indicated that the EFuNN had successfully learnt the
image segments and was then able to use these segments as a means of identifying the presence or absence of a particular type of pest damage on the apple or leaf. Using this approach, one would be able to construct a system to report on whether the fruit/leaf image is OK or if not, indicate the location of the damage.

Table 5.4 shows the confusion matrix from run 6 of the system. Each category is listed column-wise with the classification outcome listed in each row of the table.

Table 5.4: Confusion matrix from run 6

<table>
<thead>
<tr>
<th>Classified as</th>
<th>alm-f</th>
<th>alm-l</th>
<th>cm</th>
<th>lr-f</th>
<th>lr-l</th>
</tr>
</thead>
<tbody>
<tr>
<td>alm-f</td>
<td>7</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>alm-l</td>
<td>0</td>
<td>8</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>cm</td>
<td>0</td>
<td>0</td>
<td>28</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>lr-f</td>
<td>2</td>
<td>0</td>
<td>7</td>
<td>16</td>
<td>0</td>
</tr>
<tr>
<td>lr-l</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>15</td>
</tr>
<tr>
<td>Total</td>
<td>74/96</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

When a misclassification was made, the EFuNN reported a sensible result. For example when the EFuNN was tested on images of appleleaf curling midge leaf damage (alm-l), it misclassified it as leafroller leaf damage (lr-l). The same occurred with fruit damage. Codling moth fruit damage (cm) was sometimes misclassified as appleleaf curling midge fruit damage (alm-f) or leafroller fruit damage (lr-f). In most cases there were few misclassifications.

5.2.5 Comparison between the EFuNN model and other classification models

To justify the results obtained from our study, we compared the results obtained from the EFuNN using three other classifiers. The $k$-means classifier as it is a well documented traditional statistical classifier, the MLP as it is also a neural network, and the SVM since it is the current state-of-the-art in classifiers. The PRTools Pattern Recognition Toolbox for MATLAB (Duin; 2000) was used as the implementation for these three classifiers.

Parameter selection for the $k$-means classifier was based primarily on how the $k$-means classifier was tested. Using the number of rule nodes generated by the EFuNN on the training set then governed how many clusters this classifier would be initialised with.
Similarly parameters selection for the MLP classifier was based primarily on examining the number of data examples to learn but more importantly looking at their dimensionality. A visual inspection of the PCA for the data set would give some indication as to what parameters should be used. In this case high momentum was chosen was there were few data data examples to learn and so we had to maximise the chance of reducing the overall error of the MLP.

Finally the parameter selection for the SVM classifier was based primarily on the available kernel functions for the implementation of the SVM in the PRTools Pattern Recognition Toolbox. At the time the experiments were conducted this implementation did not have a sufficient range of kernels to choose from. It was decided that the polynomial kernel appeared to be the most applicable with the degree of polynomial based on the visualisation of the PCA of the data set.

In all cases the parameters used for each classifier were also selected to optimise their recognition performance. Therefore for this comparison the parameters for the three classifiers were:

- **k-means classifier**: 68 prototypes since this is the same number of rule nodes for the best run of the EFuNN.
- **MLP**: 10 hidden nodes, error goal = 0.02, momentum = 0.95, learning rate = 0.01, and trained for 200 epochs.
- **SVM**: A polynomial kernel of degree 4.

Using the same training data sets created for the previous experiment, we applied each classifier to this task. For testing the classifiers, the same method for extracting 32x32 windows from the original image was used, and the feature vector presented to the classifier.

When testing the k-means classifier we adopted a new strategy for its use. Since there were only a small number of training examples for each class, then a variant of the “winner-take-all” concept from ART was employed. The steps in the classification algorithm are outlined below:

1. Apply k-means clustering to the entire training data set using \( M \) prototypes where \( M \) is the number of rule nodes for the best run of the EFuNN.
2. For each prototype, count the number of samples for each output class that are assigned to
this prototype. The output class with the highest count of samples is then associated with this prototype.

3. Find out how many prototypes have been assigned to each output class and classify the 32x32 window to the class with the most number of prototypes assigned to it.

The rationale behind this strategy was based on examining the visualisation in Figure 5.10 where the PCA projection revealed considerable amount of overlap between the classes. Using the above method would select a class that had the most number of prototypes assigned to it based on the number of samples for each class that appear in the 32x32 window. The results of testing the classifiers can be found in Table 5.5.

<table>
<thead>
<tr>
<th>Run</th>
<th>EFuNN</th>
<th>k-means</th>
<th>MLP</th>
<th>SVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>73.95%</td>
<td>53.12%</td>
<td>36.45%</td>
<td>55.20%</td>
</tr>
<tr>
<td>2</td>
<td>75.00%</td>
<td>57.29%</td>
<td>19.79%</td>
<td>56.25%</td>
</tr>
<tr>
<td>3</td>
<td>72.91%</td>
<td>55.20%</td>
<td>34.37%</td>
<td>53.12%</td>
</tr>
<tr>
<td>4</td>
<td>74.79%</td>
<td>60.41%</td>
<td>35.41%</td>
<td>56.25%</td>
</tr>
<tr>
<td>5</td>
<td>71.85%</td>
<td>53.12%</td>
<td>37.50%</td>
<td>52.08%</td>
</tr>
<tr>
<td>6</td>
<td>76.04%</td>
<td>56.25%</td>
<td>30.20%</td>
<td>55.20%</td>
</tr>
<tr>
<td>7</td>
<td>75.00%</td>
<td>52.08%</td>
<td>38.54%</td>
<td>53.12%</td>
</tr>
<tr>
<td>8</td>
<td>72.91%</td>
<td>56.25%</td>
<td>34.37%</td>
<td>51.04%</td>
</tr>
<tr>
<td>9</td>
<td>76.04%</td>
<td>55.20%</td>
<td>36.45%</td>
<td>53.12%</td>
</tr>
<tr>
<td>10</td>
<td>73.95%</td>
<td>53.12%</td>
<td>22.91%</td>
<td>55.20%</td>
</tr>
<tr>
<td>AVG</td>
<td>74.23%</td>
<td>55.20%</td>
<td>32.60%</td>
<td>54.05%</td>
</tr>
<tr>
<td>STD</td>
<td>1.39%</td>
<td>2.50%</td>
<td>6.38%</td>
<td>1.80%</td>
</tr>
</tbody>
</table>

The results obtained indicated that the EFuNN compared favourably with the other three classifiers and on average outperformed them by a margin of 20%. This is an interesting result as one would expect the MLP to be good if not better than the EFuNN as it uses multi-pass training. One would expect that adding more layers or increasing the number of hidden nodes in the MLP may improve its performance but such experiments were not undertaken.

5.2.6 Discussion

From the results obtained from testing the EFuNN and its relative performance against three other classification models, it is evident that the EFuNN performs slightly better than the other
models but not at a level that could be regarded as significant. The rules extracted from the EFuNN reflected the distribution of the training data set, and confirm the original hypothesis that more rules would need to be evolved to delineate the appleleaf curling midge damage from the leafroller damage.

Accounting for the poor performance of the MLP could be explained by examining the nature of this connectionist model. Having a fixed number of hidden nodes limits the number of hyperplanes these models can use to separate the complex feature space shown in Figure 5.10. Selection of the optimal number of hidden nodes becomes one of trial and error. If these are too large then the ability for the MLP to generalise to new instances of images is reduced due to the possibility of over-learning the training examples.

Although the SVM was the best out of the three other classification models, it still had poor accuracy. This could be partly explained by one of the identified deficiencies of the SVM cited in Woodford (2001) as initially described by Tipping (1999) where they reported that there was no current method of finding the certainty associated with the decision of the SVM to classify a unseen data instance. Knowing this certainty may help to explain the poor performance of the SVM.

5.3 Classifying temporal image data using an EFuNN

Over the past 15 years artificial neural networks have been applied to the task of image recognition (Pan et al.; 2000; Valentin et al.; 1997). In most cases the images presented to the networks have had little or no temporal information embedded within them. The images normally fall into the category of inanimate objects. Although it has been shown in previous studies that these models can successfully recognise images of this type, the model’s ability to classify images with a temporal dimension is not as good.

Methods of determining this rate of change have been proposed including the use of Near Infrared Spectrometry (NIR) (Munro et al.; 1996; Munro; 1997). Traditional statistical techniques have been employed to analyse this data and the results obtained have been quite encouraging. An alternative approach was to use connectionist-based classifiers for this task and the results
compared to the statistical method (Kim et al.; 1997). This has proven to be a successful method.

Although this data collection procedure provides a wealth of information about the fruit, it is only based on a sample of different points around the fruit itself. Such a process may neglect some spatial characteristics of the fruit that are the result of a particular treatment programme such as its colour, shape, size, or blemishing.

Therefore, a visual inspection of the apple would be more desirable. The problem is that the browning occurs from within and by the time it appears on the surface of the apple, the damage has already been inflicted. One novel approach to solving this problem has been in the adoption of medical imaging techniques to provide more visual information with which to make this decision. One example is Magnetic Resonance Imagery (MRI), a non-destructive technique for the analysis of the human brain (Itti et al.; 1998).

Using this technology, it is then possible to perform non-destructive analysis of the fruit. MRI scans are taken at weekly intervals and then the rate of reduction of the quality of the fruit calculate from the images. Such analysis is time consuming and somewhat prone to human error. Image segmentation algorithms can automatically isolate the browned sections of the apples (Shareef et al.; 1997) but there is still the issue of determining the rate of degradation.

Neural networks are one potential solution to apply to this task, but their ability to handle temporal data is somewhat limited. Addressing this issue, recurrent neural networks and time delay neural networks have been successful but require many training examples before a satisfactory level of recognition accuracy is obtained. Current recurrent neural network models (Giles et al.; 2001; Petridis and Kehagias; 1996) do not fit comfortably into the arena of this problem as the loss of quality needs to be determined quickly and accurately using a small set of images of the same apple.

Based on the results obtained from the previous experiment we now apply the same method for the purpose of detecting browned regions within the MRI images of the apples. There are two objectives to this experiment. One is to provide justification for the choice of the method used in the previous experiment and the other is for the extraction of knowledge from the trained EFuNN in order to determine if there is any particular week where browning occurs more frequently within the apple.
5.3.1 Method

The data set comprised of 5 sets of 5 MRI images, a total of 25 MRI images in all. Each image set was of the same apple taken at after 0 days, after 7 days, after 14 days, after 21 days, and after 28 days. In order to appreciate the complexity of the problem, Figure 5.12 contains some of the MRI images along with their corresponding description in Table 5.6. Figure 5.12(a) is an image of an apple a5 at week one and no browning has occurred. However in Figure 5.12(e), the emergence of browning in apple a4 in week 5 is very apparent.

Figure 5.12: Subset of images used to test the classification system

<table>
<thead>
<tr>
<th>Filename</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a5-1</td>
<td>Apple 5 taken at 0 days.</td>
</tr>
<tr>
<td>a3-2</td>
<td>Apple 3 taken at 7 days.</td>
</tr>
<tr>
<td>a1-3</td>
<td>Apple 1 taken at 14 days.</td>
</tr>
<tr>
<td>a2-4</td>
<td>Apple 2 taken at 21 days.</td>
</tr>
<tr>
<td>a4-5</td>
<td>Apple 4 taken at 28 days.</td>
</tr>
</tbody>
</table>

Segmentation of the browned regions then occurred using the same segmentation method and manual extraction method as in the previous experiment. This process created the input feature vectors for the data set. The output vectors for the data set consisted of a vector containing 5 outputs with a value of either 1 or 0. A 1 in the output vector corresponded to the week in which the MRI scan of the apple was taken. For example if the MRI image of the apple was taken at 0 days then the output vector would be [1 0 0 0 0]. Those MRI images taken at 28 days had an output vector of [0 0 0 0 1]. A total of 38 image templates were created with the
distribution of images shown in Table 5.7.

Table 5.7: Distribution of MRI image examples

<table>
<thead>
<tr>
<th>Period</th>
<th>No. Image segments</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 Days</td>
<td>5</td>
</tr>
<tr>
<td>7 Days</td>
<td>5</td>
</tr>
<tr>
<td>14 Days</td>
<td>9</td>
</tr>
<tr>
<td>21 Days</td>
<td>9</td>
</tr>
<tr>
<td>28 Days</td>
<td>10</td>
</tr>
</tbody>
</table>

Figure 5.13 is the PCA projection of the MRI data. This visualisation indicates that there are five clusters that exist within the data set. Two of which would be easy to identify and three of which overlap with each other thus making this problem more difficult to solve.

5.3.2 Training and testing of the EFuNN

Using the data set described above, an EFuNN was initialised with the parameters of:

- 256(inputs)-2(inputMFs per input)-0(initial ruleNodes)-2(outputMFs per output)-5(outputs)

The parameters for the EFuNN were: Number of membership functions = 3, sensitivity threshold (sThr) = 0.95, learning rate (lr) = 0.01, error threshold (Err) = 0.01. The single-pass learning
mode was used to train the EFuNN.

To train the EFuNN, 33 randomly selected instances from the pool of 39 were selected where same number of training examples were selected for each of the five classes. By adopting this method, we could then test the generalisation capability of the EFuNN when tested on an entire image. This training/testing strategy was repeated using our multiple-run method.

Once the EFuNN was trained, it was tested on all 25 full sized images using this method. Table 5.8 shows the result of the testing of the EFuNN on all ten runs. With an average recognition rate of 62.80%, this method has attained a level of accuracy that was not as good as the pest damage study. This is obviously attributed to the size of the data set although the results are comparable to the previous study.

Table 5.8: Results from ten runs of the EFuNN over the MRI image data set

<table>
<thead>
<tr>
<th>Run</th>
<th>RMS (Training)</th>
<th>No. Rule Nodes</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.0</td>
<td>19</td>
<td>64.00%</td>
</tr>
<tr>
<td>2</td>
<td>0.0</td>
<td>19</td>
<td>68.00%</td>
</tr>
<tr>
<td>3</td>
<td>0.0</td>
<td>32</td>
<td>64.00%</td>
</tr>
<tr>
<td>4</td>
<td>0.0</td>
<td>33</td>
<td>64.00%</td>
</tr>
<tr>
<td>5</td>
<td>0.0</td>
<td>31</td>
<td>60.00%</td>
</tr>
<tr>
<td>6</td>
<td>0.0</td>
<td>33</td>
<td>68.00%</td>
</tr>
<tr>
<td>7</td>
<td>0.0</td>
<td>32</td>
<td>60.00%</td>
</tr>
<tr>
<td>8</td>
<td>0.0</td>
<td>33</td>
<td>60.00%</td>
</tr>
<tr>
<td>9</td>
<td>0.0</td>
<td>33</td>
<td>60.00%</td>
</tr>
<tr>
<td>10</td>
<td>0.0</td>
<td>31</td>
<td>60.00%</td>
</tr>
<tr>
<td>AVG</td>
<td></td>
<td></td>
<td>62.80%</td>
</tr>
<tr>
<td>STD</td>
<td></td>
<td></td>
<td>3.29%</td>
</tr>
</tbody>
</table>

All nineteen rules extracted from the best run are presented below in Table 5.9 but because of the large number of antecedents only the first and last antecedents of each rule are displayed.

Table 5.9: The 19 rules extracted from the EFuNN

<table>
<thead>
<tr>
<th>Rule 1</th>
<th>IF In1 is medium(0.98),..., AND In768 is medium(0.62) THEN browning at 0 days(1.00)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rule 2</td>
<td>IF In1 is medium(0.83),..., AND In768 is medium(0.64) THEN browning at 0 days(1.00)</td>
</tr>
<tr>
<td>Rule 3</td>
<td>IF In1 is medium(0.90),..., AND In768 is medium(0.68) THEN browning at 0 days(1.00)</td>
</tr>
<tr>
<td>Rule 4</td>
<td>IF In1 is high(0.60),..., AND In768 is low(0.56) THEN browning at 0 days(1.00)</td>
</tr>
<tr>
<td>Rule 5</td>
<td>IF In1 is high(0.71),..., AND In768 is low(0.61) THEN browning at 7 days(1.00)</td>
</tr>
</tbody>
</table>

(continued on next page)
Analysis of the best generated rule set indicated that there was an even distribution of rules over most of the time periods with the exception of the 7 day period. This is logical since the MRI images of the apples reflected little change in browning over that time period between day 7 and day 14.

To justify the results obtained from our study, we compared the results obtained from the EFuNN using the three other classifiers used in the previous study. The $k$-means classifier as it is a well documented traditional statistical classifier, the MLP as it is also neural network, and the SVM since it is the current state-of-the-art in classifiers. Again we use the PRTools Pattern Recognition Toolbox for MATLAB (Duin; 2000) was used as the implementation for these three classifiers.

The parameters used for each classifier were

- **$k$-means classifier**: 33 prototypes since this is the same number of rule nodes for the best run of the EFuNN.
- **MLP**: 10 hidden nodes, error goal = 0.02, momentum = 0.95, learning rate = 0.01, trained for 200 epochs
- **SVM**: A polynomial kernel of degree 4.
Using the same training data sets created for the previous experiment, we applied each classifier to this task. For testing the classifiers, the same method for extracting 32x32 windows from the original image was used, and the feature vector presented to the classifier. The results of testing the classifiers can be found in Table 5.10.

Table 5.10: Results from the other three classifiers

<table>
<thead>
<tr>
<th>Run</th>
<th>EFuNN</th>
<th>$k$-means</th>
<th>MLP</th>
<th>SVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>64.00%</td>
<td>52.00%</td>
<td>30.00%</td>
<td>28.00%</td>
</tr>
<tr>
<td>2</td>
<td>68.00%</td>
<td>56.00%</td>
<td>36.00%</td>
<td>40.00%</td>
</tr>
<tr>
<td>3</td>
<td>64.00%</td>
<td>52.00%</td>
<td>36.00%</td>
<td>40.00%</td>
</tr>
<tr>
<td>4</td>
<td>64.00%</td>
<td>56.00%</td>
<td>36.00%</td>
<td>40.00%</td>
</tr>
<tr>
<td>5</td>
<td>60.00%</td>
<td>56.00%</td>
<td>32.00%</td>
<td>40.00%</td>
</tr>
<tr>
<td>6</td>
<td>68.00%</td>
<td>56.00%</td>
<td>32.00%</td>
<td>36.00%</td>
</tr>
<tr>
<td>7</td>
<td>60.00%</td>
<td>52.00%</td>
<td>36.00%</td>
<td>48.00%</td>
</tr>
<tr>
<td>8</td>
<td>60.00%</td>
<td>52.00%</td>
<td>32.00%</td>
<td>40.00%</td>
</tr>
<tr>
<td>9</td>
<td>60.00%</td>
<td>56.00%</td>
<td>36.00%</td>
<td>48.00%</td>
</tr>
<tr>
<td>10</td>
<td>60.00%</td>
<td>52.00%</td>
<td>36.00%</td>
<td>40.00%</td>
</tr>
<tr>
<td>AVG</td>
<td>62.80%</td>
<td>54.00%</td>
<td>34.20%</td>
<td>40.00%</td>
</tr>
<tr>
<td>STD</td>
<td>3.29%</td>
<td>2.10%</td>
<td>2.39%</td>
<td>5.65%</td>
</tr>
</tbody>
</table>

The results obtained from using these three classifiers were poor compared to the performance of the EFuNN. This suggests the EFuNN copes well with data sets of very small sizes.

5.3.3 Discussion

Based on this initial experiment, it is clear that the EFuNNs performance was slightly better than the other three models. This again provides us with some more justification as to the use of the EFuNN as part of a more robust recognition image engine where there is very limited data.

In terms of the size of the architecture for the MLP, this was extremely large. This was because of the length of the input vector. Input vectors of this size require a significant amount of presentations of the training data for the MLP to successfully learn the mapping between the input vectors and the output vectors. In addition, the small number hidden nodes contained in the MLP were unable to represent the mapping between the inputs and outputs. Raising the number of hidden nodes increased the ability for the MLP to learn but create a structure that is an order of magnitude larger and thus will take even longer to train.
Secondly, there was only a small number of training images presented to the classifiers. The consequence of the small number of examples presented to the MLP, K-means, and SVM resulted in its inability to generalise to the unseen MRI image examples. Even the application of a modified version of the backpropagation learning algorithm for the MLP (Mor; 1977) did not result in better generalisation of the MLP. We would then postulate that an increase in the number of training examples would result in a better generalisation of the MLP. However the expense in obtaining the extra images may not be cost effective.

In conclusion it can be seen from the results of this experiment that the main advantage of the EFuNN over the other models was that the EFuNN was able to successfully classify the original images given only one presentation of each training image. This indicates that the EFuNN has the ability to store a better representation of the temporal nature of the data which led to slightly better performance in classifying the test images but again this would not be regarded as significant.

5.4 Summary and concluding remarks

This chapter has sought to describe a problematic area within horticulture produce management, that of visual inspection of damaged apples that governs a decision on what insecticide to spray and also determining the rate of browning of Braeburn apples using MRI images.

Through applying the EFuNN to these two problems we have identified the strengths and weaknesses of our approach when compared against other connectionist-based and statistical classifiers even with a limited number of training images each of which have high dimensionality. Using the rule extraction ability of the EFuNN also allowed some insight into the distribution of the rules used to classify unseen data instances.
Chapter 6

CBIIS for NIR data classification: A case study in horticulture

6.1 Introduction

In this chapter we introduce the concept of Near Infrared Reflectance (NIR) spectroscopy and its application to the evaluation of particular characteristics of specific species of fruit. Most work in this area has been to classify these characteristics using conventional statistical classifiers but recently connectionist-based systems have been also applied as an alternative because of their ability to realise a robust classification mechanism with desirable and high degrees of generalisation.

Like Chapter 5, we apply the EFuNN architecture to this classification problem and also compare its relative performance against other connectionist-based models and statistically-based classifiers.

This study also furthers the original contribution of the author by proposing that the EFuNN can be used as an appropriate technique for analysis of and knowledge discovery from NIR data in sources from the horticultural domain.

The organisation of this chapter is as follows: Section 6.2 introduces the technique of NIR analysis of fruit and how this data can be of use to those exponents of growing quality produce. It also covers what previous work has been conducted in this area of developing classifiers of this data. One experiment, discussed in Section 6.3, examines how the EFuNN can be applied
to the task of classifying Persimmon fruit and compares its performance against other contemporary classification techniques. A similar experiment to classify pine-needle phenotypes is also discussed in Section 6.4. Finally, in Section 6.5 we conclude with a summary of the experiments conducted along with some concluding remarks of the appropriateness of applying the EFuNN as a classification mechanism for this data domain and also its potential for knowledge-discovery from these types of data sets.

6.2 NIR Data Recognition

Visual inspection alone of a particular object may not necessarily indicate its characteristics. In Section 5.3 of Chapter 5, one non-destructive method, namely MRI was used to generate images of the internal browning of Braeburn apples, from which one could determine, with an appropriate method, the rate of browning of the apple. However such techniques can be rather cost intensive and therefore reduces the general applicability of this method.

One alternative method is to use NIR (Ozaki et al.; 2001). Here they measure the wavelength and intensity of the absorption of near-infrared light by a sample, be it a fruit or vegetable. NIR light spans approximately the 800 - 2500 nm (12,500 - 4000 cm\(^{-1}\)) range and is energetic enough to excite overtones and combinations of molecular vibrations to higher energy levels. NIR spectra is typically used for measurement of organic functional groups. This method is a non-destructive means of examining the internal features of an object that is far beyond a visual inspection of it.

NIR has been used extensively in the agricultural and food industries for the past thirty years. Geladi and Dåbakk (1995) covers several applications of how NIR is used. Although slowly accepted, recent advances in instrumentation and multivariate data analysis (chemometrics) have made NIR an attractive analytical method for use in the pharmaceutical industry, both as a qualitative and a quantitative method. Multivariate data analysis is especially necessary for near-infrared spectra because highly overlapping absorption bands, descending from weak overtone and combination bands, hamper the assignment of a signal to certain functional groups. The weakness of the absorption bands in the near-infrared region provides useful sampling advan-
tages with respect to the fundamental bands in the mid-infrared region. Even more, near-infrared spectra of samples can be collected through glass which opens the way for fast and non-invasive analysis. The convenient instrumentation of NIR spectroscopy compared to InfraRed (IR) spectroscopy makes it much more suitable for on-line monitoring and process control.

At a national level, this technique has also been used in the horticultural domain to determine the ripening characteristics of fruit. These characteristics, such as sugar, acid and moisture content can be measured using NIR. By comparing the information gained by NIR analysis with the physio-chemical properties of the fruit it is possible to relate specific NIR spectral features to desirable product attributes. Once this has been done NIR analysis will be able to be used as an objective measure of fruit quality. For example Munro et al. (1996); Munro (1997); Munro et al. (1997) used this form of data and developed a flexible visualisation environment for exploring NIR data of apples. They extended the work by Murakami et al. (1992, 1994) on analyses of NIR data of apples and pumpkins to examine how this information could then be used to manage fruit development and storage processes to maximise market acceptance.

Similar NIR data sets collected on kiwifruit spectra have been used as the basis of comparisons of different statistical and connectionist-based classifiers. In Kim et al. (1999, 2000) a comparative study was performed on spectra derived from kiwifruit berries that had six pre-harvest treatments to see which model more accurately classified this data set. The authors also demonstrated that the better feature extraction technique on this data set was when the Haar wavelet was applied. Results obtained from this study suggested that this work could be extended by comparing the relative performance of the EFuNN against other connectionist-based and statistically-based classifiers using NIR data. In addition, the EFuNN can also be used for knowledge discovery to find out some other features of the spectra which contribute to the overall classification of the fruit.
6.3 Classification of Persimmon Fruit

6.3.1 Background

The identification of desirable cultivars of fruit is an important process in maintaining the quality control process in fruit tree nurseries. New Zealand’s diverse climate has caused multiple imports of fruit that carry the same genotype and this has caused a problem where cultivars unsuited to commercial production have been propagated and planted. This has certainly been the case when growing the persimmon fruit.

Determining those persimmon that exhibit more desirable characteristics is a task that has normally been generally based on examining the morphology of the fruit. Recently Mowat and Holmes (2003) identified the need for a different method of analysing the fruit and used NIR spectroscopy on epikutlar leaf waxes of different cultivars (varieties) of persimmon. The data collected could then be used as input to a robust classifier that could be used for persimmon cultivar identification.

In their study they collected the leaves of 20 plants of each genotype and then applied a conventional technique to extract the epikutlar wax from each leaf that was then analysed using a NIR spectroscope and the readings recorded. The raw data was then transformed using standard PCA and the first 20 eigenvectors taken as the input to several different machine learning techniques, the best of which was the IBK algorithm (Aha et al.; 1995) which classified the data set at 98%.

Such data sets can naturally be applied to connectionist-based methods as well, but as opposed to the machine learning methods described in the article, no knowledge can easily be derived about which are the most important spectra to differentiate the cultivars. It would therefore be a useful task to apply a productive task to apply an EFuNN to determine if this method is as good as the accuracy of the machine learning methods. Moreover an appropriate CBIIS technique could also provide some insight into what specific parts of the spectra are more important when determining the genotype of the fruit and how many rules from the EFuNN have been associated with each genotype of the persimmon fruit.
6.3.2 Method

The same data set referred to in Mowat and Holmes (2003) was used as the basis for selecting the most appropriate connectionist-based model. 70 examples were used as the training data set and the other 40 used for the test data set. Mowat and Holmes (2003) noted that two of the five persimmon genotypes were so closely related that the spectra for them was combined resulting in 4 distinct persimmon genotypes to be classified. The distribution of the data set was 40 examples for persimmon genotype 1 (Pers1), 20 examples for persimmon genotype 2 (Pers2), 20 examples for persimmon genotype 3 (Pers3), and 30 examples for persimmon genotype 4 (Pers4).

Figure 6.1 is the PCA projection of the training data. This visualisation indicates that there are four easily discernible clusters with a small overlap between the Pers3 and Pers4.

![PCA projection of the Persimmon data set onto the first 2 eigenvectors](image)

Figure 6.1: PCA projection of persimmon training data projected onto first two eigenvectors

The output vectors for the data set consisted of a vector containing four outputs with a value of either 1 or 0. A 1 in the output vector corresponded to a particular persimmon genotype. For example if the persimmon genotype was of type 1 then the output vector would be \([1 \ 0 \ 0 \ 0]\). Those persimmon that had a genotype of type 4 had an output vector of \([0 \ 0 \ 0 \ 1]\).

We then applied the EFuNN to the task of classifying the persimmon data and obtained
the results using ten-fold cross validation. We then compared the performance of the EFuNN against similar connectionist models such as the FuNN and the MLP, and finally also compare these results against the SVM.

### 6.3.3 Training and testing of the EFuNN

An EFuNN in MATLAB was initialised with this architecture:

- 20(inputs)-2(inputMFs per input)-0(initial ruleNodes)-2(outputMFs per output)-4(outputs)

This experiment was repeated using our multiple-run method and the results of these runs displayed in Table 6.1. Both the number of rule nodes before rule extraction and after rule extraction are shown along with their respective recall rates. On average the average classification rate for the EFuNN before rule extraction was 87.80% and 87.00% after rule extraction.

<table>
<thead>
<tr>
<th>Run</th>
<th>RMS (Training)</th>
<th>No. Rule Nodes</th>
<th>Accuracy</th>
<th>Rules Extracted</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.024</td>
<td>17</td>
<td>85.00%</td>
<td>16</td>
<td>82.50%</td>
</tr>
<tr>
<td>2</td>
<td>0.026</td>
<td>20</td>
<td>92.50%</td>
<td>19</td>
<td>92.50%</td>
</tr>
<tr>
<td>3</td>
<td>0.024</td>
<td>16</td>
<td>90.00%</td>
<td>15</td>
<td>90.00%</td>
</tr>
<tr>
<td>4</td>
<td>0.023</td>
<td>18</td>
<td>95.50%</td>
<td>17</td>
<td>90.00%</td>
</tr>
<tr>
<td>5</td>
<td>0.026</td>
<td>20</td>
<td>87.50%</td>
<td>19</td>
<td>87.50%</td>
</tr>
<tr>
<td>6</td>
<td>0.025</td>
<td>18</td>
<td>82.50%</td>
<td>17</td>
<td>82.50%</td>
</tr>
<tr>
<td>7</td>
<td>0.024</td>
<td>21</td>
<td>87.50%</td>
<td>20</td>
<td>87.50%</td>
</tr>
<tr>
<td>8</td>
<td>0.025</td>
<td>15</td>
<td>87.50%</td>
<td>14</td>
<td>87.50%</td>
</tr>
<tr>
<td>9</td>
<td>0.024</td>
<td>18</td>
<td>95.00%</td>
<td>17</td>
<td>95.00%</td>
</tr>
<tr>
<td>10</td>
<td>0.023</td>
<td>16</td>
<td>75.00%</td>
<td>15</td>
<td>75.00%</td>
</tr>
<tr>
<td>AVG</td>
<td></td>
<td></td>
<td>87.80%</td>
<td></td>
<td>87.00%</td>
</tr>
<tr>
<td>STD</td>
<td></td>
<td></td>
<td>6.13%</td>
<td></td>
<td>5.74%</td>
</tr>
</tbody>
</table>

Run 9 was the superior in terms of maintaining the recall rate of the EFuNN even though it eliminated one rule. The confusion matrix of this run can be found in Table 6.2. This is comparable to the best classifier described in Mowat and Holmes (2003) where they reported a classification rate of 98.5%.
Table 6.2: EFuNN when tested on 40 instances of the persimmon data set

<table>
<thead>
<tr>
<th>Classified as</th>
<th>Pers1</th>
<th>Pers2</th>
<th>Pers3</th>
<th>Pers4</th>
<th>Total Correct</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pers1</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Pers2</td>
<td>0</td>
<td>9</td>
<td>1</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>Pers3</td>
<td>1</td>
<td>0</td>
<td>9</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>Pers4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>38/40</td>
</tr>
</tbody>
</table>

All seventeen rules are presented below in Table 6.3 but because each rule has 20 antecedents only the first and last antecedents of each rule are displayed.

Table 6.3: The 17 rules extracted from the EFuNN

1: IF In1 is MF2(0.87),...,AND In20 is MF2(0.59) THEN Pers1(1.00)  
2: IF In1 is MF2(0.91),...,AND In20 is MF2(0.65) THEN Pers1(0.98)  
3: IF In1 is MF2(0.86),...,AND In20 is MF2(0.65) THEN Pers1(1.00)  
4: IF In1 is MF2(0.89),...,AND In20 is MF2(0.63) THEN Pers1(1.00)  
5: IF In1 is MF1(0.90),...,AND In20 is MF2(0.81) THEN Pers2(1.00)  
6: IF In1 is MF1(0.91),...,AND In20 is MF1(0.88) THEN Pers2(1.00)  
7: IF In1 is MF2(0.58),...,AND In20 is MF1(0.69) THEN Pers2(1.00)  
8: IF In1 is MF1(0.87),...,AND In20 is MF2(0.57) THEN Pers2(1.00)  
9: IF In1 is MF1(0.88),...,AND In20 is MF2(0.97) THEN Pers2(1.00)  
10: IF In1 is MF1(1.00),...,AND In20 is MF2(0.61) THEN Pers2(1.00)  
11: IF In1 is MF2(0.50),...,AND In20 is MF2(0.67) THEN Pers3(1.00)  
12: IF In1 is MF2(0.66),...,AND In20 is MF2(0.52) THEN Pers3(1.00)  
13: IF In1 is MF2(0.73),...,AND In20 is MF2(0.62) THEN Pers3(1.00)  
14: IF In1 is MF1(0.52),...,AND In20 is MF2(0.68) THEN Pers4(1.00)  
15: IF In1 is MF2(0.72),...,AND In20 is MF2(0.73) THEN Pers4(1.00)  
16: IF In1 is MF2(0.67),...,AND In20 is MF2(0.58) THEN Pers4(1.00)  
17: IF In1 is MF2(0.72),...,AND In20 is MF2(0.65) THEN Pers4(1.00)  

An analysis of the extracted rules revealed two results. Firstly, there were a small number of rules associated with the Pers1 (4 rules). But the other more interesting finding was that a large number of rule nodes had been associated with Pers2 (6 rules). This would indicate a greater variability in the NIR spectra of these instances. For Pers3 (3 rules) the number of rules accommodating these instances are much smaller indicating a smaller degree of variability in these genotypes while Pers4 (4 rules) also indicated the greater degree of variability in the genotype and would also help to resolve for the overlap between these two classes.
We can also visualise the extracted rule set by using PCA to project it on to the test data. Figure 6.2 is a visualisation of the test data used in run 9.

**Figure 6.2:** PCA projection of persimmon testing data projected onto first two eigenvectors

Figure 6.3 shows the mapping of the rules fired by the EFuNN to this data set we can easily see which rule is associated with each data example.

**Figure 6.3:** PCA projection of rule nodes projected onto the testing data for run 9
6.3.4 Training and testing of the FuNN

A FuNN was then initialised with the parameters of:

- 20(inputs)-2(inputMFs per input)-10(ruleNodes)-2(outputMFs per output)-4(outputs)

Using our multiple run method the FuNN was trained on 70 random instances of the data set and tested on 40 random instances of the data set. The learning parameters for the FuNN were learning rate of 0.01, momentum constant of 0.8, terminating error of 0.001, and each instance of the FuNN was trained for 1000 epochs. The results of training and testing the FuNN are presented in Table 6.4.

Table 6.4: Results from ten runs of the FuNN on the Persimmon data set

<table>
<thead>
<tr>
<th>Run</th>
<th>RMS (Training)</th>
<th>FuNN Accuracy</th>
<th>EFuNN Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.030</td>
<td>62.50%</td>
<td>82.50%</td>
</tr>
<tr>
<td>2</td>
<td>0.008</td>
<td>85.00%</td>
<td>92.50%</td>
</tr>
<tr>
<td>3</td>
<td>0.180</td>
<td>72.50%</td>
<td>90.00%</td>
</tr>
<tr>
<td>4</td>
<td>0.008</td>
<td>80.00%</td>
<td>90.00%</td>
</tr>
<tr>
<td>5</td>
<td>0.011</td>
<td>82.50%</td>
<td>87.50%</td>
</tr>
<tr>
<td>6</td>
<td>0.008</td>
<td>80.00%</td>
<td>82.50%</td>
</tr>
<tr>
<td>7</td>
<td>0.008</td>
<td>77.50%</td>
<td>87.50%</td>
</tr>
<tr>
<td>8</td>
<td>0.010</td>
<td>87.50%</td>
<td>87.50%</td>
</tr>
<tr>
<td>9</td>
<td>0.008</td>
<td>97.50%</td>
<td>95.00%</td>
</tr>
<tr>
<td>10</td>
<td>0.007</td>
<td>92.50%</td>
<td>75.00%</td>
</tr>
<tr>
<td>AVG</td>
<td></td>
<td>81.00%</td>
<td>87.00%</td>
</tr>
<tr>
<td>STD</td>
<td></td>
<td>9.72%</td>
<td>5.74%</td>
</tr>
</tbody>
</table>

Compared to the EFuNN, the results of the FuNN in Table 6.4 were slightly worse achieving an average of 81% recall rate. The confusion matrix for the best run of the FuNN is shown in Table 6.5. Here the misclassification is between Pers3 and Pers4.
Table 6.5: FuNN when tested on 40 instances of the persimmon data set

<table>
<thead>
<tr>
<th>Classified as</th>
<th>Pers1</th>
<th>Pers2</th>
<th>Pers3</th>
<th>Pers4</th>
<th>Total Correct</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Pers1</strong></td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td><strong>Pers2</strong></td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td><strong>Pers3</strong></td>
<td>0</td>
<td>0</td>
<td>9</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td><strong>Pers4</strong></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

39/40

6.3.5 Training and testing of the MLP

The final connectionist model to be trained and tested on this data set was a MLP. It was created using the MATLAB Neural Network Toolbox (Demuth and Beale; 2001) and initialised with

- 20(inputs)-5(hidden nodes)-4(output nodes)

The scaled conjugate gradient algorithm (Moller; 1993) was used to train the network; a variation on the standard backpropagation learning rule (Rumelhart et al.; 1986a) to reduce the time required to train the MLP due to the high dimensionality of the data. The learning parameters for the MLP were learning rate of 0.01, momentum constant of 0.8, terminating error of 0.001, and each run of the MLP was trained for 1000 epochs.

Table 6.6: Results from ten runs of the MLP on the Persimmon data set

<table>
<thead>
<tr>
<th>Run</th>
<th>RMS (Training)</th>
<th>MLP Accuracy</th>
<th>EFuNN Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.120</td>
<td>92.50%</td>
<td>85.00%</td>
</tr>
<tr>
<td>2</td>
<td>0.120</td>
<td>92.50%</td>
<td>92.50%</td>
</tr>
<tr>
<td>3</td>
<td>0.180</td>
<td>72.50%</td>
<td>90.00%</td>
</tr>
<tr>
<td>4</td>
<td>0.170</td>
<td>70.00%</td>
<td>90.00%</td>
</tr>
<tr>
<td>5</td>
<td>0.260</td>
<td>50.00%</td>
<td>87.50%</td>
</tr>
<tr>
<td>6</td>
<td>0.188</td>
<td>75.00%</td>
<td>82.50%</td>
</tr>
<tr>
<td>7</td>
<td>0.187</td>
<td>92.50%</td>
<td>87.50%</td>
</tr>
<tr>
<td>8</td>
<td>0.186</td>
<td>72.50%</td>
<td>87.50%</td>
</tr>
<tr>
<td>9</td>
<td>0.179</td>
<td>75.00%</td>
<td>95.00%</td>
</tr>
<tr>
<td>10</td>
<td>0.189</td>
<td>75.00%</td>
<td>75.00%</td>
</tr>
<tr>
<td>AVG</td>
<td>0.131</td>
<td>76.75%</td>
<td>87.00%</td>
</tr>
<tr>
<td>STD</td>
<td>0.201</td>
<td>13.12%</td>
<td>5.74%</td>
</tr>
</tbody>
</table>

In Table 6.9 it can clearly been seen in the confusion matrix for the best pass that the MLP obtained a classification rate of 92.50%. However the average classification rate was only 76.75%. 

142
Table 6.7: MLP when tested on 40 instances of the persimmon data set

<table>
<thead>
<tr>
<th>Classified as</th>
<th>Pers1</th>
<th>Pers2</th>
<th>Pers3</th>
<th>Pers4</th>
<th>Total Correct</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pers1</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Pers2</td>
<td>1</td>
<td>8</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Pers3</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Pers4</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>9</td>
<td>37/40</td>
</tr>
</tbody>
</table>

Compared to the EFuNN and FuNN, this was the poorest performing connectionist model with most mis-classifications occurring between Pers2 and Pers4.

### 6.3.6 Training and testing of the SVM

To compare the performance of these connectionist models, the same data set was then used in a SVM that generated 4 polynomial kernels to classify the data. The results of testing the SVM over the ten runs are shown in Table 6.8.

Table 6.8: Results from ten runs of the SVM on the persimmon data set

<table>
<thead>
<tr>
<th>Run</th>
<th>RMS (Training)</th>
<th>SVM Accuracy</th>
<th>EFuNN Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.0</td>
<td>85.00%</td>
<td>85.00%</td>
</tr>
<tr>
<td>2</td>
<td>0.0</td>
<td>87.50%</td>
<td>92.50%</td>
</tr>
<tr>
<td>3</td>
<td>0.0</td>
<td>90.00%</td>
<td>90.00%</td>
</tr>
<tr>
<td>4</td>
<td>0.0</td>
<td>85.00%</td>
<td>90.00%</td>
</tr>
<tr>
<td>5</td>
<td>0.0</td>
<td>92.50%</td>
<td>87.50%</td>
</tr>
<tr>
<td>6</td>
<td>0.0</td>
<td>92.50%</td>
<td>82.50%</td>
</tr>
<tr>
<td>7</td>
<td>0.0</td>
<td>90.00%</td>
<td>87.50%</td>
</tr>
<tr>
<td>8</td>
<td>0.0</td>
<td>95.00%</td>
<td>87.50%</td>
</tr>
<tr>
<td>9</td>
<td>0.0</td>
<td>92.50%</td>
<td>95.00%</td>
</tr>
<tr>
<td>10</td>
<td>0.0</td>
<td>70.00%</td>
<td>75.00%</td>
</tr>
<tr>
<td>AVG</td>
<td></td>
<td>88.00%</td>
<td>87.00%</td>
</tr>
<tr>
<td>STD</td>
<td></td>
<td>7.14%</td>
<td>5.74%</td>
</tr>
</tbody>
</table>

With the average classification rate for the SVM of 88%, the performance of the EFuNN is nearly equivalent to this statistical model. The accuracy of the SVM was slightly higher but at the same time with higher variance. The results, however, are not statistically significant as these results were based on a single run. The results of the confusion matrix for the best run of the
SVM are contained in Table 6.9

Table 6.9: SVM when tested on 40 instances of the persimmon data set

<table>
<thead>
<tr>
<th>SVM Parameters: 4 polynomial kernels</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classified as</td>
</tr>
<tr>
<td>Pers1</td>
</tr>
<tr>
<td>Pers2</td>
</tr>
<tr>
<td>Pers3</td>
</tr>
<tr>
<td>Pers4</td>
</tr>
</tbody>
</table>

6.3.7 Discussion

The four models trained and tested with this NIR data set performed quite acceptably but the EFuNN came out on top and matched the performance run of the SVM. Again, as stated in Chapter 5, the main benefit of the EFuNN comes from its ability to create a good classifier with minimal training examples and much less training time than the MLP and FuNN but not as fast as the SVM.

6.4 Classification of pine-tree phenotypes

6.4.1 Problem

For a range of different plants, the leaf waxes have been shown to play an important role in environmental stress tolerance, insect, and pathogen resistance. In breeding programs, a limitation to the exploration of leaf waxes has been the lack of suitable rapid screening techniques to identify clones with desirable wax composition. Creating an appropriate classification model to assist in this task may have an application in *pinus radiata* breeding work.

To this end a pilot study was conducted by HortResearch Ruakura, Hamilton, New Zealand to collect data on the composition of pine needle wax that could then be used to classify the clone. Using a similar method as described in Section 6.3 to extract the wax, in this case, from
pine needles, a raw data set was collected of NIR data for each sample. Because of the high dimensionality of the data set generated, a Haar wavelet transformed was applied resulting in a data set where each sample was represented by 310 wavelet coefficients. There were 5 different clones to be classified and 50 spectra collected for each clone.

Figure 6.4 is the PCA projection of the training data. This visualisation indicates that although there are some appearance of clustering within this data set, it is not that obvious as there are many overlaps between the five clones.

![PCA projection of pine data onto the first 2 eigenvectors](image)

Figure 6.4: PCA projection of pine training data projected onto first two eigenvectors

6.4.2 Method

In order to generate the most appropriate connectionist-based model to classify these clones, the data set was split into 80% for training (200 examples) and 20% for testing (50 examples). We used the MLP, FuNN, and EFuNN as the connectionist models and the SVM as the statistical model for this experiment. The output vectors for the data set consisted of a vector containing 5 outputs with a value of either 1 or 0. A 1 in the output vector corresponded to a particular pine genotype. For example if the pine genotype was of type 1 then the output vector would be \([1\ 0\ 0\ 0\ 0]\). Those pine that had a genotype of type 5 had an output vector of \([0\ 0\ 0\ 0\ 1]\).


6.4.3 Training and testing of the EFuNN

An EFuNN in MATLAB was initialised with this architecture:

- 310(inputs)-2(inputMFs per input)-0(initial ruleNodes)-2(outputMFs per output)-5(outputs)

This experiment was repeated using our multiple-run method and the results of these runs displayed in Table 6.10. Both the number of rule nodes before rule extraction and after rule extraction are shown along with their respective recall rates. The average classification rate for the EFuNN before rule extraction was 95.60% and 94.80% after rule extraction.

Table 6.10: Results from ten runs of the EFuNN on the pine needle data set

<table>
<thead>
<tr>
<th>Run</th>
<th>RMS (Training)</th>
<th>No. Rule Nodes</th>
<th>Accuracy</th>
<th>Rules Extracted</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.0005</td>
<td>56</td>
<td>95.00%</td>
<td>17</td>
<td>98.00%</td>
</tr>
<tr>
<td>2</td>
<td>0.0007</td>
<td>55</td>
<td>95.00%</td>
<td>18</td>
<td>98.00%</td>
</tr>
<tr>
<td>3</td>
<td>0.0007</td>
<td>64</td>
<td>92.00%</td>
<td>13</td>
<td>92.00%</td>
</tr>
<tr>
<td>4</td>
<td>0.0004</td>
<td>83</td>
<td>94.00%</td>
<td>15</td>
<td>98.00%</td>
</tr>
<tr>
<td>5</td>
<td>0.0007</td>
<td>86</td>
<td>96.00%</td>
<td>17</td>
<td>92.00%</td>
</tr>
<tr>
<td>6</td>
<td>0.0005</td>
<td>85</td>
<td>98.00%</td>
<td>18</td>
<td>92.00%</td>
</tr>
<tr>
<td>7</td>
<td>0.0006</td>
<td>85</td>
<td>98.00%</td>
<td>13</td>
<td>94.00%</td>
</tr>
<tr>
<td>8</td>
<td>0.0004</td>
<td>84</td>
<td>96.00%</td>
<td>19</td>
<td>100.0%</td>
</tr>
<tr>
<td>9</td>
<td>0.0005</td>
<td>81</td>
<td>94.00%</td>
<td>16</td>
<td>86.00%</td>
</tr>
<tr>
<td>10</td>
<td>0.0006</td>
<td>84</td>
<td>98.00%</td>
<td>18</td>
<td>98.00%</td>
</tr>
<tr>
<td>AVG</td>
<td></td>
<td></td>
<td>95.60%</td>
<td></td>
<td>94.80%</td>
</tr>
<tr>
<td>STD</td>
<td></td>
<td></td>
<td>2.01%</td>
<td></td>
<td>4.34%</td>
</tr>
</tbody>
</table>

Because of the high classification rates produced, it would be more interesting to examine the worst run for the EFuNN to determine where the misclassifications occurred. Therefore all sixteen rules from the original EFuNN are presented in Table 6.11 from the worst run of the multiple-run process below but because each rule has 310 antecedents only the first and last antecedents of each rule are displayed.
An analysis of the rule set revealed some interesting results. Pine1, Pine2, and Pine3 had four rules allocated to classifying their respective classes but there was only one rule generated to classify Pine4 and two rules to classify Pine5.

When visualising the test data used on the worst run in Figure 6.5 we begin to see why there was only one rule generated to classify Pine4. The test data examples for Pine4 are much more widely dispersed over the PCA space than any of the other classes. Hence the EFuNN could only create one to accommodate all the data examples for this class. This would suggest that the genotype of Pine4 is much more varied than the other genotypes.
Figure 6.5: PCA projection of Pine testing data projected onto first two eigenvectors

Figure 6.6 shows the mapping of the rules fired by the EFuNN to this data set we can easily see which rule is associated with each data example.

Figure 6.6: PCA projection of rule nodes projected onto the testing data for run 9
As opposed to showing the confusion matrix for the best run. The output for the worst case, run 9, is shown. By examining Table 6.12, we begin to see where the misclassifications have occurred.

Table 6.12: EFuNN when tested on 50 instances of the pine needle data set

<table>
<thead>
<tr>
<th>Classified as</th>
<th>Pine1</th>
<th>Pine2</th>
<th>Pine3</th>
<th>Pine4</th>
<th>Pine5</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pine1</td>
<td>9</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>Pine2</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>Pine3</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td>1</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>Pine4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>7</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>Pine5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>43/50</td>
</tr>
</tbody>
</table>

Here the majority of the mis-classifications occur between Pine 3 to Pine 5. Examining the original PCA projection of the raw data set indicates that this is where most of the overlap between these classes occur. Hence the rules generate by the EFuNN for this particular run weren’t able to classify all the instances of the test data.

6.4.4 Training and testing of the FuNN

A FuNN was also created and trained on the pine needle data set. With its architecture of:

- 310(inputs)-2(inputMFs per input)-10(rule nodes)-2(outputMFs per output)-5(outputs)

Using our multiple-run method the FuNN was trained on 200 random instances of the data set and tested on 50 random instances of the data set. The learning parameters for the FuNN were learning rate of 0.01, momentum constant of 0.8, terminating error of 0.001, and each instance of the FuNN was trained for 1000 epochs.

Compared to the EFuNN, the results of the FuNN in Table 6.13 were much more varied than those obtained from the EFuNN. Ranging from 16% to 100% caused the average classification rate to be 69.80%. The confusion matrix for the worst performing run of the FuNN is shown in Table 6.14.
Table 6.13: Results from ten runs of the FuNN on the pine needle data set

<table>
<thead>
<tr>
<th>Run</th>
<th>RMS (Training)</th>
<th>FuNN Accuracy</th>
<th>EFuNN Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.340</td>
<td>40.00%</td>
<td>98.00%</td>
</tr>
<tr>
<td>2</td>
<td>0.320</td>
<td>96.00%</td>
<td>98.00%</td>
</tr>
<tr>
<td>3</td>
<td>0.400</td>
<td>100.0%</td>
<td>92.00%</td>
</tr>
<tr>
<td>4</td>
<td>0.018</td>
<td>94.00%</td>
<td>98.00%</td>
</tr>
<tr>
<td>5</td>
<td>0.017</td>
<td>90.00%</td>
<td>92.00%</td>
</tr>
<tr>
<td>6</td>
<td>0.340</td>
<td>40.00%</td>
<td>92.00%</td>
</tr>
<tr>
<td>7</td>
<td>0.020</td>
<td>88.00%</td>
<td>94.00%</td>
</tr>
<tr>
<td>8</td>
<td>0.020</td>
<td>92.00%</td>
<td>100.00%</td>
</tr>
<tr>
<td>9</td>
<td>0.340</td>
<td>42.00%</td>
<td>86.00%</td>
</tr>
<tr>
<td>10</td>
<td>0.390</td>
<td>16.00%</td>
<td>98.00%</td>
</tr>
</tbody>
</table>

**A VG** 69.80% 94.8%

**STD** 31.37% 4.34%

Table 6.14: FuNN when tested on 50 instances of the pine needle data set

<table>
<thead>
<tr>
<th>Classified as</th>
<th>Pine1</th>
<th>Pine2</th>
<th>Pine3</th>
<th>Pine4</th>
<th>Pine5</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pine1</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>6</td>
<td>0</td>
<td>16/50</td>
</tr>
<tr>
<td>Pine2</td>
<td>0</td>
<td>0</td>
<td>9</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Pine3</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>6</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Pine4</td>
<td>1</td>
<td>0</td>
<td>5</td>
<td>4</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Pine5</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>5</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Here the majority of the misclassifications occurred between Pine3 to Pine5. We would therefore infer from this table that the FuNN had difficulty in partitioning the classes compared to the EFuNN which managed to perform much better on the data set.

### 6.4.5 Training and testing of the MLP

A MLP was then initialised with the architecture of:

- 310(inputs)-10(hidden nodes)-5(outputs)

Using our multiple-run method the MLP was trained on 200 random instances of the data set and tested on 50 random instances of the data set. The learning parameters for the MLP were learning rate of 0.01, momentum constant of 0.8, terminating error of 0.001, and each instance...
of the MLP was trained for 1000 epochs. Table 6.15 shows the results of testing the MLP. On average, the classification rate for the MLP was 92.60%.

Table 6.15: Results from ten runs of the MLP on the pine needle data set

<table>
<thead>
<tr>
<th>Run</th>
<th>RMS (Training)</th>
<th>MLP Accuracy</th>
<th>EFuNN Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.021</td>
<td>92.00%</td>
<td>98.00%</td>
</tr>
<tr>
<td>2</td>
<td>0.016</td>
<td>90.00%</td>
<td>98.00%</td>
</tr>
<tr>
<td>3</td>
<td>0.207</td>
<td>80.00%</td>
<td>92.00%</td>
</tr>
<tr>
<td>4</td>
<td>0.022</td>
<td>96.00%</td>
<td>98.00%</td>
</tr>
<tr>
<td>5</td>
<td>0.021</td>
<td>96.00%</td>
<td>92.00%</td>
</tr>
<tr>
<td>6</td>
<td>0.020</td>
<td>100.00%</td>
<td>92.00%</td>
</tr>
<tr>
<td>7</td>
<td>0.024</td>
<td>92.00%</td>
<td>94.00%</td>
</tr>
<tr>
<td>8</td>
<td>0.019</td>
<td>92.00%</td>
<td>100.00%</td>
</tr>
<tr>
<td>9</td>
<td>0.150</td>
<td>88.00%</td>
<td>86.00%</td>
</tr>
<tr>
<td>10</td>
<td>0.019</td>
<td>100.00%</td>
<td>98.00%</td>
</tr>
<tr>
<td>AVG</td>
<td></td>
<td>92.60%</td>
<td>94.80%</td>
</tr>
<tr>
<td>STD</td>
<td></td>
<td>5.96%</td>
<td>4.34%</td>
</tr>
</tbody>
</table>

Table 6.16 suggests that this model had less problems with classifying the data set compared to the FuNN and was on par with the same classification rate of the EFuNN.

Table 6.16: MLP when tested on 50 instances of the pine needle data set

| MLP Parameters: Arch=310-10-5, Mom=0.8, Lr=0.01, Epochs=1000 |
|----------------|----------------|----------------|----------------|----------------|----------------|
| Classified as | Pine1 | Pine2 | Pine3 | Pine4 | Pine5 | Total |
| Pine1 | 10 | 0 | 0 | 0 | 0 | |
| Pine2 | 0 | 10 | 0 | 0 | 0 | |
| Pine3 | 9 | 0 | 0 | 0 | 1 | |
| Pine4 | 0 | 0 | 0 | 10 | 0 | |
| Pine5 | 0 | 0 | 0 | 1 | 10 | |

Examining the worst case, run 3, indicated that all the mis-classifications occurred within Pine3. This is a different result to that of the EFuNN and FuNN. We would then infer that the MLP was able to successfully partition all the other four classes but found difficulty with the Pine3 class.
6.4.6 Training and testing of the SVM

To contrast the performance of the connectionist models against a SVM classifier was generated on the training data set using a polynomial kernel of degree 4. When tested, on average 94.80% of the test instances were correctly classified as shown in Table 6.17 resulting in the best performance for this task with half the variability of the EFuNN.

Table 6.17: Results from ten runs of the SVM on the pine needle data set

<table>
<thead>
<tr>
<th>Run</th>
<th>RMS (Training)</th>
<th>SVM Accuracy</th>
<th>EFuNN Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.0</td>
<td>96.00%</td>
<td>98.00%</td>
</tr>
<tr>
<td>2</td>
<td>0.0</td>
<td>94.00%</td>
<td>98.00%</td>
</tr>
<tr>
<td>3</td>
<td>0.0</td>
<td>98.00%</td>
<td>92.00%</td>
</tr>
<tr>
<td>4</td>
<td>0.0</td>
<td>94.00%</td>
<td>98.00%</td>
</tr>
<tr>
<td>5</td>
<td>0.0</td>
<td>90.00%</td>
<td>92.00%</td>
</tr>
<tr>
<td>6</td>
<td>0.0</td>
<td>92.00%</td>
<td>92.00%</td>
</tr>
<tr>
<td>7</td>
<td>0.0</td>
<td>98.00%</td>
<td>94.00%</td>
</tr>
<tr>
<td>8</td>
<td>0.0</td>
<td>92.00%</td>
<td>100.00%</td>
</tr>
<tr>
<td>9</td>
<td>0.0</td>
<td>96.00%</td>
<td>86.00%</td>
</tr>
<tr>
<td>10</td>
<td>0.0</td>
<td>98.00%</td>
<td>98.00%</td>
</tr>
<tr>
<td>AVG</td>
<td>0.0</td>
<td>94.80%</td>
<td>94.80%</td>
</tr>
<tr>
<td>STD</td>
<td></td>
<td>2.85%</td>
<td>4.34%</td>
</tr>
</tbody>
</table>

Table 6.18 contains the confusion matrix from the worst performing SVM run, run 4. In all runs this model had problems classifying Pine5. Like the previous models the misclassification was against Pine3. However the EFuNN compensated for this problem by allocating more rule nodes to accommodate the examples for this class.

Table 6.18: SVM when tested on 50 instances of the pine needle data set

<table>
<thead>
<tr>
<th>SVM Parameters: 4 polynomial kernels</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classified as</td>
</tr>
<tr>
<td>Pine1</td>
</tr>
<tr>
<td>Pine2</td>
</tr>
<tr>
<td>Pine3</td>
</tr>
<tr>
<td>Pine4</td>
</tr>
<tr>
<td>Pine5</td>
</tr>
<tr>
<td>45/50</td>
</tr>
</tbody>
</table>
6.4.7 **Discussion**

Training and testing the pine data set on the classification models produced greater variance in the obtained results. Like the previous experiment in Section 6.3 the MLP and FuNN performed the poorest on this data set. A possible explanation for this would be the choice of the number of hidden nodes in the these respective architectures. However we would expect them to have equal if not better classification accuracy of the EFuNN as the training data set was presented to it numerous times. The SVM performed the best overall, with the performance of the EFuNN, FuNN, and MLP in descending order of performance. This could be due to the ability of the SVM to cope with data sets with very high dimensionality to create better hyperplanes that would better create regions between the different clones contained in the data set.

6.5 **Summary and concluding remarks**

In this chapter the EFuNN has been presented as a classifier for two different NIR data sets with the added advantage of rule extraction from its internal structure revealing some interesting results especially when an investigation of the poor performance of the classifier on the pine data set was made.

   Because of its on-line and incremental learning nature, samples that have been collected using this technology can easily and quickly be used to increase the accuracy of this model even with data examples that have high dimensionality.
Chapter 7

CBIIS for decision support: Adaptive Expert System design and implementation

7.1 Introduction

The traditional expert system, based on a fixed set of rules, have significantly contributed to the development of AI and intelligent engineering systems in the past two decades such as PARADISE (Liew et al.; 1994), MIDA (Domeshek et al.; 1994), and UMRAO (Gadwal et al.; 1991). But as the knowledge required to solve a task changes over time, so must its representation.

One of the main issues surrounding the architecture of expert systems is that their knowledge representation, normally in the form of rules, remains static. This “snapshot” of knowledge required, for example in a task in decision support, renders the expert system useful only if the rule-base is not required to change.

If the rule set is small and is designed to model a simple task then this deficiency poses no real problem. But when more complex tasks whose requirements justify a significantly larger rule-base or the problem is one in which the knowledge required changes over time then the application of a traditional expert systems architecture is not an appropriate choice.

Real-world problems, for example that occur in engineering, manufacturing, and orchard management, fit into this category. A successful application of an expert system in these areas would require one that operates in real-time and is on-line as these processes are inherently based on every changing requirements and knowledge in order to provide decision support. This task
is compounded as:

- The problem to be solved may require many disparate sources of data to be combined together to form the problem specification.

- The environment in which the expert system is operating constantly changes. Therefore it needs to adapt its rules and inference mechanisms to better model the dynamics of the problem domain.

Consequently more sophisticated methods and tools for building adaptive intelligent expert systems (IES) need to be developed. Such systems should be able to grow as they work, to build-up their knowledge and refine the model through interaction with the environment. This objective is part of the seven major requirements of the next generation intelligent systems that are discussed in Kasabov (1998a,b, 1999b, 2001b). To this end we propose an Adaptive Generic Expert System (AGES) design that factors in many of these requirements for the next generation of IES. AGES consists of a series of modules which are agent based and generated “on the fly” as they are needed to solve a problem. This design is extensible and can be applied to many different problem domains. Chapter 8 describes one application of AGES in the area of apple orchard management for New Zealand orchards.

### 7.2 Traditional Expert Systems

Section 3.4 of Chapter 3 introduced the architecture of a traditional expert system. Figure 7.1 graphically depicts its main components. Notice that the structure of the architecture is very rigid and makes the assumption that knowledge is acquired from experts through Knowledge Acquisition methods or techniques which are then represented as rules in the Knowledge Base. These rules obviously need to be complete enough to model all the possible instances within the problem domain. This may result in knowledge bases that contain large rule sets, redundant rules, or not enough rules to model the problem. And as a consequence may cause the system to fail or produce incorrect output, a contributing factor in the Frame problem (McCarthy and Hayes; 1969) and well documented in Morgenstern (1996).
7.3 The Adaptive Generic Expert System (AGES)

To address this major lack of functionality, a design of an expert system referred to as AGES is proposed. Figure 7.2 depicts this design that is composed of several main components:

- **An Intelligent Design Interface**: This component allows the user to specify a task to be performed by AGES.

- **An Expert Knowledge Base**: This component is the repository of the rules that stores the expert knowledge required to solve a task. The number of rules within this component alters as AGES learns to adapt itself to better model the problem domain.

- **The Data/Environment**: This is effectively an interface that AGES uses to acquire data from the environment. It may be a bridge to an on-line data-base that is frequently updated or time series data like the dynamic variability of stock market data.

- **Repository of Modules**: This component contains a repository of components that are used to solve the problem defined by the user, using rules from the expert knowledge base,
on data that is sourced from the environment. Different types of modules are used from simple data transformation techniques to neuro-fuzzy methods such as the EFuNN.

- **The Expert System**: Unlike traditional expert systems, this component is dynamically created and thus has an adaptable and extensible structure and is only in existence for as long as a problem is require to be solved.

- **Solution Interface**: This is where the output of AGES is presented to the user, the output of AGES may be in the form of rules, images/diagrams, or as text.

A session with AGES proceeds like so: The User specifies the initial problem parameters and task to be solved via the Intelligent Design Interface. Once the problem has been specified intelligent agents select Modules from the Repository of Modules that will form the basis of the solution to the problem. For example, if the task is a data transformation task, the only Modules selected are ones that will perform that task.

Modules that allow for the evolution of their rules, for example an EFuNN, may initially have no rules in them or may already be set up with rules from the Expert Knowledge Base. The Modules then combine the rules with the Data from the Environment to form the Adaptive Expert System. The Modules are then trained with the Data from the Environment. The rules may then be extracted for later analysis where they might be visualised to see how the rules map to the Data from the Environment or aggregated and re-inserted into new Modules for re-training or testing to create more optimal structure for the Modules.

Once the Modules have been trained, they are tested with new Data from the Environment and the results extracted. These then form the solution to the problem and may be further interpreted by another set of Modules which may be used to present the solution to the problem in an appropriate way. Throughout this process the Modules will dynamically adapt their rule set as the environment changes since the number and structure of rules is dependent on the data that is being presented to the Module. Modules are dynamically created, updated and connected to other Modules. Modules may be destroyed, if necessary, at a later stage of the operation of the AGES.
This more holistic approach enables a generic expert system architecture to be applied to a specific problem domain whose knowledge and data the defines the problem changes dynamically in a real-time environment. As the basis for knowledge representation in AGES is built around its rules, then it is necessary to discuss the methods used in knowledge management and acquisition through rules in AGES.

### 7.4 Rule Extraction in AGES

AGES uses rule extraction in two different ways: (1) to derive rules from a data set supplied as facts to AGES and (2) to generate from a trained modules such as an EFuNN. The type of rules that can be extracted from a data set fall into two different categories; crisp rules and fuzzy rules. Crisp rules would normally be employed in AGES if the Expert Knowledge Base, based on the OPS5 production system architecture (Forgy; 1981) is used as the rule representation. Fuzzy rules would be used as part of a EFuNN module or another fuzzy inference mechanism. To generate these different type of rules from data, we have developed a Crisp Rule Extraction Algorithm (CRE) and a Fuzzy Rule Extraction Algorithm (FRE).
7.4.1 Crisp Rule Extraction

CRE, presented in Algorithm 6, is a modification of the algorithm described in Kasabov (1996a) as initially created to generate rules suitable for a CLIPS environment and can be regarded as a set of four steps:

1. **Discretize the continuous valued data into subintervals.** In Kasabov (1996a, p. 83) the Iris benchmark data set was used as an example and the intervals for each attribute were partitioned based on the spread of values that attribute. The intervals were normally within the 0.0 to 0.99 range. This resulted in
   
   - 4 for sepal-length – [4.00, 4.99], [5.0, 5.99], [6.0, 6.99], [7 and above]
   - 3 for sepal-width – [2.00, 2.99], [3.00, 3.99], [4.00, 8.0]
   - 6 for petal-length – [1.00, 1.99], [2.00, 2.99], [3.00, 3.99], [4.00, 4.99], [5.00, 5.99], [6.00, 7.00]
   - 3 for petal-width – [0.0, 0.99], [1, 1.99], [2, 3]

2. **The data set is then discretised in a set of “representative templates (SRT)”**. After discretisation, the method generates a set of templates for each row that have a tuple containing each interval the value for the attribute falls into, the class the row belongs to and the number of instances in the original data set represented by this tuple.

3. “**A global probability vector (GPV)**”. This is a vector that is calculated from all of the different attribute values over the entire data set and represents the probability of which discretised tuple will be associated with each class in the data set.

   It was not clear in the original description of the algorithm how the intervals were calculated and whether or not they were computed manually or via some other algorithm. Also the SRT had an extra value that was the number of instances from the original data set that this tuple represented but the description of how this was used was not made clear in the original description.
**Algorithm 6:** Algorithm for crisp rule extraction from a data set

**Input:**
A set of data examples, \( D_1, D_2, \ldots, D_n \) where \( N \) is the number of data examples

\( \text{numInputs} \) is the the number of input attributes in the data set \( D \)

\( \text{numClasses} \) is the number of output classes in the data set \( D \)

\( I \) which is the number of intervals to discretise the input attributes into

**Output:**
A set of unique crisp rules \( R \)

1. **for** \( i = 1 \) to \( N \) **do**
   2. Discretise each data instance into a template, \( T \) using the method described in Kasabov (1996a, pp. 152–155);
   3. \( T_i = \text{Discretise}(T_i(1:\text{numInputs})) \);
   4. **end**

5. **for** \( i = 1 \) to \( N \) **do**
   6. Count the number of cases that have the same vector value (rule value);
   7. \( C_i = \text{CountDuplicateTemplates}(T_i) \);
   8. Create an unique rule with the number of duplicates assigned to it;
   9. \( R_i,\text{unique} = T_i \);
   10. \( R_i,\text{frequency} = C_i \);
   11. **end**

12. **for** \( i = 1 \) to \( \text{numClasses} \) **do**
   13. Count how many rules contribute to identifying each class;
   14. \( R_i,\text{frequency} = \text{FindUniqueRulesPerClass}(R) \);
   15. Eliminate those unique rules that do not contribute to 75% of the classification for each class in the data set;
   16. if \( R_i,\text{frequency}/R_{\text{frequency}} < 0.75 \) **then**
   17. Remove the rule \( R_i,\text{frequency} \) from \( R_i,\text{unique} \);
   18. **end**
   19. **end**

The modifications to the original algorithm were:

- Interval selection for each attribute was based on a parameter entered in by the user.

- The generated intervals were based on partitioning the data set and not the range of values for each input attribute that resulted in far fewer intervals. In Step 3 the input values for each data instance are discretised into a predefined number of intervals that creates a rule for the template. For example if an instance from the iris data set is \([5.1 \ 3.5 \ 1.4 \ 0.2 \ 1]\) where the first four values are the inputs and the last value is the class label then this instance is discretised using five intervals resulting in the rule template \([3 \ 2 \ 0 \ 0 \ 1]\). The class label is not discretised.
• Each generated tuple consisted of the discretised values for the input row and the output class.

• The number of occurrences for a particular tuple were used to determine the importance of the rule but was not represented as part of the tuple.

• Those tuples that did not contribute to 75% of the classification for each class in the data set were eliminated. This value acts like the confidence parameter when generating association rules (Agrawal et al.; 1996).

As an example the CRE algorithm implemented in MATLAB was applied to the benchmark data set for Iris classification (150 instances; Three classes - *Iris-Setosa*, *Iris-Versicolour*, and *Iris-Virginica*; Four attributes - Sepal Length (SL), Sepal Width (SW), Petal Length (PL), and Petal Width (PW)). Five intervals were chosen to partition the data instances. The choice of the number of partitions was the result of performing a PCA on the data set and then visualising the first two principal components of the data set on a 2D surface to determine the points where an overlap between the output classes occurred. From this analysis it was decided that five intervals be used to accommodate the data instances and using these intervals, 17 rules were generated that are listed in Table 7.1.

<table>
<thead>
<tr>
<th>Table 7.1: The 17 rules generated from the CRE algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interval Int0 is from 0.10 to 1.60</td>
</tr>
<tr>
<td>Interval Int1 is from 1.69 to 3.26</td>
</tr>
<tr>
<td>Interval Int2 is from 3.27 to 4.84</td>
</tr>
<tr>
<td>Interval Int3 is from 4.85 to 6.42</td>
</tr>
<tr>
<td>Interval Int4 is from 6.43 to 8.00</td>
</tr>
<tr>
<td>IF Sepal-Length is Int3 AND Sepal-Width is Int2 AND Petal-Length is Int0 AND Petal-Width is Int0 THEN Iris-Setosa</td>
</tr>
<tr>
<td>IF Sepal-Length is Int3 AND Sepal-Width is Int1 AND Petal-Length is Int0 AND Petal-Width is Int0 THEN Iris-Setosa</td>
</tr>
<tr>
<td>IF Sepal-Length is Int2 AND Sepal-Width is Int1 AND Petal-Length is Int0 AND Petal-Width is Int0 THEN Iris-Setosa</td>
</tr>
<tr>
<td>IF Sepal-Length is Int3 AND Sepal-Width is Int2 AND Petal-Length is Int1 AND Petal-Width is Int1 AND Iris-Setosa</td>
</tr>
<tr>
<td>(continued on next page)</td>
</tr>
</tbody>
</table>
To validate the rules extracted using this algorithm the rules were translated into CLIPS (see Kasabov (1996a)) format and tested on ten random instances of the original Iris data set, the results of which are displayed in Table 7.2. A classification rate of 80% accuracy was obtained with the two mis-classifications in italics. This result is not unexpected as there is an overlap between Iris-Virginica and Iris-Virginica which this ruleset was unable to delineate. However we would expect this method to work well on data sets where the class boundaries are well defined.
Table 7.2: Table detailing output of classifier implemented in CLIPS based on rules derived using CRE

<table>
<thead>
<tr>
<th>Sepal Length</th>
<th>Sepal Width</th>
<th>Petal Length</th>
<th>Petal Length</th>
<th>Classifier Result</th>
<th>Desired Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>3.5</td>
<td>1.4</td>
<td>0.2</td>
<td>Setosa</td>
<td>Setosa</td>
</tr>
<tr>
<td>5.0</td>
<td>3.3</td>
<td>1.4</td>
<td>0.2</td>
<td>Setosa</td>
<td>Setosa</td>
</tr>
<tr>
<td>7.0</td>
<td>3.2</td>
<td>4.7</td>
<td>1.4</td>
<td>Versicolor</td>
<td>Versicolor</td>
</tr>
<tr>
<td>6.2</td>
<td>2.9</td>
<td>4.3</td>
<td>1.3</td>
<td>Virginica</td>
<td>Versicolor</td>
</tr>
<tr>
<td>4.9</td>
<td>2.5</td>
<td>4.5</td>
<td>1.7</td>
<td>Virginica</td>
<td>Virginica</td>
</tr>
<tr>
<td>6.0</td>
<td>2.2</td>
<td>5.0</td>
<td>1.5</td>
<td>Versicolor</td>
<td>Virginica</td>
</tr>
<tr>
<td>5.7</td>
<td>2.5</td>
<td>5.0</td>
<td>2.0</td>
<td>Virginica</td>
<td>Virginica</td>
</tr>
<tr>
<td>6.7</td>
<td>2.5</td>
<td>5.8</td>
<td>1.8</td>
<td>Virginica</td>
<td>Virginica</td>
</tr>
<tr>
<td>6.3</td>
<td>2.5</td>
<td>5.0</td>
<td>1.9</td>
<td>Virginica</td>
<td>Virginica</td>
</tr>
<tr>
<td>7.2</td>
<td>3.2</td>
<td>6.0</td>
<td>1.8</td>
<td>Virginica</td>
<td>Virginica</td>
</tr>
</tbody>
</table>

7.4.2 Fuzzy Rule Extraction

FRE, presented in Algorithm 7, is a modification of the algorithm described in Kasabov (1996a, pp. 218–221) and produces rules that are similar to Mamdani-type rules. The original FRE algorithm described in Kasabov (1996a) as initially created to generate rules suitable for a CLIPS environment and is can also be regarded as a set of four steps:

1. The data is discretised using the same method covered by Kasabov (1996a) for crisp rule extraction.

2. The discretised intervals are represented as fuzzy intervals and membership functions are attached to them. Three different types of fuzzy membership functions are used. Triangular functions are used for intermediate intervals and trapezoidal membership functions are used for the end intervals; the centre of the triangular membership function is placed at the centre of the interval and the other two vertexes at the middle points of the neighbouring intervals.

3. Fuzzy rules can then be expressed as representative templates if the assumption that the intervals are substituted by their fuzzy labels and their probability strength is used as a relative class rule strength.

4. Inference over the set of fuzzy rules with fuzzification and defuzzification is performed depending on how the fuzzy rules are applied.
Algorithm 7: Algorithm for fuzzy rule extraction from a data set

**Input:**
A set of data examples, \( D_1, D_2, \ldots, D_n \) where \( N \) is the number of data examples
numInputs is the the number of input attributes in the data set \( D \)
numClasses is the number of output classes in the data set \( D \)
umMFs is the number of membership functions to fuzzify the data set \( D \) into

**Output:**
A set of unique fuzzy rules \( R \)

1. for \( i = 1 \) to \( N \) do
2.  Fuzzify the data using a predefined number of triangular membership functions. \( R_{i,fuzzy} = \text{Fuzzify} (D_i,numMFs) \);
3. end
4. for \( i = 1 \) to \( N \) do
5.  Generate a degree of confidence for each rule which is produced from multiplying the degree of membership for each variable by 0.95; \( R_{i,confidence} = \text{GenerateDegreeConfidence} (R_{i,fuzzy},0.95) \);
6. end
7. for \( i = 1 \) to \( N \) do
8.  Select the unique fuzzy rules with the highest degree of confidence for the output variable; \( R_{i,unique} = \text{SelectUniqueRules} (R_{i,fuzzy}) \);
9. end
10. for \( i = 1 \) to \( \text{numClasses} \) do
11.  Count how many rules contributed to identifying each class in the data set; \( C_{i,frequency} = \text{GenerateClassFrequency} (R_{i}) \);
12. end

The modifications to the original algorithm were:

- The user could specify the number of membership functions and degree of confidence for each fuzzy rule to be retained.

- Only triangular membership functions were used to fuzzify the data

- Rules that had the same antecedents but differed in their consequent were selected based on the one with the higher degree of confidence.

- The general nature of the fuzzy rules created could be applied to many different fuzzy systems

As an example the FRE algorithm implemented in MATLAB was applied to the same benchmark data set for Iris classification. Three membership functions were used to model each input...
variable and 18 rules were derived from the data set. The results of applying FRE to this data set are presented in Table 7.3.

Table 7.3: The 18 rules generated from the FRE algorithm

<table>
<thead>
<tr>
<th>Rule Description</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF Sepal-Length is low(1.00) AND Sepal-Width is medium(0.83) AND Petal-Length is low(0.97) AND Petal-Width is low(1.00) THEN Iris-Setosa(0.76)</td>
<td></td>
</tr>
<tr>
<td>IF Sepal-Length is medium(0.78) AND Sepal-Width is high(1.00) AND Petal-Length is low(0.83) AND Petal-Width is low(0.75) THEN Iris-Setosa(0.46)</td>
<td></td>
</tr>
<tr>
<td>IF Sepal-Length is medium(0.67) AND Sepal-Width is medium(0.75) AND Petal-Length is low(0.90) AND Petal-Width is low(0.92) THEN Iris-Setosa(0.39)</td>
<td></td>
</tr>
<tr>
<td>IF Sepal-Length is low(0.89) AND Sepal-Width is low(0.75) AND Petal-Length is low(0.90) AND Petal-Width is low(0.83) THEN Iris-Setosa(0.47)</td>
<td></td>
</tr>
<tr>
<td>IF Sepal-Length is medium(1.00) AND Sepal-Width is medium(0.67) AND Petal-Length is medium(0.98) AND Petal-Width is medium(1.00) THEN Iris-Versicolour(0.62)</td>
<td></td>
</tr>
<tr>
<td>IF Sepal-Length is medium(0.94) AND Sepal-Width is low(0.83) AND Petal-Length is medium(0.98) AND Petal-Width is medium(0.75) THEN Iris-Versicolour(0.55)</td>
<td></td>
</tr>
<tr>
<td>IF Sepal-Length is low(0.61) AND Sepal-Width is low(1.00) AND Petal-Length is medium(0.85) AND Petal-Width is medium(0.75) THEN Iris-Versicolour(0.37)</td>
<td></td>
</tr>
<tr>
<td>IF Sepal-Length is medium(0.89) AND Sepal-Width is medium(0.92) AND Petal-Length is high(0.69) AND Petal-Width is high(1.00) THEN Iris-Virginica(0.54)</td>
<td></td>
</tr>
<tr>
<td>IF Sepal-Length is medium(1.00) AND Sepal-Width is medium(0.83) AND Petal-Length is medium(0.68) AND Petal-Width is medium(0.58) THEN Iris-Virginica(0.31)</td>
<td></td>
</tr>
<tr>
<td>IF Sepal-Length is high(0.89) AND Sepal-Width is high(0.83) AND Petal-Length is high(0.73) AND Petal-Width is high(0.83) THEN Iris-Virginica(0.43)</td>
<td></td>
</tr>
<tr>
<td>IF Sepal-Length is medium(0.83) AND Sepal-Width is medium(0.92) AND Petal-Length is high(0.53) AND Petal-Width is medium(0.58) THEN Iris-Virginica(0.22)</td>
<td></td>
</tr>
<tr>
<td>IF Sepal-Length is low(0.67) AND Sepal-Width is low(0.58) AND Petal-Length is medium(0.81) AND Petal-Width is medium(0.67) THEN Iris-Virginica(0.20)</td>
<td></td>
</tr>
<tr>
<td>IF Sepal-Length is high(0.61) AND Sepal-Width is medium(1.00) AND Petal-Length is high(0.69) AND Petal-Width is medium(0.58) THEN Iris-Virginica(0.24)</td>
<td></td>
</tr>
<tr>
<td>IF Sepal-Length is medium(1.00) AND Sepal-Width is low(0.50) AND Petal-Length is high(0.56) AND Petal-Width is medium(0.92) THEN Iris-Virginica(0.24)</td>
<td></td>
</tr>
<tr>
<td>IF Sepal-Length is medium(0.83) AND Sepal-Width is medium(1.00) AND Petal-Length is medium(0.54) AND Petal-Width is high(0.83) THEN Iris-Virginica(0.36)</td>
<td></td>
</tr>
<tr>
<td>IF Sepal-Length is medium(0.78) AND Sepal-Width is low(0.58) AND Petal-Length is medium(0.64) AND Petal-Width is high(0.58) THEN Iris-Virginica(0.16)</td>
<td></td>
</tr>
<tr>
<td>IF Sepal-Length is high(0.89) AND Sepal-Width is low(0.50) AND Petal-Length is high(1.00) AND Petal-Width is high(0.83) THEN Iris-Virginica(0.35)</td>
<td></td>
</tr>
<tr>
<td>IF Sepal-Length is medium(0.94) AND Sepal-Width is low(0.83) AND Petal-Length is medium(0.64) AND Petal-Width is medium(0.83) THEN Iris-Virginica(0.40)</td>
<td></td>
</tr>
</tbody>
</table>
To validate the rules extracted using this algorithm the rules were translated into the MATLAB Fuzzy Logic Toolbox (Jang; 2001) format and tested on the same ten random instances of the original data set, the results of which are displayed in Table 7.4. A classification rate of 100% was obtained.

Table 7.4: Table detailing output of classifier implemented in MATLAB based on rules derived using FRE

<table>
<thead>
<tr>
<th>Sepal Length</th>
<th>Sepal Width</th>
<th>Petal Length</th>
<th>Petal Length</th>
<th>Classifier Result</th>
<th>Desired Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>3.5</td>
<td>1.4</td>
<td>0.2</td>
<td>Setosa</td>
<td>Setosa</td>
</tr>
<tr>
<td>5.0</td>
<td>3.3</td>
<td>1.4</td>
<td>0.2</td>
<td>Setosa</td>
<td>Setosa</td>
</tr>
<tr>
<td>7.0</td>
<td>3.2</td>
<td>4.7</td>
<td>1.4</td>
<td>Versicolor</td>
<td>Versicolor</td>
</tr>
<tr>
<td>6.2</td>
<td>2.9</td>
<td>4.3</td>
<td>1.3</td>
<td>Versicolor</td>
<td>Versicolor</td>
</tr>
<tr>
<td>4.9</td>
<td>2.5</td>
<td>4.5</td>
<td>1.7</td>
<td>Virginica</td>
<td>Virginica</td>
</tr>
<tr>
<td>6.0</td>
<td>2.2</td>
<td>5.0</td>
<td>1.5</td>
<td>Virginica</td>
<td>Virginica</td>
</tr>
<tr>
<td>5.7</td>
<td>2.5</td>
<td>5.0</td>
<td>2.0</td>
<td>Virginica</td>
<td>Virginica</td>
</tr>
<tr>
<td>6.7</td>
<td>2.5</td>
<td>5.8</td>
<td>1.8</td>
<td>Virginica</td>
<td>Virginica</td>
</tr>
<tr>
<td>6.3</td>
<td>2.5</td>
<td>5.0</td>
<td>1.9</td>
<td>Virginica</td>
<td>Virginica</td>
</tr>
<tr>
<td>7.2</td>
<td>3.2</td>
<td>6.0</td>
<td>1.8</td>
<td>Virginica</td>
<td>Virginica</td>
</tr>
</tbody>
</table>

The mis-classification between the *Iris-Versicolor* and *Iris-Virgnica* classes have been resolved. Although this set of rules exhibited good performance on this data it was based on a static rule set. Therefore one would question the efficacy of this ruleset as more unseen examples of the Iris data set are presented to the classifier.

### 7.5 Rule Insertion into an EFuNN

As an alternative insertion of rules in AGES can be carried out via two different methods. The first allows the expert to directly code the rules using a small language that defines the rules to be inserted into a module in AGES, the other is based on the output of either the previously described CRE or FRE algorithms. Both methods are designed to generate fuzzy rules or crisp rules that can then be directly inserted into an EFuNN for training and rules extracted from them using a technique similar to that described in Section 4.6 of Chapter 4.
7.6 Rule aggregation in AGES

After the rules have been extracted from the EFuNN in AGES then rule aggregation may proceed. The algorithm has already been described in Chapter 4 and successfully applied in Kasabov and Woodford (1999) to reduce the number of rules extracted from a trained EFU NN using the benchmark Iris and gas Furnace data sets whilst maintaining a high classification for the Iris data set and low Root Mean Squared Error (RMSE) for the gas furnace data set.

7.7 Results from AGES

To illustrate the AGES architecture in action, an experiment was devised to combine all the features of AGES to solve one problem; that of Iris classification. The key steps in the experiment were:

1. Train an EFuNN on 75 instances of the iris classification data set.

2. Test the EFuNN using the remainder of the data set.

3. Insert two linguistic rules into a new untrained EFuNN and then train it on the same 75 instances.

4. Test this EFuNN using the remainder of the data set.

5. Compare the two EFuNNs in terms of their overall classification accuracy.

The experiment was designed to test the hypothesis that if the two rules were inserted into an untrained EFuNN, that they would act as a predefined partition to better separate the three output classes. Figure 7.3 contains the two rules inserted into the new untrained EFuNN that were an expanded form of the rules used in the experiments in Kasabov and Woodford (1999). The algorithm to translate this linguist representation of the rules into the EFuNN representation was also developed by the author and appears in Algorithm 8. After training an EFuNN without any inserted rules, 13 rule nodes were created and they are listed below in Figure 7.5.
Algorithm 8: Algorithm for linguistic rule insertion into an EFuNN

**Input:**
A set of rules, \( R_1, R_2, \ldots, R_n \) where \( n \) is the number of rules

**Output:**
The \( W_1 \) and \( W_2 \) matrices containing the connections weights representing the linguistic rules

```plaintext
for \( i = 1 \) to \( n \) do
    antecedentFlag = false;
    for \( j = 1 \) to Length \( (R_i) \) do
        Identify the current token, \( C_j \), in the rule \( R_i \);
        if \( C_j = \text{‘then’} \) then
            antecedentFlag = true;
        end
        if \( C_j = \text{‘If’} \) then
            \( P = C_j \);
        end
        if \( P = \text{‘If’} \) or \( P = \text{‘and’} \) then
            \( A_i = C \);
        end
        if \( P = \text{‘is’} \) and antecedentFlag = false then
            \( B_i = C \);
        end
        if \( P = \text{‘degree’} \) and antecedentFlag = false then
            \( C_i = C \);
        end
        if \( P = \text{‘class’} \) and antecedentFlag = true then
            \( E_i = C \);
        end
        if \( P = \text{‘is’} \) and antecedentFlag = true then
            \( F_i = C \);
        end
        if \( P = \text{‘degree’} \) and antecedentFlag = true then
            \( G_i = C \);
        end
    end
    Assign the input condition variable in the \( W_1 \) matrix with the input condition degree for the input condition MF:
    \( W_{1_{\text{InputValue}}} = \text{SetCorrespondingValue} (A_i) \);
    \( W_{1_{\text{MF}}} = \text{SetCorrespondingMF} (B_i) \);
    \( W_{1_{\text{Degree}}} = \text{SetCorrespondingDegree} (C_i) \);
    Assign the output class in the \( W_2 \) matrix with the output class degree for the output class MF:
    \( W_{2_{\text{OutputClass}}} = \text{SetCorrespondingValue} (E_i) \);
    \( W_{2_{\text{MF}}} = \text{SetCorrespondingMF} (F_i) \);
    \( W_{2_{\text{Degree}}} = \text{SetCorrespondingDegree} (G_i) \);
end
```
Table 7.5: The 13 rules extracted from the trained EFuNN with two rules initially inserted

<table>
<thead>
<tr>
<th>Rule Description</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF Sepal-Length is low(0.58) AND Sepal-Width is high(0.71) AND Petal-Length is low(0.70) AND Petal-Width is low(0.76) THEN Iris-Setosa(1.00)</td>
<td></td>
</tr>
<tr>
<td>IF Sepal-Length is high(0.57) AND Sepal-Width is medium(0.86) AND Petal-Length is medium(0.74) AND Petal-Width is medium(0.91) THEN Iris-Versicolour(1.00)</td>
<td></td>
</tr>
<tr>
<td>IF Sepal-Length is medium(0.67) AND Sepal-Width is low(0.62) AND Petal-Length is medium(0.95) AND Petal-Width is medium(0.83) THEN Iris-Versicolour(1.00)</td>
<td></td>
</tr>
<tr>
<td>IF Sepal-Length is medium(0.67) AND Sepal-Width is medium(0.57) AND Petal-Length is medium(0.85) AND Petal-Width is medium(0.92) THEN Iris-Versicolour(1.00)</td>
<td></td>
</tr>
<tr>
<td>IF Sepal-Length is low(0.58) AND Sepal-Width is low(0.52) AND Petal-Length is medium(0.68) AND Petal-Width is medium(0.83) THEN Iris-Versicolour(1.00)</td>
<td></td>
</tr>
<tr>
<td>IF Sepal-Length is medium(0.73) AND Sepal-Width is medium(0.95) AND Petal-Length is high(0.63) AND Petal-Width is high(0.75) THEN Iris-Virginica(1.00)</td>
<td></td>
</tr>
<tr>
<td>IF Sepal-Length is low(0.70) AND Sepal-Width is low(0.52) AND Petal-Length is medium(0.81) AND Petal-Width is medium(0.67) THEN Iris-Virginica(1.00)</td>
<td></td>
</tr>
<tr>
<td>IF Sepal-Length is medium(0.61) AND Sepal-Width is low(0.52) AND Petal-Length is high(0.63) AND Petal-Width is medium(0.58) THEN Iris-Virginica(1.00)</td>
<td></td>
</tr>
<tr>
<td>IF Sepal-Length is high(0.51) AND Sepal-Width is medium(0.86) AND Petal-Length is high(0.59) AND Petal-Width is high(0.83) THEN Iris-Virginica(1.00)</td>
<td></td>
</tr>
<tr>
<td>IF Sepal-Length is medium(0.85) AND Sepal-Width is medium(0.67) AND Petal-Length is medium(0.61) AND Petal-Width is medium(0.50) THEN Iris-Virginica(1.00)</td>
<td></td>
</tr>
<tr>
<td>IF Sepal-Length is high(0.82) AND Sepal-Width is medium(0.76) AND Petal-Length is high(0.73) AND Petal-Width is medium(0.50) THEN Iris-Virginica(1.00)</td>
<td></td>
</tr>
<tr>
<td>IF Sepal-Length is medium(0.97) AND Sepal-Width is medium(0.57) AND Petal-Length is high(0.56) AND Petal-Width is medium(0.92) THEN Iris-Virginica(1.00)</td>
<td></td>
</tr>
<tr>
<td>IF Sepal-Length is medium(0.85) AND Sepal-Width is low(0.52) AND Petal-Length is medium(0.64) AND Petal-Width is medium(0.50) THEN Iris-Virginica(1.00)</td>
<td></td>
</tr>
</tbody>
</table>

Testing this EFuNN resulted in the classification accuracy of 70/75 (93.33%) as shown in Table 7.6

Table 7.6: EFuNN when tested using the 13 extracted rules

<table>
<thead>
<tr>
<th>Classified as</th>
<th>Iris Setosa</th>
<th>Iris Versicolour</th>
<th>Iris Virginica</th>
<th>Total Correct</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iris Setosa</td>
<td>25</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Iris Versicolour</td>
<td>0</td>
<td>20</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Iris Virginica</td>
<td>0</td>
<td>0</td>
<td>25</td>
<td></td>
</tr>
</tbody>
</table>

EFuNN Parameters: numMF=3, sThr=0.95, Lr=0.02, Err=0.008, Old=60, Prune=60

70/75
Inserting the two rules presented in Figure 7.3 into an untrained EFuNN and then training it created 13 rule nodes as well. These rules were extracted and then compared to the rules generated by the EFuNN that had not been populated with any rules. In Table 7.7 two rules were different and indicated by being in italics.

If Sepal-Length is low to degree 0.8 and Sepal-Width is medium to degree 0.96 and Petal-Length is high to degree 0.5 and Petal-Width is high to degree 0.3 then class Iris-Virginica is high to degree 0.75

If Sepal-Length is high to degree 0.6 and Sepal-Width is low to degree 0.45 and Petal-Length is low to degree 0.1 and Petal-Width is low to degree 0.2 then class Iris-Setosa is high to degree 0.99

**Figure 7.3:** The two rules inserted into the untrained EFuNN

**Table 7.7:** The 13 rules extracted from the trained EFuNN with no initial rules

```plaintext
IF Sepal-Length is low(0.58) AND Sepal-Width is high(0.71) AND Petal-Length is low(0.70) AND Petal-Width is low(0.75) THEN Iris-Setosa(1.00)
IF Sepal-Length is high(0.57) AND Sepal-Width is medium(0.86) AND Petal-Length is medium(0.74) AND Petal-Width is medium(0.91) THEN Iris-Versicolour(1.00)
IF Sepal-Length is medium(0.67) AND Sepal-Width is low(0.62) AND Petal-Length is medium(0.95) AND Petal-Width is medium(0.83) THEN Iris-Versicolour(1.00)
IF Sepal-Length is medium(0.67) AND Sepal-Width is medium(0.57) AND Petal-Length is medium(0.85) AND Petal-Width is medium(0.92) THEN Iris-Versicolour(1.00)
IF Sepal-Length is low(0.58) AND Sepal-Width is low(0.52) AND Petal-Length is medium(0.68) AND Petal-Width is medium(0.83) THEN Iris-Versicolour(1.00)
IF Sepal-Length is medium(0.73) AND Sepal-Width is medium(0.95) AND Petal-Length is high(0.63) AND Petal-Width is high(0.75) THEN Iris-Virginica(1.00)
IF Sepal-Length is low(0.70) AND Sepal-Width is low(0.52) AND Petal-Length is medium(0.81) AND Petal-Width is medium(0.67) THEN Iris-Virginica(1.00)
IF Sepal-Length is medium(0.61) AND Sepal-Width is low(0.52) AND Petal-Length is high(0.63) AND Petal-Width is medium(0.58) THEN Iris-Virginica(1.00)
IF Sepal-Length is medium(0.85) AND Sepal-Width is medium(0.76) AND Petal-Length is medium(0.61) AND Petal-Width is high(0.92) THEN Iris-Virginica(1.00)
IF Sepal-Length is high(1.00) AND Sepal-Width is medium(0.76) AND Petal-Length is high(0.93) AND Petal-Width is high(0.58) THEN Iris-Virginica(1.00)
IF Sepal-Length is high(0.82) AND Sepal-Width is medium(0.76) AND Petal-Length is high(0.73) AND Petal-Width is medium(0.50) THEN Iris-Virginica(1.00)
IF Sepal-Length is medium(0.85) AND Sepal-Width is medium(0.67) AND Petal-Length is medium(0.61) AND Petal-Width is medium(0.50) THEN Iris-Virginica(1.00)
```

(continued on next page)
Testing this EFuNN resulted in the classification accuracy of 73/75 (97.33%) as shown in Table 7.8.

Table 7.8: EFuNN when tested using the 13 extracted rules

<table>
<thead>
<tr>
<th>Classified as</th>
<th>Iris Setosa</th>
<th>Iris Versicolour</th>
<th>Iris Virginica</th>
<th>Total Correct</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iris Setosa</td>
<td>25</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Iris Versicolour</td>
<td>0</td>
<td>23</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Iris Virginica</td>
<td>0</td>
<td>0</td>
<td>25</td>
<td></td>
</tr>
</tbody>
</table>

73/75

7.7.1 Discussion

The results from this experiment suggested that the rule insertion into an EFuNN increases its accuracy and also alters the parameters of the rules. A combination of the rule insertion and rule extraction algorithms could therefore form the core of an adaptive inference engine that could then be used as part of a larger decision support system.

7.8 Implementation Issues

AGES is an adaptive expert system architecture and has been designed to operate in an on-line, real-time environment where the modules may either reside on the same computer or be spatially located throughout a distributed network on other computers. Choosing an appropriate computer language to implement AGES needs to satisfy these requirements:

- Be a compiled language for speed and memory efficiency. Some of the operations performed by AGES may be quite complex thus the need for a compiled language to maintain its real-time processing nature.
• The ability to be run on multiple architectures. If AGES is to be a fully distributed system then this may also mean that parts of AGES could run on multiple computer architectures running different operating systems.

• Generate modules that are small in size that can be quickly sent around the distributed network that forms AGES.

• Able to be integrated into many different modes of interaction. For example, the user interface to AGES may be a plugin for an Internet browser or its own stand alone program.

• Source different repositories of data either locally on a computers hard disk or alternatively using a remote repository of data.

The programming language Java (http://www.sun.com/software/java) satisfied all these requirements and was chosen as the preferred language to implement AGES. Java has now matured into a stable and versatile programming language for deploying on-line applications and many existing tools and techniques written in other programming languages have been ported into this powerful language. AGES employs many of these new tools such as the Voyager distributed architecture (http://www.recursionsw.com/) to facilitate the distributed nature of AGES to communicate with modules external to the computer it is running on. EFuNN (Kasabov; 1999b) has also been ported to Java, used in the Repository of Intelligent Connectionist-Based Information Systems (RICBIS) (Deng et al.; 1999) and also factored into AGES. The data repository used in AGES was chosen to be a Java based relational database engine called InstantDB from http://instantdb.enhydra.org. Finally the Java Expert System Shell (JESS) (Friedman-Hill; 1997) was the heart of the forward chaining inference engine for AGES that is based on the OPS5 production system (Forgy; 1981).

7.9 Summary and concluding remarks

Several new types of problems have created problems for conventional expert systems design and implementation especially when they are working in an on-line and real-time environment. The
AGES architecture has been proposed as one solution to the task of developing more advanced expert systems whose rules can be adapted over time as changes occur in the environment in which it works.
Chapter 8

An adaptive expert systems for horticulture: The IPMES

8.1 Introduction

The potential benefits of applying expert systems to integrated pest management has been identified for several years (e.g. (van den Ende et al.; 1996; Gerevini et al.; 1992, 1991)). With the introduction of Integrated Fruit Production (IFP) to the pipfruit industry by ENZA and HortResearch New Zealand (Walker et al.; 1997), the need for expert system technology to support its implementation and use has been highlighted. IFP emphasises human and environmental safety in a continually improving fruit production system. It aims to minimise the negative impacts of agrichemical use and therefore requires an integrated system that manages information for the refinement of IFP pest management recommendations.

The complex of pests (Table 8.1) which attack pipfruit orchards in New Zealand have been managed historically by calendar schedules of broad-spectrum insecticide applications but with the introduction of IFP, a new level of complexity has been added to orchard operations.

Under IFP, growers are required to justify all spray applications based on either pest phenology or the levels of insect pests in their orchards and, whenever possible use selective products which maximise survival of beneficial species, especially natural enemies. To achieve these objectives, they must be able to recognise pest and beneficial species, monitor the pest levels, and make applications of the most suitable chemicals when these levels exceed predetermined thresh-
Table 8.1: The major insect and mite pest complex attacking New Zealand apples

<table>
<thead>
<tr>
<th>Pest Group</th>
<th>Pest Species</th>
<th>Common name</th>
<th>Latin name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Leafrollers</td>
<td>Leafrollers</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Lightbrown apple moth</td>
<td>Epiphyas postvittana</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Greenheaded leafroller</td>
<td>Planotortrix excessana</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Brownheaded leafroller</td>
<td>Ctenopseustis herana</td>
<td></td>
</tr>
<tr>
<td>Fruit borers</td>
<td>Fruit borers</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Codling moth</td>
<td>Cydia pomonella</td>
<td></td>
</tr>
<tr>
<td>Cutworms</td>
<td>Cutworms</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Noctuid</td>
<td>Graphania mutans</td>
<td></td>
</tr>
<tr>
<td>Scale insects</td>
<td>Scale insects</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>San Jose scale</td>
<td>Quadraspidiotus perniciosus</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Oystershell scale</td>
<td>Quadraspidiotus ostreiformis</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Greedy scale</td>
<td>Hemiberlesia rapax</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Latania scale</td>
<td>Hemiberlesia lataniae</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Oleander scale</td>
<td>Aspidiotus nerii</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mussel scale</td>
<td>Lepidosaphes ulmi</td>
<td></td>
</tr>
<tr>
<td>Mealy bugs</td>
<td>Obsure mealybug</td>
<td>Pseudococcus viburni</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Citrophilus mealybug</td>
<td>Pseudococcus calceolariae</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Longtailed mealybug</td>
<td>Pseudococcus longispinus</td>
<td></td>
</tr>
<tr>
<td>Aphids</td>
<td>Woolly apple aphid</td>
<td>Eriosoma lanigerum</td>
<td></td>
</tr>
<tr>
<td>Mites</td>
<td>Mites</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>European red mite</td>
<td>Panonychus ulmi</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Twospotted mite</td>
<td>Tetranychus urticae</td>
<td></td>
</tr>
</tbody>
</table>

olds. Full record keeping is essential for the success of the programme. The orchardists manage this processes by dividing their orchards into management blocks, often of a single cultivar, and these blocks provide the basic sampling units to which all data relates.

The development of expert systems is timely to address data analysis, interpretation, integration, and management of the large number of cultivar blocks that constitute the IFP programme within New Zealand. This has risen from about 500 blocks on 88 orchards in 1996/97 (the first year of implementation) to about 5300 blocks over 700 orchards in the 1998-99 season. With increasing fruit industry commitment to IFP, there was the potential for ongoing data management and this resulted in over 10,000 blocks per annum when the IFP programme was fully implemented in 2001. HortResearch New Zealand plays a key role in the management and analysis of the IFP data which benefits both the growers and ENZA itself.

Although there are many IFP databases that exist, they are a combination of hardcopy and electronic repositories with few linkages between them. If some integration of this information was possible, it then it could provide decision support to all levels of orchard management from
the orchardist to IFP programme management.

The Integrated Pest Management Expert System (IPMES) project is being developed by the University of Otago and HortResearch New Zealand. It applies the AGES architecture and applies new learning techniques to building intelligent systems. The overall intention of this project was to develop an adaptive decision support system based on information integration. It was designed for use by ENZAFruit New Zealand (International) (ENZA), HortResearch New Zealand, packhouses, and the growers/orchardists themselves.

The overall architecture of the IPMES was designed to be flexible, adapt to the users’ requirements, able to be extended from its current functionality, and incorporate the wishes of the people who use the system. In the following sections, each component of the IPMES will be described in detail, how it was implemented, and also some examples will be presented of how the system functions. The IPMES furthers the contribution of this thesis in the form of a direct application of the AGES architecture described in Chapter 7.

8.2 Sources of Information

To understand fully how the IPMES functions, one must become aware of a number of important sources of data that are required to implement a system of this type. Figure 8.1 contains the information flow of the pest management data about the fruit from orchard to packhouse to ENZA that is used in the decision-making process.

Starting at the Grower (G) information at either the orchard or cultivar (cv) block level is recorded on spray decisions made throughout the season i.e. the pest monitored, the justifications for spraying, the choice of product used, and the date of its application. This information is supplied to ENZA as the grower’s Pest Control Records (PCR) which may affect decisions about the apples that are produced by the orchard, such as the destination market overseas. In addition to the ENZA processes, the packhouse where the fruit was sent also makes a Harvest Assessment of the fruit quality by recording defects encountered during packing. This information is then passed back to the Grower and that in turn will influence the decisions made on spraying for the next season.
These Harvest Assessments are part of the Quality Assurance (QA) and Quality Control (QC) of each Cultivar Block. Pest monitoring and PCR information collected by ENZA, and fruit defect data from the packhouses, can then be by ENZA and HortResearch New Zealand to refine the thresholds for the IFP thresholds that the orchardists will use for the next season.

Of interest to the IPMES team are four specific sources of data that are (1) primary determinants of when to spray the fruit and/or (2) records of what was applied and its effectiveness. These are the records from Weather Station, PCR’s, Pest Monitoring, and Pest Status at harvest. These sources of data were then aggregated by the author into one data base to be used by the IPMES system.
8.3 The Knowledge Base for the IPMES

Making decisions on when to spray and with what pesticide are defined by the ENZA-IFP manual developed in collaboration with HortResearch New Zealand. Contained in this document is detailed information on what pests the growers should be aware of, their description, and what pesticide to spray based on the time of the season and the numbers of a given pest species have been found in traps or on either the fruit or the trees. Two important issues affecting the IPMES are contained in this manual:

1. The decisions that are made are not just based on information recorded about the current season, but influenced by certain parameters obtained from the previous season, most importantly, the harvest assessment.

2. The thresholds that are in the IFP manual are not necessarily grounded on the analysis of the information acquired over the past two years but based on heuristics which translate to the values contained on the document. The IPMES could provide valuable information to ENZA about how to alter the values based on the processing of the expert system.

To model these rules in an expert system requires that the thresholds themselves in each rule can be modified to adapt to the changes in the national thresholds or regional thresholds. The use of a fuzzy rule base with adaptive membership functions that are altered by evolving neural networks would provide one solution which have already been used in Kasabov (1996c). In this way the thresholds would be tailored to New Zealand orchards or a particular region by automatically modifying the thresholds as new information about the pest levels are known nationally or regionally. Techniques similar to this have already been applied to create adaptive intelligent information systems by Kasabov (1996a); Kasabov, Kim, Watts and Gray (1996); Kasabov and Woodford (1999). By integrating these methods into the IPMES through the AGES architecture the adaptive nature of this on-line expert system can be realised.
8.4 The Data Base for the IPMES

The data base for the IPMES was compiled from a set of 36 orchards that were maintained by AgResearch, a Crown Research Institute (CRI) that conducted research into increasing the quality of agricultural produce within New Zealand. Apples in these orchards were grown using the rules set down in the ENZA-IFP manual for the 1998-1999 growing season. Over the period of the growing season, trap catch numbers, insecticide applications, and results of disease monitoring were recorded and then entered into an electronic data base. For the purpose of confidentiality, the actual names of the orchards were not provided. Instead a unique identifier was used. For the purpose of the examples described later on in this chapter, the orchard D1927 was used. There were 165 trap catch entries, 66 insecticide applications, and 50 disease monitoring entries for this orchard in the data base.

8.5 The Functionality of the IPMES

One of the prime requisites of the IPMES is the ability to be flexible in its decision-making capabilities to order to adapt to the various types of people that will be using the IPMES. This formed the basis of the three main stages of its functionality:

1. The ability to visualise the data about the orchard over time based on information contained in the database or to access past records of the orchard for a “What If” scenario based analysis. This would demonstrate how altering the thresholds would affect the spraying of the pesticide and in turn indicate projected pest damage in the orchard. This stage is based purely on decisions inferred from the facts contained in the database.

2. The next level is decision-based support for the risk assessment of the presence of the insects. This stage was based on both the historical data and current data stored in the database about the orchard.

3. The final level recommends applicable treatment of pesticide and use the rules and thresholds store in the rule base of the IPMES that are initially based on the ENZA-IFP manual.
8.6 Architecture of the IPMES

Figure 8.2 contains the logical architecture of the IPMES. The growers, HortResearch New Zealand, and ENZA collect the data. How it is collected depends on the type of data in question. The data is then stored in the database ready for retrieval. Users of the IPMES access the database through the Internet to update the model of their own orchard which will then be used for the purpose of data visualisation and the assessment of the risk of pest presence. The “intelligence” of the IPMES is embodied in the ENZA-IFP knowledge base and the decision support module. These two modules utilise components that alter the thresholds for the rules to accommodate the type and presence of insects within a particular orchard so that the appropriate spray decisions can be made.

![Logical architecture of the IPMES](image)

Figure 8.2: Logical architecture of the IPMES

Figure 8.3 depicts the physical architecture of the IPMES. The user specifies what task to perform through an User Interface running as a Java applet running on a web browser. This task is then interpreted by the Module Creator that decides what module from the Module Repository should be used to satisfy the query. Once the appropriate module has been selected, the selected module then performs a set of SQL queries on the IDB Database to access the appropriate data to satisfy the query and then combines this data with either a visualisation module to create a specific visualisation or the FuzzyCLIPS Rulebase to infer a decision. The result of the process-
ing is then returned to the User Interface as either a visualisation or a set of recommendations. The implementation written in Java for the IPMES and modified version of the Java EFuNN code based on Deng et al. (1999) was solely developed by the author of the thesis.

At the end of every growing season, the EFuNN Module can be invoked to access the data collected in the IDB Database to make adjustments to the FuzzyCLIPS Rulebase.

8.7 Description and Functionality of the Modules for the IPMES

The IPMES contains three different types of modules:

- Data Visualisation Modules
- Grower Advice Modules
- Threshold Improvement Modules
The orchardists use only the Data Visualisation and Grower Advice modules with ENZA and HortResearch New Zealand utilising all the Modules within the IPMES. Interaction between the Grower Advice Modules and Threshold Improvement Modules do occur and is explained later on in this chapter.

8.7.1 Data Visualisation Modules

These modules summarise the trap catches, other pest monitoring, harvest results, and pest control record information for a particular orchard block. Each sub-module for the Data Visualisation Module is described below.

Trap catch module

The functionality of this module visualises the Codling Moth (CM) and LeafRoller (LR) weekly trap catches and associated spray applications. It communicates with the Pest Control Record and trap catch databases, and the IFP Manual ruleset: Users can specify catch information and spray applications for:

1. Block/orchard versus current threshold in IFP manual.
2. Compared to previous year on the same block/orchard.
3. Compared to the district average for the same year.

As an example Figure 8.4 depicts the user interface for specifying trap catch visualisations in IPMES. In this example, the user has specified a query for visualising Codling Moth trap catch numbers over the 1998-1999 season for orchard D1927. Figure 8.5 shows the result of this query.
Figure 8.4: Making a query on trap data contained in the database

The output of this query contains two graphs. The upper graph shows the number of pheromone traps in the orchard and accumulation of codling moth in those traps over the season. The top right-hand box indicates on what date, insecticide was applied. Note that after each application, the trap values are reset to zero.

The bottom graph depicts how trap numbers of the orchard (in red) compare to the average trap number catch for the growing district (in yellow) over the 1998–1999 season. This provides the orchardist with some indication of the state of the other orchards within their district without identifying what orchards these values relate to.

Figure 8.5: The resulting output of the query specified in Figure 8.4
**Pest monitoring module**

The functionality of this module visualises the scale, woolly apple aphid, mealybug, appleleaf curling midge, European red mite, and twospotted mite pest monitoring data and associated spray application. It communicates with the Pest Control Record and pest monitoring databases and the IFP Manual ruleset. Users can specify pest monitoring and spray applications for:

1. Block/orchard versus current threshold in IFP manual.
2. Compared to previous year on the same block/orchard.
3. Compared to the district average for the same year.

**Harvest assessment data visualisation**

The functionality of this module visualises the harvest assessment data from a particular cultivar block or orchard/block. It communicates with the harvest database. Users can specify:

1. Harvest data for an individual pest insect compared to yearly threshold - for those pests where harvest data is used for this purpose e.g. mealybug, scale, and woolly apple aphid.
2. Compared to previous year on the same block/orchard.
3. Compared to the district average for the same year.

**Pest Control Record visualisation over the season without monitoring or trap data**

The functionality of this module visualises pest control entries over the season without the integration of monitoring or trap data. It alone communicates with the pest control diary database. Users can specify:

1. Spray diary information for each orchard/block by variety.
2. Compared to previous year on the same block/orchard by variety.
8.7.2 Grower Advice Modules

These modules provide advice to the grower with a range of options in maintaining the orchard in terms of spraying, pest monitoring, recent spray applications, update of database, and harvest date reminders.

The user interface for this module is more complex than the one developed for the data visualisation modules. Figure 8.6 depicts this user interface. It is partitioned into two distinct sections:

- **Trap Data Entry:** This part allows the user to enter in the date of entry, the orchard number, the, the trap identifier and the number of pests found on the trap.

- **Monitoring Data Entry:** This part allows the user to enter in pest monitoring data at the beginning of the season. Included is a check box for orchardists who are beginning to grow apples using IFP. By checking this box, advice that is given will be supplemented with other recommendations for the new grower that need to be implemented.

![Figure 8.6: The data input screen for the IPMES](image)

**Immediate spray recommendation module**

The functionality of this module provides an immediate spray recommendation to the grower (with options). The module is aware of current needs/recommendations for all pests so that options would be presented with a rationale for choice. It communicates with the pest control, trap
catch, pest monitoring, withholding period database, pests-pesticides database, harvest database from previous season, and IFP Manual ruleset. Four conditions can trigger this module:

1. After the user enters pest monitoring or trap catch data. Depending on the type of data entered either the pest monitoring or trap catch databases are updated and the FuzzyCLIPS Rulebase then invoked to suggest a possible spray application. Advice is only generated if the withholding period from a previous spray application has been completed.

2. Whenever the system infers that the data indicates spray need for a particular block. This would be without entry of current data but due to past data entry(ies). It happens when the withholding period for a particular spray has been exceeded and advice can be recommended based on current pest monitoring or trap data values.

3. Whenever the system infers that spraying is not required on a block and the reason why. This would be without entry of current data but due to past data entry(ies). This would normally happen when the withholding period for a particular spray is still in its efficacy period or the values either for pest monitoring or trap catch data are below their respective thresholds.

4. A specific time time of year. For example a highly recommended first spray for Codling Moth (CM) and LeafRoller (LR), and green tip oil for scale insects. At the beginning of the new growing season this advice is strongly recommended as a first spray application. To be compliant with IFP process it forms part of the overall advice recommended to the orchardist until records of these spray applications appear in the pest control database.

**Monitoring recommendation module**

The functionality of this module provides monitoring recommendations to the grower. The module is aware of current needs/recommendations for all pests so that options would be presented with a rationale for choice. It communicates with the harvest database from previous season, pests-pesticides database, pest control record, trap catch, pest monitoring, withholding period
database, and IFP Manual ruleset. Two situations can occur to invoke trapping and pest monitoring recommendations:

1. Whenever entering the system depending on time of year and region. The IPMES invokes the *FuzzyCLIPS Rulebase* to determine if the current date falls within the period where the monitoring of disease can occur. If this is the case the monitoring of disease is advised based on the region where the orchard is located.

2. After a specific action has been taken which requires follow up monitoring. Some spray applications require that a specific number of fruit be sampled after treatment to determine the spray’s effectiveness. If this is the case, the pest monitoring database is checked to see if the sampling has occurred after the specific period of time. This advice is presented until a record appears in the pest monitoring database that contains the details of the sampling of the fruit.

**Recent spray applications module**

The functionality of this module provides detailed information of the previous spray applications to a particular orchard. The module would be aware of current needs recommendations for all pests so that options would be presented with a rationale for choice. It communicates with the pests-pesticides, metview database and pest control record databases. Two situations can occur to invoke the Recent spray applications module:

1. Whenever entering the system. An option will be available to check the date and product of last spray applied for any pest on any specified block.

2. Estimate of residue effectiveness of the spray depending on realtime rainfall, UV levels, and time interval.

**Reminder module**

The functionality of these modules is to remind the user that certain data must be entered in order for some of the other modules to function, especially those relating to the Grower Advice pack-
It communicates with the pest control record database, trap catch database, pest monitoring database, and withholding period database which in turn influences the decisions made by the spray application modules. Two situations can occur that invoke the Reminder module:

1. When the harvest dates for each variety are missing from the withholding period database.

2. A general reminder to enter data or a message advising when data was last entered.

### 8.7.3 An example of the IPMES in action

Consider this scenario: It is fast approaching the end of the month and the orchardist enters in the number of codling moth found at the end of the week. Figure 8.7 contains a screenshot of the data entered. It is important to note the date this data was entered which was 25/12/1998. Upon submitting the data to the IPMES several modules are then invoked:

- The immediate spray recommendation module to determine if a spray application can be applied. This uses information gathered from the Recent spray applications module to determine if a spray application can be applied, and if so, which one.

- The reminder module to check if the data entered was timely enough to make an appropriate decision.

![Figure 8.7: Data supplied to the IPMES resulting in no spray recommendation](image)

The result of invoking these modules is presented in Figure 8.8. For information purposes, the last recorded spray application is displayed along with its residual efficacy period. As another
spray cannot be applied for another 17 days, no advice can be recommended. Output from the reminder module also occurs as the data collected does not fall within a time period between today’s date and a week ago.

![Output from the Decision Support Engine](image)

**Figure 8.8:** Output of IPMES with no spray recommendation

For the remainder of the current month and into the next month the orchardist enters in trap catch data. Each time new data is entered, the same recent spray, immediate spray, and reminder modules are then invoked and interrogate the updated database. This may result in a rule firing in the *FuzzyCLIPS Rulebase* if the pest numbers equal or exceed a threshold as defined in one or more of the rules. The firing of the rule then provides the basis of the advice presented to the orchardist.

In this example some spray advice can be given as the efficacy period for the previous spray application has ended and the number of pests trapped has exceeded a predefined threshold in one of the rules. Figure 8.9 displays the advice and informs the orchardist that it has been a period of time since the last spray and recommends an application of an insecticide.

By taking this advice, the orchardist will then make an entry in *IDB Database* of the details of insecticide applied. This will then have the effect of resetting all the counters for the trap catches and the process starts all over again.
8.7.4 Validation of the advice provided by the IPMES

To validate the advice provided by the IPMES, a set of test cases were used to validate the recommendations by the IPMES and can be found in Appendix A. Because the database used was primarily based on the 1998–1999 season it was both possible to interrogate the database to see if the recommendation had been actioned by the grower. In conjunction with C. Howard Wearing of HortResearch Clyde, the system was validated and then refined over a period of a year between 2000 and 2001.

Validation occurred by applying the test cases to the IPMES and recording the advice obtained by the IPMES. The test cases and resulting advice was then sent to C. Howard Wearing where he compared the advice given against what the IFP system would have recommended. Any incorrect advice were identified and reported back so that adjustments to the rule base or the inference engine could be made. The cycle continued until all the test cases produced the correct advice. The IPMES was then demonstrated to a group at HortResearch Havelock North and they agreed that it was a prototype worth investigating and developing further.
8.7.5 Threshold Improvement Module

This module provides detailed information about what thresholds to change in the IFP manual. It is initially based on at least the last three year’s data sets on a particular orchard as the improvement of the thresholds requires at least that much data before a confident recommendation could be made. The module identifies trends in the operation of an orchard, and of orchards in a region, that differ from that recommended by the current IFP ruleset. It communicates with the IFP ruleset, trap catch, pest monitoring, pest control record, harvest assessment, and packhouses databases in order to make the modifications. These modifications would normally be restricted to a particular growing district (this is the primary aim) or specific orchard (this is the secondary aim). Threshold modification would normally happen at the end of the season when the harvest assessment is completed and can be selected for a particular district, orchard, cultivar block, or apple variety depending on the criteria for analysis.

This process is made possible using a combination of the EFuNN and its associated rule extraction mechanism. As mentioned before the IPMES was designed to be used at the various levels of management in the horticultural sector. This particular function would necessarily be used by HortResearch New Zealand staff to mine for, validate, and refine the existing rules in the IFP manual. Figure 8.10 is the user interface for this module. There are functions available to insert a rule-set, extract a rule set, and adjust the thresholds. Currently the resolution of rule adjustment is at the growing district level although it was envisaged that it could be applied to a
finer resolution such as an individual orchard. The main benefit of this rule adjustment procedure is that it can occur in on-line and in real-time without having to stop the IPMES.

![Screenshot for the insertion of rules](image)

Figure 8.11: Screenshot for the insertion of rules

When rule insertion is invoked, a new window is created that allows the user to set the parameters of the data set to be loaded and inserted into an EFuNN. For example, trap catch data taken from the IDB database has been used to adjust the thresholds of the IFP manual that deal with codling moth and leafroller damage. In Figure 8.11 the user has specified that the data set has four inputs and three outputs along with three membership functions per input. Each attribute in the input vector relates to:

1. Number of days after the beginning of the growing season
2. Pest ID (0 = codling moth, 1 = leafroller)
3. Cumulative value of pests caught in traps
4. Insecticide applied (0 = no, 1 = yes)

The values for the output vector were:

1. \[ 1 \ 0 \ 0 \] - Application of Mimic insecticide
2. \[ 0 \ 1 \ 0 \] - Application of Match insecticide
3. \[ 0 \ 0 \ 1 \] - Application of Oil

The type of rules that can be inserted have the same format that has been described in Section 7.7 of Chapter 7. The rules in the IFP manual that govern insecticide application based on trap catch data were then translated into this format.
Once the rules have been inserted, threshold adjustment can begin. To adjust the thresholds of the rules, the data set from the previous season for orchard D1927 is read in and by using an EFuNN the rules can then be adjusted based on the characteristics of the data set. Figure 8.12 is a screen shot of the user interface used to modify the rules.

Finally rule extraction can begin. This procedure uses the same algorithm we developed and used in Kasabov and Woodford (1999). Figure 8.13 allows the user to set the threshold for the extracted rules.

Figure 8.14 displays a screenshot of the type of rules extracted from the process. Once the
rules have been extracted they can then be translated to the format used in the *JESS FuzzyCLIPS* inference engine that represents the IFP rules for the current season.

### 8.8 Summary and Conclusion

The marriage of the AGES architecture and its application to support the orchard management process in New Zealand has been the focus of this chapter. The IPMES has been implemented to support the decision-making process of those people directly involved in the production of quality pipfruit. Using a modular architecture and a combination of different data sets, it seeks not only to create an integrated decision support system but also possesses the functionality to extract knowledge with from which the rule base of the system can be modified.
Chapter 9

Conclusion and Future Directions

9.1 Concluding Remarks

This thesis has described a set of CBIIS method for image analysis and knowledge engineering in horticulture. Throughout the main body of this work, these methods have been applied to a set of problems that are either part of or directly related to New Zealand’s horticulture industry. This work is significant and original for the following reasons:

- **The identification, acquisition, and representation of information required for applying CBIIS to decision-making in horticulture.** Wavelets were successfully used as the feature representation of both image and NIR signals that were then used as input to different models including the MLP, SVM, FuNN, EFuNN, and $k$-means classifier.

- **The development of an incremental, on-line, adaptive, real-time learning CBIIS model to aid in the classification of horticultural data.** The EFuNN model has been applied to the problem of pest identification of damaged apples, MRI image analysis, and the classification of phenotypes of both persimmon fruit and *pinus radiata*.

- **Generic knowledge management and acquisition methods within this CBIIS model using a rule extraction and a rule insertion algorithm.** We have proposed a method for rule insertion and rule extraction, applied these methods to two traditional problems in classification and prediction, and then used these algorithms to extract knowledge from data sets derived from the horticultural domain in order to provide some insight into the processes that govern the development of the fruit or the tree.
• **An architecture for an on-line, adaptive, agent-based expert system.** AGES is a variation on the traditional expert system architecture that employs the CBIIS model as its core and proposed as a solution to the main problems identified with traditional expert systems design and implementation. The advantages of the AGES architecture are incremental learning, rule insertion, and rule extraction.

• **The application of this new expert-system architecture for decision-support in apple orchard management.** With the IPMES, the orchardist can use this system to aid in the decision-making process when managing their orchard and also to adapt the existing rules contained in the rule-base using the EFuNN model.

### 9.2 Future Directions

This work provides a foundation for future research in CBIIS and ECOS based techniques and their applications. We conclude this thesis with several important and promising research directions along the follows:

• **Identification of more appropriate features for use in ECOS models:** In Chapter 5 and Chapter 6 the use of wavelets were successfully used as salient features that could be used in a classification model. The choice of wavelet was a combination of previous studies on the use of wavelets for image analysis and the work that we described in this thesis. It would therefore be logical to further this work by examining other modern types of wavelets for analysis of the type of images and signals used in the studies described in this thesis.

• **Increasing the speed of training the EFuNN:** When an input data vector is presented to an EFuNN, we need to calculate the all distances between this input vector and each existing rule node. As the number of rule nodes increases, this may take an inordinate amount of time and reduce the on-line and real time learning nature of this novel connectionist architecture. This problem is compounded by the fact that the input vectors described
in the studies presented in this thesis are very large. A different method of calculating these distances needs to be created or else the on-line learning performance of these CBIIS models will degrade to off-line traditional models. One solution is the Zero Instruction Set Computer (ZISC) (Lindsey et al.; 1995), a hardware implementation of a fast learning architecture that mimics the learning algorithm of the EFuNN.

- **Rule Aggregation:** The number of rules or rule nodes created in EFuNN depends on the parameters to a certain extent. In some cases, however, we found that too many rules or rule nodes are created as the input data appear, and it is necessary to reduce them. However, rule reduction can only be facilitated when the EFuNN is not training. A suitable method for on-line rule aggregation needs to be developed.

- **On-line Feature Selection:** The size of the input attributes for a majority of the data sets described in this thesis are very large ranging from a 20 input vector to a 768 input vector. One issue not addressed in this thesis has been applying a relevant attribute selection technique to reduce the number of input attributes. Unlike the standard attribute selection techniques already presented, this method must also operate on-line and in real-time as the importance of the input attributes may change over time.

- **Implementing more of the AGES architecture:** The AGES architecture described in Chapter 7 was implemented in Java to the point where it could satisfy the functionality required for the IPMES. Building the entire system in Java is desirable so that other expert systems can then be designed and developed using this architecture thus validating this novel approach.

- **Further developing the IPMES:** The IPMES described in this thesis is at the prototype stage. It would be desirable to develop the system further to integrate all the other processes that entail the complex process of orchard management.

- **Finding other application areas where these techniques can be applied:** Although this work has been domain specific, there is scope for this work to be extended beyond that of
horticulture. There are other application areas where decision-making must be made based on complex images. The medical field, for example, is one area.
References


**URL:** [http://www.it.bond.edu.au/inft140/Lectures061/Lecture5/Lecture%205.htm](http://www.it.bond.edu.au/inft140/Lectures061/Lecture5/Lecture%205.htm)


**URL:** [ftp.cs.cmu.edu:/user/ai/pubs/faqs/fuzzy/fuzzy.faq](ftp.cs.cmu.edu:/user/ai/pubs/faqs/fuzzy/fuzzy.faq)


URL: http://www.iiit.net/˜vikram


221


**URL:** [http://www.hortnet.co.nz/key/](http://www.hortnet.co.nz/key/)


227
Appendix A

Test Case Scenarios for the IPMES Inference Engine

A.1 Introduction

Contained in this document are test case scenarios that have been extracted from the Agnz99 set of databases. The trap catch information, and pest disease entries, we assume, have conformed to the IPF98-99 manual guidelines. Using this data we can then submit test case scenarios to the expert system, IPMES-Engine9899, that then will report on what action the orchardist should take. The aim of this task is to validate the expert system and also make adjustments to it so that the inference reported by the IPMES is as close to what an expert working for HortResearch New Zealand would recommend.

A.2 Description of the data set

Six orchards have been used to test the inference engine and have been selected from the Agnz99 databases. They are D1164, D1784, D1891, D1927, D2017, and D2080. All the entries connected to these orchards have been taken from the 1998-1999 seasons. The number of entries per orchard is summarised using A.1

Some assumptions of have been made made regarding this data set:

- Trap catch numbers have been reset to 0 before recording of the first entry in the database
Table A.1: Data used for the test case scenarios

<table>
<thead>
<tr>
<th>Orchard</th>
<th># of trap catch entries</th>
<th># of pest disease entries</th>
<th># PCR entries</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1164</td>
<td>74</td>
<td>32</td>
<td>121</td>
</tr>
<tr>
<td>D1784</td>
<td>132</td>
<td>26</td>
<td>300</td>
</tr>
<tr>
<td>D1891</td>
<td>102</td>
<td>35</td>
<td>216</td>
</tr>
<tr>
<td>D1927</td>
<td>165</td>
<td>50</td>
<td>66</td>
</tr>
<tr>
<td>D2017</td>
<td>128</td>
<td>41</td>
<td>181</td>
</tr>
<tr>
<td>D2080</td>
<td>78</td>
<td>56</td>
<td>375</td>
</tr>
</tbody>
</table>

for a specific orchard.

- Pest disease entries have been reset to 0 before recording of the first entry in the database for a specific orchard.

A.3 Test Case Scenarios – Inference Engine V1

A.3.1 Orchard D1164: Pest Trapping Information

Case 1:

No. traps in orchard: 3

Trap entry: 2 Codling moth caught in one of the traps on 22/12/1998

IFP rules that fired: None

Recommendation: Most recent spray application was 21/12/1998 using MIMIC. Continue trapping. Do not apply another spray for this pest for at least 20 days.

Case 2:

No. traps in orchard: 3

Trap entry: 2 Leafroller caught on 29/12/1998

IFP rules that fired: None

Recommendation: Most recent spray application was 21/12/1998 using MIMIC. Continue trapping. Do not apply another spray for this pest for at least 13 days.
**Case 3:**

*No. traps in orchard:* 3  
*Trap entry:* 3 Codling moth caught on 01/10/1998  
*IFP rules that fired:* None  
*Recommendation:* You are trapping too early! Traps not recommended to be placed in field until late November/December for the Hawkes Bay Region. In general moths are not active until mid October at the earliest.

**Case 4:**

*No. traps in orchard:* 3  
*Trap entry:* 3 Codling moth caught on 01/11/1998  
*IFP rules that fired:* codling-moth-PetalFall-Nor-Nel-Mar  
*Recommendation:* Most recent spray application was 28/10/1998 using CAR80W. The first spray for codling moth in Hawkes Bay is highly recommended is Match or Mimic at Petal fall or in early November. It is possible that a new spray application can be applied. The withholding period for CAR80W has ended and it has been 2 days since you last sprayed. Application of Match or Mimic (observe WHP) is highly recommended.  
*Note:* The next PCR entry is on 30/10/1998 where MIMIC was applied.

**Case 5:**

*No. traps in orchard:* 3  
*Trap entry:* 3 Codling moth caught on 15/12/1998  
*IFP rules that fired:* None  
*Recommendation:* Most recent spray application was 01/12/1998 using LORSEC. Continue trapping. Do not apply another spray for this pest for at least another 14 days.
A.3.2 Orchard D1164: Pest monitoring

At present these cases relate to testing the rules in the inference engine without any other information available about the dynamics of the orchard. They have been developed to test that the rules actually fire when they should given the relevant sample sizes, number of samples damaged and the date that this data was entered.

Case 1:
*Pest disease entry:* 1000 fruit sampled from past season, 0% fruit infested with mealybug. First year in IFP.
*Date recorded:* 11/11/1998
*Rules that fired:* mealybug-GreenTip-Nor-Nel-Mar
*Recommendation:* Application of Oil + Applaud OR oil + chlorpyrifos is recommended

Case 2:
*Pest disease entry:* 1000 fruit sampled from past season, 0% fruit infested with mealybug. Second year in IFP.
*Rules that fired:* none
*Recommendation:* none

Case 3:
*Pest disease entry:* 1000 fruit sampled, 0.1% fruit infested with mealybug. Second year in IFP.
*Rules that fired:* mealybug-GreenTip-Nor-Nel-Mar
*Recommendation:* Application of Oil + Applaud OR oil + chlorpyrifos at green tip is recommended.

Case 4:
*Pest disease entry:* 1000 fruit sampled from previous harvest assessment, 0.5% fruit infested with mealybug. Second year in IFP.
*Rules that fired:* mealybug-TightCluster-Nor-Nel-Mar

*Recommendation:* Application of Oil + Applaud OR oil + chlorpyrifos at green tip is recommended then a tight cluster application of Applaud is recommended.

**Case 5:**

*Pest disease entry:* 1000 fruit sampled from previous harvest assessment, 1% fruit infested with mealybug. Second year in IFP.

*Rules that fired:* mealybug-LateNov-Nor-Nel-Mar

*Recommendation:* Application of Oil + Applaud OR oil + chlorpyrifos at green tip is recommended then a tight cluster application of Applaud is recommended then an application of chlorpyrifos in late November or early December.

**Case 6:**

*Pest disease entry:* 1000 fruit sampled from previous harvest assessment, 0% fruit infested with scale. Second year in IFP.

*Rules that fired:* scale-GreenTip1-AllDistricts

*Recommendation:* Application of oil before is recommended.

**Case 7:**

*Pest disease entry:* 1000 fruit sampled from previous harvest assessment, 0.1% fruit infested with scale. Second year in IFP.

*Rules that fired:* scale-GreenTip2-AllDistricts

*Recommendation:* Application of oil + Applaud is recommended or oil + chlorpyrifos is recommended and then sample 500 fruit in mid to late January.

**Case 8:**

*Pest disease entry:* 1000 fruit sampled from previous harvest assessment, 0.6% fruit infested with scale. Second year in IFP.

*Rules that fired:* scale-GreenTip3-AllDistricts
**Recommendation:** Application of oil + Applaud is recommended or oil + chlorpyrifos is recommended. Also treat with summer oil, OR diazinon, OR chlorpyrifos in mid to late January. Finally sample 500 fruit two weeks after treatment.

**Case 9:**

**Pest disease entry:** Scale present from previous harvest. 500 fruit sampled from current season, 0.05% fruit infested with scale. Second year in IFP.

**Date sampled:** 18/11/1999

**Rules that fired:** scale-MidJanuary2-AllDistricts

**Recommendation:** Application of summer oil, diazinon, OR chlorpyrifos is recommended.

**Case 10:**

**Pest disease entry:** Scale present from previous harvest. 500 fruit sampled from current season, 1% fruit infested with scale. Second year in IFP.

**Date sampled:** 18/11/1999

**Rules that fired:** scale-MidJanuary3-AllDistricts

**Recommendation:** Application of summer oil, diazinon, OR chlorpyrifos after 2-3 weeks is recommended. NOTE: observe withholding periods.

**Case 11:**

**Pest disease entry:** 1000 fruit sampled from previous harvest assessment, 1% fruit infested with appleleaf curling midge. Second year in IFP.

**Rules that fired:** appleleaf-TightCluster-AllDistricts

**Recommendation:** Ground application of diazinon is recommended.

**Case 12:**

**Pest disease entry:** Appleleaf curling midge present from previous harvest. 40 shoots sampled from current season, 21% shoots infested with appleleaf curling midge. Second year in IFP.
Date sampled: 30/11/1998  
Rules that fired: appleleaf-LateNov-AllDistricts  
Recommendation: Foliar application of diazinon is recommended.

Case 13:  
Pest disease entry: Appleleaf curling midge present from previous harvest. 40 shoots sampled from current season, 21% shoots infested with appleleaf curling midge. Second year in IFP.  
Date sampled: 30/11/1998  
Rules that fired: appleleaf-January-AllDistricts  
Recommendation: Foliar application of diazinon is recommended.

Case 14:  
Pest disease entry: 100 shoots sampled from past season, 0% shoots infested with Woolly apple aphid. First year in IFP.  
Rules that fired: waa-GreenTip-AllDistricts  
Recommendation: Application of Oil + Pirimor OR oil + chlorpyrifos is recommended.

Case 15:  
Pest disease entry: 100 shoots sampled from past season, 0% shoots infested with Woolly apple aphid. Second year in IFP.  
Date recorded: 20/12/1998  
Rules that fired: waa-NovJan1-AllDistricts  
Recommendation: Resample in four weeks time.

Case 16:  
Pest disease entry: 100 shoots sampled from current season, 5 shoots infested with Woolly apple aphid. Second year in IFP.  
Date recorded: 20/12/1998
Rules that fired: waa-NovJan2-AllDistricts
Recommendation: Resample in two weeks time.

Case 17:
Pest disease entry: 100 shoots sampled from current season, 25 shoots infested with Woolly apple aphid. Second year in IFP.
Date recorded: 20/12/1998
Rules that fired: waa-NovJan3-AllDistricts
Recommendation: Application of Pirimor or chlorpyrifos is recommended then resample in four weeks time.

Case 18:
Pest disease entry: 100 shoots sampled from current season, 65 shoots infested with Woolly apple aphid. Second year in IFP.
Date recorded: 20/12/1998
Rules that fired: waa-NovJan4-AllDistricts
Recommendation: Application of Pirimor or chlorpyrifos is recommended then resample in two weeks time.

A.4 Test Case Scenarios – Inference Engine V2

Modifications to the original inference engine:

1. The database was updated to take into account the different material ID for the same insecticide.

2. If the grower is the first time in the IFP. The database is updated automatically with green tip and tight cluster spray dates at the beginning of the season.. Otherwise a search is made of the previous season’s dates for green-tip and these dates are automatically inserted into the database. These dates however do not have a bearing on
the other spray decisions made. The reminder for green tip sprays are not necessarily on a specified date but within +/- 7 days of the green tip date entered by the grower. The green tip dates can be modified but this is an operation that is performed by the user-interface of the IPMES.

3. After an inference is made on the basis of the current state of the orchard, any reminders that have been added to the database in previous sessions with the IPMES are then checked to see if they fall into the time period for action by the grower. If this is the case then these reminders are also added to the overall advice presented to the grower. These reminders will not be displayed to the grower if there is either monitoring data for that particular pest or a PCR entry in the database within that time period.

4. The database is then updated with any reminders related to acquiring of trap catch data, spraying for pests, or sampling for pests.

A.4.1 Orchard D1164: Pest monitoring

Case 1:

Status: First Year IFP Grower

This is slightly different from the normal reporting by the inference engine. As the orchardist has specified that they are First Year IFP growers, a set of reminders that prompt the orchardist for spraying at Green Tip and Tight Cluster have been added. These dates are 1-15 September for green tip and 16-25 September for tight cluster. The output of the system over a period of time looks like this:

30/08/1998 - No reminders prompted.


This message is automatically added to any recommendation inferred by the machine until one of two scenarios occur:
1. An entry is made in the PCR database within that period of time.

2. After Green Tip or Tight Cluster period has expired, the message then reads, “Application of spray at Tight Cluster not applied”. This message appears only once.

Note: For orchard D1164, applications of insecticides were made of 01/09/1998 and 17/09/1998. Hence the grower was prompted for no reminders.

**Case 2:**

*Pest disease entry:* 1000 fruit sampled from previous harvest assessment, 1% fruit infested with mealybug. Second year in IFP.

*Rules that fired:* mealybug-LateNov-Nor-Nel-Mar. This rule is fired at the beginning of the season and the recommendation added to the reminders database.

*Recommendation:* Application of Oil + Applaud OR oil + chlorpyrifos at green tip is recommended then a tight cluster application of Applaud is recommended then an application of chlorpyrifos in late November or early December.

*Additions to the reminder database:* Two reminders for Green Tip and Tight Cluster applications are entered.

*Reminder Activated:* 23/11/1998 Application of chlorpyrifos required for two weeks from 23 November to beginning of December. This reminder is prompted every day.

*Actioned (entry in actual PCR database):* 01/12/1998 PCR entry for insecticide application made.

**Case 3:**

*Pest disease entry:* 100 shoots sampled from current season, 25 shoots infested with Woolly apple aphid. Second year in IFP.

*Date recorded:* 20/12/1998

*Rules that fired:* waa-NovJan3-AllDistricts

*Recommendation:* Application of Pirimor is recommended if this is the only pest to be controlled. Application of chlorpyrifos or diazinon is recommended if other insect pests
need control also. In all cases resample in four weeks time.

*Additions to reminder database:* A reminder to resample for Woolly Apple Aphid in four weeks.

*Reminder Activated:* 20/01/1999 Sample for Woolly Apple Aphid as it has been four weeks. This reminder is prompted every day.

*Actioned (entry in actual pest monitoring database):* 25/01/1999 Pest monitoring entry made for Woolly Apple Aphid.
Appendix B

Timings for the experiments conducted in this thesis

B.1 FWT versus DCT versus FFT

A timing on how long it took to transform an 128x128 colour image of pest damage was used as the basis for the experiment. When the RGB image was read in it was separated into the Red, Green, and Blue components of the image then a FWT\(^1\), FFT\(^2\), and DCT applied to each component using MATLABR14 SP3. The FWT applied was a 2D 4-Layer decomposition of a Daubechies wavelet with 8 bases. The average time to process all 96 images is contained in Table B.1.

<table>
<thead>
<tr>
<th>Transformation</th>
<th>Average Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FWT</td>
<td>0.0224</td>
</tr>
<tr>
<td>DCT</td>
<td>0.0087</td>
</tr>
<tr>
<td>FFT</td>
<td>0.0016</td>
</tr>
</tbody>
</table>

\(^1\)FWT using the WaveLab802 wavelet toolbox from http://www.stat-stanford.edu/~wavelab
\(^2\)Heavily optimised for speed using the FFTW library from http://www.fftw.org
B.2 Iris benchmark data set

- Benchmark Iris data set (four input attributes, three output classes, 150 examples)

- Number of training examples=105, number of testing examples=45

- $T =$ timing in seconds. Note: MLP, $k$-Means, and SVM were time and memory optimised functions in PRTools toolbox\(^3\)

- **EFuNN Parameters:** Str=0.95, Err=0.1, Lr=0.01, Old=90, Prune=90, numMF=3

- **$k$-Means classifier:** three clusters = number of classes

- **MLP:** 10 hidden nodes, error goal=0.02, momentum=0.95, learning rate=0.01, epochs=50

- **SVM:** A polynomial of degree 4

<table>
<thead>
<tr>
<th>Run</th>
<th>T(EFuNN)</th>
<th>Nodes</th>
<th>Accuracy</th>
<th>T($k$-Means)</th>
<th>Accuracy</th>
<th>T(MLP)</th>
<th>Accuracy</th>
<th>T(SVM)</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.07</td>
<td>81</td>
<td>95.56</td>
<td>0.05</td>
<td>91.11</td>
<td>3.92</td>
<td>93.33</td>
<td>0.07</td>
<td>100.0</td>
</tr>
<tr>
<td>2</td>
<td>0.99</td>
<td>81</td>
<td>97.78</td>
<td>0.06</td>
<td>95.56</td>
<td>4.02</td>
<td>95.56</td>
<td>0</td>
<td>97.78</td>
</tr>
<tr>
<td>3</td>
<td>0.90</td>
<td>82</td>
<td>93.33</td>
<td>0.01</td>
<td>93.33</td>
<td>4.71</td>
<td>93.33</td>
<td>0</td>
<td>97.78</td>
</tr>
<tr>
<td>4</td>
<td>0.89</td>
<td>82</td>
<td>95.56</td>
<td>0.03</td>
<td>95.56</td>
<td>4.00</td>
<td>97.78</td>
<td>0</td>
<td>100.0</td>
</tr>
<tr>
<td>5</td>
<td>0.83</td>
<td>81</td>
<td>95.56</td>
<td>0.03</td>
<td>97.78</td>
<td>6.24</td>
<td>95.56</td>
<td>0</td>
<td>91.11</td>
</tr>
<tr>
<td>6</td>
<td>0.27</td>
<td>82</td>
<td>97.78</td>
<td>0.04</td>
<td>97.78</td>
<td>13.21</td>
<td>100.0</td>
<td>0</td>
<td>95.56</td>
</tr>
<tr>
<td>7</td>
<td>1.05</td>
<td>82</td>
<td>95.56</td>
<td>0.01</td>
<td>95.56</td>
<td>8.08</td>
<td>97.78</td>
<td>0.01</td>
<td>97.78</td>
</tr>
<tr>
<td>8</td>
<td>1.04</td>
<td>81</td>
<td>95.56</td>
<td>0.06</td>
<td>100.0</td>
<td>6.53</td>
<td>100.0</td>
<td>0</td>
<td>95.56</td>
</tr>
<tr>
<td>9</td>
<td>0.98</td>
<td>80</td>
<td>95.56</td>
<td>0.08</td>
<td>95.56</td>
<td>4.29</td>
<td>95.56</td>
<td>0</td>
<td>95.56</td>
</tr>
<tr>
<td>10</td>
<td>1.12</td>
<td>81</td>
<td>97.78</td>
<td>0.07</td>
<td>95.56</td>
<td>8.30</td>
<td>93.33</td>
<td>0</td>
<td>91.11</td>
</tr>
</tbody>
</table>

\(^3\)http://www.prtools.org
B.3 Timings of training for the different classifiers

- Ten runs were taken for training and the results averaged\(^4\)
- \(T = \) timing in seconds. Note: MLP, \(k\)-Means, and SVM were time and memory optimised functions in PRTools toolbox \(^5\)
- FuNN used optimised functions from the MATLAB Neural Network Toolbox

<table>
<thead>
<tr>
<th>Data set</th>
<th>no. Inputs</th>
<th>no. Examples</th>
<th>T(EFuNN)</th>
<th>T(k-Means)</th>
<th>T(FuNN)</th>
<th>T(MLP)</th>
<th>T(SVM)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pest Data templates</td>
<td>768</td>
<td>115</td>
<td>3.81</td>
<td>0.12</td>
<td>N/A</td>
<td>5.08</td>
<td>0.019</td>
</tr>
<tr>
<td>MRI scan data</td>
<td>768</td>
<td>33</td>
<td>0.46</td>
<td>0.05</td>
<td>N/A</td>
<td>2.78</td>
<td>0.009</td>
</tr>
<tr>
<td>Persimmon data</td>
<td>20</td>
<td>50</td>
<td>0.2735</td>
<td>N/A</td>
<td>3.93</td>
<td>5.59</td>
<td>0.001</td>
</tr>
<tr>
<td>Pine data</td>
<td>310</td>
<td>200</td>
<td>5.1357</td>
<td>N/A</td>
<td>9.68</td>
<td>13.4</td>
<td>0.007</td>
</tr>
</tbody>
</table>

- Pest Data templates
  - **EFuNN Parameters**: Str=0.95, Err=0.1, Lr=0.01, numMF=2
  - **\(k\)-Means classifier**: 68 prototypes
  - **MLP**: 10 hidden nodes, error goal=0.02, momentum=0.95, learning rate=0.01, epochs=200
  - **SVM**: A polynomial of degree 4
- MRI scan data
  - **EFuNN Parameters**: Str=0.95, Err=0.01, Lr=0.01, numMF=3
  - **\(k\)-Means classifier**: 33 prototypes
  - **MLP**: 10 hidden nodes, error goal=0.02, momentum=0.95, learning rate=0.01, epochs=200
  - **SVM**: A polynomial of degree 4
- Persimmon data
  - **EFuNN Parameters**: Str=0.1, Err=0.2, Lr=0.25 numMF=2, Old=50, Prune=50, numMF=2

\(^4\)Timings from MATLAB 7.0r14 on a 2GHz PC with 1GB RAM running Windows 2000 SP4.
\(^5\)http://www.prtools.org

242
– **FuNN Parameters**: 10 hidden nodes, error goal=0.001, momentum=0.8, learning rate=0.01, epochs=1000, numMF=2

– **MLP**: 10 hidden nodes, error goal=0.001, momentum=0.8, learning rate=0.01, epochs=1000

– **SVM**: A polynomial of degree 4

• Pine data

– **EFuNN Parameters**: Str=0.1, Err=0.2, Lr=0.25 numMF=2, Old=200, Prune=200, numMF=2

– **FuNN Parameters**: 10 hidden nodes, error goal=0.001, momentum=0.8, learning rate=0.01, epochs=1000, numMF=2

– **MLP**: 10 hidden nodes, error goal=0.001, momentum=0.8, learning rate=0.01, epochs=1000

– **SVM**: A polynomial of degree 4