

Copyright is owned by the Author of the thesis. Permission is given for a copy to be downloaded by an individual for the purpose of research and private study only. The thesis may not be reproduced elsewhere without the permission of the Author.

Fuzzy Neural Network Interface: Development and Application

A thesis presented in partial fulfilment of the requirements for the degree of Master of
Engineering in Information Engineering at Massey University, Palmerston North,
New Zealand.

Greg Todd
2003

Abstract

This project is concerned with the development and application of an interface for a fuzzy neural network (FuNN). The original program, for which the interface was written, is a tool to research the mapping of problem knowledge to initialize the weights of a FuNN. The interface concentrates on allowing the user to efficiently manipulate network settings and to be able to easily perform large numbers of experiments. After the interface was completed, the new integrated application was used to investigate the use of problem knowledge on FuNN training in specific image processing problems.

Acknowledgements

I would like to thank my supervisor Dr. Sanj Gunetilleke for his tireless patience and help.

Contents

Abstract	i
Acknowledgements.....	ii
Contents.....	iii
Chapter 1: Introduction.....	1
Overview.....	1
Contents by chapter	1
Chapter 2: Background.....	3
Introduction.....	3
Artificial Neural Networks	5
Fuzzy Neural Networks	6
Image Processing.....	11
Rule Mapping.....	12
Chapter 3: Interface Development	14
Introduction.....	14
Planning and Analysis: Project Plan.....	14
Planning and Analysis: Analysis.....	17
Planning and Analysis: Requirements.....	18
Design.....	18
Major Design Issues	19
Requirements Elicitation.....	23
Coding.....	25
Chapter 4: Interface Results and Testing.....	26
Introduction.....	26
Graphical User Interface Results.....	26
Testing	32
Chapter 5: Iris Edge Detection	36
Introduction.....	36
Preparing the image.....	36
The Experiments	40
Post Process Procedure.....	43
Conclusion	46
Chapter 6: Pollen Classification	47
Introduction.....	47
Methods for Finding Features of Pollen Images.....	47
General Methodology.....	56
Extraction of SGLDM Data.....	58
Experiments: Training Neural Network ‘A’	59
Experiments: Training Neural Network ‘B’	63
Experiments: Training Neural Network ‘D’	65
Experiments: Combining Neural Networks.....	66
Discussion and Conclusion	70
Chapter 7: Conclusions and Discussion.....	71
Appendix 1: Pollen List.....	73
Appendix 2: User Manual.....	75
Appendix 3: Code Listings.....	82
References.....	160

Chapter 1: Introduction

Overview

This thesis can effectively be broken down into two main sections:

The primary aim of this research is to produce a functional graphical user interface for a fuzzy neural network (FuNN) application, for use by academics and students studying the use of neural networks. The original program was developed by Gunetileke (2001), and though fully functional contained only a basic, mostly command line interface. The user interface will conform as closely as possible to the client requirements.

The secondary aim is to use the program to conduct experiments to determine the effect of using neural networks on two specific image-processing problems. The first problem will involve categorising various types of pollen. Textural and shape information, such as entropy, energy and area will be used to train the neural network, so it can identify each pollen type. The second problem will involve finding the best combination of inputs, into a neural network that will detect the iris edge from test images. Two or three significant inputs will be investigated to determine what effect they have on the final output.

Contents by chapter

Chapter 2 reports some of the relevant previous work in the field of FuNN's. This includes components of image processing, neural networks and Boolean logic.

Chapter 3 details the most important parts of the development of the interface for the SuperFuNN application program. It describes the life cycle development process, planning and analysis, requirements elicitation, design and design considerations.

In Chapter 4, the results of the interface development project are presented. This includes screen shots of the actual interface and descriptions of its features. It also includes information on user interface testing and function testing including an example unit and integration test.

Chapter 5 describes the first of the two image processing problems, iris edge detection. Although the iris edge detection problem has been looked at by previous researchers, this experiment looked at the effects of changing specific input variables to find the best combination for this problem.

Chapter 6 describes the problem of pollen classification using image processing and fuzzy neural networks. It describes a series of experiments dealing with the problem, and their results.

Chapter 7 looks at the usefulness of the interface, after having used it for real world problems. It also mentions ideas for future work to be done.

In Appendix 1, the list of pollen types with their scientific name, picture and there corresponding codes used during testing are displayed.

Appendix 2 contains Chapters One Two and Three of the user manual. Chapter one describes set-up and installation instruction, while Chapter Two is about 'Getting Started' with the SuperFuNN application. Chapter Three describes an actual example experiment. It also contains a list of common user mistakes and current application shortcomings.

In Appendix 3, the program code for the interface is listed. It includes all functions that were developed specifically for the interface, as well as those existing functions (from the original application) that have been significantly altered.

Chapter 2: Background

Introduction

This chapter will explain some of the background knowledge used during the course of this project. Fuzzy Neural networks (FuNN's) combine the fields of image processing, fuzzy logic and artificial neural networks.

Fuzzy Logic

Many of the terms people use in their description of objects and situations are difficult to model using mathematical equations. Often imprecise terms such as large, small, close, far, tall and short are used as descriptors in real world situations. However, these are difficult to translate into a mathematical form that might be suitable for input into computer programs. Such terms can be called 'fuzzy' because they cannot be sharply defined (Nguyen, 1997). Mathematical modelling of fuzzy concepts, now known as fuzzy logic, was presented by Zadeh in 1965. His contention was that meaning in natural language is a matter of degree. If we have the proposition that 'John is young', then it is not always possible to assert this is true or false. If John's age is x , then the compatibility of x with 'is young' is a matter of degree. This suggests that membership in a fuzzy subset should not be on a 0 or 1 (false or true) basis, but rather on a 0 to 1 scale. Figure 2.1 below shows how one might model the fuzzy concept of young, based on age (x), using the following model:

$$Y(x) = \begin{cases} 1 & \text{if } x < 40 \\ \frac{80 - x}{40} & \text{if } 40 \leq x \leq 60 \\ \frac{70 - x}{20} & \text{if } 60 < x \leq 70 \\ 0 & \text{if } 70 < x \end{cases}$$

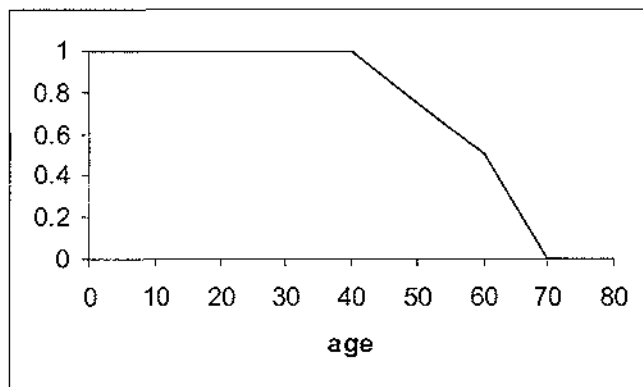


Figure 2.1 A fuzzy membership function for young.

Example of a Rule

Membership functions can be developed for both the antecedent (if ~) and consequent (then ~) parts of a rule. Figure 2.2 shows the membership functions for the antecedent and consequent of the following rule:

If a pollen is *big* then the likelihood it is a type 'A' is *high*.

Problem knowledge for current image processing problem can be described similarly.

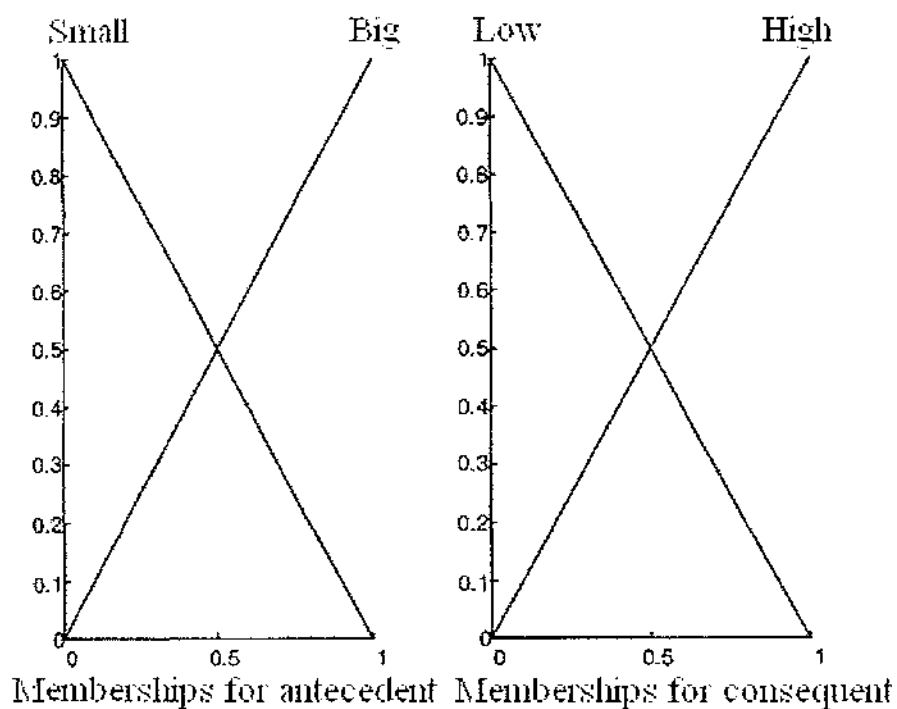


Figure 2.2 Membership function for an example rule

Artificial Neural Networks

The human brain is an example of a biological neural network. It consists of an enormously complex system of neurons, synapses, axons and dendrites. Haykin (1994) describes a neural network as a massively parallel distributed processor that has a natural propensity for storing experiential knowledge and making it available for use. He notes that it resembles the brain in two respects:

- Knowledge is acquired by the network through a learning process
- Interneuron connection strengths known as synaptic weights are used to store the knowledge

Generally speaking, an artificial neural network tries to replicate the biological neural network through the use of computers. An artificial neural network, like the human brain, can learn by experience. A 'learning' neural network makes an iterative process of adjustments to adapt to its environment. There are three basic classes of learning paradigm: supervised learning, reinforcement learning and self-organised (unsupervised) learning. Supervised learning is performed under the supervision of an external teacher. Reinforcement learning is learning of input-output mapping through a process of trial and error. In unsupervised learning, there is no external teacher to instruct synaptic weight changes in the network (Haykin, 1994). The results change as a result of the nature of the data present. The FuNN developed by Kasabov (1996) used the supervised type training algorithm, as did Gunetileke (2001) when he developed his FuNN application program.

Usually the training weights of a neural network are initialised with small random numbers. Researchers have looked at using problem knowledge to help initialise a network. Proper initialisation is one of the prerequisites for fast convergence of a feed-forward network (Thimm *et al.*, 1997). There are various ideas that have been proposed for the initialisation of neural networks. Siroki (1998) and Gunetileke (2001) studied a problem knowledge based weight initialisation scheme. Rules representing problem knowledge are generated in an 'if ~ then ~' form, and used to

initialise the input weights into a network. The rules are implemented using a fuzzy neural network architecture.

Fuzzy Neural Networks

A Fuzzy Neural network is an architecture that combines neural networks and fuzzy logic (Kasabov, 1996). Kasabov notes that a FuNN consists of five layers, represented diagrammatically below in figure 2.2. For the input layer, a node represents an input variable as crisp values. These values are fed into the condition layer which performs the ‘fuzzification’ by triangular membership functions with the centres represented as the weights (Siroki, 1998). The values from the input layer that are fed into the condition layer are all in the range 0 – 1. Gunetileke (2001) notes that expert knowledge can be used to initialise the spacing of the membership functions. An important aspect of this layer is that different inputs can have different numbers of membership functions.

The output of the condition layer is passed to the rule layer, where each node represents a single ‘fuzzy rule’. The rule layer is equivalent to the hidden layer of a multi layer perceptron (MLP) network. The activation of the rule layer node is the degree to which input data matches the antecedent component of an associated rule. Outputs from the rule layer are fed to the action element layer. In this layer each node represents a fuzzy label from the fuzzy quantification space of an output variable, for example “short”, “medium” or “long”. The activation of the node represents the degree to which this membership function is supported by the current data (Siroki, 1998).

Siroki (1998) explains that the output layer performs a modified centre of gravity defuzzification. Singletons representing centres of triangular membership functions are attached to the connections from the action to the output layer. Linear activation functions are used in this layer.

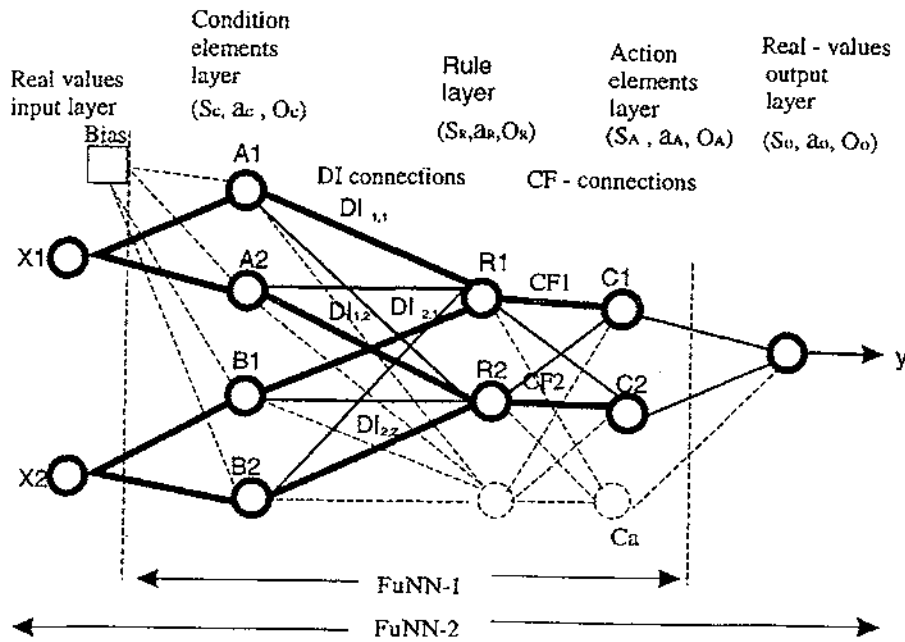


Figure 2.2 Architecture of a FuNN (Kasabov, 1996)

There are three methods to update the weights of a FuNN. The method used by Gunetileke (2001) for his FuNN application was, a partially adaptive version where the membership functions of the input and output variables do not change during training. The MLP section was trained using a modified back-propagation algorithm.

The training algorithm

This section describes the algorithm used for FuNN training as described in Gunetileke (2001). The algorithm is discussed in terms of neurons as they are the focus of the fuzzy architecture.

Forward Pass

The activation values for all the nodes in the network, from the first to the fifth layer are computed during this phase. A superscript indicates the layer and a subscript indicates a connection between layers.

Input Layer (first layer)

The input layer of neurons represents the input variables as crisp values. These values are fed into the condition layer which performs fuzzification.

Condition Layer (second layer)

This layer acts as a fuzzifier. The output from this layer is the degree to which the input belongs to the given membership function. The input weight to the condition node represents the centre for that particular membership function with the minimum and maximum determined using the centre of the adjacent membership functions. For the first and last membership functions for a variable, a shoulder is used instead. Each membership function is triangular and an input signal activates only two neighbouring membership functions simultaneously. The sum of the grades of these two membership functions for any given input is always one. For a triangular membership function the activation functions for a node i are:

$$\text{If } a_i < x < a_{i+1} \text{ then } Act_i^c = 1 - \frac{x - a_i}{a_{i+1} - a_i}$$

$$\text{If } a_{i-1} < x < a_i \text{ then } Act_i^c = 1 - \frac{a_i - x}{a_i - a_{i-1}}$$

If $x = a_i$ then $Act_i^c = 1$, where a_i is the centre of the triangular membership function.

Rule Layer (third layer)

The pre-condition matching of fuzzy rules is done through the connections from the condition layer to the rule layer. The connection weights in this layer may be set either randomly or according to a set of rules. The net input is given by the formula:

$$Net^r = \sum w_{rc} \times Act^c$$

while the activation is given by:

$$Act^r = \frac{1}{1 + e^{(-g \times Net^r)}} \text{ , where } g \text{ is the gain factor.}$$

Action Layer (forth layer)

The nodes in this layer and the connection weights function as those in the rule layer for net input and activation:

$$Net^a = \sum w_{ar} \times Act^r$$

$$Act^a = \frac{1}{1 + e^{(-g \times Net^a)}}$$

Output Layer (fifth layer)

Defuzzification to produce a crisp output value is performed in this layer. The centre of gravity (COG) defuzzification is used to convert from fuzzy to crisp.

$$Net^o = \sum w_{oa} \times Act^a$$

$$Act^o = \frac{Net^o}{\sum Act^a}$$

Backward Pass

The goal for this phase is to minimise the error function:

$$Error = \frac{1}{2} \sum (y^d - y^a)^2$$

where y^d is the desired output and y^a is the current output.

Hence the general learning rule (gradient descent) used is:

$$\Delta w \approx -\frac{\partial E}{\partial w}$$

$$w_{i+1} = w_i + \eta \left(-\frac{\partial E}{\partial w} \right) + \alpha (\Delta w_i)$$

where η is the learning rate and α is the momentum coefficient, and

$$\frac{\partial E}{\partial w} = \frac{\partial E}{\partial Net} \times \frac{\partial Net}{\partial w} = -\delta \times Act$$

Thus, the weight update rule is:

$$\Delta w_{i+1} = \eta \delta \times Act + \alpha \Delta w_i$$

Output Layer (fifth layer)

When the weights are adapted the constraining rule is taken into account, which imposes restrictions to the change of the centres of the membership functions.

$$\delta^o = -\frac{\partial E}{\partial Net^o} = -\frac{\partial E}{\partial Act^o} \times \frac{\partial Act^o}{\partial Net^o} = y^d - y^a$$

Action Layer (forth layer)

The error for each node in this layer is calculated individually based on the output error and on the activation of this node having in mind the type of membership functions (triangular) used in the defuzzification layer as well as the type of defuzzification.

$$\text{If } a_i < y < a_{i+1} \text{ then } d^a = 1 - \frac{y - a_i}{a_{i+1} - a_i}$$

$$\text{If } a_{i-1} < y < a_i \text{ then } d^a = 1 - \frac{a_i - y}{a_i - a_{i-1}}$$

$$\text{If } y = a_i \text{ then } d^a = 1.$$

Hence,

$$\delta^a = f(Net^a) \times E^a = Act^a (1 - Act^a) \times (d^a - Act^a)$$

Rule Layer (third layer)

$$\delta^r = Act^r (1 - Act^r) \times \sum (w_{ar} \times \delta^a)$$

Condition Layer (second layer)

The weight w_{ic} is assigned as follows. If x^i lies in the fuzzy segment, then the corresponding weight should be increased directly proportionally to the propagated error from the previous layer as the error is caused by the weight. This proposition can be expressed by the following equation:

$$\delta^c = Act^c \times \sum (w_{rc} \times \delta^r)$$

Thus, the weight updating rule for this layer is:

$$w_{ic,i+1} = \eta \delta^c + \alpha \Delta w_{ic,i}$$

The new centres of the input triangular membership functions are also adjusted according to a partition range as for the output layer (Gunetileke, 2001).

Image Processing

The values fed into the FuNN can be gleaned from images through the implementation of image processing techniques. Image processing refers to the altering or analysing of images through a variety of techniques. When using computers, images are normally read in as matrix of numbers, each number (or set of three numbers for colour images) represents one pixel or 'dot' on the image. Mathematical operations can be performed on this 'matrix' to obtain information from it, and even change it.

Window filters can be used to obtain information about groups of pixels in the image as shown in figure 2.2 below. In this figure, the output image is defined as a function of the pixel in a window surrounding the equivalent position in the input image Gunetileke (2001). The filtering operation can be based on linear, non-linear or rule based functions.

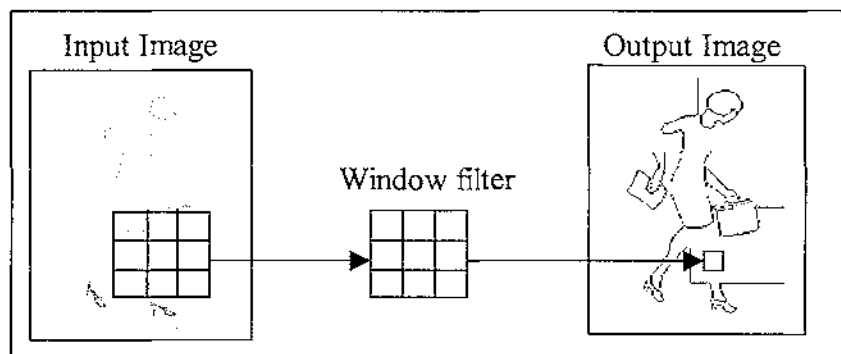


Figure 2.2 Operation of a window filter.

Often the selection of an appropriate window filter for image processing is an empirical process, which relies on skill and experience (Siroki, 1998). Sometimes it is possible to train a neural network to perform the filter operation. As most neural networks require both input and target variables to learn, so to does the neural network window filter (NNWF) (Gunetileke, 2001). Thus, for each image used for training the NNWF, an appropriate target image must be created.

Gunetileke (2001) describes the use of a fuzzy neural network window filter for use in image processing problems. The FuNNWF is represented visually in the figure 2.3 below.

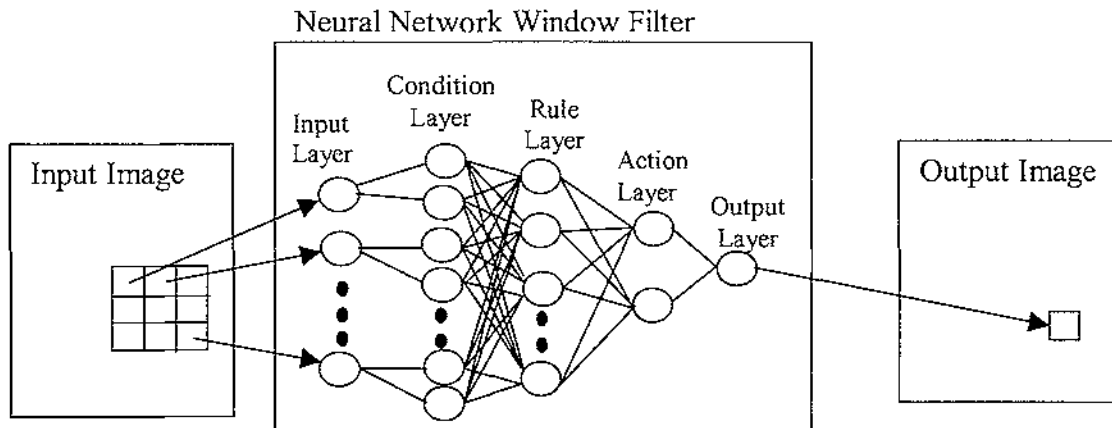


Figure 2.3 A Fuzzy neural network window filter.

Rule Mapping

Expert knowledge about a problem can be transformed into low level 'rules' that define the problem. These rules are then mapped to the weights of a FuNN during its initialisation to assist in its training. The main aims of the rules are to:

- Allow the network to start training from a position closer to the final solution
- To cover regions of the input-target space for which there is no training data
- To increase the robustness of the network

Two methods of mapping have been developed: Boolean logic rule mapping and conditional rule mapping. Boolean logic rule mapping effectively produces rules in the form:

If $I1=M2$ $I2=M1$ Then $O1=M2$

Where I - represents input values into the network, M - represents fuzzy membership values (for example $M1=0$, $M2=1$) and O - stands for network 'output'.

Conditional rule mapping produces rules of the form:

If $I1<M2$ $I2>M3$ Then $O1=M2$

Gunetileke (2001) notes that conditional rules are more powerful than Boolean logic rules. For a more detailed explanation of rule mapping see Gunetileke (2001) chapter 3.

Usually a 'complete' set of rules for describing a real world image processing problem cannot be derived. In this case the number of rule nodes in the network is made larger than the number of rules (creating 'free nodes'), so the network has enough freedom to adapt. The rules developed for real world image processing problems are unlikely to be 100 percent correct or complete. In this case the 'saturation value' associated with the rules can be adjusted according to the problem. A high saturation value represents high certainty in the correctness and completeness of each rule and effectively stops the training from changing the weights. A low saturation value means the rules are incomplete and/or uncertain and allows the training to alter the weight values relatively easily. Also, the 'quality factor' setting can be adjusted depending on the certainty of a rule. A high quality factor should be used for strong rules and a low quality factor used for weak rules. An incorrect rule with a high quality factor can have a degrading effect on network performance, as the correct rules and free nodes will try to compensate for them. As quality factor drops the weights associated with the incorrect rules can change more easily and the weights associated with the rule nodes change less (Gunetileke, 2001).

Chapter 3: Interface Development

Introduction

This chapter details the main points of the development of the application interface. It includes aspects of planning and analysis, design, coding and testing.

The largely command line interface of the original fuzzy neural network application, was readily useable only by those already familiar with the application (i.e. its author). It was restricted in its capability to perform experiments, because it often required the source code to be altered to perform different experiments. Whenever any of the input variables used in the creation of a network needed to be changed, there was no mechanism in the interface to allow this to happen, without changing the code. Thus, users without programming experience in the MATLAB programming environment were not able to use the application effectively.

Planning and Analysis: Project Plan

Purpose

The purpose of the application is to provide an interface to the Neural Network program written by Dr Gunetileke. The application is to be used primarily by academic staff (or possibly students) who already possess a reasonable knowledge in the field of Neural Networks.

Life Cycle Model

Development of the program will be done in the MATLAB environment, which uses a fairly unique 'development orientated' language. So to keep things simple, a fairly standard life-cycle model (shown below in figure 3.1) will be used.

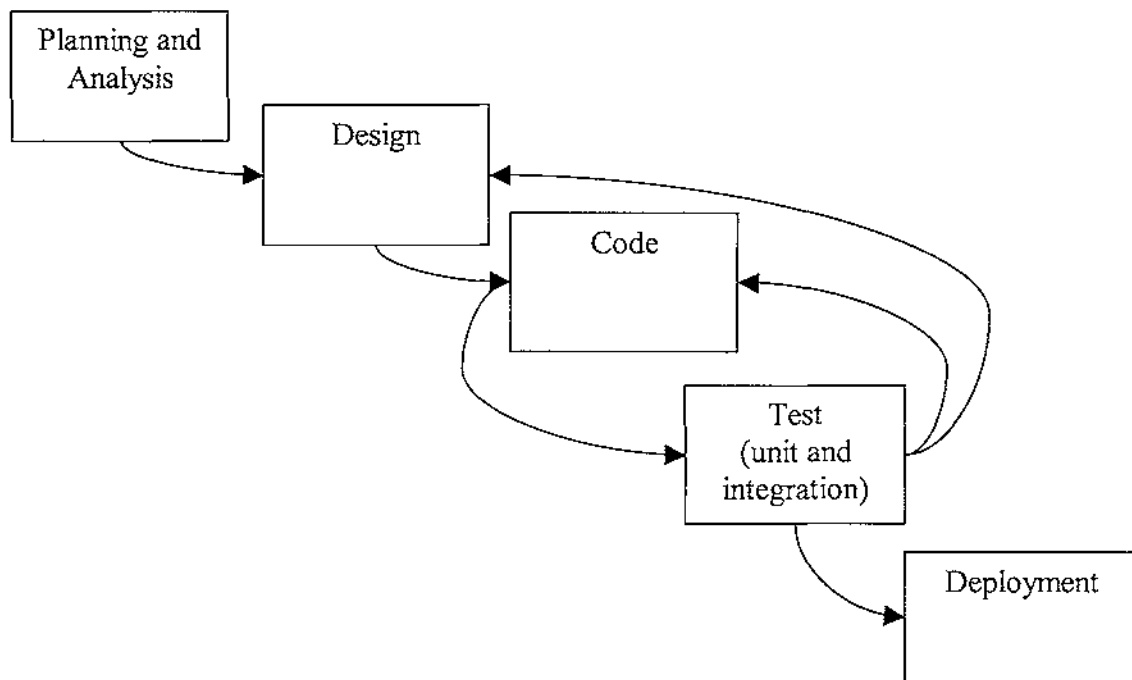


Figure 3.1 Software Life Cycle Model.

Requirements Gathering techniques

- Primarily through interviews with the main stakeholder. Being the original program code designer, the main user and the project supervisor Dr Gunetileke will be able to give invaluable input into program requirements.
- Self assessment. When the prototype of the program has been completed, I (program designer) will actually need to use the program to complete a number of neural network experiments. During this stage (as a user) I will be able to develop ideas for improvements to be made to the program.

Quality

I will endeavour to implement quality assurance procedures throughout the project, such as reviews and inspections. This will be done through formal meetings to discuss requirements, designs and code to see if they are satisfactory. We will also do thorough testing by systematically executing the software to see if it behaves as expected.

Change Management

There is no doubt that the client may want to change some of the specifications part way through the project. One way in which we can be prepared for this change is to maximise communication with the client and make sure the requirements analysis is run through the entire course of the project. With a constraint on time and not being able to extend the deadline there will be a point where we cannot alter the system, late completion is not an option.

Sub-systems: High - Level View

Input: Sub-system receives input from the user via either the keyboard or mouse, or via data already saved to disk.

Output: Output is created based on input from the calculations system and sent to the computers monitor or disk.

Interface, Input Receiver / Output Sender: This code takes in and organizes information input by the user, either through direct input or from information saved on disk, and sends it to the information processing section (code already written). After processing is complete, the interface receives the output and can display it to the screen, save it on disk or send it to a printer.

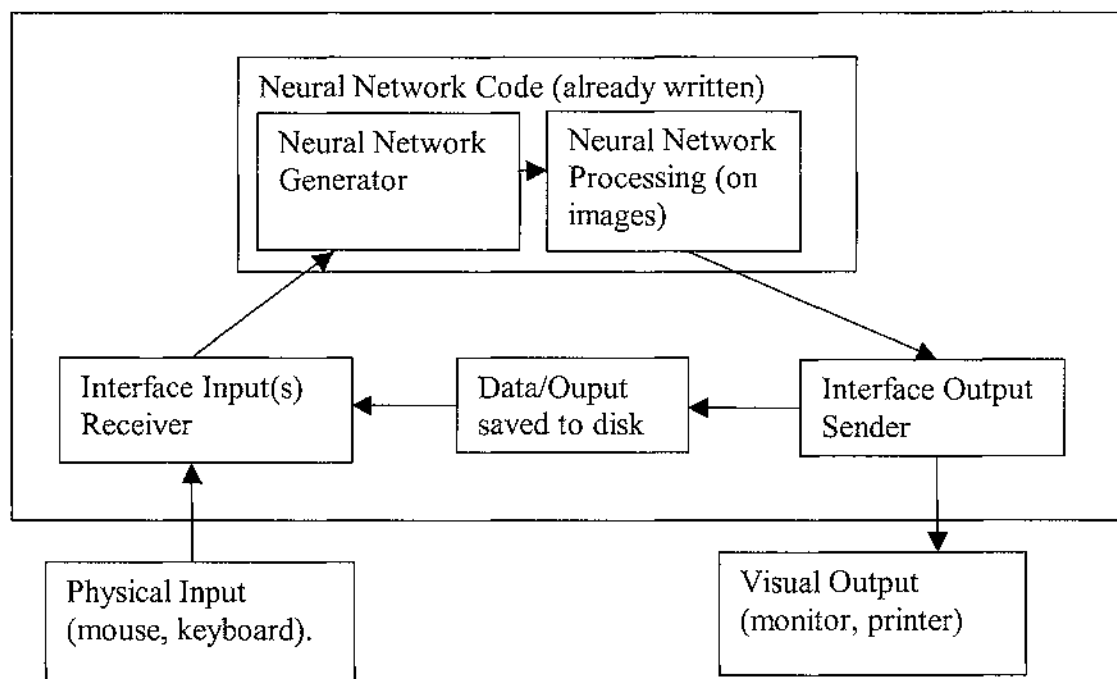


Figure 3.2 Sub-System Diagram: High Level View.

Planning and Analysis: Analysis

Domain Knowledge

The process of fuzzy neural network training involves the network learning the relationships between inputs and selected desired outputs. This is normally begun by initialising the weights of the network with random numbers. However, there are a number of alternative methods for network initialisation. Refer to chapter two for more details.

The research work of Dr Gunetileke focused on the neural network initialisation prior to training. He showed that knowledge of a problem may be represented using boolean logic and conditional rules, which could then be mapped to the initial weights of a neural network affecting its training. His research resulted in a program that's method to update the weights of a FuNN involved the input and output variables not changing during training.

Users

Dr Gunetileke and I will be the main users. Other users may include other academic staff with some knowledge/interest in neural networks and possibly even students.

The Environment

The program will almost be exclusively used on Massey University computers, running on the MATLAB platform. The computers are almost certain to have processor speeds over 1GHz, RAM over 500Mb with large capacity hard drives. The versions of MATLAB used will be 6.4 or later.

Planning and Analysis: Requirements

Specific Client Requirements: Non-Functional

Efficiency Requirements: all interactions will ideally respond within a couple of seconds.

Space Requirements: no specific limitations, but would need to fit on a 650Mb compact disk.

Portability Requirements: must run on an IBM compatible system under the MATLAB platform version 6.4 or later.

Design

High Level Data flow diagrams

The MATLAB programming environment runs its own unique programming language, which is not specifically the same as any object orientated language, or standard language such as 'C'. No examples in the literature of descriptive diagrams for the MATLAB language could be found. Thus, the choice of what style of diagrams to use to express the program visually was a difficult one. The choice to use data-flow diagrams was made because, given the uncertainty, they are simple, easy to draw and understand and fairly flexible.

The diagram below in figure 3.3 represents the highest level data flows, showing information passing between graphical user interfaces (GUI's). It shows that there were to be five separate GUI's, each handling a different area of the overall program, plus a help display. The unique handles graphicsTM employed by MATLAB, in which data is stored as an object property (akin to a global variable), has been represented as a data store. One level down from the diagram shown in figure 3.3 is the high level diagram for each GUI, showing data flows within the GUI. Diagrams for each GUI were drawn during the design process, such as the one shown in figure 3.4.

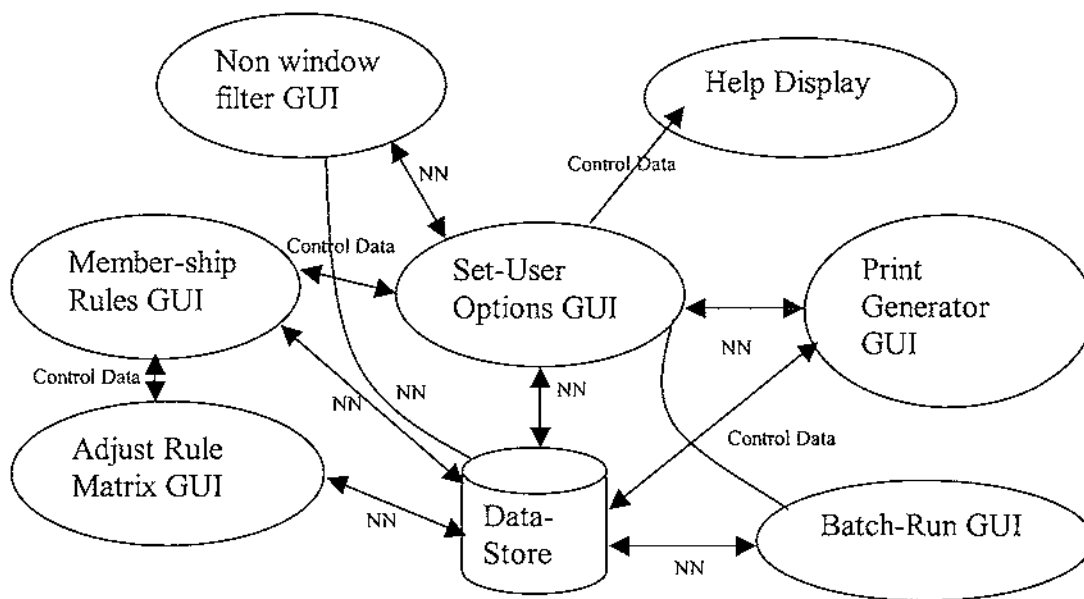


Figure 3.3 Data Flow Diagram: Highest Level

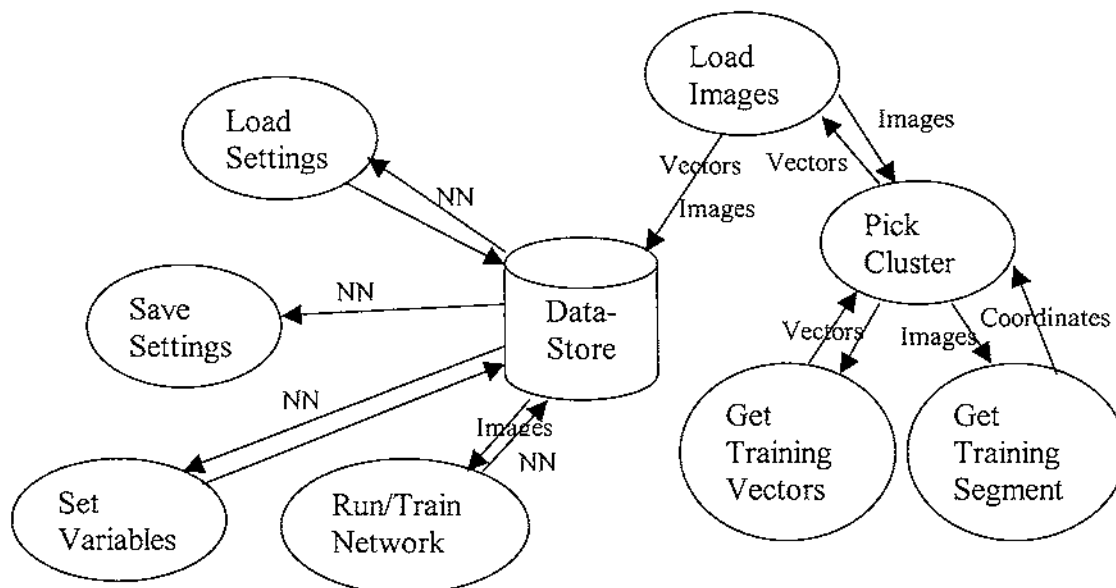


Figure 3.4 Data Flow Diagram: Set User Options.

Major Design Issues

- Which programming language/environment to use?

Given that the original source code was written in MATLAB, the choice was to either continue in MATLAB or use an external interface written in another language (e.g. C++) and import it into MATLAB. Given the high level of connectivity required between the interface and the original source code it was viewed as more desirable to have the interface written in the same language as the original source code.

- Use multiple GUI's or a single GUI?

Although using a single GUI to represent the program would be convenient because it would not require the passing of information between GUI's, it would prove troublesome because all the GUI objects would become cluttered on one screen.

- Stick with an external text file for storing Network variables?

The original code used an external text file to store network variables. This file was read into MATLAB using a parser function which read the file word by word and mapped it into appropriate MATLAB variables. However, it was deemed unnecessary to have this external file (which had to be altered by hand), because it was possible to store the variables within MATLAB itself. However, the conditional rules and Boolean logic rules are best written in a text-format, so some kind of parser function would need to operate to interpret this into a format ready to input into a neural network.

- How much emphasis should be placed on presentation versus functionality of the interface?

Given that the interface is likely to be used by only a relatively small number of academics and exclusively for research purposes, it was viewed that the presentation (placement, colour, style of buttons) would be significantly less important than usability and functionality issues.

Given the very limited range of people who will use the interface, interface design will play a secondary role to functionality. However, basic interface design techniques were considered. It is clear the main aim when designing the interface is to achieve maximum usability of the program. Effective strategies for building interactive

systems were considered 'within the context' of the interface tools provided by MATLAB.

Design Considerations: Human

Visual perception:

Colours can be used to indicate different types of information and levels of headings. It can also be used to help familiarise users with the design of the system (Hemi, 2002). Obviously the design needs to avoid colour combinations that are difficult on the eye. Although colour is 'free', it should not be over-used because it could become distracting. Font size must be large to be clear, to not so big as to clutter-up the screen. Font size and boldness can be adjusted for emphasis.

Limitations of visual processing:

General opinion suggests using avoiding the use of serif fonts, with curves, wiggles and decorative typefaces. Rauch (1996) suggests avoid using italics, because they are difficult to read.

Human memory:

Short term memory has limited capacity, with the average person remembering only 7 ± 2 digits (Dix *et al.*, 1998). Thus, efforts should be made to minimise information recall (for example, when changing GUI's), and provide retrieval cues wherever possible.

Design Considerations: Computer

Input devices:

The three available input devices used on the institute of information sciences and technology (IIST) computers are QWERTY keyboard, optical mouse and tablet. However as the tablets are not available on all the machines, it will not be used as part of the interface.

Memory and Speed.

The application will run on university computers, which usually have 256MB RAM or more with large hard-drives. They are also relatively fast with 1GHZ processors or

faster. For the interface and graphics operations that will be performed, the computers will be more than capable of handling them.

Design Considerations: Interaction (between user and machine)

For the arrangement of control and displays there are three main alternatives: 1. Functional grouping – functionally related controls are placed together. 2. Sequential grouping – controls are organised to reflect the order of their use during a typical interaction. 3. Frequency grouping – controls are organised according to how frequently they are used, with most common controls being the most accessible (Dix *et al.*, 1998). For this type of application, sequential and frequency grouping might depend on what type of problem the network is trying to solve, and so functional grouping will be used.

The interaction styles are limited by the nature of MATLAB itself, thus we are reduced to just two choices for interface style. Firstly, there is the command line interface, which is the original style for operating MATLAB. Secondly, there is a type of graphical interface using MATLAB'S 'GUIDE', which supplies objects such as pull-down menu's, push-buttons, editable and non-editable text objects, scroll-bars, graph-plots and radio buttons. This is effectively a WIMP interface as the interface is opened up in a scalable window, it uses icons such as 3D push-buttons which are all pointed at and manipulated via the 'mouse'.

For screen design and layout, active changeable elements will be separated from inactive (just displaying information) elements through the use of colour. The background of inactive elements will be the same as that of the GUI itself. Screen aesthetics will not be a high priority.

MATLAB user interface controls only allow very limited use of iconic metaphors. Also given the sophisticated level of the envisaged users, the use of metaphors is not regarded as a priority.

To assist with 'learn-ability', efforts are made to make the system predictable, so the user can work out what will happen based on past events. Also, to ensure consistency there will be likeness in input-output behaviour arising from similar situations.

Requirements Elicitation

Specific Client Requirements: Functional

During an interview with the client (also the main stakeholder) on the 20th of November 2002, a request was made to produce an interface program for a Fuzzy Neural Network application. The following specific requirements were requested:

- The start screen should be blank except for perhaps a title, with all options being presented via pull-down menus.
- Users should be able to load in saved neural network settings via dialog boxes.
- Users should be able to change the variables that are used to train the network, such as:
 - o filter window size (e.g. 3 by 3, 3 by 11)
 - o number of input neurons
 - o number of output neurons
 - o number of epochs
 - o learning rate
 - o total number of rule nodes
 - o learning rate
 - o momentum
 - o saturation
 - o type of network (FNN or MLP)
- Users should be able open input and target images with the help of dialog boxes.
- The program should check that the input and target images are of the same dimensions
- User should be able to select a cluster area using drag-select (with an elastic band effect).

- The program should display the graph of the 'fuzzy membership functions' (SNN.FuzzyMFDEF). The user should be able to click on the graph to change the values, or be able to change them manually by entering in a value via the keyboard.
- User should be able to train the network from within the application.

Further User Requirements: Functional (part 1)

Upon completion of all of the requirements requested in the interview on the 20th of November, the client then requested the following additions/alterations on the 14th of January 2003.

- The 'window filter size' variable should be automatically updated upon a change in the horizontal or vertical filter window size.
- Users should have an option to select multiple cluster areas.
- The horizontal axes of the Fuzzy membership graph must be from '0 – 1', not '1 – X' (where X is the number of fuzzy membership rules). The vertical axes will also be from '0 – 1' meaning fuzzy membership values are shown in the horizontal axes not the vertical one.
- The Fuzzy membership graph will show the 'triangular membership function'.
- The interface will show a sign on the start screen to indicate when images have been loaded.
- The 'Update graph' button should be more obvious by highlighting it in some way (or doing away with it altogether).
- Introduce a feature so the user can right click a box in the 'text-rule grid' and choose the input from a pop-up menu.
- All the boxes in the graphs representing the conditional rules should be numbered (from left to right, top to bottom).
- Users should be able to 'drag-select' a group of boxes in a graph representing a conditional rule and change them all at the same time.
- A small greyscale representation of the current conditional rule should be added.
- Introduce a batch-processing GUI, to handle consecutive processing of a large number of experiments. This would include an ability to change variables such as the number of epochs and the number of rule nodes.

- Users should be able to save the output performance data and the output image into a variable for storage for batch runs.

Further User Requirements: Functional (part 2)

Upon completion of all of the requirements requested in the interview on January 14th, the client then requested the following additions/alterations on the 16th of January 2003.

- Add rule greyscale images to the 'print-report'.
- Add ability to save extra-notes into the neural network.
- Users should be able to adjust the 'multiplication factor' for the random number generator in the batch process GUI.
- Users should be able to run the network without rules in the batch process GUI.
- Make the greyscale images representing the conditional rules greyscale if there is a rule and some other bright colour if there is no rule.

More idea's from the design team

During the course of design and testing of the program, additional ideas for features were added.

- Option to save (or not) trained network. This is due to large file size of the networks.
- Option to print images with selected cluster area's highlighted.
- Option to add user notes to 'print-out' GUI.

Coding

The coding was done entirely in the MATLAB environment. Useful resources that assisted in coding were Graphics and GUIs with MATLAB (Marchand, 1999), Building a Graphical User Interface (Mathworks Inc., 1993), and the Mathworks technical support site at www.mathworks.com/support/. The program code itself can be found in appendix 3.

Chapter 4: Interface Results and Testing

Introduction

This chapter examines the results of the development process (i.e. the interface itself), and looks at testing, including interface evaluation and unit and integration testing. As there were a very large number of tests, only a sample of what was done is included here.

Graphical User Interface Results

The introduction screen contains a simple blank screen and the title, as requested by the client. The user starts off by selecting from the pull down menus shown in figure 4.1.

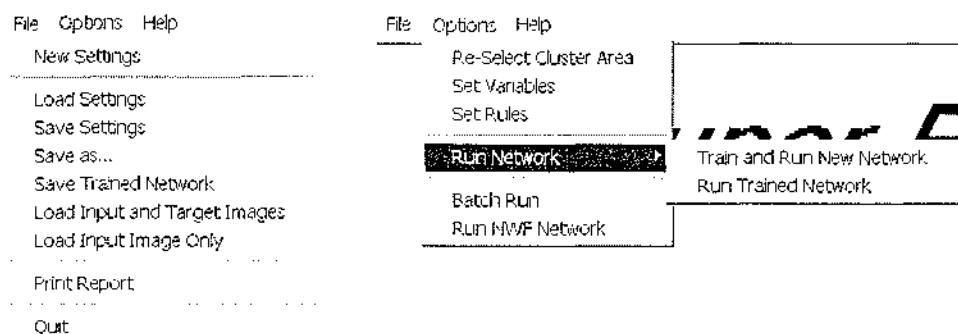


Figure 4.1 Pull down menus from 'Set variables' GUI

Almost all of the menu items contain error checking, and many are not enabled when it is not prudent for them to be activated. For example, 'Save Settings' in the file menu is disabled before 'New Settings' or 'Load Settings' have been activated.

Adjusting Input Variables

After the network settings have been loaded or new settings have been selected, the user can then adjust those variables via the Set-User-Options GUI (shown in figure 4.2). When the variable 'Is Window Filter' is set to '0', the window size variables

disappear (as they are then not relevant). The input variables are all closely grouped together, for easy comparison, and the input and target images are also made visible for the user's convenience. All variables are editable via text boxes or pull-down menus.

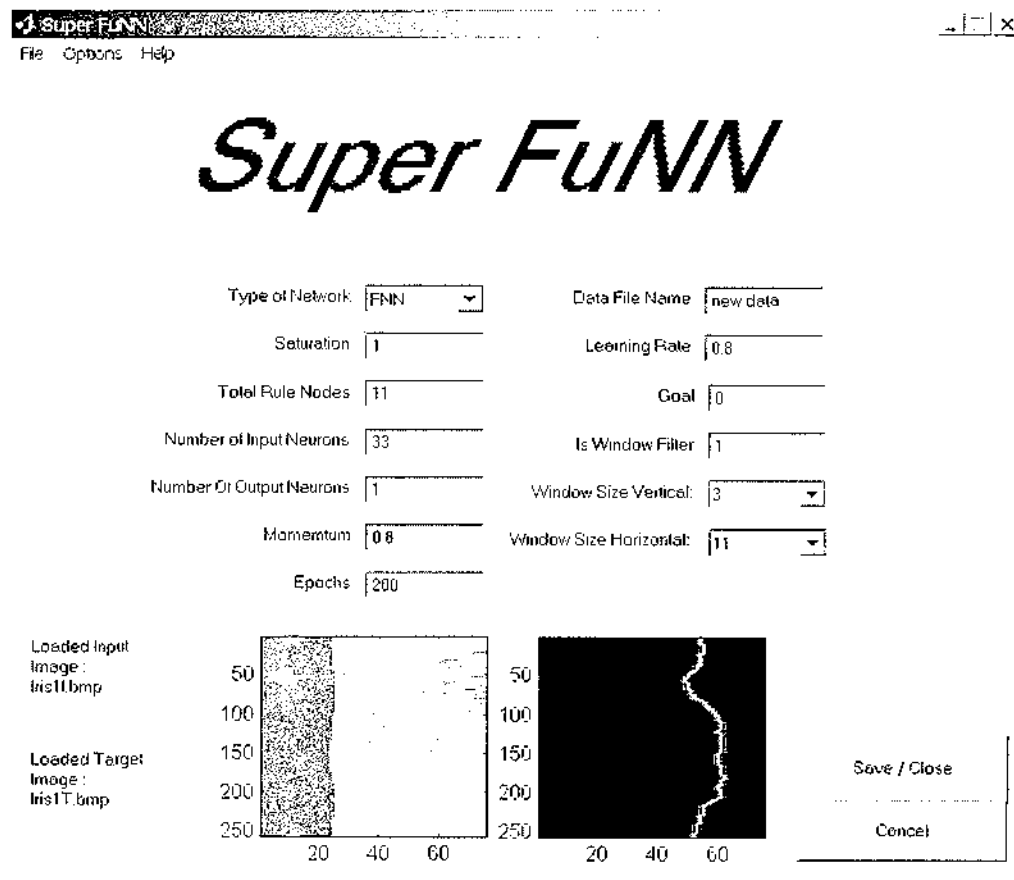


Figure 4.2 Set User Options GUI, allowing easy adjustment of input variables

Changing Fuzzy Membership Values

By choosing 'Options→Set Rules', the user will start the 'Change fuzzy membership values' GUI shown in figure 4.3. The user has the option of changing the fuzzy membership values by either using the edit boxes (which automatically updates the graph) or by simply clicking on the graph itself (which automatically updates the edit boxes). No graph is drawn for the defuzzy membership values as these are usually only two or three values. If the user clicks the 'view text rule' button the 'Adjust text rules' GUI is started. Pressing the 'Save/Close' button will return control to the 'Set User Options' GUI.

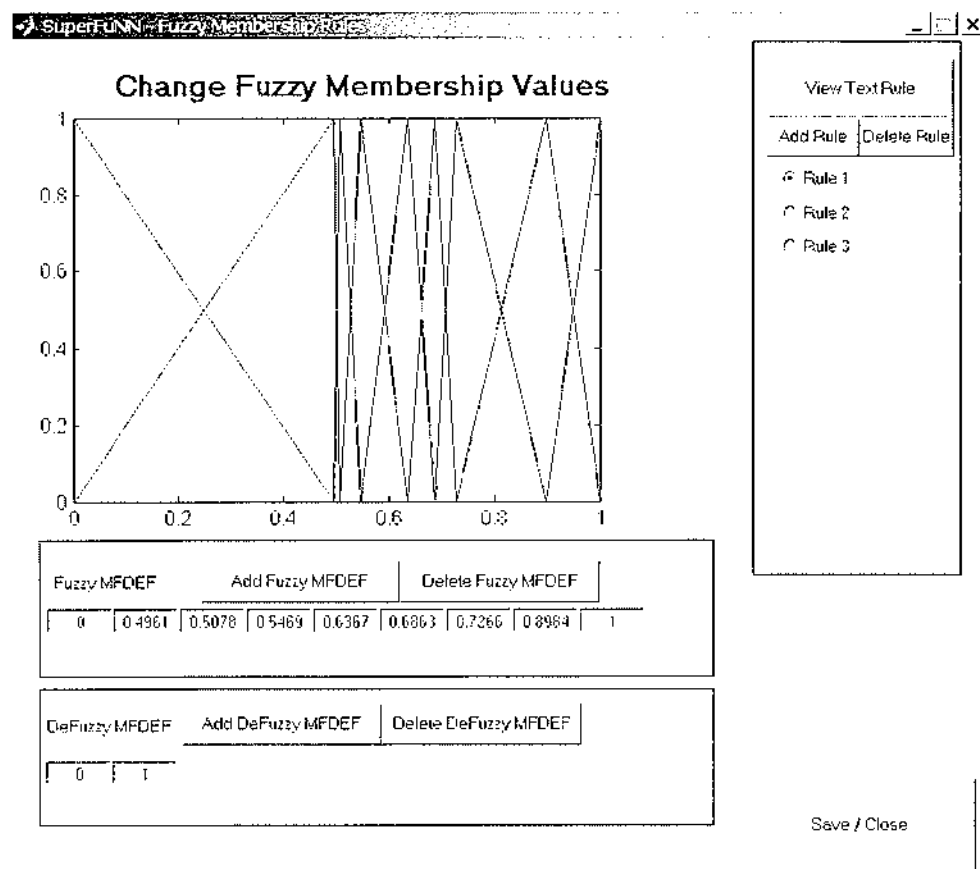


Figure 4.3 GUI for changing fuzzy membership values

Adjust Rules (representing the problem knowledge)

The 'Adjust text rules' GUI represents each rule in three forms: graph, grid and text, for easy recognition by the user. The user can use the edit box to texturally update the rule, which then automatically updates the grid and graphical versions. Alternatively the user can click or drag select on the grid to update the rule, which then automatically updates the text and the graph. This feature allows quick and easy creation or updating of text rules, without having to type anything.

The fuzzy membership values from the previous screen have also been included here to reduce the amount of memory recall needed by the user. Other variables such as epochs and saturation level were not regarded as important at 'this' stage and thus are not shown to avoid cluttering the interface. When the user has finished updating the rules, pressing the close button will return back to the 'Change fuzzy membership values' GUI.

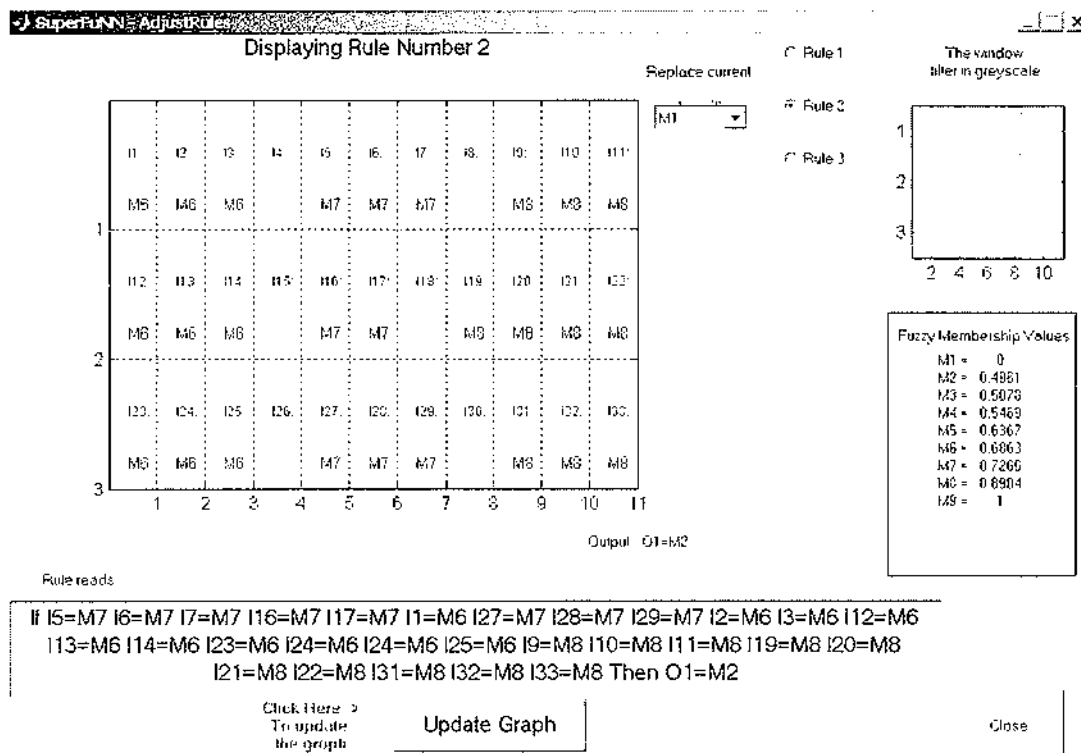


Figure 4.4 Adjust Test Rules GUI

Loading Images and Selecting Clusters

If the user selects 'File→Load Input and Target Images', they firstly be will be prompted by dialog boxes to choose two images, then asked to select a number of cluster areas. The user has the option of selecting one cluster area, or choosing to select multiple cluster areas. A screen shot of the interface after the user has selected some cluster areas is shown in figure 4.5 below. The cluster areas are selected using a rubber-band drag and select approach, and the chosen areas are highlighted by blue rectangles. Cluster areas can be selected using either the input or the target image.

Training and Running the Network

After the user has loaded input and target images and network settings (assuming it is a window-filter network), then they can choose 'Options→Run Network→Train and Run Network' or 'Options→Batch Run'. If they choose the former, then the MATLAB will create, train and then simulate the neural network. An example of this is shown in figure 4.6. The RMS error, the number of false positives and negatives and the percentage error are calculated and displayed automatically. The user can then choose to use a post processing algorithm, although this algorithm is only designed for the iris edge detection problem.

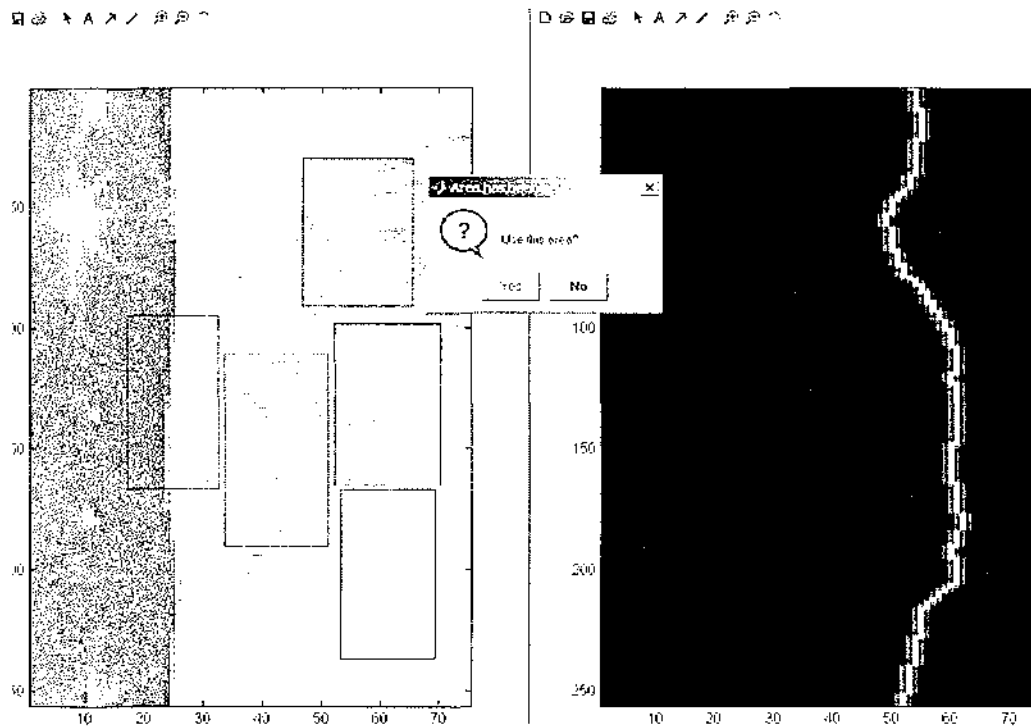


Figure 4.5 User has selected cluster areas which are highlighted by bounding rectangles

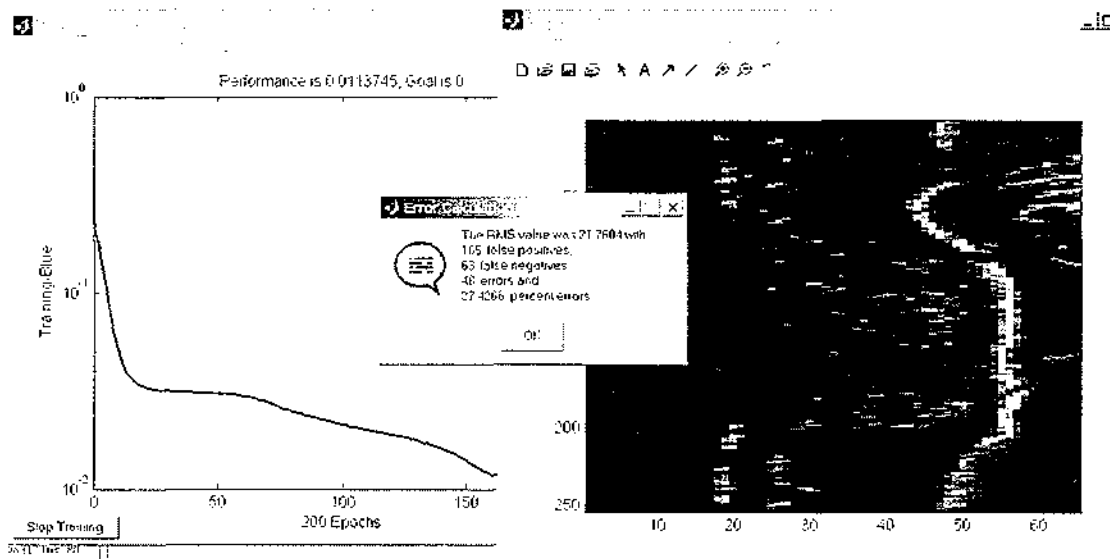


Figure 4.6 An example of the interface just after a network has been simulated.

Batch Runs

If the user selects 'Options→Batch Run' after the images and network settings have been loaded, the 'Batch Run' GUI is started (shown in figure 4.7). The network settings such as epochs, saturation level and fuzzy membership values are not shown here as it is viewed that this information is not needed by the user at this stage. The

user would generally be expected to run the network once (from the 'Set User Options' GUI) to test it, before using the Batch Process option.

From the 'Batch Run' GUI the user can choose to alter epochs, rule nodes, random input, have no rules or a combination of these. The dialog box for the option of varying epochs and rule nodes is shown in figure 4.8. The user is also given the option of repeating the experiment a number of times for each combination of input variables in order to improve the reliability of the results. When the experiments have finished the results are displayed on the screen and automatically saved to disk under a user-specified name. The user has the option of viewing the output graphs and of saving the neural network data (i.e. rule performance data, output images and network settings).

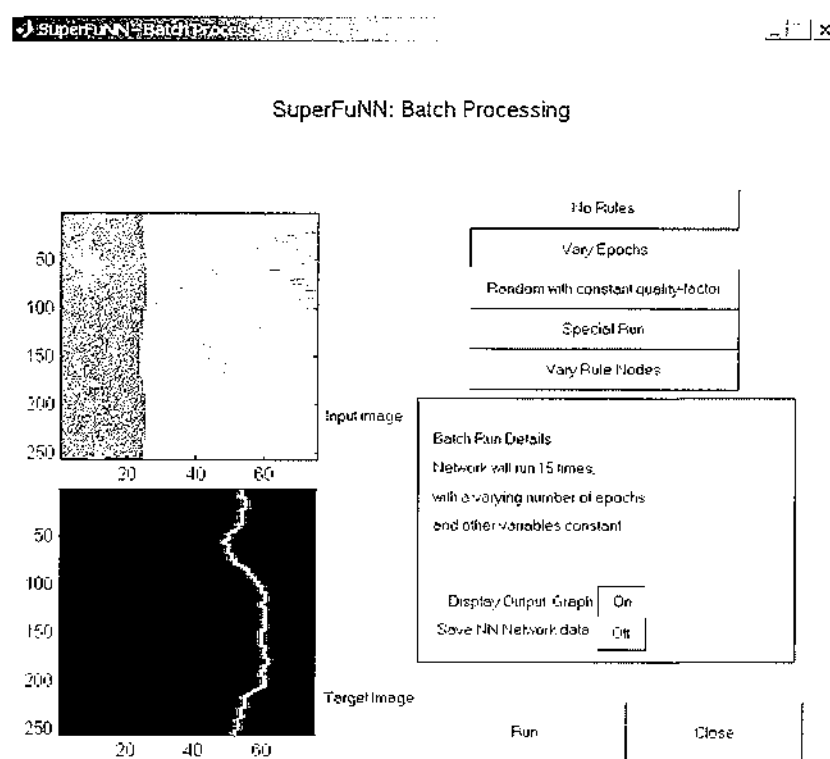


Figure 4.7 The 'Batch Run' GUI

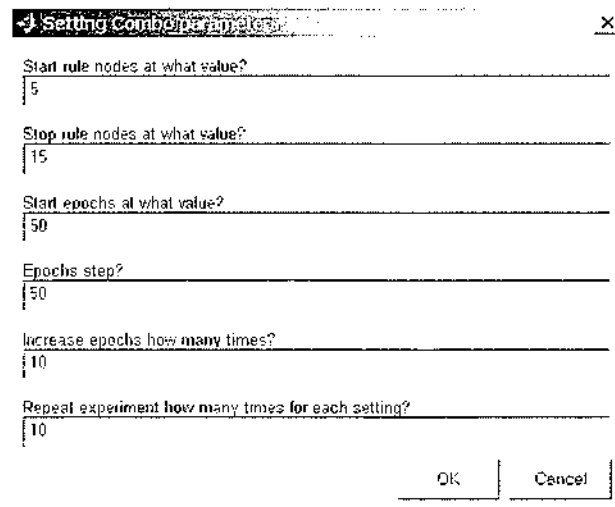


Figure 4.8 Dialog box for setting batch run parameters

Testing

Due to the nature of the project, the program was not thoroughly 'independently' tested, but was developer tested and user tested.

Function Testing

Function testing was done using two main approaches:

Unit Testing: Every function that was written was unit tested. Input stubs were provided, run through the function and the output analysed. Test data for unit testing was firstly selected to test the regular expected case. Then data was selected for equivalence and boundary testing purposes.

Integration Testing: After each function had passed the unit test, it was integrated into the rest of the application and tested again.

Test Data: Due to the time constraints, test data was prioritised, so not all possible combinations were necessarily entered. However, data to ensure the system worked under reasonably normal situations was used.

User Interface: Acceptance Testing

This was done in an informally simply by asking a principle user to test the program and provide verbal feedback on the strengths and weaknesses. Also, as the program designer was also one of the principal users, he was able to evaluate the design during practical testing. Things that were considered when evaluating the design:

- Language Used: is it relevant for the users (not too technical or overly simple)
- Minimalist design: no irrelevant or rarely used information
- User memory load minimised (when changing from GUI to GUI)
- Consistency throughout the application (includes consistent terminology)
- Feedback: to let users know what effect they are having on the system
- Good error messages: to let the user know what they did wrong and how to correct it
- Provide clear exits: so user can easily close a GUI when they are finished using it

User Interface: Checklist

A checklist was made from all of the requests garnered from the requirement elicitation (see Chapter 3). Features of the interface were compared with the checklist, and all of the items in the checklist except one were deemed to be represented in the interface. The following request was not able to be implemented:

“Introduce a feature so the user can right click a box in the ‘text-rule grid’ and choose the input from a pop-up menu.”

This was not implemented because the MATLAB interface system did not allow this action to be performed. However, a separate pop-up menu placed beside the ‘text-rule grid’ was provided. This was similar (although not the same) as what was requested.

Example Test Case

The following is one example of unit testing and one of integration testing for the ‘Find SGLDM data’ function. Testing the outputs using regular sized image inputs would be practically unrealistic, thus a small contrived data set was fed in and the

result compared to hand calculated values. Equivalence and boundary testing was not deemed feasible for this particular function.

Case Number: 45P.1

Function Name: Find_SGLDM_Data

System: Non-window filter network GUI

Test Data: A contrived test image consisting of 4 grey levels:

```

1 3 3 2 1
0 1 1 2 0
1 3 1 3 1
0 2 2 2 2
0 3 2 1 0

```

Expected Results: The SGLDM matrix for $d = \sqrt{2}$ and $\mathcal{G} = 315$ is expected to be:

```

0 0 0 2
0 2 2 1
2 2 1 0
0 1 3 0

```

The normalising factor $R = 2 + 2 + 2 + 1 + 2 + 2 + 1 + 1 + 3$
 $= 16$

$$\begin{aligned} \text{Energy} &= (4 + 4 + 4 + 1 + 4 + 4 + 1 + 1 + 9) / (16)^2 \\ &= 0.1250 \end{aligned}$$

$$\begin{aligned} \text{Entropy} &= -\{[(2/16) \times \log_2(2/16) \times 5] + [(1/16) \times \log_2(1/16) \times 3] + \\ &\quad [(3/16) \times \log_2(3/16) \times 1]\} \\ &= 3.0778 \end{aligned}$$

$$\begin{aligned} \text{Homogeneity} &= (2/10 + 2/1 + 2/2 + 1/5 + 2/5 + 2/2 + 1/1 + 1/5 + 3/2) / 16 \\ &= 0.4689 \end{aligned}$$

$$\begin{aligned} \text{Inertia} &= (9 \times 2 + 0 \times 2 + 1 \times 2 + 4 \times 1 + 4 \times 2 + 1 \times 2 + 0 \times 1 + 4 \times 1 + 1 \times 3) / 16 \\ &= 2.5625 \end{aligned}$$

$$\begin{aligned}
\text{Correlation} &= [(0 - 1.6875) \times (3 - 1.625) \times 2 + (1 - 1.6875)(2 - 1.625) \times 2 + \\
&(1 - 1.6875)(3 - 1.625) \times 1 + (2 - 1.6875)(0 - 1.625) \times 2 + (2 - 1.6875)(1 - 1.625) \times 2 \\
&+ \\
&(2 - 1.6875)(2 - 1.625) \times 1 + (3 - 1.6875)(1 - 1.625) \times 1 + (3 - 1.6875)(2 - 1.625) \times 3] \\
&/ (16 \times 0.9648 \times 0.8594) \\
&= -0.4428
\end{aligned}$$

Actual Result: All output variables from the function were the same as the hand calculated values. Thus this test is regarded as successful.

Case Number: 45P.2 (Integration test)

Function Name: Find_SGLDM_Data

System: Non-window filter network GUI

Instructions: Load a non window filter network into SuperFuNN, select 'Run NWF Network' and then choose 'Classify a Single Pollen'. Load same test image from test case 45P.1.

Expected Result: The 'Iv' parameter in the 'ClassifyPollen' function will be an array with the values = [area, circularity, 0.1250, 3.0778, 0.4689, 2.5625, -0.4428].

Actual Result: The Iv variable contained the values in the array as expected, thus the test was successful.

Conclusions

Unit testing proved very useful during coding, as many of the functions did not perform entirely as expected the first time. Once a function passed the unit testing, it was usually successful in passing the integration test as well, though not always. Interface testing was done through a combination of logic testing (does pressing this button actually produce the expected result?), informal user acceptance testing (done with the user) and checking off features with the list of requests from the requirements elicitation.

Chapter 5: Iris Edge Detection

Introduction

Verifications systems based on iris patterns, utilize the fact that everyone has their own unique iris features. The iris itself consists of pectinate ligaments adhering to a tangled mesh revealing striations, ciliary processes, crypts, rings, a corona and sometimes freckles and other features (Daugman, 1999). Part of the problem of using iris patterns is isolating the iris from the other parts of the eye. Detecting the pupil-iris edge can be done using standard edge detection algorithms, but detecting the edge of the iris between the iris itself and the white part of the eye proves a more difficult challenge.

Research by Gunetileke (2001) has shown that a FuNN can be used to detect an iris edge with 'reasonable' results. However, his research does not consider variations on all the possible input variables for training the FuNN or which combination can produce the best results for a variety of input images. The objectives of this experiment are to show that different input variables into the network have a significant effect on the output, and to find the best combination of input variables for the iris edge detection problem.

The input variables 'epochs', 'rule nodes', 'saturation level', 'window-filter size', 'fuzzy membership values' and 'quality factor' were considered for analysis. In his research, Gunetileke did considerable experimentation and analysis with fuzzy membership values, window filter size and quality factor. Also, he produced logical reasons for the settings of the saturation level. However, variations in the number of epochs for which the network was trained, and the number of rule nodes used were not tested to any significant extent for the iris edge problem.

Preparing the image

There are a number of steps involved in preparing an image of an eye (such as the one shown in figure 5.1) for processing by a neural network. An important step is determining the centre of the coordinates of the eye. The algorithm is:

- Expand the lower half of the grey-scale of the eye image to remove darker features such as eyelashes.
- Threshold the image to allow for the detection of the object with the largest area. This assumes that the pupil is the largest dark connected object in the image.

The centre pupil is defined as the centre of its bounding rectangle, as opposed to its chain coded centre of gravity (Bishop *et al.*, 2000, cited in Gunetileke, 2001).

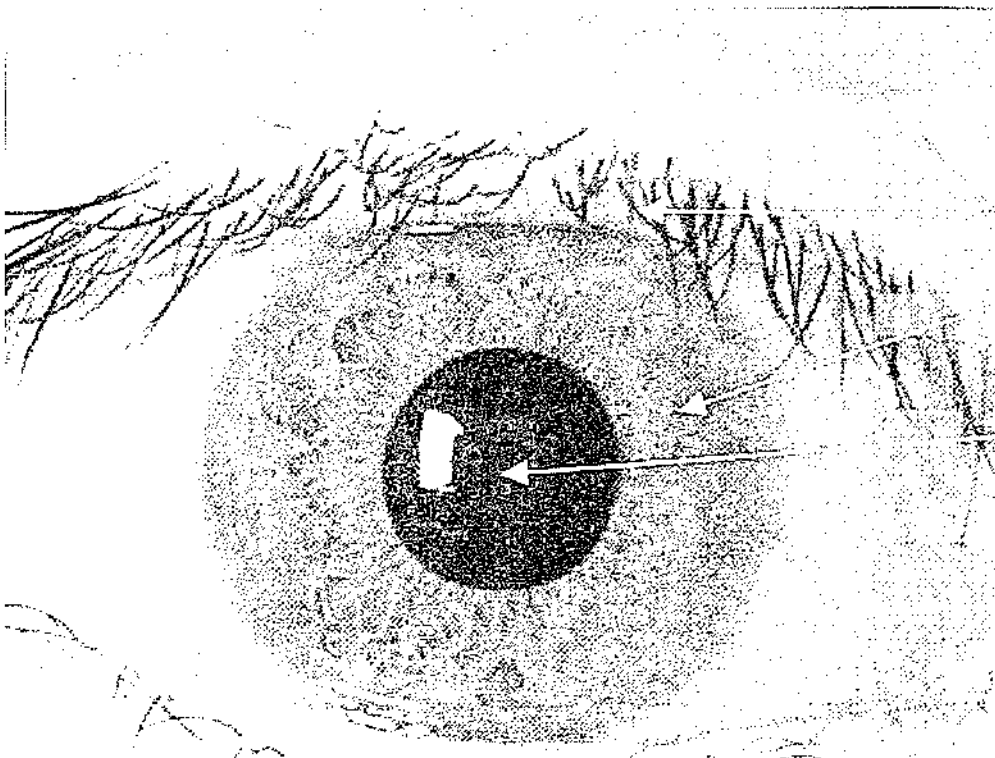


Figure 5.1 Image of an eye used for iris edge detection problem

The next stage is to use a polar transform on the image using the centre coordinates. This effectively transforms the eye from its round shape to a rectangular one, making it easier to use boundary detection methods. Figure 5.2(a) shows an image of an eye ready to input for the experiment, after it has been through the polar transform. Figure 5.2(b) highlights the pupil and iris edges of figure 5.2(a). Standard edge detection filters tend to detect the pupil edge but not the iris edge as shown in figure 5.2(c).

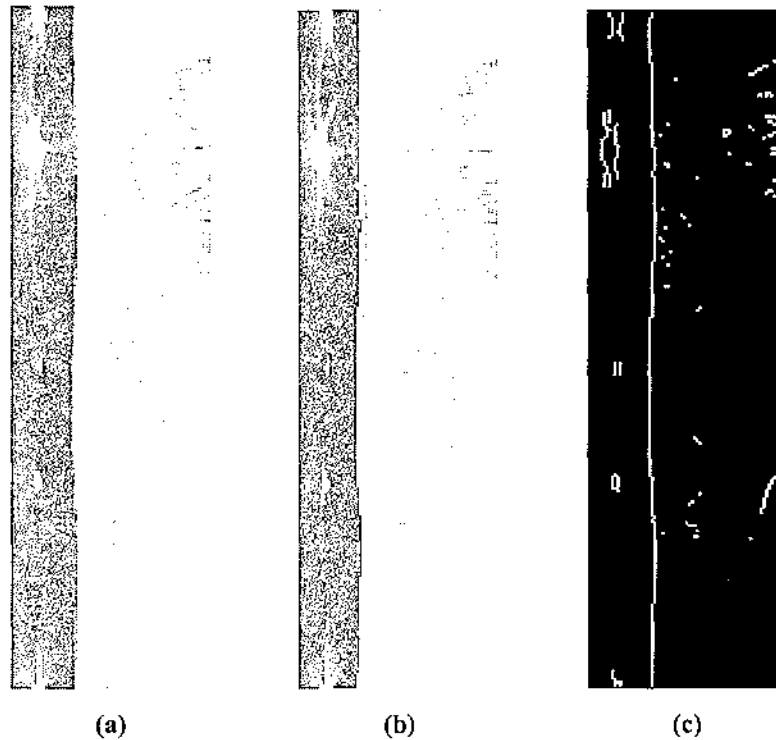


Figure 5.2 Image (a) is the input iris image. Image (b) highlights both the pupil and the iris edges. Image (c) shows the effect of a standard edge detection process on image (a).

Training Images

The input and target images used to train the network are shown in figure 5.3. They are the same as those used by Gunetileke (2001) during his experiments. The iris edge in the target image was drawn by hand. Firstly, an iris edge with a width of one pixel was drawn, and then the image was blurred with a Gaussian filter. Blurring was done to create a range of possible edge positions, negating the fact that there may have been inherent displacement errors in the original line.

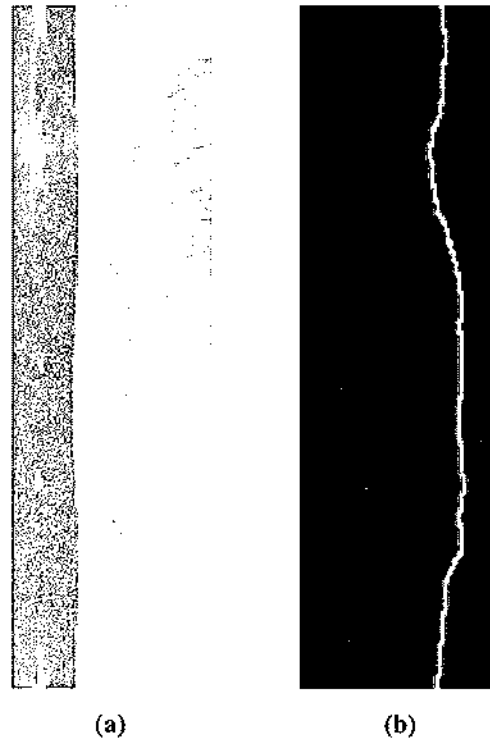


Figure 5.3 Image (a) is the input image. Image (b) is the corresponding output image.

Measuring the results

The measure of success of the network will be done using a ‘figure of merit’ approach, as was used by Gunetileke (2001). Figure 5.4 shows the figure of merit concept. The grey area of 3 represents the range of possible solutions where the iris edge can lie, as the true position of the edge is unknown. The figure is used as follows:

- If a pixel identified as an edge point by the network lies in area 1 or 2 (i.e. it shouldn’t be there), then a false positive is generated.
- If a pixel in area 3 is not identified as an edge pixel by the network, then a false negative is generated.

The error is calculated by counting the number of false positives (FP), false negatives (FN) and using the following equation:

$$\text{Error} = 0.1 \times \text{FP} + 0.5 \times \text{FN}$$

Lower weighting is given to false positives because they can be removed by post processing procedures. As the input images for this experiment have heights of 250 pixels, it is assumed that the output image can have a maximum of 500 false positives. Thus, the percentage error is calculated as:

$$\text{Percentage Error} = \text{Error} \times 100/175$$

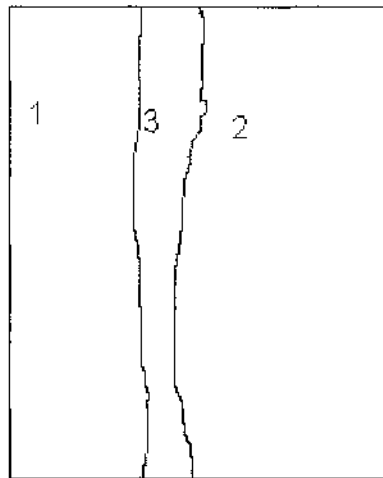


Figure 5.4 Image to demonstrate error calculation

The Experiments

Experiment 5.1

Experiment 5.1 was to determine the best combination of rule nodes and epochs to train the FuNN. The variables used to create the network for this experiment are shown below in table 5.1

The window filter rules are represented diagrammatically below in figure 5.5. The colours represent the fuzzy values in the rule, where 0 is black, 255 is white, and no rule is represented in this text by 'X' (on the computer monitor 'no rule' is presented as the colour turquoise). The input vectors were selected as shown in figure 5.6 below.

Input Layer	33 nodes (3 x 11 window)
Condition Layer	9 fuzzy membership values {0.00 0.48 0.52 0.58 0.64 0.70 0.76 0.89 1.00}, thus 297 nodes
Rule Layer	5 \rightarrow 15 nodes (3 rules + [2 \rightarrow 12] free nodes)
Action Layer	2 nodes
Output Layer	1 node

Table 5.1 Network settings for Experiment 5.1

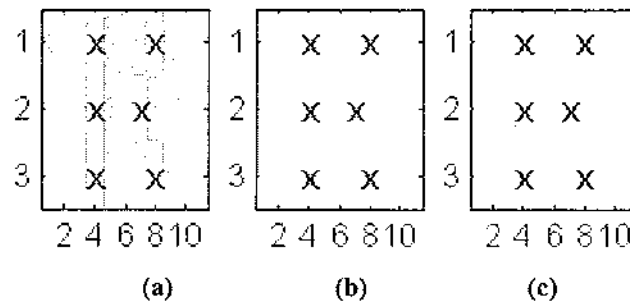


Figure 5.5 Images (a), (b) and (c) show a pictorial representation of the conditional rules 1,2 and 3 respectively, for experiment 5.1



Figure 5.6 The input image showing cluster area's selected for experiment 5.1 (enlarged horizontally to make cluster areas more visible).

In this experiment the epochs were increased in increments of 50, ranging from 50 to 1000, and the number of rule nodes was increased by 1 from 5 to 15. For each combination of rule node and epochs the experiment was repeated 10 times. Thus in total, the network was trained 2200 times. The percentage error as described above was determined and the results are shown in table 5.1 below.

Rule Node	50	100	150	200	250	300	350	400	450	500
NR	76.8	62.2	32.9	31.6	29.7	29.6	29.3	27.4	27.6	27.6
5	72.5	32.5	31.6	30.7	29.8	29.8	29.2	29.5	30.1	30.2
6	72.5	35.2	29.7	27.7	27.3	28.9	28.4	28.3	28.8	28.8
7	72.5	35.4	31.3	30.3	30.1	30.3	30.4	30.5	30.1	29.8
8	72.5	33.4	31.5	30.3	29.6	28.7	28.8	28.8	28.2	26.8
9	72.5	35.8	28.7	30.0	27.9	27.7	28.3	27.9	27.3	27.3
10	72.5	37.8	29.8	29.5	28.6	28.0	27.4	28.4	27.9	26.7
11	72.5	41.4	33.6	29.5	28.2	27.6	27.1	26.4	26.0	26.2
12	72.5	45.7	30.2	30.0	29.6	29.3	28.8	29.3	29.0	29.1
13	72.5	44.5	34.0	29.9	30.2	29.5	29.0	28.4	26.8	26.0
14	72.5	48.8	31.6	31.2	31.4	31.8	30.7	30.6	30.2	30.1
15	72.5	49.4	34.5	31.2	31.9	30.0	30.2	29.4	29.0	28.2

Rule Nodes	550	600	650	700	750	800	850	900	950	1000
Epochs										
NR	27.6	27.7	28.0	27.8	28.0	28.5	28.3	28.6	28.0	28.0
5	29.8	30.0	29.8	30.0	30.2	30.0	29.9	30.1	30.2	30.6
6	28.3	29.0	29.6	29.9	29.9	30.5	30.6	30.6	30.3	30.3
7	30.3	30.3	30.5	29.9	29.2	28.8	28.6	28.6	28.5	28.3
8	27.3	27.6	27.4	26.4	26.2	25.7	25.7	25.6	26.3	26.3
9	27.8	27.7	27.7	27.3	26.2	26.0	26.0	26.2	26.0	25.8
10	26.6	26.6	27.1	26.9	26.7	25.5	26.1	26.7	26.8	27.3
11	25.9	26.0	27.0	27.2	27.0	27.3	27.1	27.8	27.6	27.6
12	28.7	29.0	28.7	28.8	29.1	29.5	29.3	29.1	29.3	29.3
13	27.0	27.3	27.0	26.8	26.7	26.6	26.8	26.7	26.5	27.0
14	29.9	28.9	28.7	28.4	28.2	28.0	28.2	28.2	27.5	27.5
15	28.5	28.4	28.6	28.4	28.5	27.6	27.6	27.6	27.5	27.6

Table 5.1 Percentage errors for combinations of epochs and rule nodes (NR = no rules)

While it is mathematically unfeasible to prove the one particular combination is better than all other combinations, the results do show a general trend. The majority of lower errors (25.5 – 27.5%) tend to be found with 6 – 12 rule nodes and 250 or more epochs. Results did not dramatically improve over the 250 epoch mark, and thus

epoch values close to but above 250 would be the best. For much higher epoch values, the network is in danger of becoming ‘over-trained’. This would make it too inflexible to perform well for other images. The results of a typical experiment are shown in figure 5.7. Figure 5.7(a) shows the result straight after processing of input image 1, while 5.7(b) shows the result after 5.6(a) has been through post-processing.

Post Process Procedure

The algorithm for post processing is the same as that used by Gunetileke (2001). The procedure is as follows:

1. Threshold network output using a threshold value of 20.
2. Skeletonize the threshold image (this is a built-in MATLAB command)
3. Generate the negative of the skeletonized image and use the least cost path algorithm to generate the final iris edge.

Least Cost Path

This algorithm (cited in Gunetileke, 2001), was developed by Dr. D.G. Bailey at Massey University and is as follows:

1. The foreground pixels are given a weighting of 0 and the background pixels are given a weighting of 255.
2. A penalty function is added to each pixel in each row with processing done row by row. In this case the penalty function is [9 4 1 0 1 4 9].

$$E[r,c] = E[r,c] + \text{Min}\{ E[r-1,j], E[r-1,j-1]+1, E[r-1,j+1]+1, E[r-1,j-2]+4, E[r-1,j+2]+4, E[r-1,j-3]+9, E[r-1,j+3]+9 \}$$

3. The pixel with the smallest value at the bottom of the image is used as a starting point to traverse upwards, finding the least cost path through the image.



Figure 5.7 Image (a) is the output from the neural network in experiment 5.1. Image (b) is the result after image (a) has been post processed.

Experiment 5.2 – 5.5

The experiment 5.1 was repeated for experiments 5.2, 5.3 and 5.4, using different input images. The results of these experiments mirrored those of experiment 5.1 and thus merely served to reinforce those results.

Experiment 5.6

It was noticed during the previous experiments that the network did not pick up the iris edge around the eyelashes very well (as can be seen in figure 5.7(a)). This result was also found by Gunetileke (2001). So, the image was split up into 3 sections as shown below in figure 5.8(a). Section 2 includes the iris edge around the eyelashes, while sections 1 and 3 contain the rest of the image. A network with the same settings as described in experiment 5.1 was trained, for each section. The results of those networks were combined and are shown in figure 5.8(b). The post processed version of this combined image is shown in figure 5.8(c).

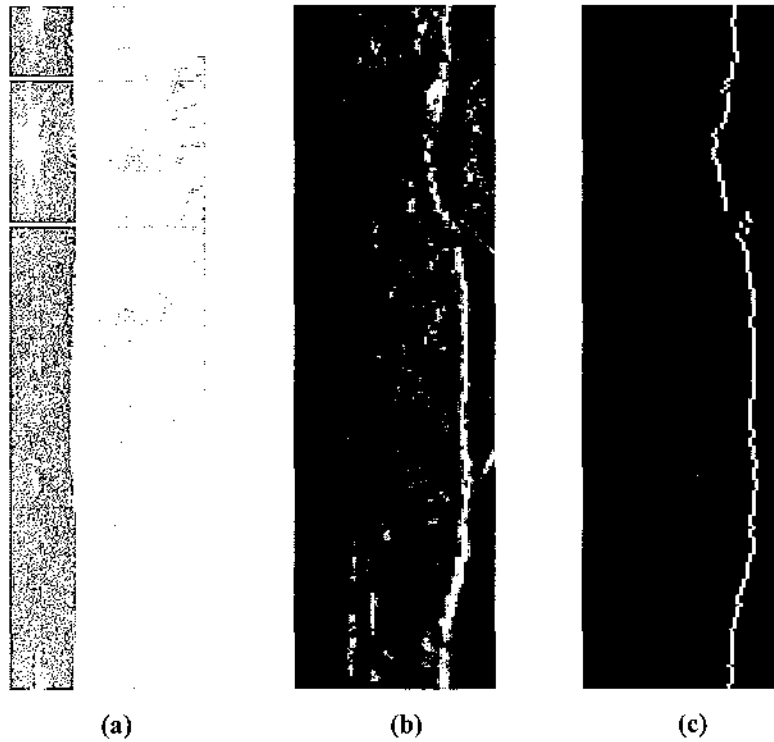


Figure 5.8 Image (a) shows how the input image was split up into 3 parts. Image (b) is the combined output from using 3 networks. Image (c) is the post processed result from (b).

Experiment 5.7

The networks that were trained for section 1 and 2 of experiment 5.6 were then applied to another image, shown in figure 5.9 below.

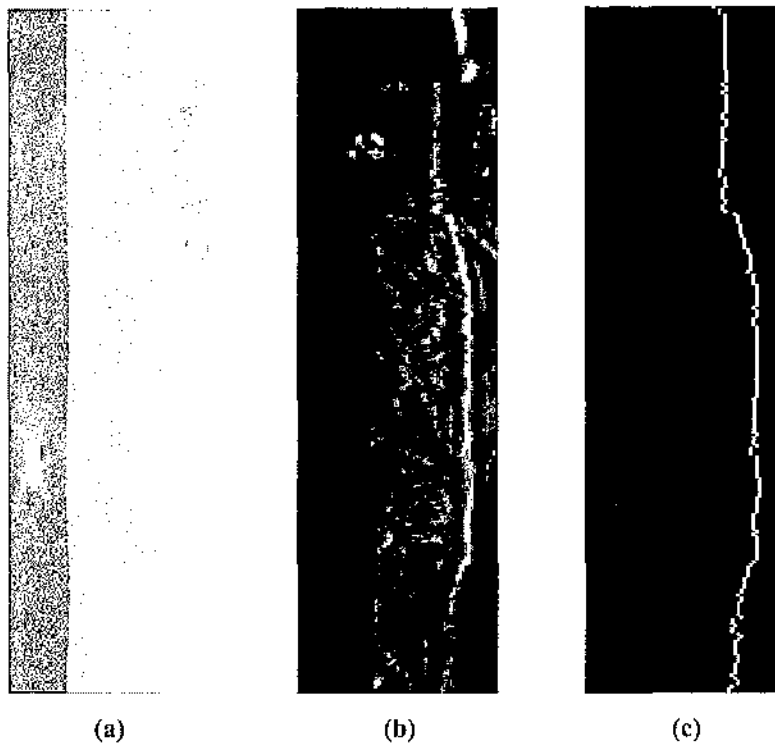


Figure 5.9 Image (a) shows a new input image. Image (b) the output. Image (c) is post processed.

The combined output is shown in figure 5.9(b), and the post processed image in figure 5.9(c). As expected the output image from experiment 5.7 was not as clear and sharp as that from 5.6. However, the post-processed image does seem to match fairly closely back to the input image.

The test of using a trained network as in experiment 5.6, on a new input image as in experiment 5.7, was done a number of times using different cluster area's for network training. Not all the results were as successful as those shown in figures 5.8 and 5.9. Cluster areas were found to have a substantial effect on the resultant images. Poorly chosen cluster simply did not allow the network to find the iris edge.

Conclusion

The number of epochs and number of rules nodes did affect the performance of the network. For the iris edge problem no one specific combination of epochs and rules nodes proved significantly better than the rest, but the results did show interesting trends. Networks trained with between 7 – 12 rule nodes and over 250 epochs, generally performed better than networks with rule nodes and epochs outside this range.

For all of the experiments conducted in this chapter, it was noticed that perhaps the most significant factor affecting the output was the choice of area used to train the network (cluster areas). Even with well written rules, good numbers of epochs and rule nodes etc, poorly chosen cluster areas resulted in poor network performance. The same network trained with only slightly different cluster areas could produce very different outputs. Thus, for future work a study of the effect of cluster selection on output could be done. This could include both the size of the cluster area, and the correlation between the network rules, the cluster area chosen to represent that rule and final network performance.

Chapter 6: Pollen Classification

Introduction

This chapter looks at the use of fuzzy neural networks for classification of pollen images. Currently researchers in the biology department use microscopes and the human eye to classify pollen for their research. However, due to the large numbers of pollen and pollen types, this process is both time consuming and (due to its laborious nature) subject to human error. It was considered that trained fuzzy neural networks could be used to classify pollen, saving both time and energy of the researchers.

Methods for Finding Features of Pollen Images

The specific problem involves a series of forty pollen types that need to be classified. For the purposes of this research, it will be assumed that the distance between the camera and the pollen, and the camera's focus are constant. However, it will not be assumed that the surrounding light is of constant intensity, as this is unlikely to be the case even in indoor settings. As the following pictures in figure 6.1 show, the colour of the pollen (light or dark), even if the same type, vary considerably. It may be due to natural variation or it could be due to different light intensities.

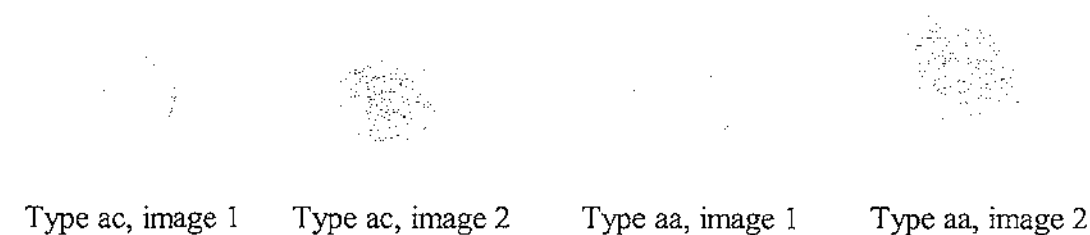


Figure 6.1 Difference in pixel intensity (colour) within pollen types.

Shape Features

Del Bimbo (1999) notes that feature based representations can be found through

1. Parametric internal methods: shapes features describe the region enclosed by the object contour
2. Parametric external methods: shape features describe the external boundary of the object.

For the pollen classification problem, internal methods were considered more suitable. The use of internal methods includes measuring area, the compactness of the region, the elongated-ness of the region, and the minimum/maximum rectangle, ellipse or circle that can be included in the region.

While size or area is a straight-forward concept, shape for real world images like pollen is slightly harder to decipher. Words such as circular, square and triangular can be used to broadly categorise shapes, but we need to measure to what degree an object is for example 'triangular'. Some shapes are more triangular or square than others, and this concept can be used as a method for classification along with size information.

Features based on digital moments also provide useful shape descriptors [3]. For a binary image where R represents the region we want to describe, then the i -th, j -th moment of R is expressed as:

$$M_{ij}(R) = \sum_{(x,y) \in R} x^i y^j$$

Thus, $M_{0,0}(R)$ represents the area of the shape. The centroid of R can be found using:

$$\mu_x = \frac{M_{10}(R)}{M_{00}(R)} \quad \mu_y = \frac{M_{01}(R)}{M_{00}(R)}$$

The i -th, j -th central moment can be described as:

$$\mu_{ij}(R) = \sum_{(x,y) \in R} (x - \mu_x)^i (y - \mu_y)^j$$

Most of the pollen images are somewhat circular in shape, so a measure of circularity would be appropriate as a shape based feature. One problem is defining what exactly a circular shape is. I shall take the view that a circle is defined as a shape where the distance from the centre to edge is the same in every direction. One prominent method that can be adapted for measuring circularity is the ‘Hough’ transform. However, as Kim & Kim (2000) note, “its disadvantages are its massive computation and memory requirements”.

Another much simpler method would be to measure the standard deviation of radii of the shape. The centre of mass could be found (representing the centre of the shape), then the distance to the shape boundary calculated and stored. The deviation of these numbers would give a measure of circularity (circular shapes would give very low standard deviations).

The measure for circularity can be described as:

$$\text{Circularity} = \sqrt{\sum_{i=1}^k \frac{(r_i - \bar{r})^2}{k}}$$

Where ‘ r_i ’ represents the distance between the centre of the shape and the edge points, with a total of ‘ k ’ edge points. I have not found this method for measuring circularity in any relevant literature, but it is possible it has been published previously. An algorithm for implementing the idea of digital moments for binary images is shown in ‘pseudo’ code below.

```

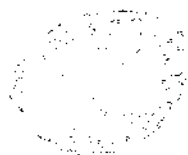
Function Find_Centre_and_Area_Of_Image (image)
% finds the centre and area (in pixels) of a binary image
[vertmax,horizmax] = size(image);
xsum = 0; ysum=0; area = 0;

for x = 1:horizmax
    for y = 1:vertmax
        if (BW2(y,x) == 0) %ie. If pixel is black
            xsum = xsum + x;
            ysum = ysum + y;
            area = area + 1; % increment area calculation
        end
    end
end

% find centroid using digital moments
xcentre = round(xsum / area); % round to nearest pixel
ycentre = round(ysum / area);

```

If the pollen types all had unique size and shape, then the methods mentioned above would be sufficient for classification. However, there are different pollen types with similar shape features, and within each type of pollen, there is significant natural variation of size and shape. The images below in figure 6.2 show some examples of different shape within pollen types.



Type ad, image 1



Type ad, Image 2



Type ac, image 1



Type ac, image 2

Figure 6.2 Differences in shapes within pollen types.

Textural Features

Texture types are something that are readily and easily recognised by humans, through variations in intensity and colour which form repeated patterns. Adjectives commonly used to describe texture include fuzzy, rough, smooth, coarse, granular fine etc. However, defining texture proves to be a more difficult task. Lew (2001) notes that all researchers agree on two points: 1. within a texture there is significant variation in intensity levels between nearby pixels. 2. texture is a homogenous property at some spatial scale larger than the resolution.

As the pollen images are taken with a constant (or almost constant) distance between the camera and the pollen, the spatial scale effect should be negated. Thus, textural features may prove a useful classifier for pollen types. The following images in figure 6.3 represent pollen that appear to the human eye to have markedly different textures.



Type pr, image 16 Type cd, image 5 Type br, image 10 Type sg, image 3

Figure 6.3 Shows the variation of textures between pollen types.

Approaches for Analysing textures

The literature suggests three main approaches for analysing textures:

Statistical Texture Measures. These are a set of features measuring properties such as contrast, correlation and entropy that are used to represent different characteristics of an image. Statistical measures gain their textural properties from variations in brightness that, although random, repeats the same patterns of randomness over the entire textured surface (McLaughlin, 1998).

Stochastic Texture Modelling. “Analysis is performed by defining a model and estimating the parameters so that the stochastic process can be reproduced from the model and associated parameters. The estimated parameters can serve as a model for texture classification.” (Lew, 2001, p54).

Structural Texture Measures. These are useful for images with two-dimensional patterns consisting of primitives or sub-patterns, which are defined in terms of spatial regions. The primitives can be of varying or deterministic shape, such as circles or hexagons. The image is formed from primitives by ‘placement’ rules that specify how the primitive is orientated, both on the image field and with respect to each other (Lew, 2001).

While some images can easily be categorised using one of the texture types listed above, many images lie somewhere in between, with components of both. For the problem of pollen classification, structural methods were examined, especially the spatial grey level dependency matrix method (also known as co-occurrence matrix theory).

Spatial Grey Level Dependency Matrix (SGLDM)

The SGLDM describes texture information in the form of energy, entropy, homogeneity, inertia and correlation. Lu *et al.*, (2002) describe obtaining the SGLDM as counting the number of times each pair of grey levels (u, v) occurs with separation distance d and in the direction specified by angle θ . Figure 6.4 below represents the ‘ u ’ and ‘ v ’ pixels visually for different values of d and θ .

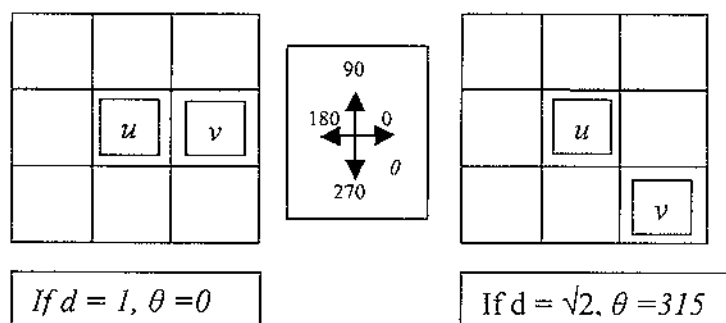


Figure 6.4 Obtaining the SGLDM matrix.

The SGLDM matrix itself is the size of the number of grey levels in the image (e.g. a 256 grey-level image would result in a 256 by 256 SGLDM matrix). For each pair of pixels (u,v) , the element in row u column v of the SGLDM matrix is incremented. For example, if u has a grey level of 256 and v has a grey level of 50, the SGLDM(256,50) is incremented by one. All possible combinations of (u,v) are calculated.

To obtain information from the SGLDM, several different features can be calculated. McLaughlin (1998), Del Bimbo (1999) and Lew (2001) suggest that the five most popular are:

$$\text{Energy} = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} [p(i, j, d, \theta)]^2$$

$$\text{Entropy} = - \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} [p(i, j, d, \theta) \cdot \log_2(p(i, j, d, \theta))]$$

$$\text{Inertia} = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} [(i - j)^2 \cdot p(i, j, d, \theta)]$$

$$\text{Homogeneity} = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} [p(i, j, d, \theta) / (1 + (i - j)^2)]$$

$$\text{Correlation} = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} [(i - \mu_x)(j - \mu_y) \cdot p(i, j, d, \theta) / (\sigma_x \sigma_y)]$$

where

$$\mu_x = \sum_{i=0}^{n-1} i \sum_{j=0}^{n-1} [p(i, j, d, \theta)]$$

$$\mu_y = \sum_{j=0}^{n-1} j \sum_{i=0}^{n-1} [p(i, j, d, \theta)]$$

$$\sigma_x = \sum_{i=0}^{n-1} (i - \mu_x)^2 \sum_{j=0}^{n-1} [p(i, j, d, \theta)]$$

$$\sigma_y = \sum_{j=0}^{n-1} (j - \mu_y)^2 \sum_{i=0}^{n-1} [p(i, j, d, \theta)]$$

The pseudo code on the following page shows a possible implementation of the SGLDM method to obtain texture data for pollen images. (Assumes $d = \sqrt{2}$ and $\theta = -45$).

Hurst Coefficient

Another method sometimes used is the Hurst coefficient. It uses the idea of ‘fractal dimension’ to characterise textures. This is obtained by plotting the greatest differences in brightness between pixels along a line, on log axes of the image area versus the search distance. The slope of this plot is the Hurst coefficient. However, this method may not be suitable when texture changes with orientation (Del Bimbo, 1999).

Function find_Textural_information (image)

% finds energy, entropy, homogeneity, inertia and correlation textural information
% using an SGLDM matrix.

[rowmax, columnmax] = size(image);

for j = 1:rowmax - 1 % for theta = -45, d = root(2)

for i = 1:columnmax - 1;

x = image(j,i) + 1; % +1 to x and y to fit into a matrix co-ordinate

y = image(j+d,i+d) + 1; % assumes $\theta = -45$, $d = \text{root}(2)$

% if $\theta = 0$, $d = 1$ then use: $y = \text{image}(j, i+d) + 1$

sgldm(y, x) = sgldm(y, x) + 1;

r = r + 1; % r is the normalising factor

end

end

% initialise variables

ux = 0; uy = 0; stdx = 0; stdy = 0; cortotal = 0;

totalenergy = 0; inertia = 0; totalhom = 0; entropy = 0; totalinert = 0;

for j = 1:glevel

for i = 1:glevel

totalenergy = totalenergy + sgldm(j,i)*sgldm(j,i);

totalhom = totalhom + sgldm(j,i) / (1+(i-j)*(i-j));

totalinert = totalinert + ((i-1)-(j-1))*((i-1)-(j-1))*sgldm(j,i);

ux = ux + ((i-1)*sgldm(j,i));

uy = uy + ((j-1)*sgldm(j,i));

if (sgldm(j,i) ~= 0)

entropy = entropy + (sgldm(j,i)/r) * (log2(sgldm(j,i)/r));

end

end

end

ux = ux / r;

uy = uy / r;

continued next page

```

for j = 1:glevel
    for i= 1:glevel
        cortotal = cortotal + ((i-1) - ux)*((j-1) - uy)*sgldm(j,i);
        stdx = stdx + ((i-1)-ux)*((i-1)-ux)*sgldm(j,i);
        stdy = stdy + ((j-1)-uy)*((j-1)-uy)*sgldm(j,i);
    end
end

stdx = stdx / r;
stdy = stdy / r;
correlation = cortotal / (r*stdx*stdy);
energy = totalenergy / (r*r);
homogeneity = totalhom / r;
entropy = -entropy ;
inertia = totalinert / r;

```

Conclusion of Methods

Given the large natural variation of both shape and textural features within each pollen type, it is unlikely that one single method or measure would provide a suitable means for classification. Select combinations of both shape and texture information using some of the methods described above, should prove useful for categorising various types of pollen. However, it remains to be seen exactly what combination proves most robust and accurate, given the natural variations occurring within each pollen-type.

General Methodology

The task of classifying all forty types of pollen in one go seemed unlikely to be a realistic possibility. It was decided to use the FuNN's in series to break down the original pollen type into smaller and smaller sub-categories, such as the example shown in figure 6.6. The original names of the pollen and the corresponding codes used during the testing are shown in appendix 1.

Firstly, a neural network was trained with data from pollen types required to be classified by that network (an example is shown in figure 6.5 below). Input values for network training were based on shape and textural information from the pollen images. It was decided to try and break the pollen into a few (two or three) categories, to avoid confusing the network. After 'test runs' the choice of target values was refined based on results of the previous test. This iterative process was continued until the majority of actual data was correctly classified by the network.

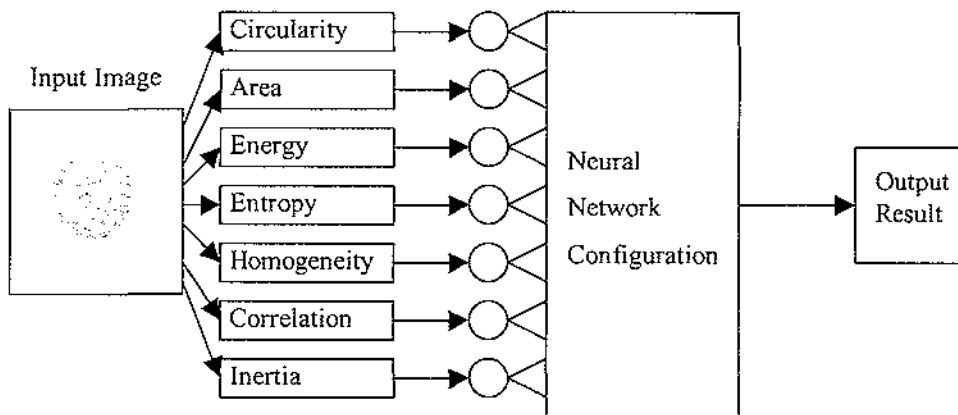


Figure 6.5 Example of Network Structure

After the initial testing, various other techniques to measure pollen features were added to the input values, to help train the network. These features were tested in combinations until the network with the least classification errors was found. Pollen which did not categorize well (i.e. half in one category half in another) were examined. Rules were then devised using characteristics of the extracted data, to try and rectify these incorrect classifications, and thus improve network performance.

This process was repeated for each new network. The test pollen images used to train each network were selected, assuming the previous network was 100 percent correct. For example, only pollen images regarded as type B were used to train the Network B. Incorrect classifications resulting from network A were disregarded during individual network testing.

For all inputs, actual values were adjusted so as to be a number between 0 and 1. For example for 'area', the following calculation was used:

Area = number of pixels in pollen / 15000 (Any results above 1 were rounded down)

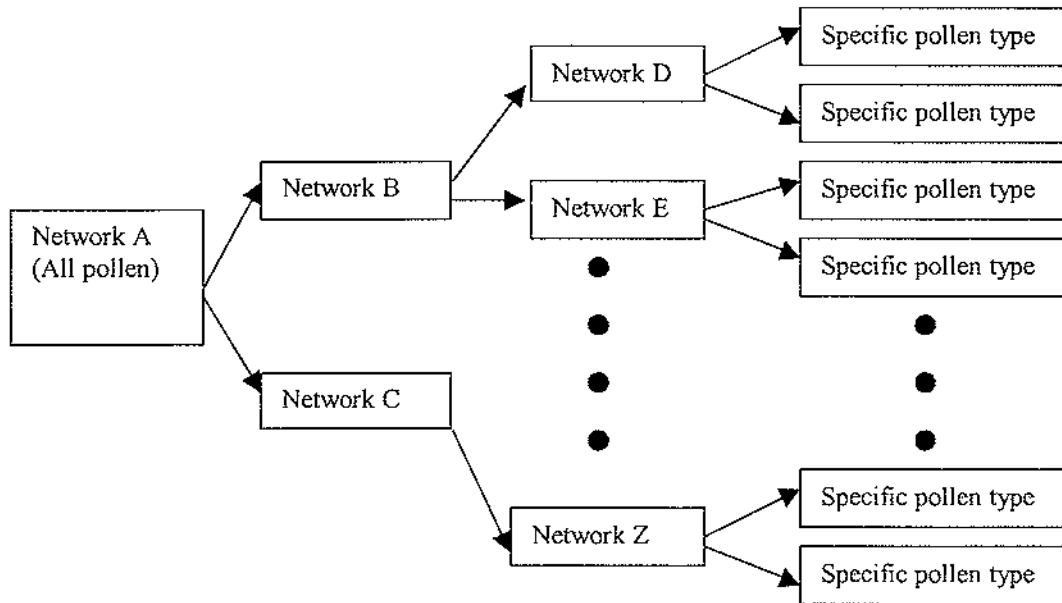


Figure 6.6 Example of a combined Pollen Classification Network

Extraction of SGLDM Data

The SGLDM matrix was applied to the pollen images to obtain information for energy, entropy, homogeneity, inertia and correlation. However, as the matrix is difficult to apply to circular objects, and the corners of the pollen images contained only 'white space', an adjustment was made. Instead of trying to apply the matrix to the whole pollen image, a square shape from the middle of each pollen was used. The centre of this square is based at the centre of mass for each pollen, and the width is calculated as 60% of the length of the pollens radius. The value of 60% was derived from an iterative process. This procedure is shown diagrammatically in figure 6.7 below.

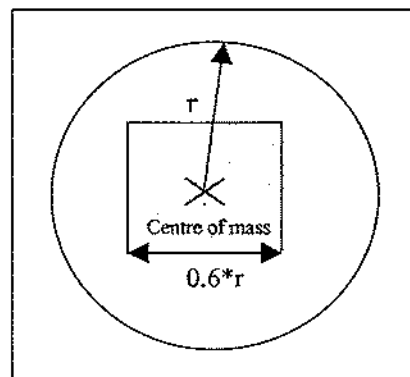


Figure 6.7 Obtaining a square shape from circular pollen for processing by the SGLDM matrix.

Experiments: Training Neural Network ‘A’

Experiment 6.1

Given the large number of pollen to be classified, it was decided to try and break them into two categories. Target values were selected originally based on two shape features - circularity and area. Although the circularity and area values for most pollen overlapped considerably, those pollen with a tendency one way or the other were categorised.

Input Layer	2 {circularity area}
Condition Layer	3 fuzzy membership values {0, 0.7, 1}, thus 6 nodes
Rule Layer	0 rules + 10 free = 10 nodes
Action Layer	2 nodes
Output Layer	1 node {0 represents type ‘C’, 1 represents type ‘B’}

Table 6.1 Network settings for experiment 6.1

Pollen types {aa, ad, ap, bd, bw, cd, cm, ctree, fa, jr, lp, pm, pr, ps, ta and zm} were categorized into type B. Five examples of each type of pollen were used for training data, a total of thirty images of each type of pollen (1200 total) were used for testing. The network settings are shown in table 6.1, and the results are shown below in table 6.2. Examples of type ‘A’ and type ‘B’ images are shown in figure 6.8 below.

Many pollen types were categorized with a high (90%+) success rate. However, others proved far less successful. Additional features were added to the training values to try and add more accuracy to the network.

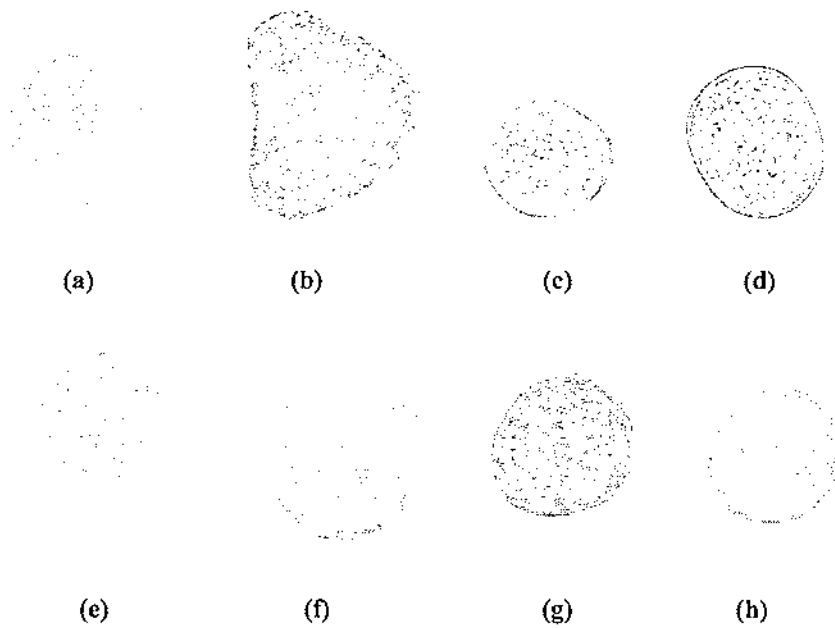


Figure 6.8 Images (a) – (d) are examples of type ‘A’ pollen and images (e) – (h) are examples of Type ‘B’

Pollen Code	Percent Classified (without rules)
cm	67%
cor	83%
dg	77%
es	53%
lop	83%
pa	80%
ps	77%
qr	77%
ro	57%
sg	80%
all others	85-100%
Total	92%

Table 6.2 Percent of pollen classified – Experiment 6.1

Experiment 6.2

The number of pollen used to train the network was increased to 10 examples of each type of pollen. Other settings remained the same as experiment 6.1 and the results are

shown in table 6.3 below. Results for specific pollen were altered, but the overall success rate did not improve. This indicates that the number of 5 pollen used to train the network is sufficient. Perhaps a using a large number of pollen (such as 100 examples) would produce slightly better results, but this may also result in over-training.

Experiment 6.3

The settings of experiment 6.1 were retained but the number of input variables used to train the network was increased. Various combinations of energy, entropy, homogeneity, inertia, correlation, area and circularity were tested.

However, on the whole the results were less successful, with total error rates around or just over 10% (meaning less than 90% correct classification). However, one particular combination using area, circularity, inertia and correlation, did prove relatively successful giving a total success rate of 92%. Although this was only as good as Experiment 6.1, it was considered that this combination would give more flexibility when writing rules to use for network training and thus was used in experiment 6.4.

Pollen Code	Percent Classified (without rules)
cm	70%
cor	77%
dg	73%
es	60%
lop	87%
pa	83%
ps	70%
qr	80%
ro	50%
sg	67%
all others	85-100%
Total	92%

Table 6.3 Percent of pollen classified – Experiment 6.2

Experiment 6.4

Using data from area, circularity, inertia and correlation to train the network, conditional rules were added to try and improve network performance. The rule used was “If $I1 > M2$ then $O1 = M2$ ”. This simply means that if area is large then output should be type B. The settings from this experiment are shown in table 6.4 and the results are shown below in table 6.5.

Input Layer	4 { area circularity inertia correlation }
Condition Layer	3 fuzzy membership values {0, 0.7, 1}, thus 12 nodes
Rule Layer	1 rule + 9 free = 10 nodes
Action Layer	2 nodes
Output Layer	1 node {0 represents type ‘C’, 1 represents type ‘B’}

Table 6.4 Network settings for Experiment 6.4

Pollen Code	Percent Classified	
	With rules	Without rules
al	87%	87%
cm	73%	50%
dg	80%	87%
es	77%	80%
lp	87%	80%
nf	87%	93%
pa	80%	87%
ps	67%	63%
ro	60%	63%
sg	87%	93%
all others	90-100%	90-100%
Total	93.3%	92.0%

Table 6.5 Percent of pollen classified – Experiment 6.4

The results show that the rule had a slight positive influence on the percentage of correctly classified pollen. Various other rules and combinations of rules were tested but the results did not improve.

Experiments: Training Neural Network 'B'

In experiment 6B, the 14 pollen classified from Network A as Type B are used for training Network B. The area calculation was adjusted for this experiment as most of the pollen had relatively large area which was rounded to '1' in the previous experiments. The new adjustment formula was:

$$\text{Area} = \text{No. pixels} / 30000$$

Experiment 6B.1

It was noticed that 'zm' pollen had high correlation low inertia and large size. Thus, it was considered as a prime candidate for separation. It was also noticed that 'ad' and 'pr' pollen had high inertia and low correlation compared to the others, so they two were designated to be separated. It was found that having all seven inputs for training the network gave the best results. Images of the zm, ad and pr pollen can be seen in figure 6.9 below. A rule was devised based on trial data to try and help with the classification process. The rule reads:

If $I1 > M3$ $I1 < M4$ $I4 < M4$ $I5 > M4$ Then $O1 = M1$. Meaning if circularity is between two set limits, entropy is low and homogeneity is high then output is low (type D). This is to help classify type D pollen. The 'zm' pollen was classified as 'zm', the 'ad' and 'pr' pollen were classified as type 'D'. The network settings are shown in table 6.6 and the results are shown below in table 6.7.

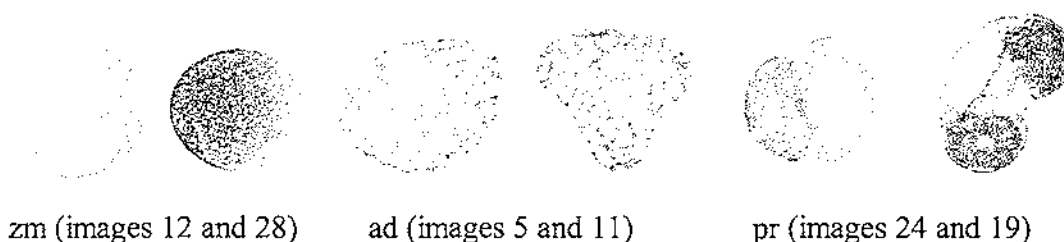


Figure 6.9 Different pollen from type 'B'

Although the result values were different for the two networks, the actual classification was identical and thus, the rule did not improve the outcome. Various

other rules were used to try and improve performance. However, the other rules either resulted in worse performance or no change. Also experiments to split the 'ad' and 'pr' pollen at this stage proved relatively unsuccessful.

Input Layer	7 { circularity area energy entropy homogeneity inertia correlation}
Condition Layer	8 fuzzy membership values {0, 0.02, 0.11, 0.14, 0.63, 0.70, 0.77, 1.00}, thus 56 nodes
Rule Layer	1 rule + 9 free = 10 nodes
Action Layer	2 nodes
Output Layer	1 node {0 = type 'D', 0.5 = type 'E', 1 = 'Zm'}

Table 6.6 Network settings for Experiment 6.B1

Pollen Type 'B'	Percent Classified	
	With Rules	Without Rules
aa	100%	100%
ad	100%	100%
ap	100%	100%
bd	100%	100%
bw	100%	100%
cd	100%	100%
ctree	100%	100%
fa	100%	100%
jr	100%	100%
lp	100%	100%
pm	100%	100%
pr	90%	90%
ta	100%	100%
zm	100%	100%
Total	98.6%	98.6%

Table 6.7 Percent of type 'B' pollen classified – Experiment 6B.1

Experiments: Training Neural Network ‘D’

Experiment 6D.1

In this experiment the type D pollen (‘ad’ and ‘pr’) were used to train the network. It was noticed that difficult to classify ‘pr’ pollen had energy values close to 1 and entropy values in a tight range from 0.1 to 0.36. Thus, the following rule was written to assist in training the network:

If I1=M9 I2>M5 I2<M6 Then O1=M1

The network settings are shown in table 6.8 and the results are shown in table 6.9.

Input Layer	7 { energy entropy homogeneity inertia correlation circularity area }
Condition Layer	9 fuzzy membership values {0, 0.02, 0.03, 0.07, 0.10, 0.36, 0.61, 0.67, 1.00}, thus 63 nodes
Rule Layer	1 rule + 9 free = 10 nodes
Action Layer	2 nodes
Output Layer	1 node {1 = ‘ad’, 0 = ‘pr’}

Table 6.8 Network settings for Experiment 6D.1

Pollen Type ‘D’	Percent Classified	
	With Rules	Without Rules
ad	93%	93%
pr	93%	93%
Total	93.3%	93.3%

Table 6.9 Percent of type ‘D’ pollen classified – Experiment 6D.1

The rule had only a very minor impact on the network output and no effect on the resulting classifications. Various rules and combinations of rules were tried, targeting the mis-classified pollen, but none affected the final classifications. Although these pollen types were visually different, they both had large natural variation and each with a few ‘exceptional’ pollen. It was these ‘exceptional’ pollen that proved difficult to classify. It is possible that a large number of images used for training could reduce this problem, however these were not available at the time of the experiments.

Experiments: Combining Neural Networks

Experiment 6S.1

In this experiment the three trained networks (A, B and D) from the previous experiments were combined together in series. Thus, inputs into networks B and D would not be 100% correct and errors would be cumulative. The overall network structure can be seen below in figure 6.10, although no actual network was implemented for 'C' or 'E' at this stage.

The results in table 6.10 below show the overall percentage correctly classified is 93.0%. Although this is lower than all the previous networks (as expected), it is not drastically lower and thus the cumulative errors did not badly affect the overall network.

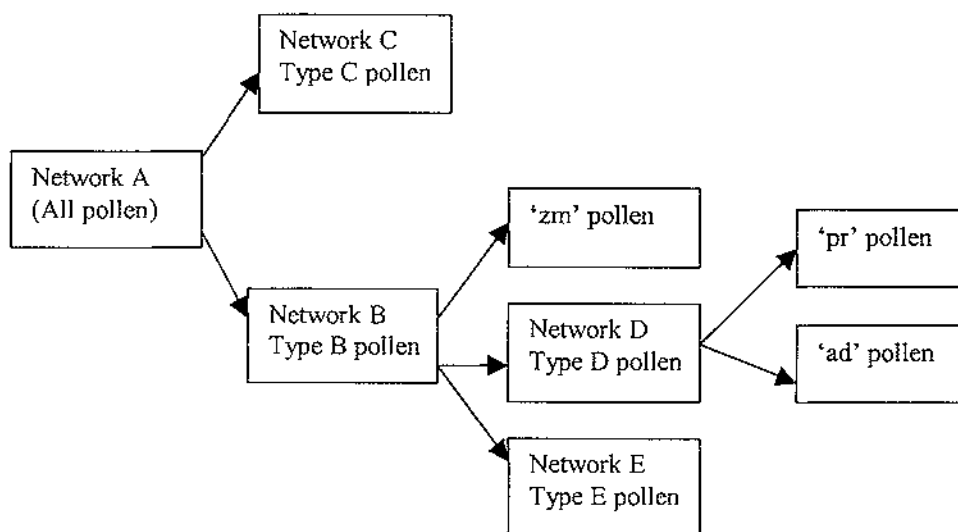


Figure 6.10 Pollen Classification Network used for experiment 6S.1

Pollen Code	Percent Classified (with rules)
al	87%
cm	77%
dg	80%
es	77%
lp	87%
pa	80%
pr	87%
ps	67%
ro	60%
all others	90-100%
Total	93.0%

Table 6.10 Percent of pollen classified using combined networks in series

Experiment 6E.1

In experiments 6E, the 11 pollen classified from Network B as Type E are used for training Network E. It was noticed that the 'ta' pollen had correlation values in a tight band between 0.11 and 0.27. Also, 'ta' had circularity in a band between 0.27 and 0.43 with area generally over 0.59. These features are represented in the rule below (remembering that I5 is correlation, I6 is circularity and I7 is area for this network).

If I5>M3 I5<M4 I6<M5 I6>M6 I7>M7 Then O1=M3

Figure 6.11 shows an example of the 'ta' pollen, and of some other visually similar pollen in the group.

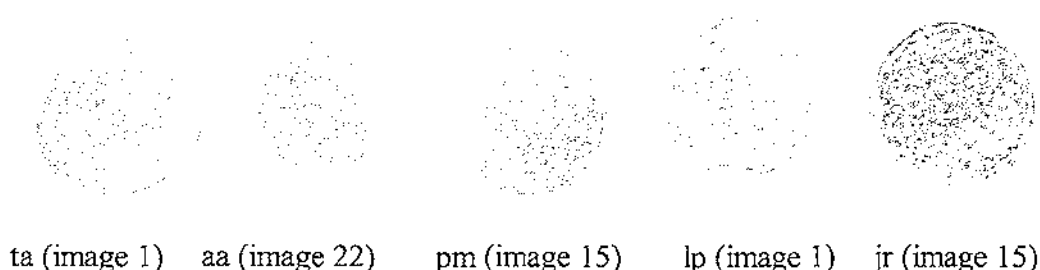


Figure 6.11 Various pollen from the Type E range, showing visual similarity

The network settings shown in table 6.11 show the network settings for this experiment. The results are shown in table 6.12 below. The identical total classification percentages with and without rules indicate that the rule had no significant impact. However, the results were very good considering the obvious visual similarities between pollen in this category.

Input Layer	7 { energy entropy homogeneity inertia correlation circularity area }
Condition Layer	8 fuzzy membership values {0, 0.02, 0.11, 0.27, 0.43, 0.52, 0.59, 1.00}, thus 56 nodes
Rule Layer	1 rule + 9 free = 10 nodes
Action Layer	2 nodes
Output Layer	1 node {0 = type 'F', 1 = 'ta'}

Table 6.11 Network settings for Experiment 6.E1

Pollen Type 'E'	Percent Classified	
	With Rules	Without Rules
aa	100%	100%
ap	100%	100%
bd	100%	100%
bw	90%	90%
cd	90%	90%
ctree	100%	100%
fa	100%	100%
jr	100%	100%
lp	100%	100%
pm	100%	100%
ta	97%	97%
Total	97.8%	97.8%

Table 6.12 Percent of type 'E' pollen classified – Experiment 6E.1

Experiment 6S.2

Experiment 6S.1 was repeated, but this time included Network E. The overall network structure is shown in figure 6.12 and the results shown in table 6.13.

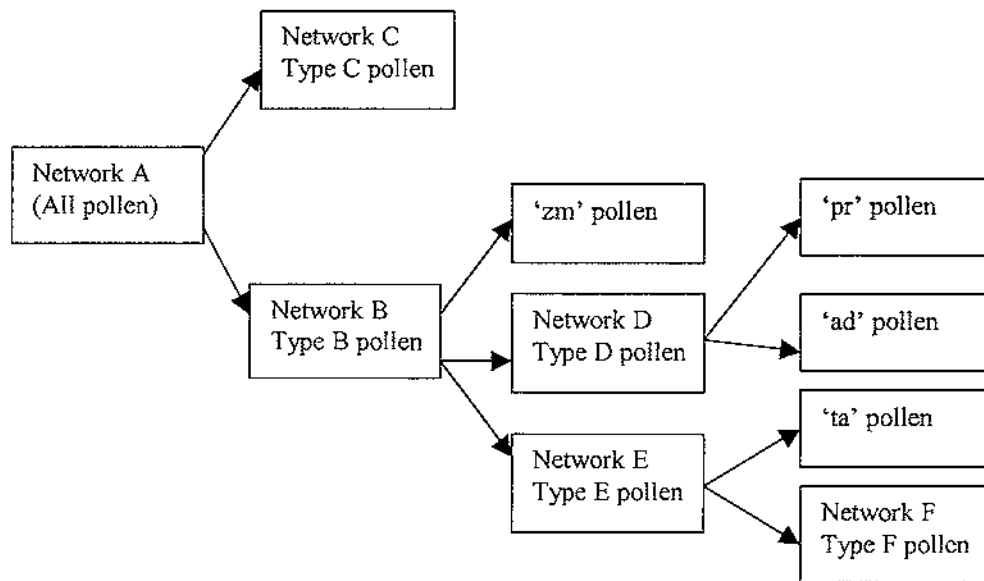


Figure 6.12 Pollen classification network used for experiment 6S.2

Pollen Code	Percent Classified (with rules)
al	87%
bw	86%
cm	77%
dg	80%
es	77%
lp	87%
pa	80
pr	87%
ps	60%
ro	60%
all others	90-100%
Total	92.3%

Table 6.13 Percent of pollen classified using combined networks in series

The total percentage correctly is only slightly down (0.7%) on the same figure from experiment 6S.1.

Final System

The final classification system, that can be implemented using the 'classify a single pollen' button in the 'Run a NWF Network' GUI, is the same as that described in

Experiment 6S.2. The previously developed networks were combined into one system, where the output of one network served as the input to the next (as is described by figure 6.12). The final classification types were type 'C', zm, pr, ad, ta and type 'F'. The success rate for each type is shown in table 6.14 below, figures are calculated based on a sample size of 1200 (i.e. 30 examples each of 40 kinds of pollen).

Pollen Type	Percent Classified
Type 'C'	92%
Type 'F'	92%
ad	93%
pr	87%
ta	97%
zm	100%
Overall	92.3%

Table 6.14 Results for combined network

Discussion and Conclusion

The results for the combined network showed that over 92 percent of pollen were successfully classified, which seems high given the large natural variation within each pollen type and similarities among pollen. However, the current system is not complete, as it does not identify all pollen types. Type C still contains 24 pollen types to be classified, while type F still has 10. Completion of the system was not attempted due to time constraints. The problem knowledge rules, while improving some of the networks, did not have a huge impact. This indicates that perhaps there was not a great deal of noise in the original data. Completion of the system based on the current format is possible, but may require the use more textural and shape features to differentiate very similar pollen types.

Chapter 7: Conclusions and Discussion

This project describes the development and use of an interface for a fuzzy neural network application. The interface development was followed through, from planning and analysis to testing. Program requirements were elicited from the client during the planning phase. These requirements along with other factors were taken into consideration during the design process, and this was eventually converted into working code. Each function in the code was tested, using unit and integration testing. Through testing, faults in the program were discovered and elements of design and coding were reworked as necessary.

When the interface development phase was nearing completion, the application itself was used to research two specific image processing problems. The first involved discovering the best combination of network settings to perform iris edge detection using a window filter network. Based on what was done with previous research, the variables 'epochs' and 'number of rule nodes' were looked at specifically. While no one combination of these two variables proved significantly better than all the others, a general range of optimal settings was discovered. It was noticed that small changes in cluster areas selected for training produced marked differences in network performance. The exact effect of the choice of cluster areas could be a target of future work.

The second image processing problem that was dealt with was the use of fuzzy neural networks for classification of pollen images. Image processing techniques were used to extract textural and shape information from images of forty different pollen types. This information was fed into a (non-window filter) network along with rules that were devised using problem knowledge from the images. It was discovered that trying to use just one network to classify all the pollen was unfeasible, and that each network was better suited to breaking the input images into two or three different categories. Each of these networks could then be combined in series to classify the pollen into smaller and smaller sub-groups, and eventually isolate individual pollen types. The networks devised during experiments in this research were combined to produce a system that isolated some pollen types, with over a 90% success rate. It was found

that the rules devised to help with classification had a small positive impact, but not very significant. This is likely to be because of the good quality of the input data, and its inherent lack of noise.

Future Work

There are various areas that future research could involve including the interface itself, the pollen experiment, the iris edge experiment and creation of 'rules' representing problem knowledge.

Though the interface conformed closely to user requirements, there are still features to be added that could further improve its usability and make it flexible enough to handle other kinds of image processing problems. In particular, the 'non-window filter' network interface could be looked at. Also, the expansion of the program to include other training methods provided by MATLAB would prove beneficial. Should the 'SuperFuNN' program prove popular as a means of research for other students, completion of internal error checking and documentation would be prudent.













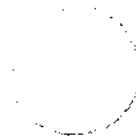




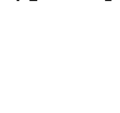



The iris edge detection problem needs further research done into the effects of the selected cluster area on network performance. In particular, the effect of cluster area in relation to the types of rules written to help train the network. Also, most of the experiments focused on using just one pair of input and target images to train the network. The effect of using multiple images to train the network could be researched.


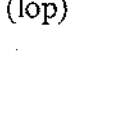






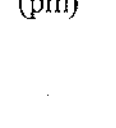










The current pollen classification system requires more work to be able to classify all forty types of pollen. To identify some very similar types of pollen, textural and shape features other than those already used may need to be employed.

Throughout the experiments, the generation of the 'rules' representing problem knowledge was done in an ad-hoc manner. Research into developing a methodology for producing these 'rules' could prove very helpful. This may involve aspects of data mining such as those described in 'Optimal and Robust Rule Set Generation (Li, 2002).

Appendix 1: Pollen List

A list of the pollen types used in chapter 6, and their corresponding abbreviations.

Anthoxanthum (aa) 	Agrostis capillaris (ac) 	Acacia dealbata (ad) 
Alnus (al) 	Acer negundo (an) 	Alopecurus pratensis (ap) 
Amaranthus powelli (apl) 	Bromus diandrus (bd) 	Betula pendula (bp) 
Brachalottis repanda (br) 	Bromus willdenowii (bw) 	Cyathea dealbata (cd) 
Cupressus macrocarpa (cm) 	Corylus (cor) 	Coprosma robusta (cr) 
Cedar Tree (ctree) 	Dactylus glomerate (dg) 	Eucalyptus sp (es) 
Festuca arundibaceae (fa) 	Holcus lanatus (hl) 	Juglans regia (jr) 

Ligustrum lucidum (ll) 	Ligustrum ovalifolium (lop) 	Lolium perenne cv nui (lp) 
Leptospermum scoparium (ls) 	Nothofagus fusca (nf) 	Poa annua (pa) 
Populus deltoides (pd) 	Plantago lanceolata (pl) 	Phalaris minor/aquatica (pm) 
Platanus orientalis (po) 	Pinus radiata (pr) 	Populus sp (ps) 
Quercus robur (qr) 	Rumex acetosella (ra) 	Rumex obtusifolius (ro) 
Sporobolus africanus (sa) 	Sequiadendron gigantea (sg) 	Triticum aestivum (ta) 
Zea mays (zm) 		

Appendix 2: User Manual

The following contains excerpts from the user manual for the 'Super FuNN' application.

SuperFuNN - User Manual

This is the user manual for the 'SuperFuNN' application. This program runs using the MATLAB application and is designed to help the user perform experiments with images using a fuzzy neural network.

Chapter 1: Set-up and Installation

System Requirements

- 500MHz CPU (minimum), 2GHz CPU (recommended)
- 128MB RAM (minimum), 512MB RAM (recommended)
- MATLAB Version 6.1 (or later), Version 6.5 (recommended)
- CD-ROM
- Win95, Win98, Win2000 or WinXP
- 5MB free hard disk space (recommended).

Installation instructions

Follow the instructions below to install and start the 'SuperFuNN' application.

- Insert the CD-ROM into your computer.
- Copy the Folder 'SuperFuNN' to your hard-drive.
- Start MATLAB on your computer and change the current directory of MATLAB to the 'SuperFuNN' folder on your hard-drive.
- Type 'startfunn' in the command window of the MATLAB application.

Chapter 2: Getting Started

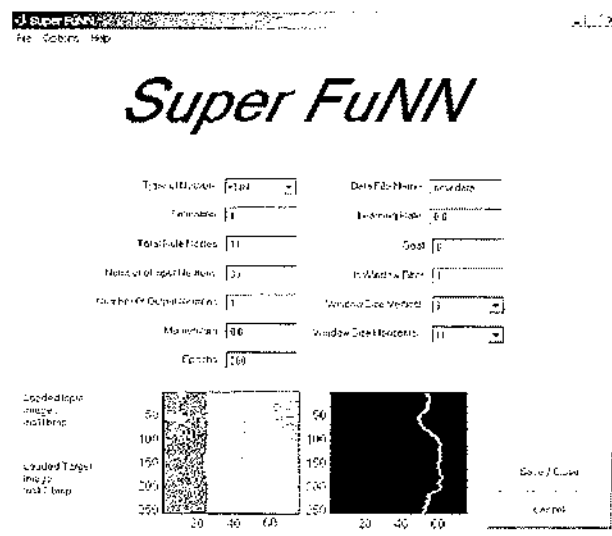
This page describes the quick start method to use the 'SuperFuNN' program.

Step 1. Load Settings. Select the 'File→Load Settings' menu to load previously saved settings, or select 'File→New Settings' to start with the default settings.

Step 2. Load Images. Select the 'File→Load Images' menu to load the first input image, then the target image (they should be of the same dimensions). You will then be asked if you want to enter multiple cluster areas, if you choose 'yes' you will be asked to enter the number of cluster areas. You will then be prompted to 'select the cluster area'. To do this use the drag-select technique with your mouse on either the input image or the target image.

Step 3. Adjust input variables. If you select the 'Options→Set Variables' menu, you will see the screen as in the picture below. You can then change input variables such as the number of epochs, the network type, number of input and output neurons, etc.

To do this, simply enter new values in the edit boxes or select new values from the pull-down menus.



Step 4. Change the Fuzzy and Defuzzy membership Values. Select the

'Options→Change Rules' menu to start the 'Change Fuzzy Membership Rules' GUI. From here you can change the Fuzzy Rule values by altering the edit boxes, this will automatically update the graph. You can also simply click on the graph itself, which will automatically update the edit boxes. From here you can also push the 'View Text Rules' button which starts the 'Adjust Rules' GUI (see step 5 below). Note, pressing cancel here will also result in changes made in the 'Adjust rules' GUI not being saved.

Step 5. Change the Rules. In the 'Adjust rules' GUI the each rule (one at a time) is displayed in text, grid and graphical formats. You can use the mouse to change values in the grid, which automatically updates the written text rule and the graphical representation. You can also change the rule by entering text in the 'edit box' displaying the rule. When you have finished editing the text, press the 'Update Graph' button to update the grid and the graphical representation. The 'Update Graph' button is only needed when updating the rule via the text box'. Press 'close' when finished and you will be returned to the 'Change Fuzzy Membership Rules' GUI.

Note: Be sure to only enter legal values in the text rule. For example, if there are only five fuzzy membership values and two defuzzy membership values, do not enter 'If I1=M6 Then O1=M3', as this will result in an error.

Step 6a. Run the window filter network. If the network is set up as a window filter network (i.e. 'IsWindowFilter' is set to 1), then from the 'Set user options' GUI select the 'Options→Run Network→Train and Run New Network' menu to train and run the network. This allows you to run the network once with the current settings, to give you an idea of network performance. The program should display the training performance graph, the output image and error calculations. If the network is performing as expected move on to Step 6, otherwise you can repeat any of the previous steps until you are satisfied.

Step 6b. Run the non-window filter network. If the network is set up as a non-window filter network (i.e. 'IsWindowFilter' is set to 0), then select the 'Options→Run NWF Network' menu to start the 'Non window filter network' GUI. From here you can press the 'Classify a single pollen' button. This enables you to

load an input image and have it classified using the currently installed criteria. For further instructions on training the network or altering the classification criteria see 'Chapter 5'.

Step 7. Use Batch Run (only for window filter networks). Select the 'Options→BatchRun' menu to start the batch run GUI. This enables the user to perform a series of runs using the current images and settings. In 'BatchRun' you can adjust 'number of epochs' and 'number of rule nodes', you can run the network with constant settings but varying the 'random' factor, or you can run the network with 'no-rules'. Batchsave produces an 'errors' file that contains information about error between the output image(s) and the target image.

Chapter 3: Example Experiment

Start the program

- Install and start the SuperFuNN application as explained in Chapter 1 of this manual.

Load the example network settings

- Select 'File→Load Settings' and choose the 'iris002' file from the settings folder.

Adjust settings, fuzzy membership values and rules

- To adjust network settings select 'Options→Set Variables'. You should now see a screen displaying input settings such as 'Type of Network', 'Saturation' and 'Epochs' (see diagram in Chapter 2). Try changing the Epochs to 400. If you wish to change the 'window filter size' it is recommended you start from 'new settings', otherwise be sure to adjust the network rules as some rules may become invalid. Press 'Save/Close' when finished.

- To adjust fuzzy membership values select 'Options→Set Rules', this will open the 'Change Fuzzy Membership values' GUI. Change any of the Fuzzy membership values by clicking on the graph or using the edit boxes situated just below the graph (press 'enter' after each alteration to make the graph change). You may change the number of values by pressing the 'Add fuzzy MFDEF' button or the 'Delete fuzzy MFDEF' button. It is recommended to keep the first value as '0' and the last value as '1', or the graph may not display properly.
- To adjust the problem knowledge 'rules', press the 'View text rules' button, this should open the 'Adjust Rules' GUI. From here you can use the mouse to click on the grid and change the rules. Select a value (say 'M2') from the pull down menu to the right of the grid, now drag the mouse over a number of boxes (holding the mouse button down), then release. All the selected boxes should now be changed to the selected value, also the text in the edit box will also change accordingly. You can also change the rule via the edit box directly, but remember to press the 'update graph' button after doing so. Be sure not to make the 'M' values higher than the number of fuzzy rules (or number of defuzzy rules in the case of the output part).
- When finished making changes to the problem knowledge rules press the 'close' button in that GUI. When finished making changes to the 'fuzzy membership values' press the 'Save/Close' button in that GUI.
- NOTE. Changing values in the 'set user options' GUI or the 'Change Fuzzy Membership values' GUI while the 'Adjust Rules' GUI is open will cause concurrency problems. Close the 'Adjust Rules' GUI or the 'Change Fuzzy Membership values' GUI when you have finished making changes in that GUI.

Load input and target images

- Select 'File→Load Input and Target Images' and choose the 'Iris1I.bmp' file for the input image and 'Iris1T.bmp' for the target image (both in the 'aImages' folder). You will then be asked if you wish to select multiple cluster areas, this may depend on the rules you want to train your network with. When selecting cluster areas, use a drag select approach. The selected area(s) will be bounded by a blue rectangle.

Trial Run the Network

- Select 'Options→Run Network→Train and Run Network' to start training and then running the network. If there are no errors in your network settings or rules, the network will be trained using the selected area from the input image and the target image. You will see the rule performance graph on the screen. The network will then be simulated on the input image and the resulting image will be displayed on the screen along with error calculations.

Batch Runs and Further Work

- You may wish to change some of the settings or rules based on your trial run. You can repeat this process as many times as you want, saving settings as you go. When ready, select 'Options→Batch Process', this will open the Batch Process GUI. From this GUI, select which variables you wish to alter during the batch run. For example, press the 'Vary Rule Nodes' button and from the resulting dialog box set the rule nodes to some appropriate values (use the default settings if you wish). After this press the 'Run' button to start the experiments. Depending on the settings, this may take some time and you may wish to simply leave the computer running until it has finished. If you do not want all the output images to be displayed, be sure to turn the 'display output graph' toggle button to 'Off'. The resulting error calculations will be displayed when processing has finished, also they will be saved to disk in the 'BatchProcess' folder.

Common User Mistakes and Known Program Bugs

- If the user changes values in the 'Set User Options' GUI or the 'Change Fuzzy membership values' GUI while the 'Adjust Rules' GUI is open, concurrency problems will occur. This may also happen while changing values in 'Set User Options' GUI when the 'Adjust Rules' GUI is open.
- When a fuzzy rule is added or deleted, the graph may not display properly until a new fuzzy membership value is entered.

- If the user selects a cluster area that is too small, then the previously selected cluster areas will be erased (only visually, the memory still contains the coordinates).
- The limit for the number of fuzzy membership values currently stands at ten (this is due to screen size limitations, more than ten values cannot be displayed on the screen).
- When the window filter size is changed after rules have been added, then the rule display in the 'Adjust Rules' GUI will be affected. In this case delete the text rules and write them again (or use the grid).
- Occasionally the program crashes when adding multiple numbers of rules. It is recommended adding one rule at a time
- Problems occur when an incorrect rule is added. The rule should be in the form "If $IX=MY$ Then $O1=MZ$ ", where Y is a number less than or equal to the number of fuzzy membership values, Z is a number less than or equal to the number of defuzzy membership values.

Appendix 3: Code Listings

The following is a listing of the functions used in the SuperFuNN application. The first function 'getSGLDMdata' is an example of a stand-alone function that is not listed under the heading of one particular GUI. The following functions are printed under the heading of the GUI they are associated with. The printing of the GUI functions are in the following order:

AdjustRules GUI
Batchprocess GUI
ChangeFuzzyMembershipValues GUI
Run NWF Network GUI
PrintGenerator GUI
SetUserOptions GUI

Then remaining functions are listed, which are not associated with any one particular GUI.

function getSGLDMdata (no specific GUI)

```
function [energy, homogeneity, entropy, inertia, correlation] = getSGLDMdata(imagein)
% find spatial grey-level dependency data
% currently assumes d = root(2), theta = -45 and image has 256 gray-levels
imagein = double(imagein);
glevel = 256;
sgldm(glevel,glevel) = 0;
r = 0; % r represents the normalising factor
% find sgldm matrix
d = 1;
theta = 0;
%-----
had = d*cos(theta*2*pi/360);
vad = d*sin(theta*2*pi/360);
had = round(had);
vad = -round(vad);

[rowmax, columnmax] = size(imagein);
%for j = 1:rowmax % for theta = 0, d = 1
for j = 1:rowmax - 1 % for theta = -45, d = root(2)
    for i = 1:columnmax - 1;
        %+1 to x and y to fit into a matrix co-ordinate
        x = imagein(j,i) + 1;
        y = imagein(j+d,i+d) + 1; %theta = -45, d = root(2)
        %y = imagein(j,i+d) + 1; % theta = 0, d = 1
        sgldm(y , x) = sgldm(y , x) + 1;
        r = r + 1;
    end
end
%disp(sgldm)
ux = 0; uy = 0; stdx = 0; stdy = 0; cortotal = 0;
totalenergy = 0; inertia = 0; totalhom = 0; entropy = 0; totalinert = 0;
% find values
for j = 1:glevel
    for i = 1:glevel
        totalenergy = totalenergy + sgldm(j,i)*sgldm(j,i);
        totalhom = totalhom + sgldm(j,i) / (1+(i-j)*(i-j));
```

```

    totalinert = totalinert + ((i-1)-(j-1))*((i-1)-(j-1))*sgldm(j,i);
    ux = ux + ((i-1)*sgldm(j,i));
    uy = uy + ((j-1)*sgldm(j,i));
    if (sgldm(j,i) ~=0)
        entropy = entropy + (sgldm(j,i)/r) * (log2(sgldm(j,i)/r));
    end
end
end
ux = ux / r;
uy = uy / r;
for j = 1:glevel
    for i= 1:glevel
        cortotal = cortotal + ((i-1) - ux)*((j-1) - uy)*sgldm(j,i);
        stdx = stdx + ((i-1)-ux)*((i-1)-ux)*sgldm(j,i);
        stdy = stdy + ((j-1)-uy)*((j-1)-uy)*sgldm(j,i);
    end
end
stdx = stdx / r;
stdy = stdy / r;

correlation = cortotal / (r*stdx*stdy);
energy = totalenergy / (r*r);
homogeneity = totalhom / r;
entropy = -entropy ;
inertia = totalinert / r;

% extra adjustments to make values 0 - 1
%assumes smimage being evaluated

energy = energy*100;
entropy = entropy / 12.5;
inertia = inertia / 1500;
correlation = correlation * 100;
if energy > 1
    energy=1;
end
if entropy > 1
    entropy=1;
end
if inertia > 1
    inertia=1;
end
end

```

Functions for AdjustRules GUI

```

function varargout = AdjustRules(varargin)
% GUI to allow the changing of text rules

if nargin == 0 % LAUNCH GUI

    fig = openfig(mfilename,'reuse');

    % Generate a structure of handles to pass to callbacks, and store it.
    handles = guihandles(fig);
    guidata(fig, handles);

%-----opening code-----
disp('    Adjust Rules section started');
load ('ARuletempqpzm10a'); %contains current rule number
set(handles.title,'userdata',Rule); %save current rule into userdata

```



```

currentrule = Rule;
disp(['Rule ', Rule]); % close('tempqpm10a'); % doesn't work
load ('AdjustRtempqpm10'); % load the temp file saved by datainput3
% gui, which contains the current SNN settings
set(handles.mydata,'Userdata',SNN); % save SNN to mydata userdata
displaygraph(handles, currentrule); % display text rule on the "graph"
displayfuzzyValues(handles, SNN.FuzzyMFDEF);

% not needed if displaying only one rule at a time
% display an appropriate number of radio buttons i.e. 1 button per rule
% and check the current rule number to switch the 'current' button on
if (SNN.NumberRules >= 1) set(handles.m1,'visible','on'); end
if (SNN.NumberRules >= 2) set(handles.m2,'visible','on'); end
if (SNN.NumberRules >= 3) set(handles.radiobutton3,'visible','on'); end
if (SNN.NumberRules >= 4) set(handles.radiobutton4,'visible','on'); end
if (SNN.NumberRules >= 5) set(handles.radiobutton5,'visible','on'); end
if (SNN.NumberRules >= 6) set(handles.radiobutton6,'visible','on'); end
if (SNN.NumberRules >= 7) set(handles.radiobutton7,'visible','on'); end
if (SNN.NumberRules >= 8) set(handles.radiobutton8,'visible','on'); end
if (SNN.NumberRules >= 9) set(handles.radiobutton9,'visible','on'); end
if (SNN.NumberRules >= 10) set(handles.radiobutton10,'visible','on'); end
if (SNN.NumberRules >= 11) set(handles.radiobutton11,'visible','on'); end

if (Rule == 1) set(handles.m1,'value',1); end
if (Rule == 2) set(handles.m2,'value',1); end
if (Rule == 3) set(handles.radiobutton3,'value',1); end
if (Rule == 4) set(handles.radiobutton4,'value',1); end
if (Rule == 5) set(handles.radiobutton5,'value',1); end
if (Rule == 6) set(handles.radiobutton6,'value',1); end
if (Rule == 7) set(handles.radiobutton7,'value',1); end
if (Rule == 8) set(handles.radiobutton8,'value',1); end
if (Rule == 9) set(handles.radiobutton9,'value',1); end
if (Rule == 10) set(handles.radiobutton10,'value',1); end
if (Rule == 11) set(handles.radiobutton11,'value',1); end

% don't display axes if not window filter type of network
if (SNN.IsWindowFilter == 0)
    set(handles.axes2,'visible','off');
    set(handles.axes3,'visible','off');
    set(handles.text8,'visible','off');
    set(handles.edit2,'visible','off');
    set(handles.text6,'visible','off');
    set(handles.text7,'visible','off');
    set(handles.text3,'visible','off');
    set(handles.accept,'string','Accept Changes');
end
% *****

    if nargin > 0
        varargin{1} = fig;
    end

elseif ischar(varargin{1}) % INVOKE NAMED SUBFUNCTION OR CALLBACK

    try
        if (nargout)
            [varargout{1:nargout}] = feval(varargin{:}); % FEVAL switchyard
        else
            feval(varargin{:}); % FEVAL switchyard
        end
    end

```

```

        catch
            disp(lasterr);
        end
    end

end

% --- Executes just before AdjustRules is made visible.
function AdjustRules_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles     structure with handles and user data (see GUIDATA)
% varargin    command line arguments to AdjustRules (see VARARGIN)

% Choose default command line output for AdjustRules
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes AdjustRules wait for user response (see UIRESUME)
% uiwait(handles.figure1);
disp('    Adjust Rules section started');
load ('ARuletempqpzm10a'); %contains current rule number
set(handles.title,'userdata',Rule);
currentrule = Rule;
disp(['Rule ', Rule]); % close('tempqpzm10a'); %doesn't work
load ('AdjustRtempqpzm10'); % load the file saved by the 'datainput3'
% gui, which contains the current SNN settings
set(handles.mydata,'Userdata',SNN); % save SNN to mydata userdata
displaygraph(handles, currentrule); % display text rule on the "graph"
% *****
% not needed if displaying only one rule at a time
% display an appropriate number of radio buttons i.e. 1 button per rule
% and check the current rule number to switch the 'current' button on
if (SNN.NumberRules >= 1) set(handles.m1,'visible','on'); end
if (SNN.NumberRules >= 2) set(handles.m2,'visible','on'); end
if (SNN.NumberRules >= 3) set(handles.radiobutton3,'visible','on'); end
if (SNN.NumberRules >= 4) set(handles.radiobutton4,'visible','on'); end
if (SNN.NumberRules >= 5) set(handles.radiobutton5,'visible','on'); end
if (SNN.NumberRules >= 6) set(handles.radiobutton6,'visible','on'); end
if (SNN.NumberRules >= 7) set(handles.radiobutton7,'visible','on'); end
if (SNN.NumberRules >= 8) set(handles.radiobutton8,'visible','on'); end
if (SNN.NumberRules >= 9) set(handles.radiobutton9,'visible','on'); end
if (SNN.NumberRules >= 10) set(handles.radiobutton10,'visible','on'); end
if (SNN.NumberRules >= 11) set(handles.radiobutton11,'visible','on'); end

if (Rule == 1) set(handles.m1,'value',1); end
if (Rule == 2) set(handles.m2,'value',1); end
if (Rule == 3) set(handles.radiobutton3,'value',1); end
if (Rule == 4) set(handles.radiobutton4,'value',1); end
if (Rule == 5) set(handles.radiobutton5,'value',1); end
if (Rule == 6) set(handles.radiobutton6,'value',1); end
if (Rule == 7) set(handles.radiobutton7,'value',1); end
if (Rule == 8) set(handles.radiobutton8,'value',1); end
if (Rule == 9) set(handles.radiobutton9,'value',1); end
if (Rule == 10) set(handles.radiobutton10,'value',1); end
if (Rule == 11) set(handles.radiobutton11,'value',1); end

```

```

% *****

% --- Outputs from this function are returned to the command line.
function varargout = AdjustRules_OutputFcn(hObject, eventdata, handles)
% varargout cell array for returning output args (see VARARGOUT);
% hObject handle to figure
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

% *****radio button callbacks*****
% --- Executes on button press in m1.
function m1_Callback(hObject, eventdata, handles)
%make sure radio buttons are mutually exclusive
off = [handles.m2,handles radiobutton3,handles radiobutton4,...
        handles radiobutton5,handles radiobutton6,handles radiobutton7,...
        handles radiobutton8,handles radiobutton9,handles radiobutton10,handles radiobutton11];
mutual_exclude(off)
set(handles.title,'userdata',1); %indicates current radio button
displaygraph(handles, 1);

function m2_Callback(hObject, eventdata, handles)
%make sure radio buttons are mutually exclusive
off = [handles.m1,handles radiobutton3,handles radiobutton4,...
        handles radiobutton5,handles radiobutton6,handles radiobutton7,...
        handles radiobutton8,handles radiobutton9,handles radiobutton10,handles radiobutton11];
mutual_exclude(off)
set(handles.title,'userdata',2); %indicates current radio button
displaygraph(handles, 2);

% --- Executes on button press in radiobutton3.
function radiobutton3_Callback(hObject, eventdata, handles)
%make sure radio buttons are mutually exclusive
off = [handles.m1,handles.m2,handles radiobutton4,...
        handles radiobutton5,handles radiobutton6,handles radiobutton7,...
        handles radiobutton8,handles radiobutton9,handles radiobutton10,handles radiobutton11];
mutual_exclude(off)
set(handles.title,'userdata',3); %indicates current radio button
displaygraph(handles, 3);

% --- Executes on button press in radiobutton4.
function radiobutton4_Callback(hObject, eventdata, handles)
%make sure radio buttons are mutually exclusive
off = [handles.m1,handles.m2,handles radiobutton3,...
        handles radiobutton5,handles radiobutton6,handles radiobutton7,...
        handles radiobutton8,handles radiobutton9,handles radiobutton10,handles radiobutton11];
mutual_exclude(off)
set(handles.title,'userdata',4); %indicates current radio button
displaygraph(handles, 4);

% --- Executes on button press in radiobutton5.
function radiobutton5_Callback(hObject, eventdata, handles)
%make sure radio buttons are mutually exclusive
off = [handles.m1,handles.m2,handles radiobutton3,handles radiobutton4,...
        handles radiobutton6,handles radiobutton7,...
        handles radiobutton8,handles radiobutton9,handles radiobutton10,handles radiobutton11];
mutual_exclude(off)

```

```

set(handles.title,'userdata',5); %indicates current radio button
displaygraph(handles, 5);

% --- Executes on button press in radiobutton6.
function radiobutton6_Callback(hObject, eventdata, handles)
%make sure radio buttons are mutually exclusive
off = [handles.m1,handles.m2,handles.radiobutton3,handles.radiobutton4,...
        handles.radiobutton5,handles.radiobutton7,...
        handles.radiobutton8,handles.radiobutton9,handles.radiobutton10,handles.radiobutton11];
mutual_exclude(off)
set(handles.title,'userdata',6); %indicates current radio button
displaygraph(handles, 6);

% --- Executes on button press in radiobutton7.
function radiobutton7_Callback(hObject, eventdata, handles)
%make sure radio buttons are mutually exclusive
off = [handles.m1,handles.m2,handles.radiobutton3,handles.radiobutton4,...
        handles.radiobutton5,handles.radiobutton6,...
        handles.radiobutton8,handles.radiobutton9,handles.radiobutton10,handles.radiobutton11];
mutual_exclude(off)
set(handles.title,'userdata',7); %indicates current radio button
displaygraph(handles, 7);

% --- Executes on button press in radiobutton8.
function radiobutton8_Callback(hObject, eventdata, handles)
%make sure radio buttons are mutually exclusive
off = [handles.m1,handles.m2,handles.radiobutton3,handles.radiobutton4,...
        handles.radiobutton5,handles.radiobutton6,handles.radiobutton7,...
        handles.radiobutton9];
mutual_exclude(off)
set(handles.title,'userdata',8); %indicates current radio button
displaygraph(handles, 8);

% --- Executes on button press in radiobutton9.
function radiobutton9_Callback(hObject, eventdata, handles)
%make sure radio buttons are mutually exclusive
off = [handles.m1,handles.m2,handles.radiobutton3,handles.radiobutton4,...
        handles.radiobutton5,handles.radiobutton6,handles.radiobutton7,...
        handles.radiobutton8,handles.radiobutton10,handles.radiobutton11];
mutual_exclude(off)
set(handles.title,'userdata',9); %indicates current radio button
displaygraph(handles, 9);

% --- Executes on button press in radiobutton10.
function radiobutton10_Callback(hObject, eventdata, handles)
%make sure radio buttons are mutually exclusive
off = [handles.m1,handles.m2,handles.radiobutton3,handles.radiobutton4,...
        handles.radiobutton5,handles.radiobutton6,handles.radiobutton7,...
        handles.radiobutton8,handles.radiobutton9,handles.radiobutton11];
mutual_exclude(off)
set(handles.title,'userdata',10); %indicates current radio button
displaygraph(handles, 10);

% --- Executes on button press in radiobutton11.
function radiobutton11_Callback(hObject, eventdata, handles)
%make sure radio buttons are mutually exclusive
off = [handles.m1,handles.m2,handles.radiobutton3,handles.radiobutton4,...
        handles.radiobutton5,handles.radiobutton6,handles.radiobutton7,...
        handles.radiobutton8,handles.radiobutton9,handles.radiobutton10];
mutual_exclude(off)

```

```

set(handles.title,'userdata',11); %indicates current radio button
displaygraph(handles, 11);

% -----update graph-----
function accept_Callback(hObject, eventdata, handles)
newrule = get(handles.ruledisplay,'string'); % get updated rule
currentrulenum = get(handles.title,'userdata'); %get current rule no.
if (currentrulenum >=0 | currentrulenum <=9)
    disp('Current rule is a valid number');
else
    disp('Current rule is a character');
end
SNN = get(handles.mydata,'userdata'); %make SNN 'local'
oldrules = SNN.Rule; % get old rule from SNN
UpdatedRules = updatrules(oldrules,newrule,currentrulenum,SNN.NumberRules);%
insert updated rule
SNN.Rule = UpdatedRules; % put update back into SNN
set(handles.mydata,'userdata',SNN); %save SNN to make it 'global' again
%check to see if output is within defuzzymfdef values
displaygraph(handles, currentrulenum);

% ---subfunction-----
function mutual_exclude(off)
set(off,'Value',0)

function r = getCurrentButton(buttons)
for i = 1:length(buttons)
    if (get(buttons(i),'Value') == 0)
        r = buttons(i);
    end
end

% --- Executes during object creation, after setting all properties.
function ruledisplay_CreateFcn(hObject, eventdata, handles)

if ispc
set(hObject,'BackgroundColor','white');
else
set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
%-----
function ruledisplay_Callback(hObject, eventdata, handles)
% nothing specific done until 'accept' button is pressed
%-----
function closebutton_Callback(hObject, eventdata, handles)

SNN = get(handles.mydata,'userdata'); % make SNN local
save('AdjustRtempqpzm10','SNN'); %save updated settings to the file
% to be read in by 'datainput3' GUI
disp('    AdjustRules section closed');
close;

%-----
function displayfuzzyValues(handles, fuzzy)
% display the fuzzy membership values
[x,num] = size (fuzzy);
mvals="";
for i = 1:num
    mvals = [mvals;'M',num2str(i),' ='];

```

```

end
set(handles.mvalues,'string',mvals);
set(handles.fuzzyValues,'string',fuzzy); % display fuzzy membership values
%-----
function displaygraph(handles, currentrule)
% given handles and currentrule (number),
% displays the currentrule in editbox and graph

SNN = get(handles.mydata,'userdata');
tline = getrule(SNN.Rule,currentrule,SNN.NumberRules); %get rule no. from SNN.Rule
tline2 = tline;
set(handles.title,'string',["Displaying Rule Number ", num2str(currentrule)]); % display window
title
set(handles.ruledisplay,'string',tline); % display the rule itself

if (SNN.IsWindowFilter ~= 0)
axes(handles.axes2); % set current axes to 'axes2'
% *****clear the graph*****
plot(1);
set(handles.axes2,'color',[1 1 0],...
    'xlim',[ 0 SNN.WFHSize],'ylim',[ 0 SNN.WFVSize],...
    'xgrid','on','ygrid','on','ydir','reverse',...
    'xtick',[1,2,3,4,5,6,7,8,9,10,11,12],...
    'ytick',[1,2,3,4,5,6,7,8,9,10,11,12]);
% put in numbers
boxnum = 0;
% adjust text variables
ifs = 18 - SNN.WFHSize;%'I' font size
rfs = ifs + 2;
if (SNN.WFHSize < 7)
    if (SNN.WFHSize < 3)
        valign = 0.6;
    else valign = 0.7;
    end
else
    valign = 0.8;
end
% put in boxnumbers
for x = 1:SNN.WFVSize
    for y = 1:SNN.WFHSize
        boxnum = boxnum + 1;
        text(y - 0.6 ,x - 0.6 ,['I',num2str(boxnum),':'],...
            'FontSize',ifs,'HorizontalAlignment','left','EraseMode','normal');
    end
end

% must alter numdown to numupdown, to represent the fact the the filter
% window reads left to right, top to bottom, but the graph is displayed
% left to right, bottom to top.....
%*****GRAPH: display current rule*****
[currentword, tline] = ExtractWord(tline); % get 'if' word of line
[currentword, tline] = ExtractWord(tline); %get 'first' rule-part
while ~isempty(tline)
    switch currentword
        case 'Then'
            [result, tline] = ExtractWord(tline);%get resultpart of rule
            set(handles.outputEq,'string',result); % display result part
            break
        otherwise %read the rule
            [cellnum, ruletype , op] = SplitWord(currentword);

```

```

[pos,i] = ExtractNumber(cellnum); % e.g. i = 'l', pos = 6
numdown = fix(pos/SNN.WFHSize);
numacross = mod(pos, SNN.WFHSize);
if numacross == 0 %ie at the end of the row
    numacross = SNN.WFHSize;
    numdown = numdown - 1;
end
% position text within the rectangle
text(numacross - 0.6 ,numdown + valign ,ruletype,...
'FontSize',rfs,'HorizontalAlignment','left','EraseMode','normal')
end %switch
[currentword, tline] = ExtractWord(tline); % get next word of line
end %while

% now display text rule in greyscale representation
% note 0 = black, 255 = white, 30 = green
% create a default image 'gsi' of the same size as filterwindow
gsi(SNN.WFVSize,SNN.WFHSize,3) = 0;
gsisafe(SNN.WFVSize,SNN.WFHSize) = 0;
gsi = gsi+256; % set all values to over proper range*****
gsisafe = gsisafe + 256;
axes(handles.axes3); % set current axes to 'axes3'
% *****clear the graph*****
plot(1);
set(handles.axes3,'color',[1 1 0],...
'xlim',[ 0 SNN.WFHSize],'ylim',[ 0 SNN.WFVSize],...
'xgrid','on','ygrid','on',...
'tick',[1,2,3,4,5,6,7,8,9,10,11,12],...
'tick',[1,2,3,4,5,6,7,8,9,10,11,12]);

%*****GRAPH: display current rule in greyscale*****
[currentword, tline2] = ExtractWord(tline2); % get 'if' word of line
[currentword, tline2] = ExtractWord(tline2); %get 'first' rule-part
while ~isempty(tline2)
    switch currentword
        case 'Then'
            [result, tline2] = ExtractWord(tline2); % get result part
            % do nothing yet? display result part of rule ?
            break
        otherwise %read the rule
            [cellnum, ruletype , op] = SplitWord(currentword);
            [pos,i] = ExtractNumber(cellnum); % e.g. i = 'l', pos = 6
            [MFDEFpos,m] = ExtractNumber(ruletype); %eg m = 'M', MFDEFpos=2
            numdown = ceil(pos/SNN.WFHSize); %rounds up
            numacross = mod(pos, SNN.WFHSize);
            if numacross == 0 %ie at the end of the row
                numacross = SNN.WFHSize;
            end
            intensity = SNN.FuzzyMFDEF(MFDEFpos); % read the intensity
            gsi(numdown,numacross,1) = intensity; % create image map
            gsi(numdown,numacross,2) = intensity;
            gsi(numdown,numacross,3) = intensity;
            gsisafe(numdown,numacross) = intensity*255;
        end %switch
    [currentword, tline2] = ExtractWord(tline2); % get next word of line
end %while

% display the built up image map (array with values from 0 - 255)
%image(gsi);
%colormap(gray(256));

```

```

% display 'x' where there is no rule
horixadjust = SNN.WFHSize * 0.04;
for x = 1:SNN.WFHSize
    for y = 1:SNN.WFVSize
        if gsi(y,x,1) == 256 % ie. if no rule has been written in this pos
            %text(x - horixadjust,y,'x','FontSize',rfs+4,...
            % 'HorizontalAlignment','left','EraseMode','normal');
            gsi(y,x,1) = 0.30;
            gsi(y,x,2) = 0.60;
            gsi(y,x,3) = 0.90; % turquoise / blue colour
            gsisafe(y,x) = 0;
        end
    end
end
image(gsi);
end %first 'if'

% --- Executes on mouse press over axes background.
function axes2_ButtonDownFcn(hObject, eventdata, handles)

% --- Executes on mouse press over figure background, over a disabled or
% --- inactive control, or over an axes background.
function figure1_WindowButtonDownFcn(hObject, eventdata, handles)

% Allows drag select or single select

% Hint: get(hObject,'Value') returns toggle state of dragselectbutton
SNN = get(handles.mydata,'UserData'); % make local SNN = "global" SNN

%k = waitforbuttonpress;
point1 = get(gca,'CurrentPoint'); % button down detected
finalRect = rbbox; % return figure units
point2 = get(gca,'CurrentPoint'); % button up detected
point1 = point1(1,1:2) % extract x and y
point2 = point2(1,1:2)
p1 = min(point1,point2); % calculate locations
offset = abs(point1-point2); % and dimensions
% x = [p1(1) p1(1)+offset(1) p1(1)+offset(1) p1(1) p1(1)];
% y = [p1(2) p1(2) p1(2)+offset(2) p1(2)+offset(2) p1(2)];

% now make sure x1<x2 and y1<y2
if point1(1) < point2(1)
    x1 = ceil(point1(1));
    x2 = ceil(point2(1));
else
    x1 = ceil(point2(1));
    x2 = ceil(point1(1));
end
if point1(2) < point2(2)
    y1 = ceil(point1(2));
    y2 = ceil(point2(2));
else
    y1 = ceil(point2(2));
    y2 = ceil(point1(2));
end

%if buttonclick not in graph
if (x1 <= 0) | (y1 <= 0) | (x1 > SNN.WFHSize) | (y1 > SNN.WFVSize) | (x2 <= 0) | (y2 <= 0) |
(x2 > SNN.WFHSize) | (y2 > SNN.WFVSize)

```



```

    disp('not in graph'); % do nothing
else % update SNN.Rule and redisplay both graph and editbox
    newvalue = num2str(get(handles.edit2,'value')) % new rule eg. 'M2'
    newvalue = ['M',newvalue];
    for x = x1:x2
        for y = y1:y2
            % update SNN.Rule and redisplay both graph and editbox
            foundtarget = 'false';
            targetpos = (ceil(y)-1)*SNN.WFHSsize + ceil(x); % target position, eg. '8' as in 'I8'
            disp(['x = ', num2str(ceil(x)),', y = ',num2str(ceil(y))]);
            currentrule = get(handles.title,'userdata'); %ie. current rule number is...
            tline = getrule(SNN.Rule,currentrule,SNN.NumberRules); % get current rule from
SNN.Rule
            [currentword, tline] = ExtractWord(tline); % get 'if' word of line
            firstpart = currentword; %firstpart is all parts algorithm has "read" over
            [currentword, tline] = ExtractWord(tline); %get 'first' rule-part, (eg. 'I1=M3')

            while ~strcmp(currentword,'Then') % while we haven't reached 'Then' word
                [cellnum, ruletype, op] = SplitWord(currentword);
                [pos,i] = ExtractNumber(cellnum); % e.g. i = 'I', pos = 6
                if pos == targetpos % if we encounter the target position, update the rule
                    tline = [firstpart,' ',num2str(targetpos),'=',newvalue,tline]
                    SNN.Rule = updatrules(SNN.Rule,tline,currentrule,SNN.NumberRules);
                    % update rule ....
                    %disp(['WDB target, pos ',num2str(pos)]);
                    foundtarget = 'true';
                    set(handles.mydata,'userdata',SNN);
                    displaygraph(handles,currentrule); % redraw the graph
                    break
                    % note. we could put in "else if pos > targetpos" here to insert rule 'in order',
but....
                    % will assume that the rule parts 'I1 I2 etc' may NOT be in order
                end % end if
                firstpart = [firstpart,' ',currentword]; % currentword was not the target so add it on to
firstpart
            [currentword, tline] = ExtractWord(tline); % get next word of line
            end %while

            if strcmp(foundtarget,'false') % if clicked square currently has no 'rule' in it
                % add extra rule part to the beginning of currentrule (ie. just after 'If' word)
                tline = getrule(SNN.Rule,currentrule,SNN.NumberRules); % get rule number from
SNN.Rule
                [currentword, tline] = ExtractWord(tline); % get 'if' word of line
                tline = [currentword,' ',num2str(targetpos),'=',newvalue,tline] % build currentrule
                SNN.Rule = updatrules(SNN.Rule,tline,currentrule,SNN.NumberRules); % build
rule-line
                set(handles.mydata,'userdata',SNN);
                displaygraph(handles,currentrule);
            end % if
        end % for y
    end % for x
end % if else

% --- Executes during object creation, after setting all properties.
function edit2_CreateFcn(hObject, eventdata, handles)

if ispc
    set(hObject,'BackgroundColor','white');
else

```

```

    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function edit2_Callback(hObject, eventdata, handles)
% Hints: get(hObject,'String') returns contents of edit2 as text
%       str2double(get(hObject,'String')) returns contents of edit2 as a double
SNN = get(handles.mydata,'UserData'); % make local variable SNN = "global" SNN
if (get(hObject,'Value') > SNN.FuzzyMF)
    msgbox('The value you have selected exceeds the number of FuzzyMF
rules','Error','warn');
    uiwait;
    set(hObject,'Value',1); % ie. reset it to 'M1'
end

```

Functions for 'BatchProcess' GUI

```

function varargout = batchprocess(varargin)

if isstruct(varargin{1}) % LAUNCH GUI - Check to make sure that first variable is a structure

    fig = openfig(mfilename,'reuse');

    % Use system color scheme for figure:
    set(fig,'Color',get(0,'defaultUicontrolBackgroundColor'));

    % Generate a structure of handles to pass to callbacks, and store it.
    handles = guihandles(fig);
    guidata(fig, handles);

% -----opening code-----
disp(' Batch Processing section started');
load ('tempqpm10');
set(handles.mydata,'userdata',SNN);
set(handles.timage,'userdata',Timage);
set(handles.pimage,'userdata',Pimage);
set(handles.ivinfo,'userdata',Iv);
set(handles.tvinfo,'userdata',Tv);

set(handles.pushbutton3,'visible','off');
if SNN.IsWindowFilter == 1
    % show input and target images
    axes(handles.baxes1); % make this axes current
    set(handles.baxes1,'ydir','normal');
    image(Pimage);
    colormap(gray(256));
    axes(handles.baxes2); % make this axes current
    set(handles.baxes2,'ydir','normal');
    image(Timage);
    colormap(gray(256));
    set(handles.nonwindowfilter,'visible','off');
    set(handles.classifypollen,'visible','off');
else
    set(handles.baxes1,'visible','off');
    set(handles.baxes2,'visible','off');
    set(handles.norulesbutton,'visible','off');
    set(handles.random,'visible','off');
    set(handles.togglebutton2,'visible','off');
    set(handles.norulesbutton,'visible','off');

```

```

    set(handles.varyrulenodes,'visible','off');
    set(handles.comborun,'visible','off');
    set(handles.runbutton,'visible','off');
end

% show batch run details
%set(handles.timage,'string',['Network will run ',get(handles.numofruns,'string'),' times']);
% set current option to 'nothing'
set(handles.runbutton,'userdata','no choice');
%-----end opening code-----

    if nargout > 0
        varargout{1} = fig;
    end

elseif ischar(varargin{1}) % INVOKE NAMED SUBFUNCTION OR CALLBACK

    try
        if (nargout)
            [varargout{1:nargout}] = feval(varargin{:}); % FEVAL switchyard
        else
            feval(varargin{:}); % FEVAL switchyard
        end
    catch
        disp(lasterr);
    end
end

% --- Executes just before batchprocess is made visible.
function batchprocess_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to batchprocess (see VARARGIN)

% Choose default command line output for batchprocess
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes batchprocess wait for user response (see UIRESUME)
% uiwait(handles.figure1);
disp('  Batch Processing section started');
load ('tempqpzm10');
set(handles.mydata,'userdata',SNN);
set(handles.timage,'userdata',Timage);
set(handles.pimage,'userdata',Pimage);
set(handles.ivinfo,'userdata',lv);
set(handles.tvinfo,'userdata',Tv);

% show input and target images
axes(handles.baxes1); % make this axes current
set(handles.baxes1,'ydir','normal');
image(Pimage);
colormap(gray(256));
axes(handles.baxes2); % make this axes current
set(handles.baxes2,'ydir','normal');

```

```

image(Timage);
colormap(gray(256));

% show batch run details
%set(handles.timage,'string',['Network will run ',get(handles.numofruns,'string'),' times']);
% set current option to 'nothing'
set(handles.runbutton,'userdata','no choice');

% --- Outputs from this function are returned to the command line.
function varargout = batchprocess_OutputFcn(hObject, eventdata, handles)

varargout{1} = handles.output;

% --- Executes during object creation, after setting all properties.
function numofruns_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function numofruns_Callback(hObject, eventdata, handles)

% Hints: get(hObject,'String') returns contents of numofruns as text
%      str2double(get(hObject,'String')) returns contents of numofruns as a double
if (get(hObject,'String') >=1) & (get(hObject,'String') <=1000)

    % do nothing, valid entry
else
    msgbox('Please enter a number between 1 and 1000','Number of Runs','warn');
    uiwait
    set(numofruns,'string',10);
end

function mutual_exclude(off)
set(off,'Value',0)

function displaydetails(handles,repeat,epochs,explain)
set(handles.timage,'string',['Network will run ',repeat,' times,']);
set(handles.ivinfo,'string',['with a ',epochs,' number of epochs.']);
set(handles.pimage,'string',explain);
%set(handles.tvinfo,'string',['and a ',qfactor,' quality factor.']);

% --- Executes on button press in random.
function random_Callback(hObject, eventdata, handles)

% Hint: get(hObject,'Value') returns toggle state of random
set(handles.runbutton,'userdata','random');
%set(handles.btext3,'visible','on');
%set(handles.numofruns,'visible','on');
mutual_exclude([handles.togglebutton2,handles.norulesbutton,handles.varyrulenodes]);

% get batch-run data
rval = inputdlg('Run the network how many times?',...
    'Multiplication factor for random number generator',...

```

```

    'Quality Factor (enter a number between 0.1 and 1}',...
    'Setting Random parameters',[1 1 1],...
    {'3','10','0.5'});
rvalues.repeat = str2num(cell2mat(rval(1)));
rvalues.mulfactor = str2num(cell2mat(rval(2)));
rvalues.qfactor = str2num(cell2mat(rval(3))) %displays
rvalues.type = 'Random';
set(handles.random,'userdata',rvalues);
displaydetails(handles,cell2mat(rval(1)),'constant',...
    'and other variables constant');

% --- Executes on button press in togglebutton2 (ie. Epochs).
function togglebutton2_Callback(hObject, eventdata, handles)

% Hint: get(hObject,'Value') returns toggle state of togglebutton2
set(handles.runbutton,'userdata','epochs');
mutual_exclude([handles.random,handles.norulesbutton,handles.varyrulenodes]);
%set(handles.btext3,'visible','off');
%set(handles.numofruns,'visible','off');
% get batch-run data
epochrange = inputdlg({'Start epochs at what value?',...
    'Increase epochs by how many each run?',...
    'For each run preat experiment how many times?',...
    'Increase the epochs how many times?',...
    'Setting Epoch parameters',[1 1 1 1],...
    {'50','10','1','1'}});
epochs.startnum = str2num(cell2mat(epochrange(1)));
epochs.step = str2num(cell2mat(epochrange(2)));
epochs.numtimes = str2num(cell2mat(epochrange(3)));
epochs.repeat = str2num(cell2mat(epochrange(4))) % displays
epochs.type = 'Varying epochs';
set(handles.togglebutton2,'userdata',epochs);
displaydetails(handles,num2str(epochs.repeat * epochs.numtimes),'varying',...
    'and other variables constant');

% --- Executes on button press in norulesbutton.
function norulesbutton_Callback(hObject, eventdata, handles)
set(handles.runbutton,'userdata','norulescombo');
mutual_exclude([handles.random,handles.togglebutton2,handles.varyrulenodes]);
% get batch-run data
vnodes = inputdlg({'Start rule nodes at what value?',...
    'Stop rule nodes at what value?',...
    'Start epochs at what value?',...
    'Epochs step?',...
    'Increase epochs how many times?',...
    'Repeat experiment how many times for each setting?',...
    'Setting no-rules-combo parameters',[1 1 1 1 1 1],...
    {'5','15','50','50','10','10'}});
nodes.startnrn = str2num(cell2mat(vnodes(1)));
nodes.stopnrn = str2num(cell2mat(vnodes(2)));
nodes.startepochs = str2num(cell2mat(vnodes(3)));
nodes.stepepochs = str2num(cell2mat(vnodes(4)));
nodes.increaseepochs = str2num(cell2mat(vnodes(5)));
nodes.repeat = str2num(cell2mat(vnodes(6))) % displays
nodes.type = 'combo';
set(handles.varyrulenodes,'userdata',nodes);
displaydetails(handles,num2str(nodes.increaseepochs * nodes.repeat*(nodes.stopnrn -
nodes.startnrn + 1)),...
    'varying','no rules combo');

```

```

% -----
function varargout = varyrulenodes_Callback(h, eventdata, handles, varargin)
set(handles.runbutton,'userdata','varyrulenodes');
mutual_exclude([handles.random,handles.togglebutton2,handles.norulesbutton]);
% get batch-run data
vnodes = inputdlg('Start rule nodes at what value?',...
    'End rule nodes at what value?',...
    'Run the network how many times for each rule node?',...
    'Setting Rule Node parameters',[1 1 1],...
    {'5','15','10'});
nodes.startnum = str2num(cell2mat(vnodes(1)));
nodes.stop = str2num(cell2mat(vnodes(2)));
nodes.repeat = str2num(cell2mat(vnodes(3))) % displays
nodes.type = 'varying rule nodes';
set(handles.varyrulenodes,'userdata',nodes);
displaydetails(handles,num2str(nodes.repeat*(nodes.stop - nodes.startnum + 1)),...
    'constant','and with a varying number of rule nodes');

% -----
function varargout = comborun_Callback(h, eventdata, handles, varargin)

set(handles.runbutton,'userdata','comborun');
mutual_exclude([handles.random,handles.togglebutton2,handles.norulesbutton]);
% get batch-run data
vnodes = inputdlg('Start rule nodes at what value?',...
    'Stop rule nodes at what value?',...
    'Start epochs at what value?',...
    'Epochs step?',...
    'Increase epochs how many times?',...
    'Repeat experiment how many times for each setting?',...
    'Setting Combo parameters',[1 1 1 1 1 1],...
    {'5','15','50','50','10','10'});
nodes.startn = str2num(cell2mat(vnodes(1)));
nodes.stopr = str2num(cell2mat(vnodes(2)));
nodes.startepochs = str2num(cell2mat(vnodes(3)));
nodes.steepochs = str2num(cell2mat(vnodes(4)));
nodes.increaseepochs = str2num(cell2mat(vnodes(5)));
nodes.repeat = str2num(cell2mat(vnodes(6))) % displays
nodes.type = 'combo';
set(handles.varyrulenodes,'userdata',nodes);
displaydetails(handles,num2str(nodes.increaseepochs * nodes.repeat*(nodes.stopr -
nodes.startn + 1)),...
    'varying','combo');

% -----
% --- Executes on button press in runbutton.
function runbutton_Callback(hObject, eventdata, handles)

% load in data
SNN = get(handles.mydata,'UserData'); % make local variable SNN = "global" SNN
Timage = get(handles.timage,'userdata');
Pimage = get(handles.pimage,'userdata');
Iv = get(handles.ivinfo,'userdata');
Tv = get(handles.tvinfo,'userdata');

% save 'mat' file containing SNN data
savefilename = get(handles.runname,'string');
[fname,fpath] = uiputfile(['batchsaves\',savefilename],'Choose where to save batch runs');

```

```

% initialize error calculation variables;
falsepositives = 0;
falsenegatives = 0;
percenterror = 0;
RMS = 0; fp=0; fn=0; err=0; percerr=0;

figcounter = 0; % variable to count display number on output graphs
ocounter = 0; % used to increment the random state variable

choice = get(handles.runbutton,'userdata');
switch choice
case 'random' % Random button was chosen
    disp('Began parsing the Network');
    SNN = parse3(SNN);

    rvalues = get(handles.random,'userdata');
    multifactor = rvalues.multifactor;
    numruns = rvalues.repeat;
    qfactor = rvalues.qfactor;
    totalpe = 0;
    for i = 1:numruns
        disp(['Running batch number ',num2str(i),' of ',num2str(numruns)]);
        F = CreateFuNN2(SNN); %create the FuNN
        %% load weight into FuNN
        randn('state',i*multifactor); % *****
        R = SNN.RuleMatrix*qfactor;
        R = R + randn(size(SNN.RuleMatrix))*0.1; %88
        A = SNN.ActionMatrix*qfactor;
        A = A + randn(size(SNN.ActionMatrix))*0.1;%88
        F = ModifyRuleMatrix(F, R);
        F = ModifyActionMatrix(F, A);
        F = ModifyEpochs(F, SNN.Epochs);

        %% train NN using input/target vectors
        [F, RulePerf] = TrainFuNN(F, Iv, Tv);

        %% process input image using trained FuNN
        RuleO = ProcessImageUsingFuNN(Pimage, F);

        %% crop target image to same size as output image for error calculation purposes
        NT = CropTargetToOutput(Timage, RuleO, [F.WFVSize F.WFHSize]);

        % save performance data and output graph in SNN variable
        SNN.RuleO = RuleO; % output graph
        SNN.RulePerf = RulePerf;
        %SNN.F = F; % save trained network
        SNN.RMS = sqrt(mse(NT - RuleO)); % save RMS data

        %% save work area
        if (get(handles.savenndata,'value') == 1)
            save([fpath,fname,num2str(i),'.mat'],'SNN');
        end

        if (get(handles.displayoutput,'value') == 1)
            displayGraphOutput(handles,RuleO,i);
        end

        % error calculations iris edge --[fp, fn, e]=calcerror2(Target, Output)

```

```

[fp fn err percerr] = calcerror2( NT, RuleO);
falsepositives(i) = fp
falsenegatives(i) = fn
percenterror(i) = percerr
RMS(i) = sqrt(mse(NT - RuleO))
totalpe = totalpe + percerr;

end % for
avgpe = totalpe / numruns;
settingsdata = epochs;

case 'epochs' % Epochs button was chosen
disp('Begun parsing the Network');
SNN = parse3(SNN);
epochs = get(handles.togglebutton2,'userdata');
for i = 1:epochs.repeat
    ocounter = 0; % reset for each new combination
    totalpe = 0;
    for j = 1:epochs.numtimes
        currentepochs = ((i-1)*epochs.step) + epochs.startnum;
        disp(['Running batch number ',num2str(i),' of ',num2str(epochs.repeat),' with ',num2str(currentepochs),' epochs']);
        F = CreateFuNN2(SNN); %create the FuNN
        %% load weight into FuNN
        randn('state',ocounter);
        R = SNN.RuleMatrix*0.5;
        R = R + randn(size(SNN.RuleMatrix))*0.1;
        A = SNN.ActionMatrix*0.5;
        A = A + randn(size(SNN.ActionMatrix))*0.1;
        F = ModifyRuleMatrix(F, R);
        F = ModifyActionMatrix(F, A);
        % next line changes the number of epochs
        F = ModifyEpochs(F,currentepochs);

        %% train NN using input/target vectors
        [F, RulePerf] = TrainFuNN(F, Iv, Tv);

        %% process input image using trained FuNN
        RuleO = ProcessImageUsingFuNN(Pimage, F);

        %% crop target image to same size as output image for error calculation purposes
        NT = CropTargetToOutput(Timage, RuleO, [F.WFVSize F.WFHSize]);

        % save performance data and output graph in SNN variable
        SNN.RuleO = RuleO;
        SNN.RulePerf = RulePerf;
        %SNN.F = F; % save trained network
        SNN.RMS(i,j) = sqrt(mse(NT - RuleO)); % save RMS data

        % error calculations iris edge --[fp, fn, e]=calcerror2(Target, Output)
        [fp fn err percerr] = calcerror2( NT, RuleO);
        falsepositives(i,j) = fp
        falsenegatives(i,j) = fn
        percenterror(i,j) = percerr
        RMS(i,j) = sqrt(mse(NT - RuleO))
        totalpe = totalpe + percerr;

        ocounter = ocounter+1;
        %% save work area

```



```

if (get(handles.savenndata,'value') == 1)
    save([fpath,fname,num2str(i),'.mat'],'SNN');
end

if (get(handles.displayoutput,'value') == 1)
    figcounter = figcounter + 1;
    displayGraphOutput(handles,RuleO,figcounter);
end

end % for j
avgpe(i) = totalpe / epochs.numtimes;
end % for i
settingsdata = epochs;

case 'varyrulenodes' % vary rule nodes button was chosen
    figcounter = 0; ocounter=0;
    nodes = get(handles.varyrulenodes,'userdata');
    for i = nodes.startnum:nodes.stop
        ocounter = 0; % reset for each new combination of rule nodes
        totalpe = 0;
        for j = 1:nodes.repeat
            %currentnumnodes = ((i-1)*1) + nodes.startnum;
            % update number of rules nodes on each run
            SNN.TotalRuleNodes = i; %changed here
            % must parse the network here after SNN has been changed
            disp('Parsing the Network');
            SNN = parse3(SNN);

            disp(['Running batch no. ',num2str(i - nodes.startnum + 1),' of ',num2str(nodes.stop -
nodes.startnum + 1)]);
            disp(['Repetition no.',num2str(j),' of ',num2str(nodes.repeat),' with ',num2str(i),' rule
nodes']);
            F = CreateFuNN2(SNN); %create the FuNN
            %% load weight into FuNN
            % must use figcounter to avoid repeating experiment unnecessarily
            randn('state',ocounter);
            R = SNN.RuleMatrix*0.5;
            R = R + randn(size(SNN.RuleMatrix))*0.1;
            A = SNN.ActionMatrix*0.5;
            A = A + randn(size(SNN.ActionMatrix))*0.1;
            F = ModifyRuleMatrix(F, R);
            F = ModifyActionMatrix(F, A);
            % next line changes the number of epochs
            F = ModifyEpochs(F,SNN.Epochs);

            %% train NN using input/target vectors
            [F, RulePerf] = TrainFuNN(F, Iv, Tv);

            %% process input image using trained FuNN
            RuleO = ProcessImageUsingFuNN(Pimage, F);

            %% crop target image to same size as output image for error calculation purposes
            NT = CropTargetToOutput(Timage, RuleO, [F.WFVSize F.WFHSize]);

            % save performance data and output graph in SNN variable
            SNN.RuleO = RuleO;
            SNN.RulePerf = RulePerf;

```

```

%SNN.F = F; % save trained network
SNN.RMS = sqrt(mse(NT - RuleO)); % save RMS data

% error calculations
[fp fn err percerr] = calcerror2( NT, RuleO);
falsepositives((i - nodes.startnum + 1),j) = fp
falsenegatives((i - nodes.startnum + 1),j) = fn
percenterror((i - nodes.startnum + 1),j) = percerr
RMS((i - nodes.startnum + 1),j) = sqrt(mse(NT - RuleO))
totalpe = totalpe + percerr;

ocounter = ocounter+1;
%% save work area
if (get(handles.savenndata,'value') == 1)
    save([fpath,fname,num2str(i),'.mat'],'SNN');
end

if (get(handles.displayoutput,'value') == 1)
    figcounter = figcounter + 1;
    displayGraphOutput(handles,RuleO,figcounter);
end

end % for j
avgpe(i) = totalpe / nodes.repeat;
end % for i
settingsdata = nodes;

case 'norulescombo' % vary rule nodes button was chosen
    figcounter = 0; ocounter = 0;
    nodes = get(handles.varyrulenodes,'userdata');
    nodemax = nodes.stoprn - nodes.startrn + 1;

    for i = 1:nodemax
        for ep = 1:nodes.increaseepochs
            ocounter = 0; % resets for eachnew combo of variables
            totalpe = 0;
            for j = 1:nodes.repeat

                SNN.TotalRuleNodes = i + nodes.startrn - 1; %changed here
                currentepochs = nodes.startepochs + ((ep-1)*nodes.steepochs);
                % must parse the network here after SNN has been changed
                disp('Parsing the Network');
                SNN = parse3(SNN);

                disp(['Running with rn ',num2str(i),' of ',num2str(nodemax)]);
                disp(['Running with ep ',num2str(ep),' of ',num2str(nodes.increaseepochs)]);
                disp(['Repetition no.',num2str(j),' of ',num2str(nodes.repeat)]);
                F = CreateFuNN2(SNN); %create the FuNN
                %% load weight into FuNN
                % must use figcounter to avoid repeating experiment unnecessarily
                randn('state',ocounter);

                R = randn(size(SNN.RuleMatrix)); % init matrix with random numbers
                A = randn(size(SNN.ActionMatrix)); % % init matrix with random numbers

                F = ModifyRuleMatrix(F, R);
                F = ModifyActionMatrix(F, A);
                % next line changes the number of epochs
                F = ModifyEpochs(F,currentepochs); % epochs used here
            end
        end
    end
end

```

```

%% train NN using input/target vectors
[F, RulePerf] = TrainFuNN(F, Iv, Tv);

%% process input image using trained FuNN
RuleO = ProcessImageUsingFuNN(Pimage, F);

%% crop target image to same size as output image for error calculation purposes
NT = CropTargetToOutput(Timage, RuleO, [F.WFVSize F.WFHSize]);

% save performance data and output graph in SNN variable
SNN.RuleO = RuleO;
SNN.RulePerf = RulePerf;
%SNN.F = F; % save trained network
SNN.RMS = sqrt(mse(NT - RuleO)); % save RMS data

% error calculations
[fp fn err percerr] = calcerror2( NT, RuleO);
falsepositives(i,ep,j) = fp
falsenegatives(i,ep,j) = fn
percenterror(i,ep,j) = percerr
RMS(i,ep,j) = sqrt(mse(NT - RuleO))
totalpe = totalpe + percerr;

ocounter = ocounter + 1;
%% save work area
if (get(handles.savenndata,'value') == 1)
    save([fpath,fname,num2str(i),'.mat'],'SNN');
end

if (get(handles.displayoutput,'value') == 1)
    figcounter = figcounter + 1;
    displayGraphOutput(handles,RuleO,figcounter);
end

end % for j
avgpe(i,ep) = totalpe / nodes.repeat;
end % for ep
end % for i
settingsdata = nodes;

case 'xxxxx' % no rules
    numruns = get(handles.norulesbutton,'userdata');
    disp('Begun parsing the Network');
    SNN = parse3(SNN);
    totalpe=0;
    for i = 1:numruns.repeat
        disp(['Running batch number ',num2str(i),' of ',num2str(numruns.repeat)]);
        F = CreateFuNN2(SNN); %create the FuNN
        %% load weight into FuNN
        randn('state',j*10); % *****
        %R = SNN.RuleMatrix*0.5;
        %R = R + randn(size(SNN.RuleMatrix))*0.1; %88
        R = randn(size(SNN.RuleMatrix)); % init matrix with random numbers
        %A = SNN.ActionMatrix*0.5;
        %A = A + randn(size(SNN.ActionMatrix))*0.1;%88
        A = randn(size(SNN.ActionMatrix)); % % init matrix with random numbers

        F = ModifyRuleMatrix(F, R);
        F = ModifyActionMatrix(F, A);
    end
end

```

```

F = ModifyEpochs(F, SNN.Epochs);

%% train NN using input/target vectors
[F, RulePerf] = TrainFuNN(F, Iv, Tv);

%% process input image using trained FuNN
RuleO = ProcessImageUsingFuNN(Pimage, F);

%% crop target image to same size as output image for error calculation purposes
NT = CropTargetToOutput(Timage, RuleO, [F.WFVSize F.WFHSize]);

% save performance data and output graph in SNN variable
SNN.RuleO = RuleO;
SNN.RulePerf = RulePerf;
SNN.F = F; % save trained network
SNN.RMS = sqrt(mse(NT - RuleO)); % save RMS data

%% save work area
if (get(handles.savenndata,'value') == 1)
    save([fpath,fname,num2str(i),'.mat'],'SNN');
end

if (get(handles.displayoutput,'value') == 1)
    displayGraphOutput(handles,RuleO,i);
end

% error calculations
[fp fn err percerr] = calcerror2( NT, RuleO);
falsepositives(i) = fp
falsenegatives(i) = fn
percenterror(i) = percerr
RMS(i) = sqrt(mse(NT - RuleO))
totalpe=totalpe + percerr;

end % for
avgpe = totalpe / numruns.repeat
settingsdata = numruns;

case 'comborun' % vary rule nodes button was chosen
    figcounter = 0; ocounter = 0;
    nodes = get(handles.varyrulenodes,'userdata');
    nodemax = nodes.stoprn - nodes.startrn + 1;

    for i = 1:nodemax
        for ep = 1:nodes.increaseepochs
            ocounter = 0; % resets for eachnew combo of variables
            totalpe = 0;
            for j = 1:nodes.repeat

                SNN.TotalRuleNodes = i + nodes.startrn - 1; %changed here
                currentepochs = nodes.startepochs + ((ep-1)*nodes.steepochs);
                % must parse the network here after SNN has been changed
                disp('Parsing the Network');
                SNN = parse3(SNN);

                disp(['Running with rn ',num2str(i),' of ',num2str(nodemax)]);
                disp(['Running with ep ',num2str(ep),' of ',num2str(nodes.increaseepochs)]);
                disp(['Repetition no.',num2str(j),' of ',num2str(nodes.repeat)]);
            end
        end
    end

```

```

F = CreateFuNN2(SNN); %create the FuNN
%% load weight into FuNN
% must use figcounter to avoid repeating experiment unnecessarily
randn('state',ocounter);
R = SNN.RuleMatrix*0.5;
R = R + randn(size(SNN.RuleMatrix))*0.1;
A = SNN.ActionMatrix*0.5;
A = A + randn(size(SNN.ActionMatrix))*0.1;
F = ModifyRuleMatrix(F, R);
F = ModifyActionMatrix(F, A);
% next line changes the number of epochs
F = ModifyEpochs(F,currentepochs); % epochs used here

%% train NN using input/target vectors
[F, RulePerf] = TrainFuNN(F, Iv, Tv);

%% process input image using trained FuNN
RuleO = ProcessImageUsingFuNN(Pimage, F);

%% crop target image to same size as output image for error calculation purposes
NT = CropTargetToOutput(Timage, RuleO, [F.WFVSize F.WFHSize]);

% save performance data and output graph in SNN variable
SNN.RuleO = RuleO;
SNN.RulePerf = RulePerf;
%SNN.F = F; % save trained network
SNN.RMS = sqrt(mse(NT - RuleO)); % save RMS data

% error calculations
[fp fn err percerr] = calcerror2( NT, RuleO);
falsepositives(i,ep,j) = fp
falsenegatives(i,ep,j) = fn
percenterror(i,ep,j) = percerr
RMS(i,ep,j) = sqrt(mse(NT - RuleO))
totalpe = totalpe + percerr;

ocounter = ocounter + 1;
%% save work area
if (get(handles.savenndata,'value') == 1)
    save([fpath,fname,num2str(i),'.mat'],'SNN');
end

if (get(handles.displayoutput,'value') == 1)
    figcounter = figcounter + 1;
    displayGraphOutput(handles,RuleO,figcounter);
end

end % for j
avgpe(i,ep) = totalpe / nodes.repeat;
end % for ep
end % for i
settingsdata = nodes;
otherwise
    msgbox('Please select a run-time option','Error','warn');
    uiwait

end % switch

% display error calculations and the end of processing
if (SNN.IsWindowFilter == 1)

```

```

    disp('here');
    msgbox(['The RMS value was ',num2str(RMS),' with ',[num2str(fp),' false positives,'],[num2str(fn),' false negatives '],[num2str(err),' errors and '],[num2str(percerr),' percent errors'] ],...
    'Error Calculation','help');
    save(['batchsaves\',fname,'Errors'],'RMS','falsepositives','falsenegatives','percenterror','setting sdata','avgpe');
end

% -----
function displayGraphOutput(handles,RuleO,i)
    %% create new window to display output image
    output_handle = figure('name','Output'); %not needed

    %% create new window to display output image
    figure(i)
    image(RuleO);
    colormap(gray(256));

% --- Executes during object creation, after setting all properties.
function runname_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function runname_Callback(hObject, eventdata, handles)

% --- Executes on button press in closebutton.
function closebutton_Callback(hObject, eventdata, handles)

close; % closes the GUI without any extra saving

% --- Executes on button press in displayoutput.
function displayoutput_Callback(hObject, eventdata, handles)
% Hint: get(hObject,'Value') returns toggle state of displayoutput

if get(hObject,'Value') == 1
    set(handles.displayoutput,'string','On');
else
    set(handles.displayoutput,'string','Off');
end

% -----
function varargout = savenndata_Callback(hObject, eventdata, handles)

% Hint: get(hObject,'Value') returns toggle state of displayoutput

if get(hObject,'Value') == 1
    set(handles.savenndata,'string','On');
else
    set(handles.savenndata,'string','Off');
end

```

```

% --- Executes on button press in nonwindowfilter.
function nonwindowfilter_Callback(hObject, eventdata, handles)
% trains network with rules for pollen classification
SNN = get(handles.mydata,'UserData'); % make local variable SNN = "global" SNN
set(handles.runbutton,'userdata','nonwindowfilter');

% get batch-run data *****
%get the network ready
disp('Begun parsing the Network');
SNN = parse3(SNN);

F = CreateFuNN2(SNN); %create the FuNN
%% load weight into FuNN
randn('state',0);
R = SNN.RuleMatrix*0.5; % quality factor is 0.5
R = R + randn(size(SNN.RuleMatrix))*0.1;
A = SNN.ActionMatrix*0.5;
A = A + randn(size(SNN.ActionMatrix))*0.1;
F = ModifyRuleMatrix(F, R);
F = ModifyActionMatrix(F, A);
F = ModifyEpochs(F, SNN.Epochs);
disp('Network ready');
%disp(F);

qanswer = questdlg('Do you want to find sgldm data?',...
    'Network Type?','Yes','No','Yes');
if strcmp(qanswer,'No')
    lv = getinputvectors2;
    disp('Input vectors found');
    Tv = gettargetvectors2(lv);
    disp('Target vectors found');
else
    lv = getinputvectorssgldm
    Tv = gettargetvectorssgldm(lv)
end
%% train NN using input/target vectors
[F, RulePerf] = TrainFuNN(F, lv, Tv);
% F now represents the trained network that can be used on any image
% set it so classify function can use it
    %msgbox('Network has been trained','Notice','help');
    %uiwait;
    %close; % rule performance window
set(handles.nonwindowfilter,'userdata',F);
%temp*****
%classifyPollen2(F);
set(handles.nonwindowfilter,'value',0);

% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes on button press in classifypollen.
function classifypollen_Callback(hObject, eventdata, handles)
% hObject    handle to classifypollen (see GCBO)

```

```

% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

F = get(handles.nonwindowfilter,'userdata');
[filename,path] = uigetfile('aimages\*.tif','Load Image to be classified');
[Pimage,pmap] = imread([path,filename]);
%Pimage = double(Pimage);
image(Pimage); % display the image
colormap(pmap);
msgbox('Image loaded','Notice','help');
uiwait
disp('Processing Image');
% code here does the classification
%find the interesting info for each image
[ycentre,xcentre,horizmaxrun,vertmaxrun,BW2,area,pts,avgPixelIntensity] =
findcentremass(Pimage);
[circularity,rmean] = findcircularity(ycentre,xcentre,vertmaxrun,horizmaxrun,BW2);
lv = [circularity;area];

disp('Circularity');
disp(circularity);
disp('Area');
disp(area);
% simulate the FuNN and record data
results = SimFuNN(F, lv);

%display results
displayresult2(1,1, results, filename, circularity,0);

%msgbox({'Result for',filename,'API: ',num2str(avgPixelIntensity),'Area:',num2str(area)},...
% 'Results','help');
%uiwait;
%%classifyPollen(F);

% -----
function varargout = classifygroupimages_Callback(h, eventdata, handles, varargin)

F = get(handles.nonwindowfilter,'userdata');
qanswer = questdlg('Do you want to classify sgldm data?',...
    'Classification Type?','Yes','No','Yes');
if strcmp(qanswer,'Yes')
    classifyPollenSGLDM(F);
else
    classifyPollen2(F);
end

% -----
function varargout = trainwithnorules_Callback(h, eventdata, handles, varargin)
% trains network with-out rules for pollen classification
SNN = get(handles.mydata,'UserData'); % make local variable SNN = "global" SNN

% get batch-run data *****
%get the network ready
disp('Begun parsing the Network');
SNN = parse3(SNN);

F = CreateFuNN2(SNN); %create the FuNN

```



```

%% load weight into FuNN
randn('state',0);
%R = SNN.RuleMatrix*0.5; % quality factor is 0.5
R = randn(size(SNN.RuleMatrix))*0.1;
%A = SNN.ActionMatrix*0.5;
A = randn(size(SNN.ActionMatrix))*0.1;
F = ModifyRuleMatrix(F, R);
F = ModifyActionMatrix(F, A);
F = ModifyEpochs(F, SNN.Epochs);
disp('Network ready');

qanswer = questdlg('Do you want to find sgldm data?',...
    'Network Type?', 'Yes', 'No', 'Yes');
if strcmp(qanswer, 'Yes')
    lv = getinputvectorssgldm;
    disp('Input vectors found');
    Tv = gettargetvectorssgldm(lv);
    disp('Target vectors found');
else
    lv = getinputvectors2;
    Tv = gettargetvectors2(lv);
end

%% train NN using input/target vectors
[F, RulePerf] = TrainFuNN(F, lv, Tv);

set(handles.nonwindowfilter, 'userdata', F);
set(handles.nonwindowfilter, 'value', 0);

```

Functions for 'ChangeFuzzyMembershipValues' GUI

```

function varargout = DataInput3(varargin)
% This GUI is used to change fuzzy membership rules
% the 'Adjust rules GUI can be called from here

if isstruct(varargin{1}) % LAUNCH GUI - Check to make sure that first variable is a structure

    fig = openfig(mfilename, 'reuse');

    % Use system color scheme for figure:
    set(fig, 'Color', get(0, 'defaultUicontrolBackgroundColor'));

    % Generate a structure of handles to pass to callbacks, and store it.
    handles = guihandles(fig);
    guidata(fig, handles);

% -----opening code-----

disp(' Data input section started');
load ('tempqpzm10');
set(handles.mysnndata, 'Userdata', SNN); % save SNN to mydata userdata
%set(handles.edit1, 'Userdata', info); % save file info into user data for later use (saving)

% display only enough edit boxes as required, ie. one per Fuzzy rule
if SNN.FuzzyMF >= 1 set(handles.edit1, 'string', SNN.FuzzyMFDEF(1), 'visible', 'On'); end
if SNN.FuzzyMF >= 2 set(handles.edit2, 'string', SNN.FuzzyMFDEF(2), 'visible', 'On'); end
if SNN.FuzzyMF >= 3 set(handles.edit3, 'string', SNN.FuzzyMFDEF(3), 'visible', 'On'); end
if SNN.FuzzyMF >= 4 set(handles.edit4, 'string', SNN.FuzzyMFDEF(4), 'visible', 'On'); end

```

```

    if SNN.FuzzyMF >= 5 set(handles.edit5,'string', SNN.FuzzyMFDEF(5),'visible','On'); end
    if SNN.FuzzyMF >= 6 set(handles.edit6,'string', SNN.FuzzyMFDEF(6),'visible','On'); end
    if SNN.FuzzyMF >= 7 set(handles.edit7,'string', SNN.FuzzyMFDEF(7),'visible','On'); end
    if SNN.FuzzyMF >= 8 set(handles.edit8,'string', SNN.FuzzyMFDEF(8),'visible','On'); end
    if SNN.FuzzyMF >= 9 set(handles.edit9,'string', SNN.FuzzyMFDEF(9),'visible','On'); end
    if SNN.FuzzyMF >= 10 set(handles.edit10,'string', SNN.FuzzyMFDEF(10),'visible','On');
end
    if SNN.FuzzyMF >= 11 set(handles.edit11,'string', SNN.FuzzyMFDEF(11),'visible','On');
end
    if SNN.FuzzyMF >= 12
        msgbox('There are too many rules to display them all','Warning','warn');
    end
    % display only enough edit boxes as required, ie. one per DeFuzzy rule
    if SNN.DefuzzyMF >= 1
        set(handles.defuzzy1,'string',SNN.DefuzzyMFDEF(1),'visible','on');end
    if SNN.DefuzzyMF >= 2
        set(handles.defuzzy2,'string',SNN.DefuzzyMFDEF(2),'visible','on');end
    if SNN.DefuzzyMF >= 3
        set(handles.defuzzy3,'string',SNN.DefuzzyMFDEF(3),'visible','on');end
    if SNN.DefuzzyMF >= 4
        set(handles.defuzzy4,'string',SNN.DefuzzyMFDEF(4),'visible','on');end
    if SNN.DefuzzyMF >= 5
        set(handles.defuzzy5,'string',SNN.DefuzzyMFDEF(5),'visible','on');end
    if SNN.DefuzzyMF >= 6
        msgbox('There are too many rules to display them all','Warning','warn');
    end

    % display an appropriate number of radio buttons, i.e. 1 button per "Text" rule
    if SNN.NumberRules >= 1
        set(handles.radiobutton1,'visible','on',...
            'value',1); %turn this button 'on'
        set(handles.adjustrules,'userdata',1); % set value indicator to '1'
    end
    if SNN.NumberRules >= 2 set(handles.radiobutton2,'visible','on'); end
    if SNN.NumberRules >= 3 set(handles.radiobutton3,'visible','on');end
    if SNN.NumberRules >= 4 set(handles.radiobutton4,'visible','on');end
    if SNN.NumberRules >= 5 set(handles.radiobutton5,'visible','on');end
    if SNN.NumberRules >= 6 set(handles.radiobutton6,'visible','on');end
    if SNN.NumberRules >= 7 set(handles.radiobutton7,'visible','on');end
    if SNN.NumberRules >= 8 set(handles.radiobutton8,'visible','on');end
    if SNN.NumberRules >= 9 set(handles.radiobutton9,'visible','on');end
    if SNN.NumberRules >= 10 set(handles.radiobutton10,'visible','on');end
    if SNN.NumberRules >= 11 set(handles.radiobutton11,'visible','on');end

    plotgraph(handles,SNN);

    % ---end of my opening routines-----

    if nargout > 0
        varargout{1} = fig;
    end

elseif ischar(varargin{1}) % INVOKE NAMED SUBFUNCTION OR CALLBACK
    try
        if (nargout)
            [varargout{1:nargout}] = feval(varargin{:}); % FEVAL switchyard
        else
            feval(varargin{:}); % FEVAL switchyard
        end
    end

```

```

        catch
            disp(lasterr);
        end

    end

end

% ---subfunctions-----
function mutual_exclude(off)
set(off,'Value',0)

function plotgraph(handles,SNN)
% plot graph of the FuzzyMFDEF data with values on y axis
% x axis values are all '1'.
[yvalues1,yvalues2] = getYvalues(SNN);
hold off;
plot(SNN.FuzzyMFDEF,yvalues1); % plot the graph of the FuzzyMFDEF data
hold on; % so next plot line doesn't write-over previous plot line
plot(SNN.FuzzyMFDEF,yvalues2); % plot the graph of the FuzzyMFDEF data
hold off;

function [yv1,yv2] = getYvalues(SNN)
% returns 2 arrays [1,0,1,0,1...] and [0,1,0,1,0...]
% Each array is the length of SNN.FuzzyMF
yvalues1 = [];
yvalues2 = [];
for i = 1:SNN.FuzzyMF
    if mod(i,2)
        yvalues1 = [yvalues1,1];
        yvalues2 = [yvalues2,0];
    else
        yvalues1 = [yvalues1,0];
        yvalues2 = [yvalues2,1];
    end
end
yv1 = yvalues1;
yv2 = yvalues2;
%-----
function r = replotgraph(handles, newvalue, boxnumber)
% checks to see if the value entered in the edit box is between 0 and 1
% if so, 'records' the new value & re-plots the graph, if not returns 'error'
report = 'ok';
SNN = get(handles.mysnndata,'UserData'); % make local variable SNN = "global" SNN
if (newvalue >=0) & (newvalue <=1)
    SNN.FuzzyMFDEF(boxnumber)=newvalue; % update changed variable
    set(handles.mysnndata,'UserData',SNN); % set 'global' SNN
    plotgraph(handles,SNN); % plot the graph of the FuzzyMFDEF data
else
    msgbox('This value is out of bounds, please re-enter a value between "0" and "1",...
        'Value Out Of Bounds','error');
    uiwait;
    report = 'error';
end
r = report;
% -----
function varargout = edit1_Callback(h, eventdata, handles, varargin)
newvalue = str2num(get(handles.edit1,'string'));
if (strcmp(replotgraph(handles, newvalue, 1),'error'))
    SNN = get(handles.mysnndata,'UserData');
    set(handles.edit1,'string',SNN.FuzzyMFDEF(1)); %reset the variable on the interface

```

```

end

% -----
function varargout = edit2_Callback(h, eventdata, handles, varargin)
newvalue = str2num(get(handles.edit2,'string'));
if (strcmp(replotgraph(handles, newvalue, 2),'error'))
    SNN = get(handles.mysnndata,'UserData');
    set(handles.edit2,'string',SNN.FuzzyMFDEF(2)); %reset the variable on the interface
end

% -----
function varargout = edit3_Callback(h, eventdata, handles, varargin)
newvalue = str2num(get(handles.edit3,'string'));
if (strcmp(replotgraph(handles, newvalue, 3),'error'))
    SNN = get(handles.mysnndata,'UserData');
    set(handles.edit3,'string',SNN.FuzzyMFDEF(3)); %reset the variable on the interface
end

% -----
function varargout = edit4_Callback(h, eventdata, handles, varargin)
newvalue = str2num(get(handles.edit4,'string'));
if (strcmp(replotgraph(handles, newvalue, 4),'error'))
    SNN = get(handles.mysnndata,'UserData');
    set(handles.edit4,'string',SNN.FuzzyMFDEF(4)); %reset the variable on the interface
end

% -----
function varargout = edit5_Callback(h, eventdata, handles, varargin)
newvalue = str2num(get(handles.edit5,'string'));
if (strcmp(replotgraph(handles, newvalue, 5),'error'))
    SNN = get(handles.mysnndata,'UserData');
    set(handles.edit5,'string',SNN.FuzzyMFDEF(5)); %reset the variable on the interface
end

% -----
function varargout = edit6_Callback(h, eventdata, handles, varargin)
newvalue = str2num(get(handles.edit6,'string'));
if (strcmp(replotgraph(handles, newvalue, 6),'error'))
    SNN = get(handles.mysnndata,'UserData');
    set(handles.edit6,'string',SNN.FuzzyMFDEF(6)); %reset the variable on the interface
end

% -----
function varargout = edit7_Callback(h, eventdata, handles, varargin)
newvalue = str2num(get(handles.edit7,'string'));
if (strcmp(replotgraph(handles, newvalue, 7),'error'))
    SNN = get(handles.mysnndata,'UserData');
    set(handles.edit7,'string',SNN.FuzzyMFDEF(7)); %reset the variable on the interface
end

% -----
function varargout = edit8_Callback(h, eventdata, handles, varargin)
newvalue = str2num(get(handles.edit8,'string'));
if (strcmp(replotgraph(handles, newvalue, 8),'error'))
    SNN = get(handles.mysnndata,'UserData');
    set(handles.edit8,'string',SNN.FuzzyMFDEF(8)); %reset the variable on the interface
end

% -----
function varargout = edit9_Callback(h, eventdata, handles, varargin)

```

```

newvalue = str2num(get(handles.edit9,'string'));
if (strcmp(replotgraph(handles, newvalue, 9),'error'))
    SNN = get(handles.mysnndata,'UserData');
    set(handles.edit9,'string',SNN.FuzzyMFDEF(9)); %reset the variable on the interface
end

% -----
function varargout = edit10_Callback(h, eventdata, handles, varargin)
newvalue = str2num(get(handles.edit10,'string'));
SNN = get(handles.mysnndata,'UserData'); % make local variable SNN = "global" SNN
if (newvalue >=0) & (newvalue <=1)
    SNN.FuzzyMFDEF(10)=newvalue; % update changed variable
    set(handles.mysnndata,'UserData',SNN); % set 'global' SNN
    plot(SNN.FuzzyMFDEF); % plot the graph of the FuzzyMFDEF data
else
    msgbox('This value is out of bounds, please re-enter a value between "0" and "1"',...
        'Value Out Of Bounds','error');
    uiwait;
    set(handles.edit10,'string',SNN.FuzzyMFDEF(10)); %reset the variable on the interface
end

% -----
function varargout = edit11_Callback(h, eventdata, handles, varargin)
newvalue = str2num(get(handles.edit11,'string'));
if (strcmp(replotgraph(handles, newvalue, 11),'error'))
    SNN = get(handles.mysnndata,'UserData');
    set(handles.edit11,'string',SNN.FuzzyMFDEF(11)); %reset the variable on the interface
end

% -----
function varargout = okbutton_Callback(h, eventdata, handles, varargin)
% appears on GUI as "Save/Close" button
SNN = get(handles.mysnndata,'UserData'); % make local variable SNN = "global" SNN
% Don't need to set all SNN variables here because they have been reset during edit-box
callbacks
%info = get(handles.edit1,'Userdata');
%save([info.fpath,info.fname], 'SNN'); %load in the saved settings file (ie SNN variable)
%disp(SNN.Rule);
disp(' Closing datainput');
save('tempqpzm10','SNN'); %save to a temp file to be read by 'main' GUI
close;

% --- Executes on button press in "Adjustrule Matrix"-----
function adjustrules_Callback(hObject, eventdata, handles)

SNN = get(handles.mysnndata,'UserData'); % make local variable SNN = "global" SNN
Rule = get(handles.adjustrules,'userdata'); % put currentrule number into 'Rule'
save('ARuletempqpzm10a','Rule'); % save 'Rule' to a temp file to be read by 'AdjustRules'
GUI
save('AdjustRtempqpzm10','SNN'); %save 'SNN' to a temp file to be read by 'AdjustRules'
GUI
AdjustRules; % call adjustrules GUI
uiwait %wait for AdjustRules GUI to return

load('AdjustRtempqpzm10','SNN'); % load temp file containing 'SNN' saved by the
'AdjustRules' GUI
disp(' Control back to datainput');
disp([' Rule as read by DataInput ',SNN.Rule]);

```

```

set(handles.mysnndata,'userdata',SNN); % set 'global' SNN
%no need to redisplay variables as Adjust Rules doesn't change anything visible on this
screen
set(handles.adjustrules,'userdata',Rule); %indicates current radio button

%*****RADIO BUTTONS*****
% --- Executes on button press in radiobutton1.
function radiobutton1_Callback(hObject, eventdata, handles)
%make sure radio buttons are mutually exclusive
off = [handles.radiobutton2,handles.radiobutton3,...
    handles.radiobutton4,handles.radiobutton5,handles.radiobutton6,...
    handles.radiobutton7,handles.radiobutton8,handles.radiobutton9,...
    handles.radiobutton10,handles.radiobutton11,handles.radiobutton12];
mutual_exclude(off);
set(handles.adjustrules,'userdata',1); %indicates current radio button

% --- Executes on button press in radiobutton2.
function radiobutton2_Callback(hObject, eventdata, handles)
%make sure radio buttons are mutually exclusive
off = [handles.radiobutton1,handles.radiobutton3,...
    handles.radiobutton4,handles.radiobutton5,handles.radiobutton6,...
    handles.radiobutton7,handles.radiobutton8,handles.radiobutton9,...
    handles.radiobutton10,handles.radiobutton11,handles.radiobutton12];
mutual_exclude(off);
set(handles.adjustrules,'userdata',2); %indicates current radio button

% --- Executes on button press in radiobutton3.
function radiobutton3_Callback(hObject, eventdata, handles)
%make sure radio buttons are mutually exclusive
off = [handles.radiobutton1,handles.radiobutton2,...
    handles.radiobutton4,handles.radiobutton5,handles.radiobutton6,...
    handles.radiobutton7,handles.radiobutton8,handles.radiobutton9,...
    handles.radiobutton10,handles.radiobutton11,handles.radiobutton12];
mutual_exclude(off);
set(handles.adjustrules,'userdata',3); %indicates current radio button

% --- Executes on button press in radiobutton4.
function radiobutton4_Callback(hObject, eventdata, handles)
%make sure radio buttons are mutually exclusive
off = [handles.radiobutton1,handles.radiobutton2,handles.radiobutton3,...
    handles.radiobutton5,handles.radiobutton6,...
    handles.radiobutton7,handles.radiobutton8,handles.radiobutton9,...
    handles.radiobutton10,handles.radiobutton11,handles.radiobutton12];
mutual_exclude(off);
set(handles.adjustrules,'userdata',4); %indicates current radio button

% --- Executes on button press in radiobutton5.
function radiobutton5_Callback(hObject, eventdata, handles)
%make sure radio buttons are mutually exclusive
off = [handles.radiobutton1,handles.radiobutton2,handles.radiobutton3,...
    handles.radiobutton4,handles.radiobutton6,...
    handles.radiobutton7,handles.radiobutton8,handles.radiobutton9,...
    handles.radiobutton10,handles.radiobutton11,handles.radiobutton12];
mutual_exclude(off);
set(handles.adjustrules,'userdata',5); %indicates current radio button

% --- Executes on button press in radiobutton6.
function radiobutton6_Callback(hObject, eventdata, handles)
%make sure radio buttons are mutually exclusive
off = [handles.radiobutton1,handles.radiobutton2,handles.radiobutton3,...

```

```

handles radiobutton4,handles radiobutton5,...
handles radiobutton7,handles radiobutton8,handles radiobutton9,...
handles radiobutton10,handles radiobutton11,handles radiobutton12];
mutual_exclude(off);
set(handles.adjustrules,'userdata',6); %indicates current radio button

% --- Executes on button press in radiobutton7.
function radiobutton7_Callback(hObject, eventdata, handles)
%make sure radio buttons are mutually exclusive
off = [handles radiobutton1,handles radiobutton2,handles radiobutton3,...
handles radiobutton4,handles radiobutton5,handles radiobutton6,...
handles radiobutton8,handles radiobutton9,...
handles radiobutton10,handles radiobutton11,handles radiobutton12];
mutual_exclude(off);
set(handles.adjustrules,'userdata',7); %indicates current radio button

% --- Executes on button press in radiobutton8.
function radiobutton8_Callback(hObject, eventdata, handles)
%make sure radio buttons are mutually exclusive
off = [handles radiobutton1,handles radiobutton2,handles radiobutton3,...
handles radiobutton4,handles radiobutton5,handles radiobutton6,...
handles radiobutton7,handles radiobutton9,...
handles radiobutton10,handles radiobutton11,handles radiobutton12];
mutual_exclude(off);
set(handles.adjustrules,'userdata',8); %indicates current radio button

% --- Executes on button press in radiobutton9.
function radiobutton9_Callback(hObject, eventdata, handles)
%make sure radio buttons are mutually exclusive
off = [handles radiobutton1,handles radiobutton2,handles radiobutton3,...
handles radiobutton4,handles radiobutton5,handles radiobutton6,...
handles radiobutton7,handles radiobutton8,...
handles radiobutton10,handles radiobutton11,handles radiobutton12];
mutual_exclude(off);
set(handles.adjustrules,'userdata',9); %indicates current radio button

% --- Executes on button press in radiobutton10.
function radiobutton10_Callback(hObject, eventdata, handles)
%make sure radio buttons are mutually exclusive
off = [handles radiobutton1,handles radiobutton2,handles radiobutton3,...
handles radiobutton4,handles radiobutton5,handles radiobutton6,...
handles radiobutton7,handles radiobutton8,handles radiobutton9,...
handles radiobutton11,handles radiobutton12];
mutual_exclude(off);
set(handles.adjustrules,'userdata',10); %indicates current radio button

% --- Executes on button press in radiobutton11.
function radiobutton11_Callback(hObject, eventdata, handles)
%make sure radio buttons are mutually exclusive
off = [handles radiobutton1,handles radiobutton2,handles radiobutton3,...
handles radiobutton4,handles radiobutton5,handles radiobutton6,...
handles radiobutton7,handles radiobutton8,handles radiobutton9,...
handles radiobutton10,handles radiobutton12];
mutual_exclude(off);
set(handles.adjustrules,'userdata',11); %indicates current radio button

% --- Executes on button press in radiobutton12.
function radiobutton12_Callback(hObject, eventdata, handles)
%make sure radio buttons are mutually exclusive
off = [handles radiobutton1,handles radiobutton2,handles radiobutton3,...

```

```

handles radiobutton4,handles radiobutton5,handles radiobutton6,...
handles radiobutton7,handles radiobutton8,handles radiobutton9,...
handles radiobutton10,handles radiobutton11;
mutual_exclude(off);
set(handles.adjustrules,'userdata',12); %indicates current radio button

```

```

function checkDefuzzychange(handles, newvalue, boxnumber)
% not necessary yet, with only 3 edit boxes

```

```

function defuzzy1_CreateFcn(hObject, eventdata, handles)
function defuzzy2_CreateFcn(hObject, eventdata, handles)
function defuzzy3_CreateFcn(hObject, eventdata, handles)

```

```

function defuzzy1_Callback(hObject, eventdata, handles)
% str2double(get(hObject,'String')) returns contents of defuzzy1 as a double
%checkDefuzzychange(handles, str2double(get(hObject,'String')), 1)
if (str2double(get(hObject,'String')) > 1) | (str2double(get(hObject,'String')) < 0)
    msgbox('This value is out of range';'Please enter a value between 0 and 1','Error','warn');
else
    SNN = get(handles.mysnndata,'UserData'); % make local variable SNN = "global" SNN
    SNN.DefuzzyMFDEF(1) = str2double(get(hObject,'String'));
    set(handles.mysnndata,'userdata',SNN); % set changes
end

```

```

function defuzzy2_Callback(hObject, eventdata, handles)
if (str2double(get(hObject,'String')) > 1) | (str2double(get(hObject,'String')) < 0)
    msgbox('This value is out of range';'Please enter a value between 0 and 1','Error','warn');
else
    SNN = get(handles.mysnndata,'UserData'); % make local variable SNN = "global" SNN
    SNN.DefuzzyMFDEF(2) = str2double(get(hObject,'String'));
    set(handles.mysnndata,'userdata',SNN); % set changes
end

```

```

function defuzzy3_Callback(hObject, eventdata, handles)
if (str2double(get(hObject,'String')) > 1) | (str2double(get(hObject,'String')) < 0)
    msgbox('This value is out of range';'Please enter a value between 0 and 1','Error','warn');
else
    SNN = get(handles.mysnndata,'UserData'); % make local variable SNN = "global" SNN
    SNN.DefuzzyMFDEF(3) = str2double(get(hObject,'String'));
    set(handles.mysnndata,'userdata',SNN); % set changes
end

```

```

function defuzzy4_Callback(hObject, eventdata, handles)
if (str2double(get(hObject,'String')) > 1) | (str2double(get(hObject,'String')) < 0)
    msgbox('This value is out of range';'Please enter a value between 0 and 1','Error','warn');
else
    SNN = get(handles.mysnndata,'UserData'); % make local variable SNN = "global" SNN
    SNN.DefuzzyMFDEF(4) = str2double(get(hObject,'String'));
    set(handles.mysnndata,'userdata',SNN); % set changes
end

```

```

function defuzzy5_Callback(hObject, eventdata, handles)
if (str2double(get(hObject,'String')) > 1) | (str2double(get(hObject,'String')) < 0)
    msgbox('This value is out of range';'Please enter a value between 0 and 1','Error','warn');
else
    SNN = get(handles.mysnndata,'UserData'); % make local variable SNN = "global" SNN
    SNN.DefuzzyMFDEF(5) = str2double(get(hObject,'String'));
    set(handles.mysnndata,'userdata',SNN); % set changes
end

```



```

% --- Executes on button press in addrule.
function addrule_Callback(hObject, eventdata, handles)
% NOTE this refers to a FuzzyMFDEF Rule!!!!!!!!!!!!!!
SNN = get(handles.mysnndata,'UserData'); % make local variable SNN = "global" SNN
%disp( ['Current FuzzyMFDEF is ',SNN.FuzzyMF]);
SNN.FuzzyMF = SNN.FuzzyMF + 1; %add '1' to the number of ....
SNN.FuzzyMFDEF(SNN.FuzzyMF) = 1; %set new value to '1'

%need to make this more efficient
% make sure the correct number of values are shown on the interface
if (SNN.FuzzyMF == 1) set(handles.edit1,'string','1', 'visible','On'); end
if SNN.FuzzyMF == 2 set(handles.edit2,'string','1', 'visible','On');end
if SNN.FuzzyMF == 3 set(handles.edit3,'string','1', 'visible','On');end
if SNN.FuzzyMF == 4 set(handles.edit4,'string','1', 'visible','On');end
if SNN.FuzzyMF == 5 set(handles.edit5,'string','1', 'visible','On');end
if SNN.FuzzyMF == 6 set(handles.edit6,'string','1', 'visible','On');end
if SNN.FuzzyMF == 7 set(handles.edit7,'string','1', 'visible','On'); end
if SNN.FuzzyMF == 8 set(handles.edit8,'string','1', 'visible','On'); end
if SNN.FuzzyMF == 9 set(handles.edit9,'string','1', 'visible','On'); end
if SNN.FuzzyMF == 10 set(handles.edit10,'string','1','visible','On'); end
if SNN.FuzzyMF == 11 set(handles.edit11,'string','1','visible','On'); end

set(handles.mysnndata,'userdata',SNN); % set changes
plot(SNN.FuzzyMFDEF); % plot the graph of the 'new' FuzzyMFDEF data
msgbox('The new FuzzyMFDEF has been set to the default value of "1", you may change
that if you wish','NOTE','help');
uiwait;
if SNN.FuzzyMF >=12
    msgbox('There are too many rules to display them all','Warning','warn');
end
%uiwait;

% --- Executes on button press in deleterule.
function deleterule_Callback(hObject, eventdata, handles)

SNN = get(handles.mysnndata,'UserData'); % make local variable SNN = "global" SNN
SNN.FuzzyMFDEF(SNN.FuzzyMF) = ""; %delete the last value
SNN.FuzzyMF = SNN.FuzzyMF - 1; % subtract '1' from the number of ....

%answer = inputdlg('Which number rule do you wish to delete?','...
% 'Delete a Rule',[1]) %display=====
%uiwait;
%answer = str2num(cell2mat(answer));

%if ~isempty(answer)
    %SNN.FuzzyMFDEF(answer) = ""; %delete the specified value
    %end

set(handles.mysnndata,'userdata',SNN); % set changes
plot(SNN.FuzzyMFDEF); % plot the graph of the 'new' FuzzyMFDEF data

%if SNN.FuzzyMF >= 1 set(handles.edit1,'string',SNN.FuzzyMFDEF(1)); end
%if SNN.FuzzyMF >= 2 set(handles.edit2,'string',SNN.FuzzyMFDEF(2));end
%if SNN.FuzzyMF >= 3 set(handles.edit3,'string',SNN.FuzzyMFDEF(3));end
%if SNN.FuzzyMF >= 4 set(handles.edit4,'string',SNN.FuzzyMFDEF(4));end
%if SNN.FuzzyMF >= 5 set(handles.edit5,'string',SNN.FuzzyMFDEF(5));end
%if SNN.FuzzyMF >= 6 set(handles.edit6,'visible','Off');end

```

```

%if SNN.FuzzyMF >= 7 set(handles.edit7,'visible','Off'); end
%if SNN.FuzzyMF >= 8 set(handles.edit8,'visible','Off'); end
%if SNN.FuzzyMF >= 9 set(handles.edit9,'visible','Off'); end

if SNN.FuzzyMF == 1 set(handles.edit2,'visible','Off'); end
if SNN.FuzzyMF == 2 set(handles.edit3,'visible','Off');end
if SNN.FuzzyMF == 3 set(handles.edit4,'visible','Off');end
if SNN.FuzzyMF == 4 set(handles.edit5,'visible','Off');end
if SNN.FuzzyMF == 5 set(handles.edit6,'visible','Off');end
if SNN.FuzzyMF == 6 set(handles.edit7,'visible','Off');end
if SNN.FuzzyMF == 7 set(handles.edit8,'visible','Off'); end
if SNN.FuzzyMF == 8 set(handles.edit9,'visible','Off'); end
if SNN.FuzzyMF == 9 set(handles.edit10,'visible','Off'); end

% --- Executes on button press in addtextrule.
function addtextrule_Callback(hObject, eventdata, handles)

SNN = get(handles.mysnndata,'UserData'); % make local variable SNN = "global" SNN
SNN.NumberRules = SNN.NumberRules + 1;
SNN.Rule = [SNN.Rule, ' If I1=M1 Then O1=M1']; % add default rule to SNN.Rule
set(handles.mysnndata,'userdata',SNN); % set changes

if SNN.NumberRules >= 1 set(handles radiobutton1,'visible','on'); end
if SNN.NumberRules >= 2 set(handles radiobutton2,'visible','on'); end
if SNN.NumberRules >= 3 set(handles radiobutton3,'visible','on');end
if SNN.NumberRules >= 4 set(handles radiobutton4,'visible','on');end
if SNN.NumberRules >= 5 set(handles radiobutton5,'visible','on');end
if SNN.NumberRules >= 6 set(handles radiobutton6,'visible','on');end
if SNN.NumberRules >= 7 set(handles radiobutton7,'visible','on');end
if SNN.NumberRules >= 8 set(handles radiobutton8,'visible','on');end
if SNN.NumberRules >= 9 set(handles radiobutton9,'visible','on');end
if SNN.NumberRules >= 10 set(handles radiobutton10,'visible','on');end
if SNN.NumberRules >= 11 set(handles radiobutton11,'visible','on');end
if SNN.NumberRules >= 12
    msgbox('Come-on, that"s too many rules!','Error','warn');
else
    msgbox(['You have just added rule number ',num2str(SNN.NumberRules)],'Note','help');
    uiwait
end

% call adjust rules GUI
%Rule = SNN.NumberRules;
%save('tempqpzm10a','Rule');%save current ruelnumber to be read by 'AdjustRules' GUI
%save('tempqpzm10','SNN'); %save to a temp file to be read by 'AdjustRules' GUI
%AdjustRules; % call adjustrules GUI
%uiwait %wait for AdjustRules GUI to return
%load('tempqpzm10','SNN'); %load updated SNN
disp(' New Rule added');
disp([' New Rule ',num2str(SNN.NumberRules),'. (All) Rule read by DataInput ',SNN.Rule]);
set(handles.mysnndata,'userdata',SNN);

% --- Executes on button press in deletetextrule.
function deletetextrule_Callback(hObject, eventdata, handles)
% deletes the currently selected text rule
SNN = get(handles.mysnndata,'UserData'); % make local variable SNN = "global" SNN
if SNN.NumberRules == 0
    msgbox('There are no rules left to delete','Warning','warn');
else
    current = get(handles.adjustrules,'userdata'); % the rule currently selected

```

```

SNN.Rule = strrep(SNN.Rule,getrule(SNN.Rule,current,SNN.NumberRules),"); % replace
selected rule with " (nothing)
SNN.NumberRules = SNN.NumberRules - 1; % sub 1 from number of rules
disp(['Rule deleted. Rule line now reads ', SNN.Rule]);
% update screen display
if SNN.NumberRules < 1    set(handles.radiobutton1,'visible','off'); end
if SNN.NumberRules < 2    set(handles.radiobutton2,'visible','off'); end
if SNN.NumberRules < 3    set(handles.radiobutton3,'visible','off');end
if SNN.NumberRules < 4    set(handles.radiobutton4,'visible','off');end
if SNN.NumberRules < 5    set(handles.radiobutton5,'visible','off');end
if SNN.NumberRules < 6    set(handles.radiobutton6,'visible','off');end
if SNN.NumberRules < 7    set(handles.radiobutton7,'visible','off');end
if SNN.NumberRules < 8    set(handles.radiobutton8,'visible','off');end
if SNN.NumberRules < 9    set(handles.radiobutton9,'visible','off');end
if SNN.NumberRules < 10    set(handles.radiobutton10,'visible','off');end
if SNN.NumberRules < 11    set(handles.radiobutton11,'visible','off');end
if (current > SNN.NumberRules) % if deleted rule was the last one, make current rule Rule 1
    set(handles.radiobutton1,'value',1);
    set(handles.adjustrules,'userdata',1);
end % end if
set(handles.mysnndata,'userdata',SNN);
end % end if / else

% --- Executes on mouse press over axes background.
function axes1_ButtonDownFcn(hObject, eventdata, handles)
% does nothing

% --- Executes on mouse press over figure background, over a disabled or
% --- inactive control, or over an axes background.
function figure1_WindowButtonDownFcn(hObject, eventdata, handles)
disp('Resetting FuzzyMFDEF graph using mouse-click');
SNN = get(handles.mysnndata,'UserData'); % make local variable SNN = "global" SNN

minx = min(SNN.FuzzyMFDEF); % usually is '0'
maxx = max(SNN.FuzzyMFDEF); % usually is '1'
miny = 0;
maxy = 1;
%[col, maxx] = size(SNN.FuzzyMFDEF)
a = get(gca,'CurrentPoint'); % button down detected, point stored in 'a'
x = a(1,1) % x value of button click (relative to the graph)
newy = a(1,2); % y value of button click (relative to the graph)

if (newy >=miny) & (newy <=maxy) & (x > minx) & (x < maxx) % if click is in graph
    disp('here');
    % determine which value we want to change
    % next 5 lines where the idea of Sanj Gunetilleke
    a = SNN.FuzzyMFDEF;
    b = a - x;
    b = abs(b);
    c = min(b);
    changevalue = find(b == c);
    SNN.FuzzyMFDEF(changevalue) = x; % reset the new value for FuzzyMFDEF
    plotgraph(handles,SNN); % plot the graph of the FuzzyMFDEF data
    set(handles.mysnndata,'userdata',SNN); % set SNN back to 'global'
    % make sure the correct numbers are shown on the interface
    % must round newy to
    % 4.d.p
    if changevalue == 1    set(handles.edit1,'string',x); end
    if changevalue == 2    set(handles.edit2,'string',x);end

```

```

    if changevalue == 3    set(handles.edit3,'string',x);end
    if changevalue == 4    set(handles.edit4,'string',x);end
    if changevalue == 5    set(handles.edit5,'string',x);end
    if changevalue == 6    set(handles.edit6,'string',x);end
    if changevalue == 7    set(handles.edit7,'string',x); end
    if changevalue == 8    set(handles.edit8,'string',x); end
    if changevalue == 9    set(handles.edit9,'string',x); end
    if changevalue == 10   set(handles.edit10,'string',x);end
end
%-----

% --- Executes on button press in adddefuzzyrule.
function adddefuzzyrule_Callback(hObject, eventdata, handles)
SNN = get(handles.mysnndata,'UserData'); % make local variable SNN = "global" SNN
%disp( ['Current FuzzyMFDEF is ',SNN.FuzzyMFDEF]);
if SNN.DefuzzyMF < 5
    SNN.DefuzzyMF = SNN.DefuzzyMF + 1; %add '1' to the number of ....
    SNN.DefuzzyMFDEF(SNN.DefuzzyMF) = 1; %set new value to '1'
    % make sure the correct number of values are shown on the interface
    if SNN.DefuzzyMF == 1    set(handles.defuzzy1,'string','1', 'visible','On'); end
    if SNN.DefuzzyMF == 2    set(handles.defuzzy2,'string','1', 'visible','On');end
    if SNN.DefuzzyMF == 3    set(handles.defuzzy3,'string','1', 'visible','On');end
    if SNN.DefuzzyMF == 4    set(handles.defuzzy4,'string','1', 'visible','On');end
    if SNN.DefuzzyMF == 5    set(handles.defuzzy5,'string','1', 'visible','On');end
    set(handles.mysnndata,'userdata',SNN); % set changes
else
    msgbox('Can't display that many DeFuzzy rules','Error','warn');
end

% --- Executes on button press in deletedefuzzyrule.
function deletedefuzzyrule_Callback(hObject, eventdata, handles)
SNN = get(handles.mysnndata,'UserData'); % make local variable SNN = "global" SNN
if SNN.DefuzzyMF >= 1
    SNN.DefuzzyMF = SNN.DefuzzyMF - 1; %subtract '1' from the number of rules
    % make sure the correct number of values are shown on the interface
    if SNN.DefuzzyMF < 1    set(handles.defuzzy1,'string','1', 'visible','Off'); end
    if SNN.DefuzzyMF < 2    set(handles.defuzzy2,'string','1', 'visible','Off');end
    if SNN.DefuzzyMF < 3    set(handles.defuzzy3,'string','1', 'visible','Off');end
    if SNN.DefuzzyMF < 4    set(handles.defuzzy4,'string','1', 'visible','Off');end
    if SNN.DefuzzyMF < 5    set(handles.defuzzy5,'string','1', 'visible','Off');end
    set(handles.mysnndata,'userdata',SNN); % set changes
else
    msgbox('There are no DeFuzzy rules to delete','Error','warn');
end

%-----NOT CURRENTLY USED FUNCTIONS-----
% -----
function updategraph(newvalue,position)
    SNN = get(handles.mysnndata,'UserData'); % make local variable SNN = "global" SNN
    SNN.FuzzyMFDEF(position)=newvalue;%update changed variable
    set(handles.mydata,'UserData',SNN);%set 'global' SNN
    plot(SNN.FuzzyMFDEF); % plot the graph of the FuzzyMFDEF data

% --- Executes during object creation, after setting all properties.
function defuzzy4_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');

```

```

else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

```

```

% --- Executes during object creation, after setting all properties.
function defuzzy5_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

```

Functions for 'Run NWF Network' GUI

```

function varargout = NWFnetworkPollen(varargin)

```

```

% Begin initialization code - DO NOT EDIT

```

```

gui_Singleton = 1;
gui_State = struct('gui_Name',    mfilename, ...
    'gui_Singleton', gui_Singleton, ...
    'gui_OpeningFcn', @NWFnetworkPollen_OpeningFcn, ...
    'gui_OutputFcn', @NWFnetworkPollen_OutputFcn, ...
    'gui_LayoutFcn', [] , ...
    'gui_Callback', []);

```

```

if nargin & isstr(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

```

```

if nargout

```

```

    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end

```

```

% End initialization code - DO NOT EDIT

```

```

% --- Executes just before NWFnetworkPollen is made visible.
function NWFnetworkPollen_OpeningFcn(hObject, eventdata, handles, varargin)
% Note:
% userdata of 'text1' holds the trained network
% userdata of 'mydata' holds NN variables
% Choose default command line output for NWFnetworkPollen

```

```

handles.output = hObject;
% Update handles structure
guidata(hObject, handles);
set(handles.axes1,'visible','Off');
disp(' Non window Filter Network section started');
load ('tempqpzm10');
set(handles.mydata,'userdata',SNN);
set(handles.typeA,'userdata','A');

```

```

% --- Outputs from this function are returned to the command line.

```

```

function varargout = NWFnetworkPollen_OutputFcn(hObject, eventdata, handles)
% Get default command line output from handles structure
varargout{1} = handles.output;

% -----Training functions-----
% --- Executes on button press in trainNoRules.
function trainNoRules_Callback(hObject, eventdata, handles)

% trains network with rules for pollen classification
SNN = get(handles.mydata,'UserData'); % make local variable SNN = "global" SNN

%get the network ready
disp('Begun parsing the Network');
SNN = parse3(SNN);

F = CreateFuNN2(SNN); %create the FuNN
%% load weight into FuNN
randn('state',0);
%R = SNN.RuleMatrix*0.5; % quality factor is 0.5
R = randn(size(SNN.RuleMatrix))*0.1;
%A = SNN.ActionMatrix*0.5;
A = randn(size(SNN.ActionMatrix))*0.1;
F = ModifyRuleMatrix(F, R);
F = ModifyActionMatrix(F, A);
F = ModifyEpochs(F, SNN.Epochs);
disp('Network ready');

lv = getinputvectorssgldm(get(handles.typeA,'userdata'));
Tv = gettargetvectorssgldm(lv,get(handles.typeA,'userdata'));

%% train NN using input/target vectors
[F, RulePerf] = TrainFuNN(F, lv, Tv);
set(handles.text1,'userdata',F);
% F now represents the trained network that can be used on any image

msgbox('Network has been trained','Notice','help');
uiwait;
close; % rule performance window

function trainwithrules_Callback(hObject, eventdata, handles)

% trains network with rules for pollen classification

SNN = get(handles.mydata,'UserData'); % make local variable SNN = "global" SNN

%get the network ready

disp('Begun parsing the Network');

SNN = parse3(SNN);

F = CreateFuNN2(SNN); %create the FuNN

```

```

%% load weight into FuNN

randn('state',0);

R = SNN.RuleMatrix*0.5; % quality factor is 0.5

R = R + randn(size(SNN.RuleMatrix))*0.1;

A = SNN.ActionMatrix*0.5;

A = A + randn(size(SNN.ActionMatrix))*0.1;

F = ModifyRuleMatrix(F, R);

F = ModifyActionMatrix(F, A);

F = ModifyEpochs(F, SNN.Epochs);

disp('Network ready');


lv = getinputvectorssgldm(get(handles.typeA,'userdata'));
Tv = gettargetvectorssgldm(lv,get(handles.typeA,'userdata'));


%% train NN using input/target vectors

[F, RulePerf] = TrainFuNN(F, lv, Tv);

set(handles.text1,'userdata',F);

% F now represents the trained network that can be used on any image


msgbox('Network has been trained','Notice','help');

uiwait;

close; % rule performance window

% --- Executes on button press in saveTrainedNetwork.
function saveTrainedNetwork_Callback(hObject, eventdata, handles)
%save the trained network using the variable NN

NN = get(handles.text1,'userdata');

if ~isempty(NN)

    [fname,fpath] = uiputfile('Trained_NN.mat','Save Trained Network');

    save ([fpath,fname], 'NN');
else

    msgbox('The network has not yet been trained','Error','warn');

    uiwait;

```

```

end

% -----Classify functions-----
% --- Executes on button press in classifySingle.
function classifySingle_Callback(hObject, eventdata, handles)

% load pollen image

[filename,path] = uigetfile('almages\*.tif','Load Image to be classified');

Pimage = imread([path,filename]);
set(handles.axes1,'visible','On');
image(Pimage);
colormap(gray(256));

%find the interesting info for the image
[ycentre,xcentre,horizmaxrun,vertmaxrun,BW2,area,pts,avgPixelIntensity] =
findcentremass(Pimage,2);
[circ,rmean] = findcircularity(ycentre,xcentre,vertmaxrun,horizmaxrun,BW2);
smimage = findCentralArea(Pimage,rmean,ycentre,xcentre);
[energy, homogeneity, entropy, inertia, correlation] = getSGLDMdata(smimage);

%formulate network and input vectors for type A (All)
load('TrainedNetworks\PollenTypeAll.mat');
lv = [area; circ ; inertia; correlation];
result = SimFuNN(NN, lv) %display

if result > 0.25 % type B

    %formulate network and input vectors for type B
    load('TrainedNetworks\PollenTypeB.mat');
    lv = [energy;homogeneity;entropy;inertia;correlation;circ;area];
    result = SimFuNN(NN, lv) %display

    if result > 0.67
        msgbox('This pollen type is "Zm"', 'Result', 'help');
        uiwait
    elseif result < 0.3
        msgbox('This pollen is type "D"', 'Result', 'help');
        uiwait
    else
        %formulate network and input vectors for type typeD
        load('TrainedNetworks\PollenTypeDnr.mat');
        lv = [energy;homogeneity;entropy;inertia;correlation;circ;area];
        result = SimFuNN(NN, lv) %display
        if result > 0.11
            msgbox('This pollen type is "Pr"', 'Result', 'help');
            uiwait
        else
            msgbox('This pollen type is "Ad"', 'Result', 'help');
            uiwait
        end
    end
else % else type C
    msgbox('This pollen is type "C"', 'Result', 'help');
    uiwait
end

% --- Executes on button press in classifyGroup.

```



```

function classifyGroup_Callback(hObject, eventdata, handles)
% classify a set group of pollen
maxptypes = 40;
numofpollen = 30;
r=""; dname="";
%reset variables
Zm(maxptypes) = 0; typeF(maxptypes)=0; Ta(maxptypes)=0;
Pr(maxptypes)=0; Ad(maxptypes) =0; typeC(maxptypes)=0;
for pollentype = 1:maxptypes
    for i = (1):(numofpollen)
        name = getAllPollen(pollentype,i);
        Pimage = imread(name);

        %find the interesting info for the image
        [ycentre,xcentre,horizmaxrun,vertmaxrun,BW2,area1,area2,avgPixelIntensity] =
findcentremass(Pimage,3);
        [circ,rmean] = findcircularity(ycentre,xcentre,vertmaxrun,horizmaxrun,BW2);
        smimage = findCentralArea(Pimage,rmean,ycentre,xcentre);
        [energy, homogeneity, entropy, inertia, correlation] = getSGLDMdata(smimage);

        %formulate network and input vectors for type A (All)
        nnA = load('TrainedNetworks\PollenTypeAll.mat');
        lv = [area1; circ ; inertia; correlation];
        result = SimFuNN(nnA.NN, lv);

        if result > 0.25 % type B
            %formulate network and input vectors for type B
            load('TrainedNetworks\PollenTypeB.mat');
            lv = [energy;homogeneity;entropy;inertia;correlation;circ;area2];
            result = SimFuNN(NN, lv);
            if result > 0.67
                Zm(pollentype) = Zm(pollentype) + 1;
            elseif result < 0.3
                %formulate network and input vectors for type E
                load('TrainedNetworks\PollenTypeE.mat');
                lv = [energy;homogeneity;entropy;inertia;correlation;circ;area2];
                result = SimFuNN(NN, lv);
                if result > 0.7
                    Ta(pollentype)=Ta(pollentype)+1;
                else
                    typeF(pollentype)=typeF(pollentype)+1;
                end
            else
                %formulate network and input vectors for type typeD
                load('TrainedNetworks\PollenTypeDnr.mat');
                lv = [energy;homogeneity;entropy;inertia;correlation;circ;area2];
                result = SimFuNN(NN, lv);
                if result > 0.11
                    Pr(pollentype)=Pr(pollentype)+1;
                else
                    Ad(pollentype)=Ad(pollentype)+1;
                end
            end
        else % else type C
            typeC(pollentype) =typeC(pollentype)+1;
        end %end if else
    end %for i
    dname = [dname,name(9:10)];
end %for pollentype
% display results

```

```

x="";
for j=1:maxptypes
    disp(['Pollen Type ',dname((j*2)-1:j*2),': Type C=',num2str(typeC(j)), ' Type F=',...
        num2str(typeF(j)), ' Ad=',num2str(Ad(j)), ' Pr=',num2str(Pr(j)),...
        ' Ta=',num2str(Ta(j)), ' Zm=',num2str(Zm(j))]);
end

% --- Executes on button press in pushbutton6 "use trained network".
function pushbutton6_Callback(hObject, eventdata, handles)

F = get(handles.text1,'userdata');

classifyPollenSGLDM(F,get(handles.typeA,'userdata'));

% --- Executes on button press in close.
function close_Callback(hObject, eventdata, handles)
close;

% --- Executes on button press in typeA.
function typeA_Callback(hObject, eventdata, handles)
off = [handles.typeB,handles.typeC,handles.typeD,handles.typeE,handles.typeF];
mutual_exclude(off);
set(handles.typeA,'userdata','A');

% --- Executes on button press in typeB.
function typeB_Callback(hObject, eventdata, handles)
off = [handles.typeA,handles.typeC,handles.typeD,handles.typeE,handles.typeF];
mutual_exclude(off);
set(handles.typeA,'userdata','B');

% --- Executes on button press in typeC.
function typeC_Callback(hObject, eventdata, handles)
off = [handles.typeB,handles.typeA,handles.typeD,handles.typeE,handles.typeF];
mutual_exclude(off);
set(handles.typeA,'userdata','C');

% --- Executes on button press in typeD.
function typeD_Callback(hObject, eventdata, handles)
off = [handles.typeB,handles.typeC,handles.typeA,handles.typeE,handles.typeF];
mutual_exclude(off);
set(handles.typeA,'userdata','D');

% --- Executes on button press in typeE.
function typeE_Callback(hObject, eventdata, handles)
off = [handles.typeB,handles.typeC,handles.typeD,handles.typeA,handles.typeF];
mutual_exclude(off);
set(handles.typeA,'userdata','E');

% ---subfunction-----
function mutual_exclude(off)
set(off,'Value',0)

% --- Executes on button press in typeF.
function typeF_Callback(hObject, eventdata, handles)
off = [handles.typeB,handles.typeC,handles.typeD,handles.typeA,handles.typeE];

```

```
mutual_exclude(off);
set(handles.typeA,'userdata','F');
```

Functions for Print Generator GUI

```
function varargout = PrintGenerator1(varargin)
% this Gui allows Netowrk settings to be printed
if nargin == 0 % LAUNCH GUI

    fig = openfig(mfilename,'reuse');

    % Generate a structure of handles to pass to callbacks, and store it.
    handles = guihandles(fig);
    guidata(fig, handles);
%-----
% Set Up page-design from here
%-----

load ('tempqpzm10');
set(handles.text33,'userdata',SNN);%text33 userdata contains 'SNN' variable

set(handles.datafilename,'string',SNN.DataFileName);
set(handles.learnrate,'string',SNN.LearningRate);
set(handles.saturation,'string',SNN.Saturation);
set(handles.typeofnetwork,'string',SNN.TypeOfNetwork);
set(handles.totalrulenodes,'string',SNN.TotalRuleNodes);
set(handles.numinputneurons,'string',SNN.NumberOfInputNeurons);
set(handles.momentum,'string',SNN.Momentum);
set(handles.epochs,'string',SNN.Epochs);
set(handles.numoutputneurons,'string',SNN.NumberOfOutputNeurons);
set(handles.goal,'string',SNN.Goal);
set(handles.windowsizevert,'string',SNN.WFVSize);
set(handles.windowsizehoriz,'string',SNN.WFHSize);
set(handles.iswindowfilter,'string',SNN.IsWindowFilter);
set(handles.numfuzzyrules,'string',SNN.FuzzyMF);
set(handles.fuzzyrules,'string',SNN.FuzzyMFDEF);
set(handles.defuzzyrules,'string',SNN.DefuzzyMFDEF);
set(handles.usernotes,'string',SNN.UserNotes);

displaygraphs(handles); % display all rules in greyscale

%display the rule line
for i = 1:SNN.NumberRules
    rules{i} = ['Rule ',num2str(i),': ',getrule(SNN.Rule,i,SNN.NumberRules)];
end
set(handles.ruledisplay,'string',rules);
%-----

    if nargout > 0
        varargout{1} = fig;
    end

elseif ischar(varargin{1}) % INVOKE NAMED SUBFUNCTION OR CALLBACK

    try
        if (nargout)
            [varargout{1:nargout}] = feval(varargin{:}); % FEVAL switchyard
```

```

        else
            feval(varargin{:}); % FEVAL switchyard
        end
    catch
        disp(lasterr);
    end

end

% -----
function varargout = printbutton_Callback(h, eventdata, handles, varargin)

%make print and cancel buttons invisible
set(handles.printbutton,'visible','Off');
set(handles.cancelbutton,'visible','Off');
if ~strcmp((get(handles.usernotes,'userdata')), 'true') % if no usernotes
    set(handles.usernotes,'visible','Off');
else
    SNN = get(handles.text33,'userdata');
    SNN.UserNotes = get(handles.usernotes,'string');
    set(handles.usernotes,'visible','On');
    save ('tempqpzm10','SNN');
end
%need the next line ?
% set(gcf,'PaperPositionMode','auto')
print;
disp('Leaving Print-report-Generator (printed)');
close

% -----
function varargout = cancelbutton_Callback(h, eventdata, handles, varargin)
q = (get(handles.usernotes,'userdata'));
if strcmp(q,'true') % if usernotes
    qans = questdlg('Do you want to save usernotes?','Save ?','Yes','No','Yes');
    if strcmp(qans,'Yes')
        SNN = get(handles.text33,'userdata');
        SNN.UserNotes = get(handles.usernotes,'string');
        save ('tempqpzm10','SNN');
        disp(['saving usernotes: ',SNN.UserNotes]);
    end
end
disp('Leaving Print-report-Generator (did not print)');
close

% -----
function varargout = usernotes_Callback(h, eventdata, handles, varargin)

set(handles.usernotes,'userdata','true');
% may wish to create an extra field if we want to save usernotes
% SNN.usernotes = get(handles.usernotes,'String');

% -----
function displaygraphs(handles)
% display all rules in greyscale on axes 1 -> SNN.NumberRules
SNN = get(handles.text33,'userdata');
if SNN.NumberRules == 0
    %do nothing
else

```

```

for currentrule = 1:SNN.NumberRules
    tline2 = getrule(SNN.Rule,currentrule,SNN.NumberRules); % get rule number from
SNN.Rule
    % make current-axes current
    if currentrule==1 axes(handles.axes1); end
    if currentrule==2 axes(handles.axes2); end
    if currentrule==3 axes(handles.axes3); end
    if currentrule==4 axes(handles.axes4); end
    if currentrule==5 axes(handles.axes5); end
    if currentrule==6 axes(handles.axes6); end
    if currentrule==7 axes(handles.axes7); end
    if currentrule==8 axes(handles.axes8); end
    if currentrule==9 axes(handles.axes9); end

    % create a default image 'gsi' (all black) of the same size as filterwindow
    gsisafe = []; % reset gsisafe
    gsisafe(SNN.WFVSize,SNN.WFHSize) = 0;% set dimensions
    gsisafe = gsisafe + 257; % set all values to over proper range*****
    plot(1);% ****clear the graph*****

    %*****display current rule in greyscale*****
    [currentword, tline2] = ExtractWord(tline2); % get 'if' word of line
    [currentword, tline2] = ExtractWord(tline2); %get 'first' rule-part
    while ~isempty(tline2)
        switch currentword
            case 'Then'
                % [result, tline2] = ExtractWord(tline2); % get result part of rule
                % do nothing yet? display result part of rule ?
                break
            otherwise %read the rule
                [cellnum, ruletype, op] = SplitWord(currentword);
                [pos,i] = ExtractNumber(cellnum); % e.g. i = 'I', pos = 6
                [MFDEFpos,m] = ExtractNumber(ruletype); % e.g. m = 'M', MFDEFpos = 2
                numdown = ceil(pos/SNN.WFHSize); %rounds up
                numacross = mod(pos, SNN.WFHSize);
                if numacross == 0 %ie at the end of the row
                    numacross = SNN.WFHSize;
                end
                intensity = SNN.FuzzyMFDEF(MFDEFpos); % read the intensity value
                gsisafe(numdown,numacross) = intensity*255;
            end %switch
        [currentword, tline2] = ExtractWord(tline2); % get next word of line
    end %while

    % display the built up image map (array with values from 0 - 255)
    image(gsisafe);
    colormap(gray(256));

    % display 'x' where there is no rule
    horixadjust = SNN.WFHSize * 0.04;
    for x = 1:SNN.WFHSize
        for y = 1:SNN.WFVSize
            if gsisafe(y,x) == 257 % ie. if no rule has been written in this pos
                text(x - horixadjust,y,'x','FontSize',12,...
                    'HorizontalAlignment','left');
                %disp(['currentrule ',num2str(currentrule),' ',num2str(x),num2str(y)]);
            end
        end
    end
end

```

```

    end %if else
end % for loop

```

Functions for SetUserOptions GUI

```

function varargout = setUserOptions(varargin)
% The main GUI for the Super FuNN application.
% From here, all settings and images etc can be loadad, and the other
% related GUI's called.
% Author Greg Todd. February, 2003.

if nargin == 0 % LAUNCH GUI

    fig = openfig(mfilename,'reuse');

    % Generate a structure of handles to pass to callbacks, and store it.
    handles = guihandles(fig);
    guidata(fig, handles);

    %----- opening details-----
    disp('SuperFuNN Started');
    % disable certain menus
    % file menu
    set(handles.loadnongrayscale,'enable','off');
    set(handles.printreport,'enable','off');
    set(handles.Untitled_3,'enable','off'); %save trained
    set(handles.loadimages,'enable','off');
    set(handles.saveas,'enable','off');
    set(handles.savesettings,'enable','off');
    %option menu
    set(handles.reselectcluster,'enable','off');
    set(handles.setvariables,'enable','off');
    set(handles.setrules,'enable','off');
    set(handles.runnew,'enable','off'); %network
    set(handles.runtrained,'enable','off'); %network
    set(handles.batchrun,'enable','off');
    set(handles.runNWFnetwork,'enable','off');

    %-----end of opening details-----
    if nargout > 0
        varargout{1} = fig;
    end

elseif ischar(varargin{1}) % INVOKE NAMED SUBFUNCTION OR CALLBACK

    try
        if (nargout)
            [varargout{1:nargout}] = feval(varargin{:}); % FEVAL switchyard
        else
            feval(varargin{:}); % FEVAL switchyard
        end
    catch
        disp(lasterr);
    end

end

%-----
function enableMenu(handles,level)

```

```

% enables appropriate pull down menus based on level
if level == 1 % variables loaded
    set(handles.printreport,'enable','on');
    set(handles.loadimages,'enable','on');
    set(handles.saveas,'enable','on');
    set(handles.savesettings,'enable','on');
    set(handles.setvariables,'enable','on');
    set(handles.setrules,'enable','on');
elseif level == 2 %netowrk can now run, images loaded
    set(handles.reselectcluster,'enable','on');
    set(handles.runnew,'enable','on');
    set(handles.batchrun,'enable','on');
elseif level == 3
    set(handles.runtrained,'enable','on');
    set(handles.Untitled_3,'enable','on');%save trained
elseif level == 4 %NWF netowrk
    set(handles.runNWFnetwork,'enable','on');
    set(handles.batchrun,'enable','off');
end

% Information placeholders (using 'userdata' properties)
% text - 'mydata' contains the SNN structure
% text - 'saveinfo' contains the filename and path data
% text - 'Tvinfo' contains the Tv variable
% text - 'lvinfo' contains the lv variable
% text - 'finfo' contains the 'Timage'
% pushbutton - 'pushbutton1' contains the 'Pimage'
% text - 'text1' contains 'NNtrained'
% -----

% -----UPDATE-----ACCEPT BUTTON-----
function varargout = pushbutton1_Callback(h, eventdata, handles, varargin)
% do error checking, if no errors accept all variables into SNN

SNN = get(handles.mydata,'UserData'); % create a local variable SNN equal to mydata
'userdata'
%set local SNN variable to appropriate interface settings
SNN.ParserVersion = '2.0'; % this variable is not actually needed
SNN.DataFileName = get(handles.datafilename,'String');
SNN.LearningRate = str2num(get(handles.learnrate,'String'));

if get(handles.typeofnetwork,'value') == 1
    SNN.TypeOfNetwork = 'FNN';
else
    SNN.TypeOfNetwork = 'MLP';
end
SNN.Saturation = str2num(get(handles.saturation,'String'));
SNN.TotalRuleNodes = str2num(get(handles.totalrulenodes,'String'));
SNN.NumberOfInputNeurons = str2num(get(handles.numinputnuerons,'String'));
SNN.NumberOfOutputNeurons = str2num(get(handles.numoutputneurons,'String'));
SNN.Momentum = str2num(get(handles.momentum,'String'));
SNN.Epochs = str2num(get(handles.epochs,'String'));
SNN.Goal = str2num(get(handles.goal,'String'));
SNN.WFVSize = get(handles.windowsizevert,'Value'); % because its taken from a pop-up
menu
SNN.WFHSize = get(handles.windowsizehoriz,'Value'); % because its taken from a pop-up
menu
SNN.IsWindowFilter = str2num(get(handles.iswindowfilter,'String'));
if SNN.IsWindowFilter == 0
    enableMenu(handles,4);

```

```

else
    enableMenu(handles,1);
    enableMenu(handles,2);
end
set(handles.mydata,'UserData',SNN); % reset user data to local variable
if (strcmp(displaydata(handles), 'error'))
    %do nothing, leave in set variables mode
else
    hidevariables(handles);
end

% -----CANCEL BUTTON-----
function varargout = pushbutton2_Callback(h, eventdata, handles, varargin)
% leaves 'Set Variables' screen without saving any changes made
hidevariables(handles);

% -----
function hidevariables(handles)
% bring back starting panels and make pop-up menus disappear
set(handles.coverpanel,'Visible','On');
set(handles.covertext,'Visible','On');
set(handles.displayaxesi,'Visible','On');
set(handles.displayaxest,'Visible','On');
set(handles.windowsizevert,'Visible','Off');
set(handles.windowsizehoriz,'Visible','Off');
set(handles.typeofnetwork,'Visible','Off');
if (str2num(get(handles.iswindowfilter,'string')) == 0)
    %set(handles.windowsizehoriz,'visible','off');
    %set(handles.windowsizevert,'visible','off');
    set(handles.text5,'visible','off');
    set(handles.text11,'visible','off');
else
    %set(handles.windowsizehoriz,'visible','on');
    %set(handles.windowsizevert,'visible','on');
    set(handles.text5,'visible','on');
    set(handles.text11,'visible','on');
end

% -----
function varargout = saturation_Callback(h, eventdata, handles, varargin)
currentsaved = 'f'; %indicate data has been changed since last save
set(handles.text2,'Userdata',currentsaved);%save back to 'global' saveinfo

% -----
function varargout = numoutputneurons_Callback(h, eventdata, handles, varargin)
currentsaved = 'f'; %indicate data has been changed since last save
set(handles.text2,'Userdata',currentsaved);%save back to 'global' saveinfo

% -----
function varargout = momentum_Callback(h, eventdata, handles, varargin)
currentsaved = 'f'; %indicate data has been changed since last save
set(handles.text2,'Userdata',currentsaved);%save back to 'global' saveinfo

% -----
function varargout = epochs_Callback(h, eventdata, handles, varargin)
currentsaved = 'f'; %indicate data has been changed since last save
set(handles.text2,'Userdata',currentsaved);%save back to 'global' saveinfo

% -----
function varargout = goal_Callback(h, eventdata, handles, varargin)

```



```

currentsaved = 'f'; %indicate data has been changed since last save
set(handles.text2,'Userdata',currentsaved);%save back to 'global' saveinfo

% -----
function varargout = windowsizevert_Callback(h, eventdata, handles, varargin)
currentsaved = 'f'; %indicate data has been changed since last save
set(handles.text2,'Userdata',currentsaved);%save back to 'global' saveinfo
% update the number of input neurons
SNN = get(handles.mydata,'UserData'); % make local variable SNN = "global" SNN
SNN.NumberOfInputNeurons = get(handles.windowsizehoriz,'Value') *
get(handles.windowsizevert,'Value');
set(handles.numinputnuerons,'string',SNN.NumberOfInputNeurons);
set(handles.mydata,'UserData',SNN);

% -----
function varargout = numinputnuerons_Callback(h, eventdata, handles, varargin)
currentsaved = 'f'; %indicate data has been changed since last save
set(handles.text2,'Userdata',currentsaved);%save back to 'global' saveinfo

% -----
function varargout = totalrulenodes_Callback(h, eventdata, handles, varargin)
currentsaved = 'f'; %indicate data has been changed since last save
set(handles.text2,'Userdata',currentsaved);%save back to 'global' saveinfo

% -----
function varargout = typeofnetwork_Callback(h, eventdata, handles, varargin)
currentsaved = 'f'; %indicate data has been changed since last save
set(handles.text2,'Userdata',currentsaved);%save back to 'global' saveinfo

% -----
function varargout = learnrate_Callback(h, eventdata, handles, varargin)
currentsaved = 'f'; %indicate data has been changed since last save
set(handles.text2,'Userdata',currentsaved);%save back to 'global' saveinfo

% -----
function varargout = windowsizehoriz_Callback(h, eventdata, handles, varargin)
currentsaved = 'f'; %indicate data has been changed since last save
set(handles.text2,'Userdata',currentsaved);%save back to 'global' saveinfo
% update the number of input neurons
SNN = get(handles.mydata,'UserData'); % make local variable SNN = "global" SNN
SNN.NumberOfInputNeurons = get(handles.windowsizehoriz,'Value') *
get(handles.windowsizevert,'Value');
set(handles.numinputnuerons,'string',SNN.NumberOfInputNeurons);
set(handles.mydata,'UserData',SNN);

% -----
function varargout = datafilename_Callback(h, eventdata, handles, varargin)
currentsaved = 'f'; %indicate data has been changed since last save
set(handles.text2,'Userdata',currentsaved);%save back to 'global' saveinfo

% -----
function varargout = File_Callback(h, eventdata, handles, varargin)
%no action, just indicates the file menu has been clicked
% -----
function varargout = options_Callback(h, eventdata, handles, varargin)
%no action, just indicates the options menu has been clicked
% -----
% buttons and menu function callbacks*****
% -----
function varargout = setvariables_Callback(h, eventdata, handles, varargin)

```

```

%check that saved settings or 'new' settings have been loaded
if isempty(get(handles.mydata,'userdata'))
    msgbox('You must either select "File->Load Settings" or "File->New Settings" before you
    can adjust variables','Note','warn');
else
    SNN = get(handles.mydata,'UserData'); % make local variable SNN = "global" SNN
    displaydata(handles);
    %get rid of starting panels and bring up pop-up menus
    set(handles.coverpanel,'Visible','Off');
    %set(handles.covertext,'Visible','Off');
    set(handles.windowsizevert,'Visible','on');
    set(handles.windowsizehoriz,'Visible','on');
    set(handles.typeofnetwork,'Visible','On');
end

% -----
function varargout = loadsettings_Callback(h, eventdata, handles, varargin)

% load 'mat' file containing SNN data
[fname,fpath] = uigetfile('Settings\*.mat','Settings File');
load([fpath,fname]);
info.fname = fname;
info.fpath = fpath;
set(handles.saveinfo,'UserData',info);% for use later when saving settings
set(handles.mydata,'Userdata',SNN); %set SNN to userdata so that all functions can now
reference SNN
displaydata(handles);
currentsaved = 't';
set(handles.text2,'Userdata',currentsaved);%save back to 'global' saveinfo
enableMenu(handles,1);
if SNN.IsWindowFilter == 0
    enableMenu(handles,4);
end

disp('Set-User-Options Settings loaded');

% -----
function varargout = savesettings_Callback(h, eventdata, handles, varargin)
% save file settings using existing filename
SNN = get(handles.mydata,'UserData'); % make local variable SNN = "global" SNN
info = get(handles.saveinfo, 'UserData'); %get save info i.e filename & path
fpath = info.fpath;
fname = info.fname;
save ([fpath,fname], 'SNN');%save current data to file
currentsaved = 't'; %indicate data has been saved since last change
set(handles.saveinfo,'Userdata',info);%save back to 'global' saveinfo
set(handles.text2,'Userdata',currentsaved);
% -----
function varargout = iswindowfilter_Callback(h, eventdata, handles, varargin)
if (str2num(get(handles.iswindowfilter,'string')) == 0)
    set(handles.windowsizehoriz,'visible','off');
    set(handles.windowsizevert,'visible','off');
    set(handles.text5,'visible','off');
    set(handles.text11,'visible','off');
else
    set(handles.windowsizehoriz,'visible','on');
    set(handles.windowsizevert,'visible','on');
    set(handles.text5,'visible','on');
    set(handles.text11,'visible','on');
end
end

```

```

% -----
function varargout = setrules_Callback(h, eventdata, handles, varargin)
% saves the current settings to file, then passes file info (name & path)
% to a 'datainput' GUI (which manipulates other parts of the file), waits
% for the return from 'datainput' then reloads the settings files

if isempty(get(handles.mydata,'userdata'))
    msgbox('You must either select "File->Load Settings" or "File->New Settings" before you
    can adjust variables','Note','warn');
else
    SNN = get(handles.mydata,'UserData'); % make local variable SNN = "global" SNN
    %info = get(handles.saveinfo, 'UserData'); %get save info i.e filename & path
    %fpath = info.fpath;
    %fname = info.fname;
    %save ([fpath,fname], 'SNN'); %save current data to file (for reading by datainput gui)
    save ('tempqpzm10', 'SNN');
    %handles.info = info; % Save the data to the handles structure %handles.MSNN = MSNN;
    %guidata(h,handles); % Stored in the application data of the figure
    %SNN = DataInput(handles); % Call the DataInput gui and pass it the handles structure
    %which contains the 'SNN' data
    DataInput3(handles);
    uiwait
    %load([fpath,fname]); %reload updated file+++++++
    load('tempqpzm10');
    disp('Control back to Set-user-options');
    disp(['Rule as read by Set-user-options ',SNN.Rule]);
    %set SNN to the userdata of the object 'data' so that all functions can now reference SNN
    set(handles.mydata,'UserData',SNN);

end %if else
%don't need to reset values as these are not changed by the other GUI's

% -----
function quitset_Callback(hObject, eventdata, handles)
% quit out of the program
currentsaved = get(handles.text2,'userdata');
if ( currentsaved ~= 't') % if data has not been saved
    %check that most recent changes have been saved
    qanswer = questdlg('Do you want to save changes?',...
        'Save Changes?','Yes','No','No');
    if strcmp(qanswer,'Yes')
        SNN = get(handles.mydata,'UserData'); % make local variable SNN = "global" SNN
        [fname,fpath] = uiputfile('*.mat','Settings File');
        save ([fpath,fname], 'SNN');
        disp('At saving stage...');
        disp(SNN.Rule);
        close;
    else
        close; % closes the GUI without saving
    end
else
    close; % closes the GUI without saving
end
% -----
function saveas_Callback(hObject, eventdata, handles)
% save data under a new filename
SNN = get(handles.mydata,'UserData'); % make local variable SNN = "global" SNN
% save 'mat' file containing SNN data
[fname,fpath] = uiputfile('Settings\*.mat','Settings File');

```

```

save ([fpath,fname],'SNN');
%update saveinfo in case user continues to change variables...
info.fname = fname;
info.fpath = fpath;
currentsaved = 't'; %indicate that the data has been saved up-to-date
set(handles.saveinfo,'Userdata',info);
set(handles.text2,'userdata',currentsaved);

% -----
function loadimages_Callback(hObject, eventdata, handles)
loadtheimages(handles);

% -----
function [r, map]=getspecialin(fname,fpath)
[P,pmap] = imread([fpath,fname]);
P = double(P);
r = P;
map = pmap;

% -----
function loadtheimages(handles)
if isempty(get(handles.mydata,'userdata'))
    msgbox('Please either select "File->Load Settings" or "File->Set New" data before loading
images','Warning','warn');
    uiwait;
else

Pf = figure(1);
screen = get(0,'ScreenSize'); % left, bottom, width, height
pos1 = [1,1,screen(3)/2,screen(4)*5/6];
pos2 = [screen(3)/2,1,screen(3)/2,screen(4)*5/6];
set(Pf,'Position',pos1); % position figure
[fname,fpath,r1,map1,name1] = getInputImage2(Pf); % load input image
Pimage = getspecialin(fname,fpath); % subfunction listed above

%check to see if illumination process is desired
%ma = questdlg('Do you wish to use the "Illumination Procedure"','Image
transform','Yes','No','No');
%if strcmp(ma,'Yes')
%    CorrectIllumination(Pimage);
%    image(Pimage);
%end
% set Pimage to userdata to make it 'global'
set(handles.text17,'userdata',Pimage);

Tf = figure(2);
set(Tf,'Position',pos2); % position figure
[fname,fpath, r2,map2, name2] = getTargetImage2(Tf); % load target image
Timage = getspecialin(fname,fpath); % subfunction listed above
% set Timage to userdata to make it 'global'
set(handles.finfo,'userdata',Timage);

SNN = get(handles.mydata,'UserData'); % make local variable SNN = "global" SNN

ma = questdlg('Do you wish to select multiple areas?','Cluster Area','Yes','No','No');
if strcmp(ma,'Yes')
    na = inputdlg('How many cluster area"s do you wish to select ?','Cluster Area Selection',[1])
    answer = str2num(cell2mat(na));
    %uiwait;
else

```

```

    answer = 1; % select only 1 area
end
msgbox('Please select the "Cluster Area" on either the input image or the target image',...
    'Select Cluster','help');
uiwait;
figure(2); % push figure 2 towards the front of the display again

% pick sub areas in image to train (changed from F.WFHSize to SNN.WFHSize)
[lv, Tv] = pickCluster2(answer, Pimage, r1, map1, Timage, r2, map2, Pf, [SNN.WFHSize
SNN.WFVSize]);

set(handles.lvinfos,'userdata',lv); % set lv to userdata so it can 'global'
set(handles.tvinfos,'userdata',Tv); % set Tv to userdata so it can 'global'
close Figure No. 1;
close Figure No. 2;
disp('Images loaded and cluster area selected');
% show that images have been loaded
set(handles.specialinfo,'string',{'Loaded Input Image : ',name1});
set(handles.specialinfo,'visible','on');
set(handles.specialinfo2,'string',{'Loaded Target Image : ',name2});
set(handles.specialinfo2,'visible','on');
axes(handles.displayaxest); % make this axes current
set(handles.displayaxest,'Visible','On');
image(Timage);
axes(handles.displayaxesi); % make this axes current
set(handles.displayaxesi,'Visible','On');
image(r1);
colormap(map1);
enableMenu(handles,2);
end %if else

% -----
function varargout = printreport_Callback(h, eventdata, handles, varargin)

disp('Calling print generator GUI');
SNN = get(handles.mydata,'UserData'); % make local variable SNN = "global" SNN
save ('tempqpzm10','SNN');
PrintGenerator1; % call printgenerator GUI
uiwait;
% reload file and update data (i.e. may have added user-notes...)
load('tempqpzm10');
set(handles.mydata,'Userdata',SNN);

% -----
function newsettings_Callback(hObject, eventdata, handles)
% Create new settings -> give all variables some DEFAULT values
% adjust these as necessary
SNN.DataFileName = 'new data';
SNN.FuzzyMF = 2;
SNN.FuzzyMFDEF = [0 1];
SNN.DefuzzyMF = 2;
SNN.DefuzzyMFDEF = [0 1];
SNN.LearningRate = 0.8000;
SNN.TypeOfNetwork = 'FNN';
SNN.Saturation = 1;
SNN.TotalRuleNodes = 10;
SNN.NumberOfInputNeurons = 4;
SNN.NumberOfOutputNeurons = 1;
SNN.Momentum = 0.8000;
SNN.Epochs = 200;

```

```

SNN.Goal = 0;
SNN.WFVSize = 2;
SNN.WFHSize = 2;
SNN.IsWindowFilter = 1; %change this to a 'yes' or 'no' answer ?
SNN.NumberRules = 1;
SNN.Rule = 'If I1=M1 Then O1=M1';
SNN.UserNotes = 'Write user notes here';
%SNN.F = ""; % indicates netowrk has not been trained yet

set(handles.mydata,'Userdata',SNN); %set default settings to 'global' SNN
%displaydata(handles);
currentsaved = 't';
set(handles.text2,'userdata',currentsaved);
enableMenu(handles,1);
disp('Set-User-Options Default Settings loaded');

% -----
function helpmenu_Callback(hObject, eventdata, handles)
% this indicates the help menu has been accessed
% -----
% this calls the help web pages
function generalhelp_Callback(hObject, eventdata, handles)
HelpPath = which('test_help.html');
web(HelpPath);

%-----
function r = displaydata(handles)
% display data in the SNN variable to the screen
% if all SNN data is valid, returns 'ok' otherwise returns 'error'
SNN = get(handles.mydata,'userdata');
result = 'ok';

if strcmp(SNN.TypeOfNetwork,'FNN')
    set(handles.typeofnetwork,'value',1);
elseif strcmp(SNN.TypeOfNetwork,'MLP')
    set(handles.typeofnetwork,'value',2);
else
    msgbox('File contains incorrect "Type of Network" value, it will be automatically adjusted to "FNN"', 'Warning', 'warn');
    uiwait;
    set(handles.typeofnetwork,'value',1);
end
set(handles.datafilename,'String',SNN.DataFileName);
set(handles.learnrate,'String',SNN.LearningRate);
set(handles.saturation,'String',SNN.Saturation);
set(handles.totalrulenodes,'String',SNN.TotalRuleNodes);
set(handles.numoutputneurons,'String',SNN.NumberOfOutputNeurons);
set(handles.momentum,'String',SNN.Momentum);
set(handles.epochs,'String',SNN.Epochs);
set(handles.goal,'String',SNN.Goal);

if (SNN.IsWindowFilter == 0)
    set(handles.windowsizevert,'Visible','off');
    set(handles.windowsizehoriz,'Visible','off');
else
    %set(handles.windowsizevert,'Visible','on');
    %set(handles.windowsizehoriz,'Visible','on');
    set(handles.windowsizevert,'Value',SNN.WFVSize); % because its displayed in a pop-up
menu

```

```

    set(handles.window_sizehoriz,'Value',SNN.WFHSize); % because its displayed in a pop-up
    menu
end

set(handles.numinputnuerons,'String',SNN.NumberOfInputNeurons);
if ((SNN.WFVSize * SNN.WFHSize) ~= SNN.NumberOfInputNeurons) & (SNN.IsWindowFilter
~= 0)
    msgbox('The number of input neurons does not match the filter window size','Error','Warn');
    result = 'error';
end
set(handles.iswindowfilter,'String',SNN.IsWindowFilter);
if (isempty(get(handles.text17,'userdata'))))
    set(handles.displayaxesi,'visible','off');
end
if (isempty(get(handles.finfo,'userdata'))))
    set(handles.displayaxest,'visible','off');
end

r = result;

% -----
function batchrun_Callback(hObject, eventdata, handles)

%load 'SNN', 'Iv', 'Tv', Pimage and Timage data
SNN = get(handles.mydata,'UserData'); % make local variable SNN = "global" SNN
Iv = get(handles.Ivinfo,'userdata'); % make local variable Iv = "global" Iv
Tv = get(handles.Tvinfo,'userdata'); % make local variable Tv = "global" Tv
Pimage = get(handles.text17,'userdata'); %make local variable Pimage = "global" Pimage
Timage = get(handles.finfo,'userdata');

if (SNN.IsWindowFilter == 1)

% make sure we can't run the network without Settings and images etc etc
if isempty(get(handles.text17,'userdata'))
    msgbox({'Input image has not been loaded';'The Network could not run'},'Error','warn');
    uiwait;
elseif isempty(get(handles.finfo,'userdata'))
    msgbox({'Target image has not been loaded';'The Network could not run'},'Error','warn');
    uiwait;
elseif isempty(get(handles.mydata,'userdata')) % actually this check is redundant...
    msgbox({'Please load or "set new" data before loading images';'The Network could not
run'},'Warning','warn');
    uiwait;
elseif (((SNN.WFHSize * SNN.WFVSize) ~= (SNN.NumberOfInputNeurons)) |
(SNN.NumberOfInputNeurons == 0))
    msgbox({'The "number of input neurons" or "Window Filter size" do not correspond';'The
Network could not run'},'Error','warn');
    uiwait;
else % Call the batchprocess GUI
    save ('tempqpzm10','SNN','Iv','Tv','Pimage','Timage');

    batchprocess(handles);
    uiwait
    load('tempqpzm10');
    disp('Control back to Set-user-options');
    %set SNN to the userdata of the object 'data' so that all functions can now reference SNN
    set(handles.mydata,'UserData',SNN);

end % if else

```

```

else % if not window filter network
    save('tempqpzm10','SNN','lv','Tv','Pimage','Timage');
    batchprocess(handles);
    uiwait
    load('tempqpzm10');
    disp('Control back to Set-user-options');
    %set SNN to the userdata of the object 'data' so that all functions can now reference SNN
    set(handles.mydata,'Userdata',SNN);
end

% -----
function runnetwork_Callback(hObject, eventdata, handles)

% -----
% don't need to offer to save output image as it's already offered
% via the matlab figure window...

% -----
% train and run a NN
function runnew_Callback(hObject, eventdata, handles)
% Here the actual processing is done
SNN = get(handles.mydata,'UserData'); % make local variable SNN = "global" SNN

if (SNN.IsWindowFilter == 1) % then run window filter network
% must put ***more*** error checking in here !!!!!
% make sure we can't run the network without Settings and images etc etc
if isempty(get(handles.text17,'userdata'))
    msgbox({'Input image has not been loaded';'The Network could not run'},'Error','warn');
    uiwait;
elseif isempty(get(handles.finfo,'userdata'))
    msgbox({'Target image has not been loaded';'The Network could not run'},'Error','warn');
    uiwait;
elseif isempty(get(handles.mydata,'userdata')) % actually this check is redundant...
    msgbox({'Please load or "set new" data before loading images';'The Network could not
run'},'Warning','warn');
    uiwait;
elseif (((SNN.WFHSIZE * SNN.WFVSIZE) ~= (SNN.NumberOfInputNeurons)) |
(SNN.NumberOfInputNeurons == 0))
    if (SNN.IsWindowFilter == 1)
        msgbox({'The "number of input neurons" or "Window Filter size" do not correspond';'The
Network could not run'},'Error','warn');
        uiwait;
    end
else % RUN THE NETWORK

% first we must parse the 'rules' to produce the updated 'SNN.RuleMatrix'
% and the 'SNN.ActionMatrix'
disp('Begun parsing the Network');
SNN = parse3(SNN);
disp('Finished parsing the Network');

%load 'lv', 'Tv', Pimage and Timage data
lv = get(handles.lvinfo,'userdata');% make local variable lv = "global" lv
Tv = get(handles.tvinfo,'userdata');% make local variable Tv = "global" Tv
Pimage = get(handles.text17,'userdata'); %make local variable Pimage = "global" Pimage
Timage = get(handles.finfo,'userdata');% make local variable Timage = "global" Timage

```



```

F = CreateFuNN2(SNN); %create the FuNN
%% load weight into FuNN
randn('state',0);
R = SNN.RuleMatrix*0.5; % quality factor is 0.5
R = R + randn(size(SNN.RuleMatrix))*0.1;
A = SNN.ActionMatrix*0.5;
A = A + randn(size(SNN.ActionMatrix))*0.1;
F = ModifyRuleMatrix(F, R);
F = ModifyActionMatrix(F, A);

%% modify the number of epochs to be used * optional
% old code was F = ModifyEpochs(F, 300); making 300 epochs everytime...
F = ModifyEpochs(F, SNN.Epochs); % uses 300 epochs...Make this variable??

disp('Running the Network');
%% train NN using input/target vectors
[F, RulePerf] = TrainFuNN(F, Iv, Tv);
% F now represents the trained network that can be used on any image

%% process input image using trained FuNN
RuleO = ProcessImageUsingFuNN(Pimage, F);
disp('size of input image');
disp(size(Pimage));
disp('size of target image');
disp(size(Timage));
disp('size of output image');
disp(size(RuleO));
[height, width] = size(RuleO);
%if height < width

%% crop target image to same size as output image for error calculation purposes
NT = CropTargetToOutput(Timage, RuleO, [F.WFVSize F.WFHSize]);
disp('size of cropped target');
disp(size(NT));

%% create new window to display output image
output_handle = figure('name','Output'); %not needed
image(RuleO);
colormap(gray(256));
disp('Network Finished');

% error calculations iris edge --[fp, fn, e]=calcerror2(Target, Output)
[fp fn err percerr] = calcerror2( NT, RuleO);
%imsave(NT,'almages\newtarget.bmp','bmp');
%imsave(RuleO,'almages\output.bmp','bmp');

% save performance data and output graph in SNN variable
SNN.RuleO = RuleO; % output image
SNN.RulePerf = RulePerf; % performance data
SNN.F = F; % the trained neural network
set(handles.mydata,'UserData',SNN);
%indicate current settings have been trained into a NN
set(handles.text1,'userdata','NNtrained');

% error calculations RMS
RMS = sqrt(mse(NT - RuleO));

% display error calculations
disp({'RMS = ',num2str(RMS)});

```

```

msgbox(['The RMS value was ',num2str(RMS),' with ',[num2str(fp),' false positives,'],[num2str(fn),' false negatives '],[num2str(err),' errors and '],[num2str(percerr),' percent errors'] },...
'Error Calculation','help');
uiwait
%offer chance to post process image
qanswer = questdlg('Do you want to use a post processing procedure?',...
'Post Process?','Yes','No','No');
if strcmp(qanswer,'Yes')
    qanswer = questdlg('Do you want to save the pre processed image?',...
'Save Image?','Yes','No','No');
    if strcmp(qanswer,'Yes')
        RuleO = uint8(RuleO);
        [fname,fpath] = uiputfile('almages\*.bmp','Pre-Processed Output File');
        imwrite(RuleO,[fpath,fname,'.bmp'],'bmp');
        RuleO = double(RuleO);
    end
    disp('Started post processing');
    RuleO = 255 - RuleO;% prepare image for post processing
    finalim = findPathIris(RuleO);
    figure;
    imshow(finalim);
    qanswer = questdlg('Do you want to save the post processed image?',...
'Save Image?','Yes','No','No');
    if strcmp(qanswer,'Yes')
        [fname,fpath] = uiputfile('almages\*.bmp','Processed Output File');
        imwrite(finalim,[fpath,fname,'.bmp'],'bmp');
    end
end
enableMenu(handles,3);

end % (isempty) if else

else
    msgbox('Please use batch-run for non window filter networks','Notice','help');
    uiwait;

end

% apply an already trained NN
% -----
function runtrained_Callback(hObject, eventdata, handles)

if strcmp(get(handles.text1,'userdata'),'NNtrained')
    qans = questdlg('Use current trained network?','Settings','Yes','No','No');
    end

if strcmp(get(handles.text1,'userdata'),'NNtrained') & strcmp(qans,'Yes')
    SNN = get(handles.mydata,'userdata');
    F = SNN.F;
else
    % load trained network
    [fname,fpath] = uigetfile('*.mat','Load Trained Network');
    load([fpath,fname]);
    % ie. F = F
end

% F now represents the trained network that can be used on any image

```

```

if ~isempty(get(handles.text17,'userdata'))
    qans = questdlg('Use current loaded images?','Settings','Yes','No','No');
end
if isempty(get(handles.text17,'userdata')) | strcmp(qans,'No')
    %loadtheimages(handles);
    [filename,path] = uigetfile('alimages\*.bmp','Load Input Image');
    [P,pmap] = imread([path,filename]);
    Pimage = double(P);
else
    Pimage = get(handles.text17,'userdata'); %make local variable Pimage
end

%load 'lv', 'Tv', Pimage and Timage data.
%lv = get(handles.lvinfo,'userdata');% make local variable lv = "global" lv
%Tv = get(handles.tvinfo,'userdata');% make local variable Tv = "global" Tv
%Pimage = get(handles.text17,'userdata'); %make local variable Pimage = "global" Pimage
Timage = get(handles.finfo,'userdata');% make local variable Timage = "global" Timage

%% process input image using trained FuNN
RuleO = ProcessImageUsingFuNN(Pimage, F);

%% crop target image to same size as output image for error calculation purposes
NT = CropTargetToOutput(Timage, RuleO, [F.WFVSize F.WFHSize]);

%% create new window to display output image
output_handle = figure('name','Output'); %not needed
image(RuleO);
colormap(gray(256));

% display error calculations
RMS = sqrt(mse(NT - RuleO));
[fp fn err percerr] = calcerror2( NT, RuleO)
msgbox(['The RMS value was ',num2str(RMS),' with ',[num2str(fp),' false positives',
'],[num2str(fn),' false negatives '],[num2str(err),' errors and '],[num2str(percerr),' percent
errors'] },...
'Error Calculation','help');uiwait
%offer chance to post process image
qanswer = questdlg('Do you want to use a post processing procedure?',...
'Post Process?','Yes','No','No');
if strcmp(qanswer,'Yes')
    qanswer = questdlg('Do you want to save the pre processed image?',...
'Save Image?','Yes','No','No');
    if strcmp(qanswer,'Yes')
        RuleO = uint8(RuleO);
        [fname,fpath] = uiputfile('alimages\*.bmp','Pre-Processed Output File');
        imwrite(RuleO,[fpath,fname,'.bmp'],'bmp');
        RuleO = double(RuleO);
    end
    [imax,jmax] = size(RuleO);
    for i=1:imax
        for j=1:jmax
            inver(i,j) = 256 - RuleO(i,j);
        end
    end
    RuleO = 255 - RuleO; % prepare image for post processing
    disp('Started post processing');
    finalim = findPathIris(RuleO);
    figure;
    imshow(finalim);
    qanswer = questdlg('Do you want to save the post processed image?',...

```

```

        'Save Image?','Yes','No','No');
    if strcmp(qanswer,'Yes')
        [fname,fpath] = uiputfile('alimages\*.bmp','Processed Output File');
        imwrite(finalim,[fpath,fname,'.bmp'],'bmp');
    end
end

% -----saved trained network-----
% -----
function Untitled_3_Callback(hObject, eventdata, handles)

SNN = get(handles.mydata,'UserData'); % make local variable SNN = "global" SNN

if ~strcmp(get(handles.text1,'userdata'),'NNtrained')
    msgbox('The network has not yet been trained','Error','warn');
    uiwait
else
    F = SNN.F;
    % save 'mat' file containing SNN data
    [fname,fpath] = uiputfile('Trained_NN.mat','Save Trained Network');
    save ([fpath,fname],'F');
end

% ----load input image only-----
function loadnongrayscale_Callback(hObject, eventdata, handles)

SNN = get(handles.mydata,'UserData'); % make local variable SNN = "global" SNN

if isempty(get(handles.mydata,'userdata'))
    msgbox({'Please either select "File->Load Settings" or',...
        '"File->Set New" data before loading images'},...
        'Warning','warn');
    uiwait;
elseif (SNN.IsWindowFilter == 1)
    disp('here');
    msgbox({'This option is meant for Non-window filter networks.',...
        'Please change the window filter setting to "0",...
        'or choose the "Load Input and Target Images" option.'},...
        'Warning','warn');
else
    Pf = figure(1);
    screen = get(0,'ScreenSize'); % left, bottom, width, height
    pos1 = [1,1,screen(3)/2,screen(4)*5/6];
    pos2 = [screen(3)/2,1,screen(3)/2,screen(4)*5/6];
    set(Pf,'Position',pos1); % position figure
    [filename,path] = uigetfile('alimages\*.bmp','Load Input Image');
    [Pimage,pmap] = imread([path,filename]);
    Pimage = double(Pimage);

    image(Pimage); % display the image
    colormap(pmap);
    msgbox('Input image loaded','Notice','help');
    uiwait
    close; % closes input image window

    % set Pimage to userdata to make it 'global'
    set(handles.text17,'userdata',Pimage);

    % show that images have been loaded

```

```

set(handles.specialinfo,'string',{'Loaded Input Image : ',filename});

axes(handles.displayaxest); % make this axes current
set(handles.displayaxest,'Visible','Off');

axes(handles.displayaxesi); % make this axes current
set(handles.displayaxesi,'Visible','On');
image(Pimage);
colormap(pmap);

end %if else

% -----
function reselectcluster_Callback(hObject, eventdata, handles)

SNN = get(handles.mydata,'UserData'); % make local variable SNN = "global" SNN
[Pimage] = get(handles.text17,'userdata'); %make local variable Pimage = "global" Pimage
[Timage] = get(handles.finfo,'userdata'); % make local variable Timage = "global" Timage
map1 = colormap(gray(256));
map2 = colormap(gray(256));
r1 = Pimage;
r2 = Timage;
ma = questdlg('Do you wish to select multiple areas?','Cluster Area','Yes','No','No');
if strcmp(ma,'Yes')
    na = inputdlg('How many cluster area"s do you wish to select ?','Cluster Area Selection',[1])
    answer = str2num(cell2mat(na));
    %uiwait;
else
    answer = 1; % select only 1 area
end
Pf = figure(1);
image(Pimage);
colormap(map1);

% pick sub areas in image to train (changed from F.WFHSize to SNN.WFHSize)
[lv, Tv] = pickCluster2(answer, Pimage, r1, map1, Timage,r2, map2, Pf, [SNN.WFHSize
SNN.WFVSize]);

set(handles.lvinfo,'userdata',lv); % set lv to userdata so it can 'global'
set(handles.tvinfo,'userdata',Tv); % set Tv to userdata so it can 'global'
close Figure No. 1;
close Figure No. 2;
disp('Images loaded and cluster area selected');

% -----
function runNWFnetwork_Callback(hObject, eventdata, handles)
% calls the NWFNetworkPollen GUI

%load 'SNN', 'lv', 'Tv', Pimage and Timage data
SNN = get(handles.mydata,'UserData'); % make local variable SNN = "global" SNN

if (SNN.IsWindowFilter == 0) & (~isempty(SNN))
    lv=0; Tv=0; Pimage=0; Timage=0;
    save ('tempqpzm10','SNN');
    NWFNetworkPollen(handles);
    uiwait
    load('tempqpzm10');
    disp('Control back to Set-user-options');
    %set SNN to the userdata of the object 'data' so that all functions can now reference SNN

```

```

    set(handles.mydata,'Userdata',SNN);
else
    msgbox('This is not a "non-window filter" (NWF) network.','Error','warn');
    uiwait;
end

```

Functions not associated with just one particular GUI follow:

Function UpdateRules

```

function update = updatrules(oldrules,newrule,rulenum,totalnumberrules)
% given the 'list' of old-rules, a new 'updated' rule and its rule-number
% this function returns an updated 'list' of rules
temprule = "";
% get to start position of rule to be updated
if rulenum == 1
    [currentword,rest] = ExtractWord(oldrules);
    %rulestart = wholeline;
else
    [currentword,rest] = ExtractWord(oldrules);%first 'if'

    for x = 1:(rulenum-1)
        temprule = [temprule,' ',currentword]; %put in 'if'
        [currentword,rest] = ExtractWord(rest);%next word after 'if'
        %temprule = [temprule,' ',currentword];
        while ~strcmp(currentword,'If') % continue to next if
            temprule = [temprule,' ',currentword];
            [currentword,rest] = ExtractWord(rest);
        end %while
        %temprule = [temprule,' ',currentword]; %put in next if
    end %for
end %if else

% at this point currentword is 'If' of rule to be replaced
% temprule hold all info before rule to be replaced
% with the remainder in 'rest'
% now insert updated rule

if rulenum == totalnumberrules % i.e if changing the last rule...
    temprule = [temprule,' ',newrule]; % put it on the end and forget the rest
else
    temprule = [temprule,' ',newrule,' '];%ie. temprule=start+newrule
    %get rid of the rest of the oldrule being replaced,rest=oldrule+lastrules
    [currentword,rest] = ExtractWord(rest); % next word after 'if' of old rule
    while (~strcmp(currentword,'If') ) % continue to 'If' of the next rule
        if isempty(rest)
            break
        end
        [currentword,rest] = ExtractWord(rest);
    end %while
    % we are now at the 'If' of the next rule (after the oldrule)
    if ~isempty(rest)
        temprule = [temprule,currentword,rest]; % need a space here???
    end % if
end % if else
disp('UpdateRules Function Completed');

update = temprule;

```

Function ClassifyPollenSGLDM

```
function classifyPollenSGLDM(FNN,type)
% classifies pollen given a trained network and a pollen type
% note!!! uses a sample size of 30

disp('Classify Pollen SGLDM procedure begun...');
if strcmp(type,'A')
    maxptypes =40;
elseif strcmp(type,'B')
    maxptypes = 14;
elseif strcmp(type,'D')
    maxptypes = 2;
elseif strcmp(type,'E')
    maxptypes = 11;
elseif strcmp(type,'F')
    maxptypes = 10;
end
numofpollen = 30; % sample size
for pollentype = 1:maxptypes
    for i = 1:numofpollen
        name = getname(pollentype,i,type);
        Pimage = imread(name);

        %find the interesting info for each image
        [ycentre,xcentre,horizmaxrun,vertmaxrun,BW2,area,pts,avgPixelIntensity] =
findcentremass(Pimage,2);
        [circ,rmean] = findcircularity(ycentre,xcentre,vertmaxrun,horizmaxrun,BW2);
        smimage = findCentralArea(Pimage,rmean,ycentre,xcentre);
        [energy, homogeneity, entropy, inertia, correlation] = getSGLDMdata(smimage);

        %formulate input vectors
        %lv = [entropy;inertia;correlation;area];
        lv = [energy;homogeneity;entropy;inertia;correlation;circ;area]; % 7 NOW!!!!!!!!!!!!
        %lv = [energy;inertia;correlation]; % 3
        %lv = [energy;homogeneity;entropy;inertia]; %4

        % simulate the FuNN and record data
        cir(pollentype,i) = circ;
        are(pollentype,i) = area;
        ene(pollentype,i) = energy;
        hom(pollentype,i) = homogeneity;
        ent(pollentype,i) = entropy;
        ine(pollentype,i) = inertia;
        cor(pollentype,i) = correlation;
        results(pollentype,i) = SimFuNN(FNN, lv);
        pollennames{pollentype} = name;
    end
    avgene(pollentype,3) = mean(ene(pollentype,:));
    avgene(pollentype,2) = max(ene(pollentype,:));
    avgene(pollentype,1) = min(ene(pollentype,:));
    avghom(pollentype,3) = mean(hom(pollentype,:));
    avghom(pollentype,2) = max(hom(pollentype,:));
    avghom(pollentype,1) = min(hom(pollentype,:));
    avgent(pollentype,3) = mean(ent(pollentype,:));
    avgent(pollentype,2) = max(ent(pollentype,:));
    avgent(pollentype,1) = min(ent(pollentype,:));
    avgine(pollentype,3) = mean(ine(pollentype,:));
    avgine(pollentype,2) = max(ine(pollentype,:));
```

```

    avgine(pollentype,1) = min(ine(pollentype,:));
    avgcor(pollentype,3) = mean(cor(pollentype,:));
    avgcor(pollentype,2) = max(cor(pollentype,:));
    avgcor(pollentype,1) = min(cor(pollentype,:));
    avgcir(pollentype,3) = mean(cir(pollentype,:));
    avgcir(pollentype,2) = max(cir(pollentype,:));
    avgcir(pollentype,1) = min(cir(pollentype,:));
    avgare(pollentype,3) = mean(are(pollentype,:));
    avgare(pollentype,2) = max(are(pollentype,:));
    avgare(pollentype,1) = min(are(pollentype,:));
end
disp('Energy');
disp([avgene])
disp('homogeneity');
disp([avghom])
disp('entropy');
disp([avgent])
disp('inertia');
disp([avgine])
disp('correlation');
disp([avgcor])
disp('circ');
disp([avgcir])
disp('area');
disp([avgare])
disp('Results');
results
disp('*****Special Results*****');
disp('energy');
disp(ene(:,1:11));
disp('hom');
disp(hom(:,1:11));
disp('Entropy');
disp(ent(:,1:11));
disp('Inertia');
disp(ine(:,1:11));
disp('correlation');
disp(cor(:,1:11));
disp('circ')
disp(cir(:,1:11));
disp('area');
disp(are(:,1:11));
disp('*****');
displayresultSGLDM1(results,type);

```

Function displayresultsSGLDM1

```

function displayresultSGLDM1(results,type)
% diplays results to the screen given results variable
% and the pollen type being classified

[numtypesp,numsamples] = size(results);

if strcmp(type,'A')
    for pollentype = 1:numtypesp
        bignonc=0; bigc =0; smallc=0;
        for i =1:numofpollen %numofpollen
            if results(pollentype,i) > 0.75
                bignonc = bignonc + 1;
            elseif results(pollentype,i) > 0.25

```



```

        bigc = bigc + 1;
    else
        smallc = smallc + 1;
    end
end
disp(['Type B ',num2str(bignonc+bigc),' Type C ',num2str(smallc)]);
end
elseif strcmp(type,'B')
    for i = 1:numtypesp
        zm=0; noteither = 0;ad =0;
        for j = 1:numsamples
            if results(i,j) > 0.68
                zm =zm + 1;
            elseif results(i,j) < 0.3
                ad = ad + 1;
            else
                noteither = noteither + 1;
            end
        end
    end
    %disp(['Zm ', num2str(zm),' Not either ',num2str(ad),' Ac52 ',num2str(noteither)]);
    disp(['Zm ', num2str(zm),' Type D ',num2str(ad),' Type E ',num2str(noteither)]);
end

elseif strcmp(type,'D')
    for i = 1:numtypesp
        pr = 0; ad =0;
        for j = 1:numsamples
            if results(i,j) > 0.11
                pr =pr + 1;
            else
                ad = ad + 1;
            end
        end
    end
    disp(['Ad ', num2str(ad),' Pr ',num2str(pr)]);
end

elseif strcmp(type,'E')
    for i = 1:numtypesp
        typeF = 0; bw =0; ta=0;
        for j = 1:numsamples
            if (results(i,j) > 0.7)
                ta = ta + 1;
            elseif (results(i,j) < 0)
                bw=bw+1;
            else %results(i,j) >= 0.75
                typeF =typeF + 1;
            end
        end
    end
    disp(['ta ',num2str(ta),' Type g ',num2str(bw),' Type F ', num2str(typeF)]);
end

elseif strcmp(type,'F')
    for i = 1:numtypesp
        typeG = 0; typeH = 0;
        for j = 1:numsamples
            if (results(i,j) > 0.5)
                typeG = typeG + 1;
            else
                typeH =typeH + 1;
            end
        end
    end
end

```

```

        end
        disp(['Type G ',num2str(typeG),', Type H ',num2str(typeH)]);
    end
end

```

Function findCentralArea

```

function [smimage] = findCentralArea(Pimage,rmean,ycentre,xcentre)
% finds the square area based at the centre of mass of each pollen
% to be used for applying the SGLDM matrix.
% inputs: image, mean radius of image, centre coordinates (y,x)
% output: rectangular image ready for processing by SGLDM matrix
hw = round(rmean*0.6);
sm = Pimage((ycentre-hw):(ycentre+hw) , (xcentre-hw):(xcentre+hw));
smimage = sm;

```

Function findcentremass

```

function [ycentre,xcentre,horizmax,vertmax,BW2,area1,area2,avgPixelIntensity] =
findcentremass(inimage,type)
% given a greyscale image, returns centre-of-mass, area, binary image,
% dimensions of the image and list of all points (x,y) in the image.
[vertmax,horizmax] = size(inimage);

% turn input into a binary blob (blob is black)
input = double(inimage);
BW = im2bw(input/256,250/256);
SE = strel('arbitrary',eye(5));
BW2 = imerode(BW,SE);
% do it twice to ensure white holes are filled in
BW2 = imerode(BW2,SE);

xsum = 0; ysum=0; area = 0; pts = 0;totalPixelIntensity =0;
% find the centre of mass
for x = 1:horizmax
    for y = 1:vertmax
        if (BW2(y,x) == 0) %ie. black
            xsum = xsum + x;% a pt on the blob
            ysum = ysum + y;% a pt on the blob
            area = area + 1;% increment area caculation
            %pts(1,area) = x; %pts(2,area) = y;
            % calculate avg pixel intensity of original image
            %totalPixelIntensity = totalPixelIntensity + double(inimage(y,x));
        end % if
    end %for y
end % for x
% calculate first moments
xcentre = round(xsum / area);
ycentre = round(ysum / area);
%avgPixelIntensity = round(totalPixelIntensity / area);

area2=0; area1 = 0;
% make area value between 0 small - 1 big
if type == 1
    area1 = area / 15000;
elseif type == 2
    area1 = area / 30000;
else
    area1 = area / 15000;

```

```

    area2 = area / 30000;
end
if area1 > 1
    area1 = 1;
end
if area2 > 1
    area2 = 1;
end
avgPixelIntensity=0;

```

Function findcircularity

```

function [rstd,rmean] = findcircularity(ycentre,xcentre,ymax,xmax,inimage)
% finds a pollens circularity
% start at the centre and find the radius lengths
% at 0d (straight up)and 45,90,135,180,225,270,315.
% then find the std dev of the radii lengths
endsignal = 0;
rmean = 0;
r1 = 0; %radius r1
r2 = 0; r3 = 0; r4=0; r5=0; r6=0; r7=0;r8=0;
r9=0; r10=0; r11=0; r12=0; r13=0; r14=0; r15=0; r16=0;
xpoff=0; ypoff=0;
radius1 = 1;
xpos = xcentre; ypos = ycentre;
xneg = xcentre; yneg = ycentre;
while (endsignal == 0)
    if (inimage(ypos,xcentre)==0) %down
        r1 = r1 + 1; % extend length of r1
    end
    if (inimage(ypos,xpos)==0) %down right
        r2 = r2 + 1;
    end
    if(inimage(ycentre,xpos)==0) %right
        r3 = r3 + 1;
    end
    if(inimage(ycentre,xneg)==0) %left
        r4 = r4 + 1;
    end
    if(inimage(yneg,xneg)==0) %left up
        r5 = r5 + 1;
    end
    if(inimage(yneg,xcentre)==0) %up
        r6 = r6 + 1;
    end
    if(inimage(ypos,xneg)==0) %down left
        r7 = r7 + 1;
    end
    if(inimage(yneg,xpos)==0) %up right
        r8 = r8 + 1;
    end
    % more
    if(inimage(yneg,xcentre+round(xpoff))==0) % 1 oclock
        r9 = r9 + 1;
    end
    if(inimage(ycentre-round(ypoff),xpos)==0) % 2 oclock
        r10 = r10 + 1;
    end
    if(inimage(ycentre+round(ypoff),xpos)==0) % 4 oclock
        r11 = r11 + 1;
    end
end

```

```

end
if (inimage(ypos,xcentre+round(xpoff))==0) % 5 oclock
    r12 = r12 + 1; % extend length of r1
end
if (inimage(ypos,xcentre-round(xpoff))==0) % 7 oclock
    r13 = r13 + 1;
end
if(inimage(ycentre+round(ypoff),xneg)==0) % 8 oclock
    r14 = r14 + 1;
end
if(inimage(ycentre-round(ypoff),xneg)==0) % 10 oclock
    r15 = r15 + 1;
end
if(inimage(yneg,xcentre-round(xpoff))==0) % 11 oclock
    r16 = r16 + 1;
end

ypoff=ypoff+0.5;
xpoff=xpoff+0.5;
ypos=ypos+1;
xpos=xpos+1;
xneg=xneg-1;
yneg=yneg-1;
%check we haven't reached the edges of the image
if (ypos>ymax)|(xpos>xmax)|(xneg==0)|(yneg==0)
    endsignal=1;
end
end % while
%r2 = round(1.4142*r2);

%process radius data
rvalues = [r1,r2,r3,r4,r5,r6,r7,r8,r9,r10,r11,r12,r13,r14,r15,r16];
%rvalues = [r1,r2,r3,r4,r5,r6,r7,r8];
rmean = (r1+r2+r3+r4+r5+r6+r7+r8+r9+r10+r11+r12+r13+r14+r15+r16) / 16;
%rmean = (r1+r2+r3+r4+r5+r6+r7+r8)/8;
% allow for image size
rvalues = rvalues / ((ymax+xmax)/1000);
rstd = std(rvalues);

% just to get the result between 0 - 100
% 0 = circular, 1 = not-circular
rstd = rstd / 50;
if rstd > 1
    rstd = 1;
end

```

Function getInputImage2

```

function [filename,path, r, map, name]=getInputImage2(hHandle)
% function to load input image into Input Figure Window
% use InputFigure Window
[filename,path] = uigetfile('alimages\*.bmp','Load Input Image');
[P,pmap] = imread([path,filename]);
P = double(P);
figure(hHandle);
image(P); % display the image
colormap(pmap);
map = pmap;
r=P;
name = filename;

```

Function getinputvectorssgldm

```
function finallv = getinputvectorssgldm(type)
% returns input vector for netowrk, given pollen type
if strcmp(type,'A')
    maxptypes = 40;
elseif strcmp(type,'B')
    maxptypes = 14;
elseif strcmp(type,'D')
    maxptypes = 2;
elseif strcmp(type,'E')
    maxptypes = 11;
elseif strcmp(type,'F')
    maxptypes = 10;
end
numofpollen = 5;
counter = 0;
qanswer = questdlg('Do you wish to load a special file for the input vectors?',...
    'Input Vectors','Yes','No','No');
if strcmp(qanswer,'No')

    for pollentype = 1:maxptypes
        for i = 1:numofpollen
            name = getname(pollentype,i,type);
            Pimage = imread(name);
            [ycentre,xcentre,horizmaxrun,vertmaxrun,BW2,area,pts,avgPixelIntensity] =
findcentremass(Pimage,2);
            [circ,rmean] = findcircularity(ycentre,xcentre,vertmaxrun,horizmaxrun,BW2);
            smimage = findCentralArea(Pimage,rmean,ycentre,xcentre);
            %find the interesting info for each image
            [energy, homogeneity, entropy, inertia,correlation] = getSGLDMdata(smimage);

            %formulate input vectors
            counter = counter + 1;
            lvne(counter) = energy;
            lvhom(counter) = homogeneity;
            lvne(counter) = entropy;
            lvine(counter) = inertia;
            lvcor(counter) = correlation;
            lvcirc(counter) = circ;
            lvarea(counter) = area;
            %store input vectors for matlab display
            e1(pollentype,i) = energy;
            e2(pollentype,i) = entropy;
            c(pollentype,i) = correlation;
            in(pollentype,i) = inertia;
            h(pollentype,i) = homogeneity;
        end
    end
    %finallv = [lvne;lvine;lvcor;lvarea] % 4
    finallv = [lvne;lvhom;lvne;lvine;lvcor;lvcirc;lvarea] % 7
    disp('Input vectors');
    disp('energy');
    disp(e1)
    disp('entropy');
    disp(e2)
    disp('cor');
    disp(c);
    disp('inertia');
```

```

disp(in);
disp('hom');
disp(h);

else
    [fname,fpath] = uigetfile('*.mat');
    x = load ([fpath,fname]);
    disp('final lv');
    finallv = x.finallv
end

```

Function getname

```

function fname = getname(pollentype,i,type)
% get filenames from set list
%note. This function needs to be adjusted if a new set of input images
% are to be classified
setlist_TypeF ={'aa47\47','ap122\122','bd76\76','bw144\144','cd147\147';...
    'ctree34\34','fa120\120','jr45\45','lp141\141','pm81\81'};
setlist_TypeE
= {'aa47\47','ap122\122','bd76\76','bw144\144','cd147\147','ctree34\34','fa120\120','jr45\45';...
    'lp141\141','pm81\81','ta59\59'};
setlist_TypeD ={'ad157\157','pr142\142'};
setlist_TypeB
= {'aa47\47','ad157\157','ap122\122','bd76\76','bw144\144','cd147\147','ctree34\34','fa120\120'
    'jr45\45';...
    'lp141\141','pm81\81','pr142\142','ta59\59','zm149\149'};
setlist_All = {'aa47\47','ac52\52','ad157\157','al160\160','an40\40','ap122\122','apl243\243';...
    'bd76\76','bp36\36','br46\46','bw144\144','cd147\147','cm64\64','cor159\159','cr33\33';...
    'ctree34\34','dg119\119','es61\61','fa120\120','hl54\54','jr45\45','ll148\148';...
    'lop138\138','lp141\141','ls221\221','nf42\42','pa84\84','pd37\37','pl39\39','pm81\81';...
    'po207\207','pr142\142','ps26\26','qr11\11','ra70\70','ro72\72','sa129\129','sg35\35';...
    'ta59\59','zm149\149'};

if i <=9
    if strcmp(type,'A')
        fname = ['almages\'',cell2mat(setlist_All(pollentype)), '0',num2str(i),'.tif'];
    elseif strcmp(type,'B')
        fname = ['almages\'',cell2mat(setlist_TypeB(pollentype)), '0',num2str(i),'.tif'];
    elseif strcmp(type,'D')
        fname = ['almages\'',cell2mat(setlist_TypeD(pollentype)), '0',num2str(i),'.tif'];
    elseif strcmp(type,'E')
        fname = ['almages\'',cell2mat(setlist_TypeE(pollentype)), '0',num2str(i),'.tif'];
    elseif strcmp(type,'F')
        fname = ['almages\'',cell2mat(setlist_TypeF(pollentype)), '0',num2str(i),'.tif'];
    end
else
    if strcmp(type,'A')
        fname = ['almages\'',cell2mat(setlist_All(pollentype)),num2str(i),'.tif'];
    elseif strcmp(type,'B')
        fname = ['almages\'',cell2mat(setlist_TypeB(pollentype)),num2str(i),'.tif'];
    elseif strcmp(type,'D')
        fname = ['almages\'',cell2mat(setlist_TypeD(pollentype)),num2str(i),'.tif'];
    elseif strcmp(type,'E')
        fname = ['almages\'',cell2mat(setlist_TypeE(pollentype)),num2str(i),'.tif'];
    elseif strcmp(type,'F')
        fname = ['almages\'',cell2mat(setlist_TypeF(pollentype)),num2str(i),'.tif'];
    end
end
end

```

```
disp(fname);
```

Function getrule

```
function rule = getrule(wholeline,rulenum,totalnumrules)
% returns rule number 'rulenum' from the rule variable 'wholeline'
temprule = "";
% get to start position of desired rule
if rulenum == 1
    [currentword,rest] = ExtractWord(wholeline); %gets first 'if'
    %rulestart = wholeline;
else
    [currentword,rest] = ExtractWord(wholeline);% gets first 'if'

    for x = 1:(rulenum-1)
        [currentword,rest] = ExtractWord(rest);%next word after 'if'
        while ~strcmp(currentword,'If') % continue to next if
            [currentword,rest] = ExtractWord(rest);
        end %while
    end %for
end %if else
% at this point currentword is 'if' (of desired rule)

[currentword,rest] = ExtractWord(rest); %next word after 'if'
temprule = ['If ',currentword]; %build up rule - first part
while (~strcmp(currentword,'Then') ) % continue to 'then'
    if isempty(rest)
        %temprule = [temprule,currentword];
        break
    end
    [currentword,rest] = ExtractWord(rest);
    temprule = [temprule,' ',currentword]; %build up rule
    %disp(['temprule ',temprule]);
end %while

[currentword,rest] = ExtractWord(rest); % get last 'result' part
temprule = [temprule,' ',currentword]; % add result part
rule = temprule;
```

Function getTargetImage2

```
function [filename,path, r,map, name] =getTargetImage2(hHandle)
% function to load target image to TargetFigure Window

[filename,path]=uigetfile('alimages\*.bmp','Load Target Image');
[T,tmap] = imread([path,filename]);
figure(hHandle);
image(T);
colormap(tmap);
T=double(T);
map = tmap;
r=T;
name = filename;
```

Function gettargetvectorssgldm

```
function finalTv = gettargetvectorssgldm(lv,type)
```

```

% given input vectors produces desired output vectors

[maxrows,maxnumber]=size(lv);

if strcmp(type,'A')
    for i = 1:maxnumber % for all values in input

        if lv(1,i) < 0.52 % circle-like
            if lv(2,i) > 0.5 % big
                Tv(i) = 0.5; %big circle
            else
                Tv(i) = 0; % small circle
            end
        else % if >= 0.55 non circle
            Tv(i) = 1; % big non-circle
        end
    end % for

elseif strcmp(type,'B') % stage : split up type B
    Tv(1,maxnumber) = 0;
    for x=1:((maxnumber/14)) % because 14 pollen types in type B
        Tv(1,2*(maxnumber/14)-(maxnumber/14)+x) = 0.5; %Type D
        Tv(1,12*(maxnumber/14)-(maxnumber/14)+x) = 0.5; %type D
        Tv(1,14*(maxnumber/14)-(maxnumber/14)+x) = 1; %zm
    end % end for

elseif strcmp(type,'D') %stage: split up type D (ie 'ad' and 'pr')
    Tv(1,maxnumber) = 0;
    for x=1:((maxnumber/2)) % because 2 pollen types in type D
        Tv(1,2*(maxnumber/2)-(maxnumber/2)+x) = 1;
    end

elseif strcmp(type,'E') %stage: split up type E (11 types)
    Tv(1,maxnumber) = 0;
    for x=1:((maxnumber/11))
        Tv(1,11*(maxnumber/11)-(maxnumber/11)+x) = 1; % ta
    end

elseif strcmp(type,'F') %stage: split up type E (11 types)
    Tv(1,maxnumber) = 0;
    for x=1:((maxnumber/10))
        Tv(1,1*(maxnumber/10)-(maxnumber/10)+x) = 0;
        Tv(1,2*(maxnumber/10)-(maxnumber/10)+x) = 1;
        Tv(1,3*(maxnumber/10)-(maxnumber/10)+x) = 0;
        Tv(1,4*(maxnumber/10)-(maxnumber/10)+x) = 1;
        Tv(1,5*(maxnumber/10)-(maxnumber/10)+x) = 1;
    end
else
    disp('This type is not implemented yet');
end% end if else
finalTv = Tv;

```

Function getTrainingSegment2

```

function [x1,y1,x2,y2]=getTrainingSegment2(H)
% function to get the area to train using the mouse
% makes sure pt1 is above and left of pt2 and returns
% also plots the selected area on the figure
% functions used: -
figure(H);

```



```

hold on;

k = waitforbuttonpress;
point1 = get(gca,'CurrentPoint'); % button down detected
finalRect = rbbox; % return figure units
point2 = get(gca,'CurrentPoint'); % button up detected
point1 = point1(1,1:2); % extract x and y
point2 = point2(1,1:2);
p1 = min(point1,point2); % calculate locations
offset = abs(point1-point2); % and dimensions
x = [p1(1) p1(1)+offset(1) p1(1)+offset(1) p1(1) p1(1)];
y = [p1(2) p1(2)+offset(2) p1(2)+offset(2) p1(2) p1(2)];
hold on
axis manual
plot(x,y) % redraw in dataspace units
% now make sure x1<x2 and y1<y2
if point1(1) < point2(1)
    x1 = round(point1(1));
    x2 = round(point2(1));
else
    x1 = round(point2(1));
    x2 = round(point1(1));
end
if point1(2) < point2(2)
    y1 = round(point1(2));
    y2 = round(point2(2));
else
    y1 = round(point2(2));
    y2 = round(point1(2));
end
hold off;

```

Function parse3

```

function r = parse3(NN)

%1. the values for DataFilename -> Goal have ALL BEEN SET ALREADY
%2. Fuzzification has effectively been done also
%3. DeFuzzification has effectively been done also
%4. 'InitializeMatrix' sets the size of RuleMatrix and ActionMatrix
NN.RuleMatrix = zeros(NN.TotalRuleNodes, NN.NumberOfInputNeurons * NN.FuzzyMF);
NN.ActionMatrix = zeros(NN.DefuzzyMF, NN.TotalRuleNodes);
if NN.TypeOfNetwork == 'MLP'
    NN.RuleBiasMatrix = zeros(NN.TotalRuleNodes, 1);
end
if NN.IsWindowFilter
    if NN.NumberOfInputNeurons ~= (NN.WFHSize*NN.WFVSize)
        warndlg('Mismatch in Input nodes and Window Filter size. FuNN will not work properly!');
        %break;
    end
end
% 'Fuzzification' (all done already)
% future expansion
%NN.FuzzyMFDEF = [];
%for FMc=1:NN.NumberOfInputNeurons
%    NN.FuzzyMFDEF = [NN.FuzzyMFDEF FMc];
%end
% case 'Defuzzification'

```

```

%5. Rule lines. I have changed this slightly beacuse the rules are now all
% in one long string and do not have 'R 1' or 'R 2' before each one.
CurrentRule = 1;
[CurrentWord, Line] = ExtractWord(NN.Rule); %ie. read the very first 'If'
while ~isempty(Line)
%at this stage the CurrentWord is 'If', Line contains the rest of the rule(s)
    ProcessConsequent = 0;

    while ~isempty(Line)
        [CurrentWord, Line] = ExtractWord(Line);

        if upper(CurrentWord(1,1:3)) == 'THE'
            ProcessConsequent = 1;
        end

        switch ProcessConsequent
        case 0
            [left, right, opt] = SplitWord(CurrentWord);
            left = ExtractNumber(left);
            right = ExtractNumber(right);
            if (isempty(left) | isempty(right))
                fprintf('Error!!!! Input or membership funtion not a numeric\n');
            end
            switch opt
            case '='
                NN.RuleMatrix(CurrentRule, ((left-1).*NN.FuzzyMF+1):(left.*NN.FuzzyMF)) =
ones(1,NN.FuzzyMF).*-1;
                NN.RuleMatrix(CurrentRule, ((left-1).*NN.FuzzyMF + right)) = 1;
            case '<'
                NN.RuleMatrix(CurrentRule, ((left-1).*NN.FuzzyMF+1):(left.*NN.FuzzyMF)) =
ones(1,NN.FuzzyMF).*-10;
                NN.RuleMatrix(CurrentRule, ((left-1).*NN.FuzzyMF+1):(left-1).*NN.FuzzyMF +
right)) = 10;
            case '>'
                NN.RuleMatrix(CurrentRule, ((left-1).*NN.FuzzyMF+1):(left.*NN.FuzzyMF)) =
ones(1,NN.FuzzyMF).*-20;
                NN.RuleMatrix(CurrentRule, ((left-1).*NN.FuzzyMF + right):left.*NN.FuzzyMF) = 20;
            end
        case 1 % process the 'then O1=M2' part
            [CurrentWord, Line] = ExtractWord(Line);
            [left, right] = SplitWord(CurrentWord);
            left = ExtractNumber(left);
            right = ExtractNumber(right);

            NN.ActionMatrix(((left-1).*NN.DefuzzyMF+1):(left.*NN.DefuzzyMF), CurrentRule) =
ones(NN.DefuzzyMF,1).*-1;
            NN.ActionMatrix(((left-1).*NN.FuzzyMF + right), CurrentRule) = 1;
            ProcessConsequent = 0; %reset it --not needed ??
            break; %out of the 'inner' while loop
        end % switch
    end % while
    % breaks to 'here' at the end of a rule
    if CurrentRule < NN.NumberRules % if we haven't already processed the last rule
        %disp(Line);
        [CurrentWord, Line] = ExtractWord(Line); %get 'If' of the next rule
        CurrentRule = CurrentRule + 1; % we will now process next rule
    else
        %disp ('end here');
        break;
    end %if

```

```

end % outer while loop

%6. Process Rules
NN = ProcessRuleMatrix2(NN);
%7. Process Actions
NN = ProcessActionMatrix(NN);

r=NN; % return the 'parsed' neural network

```

function pickcluster2

```

function [finalV, finalT] = pickcluster2(numareas, Pimage, P1, map1, Timage, P2, map2,
figureNumber, WFsize)
% function to pick a cluster from the input or target image
% returns area picked in input & target vector format
% functions used: getTrainingSegment2, getTrainingVectors

fV=[];
fT=[];

for i = 1:numareas % repeat until correct number of clusters have been selected

qanswer = 0;
while (qanswer == 0) %ie. loop until suitable area has been selected
    Vc=[];
    Tc=[];
    figure(2); % push figure 2 towards the front of the display again
    [x1,y1,x2,y2]=getTrainingSegment2(figureNumber);
    [Vc,Tc]=getTrainingVectors(x1,y1,x2,y2,Pimage,Timage,WFsize);

    if ~isempty(Vc)
        qanswer = questdlg('Use this area?','Area has been selected','Yes','No','No');
        if strcmp(qanswer,'Yes')
            fV = [fV Vc];
            fT = [fT Tc];
        else % ie. if area is not wanted by user
            % delete current figures and replace them (so as to remove the blue box)
            qanswer = 0;
            close Figure No. 1;
            close Figure No. 2;
            % determine on-screen positions
            screen = get(0,'ScreenSize'); % left, bottom, width, height
            pos1 = [1,1,screen(3)/2,screen(4)*5/6];
            pos2 = [screen(3)/2,1,screen(3)/2,screen(4)*5/6];

            % display input image
            Pf = figure(1);
            set(Pf,'Position',pos1); % position figure
            P1 = double(P1);
            image(P1);
            colormap(map1);

            % display target image
            Tf = figure(2);
            set(Tf,'Position',pos2); % position figure
            P2 = double(P2);
            image(P2);
            colormap(map2);
        end
    end
end

```

```

end
else % ie. if area is too small
    % delete current figures and replace them (so as to remove the blue box)
    close Figure No. 1;
    close Figure No. 2;
    % determine on-screen positions
    screen = get(0,'ScreenSize'); % left, bottom, width, height
    pos1 = [1,1,screen(3)/2,screen(4)*5/6];
    pos2 = [screen(3)/2,1,screen(3)/2,screen(4)*5/6];

    % display input image
    Pf = figure(1);
    set(Pf,'Position',pos1); % position figure
    P1 = double(P1);
    image(P1);
    colormap(map1);

    % display target image
    Tf = figure(2);
    set(Tf,'Position',pos2); % position figure
    P2 = double(P2);
    image(P2);
    colormap(map2);
end % if else
end % while
end % for

% give option to print out selected areas
panswer = questdlg('Print area selections?',...
    'Print Option','Yes','No','No');
if strcmp(panswer,'Yes')
    fanswer = questdlg('Print input or target image?',...
        'Print Option','Input','Target','Input');
    if strcmp(fanswer,'Input')
        figure(1);
        print;
    else
        figure(2);
        print;
    end % if else
end % end if

finalV = fV;
finalT = fT;

```

References

- Bishop, A., Bailey, D. G. and Hodgson, R. M. (2000). Verification by Iris Recognition. Projects, Vol. 9, 2000. College of Science, Massey University, New Zealand.
- Daugman, J. (1999). Recognising Persons By Their Iris Pattern. Biometrics, Personal Identification in Networked Society. Edited by Jain, A.K., Bolle, R., Pankanti, S. Kluwer Academic Publishers.
- Dix, A. J., Finlay, J. E., Abowd, G. D. and Beale, R. (1998). Human Computer Interaction. Prentice Hall Europe, London.
- Del Bimbo, A. (1999). Visual information retrieval. Morgan Kaaufmann Publishers, San Francisco
- Gunetileke, S. (2000). The Effects of Using Problem Knowledge in a Neural Network for Image Processing Tasks. Doctoral Thesis, Massey University, New Zealand.
- Haykin, S. (1994). Neural Networks: A Comprehensive Foundation. Upper Saddle River, N.J. : Prentice-Hall, c1994
- Hemmi, J. A. (2002). Differentiating Online Help from Printed Documentation. Intercom, July/August 2002.
- Kasabov, N. K. (1996). Foundations of Fuzzy neural networks, Fuzzy Systems and Knowledge Engineering. The MIT Press, London.
- Kim, H.S. and Kim, J.H. (2000). Circle Detection Method Using Intersecting Chords. (Accessed 15/02/03). <http://vivaldi.kaist.ac.kr/iclab>
- Lew, M.S. (2001). Principles of visual information retrieval. London : Springer.

Li, J. (2002). Optimal and Robust Rule Set Generation. Doctoral Thesis, Griffith University, School of Computing and Information Technology, Australia.

Lu, M., Wang, F. and Wang, Y. (2002). Using Image Analysis Methods To Evaluate The Performance of Coal Flotation. (Accessed 12/02/03). <http://www.min-eng.com/protected/flot03ex/lu.doc>

Marchand, P. (1999). Graphics and GUI's with MATLAB (second edition). CRC Press LLC.

McLaughlin, J.L. (1998). A Tutorial on Texture, Final Year Project. Institute of Information Sciences and Technology, Massey University, New Zealand.

Nguyen, H.T. and Walker, E.A. (1997). A First Course in Fuzzy Logic. CRC Press Inc.

Rauch, T. (1996). The Role of Display Tools in Implementing Effective Online Help Systems. (Accessed 10/01/03). <http://www.acm.org/chapters/trichi/newslettters/jul96/online.help.html>

Siroki, S. (1998). Neural Network Based Image Processing: Report on Massey Post Doctoral Fellowship. Massey University, New Zealand.

The Mathworks Inc. (2003). The Mathworks technical support site. (Accessed 15/02/03). www.mathworks.com/support/

The Mathworks Inc. (1993). Building a Graphical User Interface. MathWorks Inc: Massachusetts.

The Mathworks Inc. (1993). MATLAB Users Guide. MathWorks Inc: Massachusetts.

Thimm, G. and Fiesler, E. (1997). Higher Order and Multilayer Perceptron Initialisation. IEEE Transactions on Neural Networks, Vol. 8, No. 2, 1997.