

# **A COMPARATIVE STUDY OF NEURAL NETWORK ALGORITHMS**

By

Nicholas M. Sandirasegaram

A Thesis

Submitted to the Faculty of Graduate Studies and Research  
Through Electrical & Computer Engineering  
In Partial Fulfillment of the Requirements for  
The Degree of Master of Applied Science at the  
University of Windsor

Windsor, Ontario, Canada

January 2001



National Library  
of Canada

Acquisitions and  
Bibliographic Services

395 Wellington Street  
Ottawa ON K1A 0N4  
Canada

Bibliothèque nationale  
du Canada

Acquisitions et  
services bibliographiques

395, rue Wellington  
Ottawa ON K1A 0N4  
Canada

*Your file Votre référence*

*Our file Notre référence*

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-62279-7

Canada

928734

©2001 Nicholas M. Sandirasegaram

## Abstract

Various Neural network models are investigated for Optical Character Recognition application and a Multi-layer Feed forward neural network is trained using a Fast training algorithm. Then the fast training algorithm is compared with the delta rule training algorithm.

The various neural network models studied in this thesis are Hopfield, Hamming, Carpenter/Grssberg, Kohonen, Single layer and Multi-layer neural networks. These models are trained using Arabic numbers and investigated for training speed of the network, number of patterns that can be trained, size of the network, speed of the trained network for test data and noise sensitivity.

The Multi-layer feed forward neural network is trained using the Fast training algorithm and delta rule training algorithm. Then both algorithms are compared for speed and generalization capability of Optical Character Recognition application. The trained and tested data are the 26 English capital letters, Times New Roman font and a font size of 16.

All of the programs are implemented in visual C++ environment. The results are recorded and analyzed in this thesis. It was found that the Multi layer feed forward neural network is the best of all the other neural networks in the numeric recognition but it has a weakness of slow training process. Training speed of the Fast Training Algorithm is faster than the standard Delta Rule Algorithm and the generalization capability is also better compared to the Delta Rule Algorithm

## ACKNOWLEDGEMENTS

I would like to give my sincere thanks to my supervisor, Dr. M. A. Sid-Ahmed. His encouragement and innovative ideas enabled me to successfully complete this thesis.

I wish to thank Dr. J. J. Soltis for his comments and direction in different neural network models. I like to express my sincere appreciation to Dr. G. W. Rankin for his recommendations and sincere support.

I also thank my friends for their concerns and inspirations and my family members for the constant support and encouragement towards completion of my thesis.

# **Table of Contents**

<b>Abstract</b>	iv
<b>Acknowledgements</b>	v
<b>Table of Contents</b>	vi
<b>Chapter I</b>	
1.1    Introduction	1
1.2    Overview of the OCR technique	2
1.3    Pattern Classifier	5
1.3.1    Traditional Classifier	5
1.3.2    Neural Network Classifier	6
1.4    Thesis Organization	6
<b>Chapter II</b>	
2.1    Neural Network	8
2.2    Biological Neuron	9
2.3    Artificial Neuron	10
2.3.1    Activation Rule	11
2.3.2    Activation Function	12
2.3.3    Training Rule	13
2.3.3.1    Supervised Training Rule	13
2.3.3.2    Unsupervised Training Rule	14
2.3.4    Connectivity Rule	14
2.4    Training and Testing Set	16
2.4.1    Training and Testing Set of Arabic Numbers	16
2.4.2    Training and Testing Set of English Capital Letters	18
<b>Chapter III</b>	
3.1    Various Neural Network Algorithms and Their Performances	23

3.2	Neural Networks Connections, Layers and Learning Methods	25
3.2.1	The Hamming Neural Network	25
3.2.2	The Carpenter \ Grossberg Neural Network	28
3.2.3	Kohonen's Self Organizing Feature MAPS Neural Network	31
3.2.4	The Hopfield Neural Network	33
3.2.5	The Single Layer Perceptron Neural Network	35
3.2.6	The Multi Layer Perceptron Neural Network	37
3.3	Investigation using each NN Model	45
3.3.1	Training Speed of the Network	46
3.3.2	Number of Patterns that can be Trained	47
3.3.3	Size of the Neural Network	47
3.3.4	Speed of the Trained Network for Test Data	48
3.3.5	Noise Sensitivity Test	49
3.3.5	Image Translation Test	49
3.4	Experimentation Results	50
3.4.1	The Hamming Neural Network	50
3.4.2	The Carpenter \ Grossberg Neural Network	51
3.4.3	The Kohonen's Neural Network	52
3.4.4	The Hopfield Neural Network	53
3.4.5	The Single Layer Perceptron Neural Network	55
3.4.6	The Multi Layer Perceptron Neural Network	56
3.5	Discussions	58

#### **Chapter IV**

4.1	Fast Multi Layer Perceptron (MLP) Training Algorithm	60
4.2	Fast Training Algorithm	61
4.3	Comparison of MLP Training Algorithms	67

#### **Chapter V**

5.1	Conclusion	72
5.2	Future Direction	73

References	74
------------	----

## Appendix

A	Source Code for the Six NN Models	78
B	Source Code for the Fast Training Algorithm and the Delta Rule Training Algorithms	162
	Vita Auctoris	231

# **Chapter I**

## **1.1 Introduction**

The process of converting the images of a text into a form so that the computer can manipulate the text is called Optical Character Recognition (OCR). OCR can offer important improvements in computer productivity because while computers can process data very quickly, data entry process is still very slow and tedious and has been considered as the real bottleneck in data processing.

OCR technology started in the early 20<sup>th</sup> century. In 1929, a German, Tauscheck [MOR92] obtained the first patent for optical character recognition (OCR). In 1951, the modern OCR technology started with the great American pioneer David Shepard. He founded the Intelligent Machine Research Corporation in Washington, D.C in 1950 to develop and build an OCR engine [SRI92]. Following Shepard, J. Rainbow developed an engine in 1954 that was able to read uppercase type at the speed of one character per minute [SRI92]. This engine is built for a particular font and size of the character. In the late 1960's and early 1970's, large number of commercial OCR engines were developed but the cost of the OCR engines was very high. The price of an engine was about one million US-dollars. Government agencies or large corporations used the OCR engine.

Today, an OCR system is very fast and can recognize several hundred characters per minute. Further this system is less expensive and capable of recognizing multi fonts and font sizes.

In the early days of pattern recognition research, characters were considered very handy to deal with and were regarded as a problem that could be solved easily. However, after some initial easy progress in the area, it became clear that the problem of character recognition is actually quite difficult [MOR92].

Two different purposes of commercial OCR systems are in use and they are task-specific readers and general purpose page readers. A task-specific reader involves particular document types. Examples of task-specific readers are address readers, form readers, cheque readers, Passport Readers etc. General-purpose page readers are designed to handle a broader range of documents such as business letters, technical writing and newspapers.

## 1.2 Overview of the OCR Technique

The OCR problem can be divided into seven stages. These stages are shown in the figure 1.1 and they are scanner, pre-processing, page segmentation, feature extraction, pattern classification, post-preprocessing and output file.

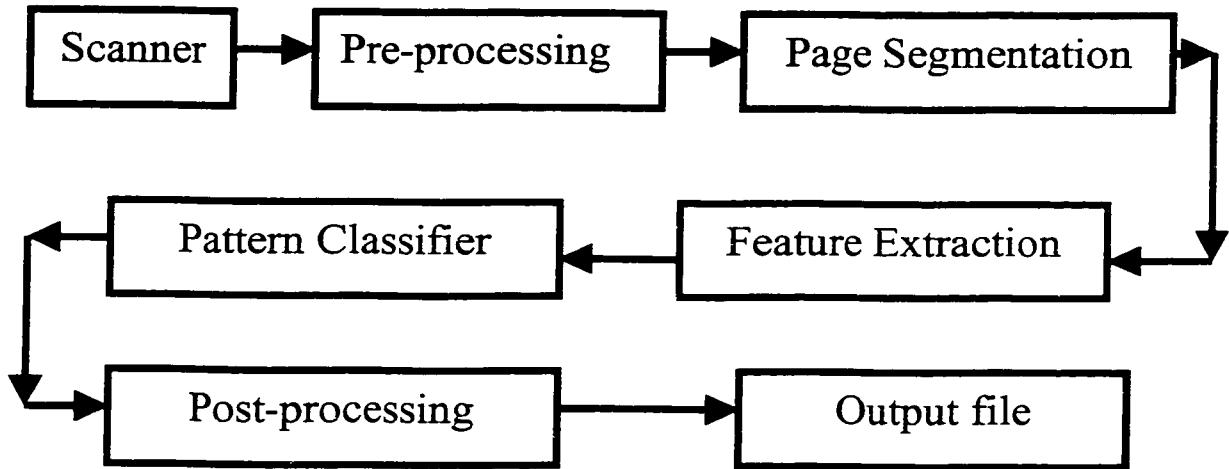


Figure 1.1: Stages in the OCR process

The scanner converts the paper document into an image document and saves the image into a file with one of the image formats such as TIFF, bmp etc.

In the pre-processing stage, the image is subjected to the following processes: binarization, noise elimination, skew estimation and correction. All three problems affect the OCR performance. More technical information on the subject can be found in the articles [SAU00], [OKU99], [MOR99] and [KAU98].

Automatic page segmentation of a digitized image is a necessary process of the OCR system capable of understanding text and non-text regions in the image [SAU95] [SAU97]. Non-text area means the region of graphics, halftone images, line drawings, etc.

Feature extraction is the extraction of image parts that are meaningful for the application. Many types of feature extractions exist; some of them are Fourier expansion, Zernike Moments, Karhunen-Loeve expansion, etc. More technical information on this subject can be found in the Mori [MOR99] book.

Pattern classifier assigns a feature vector to a decision region that corresponds to a class. The pattern classifier is discussed in more detail in the next section 1.3.

Post-processing (correction errors) is designed to improve the quality of text produced by an OCR device [TAG94]. This system is composed of numerous parts: an information retrieval system, dictionary, and a collection of algorithms and heuristics designed to correct as many OCR errors as possible [TAG94].

After completing the final correction, the recognized text is written down in the file. The file format can vary and it depends on the purpose of its application. Format files such as text, word, html, pdf, etc. are used.

## 1.3 Pattern Classifier

Pattern classification consists of assigning entities, described by feature vectors, to predefined groups of patterns. Two types of classifiers are discussed below: the traditional classifier and neural network Classifier.

### 1.3.1 Traditional Classifier

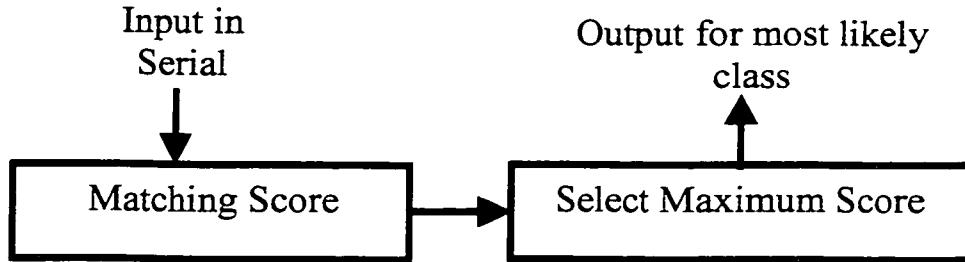


Figure 1.3.1: Traditional Classifier

The Traditional classifier contains two stages [LIP87] as shown in the figure 1.3.1. The two stages are matching score and maximum score selectors. In the first stage the feature vectors are entered sequentially and they are matched with the trained pattern. The matching score indicates the closeness of the input patterns to the trained patterns. A trained pattern means that the all-possible shapes of patterns are already stored in the database. The next stage takes the matching score and selects the best-matched class. Multivariate Gaussian distributions are often used for computing matching scores; however, this technique needs more storage space and also the process consumes time because it processes the data sequentially.

### 1.3.2 Neural Network Classifier

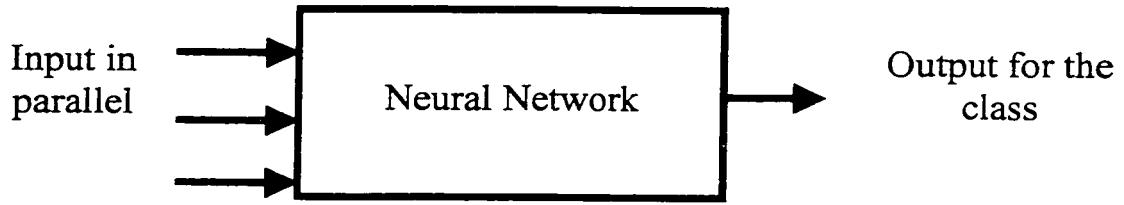


Figure 1.3.2: Neural Network Classifier

Neural network is a massively parallel computing system, which contains large numbers of simple processors and many interconnections [ANI00]. It takes input patterns in parallel and computes them in parallel to give the result. In neural networks, knowledge of each trained character is distributed through the system. The weights contain the knowledge of the trained characters. These weights are used to recognize unknown characters. The classification process is faster because the neural network is processing in parallel and storage space is less because the trained patterns are not stored in the memory.

## 1.4 Thesis Organization

In chapter I, history, application and importance of OCR are discussed. A survey of OCR classifier is also presented.

In chapter II, concepts of neural networks are described using biological neurons and the active function of a neural network is discussed.

In chapter III, various neural network algorithms are implemented and applied to Arabic number recognition and also their performance is discussed.

In chapter IV, a fast training algorithm is implemented for the Multi layer feed forward neural network. It is compared with the delta rule training algorithm. The both algorithms are applied for real English capital letters. The performance of both algorithms is discussed.

In chapter V, a summary of the research carried out for this thesis is presented and the conclusions are discussed.

The source codes of the program (written in Microsoft Visual C++) are given in the appendices.

# **Chapter II**

## **2.1 Neural Network**

The term “Neural Networks” (NN) always refers to Artificial Neural Network. As with most people, the term “Artificial” will be omitted in this thesis. So far, there is no universally accepted definition of a Neural Network. A Neural Network is a network of many simple processing units (neurons), each of the neurons having interconnections (weights) with other neurons. It has a training rule whereby the weights of connections are adjusted on the basis of data, which is an example of the data that will be used in the applications. Training means learning the relation between the input and the expected output. During this training, the NN develops the ability for generalization beyond the training data. After training, the weights are not altered and the NN makes decisions based on the weights’ strength, without resorting to any other resource in decision-making.

A Neural Network is an information-processing paradigm that is inspired by the way the biological nervous systems, such as the brain, processes the information.

But the biological neural networks are much more complicated than the mathematical models we use for NNs. We should take a look at the basic principle of the operation of biological neurons to understand the neural network properly.

## 2.2 Biological Neuron

A human being contains about  $10^{11}$  neurons in the brain [JOH91]. Neurons are a vast number of single, interconnected cellular units [MUL95]. Each single neuron is connected with other neurons in a complex structure to form a network.

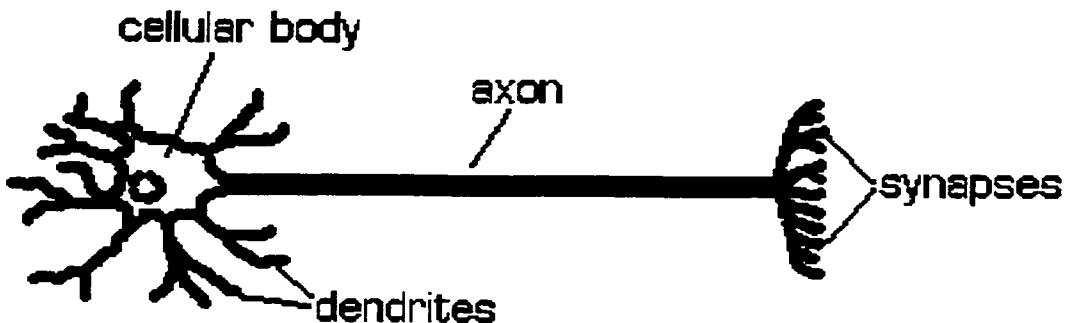


Figure 2.2: Single Biological Neuron

This complex network collects messages from other organs for information processing, and sends the collected messages to other organs, after decision making, to control their functions properly. A simple structure of a typical single neuron is shown in Figure 2.2.

The basic neuron consists of a cellular body (soma), dendrites, axon and synapses. A cellular body receives its input from its dendrites. The cellular body processes

the input and changes the electrical potential of its axon accordingly. The axon carries the output signals to other dendrites of neighboring neurons via synapses. The synapses are electrochemical connections between neurons. Therefore, the signals coming from different neurons are weighted according to the synapses' strength, and then the signals are summed up in the dendrites and passed on to the soma for a decision. The soma evaluates the signal and if the signal exceeds some threshold value, then the soma fires the output. The axon carries the output to the other neurons via the dendrites. During the learning process the synapses are altered (weakens or strengthens the connection) to get the desired response.

## 2.3 Artificial Neuron

Similar concepts of the biological neuron are applied in the artificial neuron. But still the artificial neuron is by far a simplified model compared to the biological neuron [KAR96].

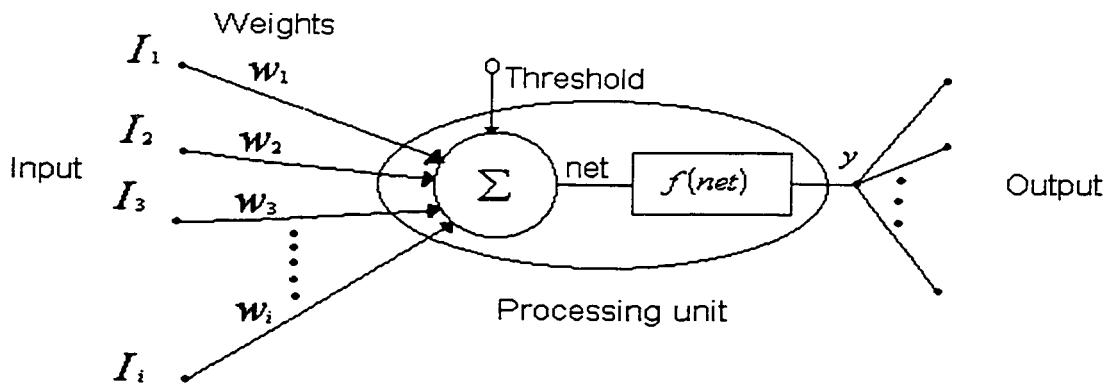


Figure 2.3: Simple mathematical artificial neuron

Like a biological model, a neural network also has neurons and connections between neurons. A simple mathematical model of a neuron is shown in Figure 2.3. An artificial neuron contains four components: input, weights, processing unit and output. The input is obtained (example  $I_1, I_2 \dots I_i$ ) from the other neurons or from the outside of the neural network. The weights ( $w_1, w_2 \dots w_i$ ) are connections between neurons which have an influence on the response of the neurons. A processing unit sums up the weighted input and threshold value and passes the signal through the activation function (transfer function) to the output. The output carries the output ( $y$ ) to the other neurons or outside of the neural network. During the processing of the information the input is multiplied by the appropriate weight and then passed through the transfer function, after summing up of all the weighted input and threshold value.

Now we can look at the artificial neuron from an engineering point of view. An artificial neuron can be divided into four areas such as activation rule, activation function, training rule, and connectivity rule.

### 2.3.1 Activation Rule

The activation rule is one that measures the input level of activity of the neuron. How does it measure the input level of activity? It measures the input level activity through simple addition and multiplication of input signals. The activation rule sums up the result of the input signal vector multiplied by appropriate weight (connection strength). Equation 2.3.1 shows the mathematical computation of the activation rule.

$$y = \sum_{i=0}^{N-1} I_i w_i \quad \dots \rightarrow 2.3.1$$

Where  $I_i$  – Input signal for the input node i

$w_i$  – Appropriate weight for the input node i

y - Activity level

N - Number of input (node)

### 2.3.2 Activation Function

The activation function is a function that is used to transform the activation level of an input neuron into an output signal. Typically, activation functions have a squashing effect. The activation function gives information to the processing neuron about how active are the connections to this neuron. Different types of activation functions exist [KAR 96], but in all these cases their purpose is to determine the neuron's output signal level as a function of the input signal level to the neuron.

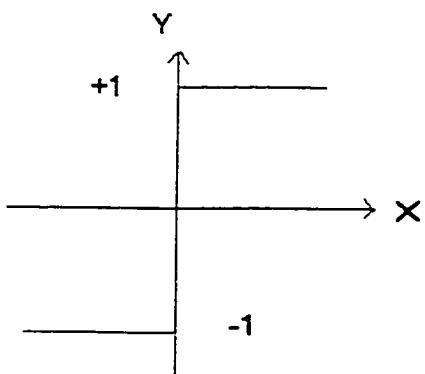


Figure 2.3.2a: Non-linear function (Step)

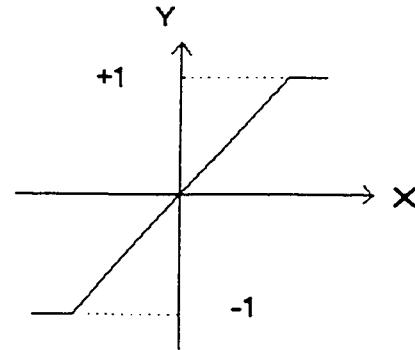


Figure 2.3.2b: linear function (Ramp)

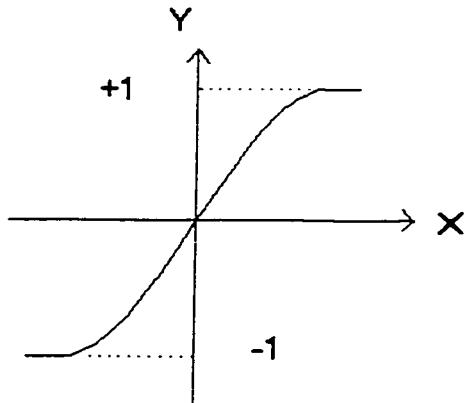


Figure 2.3.2c: Semi-Nonlinear function (Sigmoid)

There are three types of activation functions: linear, non-linear and semi-linear functions.

Examples of these types of functions are shown in the Figure 2.3.2.

### 2.3.3 Training Rule

Learning or training is a process of adjusting weights (connection strength) to get the desired response to each of the input signals. Thus the weights play an important role in determining the neuron response to the input signals. There are two types of training rules, they are the supervised and the unsupervised training rules.

#### 2.3.3.1 Supervised Training Rule

In this supervised training rule, the input and desired output are known before starting the training process. The supervised learning rule has a learning phase, which forces (teaches) the neural network (weights) to get the desired output response for the given

input. Examples that use this type of training neural networks are single perceptron and multi-layer perceptron neural networks.

### **2.3.3.2 Unsupervised Training Rule**

In the unsupervised training rule, only input is known prior to the training process. The training process starts with an input data and tries to find the relationship between the various training data. It then maps them in the output nodes of the network. Examples of using this type of training neural networks are Kohonen's neural network and the Hamming network.

### **2.3.4 Connectivity Rule**

Even though the neural network is inspired by the biological neuron system, the types of connections between the various nodes within a layer and the nodes in different layers are simple when compared to the complicated connections among the neurons of our brain. In a NN, there are any number of nodes in a layer and any number of layers in a network. The neural network behavior differs from one another depending on how the various layers are inter-connected among themselves. The first layer is called the input layer and it receives the input from the outside world or from the feed back connections of the neurons. The last layer is called the output layer and produces the output. The layers between the output and the input layers are called the hidden layers. There is no rule to say that a neural network should have hidden layers. Connections can be made in the following ways; examples of the types of neural network are listed side by side.

- i) Single layer and feed forward connection – single perceptron neural network and Kohonen's network
- ii) Single layer and feedback connection – Hopfield net and Carpenter\Grossberg network
- iii) Multi layer and feed forward connection – multi layer perceptron neural network
- iv) Multi layer and feedback connection – Hamming network

## 2.4 Training and Testing Set

In this thesis two types of data are used. One are the Arabic numerals (Chapter III) created by the author (12 by 10 pixels) and the English capital letters of Times New Roman with a font size of 16 (Chapter IV).

### 2.4.1 Training and Testing Set of Arabic Numerals

Arabic numbers are used for training and testing all the six neural networks. The numbers used for the training set are shown in the figure 2.4a. One sample of each of these numbers is used for training.

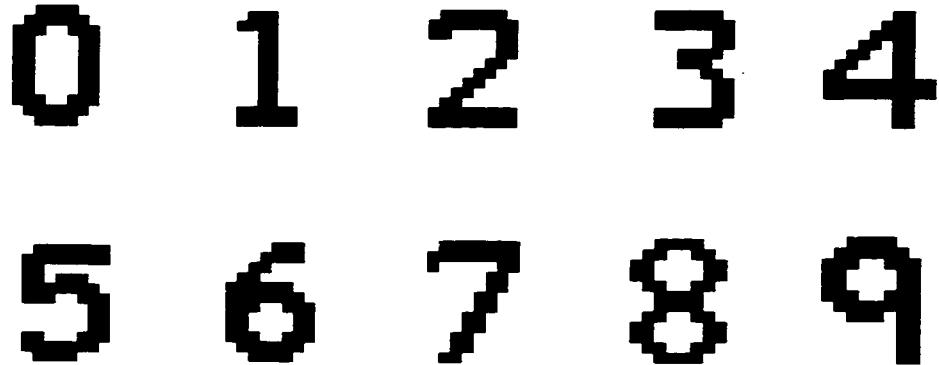
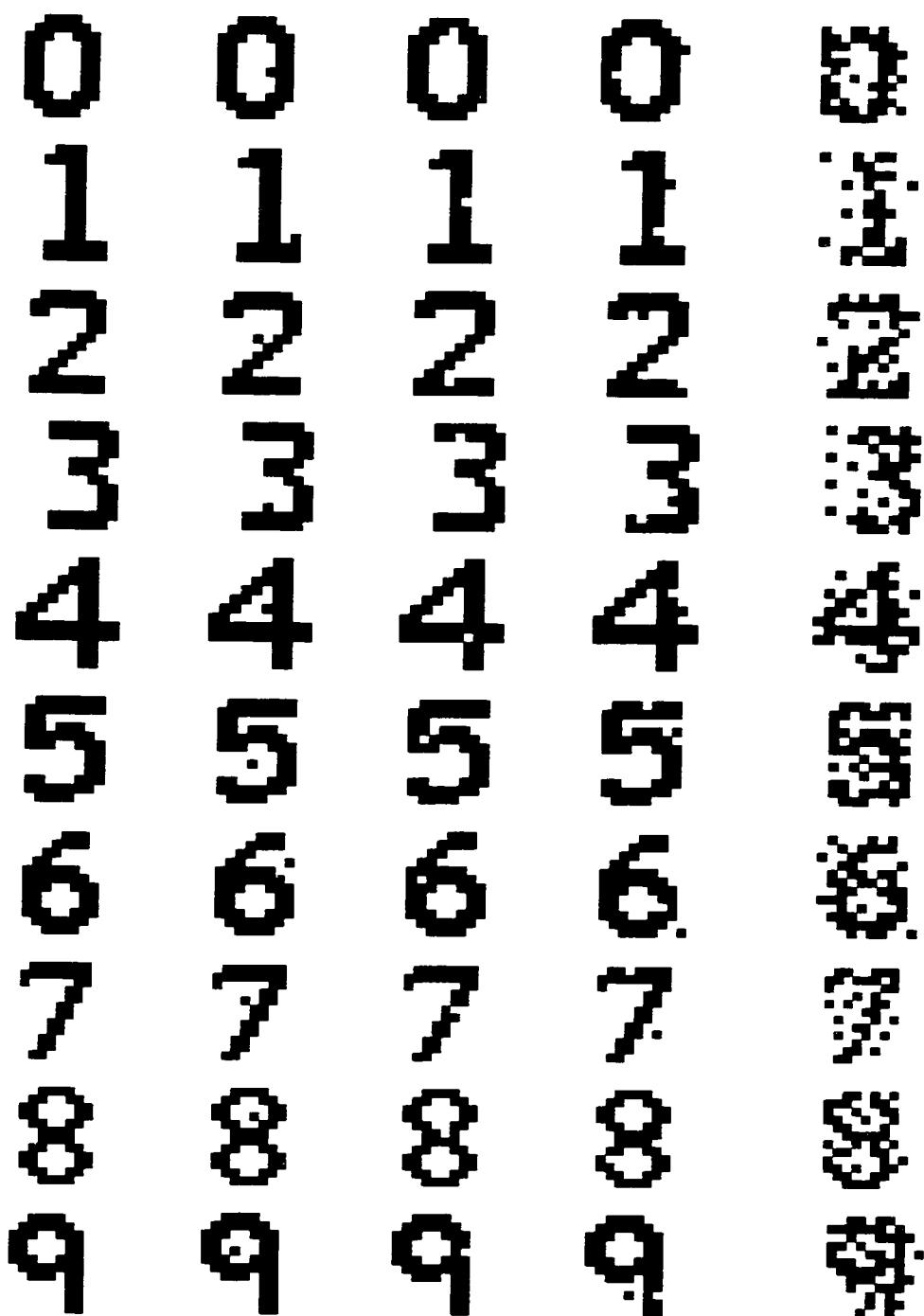


Figure 2.4.1a. Training set -Arabic numerals

The size of the testing image is 12 by 10 pixels. The testing set images are shown in Figures 2.4b and 2.4c. To measure the testing speed of the trained NNs, first, images (Without Noise images) of all the numbers in the figure 2.4b is used. For the noise sensitivity test, all the images in the figure 2.4b are used. The images in the figure 2.4c are used for the translated image test in the chapter III.



Without Noise	Added 1 pixel	Removed 1 pixel	Added & Removed 1 pixel	Added & Removed more than one pixel
------------------	------------------	--------------------	----------------------------	--

Figure 2.4.1b. Testing Set – Arabic numerals from 0 to 9

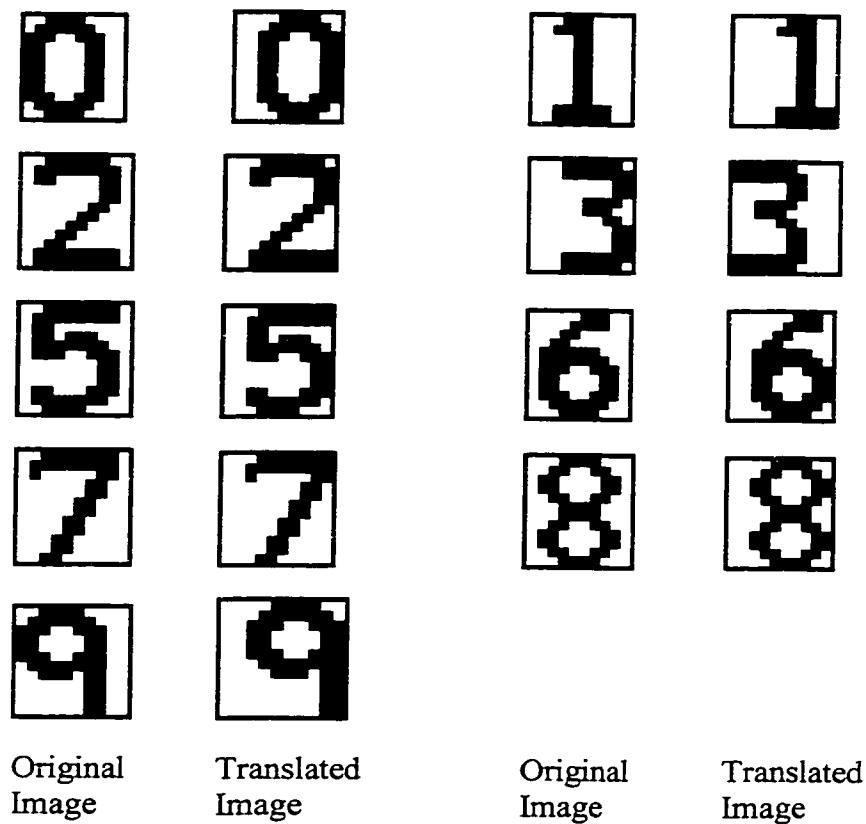


Figure 4.4.1c. Testing Set – Translated Images

## 2.4.2 Training and Testing Set of English Capital Letters

The training and testing sets are used to compare the Fast training algorithm and the Delta training rule algorithm in Chapter IV. In this experiment, all the English capital letters are used in the training set as well as in the testing set. The font used is Times New Roman of font size 16. A total of 650 samples are used in the training set.

$$\begin{aligned}
 \text{Total training samples} &= 25 \text{ samples/character} * 26 \text{ character} \\
 &= 650 \text{ samples}
 \end{aligned}$$

The images shown in Figures 2.4.2a, 2.4.2b, 2.4.2c and 2.4.2d are used for training the NN using the both algorithms. One hundred and thirty samples are used in this testing set.

$$\begin{aligned}\text{Total testing samples} &= 5 \text{ samples / character} * 26 \text{ characters} \\ &= 130 \text{ samples}\end{aligned}$$

The images used for testing are shown in the figure 2.4.2e.

Training Set														
A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
B	B	B	B	B	B	B	B	B	B	B	B	B	B	B
B	B	B	B	B	B	B	B	B	B	B	B	B	B	B
C	C	C	C	C	C	C	C	C	C	C	C	C	C	C
C	C	C	C	C	C	C	C	C	C	C	C	C	C	C
D	D	D	D	D	D	D	D	D	D	D	D	D	D	D
D	D	D	D	D	D	D	D	D	D	D	D	D	D	D
E	E	E	E	E	E	E	E	E	E	E	E	E	E	E
E	E	E	E	E	E	E	E	E	E	E	E	E	E	E
F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
G	G	G	G	G	G	G	G	G	G	G	G	G	G	G
G	G	G	G	G	G	G	G	G	G	G	G	G	G	G
H	H	H	H	H	H	H	H	H	H	H	H	H	H	H
H	H	H	H	H	H	H	H	H	H	H	H	H	H	H

Figure 2.4.2a Training Set – Samples from A to H

Training Set														
I	I	I	I	I	I	I	I	I	I	I	I	I	I	I
I	I	I	I	I	I	I	I	I	I	I	I	I	I	I
J	J	J	J	J	J	J	J	J	J	J	J	J	J	J
J	J	J	J	J	J	J	J	J	J	J	J	J	J	J
K	K	K	K	K	K	K	K	K	K	K	K	K	K	K
K	K	K	K	K	K	K	K	K	K	K	K	K	K	K
L	L	L	L	L	L	L	L	L	L	L	L	L	L	L
L	L	L	L	L	L	L	L	L	L	L	L	L	L	L
M	M	M	M	M	M	M	M	M	M	M	M	M	M	M
M	M	M	M	M	M	M	M	M	M	M	M	M	M	M
N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
P	P	P	P	P	P	P	P	P	P	P	P	P	P	P
P	P	P	P	P	P	P	P	P	P	P	P	P	P	P

Figure 2.4.2b Training Set – Samples from I to P

Training Set															
Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q
Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q	Q
R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R
R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R
S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S
S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S
T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T
T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T
U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U
V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V
V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V
W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W
W	W	W	W	W	W	W	W	W	W	W	W	W	W	W	W
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Figure 2.4.2c Training Set – Samples from Q to X

Training Set															
Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z
Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z

Figure 2.4.2d Training Set – Samples of Y and Z

Testing Set															
A	A	A	A	A	B	B	B	B	B	C	C	C	C	C	C
C	C	D	D	D	D	D	E	E	E	E	E	E	F		
F	F	F	F	G	G	G	G	G	G	H	H	H	H	H	H
H	I	I	I	I	I	J	J	J	J	J	K	K	K	K	K
K	K	K	L	L	L	L	L	M	M	M	M	M	M	M	M
N	N	N	N	N	N	O	O	O	O	O	P	P	P	P	P
P	P	Q	Q	Q	Q	Q	R	R	R	R	R	R	S		
S	S	S	S	T	T	T	T	T	T	U	U	U	U	U	U
U	V	V	V	V	V	W	W	W	W	W	W	X	X	X	X
X	X	X	Y	Y	Y	Y	Y	Z	Z	Z	Z	Z	Z	Z	Z

Figure 2.4.2e Testing Set

# **Chapter III**

## **3.1 Various Neural Network Algorithms and their Performances**

The behavior of a neural network depends on the connections between the neurons, the number of layers and the training methods. Connections to a neural network can be either a feedforward connection or feedback connection. In the feedforward connection, the data from the neurons of a lower layer are propagated forward to the neurons of an upper layer. Feedback connections bring the data from the neurons of an upper layer back to the neurons of a lower layer. In any neural network, there will be an input layer and an output layer. Another type of layer called hidden layer, lies between the input and the output layers. Any number of hidden layers can be in a network. The first such layer is called the first hidden layer, second one is called the second hidden layer, and so on. There are two types of training methods and they are the supervised and the unsupervised methods.

There are two phases of information processing; the learning phase and the retrieval phase. In the learning phase, training set (example of the application data) is used to

determine the neural network's weight parameter. In the retrieving phase, the trained neural network model will be used to process the given patterns and classify the patterns.

The Neural Networks in Figure 3.1 are grouped on the basis of their types of connections between the neurons and the learning methods. First they are divided into NNs with supervised and unsupervised training. Then the NNs are divided into NNs with feedback and non-feedback connections. More discussion on each of these neural networks is carried out in the coming sections.

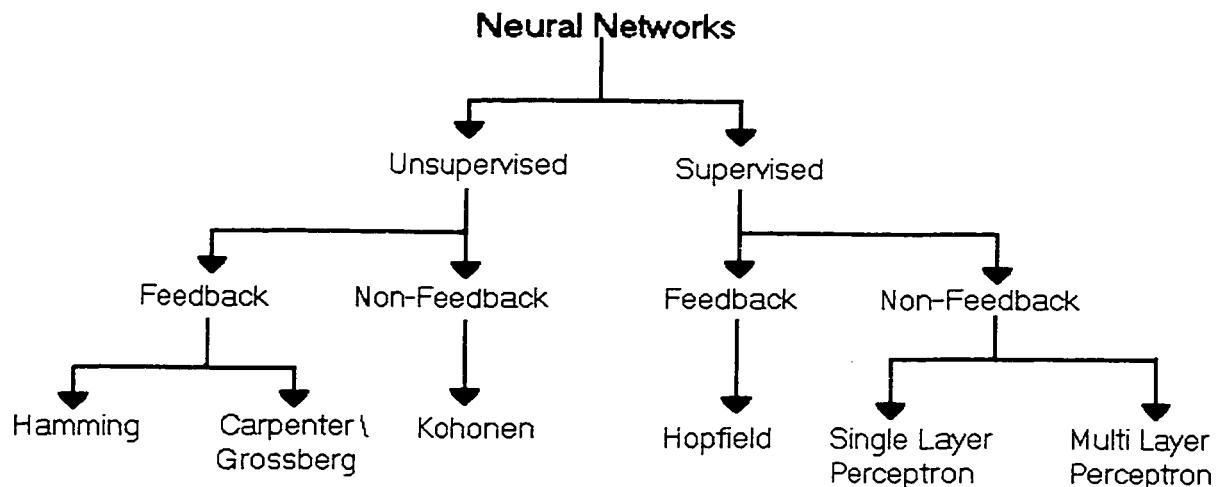


Figure 3.1 A taxonomy of neural networks

In section 3.2, six neural networks with different types of connections, layer structure and learning methods are described. In section 3.3, experimental procedures are described. In section 3.4, the results are given and these results of each neural network are discussed in section 3.5.

## 3.2 Neural Networks Connections, Layers and Learning Methods

First, neural networks with feedback connections are considered and then NN with non-feedback connections are considered.

### 3.2.1 The Hamming Neural Network

The layers and connections of the Hamming neural network are shown in the Figure

3.2.1. There are three layers: the input layer, one hidden layer and the output layer. The connection between the first layer and the hidden layer is a feedforward connection. The connection goes through the multiplier of the weight vector. The connection between the hidden and the output layers is also a feedforward connection. This feedforward connection with the output layer is there at the time of initialization and is not there after initialization. Each of the output layer nodes is connected to itself and to every other node of the output layer by the feedback connection. These feedback connections go through the multiplier of the weight vector.

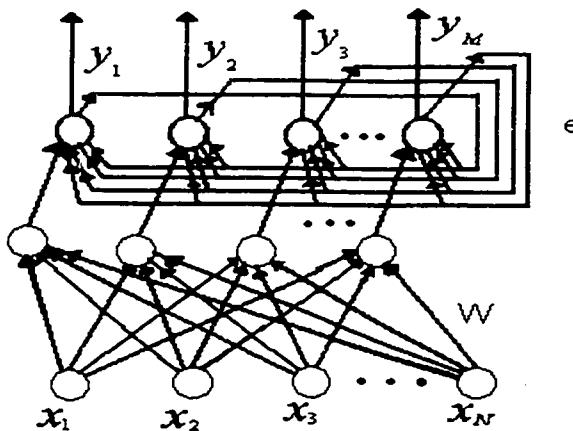


Figure 3.2.1 The Hamming Neural Network connections and nodes in each layer

The unsupervised training method is applied for training this neural network. A binary image is applied to the input layer. The weights are trained by storing all the binary information in the weights directed towards one node for one pattern. Similarly for a second pattern, a second node is selected and the weights directed towards that node are trained by storing the image information in these weights. In a mathematical way, it can be summarized in a simple equation (equation 3.2.1a) to show the simple training process:

$$w_{ij} = \frac{x_i^j}{2} \quad 3.2.1a$$

where

$$0 \leq i < N \text{ and } 0 \leq j < M$$

Where  $x$  is the input element of training pattern  $j$ , which is either one or zero and  $w_{ij}$  is weight connection between input node  $i$  to the middle node  $j$ . The number of training patterns is equal to the number of nodes in the middle layer.  $N$  is the number of information about the input image (number of input to the NN).  $M$  is the number of classes in the training patterns. Each of the middle nodes has a threshold value. It has been set as follows: first, count the number of black pixels in each of the training patterns, select the minimum values and then divide them by two. Equation 3.2.1b and 3.2.1c show the process.

$$M_{black\_pixel} = \min(count_{black\_pixel}(x^j)) \quad 3.2.1b$$

$$\theta_j = \frac{M_{black\_pixel}}{2} \quad 3.2.1c$$

$$\varepsilon = \frac{0.9}{M} \quad 3.2.1d$$

The feedback weights in the output layer are set with small constant values while the feedback to itself is always set at one. Equation 3.2.1.d shows the setting of the feedback weights, where M is the number of classes in the training patterns.

Testing of this neural network is different to that of training the NN. First, compute the middle nodes' values using the testing pattern, the feedforward weights and the threshold value. This process is called matching the pattern with trained patterns. The calculation is shown below

$$y_j(0) = f_t \left[ \sum_{i=0}^{N-1} w_{ij}x_i - \Theta_j \right], \quad 0 \leq j < M \quad 3.2.1e$$

Where  $f_t$  is the active function and it is a ramp function going from 0 to N/2 (where N is the number of input nodes). Initially (t=0) the output nodes are set using this middle layer nodes' values. Then, start the iteration to pick the maximum score from the previous matching score. This procedure can be shown mathematically as follows:

$$y_j(t+1) = f_t \left[ y_j(t) - \sum_{k \neq j} y_k(t) \right], \quad 0 \leq j, k < M \quad 3.2.1f$$

The above process is repeated until the output nodes of only one node remains positive while the others are zero. The name of the class belonging to the positive node is assigned to this tested pattern.

### 3.2.2 The Carpenter \ Grossberg Neural Network

This NN is not like the Hamming NN; it has two layers and they are the input layer and the output layer. There are feedforward connections via feedforward weights from the input layer to the output layer. Feedback connections are made via feedback weights from the output layer to the input layer. The layers and connections of the Hamming neural network are shown in Figure 2.2.2.

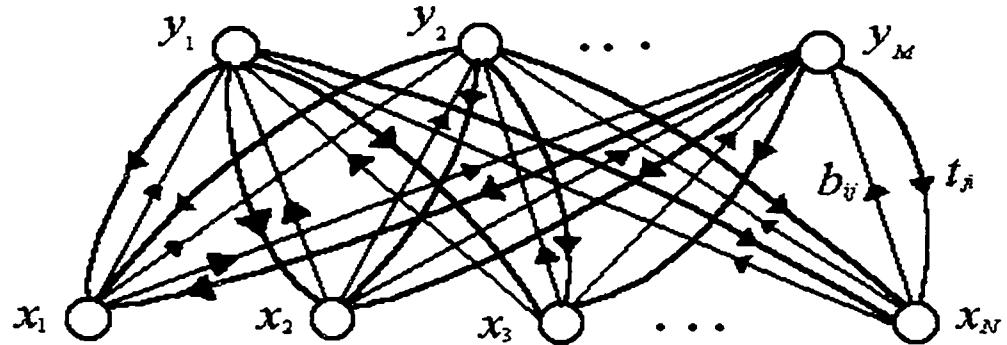


Figure 3.2.2 The Carpenter \ Grossberg Neural Network connections and nodes in each layer

The training method of Carpenter \ Grossberg NN is an unsupervised training method. The clustering algorithm and vigilance test are used in the training process. The clustering algorithm clusters the input pattern and the vigilance test decides which pattern is assigned to which node. The matching algorithm is shown in equation 3.2.2a

$$y_j = \sum_{i=0}^{N-1} b_{ij}(t)x_i, \quad 0 \leq j < M \quad 3.2.2a$$

Where  $y_j$  is the output value of the output node  $j$ ,  $x_i$  is the input of the training pattern, which is the binary value 1 or 0 and  $b_{ij}$  are the feedforward weights and they are initially set with small values. The feedback weights are initially set to one. The initial setting of the feedforward ( $b_{ij}$ ) and feedback ( $t_{ji}$ ) weights is shown below.

$$b_{ij}(0) = \frac{1}{1 + N} \quad 3.2.2b$$

$$t_{ij}(0) = 1 \quad 3.2.2c$$

$$0 \leq i < N, \quad 0 \leq j < M$$

$$\rho = 0.8$$

Where  $\rho$  is the vigilance threshold, which indicates how close an input pattern should be to a stored pattern to be matched. After the matching score is calculated, the best matching score node ( $j^*$ ) is selected.

$$j^* = \text{Node } j_{\max}(y_j) \quad 3.2.2d$$

The selected node ( $j^*$ ) is tested for vigilance test as follow.

$$X = \sum_{i=0}^{N-1} x_i \quad 3.2.2e$$

$$TX = \sum_{i=0}^{N-1} t_{ij} \cdot x_i \quad 3.2.2f$$

$$\rho < \frac{TX}{X}$$

3.2.2g

If the last equation (eq 3.2.2g) is false, go back and select the next node with the maximum score and do the vigilance test again. If the last equation is true, then adapt the weights to the selected node only. The adapting algorithms for both weights are given below

$$t_{ij}^*(t+1) = t_{ij}^*(t) x_i \quad 3.2.2h$$

$$b_{ij}^* = \frac{t_{ij}^*(t) x_i}{0.5 + \sum_{i=0}^{N-1} t_{ij}^*(t) x_i} \quad 3.2.2i$$

Repeat these steps for the other patterns without initializing the weights. The recognition of the unknown pattern is similar to the training process except the adoption of weights and initialization of weights is not done. The NN takes the trained weights and computes the matching score for the testing pattern using equation 3.2.2a. Now, select the node of the maximum matching score and then perform the Vigilance test. If the test is true, the class of the node is assigned to the test pattern. Otherwise the tested pattern is rejected.

### 3.2.3 Kohonen's Self Organizing Feature MAPS Neural Network

The NN contains two layers; the input layer and the output layer. Kohonen's neural network has only feedforward connections. The entire input nodes are connected to all the output nodes via the variable connection weights. In this NN, weights are not multiplied, but they are subtracted from the input. The output nodes are arranged in a form of two-dimensional array. Figure 3.2.3 shows the connections of the Kohonen's NN.

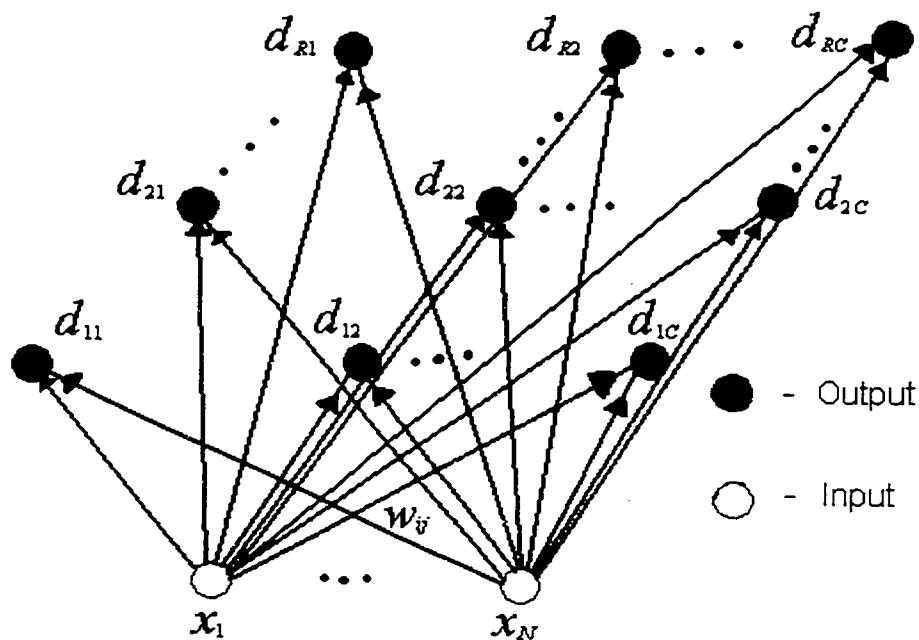


Figure 3.2.3 The Kohonen's Neural Network connections and nodes in each layer

The training process of Kohonen's NN is an unsupervised training method. The training algorithm is called the self-organizing feature map [KOH84]. When a training input pattern is presented to the input layer, the distance between the input pattern and the

nodes of the output layer values are calculated. Equation 3.2.3a indicates the mathematical process.

$$d_j = \sum_{i=0}^{N-1} (x_i^s - w_{ij})^2 \quad 3.2.3a$$

Where  $d_j$  is the sum of the squares of the distance between the input nodes and the output node  $j$ ,  $x_i^s$  is an input value of the input node  $i$  for the training pattern  $s$  and  $w_{ij}$  is the weight connection from the input node  $i$  to the output node  $j$ . These weights are initially set to small random values. Then select the minimum distance as shown in equation 3.2.3b, which is the winning output node that produces the strongest response. After determining the minimum distance node, weights are adjusted for the minimum distance node as well as for all the other output nodes within its neighborhood. The weights are updated using the learning rate, and the distance between the output node and the weight connected to the input node. Equation 3.2.3c is used for the updating of weights.

$$j^* = \min(d_j) \quad 3.2.3b$$

$$w_{ij}(t+1) = w_{ij}(t) + \eta(t)(x_i^s - w_{ij}(t)) \quad 3.2.3c$$

Where  $\eta(t)$ , the learning rate function ( $1 > \eta(t) > 0$ ), is one that decreases with time. This training process continues for all training patterns. In the recognition process, the unknown pattern is applied to the input layer and then distances are computed from each

output node using equation 3.2.3a. The minimum distance node class is selected for the unknown pattern.

### 3.2.4 The Hopfield Neural Network

The neural network contains only one layer. The input feeds to the layer at time zero and then the feedback from the output nodes are used as input. Output nodes are fed back to the inputs via variable connection weights. There are no self-feedback connection weights in this NN. Figure 3.2.4 shows the connections and the layers of the NN.

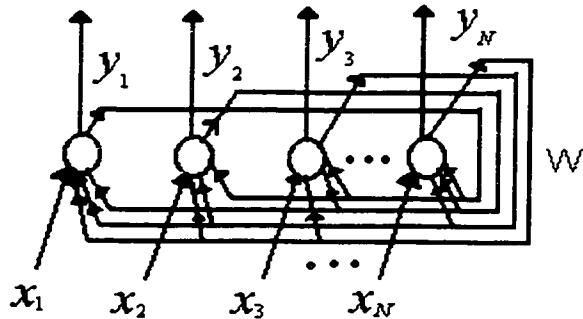


Figure 3.2.4 The Hopfield Neural Network connections and nodes in each layer

The training method for the Hopfield NN is a supervised training method. The NN knows the input and the output values of the NN before the training process. The only unknown for this NN is weights. The NN takes the binary input values +1 and -1. The training of this network is done by the Hebb rule. The training process is given below

$$w_{ij} = \sum_{s=0}^{M-1} x_i^s x_j^s, \quad i \neq j \quad 3.2.4a$$

and

$$w_{ij} = 0, \quad i = j, \quad 0 \leq i, j < N \quad 3.2.4b$$

Where  $w_{ij}$  is the weight connection between the input node  $i$  and the output node  $j$ ,  $x_i^s$  is the input element of training pattern  $s$ ,  $M$  is the number of patterns used for training and  $N$  is the number of nodes in the layer (number of input components).

During the recognition process, an unknown input is applied to the NN at time zero. Then the NN forces the output to match one of the trained patterns. The recognition process is shown in equations 3.2.4c and 3.2.4d.

$$y_j(0) = x_j, \quad 0 \leq j < N \quad 3.2.4c$$

$$y_j(t+1) = f_h \left[ \sum_{i=0}^{N-1} w_{ij} y_i(t) \right], \quad 0 \leq j < N \quad 3.2.4d$$

Where  $y_j$  is the output of the output node  $j$ ,  $w_{ij}$  is the connecting weight from the output of the output node  $i$  to the input of the output node  $j$ ,  $N$  is the number of input elements and the  $f_h$  is the hard limiting nonlinearity active function. The active function is a step function, which goes from  $-1$  to  $+1$  at the origin (eq 3.2.4e and eq 3.2.4f). The recognition process (eq 3.2.4d) continues until there is no change in the output states on successive iterations.

$$f_h(z) = 1, \quad z \geq 0 \quad 3.2.4e$$

$$f_h(z) = 0, \quad z < 0 \quad 3.2.4f$$

### 3.2.5 The Single Layer Perceptron Neural Network

The neural network contains two layers; the input layer and the output layer. The connections between the output nodes and the input nodes are feedforward connections via variable weights. The connections and layers of the single layer perceptron are shown in the Figure 3.2.5.

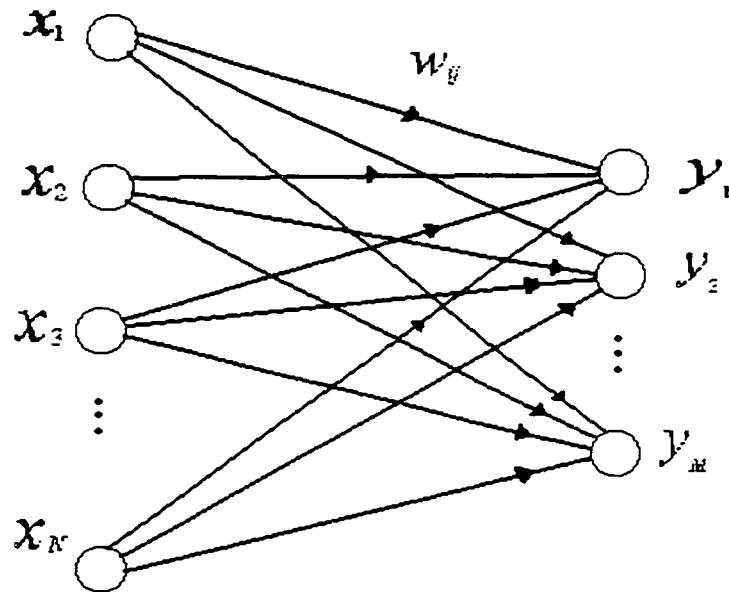


Figure 3.2.5 The Single Layer Perceptron Neural Network connections and nodes in each layer

The supervised learning method is used to train this NN. Here the input and the input's target output values are applied to the input nodes and output nodes of the neural network respectively and then the sum of weights multiplied by the input are computed. The

computed sum is passed through the active function. The process is represented by the equation given below:

$$y_j(t) = f_h \left[ \sum_{i=0}^{N-1} w_{ij}(t) x_i^s - \Theta_j \right] \quad 3.2.5a$$

Where  $y_j$  is the output value of the output node  $j$ ,  $x_i^s$  is the input element  $i$  of the training pattern  $s$ ,  $\Theta_j$  is the threshold value of the node  $j$  and  $w_{ij}$  is the connection weights from the input node  $i$  to the output node  $j$ . The weights are initialized with small random values. The  $f_h$  is the hard limiting nonlinearity active function. This active function is a step function, which goes from  $-1$  to  $+1$  at the origin (eq 3.2.4e and eq 3.2.4f). After computing the output node values, the weights are updated according to the error in each of the output nodes. Equation 3.2.5b indicates the adoption of the weight from input node  $i$  to output node  $j$ .

$$w_{ij}(t+1) = w_{ij}(t) + \eta (d_j(t) - y_j(t)) x_i^s \quad 3.2.5b$$

Where  $d_j(t)$  is the desired output (target output) of the output node  $j$ ,  $y_j$  is the calculated output of the output node  $j$ ,  $x_i^s$  is the input element of the currently training pattern  $s$  to the input node  $i$  and  $\eta$  is the learning rate. The training process is repeated for other training patterns.

Simply, the recognition of an unknown pattern is carried out as follows: the unknown pattern is given at the input nodes and then the input is weighted using trained weights as in Equation 3.2.5a. If only one of the output nodes is close (the threshold value must be set by user) to one and the other nodes are close to zero, then the node class that is close to one is recognized as the class of the unknown pattern. Otherwise the unknown pattern is rejected.

### 3.2.6 The Multi Layer Perceptron Neural Network

The NN contains many layers; the input layer, the hidden layers and the output layer. The connections and layers of a multi layer NN is shown in Figure 3.2.6. It has only one hidden layer. The input nodes and the hidden nodes are connected via variable weights using the feedforward connections. The outputs of hidden nodes are connected to the input of the output layer nodes via weights. There are no direct connections between the input nodes and the output nodes. The hidden nodes are not connected to the outside world.

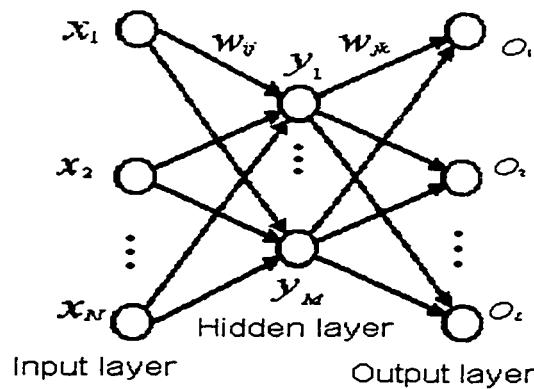


Figure 3.2.6 The Multi Layer Perceptron Neural Network connections and nodes in each layer

The multi layer neural network uses the supervised training method to train the NN. The training algorithms are called backpropagation training algorithms. The standard backpropagation algorithm is a generalized delta rule. The generalized delta rule is used here to train the multi layer neural network. The input elements (training pattern) and target values (particular input's target value) are presented to the input nodes and the output nodes of the NN respectively. All the weights are initialized with small random values at the beginning of the training process. The hidden nodes' values are computed as shown in Equation 3.2.6a. The input are weighted and then passed through the active function.

$$y_j(t) = f_s(z_j(t)) \quad 3.2.6a$$

$$\text{Where } z_j(t) = \sum_{i=0}^{N-1} w_{ij}(t) x_i^s - \Theta_j \quad 3.2.6b$$

Where  $y_j$  is the output of the hidden node  $j$ ,  $x_i^s$  is the input element of the input node  $i$  of the training pattern  $s$ ,  $\Theta_j$  is a threshold value of the hidden node  $j$ ,  $w_{ij}$  is the connection weight from the input node  $i$  to the hidden node  $j$  and  $f_s$  is the partially non-linearity active function, which is a sigmoid function going from  $-1$  to  $+1$  (eq 3.2.6c). These hidden nodes' values are propagated in the forward direction to the output layer nodes. The output of the output layer nodes are calculated in a similar way as the output of the hidden layer nodes are computed. The output of the output layer nodes are calculated using equation 3.2.6d.

$$f_s(z_j(t)) = \frac{2}{1+e^{-z_j(t)}} - 1 \quad 3.2.6c$$

$$O_l(t) = f_s(z_l(t)) \quad 3.2.6d$$

$$\text{Where } z_l(t) = \sum_{j=0}^{M-1} w_{jl}(t) y_j - \Theta_l \quad 3.2.6e$$

Where  $O_l$  is the output of the output node  $j$ ,  $y_j$  is the output of the hidden node  $j$ ,  $\Theta_l$  is a threshold value of the output node  $l$  and  $w_{jl}$  is the connection weights from the hidden node  $j$  to the output node  $l$ . The calculated output of the output nodes' values are compared with target output values. Total mean square of errors,  $E$ , is computed using all the training patterns of the calculated output and the target output. The calculation of total error,  $E_T$ , is defined in equation 3.2.6f.

$$E_T = \frac{1}{2} \sum_{k=0}^{P-1} \sum_{l=0}^{M-1} (t_l^k - O_l^k)^2 \quad 3.2.6f$$

Where  $O_l^k$  is the calculated output value of the output node  $l$  for the training pattern  $k$ ,  $t_l^k$  is the target value of the node  $l$  for the pattern  $k$ ,  $P$  is the number of training patterns and  $M$  is the number of output nodes in the output layer. The total error can be written as a function of weights. The delta rule, computes the gradient of the total error with respect to each weight and then the weights are altered in a direction opposite to the measured gradient. The algorithm for updating the weight is defined in the equation 3.2.6g.

$$w(t+1) = w(t) - \eta \frac{\partial E_r}{\partial w} \quad 3.2.6g$$

Where  $w(t)$  is the current weight,  $w(t+1)$  is the new weight,  $\eta$  is the learning rate

constant and  $\frac{\partial E}{\partial w}$  is the gradient of the total error with respect to the updating weight.

Calculation of the gradient of the error with respect to the weights between the hidden layer and the output layer is given below. For simplicity of the calculation, the derivation given below is for only one training pattern.

Using the chain rule,  $\frac{\partial E}{\partial w_{jl}}$  is

$$\frac{\partial E}{\partial w_{jl}} = \frac{\partial E}{\partial O_l} \frac{\partial O_l}{\partial z_l} \frac{\partial z_l}{\partial w_{jl}} \quad 3.2.6h$$

From eq 3.2.6f and eq 3.2.6h

$$\frac{\partial E}{\partial O_l} = \frac{1}{2} 2(t_l - O_l)(-1)$$

$$\frac{\partial E}{\partial O_l} = (O_l - t_l) \quad 3.2.6i$$

From eq 3.2.6c, eq 3.2.6d and 3.2.6h

$$\frac{\partial O_l}{\partial z_l} = \frac{\partial \left( \frac{2}{1 + e^{-z_l}} - 1 \right)}{\partial z_l}$$

$$\frac{\partial O_l}{\partial z_l} = \frac{2e^{-z_l}}{(1+e^{-z_l})^2} \quad 3.2.6j$$

Rewrite the right hand side of equation 3.2.6j in terms of  $O_l$

$$\frac{2e^{-z_l}}{(1+e^{-z_l})^2} = \frac{1}{2} \left( 1 - \left( 1 - \frac{2}{1+e^{-z_l}} \right)^2 \right) = \frac{1}{2} \left( 1 - (O_l)^2 \right)$$

$$\frac{\partial O_l}{\partial z_l} = \frac{1}{2} \left( 1 - (O_l)^2 \right) \quad 3.2.6k$$

From equation 3.2.6e and eq 3.2.6h

$$\frac{\partial z_l}{\partial w_{jl}} = \frac{\partial \left( \sum_{j=0}^{M-1} w_{jl}(t) y_j - \Theta_l \right)}{\partial w_{jl}} \quad 3.2.6l$$

$$\frac{\partial z_l}{\partial w_{jl}} = y_j$$

From equation 3.2.6h, eq 3.2.6i, eq 3.2.6k and eq 3.2.6l

$$\frac{\partial E}{\partial w_{jl}} = \frac{1}{2} \left( 1 - (O_l)^2 \right) (O_l - t_l) y_j = \delta_l y_j \quad 3.2.6m$$

$$\text{where } \delta_l = \frac{1}{2} \left( 1 - (O_l)^2 \right) (O_l - t_l) \quad 3.2.6n$$

The equation 3.2.6m shows the gradient of one pattern error  $E$  (output node l) with respect to weight  $w_{jl}$  (connection weight hidden node j to output node l). For all the training patterns, from equation 3.2.6m

$$\frac{\partial E_T}{\partial w_{jl}} = \sum_{k=0}^{P-1} \delta_i^k y_j^k \quad 3.2.6o$$

Where P is the number of patterns in the training set and  $\delta_i^k$  is the gradient of the error at node i with respect to weight  $w_{jl}$  for the training pattern k. The weights between the hidden nodes and the output nodes are updated after computing the gradient of the total error with respect to the weights. The equation for updating of the weights (between the hidden nodes and the output nodes) is given below.

From equation 3.2.6g and eq 3.2.6o

$$w_{jl}(t+1) = w_{jl}(t) - \eta \frac{\partial E_T}{\partial w_{jl}} \quad 3.2.6p$$

Consider updating the weights between the input and the hidden layers. The gradient of the error with respect to the weights between the input and the hidden layers can be calculated as follows:

First, find the gradient of the error of one output node with respect to the hidden nodes. Then, to get the actual gradient of the error with respect to the hidden nodes add all the gradients of error of all the output nodes with respect to the hidden nodes.

The gradient from one output node i, using the chain rule

$$\frac{\partial E}{\partial y_j^l} = \frac{\partial E}{\partial O_l} \frac{\partial O_l}{\partial Z_l} \frac{\partial Z_l}{\partial y_j^l} \quad 3.2.6q$$

From equation 3.2.6i and eq 3.2.6k

$$\frac{\partial E}{\partial Z_l} = \frac{1}{2} (1 - (O_l)^2) (O_l - t_l)$$

Using the above equation with eq 3.2.6n

$$\frac{\partial E}{\partial Z_l} = \delta_l \quad 3.2.6r$$

From equations 3.2.6q and 3.2.6e

$$\frac{\partial Z_l}{\partial y_j^l} = \frac{\partial \left( \sum_{j=0}^{M-1} w_{jl}(t) y_j - \Theta_l \right)}{\partial y_j^l}$$

$$\frac{\partial Z_l}{\partial y_j^l} = w_{jl}(t) \quad 3.2.6s$$

From eq. 3.2.6r and eq. 3.2.6s

$$\frac{\partial E}{\partial y_j^l} = \delta_l w_{jl}$$

The above equation applies only to one output node l. Therefore, the actual gradient for the hidden node j is

$$\frac{\partial E}{\partial y_j} = \sum_{l=0}^{L-1} \delta_l w_{jl} \quad 3.2.6t$$

Gradient of error with respect to weights

Using the chain rule,  $\frac{\partial E}{\partial w_{ij}}$  is

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial z_j} \frac{\partial z_j}{\partial w_{ij}} \quad 3.2.6u$$

From eq.3.2.6u, eq.3.2.6a, eq.3.2.6c and 3.2.6k

$$\frac{\partial y_j}{\partial z_j} = \frac{1}{2} \left( 1 - (y_j)^2 \right) \quad 3.2.6v$$

From eq.3.2.6u, eq.3.2.6b and 3.2.6k

$$\frac{\partial z_j}{\partial w_{ij}} = x_i^s \quad 3.2.6w$$

From equation 3.2.6.t, eq.23.2.6u, eq.3.2.6v and eq.2.3.6w

$$\frac{\partial E}{\partial w_{ij}} = \left( \frac{1}{2} \left( 1 - (y_j)^2 \right) \sum_{l=0}^{L-1} w_{lj} \delta_l \right) x_i^s = \delta_j x_i^s \quad 3.2.6x$$

The equation 3.2.6x shows the gradient of a pattern error, E, with respect to weight  $w_{ij}$  (connection weight of the hidden node j to the output node l). The gradient of the pattern error for all training patterns, from equation 3.2.6x is given by

$$\frac{\partial E_T}{\partial w_{ij}} = \sum_{s=0}^{P-1} \delta_j^s x_i^s \quad 3.2.6y$$

Where P is the number of patterns in the training set and  $\delta_j^s$  is the gradient of the error at the hidden node j with respect to the weight  $w_{ij}$  for the training pattern s. The weights between the input nodes and the hidden nodes are updated after computing the gradient of

the total error with respect to the weights. The updating of the weights (between input nodes and hidden nodes) is shown below.

Using equations 3.2.6g and 3.2.6y

$$w_{ij}(t+1) = w_{ij}(t) - \eta \frac{\partial E_T}{\partial w_{ij}} \quad 3.2.6z$$

After updating all the weights, the NN is tested for error (sum mean square error). If the error is acceptable, then the training is completed. Otherwise the above training process is repeated. The calculation of error is defined by Equation 3.2.6f.

The recognition process is simple as in other neural networks. The NN computed the output of the output layer nodes using the equations 3.2.6a and 3.2.6d. . If only one output is close (threshold value has to be set by user) to one and the output of other nodes are close to zero, then the class of the node with output close to one is selected as the class of the unknown pattern. Otherwise the unknown pattern is rejected.

### 3.3 Investigation using each NN Model

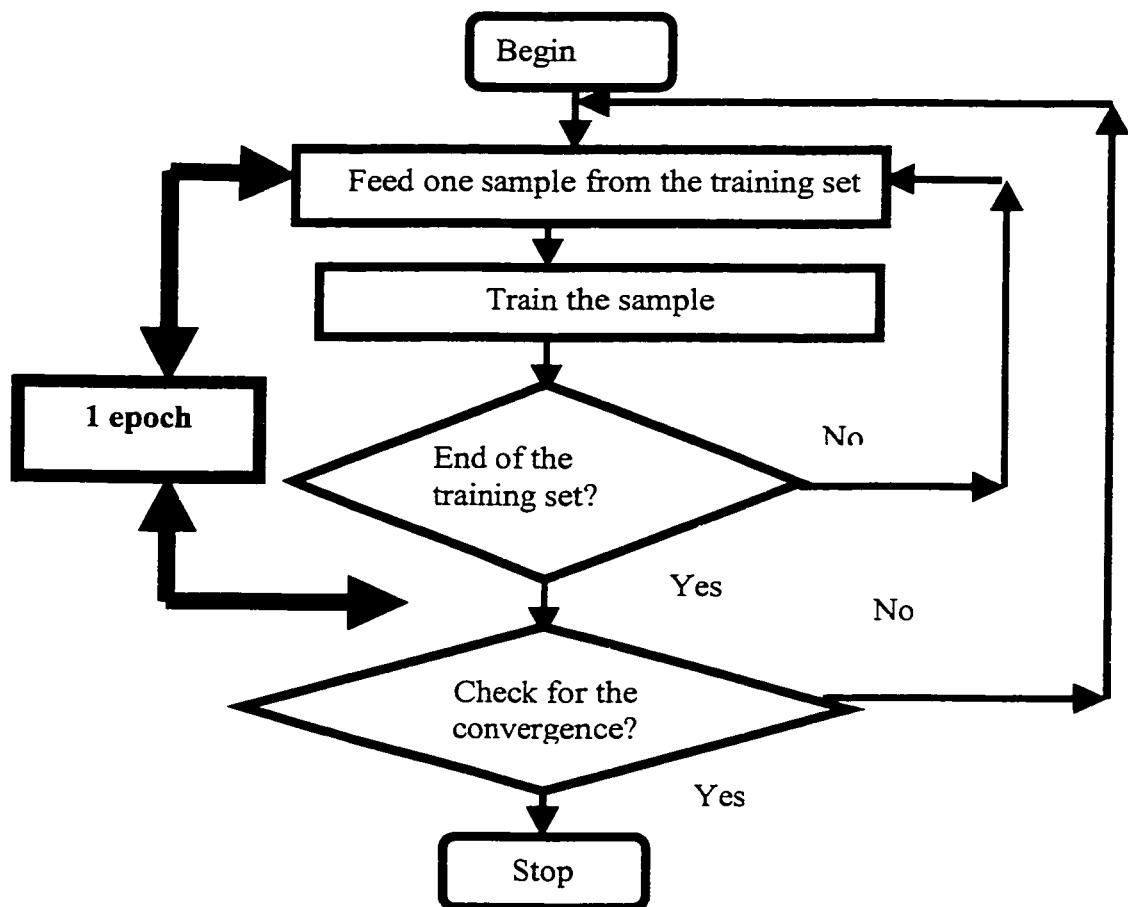
All the neural networks are investigated for the following:

- i) Training speed of the network
- ii) Number of patterns that can be trained
- iii) Size of the neural network
- iv) Speed of the trained network for test data
- v) Noise sensitivity test
- vi) Image translation test

The following sections explain how to achieve the investigation task of each of the above areas.

### 3.3.1 Training Speed of the Network

The training speeds of each of the neural networks is measured. The measuring unit is the epoch. One epoch is a complete cycle in which all the training patterns are used to train the NN once.



Flow chart 3.3.1. Scale of the one epoch

Flow chart 3.3.1 shows the definition of one epoch. All NNs training speed is measured in epochs.

### **3.3.2 Number of Patterns that can be Trained**

In this experiment, each of the neural networks is tested for a number of different patterns that can be trained. First of all, the training patterns are subjected to the training process of NN and then the NN is tested using the same training pattern. If all the patterns are recognized, the program exits. If all the patterns are not recognized, then another phase of training is started. In the second phase of training, all the possible training sets are arranged and then the training process as well as testing process are repeated. For example, let there be only four training patterns in the training set and they are a, b, c and d. First, all the patterns (abcd) are used to train the NN. Assume that only a and d are trained while b and c are not trained. Since all the patterns were not recognized, a second training phase is selected. The training sets selected for the second training phase are abc, abd, acd, bcd, ab, ac, ad, bc, bd, cd, a, b, c and d. These training sets are trained and tested. The maximum number of patterns trained with these training sets is recorded.

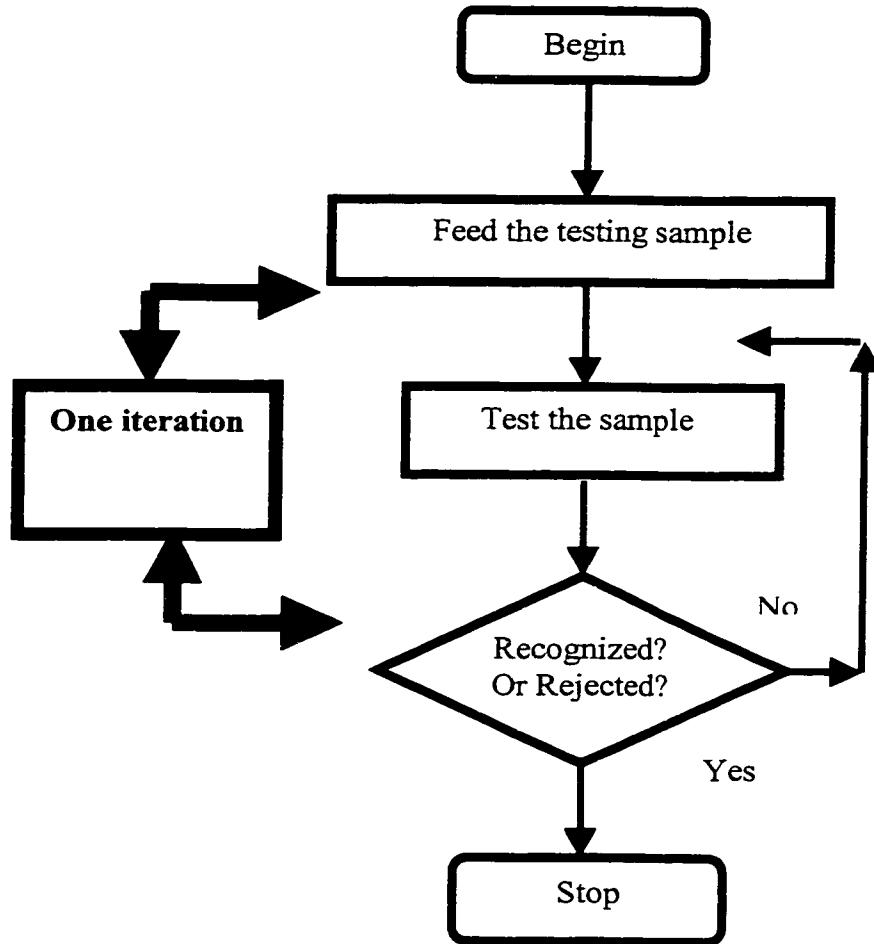
### **3.3.3 Size of the Neural Network**

The neural network size depends on the number of weight connections, and the number of nodes in the hidden layers and the output layer.

$$\begin{aligned} \text{Size of the NN} = & \text{Number of weight connections} + \text{Number of nodes in the hidden layers} \\ & + \text{Number of nodes in the output layer} \end{aligned}$$

The size is directly proportional to the storage space needed and inversely proportional to the speed of the testing and training process.

### 3.3.4 Speed of the Trained Network for Test Data



Flow chart 3.3.4 Testing speed

All the NNs are tested to determine the speed of the trained network for the given test data. The measuring unit is the iteration. With the testing data, if a NN goes through one cycle of testing process, it is called an iteration. The flow chart 3.3.4 shows the definition of one iteration. Some of the neural networks need more than one iteration to decide the result (which recognizes the sample or rejects the sample).

### **3.3.5 Noise Sensitivity Test**

The noise sensitivity test is the test that is done on the noise images to determine the noise sensitivity due to the addition of black pixels or removal of black pixels or addition and removal of black pixels. Examples of images are shown below.

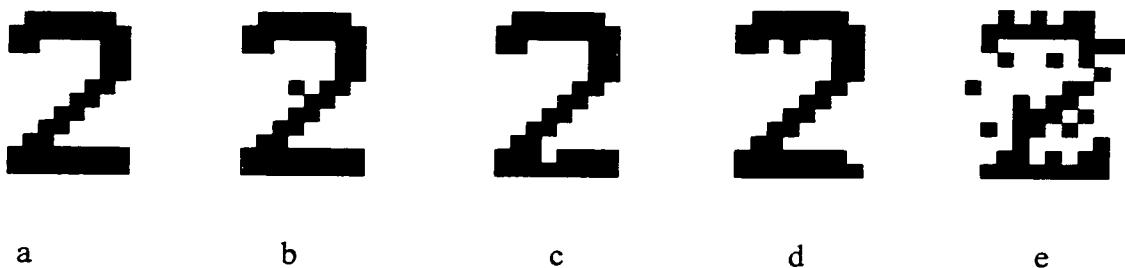


Figure 3.3.5 a) Original image, b) Removing one pixel, c) Adding one pixel and d) Removing and adding one pixel, e) Removing and Adding more than one pixel.

The figure 3.3.5 shows the noise images of the original image, removing black pixels from the image, adding black pixels to the image, adding and removing black pixels to the image.

### **3.3.5 Image Translation Test**

Image translation is the displacement of an image to another location within the same window. A trained image is displaced within the same window and tested for the image translation test.

## 3.4 Experimentation Results

The experiments are described in section 3.3. All the neural networks use the same testing set. The results of each of the neural networks are listed below.

### 3.4.1 The Hamming Neural Network

Using all the training patterns, the NN is trained without any problem. The training process took one epoch to train the Hamming NN. The size of the network is 1310 processing units. The size of the network is calculated as given below

$$\begin{aligned}\text{Size} &= \text{Total weights} + \text{total nodes in the hidden layers and the output layer} \\ &= (120 \text{ (input nodes)} * 10 \text{ (output nodes)} + 10 * 10 \text{ (output feed back} \\ &\quad \text{weights)}) + 10 \text{ (output nodes, and not hidden nodes)} \\ &= (1200 + 100) + 10 = 1310\end{aligned}$$

The speed of the trained network for test data varies with different numbers. It varies from 5 to 14 iterations. All the noise test images are well recognized (except the 7 and 9 images) by the NN. The translation image is recognized well compare to other NNs. Still the result is poorly low in the translated image test. All the results are summarized in Tables 3.4.1a and 3.41b.

Test \ Neural Net	Hamming
Training speed of the network	1
Number of patterns that can be trained	All
Size of the neural network	1310
Speed of the trained network for test data	5-14

Table 3.4.1a. Test results - The Speed and size of the NN.

Test\Image	0	1	2	3	4	5	6	7	8	9
Without Noise	R	R	R	R	R	R	R	R	R	R
Add one pixel	R	R	R	R	R	R	R	R	R	R
Remove one pixel	R	R	R	R	R	R	R	R	R	R
Add and Remove one pixel	R	R	R	R	R	R	R	R	R	R
Add and Remove more than one pixel	R	R	R	R	R	R	NR	R	NR	NR
Translated Image	R	NR	R	NR	-	R	R	NR	NR	NR

R – Recognized

NR – Not Recognized

Table 3.4.1b. Test results – Noise and translated images

### 3.4.2 The Carpenter \ Grossberg Neural Network

This neural network is trained with the entire training pattern. The training process took one epoch to train the NN. The size of the Carpenter \ Grossberg neural network is 2530 processing units. The size of the network is calculated as shown below

$$\begin{aligned}
 \text{Size} &= \text{weights} + \text{nodes} \\
 &= (120 * 10 + 10 * 120) + (10+120) \\
 &= 2530
 \end{aligned}$$

It takes one iteration to recognize or reject the testing data. Therefore, the speed of the trained network for test data is one iteration. All the noise test images are correctly recognized (except the 6, 7 and 9 images) by the NN. In the translated image test, only the translation image of 5 is recognized. All the results are summarized in Tables 3.4.2a and 3.4.2b.

Test \ Neural Net	Carpenter \ Grossberg
Training speed of the network	1
Number of patterns that can be trained	All
Size of the neural network	2530
Speed of the trained network for test data	1

Table 3.4.2a. Test results - The Speed and size of the NN.

Test\Image	0	1	2	3	4	5	6	7	8	9
Without Noise	R	R	R	R	R	R	R	R	R	R
Add one pixel	R	R	R	R	R	R	R	R	R	R
Remove one pixel	R	R	R	R	R	R	R	R	R	R
Add and Remove one pixel	R	R	R	R	R	R	R	R	R	R
Add and Remove more than one pixel	R	R	R	R	R	R	NR	NR	R	NR
Translated Image	NR	NR	NR	NR	-	R	NR	NR	NR	NR

R – Recognized

NR – Not Recognized

Table 3.4.2b. Test results - Test results – Noise and translated images

### 3.4.3 The Kohonen's Neural Network

The Kohonen's NN is trained using the entire training set. It took 1-3 epochs to train this NN. The training speed varies depending on the initial value of the weights at the beginning of the training process. The size of the neural network is 2420 processing units. The calculation of this neural network's size is given below

$$\text{Size} = \text{weights} + \text{nodes}$$

$$= (120 * 20) + (20)$$

$$= 2420$$

It always takes one iteration to recognize a number. All the noise images are recognized.

But the NN has failed the translation test (except the translated images 5 and 6). A summary of the results is shown in Tables 3.4.3a and 3.4.3b.

Test \ Neural Net	Kohonen's
Training speed of the network	1-3
Number of patterns that can be trained	All
Size of the neural network	2420
Speed of the trained network for test data	1

Table 3.4.3a. Test results - The Speed and size of the NN.

Test\Image	0	1	2	3	4	5	6	7	8	9
Without Noise	R	R	R	R	R	R	R	R	R	R
Add one pixel	R	R	R	R	R	R	R	R	R	R
Remove one pixel	R	R	R	R	R	R	R	R	R	R
Add and Remove one pixel	R	R	R	R	R	R	R	R	R	R
Add and Remove more than one pixel	R	R	R	R	R	R	R	R	R	R
Translated Image	NR	NR	NR	NR	-	R	R	NR	NR	NR

R – Recognized

NR – Not Recognized

Table 3.4.3b. Test results - Test results – Noise and translated images

#### 3.4.4 The Hopfield Neural Network

This NN cannot be trained using all the training patterns. The NN was able to be trained using only a maximum number of six patterns. The training speed of the network is always one epoch. The size of the NN is 14520 processing units. The calculation of the size of the network is given below

$$\begin{aligned}
 \text{Size} &= \text{Weights} + \text{Nodes} \\
 &= (120*120) + 120 \\
 &= 14520
 \end{aligned}$$

The speed of the trained network for the test data is two iterations. It has failed to recognize the translation image. The images with noise are recognized without any problem. The results are summarized in Tables 3.4.4a and 3.4.4b.

Test \ Neural Net	Hopfield
Training speed of the network	1
Number of patterns that can be trained	6
Size of the neural network	14520
Speed of the trained network for test data	2

Table 3.4.4a. Test results - The Speed and size of the NN.

Test\Image	0	1	3	4	6	9
Without Noise	R	R	R	R	R	R
Add one pixel	R	R	R	R	R	R
Remove one pixel	R	R	R	R	R	R
Add and Remove one pixel	R	R	R	R	R	R
Add and Remove more than one pixel	R	R	R	R	R	R
Translated Image	NR	NR	NR	-	NR	NR

R – Recognized

NR – Not Recognized

Table 3.4.4b. Test results - Test results – Noise and translated images

### 3.4.5 The Single Layer Perceptron Neural Network

The NN is trained with all training patterns. The weights of neural network are initialized with small random numbers, and therefore the training speed will also vary. The training speed for this application is 2 to 3 epochs. The size of the NN is 1210 processing units.

The calculation of the NN size is given below

$$\begin{aligned}\text{Size} &= \text{Weights} + \text{Nodes} \\ &= (120 * 10) + 10 \\ &= 1210\end{aligned}$$

The speed of the training NN for the test data is found to be one iteration. It failed in the image translation test. The NN performs poorly using the noise images. The summary of the results is shown in Tables 3.4.5a and 3.4.5b.

Test \ Neural Net	Single Layer Perceptron
Training speed of the network	2-3
Number of patterns that can be trained	All
Size of the neural network	1210
Speed of the trained network for test data	1

Table 3.4.5a. Test results - The Speed and size of the NN.

Test\Image	0	1	2	3	4	5	6	7	8	9
Without Noise	R	R	R	R	R	R	R	R	R	R
Add one pixel	R	R	R	R	R	R	R	R	R	R
Remove one pixel	R	R	R	R	R	R	R	R	R	R
Add and Remove one pixel	R	R	NR	R	R	R	R	R	R	R
Add and Remove more than one pixel	R	NR	NR	R	NR	R	R	NR	R	R
Translated Image	NR	NR	NR	NR	-	NR	NR	NR	NR	NR

R – Recognized

NR – Not Recognized

Table 3.4.5b. Test results - Test results – Noise and translated images

### 3.4.6 The Multi Layer Perceptron Neural Network

The Multi layer neural network is trained using all the training patterns. The training speed of the NN depends on the values of the weights at the beginning of the training process. The weights are set with small random values at the beginning of the training process. At one time, the training speed of the NN was 85 epochs and at another time the training speed was 176 epochs. The size of the network is 1975 processing units. The calculation of the network's size is given below

$$\begin{aligned}
 \text{Size} &= \text{Weights} + \text{Nodes} \\
 &= ((120 * 15) + (15 * 10)) + (15 + 10) \\
 &= 1975
 \end{aligned}$$

Always the speed of the trained network for the test data is one iteration. Like other NNs, it also fails in the image translation test (except for the translated image 6). In the noise sensitivity test, it recognized all the noise images. The results are summarized in Tables 3.4.6a and 3.4.6b.

Test \ Neural Net	Multi Layer Perceptron
Training speed of the network	85,176
Number of patterns that can be trained	All
Size of the neural network	1975
Speed of the trained network for test data	1

Table 3.4.6a. Test results - The Speed and size of the NN.

TestImage	0	1	2	3	4	5	6	7	8	9
Without Noise	R	R	R	R	R	R	R	R	R	R
Add one pixel	R	R	R	R	R	R	R	R	R	R
Remove one pixel	R	R	R	R	R	R	R	R	R	R
Add and Remove one pixel	R	R	R	R	R	R	R	R	R	R
Add and Remove more than one pixel	R	R	R	R	R	R	R	R	R	R
Translated Image	NR	NR	NR	NR	-	NR	R	NR	NR	NR

R – Recognized

NR – Not Recognized

Table 3.4.6b. Test results - Test results – Noise and translated images

### **3.5 Discussions**

Kohonen's NN and Multi layer NN responded very well in the noise sensitivity test. Except for the Hopfield NN, all the other neural networks can be trained using the entire pattern for this application. The size of these neural networks varies from 1210 to 2530 processing units with the exception of the Hopfield NN, the size of which is 14520 processing units. The speed of the trained networks with the test data was one iteration for the Multi Layer Perceptron, Single Layer Perceptron, Kohonen's NN and Carpenter \ Grossberg NN. In the case of the Hopfield and the Hamming neural networks, more than one iteration was needed to recognize a number. The training speed of the Hopfield and the Hamming NNs was always one epoch. The training speed of Carpenter/ Grossberg, Kohonen's and Single Layer Perceptron NNs varied by a small number of epochs (less than 5 epochs and it depends on the initial weights for Kohonen's NN and Single Layer Perceptron). The Multi Layer Perceptron NN had the slowest training speed compared to all other NNs.

All the NNs failed in the image translation test, which implies that these images should be added to the training set. Therefore, all the translated images are added to the training set data and then all the NNs (except Hopfield NN) are retrained. These retrained neural networks recognized the entire training data set. The Carpenter \ Grossberg, the Hamming and the Kohonen's neural networks contain stored information about the translated images as new patterns. Hence, new connection weights and nodes are added

to these neural networks in order to learn the translated images. Therefore, the size of these neural networks increases as the number of patterns is added to the training set. In the case of translated images for the Single Layer Perceptron and Multi Layer Perceptron only the connection weights are adopted. Hence, there is no additional node or connection is added. Therefore the size of the neural network does not change. A solution for the translated image problem is to apply moment invariants feature extraction algorithm to the translated images before feeding to the neural networks.

# **Chapter IV**

## **4.1 Fast Multi Layer Perceptron (MLP) Training Algorithm**

The Multi Layer Perceptron trained with the delta rule algorithm has successfully been used in many experimental works [SEJ87] [BUR88]. The Delta rule algorithm iteratively adjusts the weights using the gradient of the error surface so that the differences between the outputs of the MLP and the desired outputs are minimized. However, the convergence speed is inherently slow. For large number of training data, it may take days to train the NN. Therefore, a different method must be considered for a large numbers of training set data. In this chapter, a fast training algorithm is described and then it is compared with the Delta rule algorithm.

## 4.2 Fast Training Algorithm

A one hidden layer MLP is shown in Figure 4.2. The input and output values of all the nodes are not known. Only the input values of the input layer's nodes ( $x_i$ ) and the desired output values of the output layer's nodes ( $d_k$ ) are known. If the input value of each of the nodes and the summation output of each of the nodes are known, then the problem reduces to a system of linear equations that relates the weights to the summation output and the input of that node [ROB92].

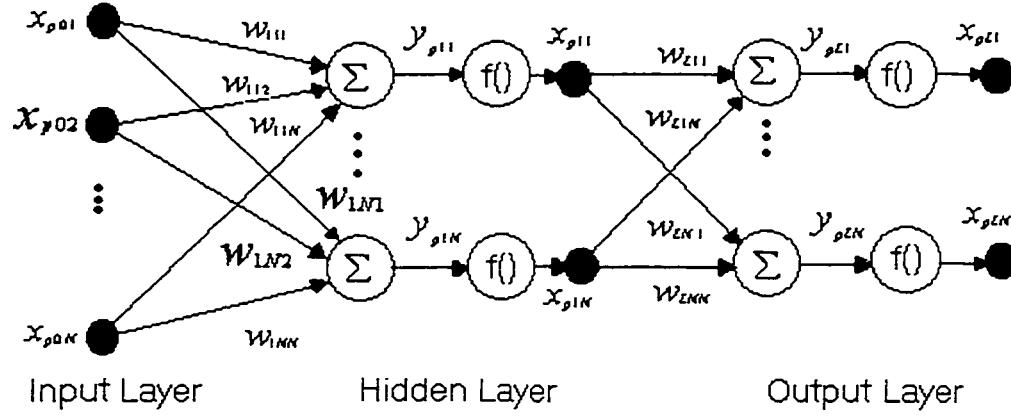


Figure 4.2. The Multi Layer Perceptron Neural Network connections and nodes in each layer

It is possible to estimate the desired summation output of each of the nodes. The equation for the specific node is as follows

$$d_{pjk} = y_{pjk} + \mu e_{pjk} \quad 4.2a$$

Where  $d_{pj}$  is the estimated desired summation output,  $y_{pj}$  is the calculated summation output,  $\mu$  is the step size, and  $e_{pj}$  is the calculated error signal for the node  $k$  of layer  $j$  and the training pattern  $p$ .  $\mu e_{pj}$  is the estimated error signal for the node  $k$ . The  $e_{pj}$  for the output and hidden nodes are derived in chapter 3 (see eq.3.26n and eq.3.26x). Now, all the estimated input and output values are known. Using these estimates in the system of linear equations, it is possible to estimate the weights (not exact weights). Here, the problem is to minimize the error, with respect to the summation outputs  $y_p$ . The total mean square error  $E$  is given by

$$E = \frac{1}{2} \sum_{p=1}^M (d_p - y_p)^2 \quad 4.2b$$

Where  $d_p$  is the desired summation outputs,  $y_p$  is the calculated summation output for the pattern  $p$  and  $M$  is the number of training patterns. To minimize this error, partial derivatives of  $E$  with respect to each weight ( $w_n$ ) are taken in equation 4.2b and set equal to zero.

Therefore,

$$\frac{\partial E}{\partial w_n} = \frac{1}{2} * 2 \sum_{p=1}^M (d_p - y_p) (-1) \frac{\partial y_p}{\partial w_n} = 0 \text{ for minimum error}$$

Where  $y_p = \sum_{i=0}^N w_i x_{pi}$  and  $n$  is 0 through  $N$

$$\sum_{p=1}^M (d_p - \sum_{i=0}^N w_i x_{pi}) x_{pn} = 0$$

$$\sum_{p=1}^M d_p x_{pn} = \sum_{p=1}^M x_{pn} \sum_{i=0}^N w_i x_{pi}$$

By changing the right summation to a vector multiplication

$$\begin{aligned} \sum_{p=1}^M d_p x_p &= \sum_{p=1}^M x_p w^T x_p \\ &= \sum_{p=1}^M x_p x_p^T w \end{aligned} \quad 4.2c$$

By definition

$$R = \sum_{p=1}^M x_p x_p^T \quad 4.2d$$

$$\text{and } P = \sum_{p=1}^M d_p x_p \quad 4.2e$$

Using equations 4.2d and 4.2e in eq.4.2c

$$P = R w \quad 4.2f$$

To solve the weights, multiply eq.4.2f by  $R^{-1}$

$$w = R^{-1} P \quad 4.2g$$

To update the weights, all the training patterns have to be run through the network. But it is also possible to update the weights in each iteration by modifying equations 4.2d and 4.2e, and selecting the training patterns to the network randomly.

$$R(t) = \sum_{k=1}^t x(k) x^T(k) \quad 4.2h$$

$$P(t) = \sum_{k=1}^t d(k) x(k) \quad 4.2i$$

Eq.4.2g becomes

$$w(t) = R^{-1}(t)p(t) \quad 4.2j$$

In the above equations, t is the number of the present iterations. At the beginning of the training,  $R(t)$  and  $p(t)$  are estimated from the previous layer data. This will create a problem because the previous layers are not trained at the beginning of the training process. The problem is solved by adding a factor (forgetting factor) that is of small value, at the beginning of training process; then the forgetting factor increases with iterations (the forgetting factor between 0 and 1). The eq.4.2h and eq.4.2i can be rewritten as

$$R(t) = \sum_{k=1}^t b^{t-k} x(k) x^T(k) \quad 4.2k$$

$$p(t) = \sum_{k=1}^t b^{t-k} d(k)x(k) \quad 4.2l$$

Where  $b$  is the forgetting factor, which increases with iterations. Now eq.4.2k and eq.4.2l can be rewritten into recursive form as follows

$$R(t) = bR(t-1) + x(t)x^T(t) \quad 4.2m$$

$$p(t) = bp(t-1) + d(t)x(t) \quad 4.2n$$

To update the weights, a recursive equation is needed for  $R^{-1}$ . This updating can be achieved by using the Kalman filter technique and adapting the treatment in [ROB92]

[PRO83] [HAY86] to the solution in eq.4.2j iteratively. The final solution is given below. For the  $j^{\text{th}}$  layer

The Kalman gain vector:

$$k_j(t) = \frac{R_j^{-1}(t-1) x_{j-1}(t)}{b_j + x_{j-1}^T(t) R_j^{-1}(t-1) x_{j-1}(t)} \quad 4.2o$$

To update the inverse matrix of R:

$$R_j^{-1}(t) = \frac{R_j^{-1}(t-1) - k_j(t) x_{j-1}^T(t) R_j^{-1}(t-1)}{b_j} \quad 4.2p$$

Numerical experience with the Kalman algorithm indicates that it is sensitive to the effects of computer round off and is susceptible to accuracy degradation due to the differencing of positive terms [OSA94]. Therefore, the treatments in [OSA94] and [GER77] to the Kalman algorithm are adopted for the above equations 4.2o and 4.2p.

We get

$$k_j = S_j \nu_j \quad 4.2q$$

Where  $\nu_j = (x_j S_j)^T$

Where  $k_j$  is the unweighted kalman gain,  $S_j$  is the square root of the inverse of the covariance matrix  $R$  and  $x_j$  is the input to the node  $j$ . Updating the inverse matrix  $S_j$  follows:

$$S_{j+1} = \frac{S_j - (\gamma_j k_j) v_j^T}{b_j} \quad 4.2r$$

where  $\gamma_j = \frac{\sigma_j}{1 + \sqrt{\sigma_j}}$  and  $\sigma_j = \frac{1}{(v_j^T v_j + b_j)}$

Where  $b_j$  is the forgetting factor. Using eq.4.2q, the weights can be updated as follows

For the output layer:

$$w_{Lk}(t) = w_{Lk}(t-1) + k_L(t)(d_k - y_{Lk}) \quad 4.2s$$

For the hidden layers:

$$w_{jk}(t) = w_{jk}(t-1) + k_j(t)e_{jk}(t)\mu_j \quad 4.2t$$

Where  $\mu_j$  is the step size,  $t$  is the present interaction number and  $k$  is the node in the layer.

### 4.3 Comparison of MLP Training Algorithms

The Fast training algorithm and Delta rule training algorithms are compared on basis of training speed and generalization capability. The training speed is measured in epochs. The generalization capability is tested in recognition rate using untrained data (testing data). In the Delta rule, the learning rate constant is selected by experimentation. In this experiment, six different learning rates are selected: 0.01, 0.1, 0.2, 0.5, 1 and 10. The NN is trained starting with same initial weights for each selected learning rate constant using the training data. Figure 4.3a shows the training speeds of the different learning rate values. The learning rate value 0.2 performed well on the training data compared to other learning rate values. Therefore the learning rate value 0.2 is used in the Delta rule training method for further experimentation.

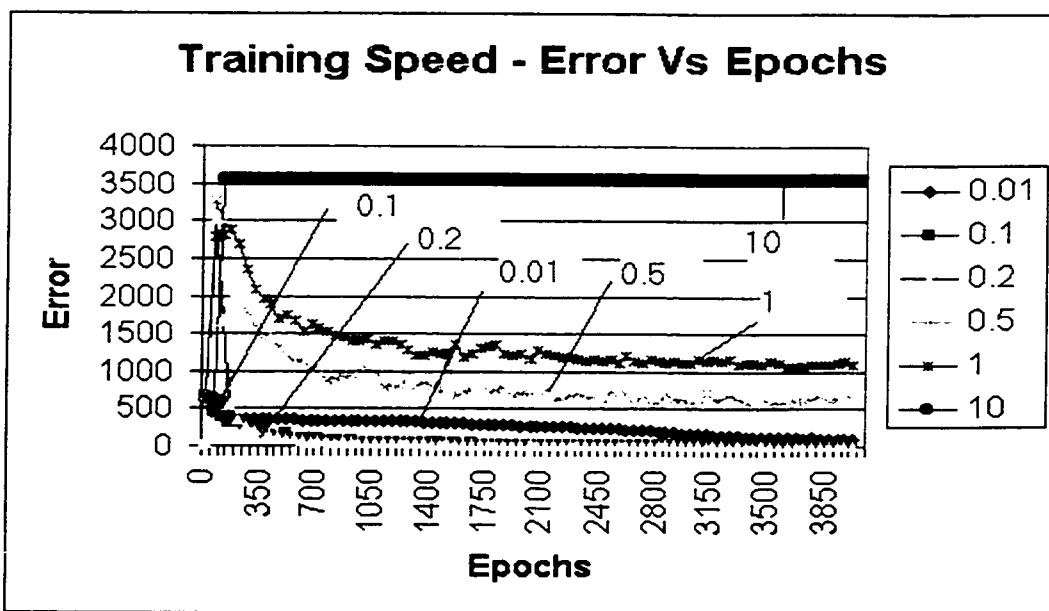


Figure 4.3a Training speed of the Delta rule-training algorithm for various learning rates

To compare the algorithms, the training process is repeated five times. In each trial, the initial weights do not have the same value, but both the algorithms use the same initial weights in each trial. To measure the training speed and generalization capability, both algorithms are started at the same point on the error surface. Five trials were considered to obtain a better result for training speed and generalization capability tests. The results are shown in Figure 4.3b to Figure 4.3k.

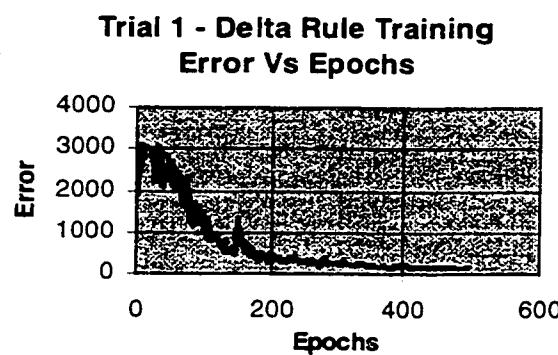


Figure 4.3b. Trial 1 – Performance of the Delta rule training method

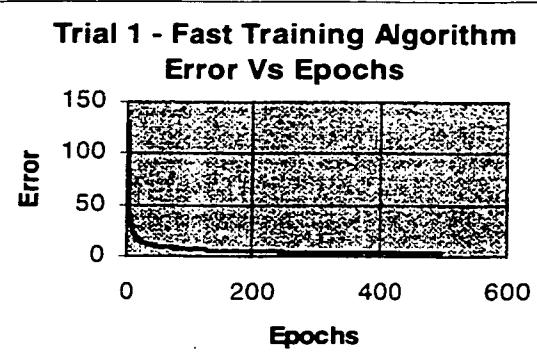


Figure 4.3c. Trial 1 – Performance of the Fast training method

The Trial 1 result of the Delta rule is shown in Figure 4.3b and the result of the Fast training algorithm is shown in Figure 4.3c (other trials are shown in a similar way). The result of both the algorithms cannot be plotted on the same graph because their error range differs by a wide scale. It is easy to notice in all the trials that the fast training algorithms are much faster than the standard Delta rule algorithm. Table 4.3a shows the numerical value of total mean square error on each epoch for both algorithms and all the five trials. For example, on the 5<sup>th</sup> epoch the total mean square error for the Fast training algorithm is less than 60 in each trial. But the total mean square error for the delta rule algorithm is more than 1000 on the 5<sup>th</sup> epoch in each trial.

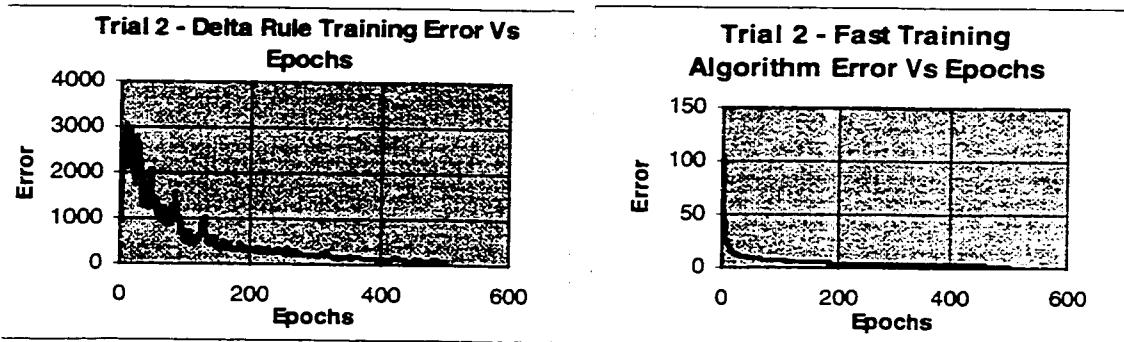


Figure 4.3d. Trial 2 – Performance of the Delta rule training method

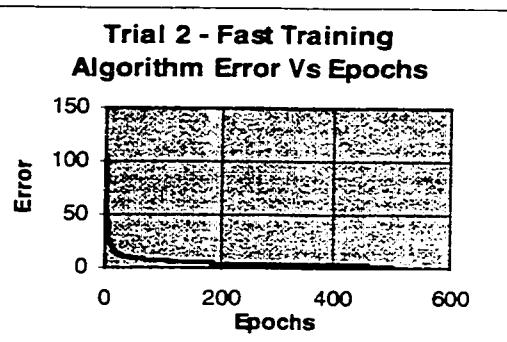


Figure 4.3e. Trial 2 – Performance of the Fast training method

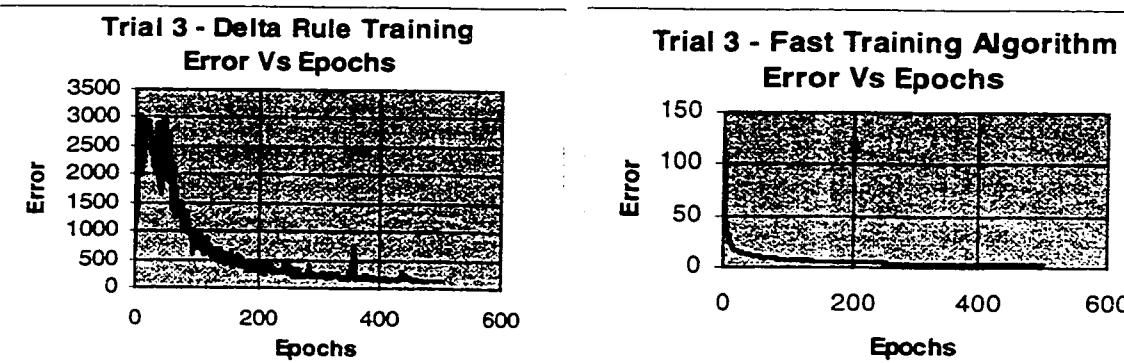


Figure 4.3f. Trial 3 – Performance of the Delta rule training method

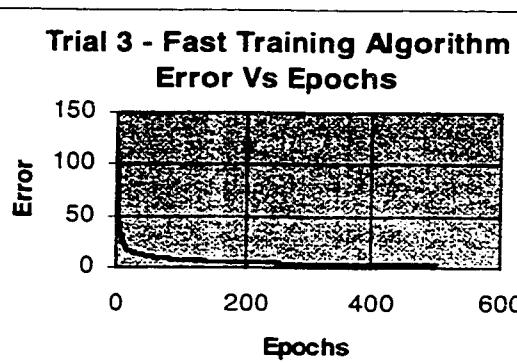


Figure 4.3g. Trial 3 – Performance of the Fast training method

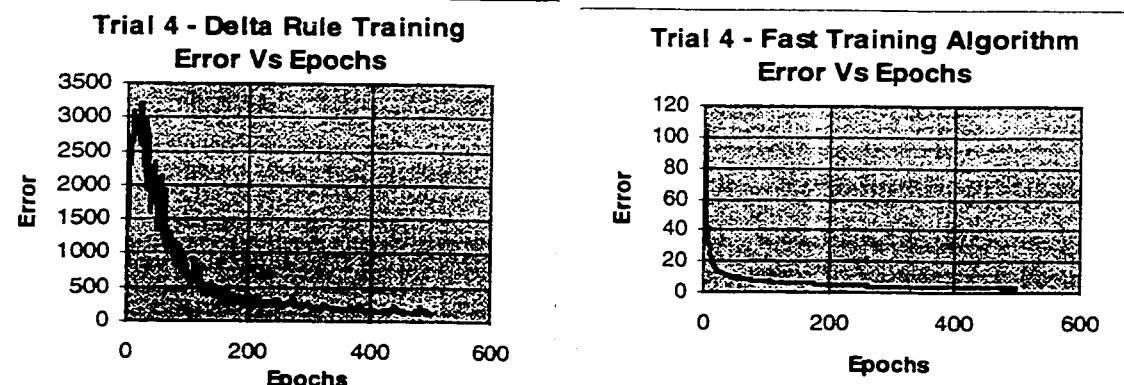


Figure 4.3h. Trial 4 – Performance of the Delta rule training method

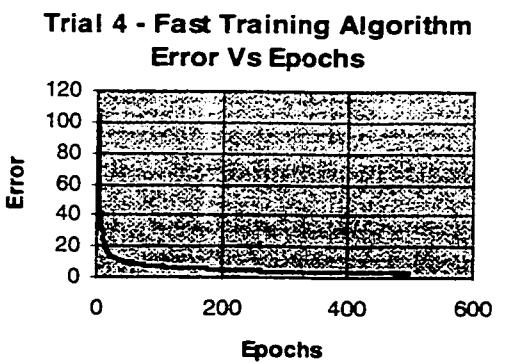


Figure 4.3i. Trial 4 – Performance of the Fast training method

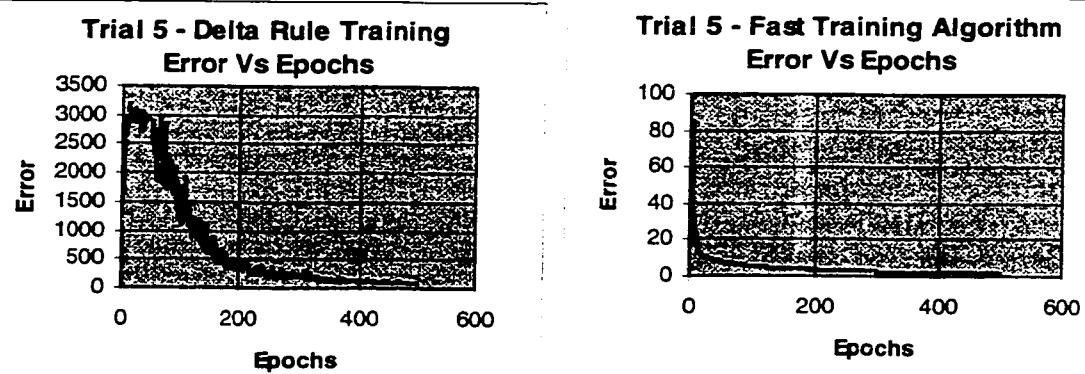


Figure 4.3j. Trial 5 – Performance of the Delta rule training method

Figure 4.3k. Trial 5 – Performance of the Fast training method

Epochs	Delta Trial 1	Fast Trial 1	Delta Trial 2	Fast Trial 2	Delta Trial 3	Fast Trial 3	Delta Trial 4	Fast Trial 4	Delta Trial 5	Fast Trial 5
1	891.56	702.73	683.40	810.49	827.95	848.88	1403.3	752.33	1019.0	709.45
5	2779.4	48.62	2732.5	53.48	2113.2	48.51	1835.6	51.8	2847.4	33.56
10	3097.73	28.76	3069.58	25.8	1999.8	30.57	2900.2	26.832	3179.5	19.67
50	2230.90	10.23	1612.93	9.65	1915.2	11.56	1319.9	10.1	2807.7	7.96
100	1100.02	7.41	490.99	7.03	797.37	7.78	709.28	7.31	1644.2	5.38
200	355.47	5.36	317.73	4.31	327.77	5.47	318.91	5.21	343.06	3.57
300	268.18	4.18	194.71	3.41	222.78	4.36	284.21	4.01	198.16	2.68
400	211.21	3.58	111.9	2.98	161.15	3.71	176.10	3.46	123.21	2.25
500	151.53	3.146	88.68	2.55	126.63	3.24	130.57	3.09	100.38	1.91
600	130.07	2.83	92.62	2.36	112.48	2.83	116.01	2.86	72.54	1.71

Table 4.3a. The total mean square error in each trial for both algorithms

After being trained with both algorithms, the NN is tested with the new data (called testing set data), which was not in the training set. A total of 130 characters are tested (5 for each English capital letter,  $5 * 26 = 130$ ). The results of this test are shown in Table 4.3b.

Trial	Delta Rule Training	Fast Training Algorithm
1	(118/130) 90.8%	(127/130) 97.7%
2	(123/130) 94.6%	(128/130) 98.5%
3	(121/130) 93.1%	(127/130) 97.7%
4	(128/130) 98.5%	(129/130) 99.2%
5	(121/130) 93.1%	(129/130) 99.2%

Table 4.3b. Testing result of both algorithms

From Table 4.3b, the recognition result of the NN, which was trained by the Fast training algorithm, is 97.7% or more than 97.7%. But in the same table, the recognition result of the NN, which was trained by the Delta rule algorithm, is between 90% and 95% except in the 4<sup>th</sup> trial, where it is 98.5%. Thus, the results show that the NN trained by the Fast training algorithm has better generalization ability than the NN trained by the Delta rule algorithm. However, sometimes the NN trained by the Delta rule algorithm performs as well with unknown data set as the NN trained by the Fast training algorithm.

# **Chapter V**

## **5.1 Conclusion**

In this thesis, two objectives in neural networks are discussed based on an OCR application. The first one, which is discussed in chapter III, is the experimentation of various Neural Network models in an OCR application (all ten Arabic numerals). The second objective of this thesis is to apply a Fast training algorithm to train the Multi Layer Perceptron (MLP) neural network, in which the applied algorithm is faster than the standard Delta training rule algorithm. It is discussed in chapter IV.

The Hopfield Neural Network is limited in the number of patterns it can remember and the size of the NN is very large. . It is observed that the Arabic numeric recognition was better using the Kohonen's and Multi layer neural networks in noise-sensitivity test. Performance of the Single Layer Perceptron neural network is observed to be poor for the noise sensitivity test. The Hamming Net, Kohonen's NN and Carpenter \ Grossberg NN perform well in training speed and testing speed, however, the size of the NNs will increase by the addition of more samples to the training set. For example, consider a sample not recognized by the NN; then this pattern should be added to the training set so

that it will be recognized next time. The NN trained using this sample as a new pattern causes an increase in the size of the NN. The MLP NN has a better performance compared to the other neural networks. The training of the MLP NN is a very long process compared to the training of the other NNs.

The Fast training algorithm and the Delta rule algorithm are compared for training speed and generalization ability. The standard training algorithm for the MLP NN is the Delta rule algorithm. The training speed of the Fast training algorithm is much faster than the Delta rule algorithm. The generalization ability of the Fast training algorithm is always better than standard training algorithm. The recognition rate achieved more than 97.5% in Times New Roman font with a font size of 16.

## 5.2 Future Direction

Some recommendations for future researches are given below:

- i) The NNs should be experimented on regarding the extracting features from images.
- ii) A combined method of fuzzy logic and neural network should be considered for OCR applications.
- iii) H-Net and Quantum NN should be investigated in OCR applications.

## References

- [ANI00] Anil K. Jain, Robert P.W. Duin, Jianchang Mao, “Statistical Pattern Recognition: A Review”, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 22, No. 1, P4-37, 2000.
- [BUR88] D.J Burr, “Experiments on neural net recognition of spoken and written text”, IEEE Trans. Acoust., Speech, Signal Processing, Vol.30, no.7, p. 1162 – 1168, 1988.
- [GER77] Gerald J. Bierman,”Factorization Methods for Discrete Sequential Estimation”, Academic Press, 1977.
- [HAY86] S. S. Haykin, Adaptive Filter Theory, Englewood Cliffs, NJ, Prentice-Hall, 1986.
- [JOH91] John Hertz, Anders Krogh, Richard G. Palmer, “ Introduction to The Theory of Neural Computation”, Addison-Wesley Publishing Company, 1991.
- [KAR96] Stamatios V. Kartalopoulos,”Understanding Neural Networks and Fuzzy Logic”, IEEE Press, 1996.
- [KAU98] Kauniskangas H & Sauvola J (1998) An automated defect management for document images. Proc. 14<sup>th</sup> International Conference on Pattern Recognition, August 17-20, Brisbane, Australia, 1288-1294.
- [KHA88] F. E. Khaly, “Recognition of Printed Arabic Characters”, M.A.Sc. Thesis, University of Windsor, Canada, 1988.

- [KOH84] T. Kohonen, Self-Organization and Associative Memory, Springer – Verlag, Berlin, 1984.
- [LIP87] R. P. Lippmann, "An introduction to computing with neural networks", IEEE ASSP Mag. Vol.4, No.2, 1987.
- [MOR99] Shunji Mori, Hirobumi Nishida, Hiromitsu Yamada, Optical Character recognition, John Wiley & Sons, 1999.
- [MOR92] Mori, Shunji , Suen, Ching Y. and Yamamoto, Kazuhiko. "Historical Review of OCR: Research and Development." Document Image Analysis. 1992. p. 244-269.
- [MUL95] B. Muller, J. Reinhardt, and M.T. Strickland, " Neural Networks", Springer-Verlag Berlin Heidelberg, 1995.
- [OKU99] Okun O, Pietikäinen M & Sauvola J (1999) Document skew estimation without angle range restriction. International Journal on Document Analysis and Recognition 2(2/3), 132-144.
- [OSA94] Osama Abdel-Wahha Ahmed,"Application of artificial neural networks to optical characters recognition". MSc. Thesis, King Fahd University of Petroleum & Minerals, Saudi Arabia, 1994.
- [PRO83] J. G. Proakis, Digital Communications, New York, McGraw-Hill, 1983.
- [ROB92] R. S. Scalero, and N. Tepedelenlioglu, "A Fast New Algorithm for Training Feedforward Neural Networks", IEEE Transactions on Signal Processing, Vol. 40, No.1, 1992.

- [SAU00] Sauvola J & Pietikäinen M (2000) Adaptive document image binarization. *Pattern Recognition*, 33(2), 225-236.
- [SAU95] Sauvola, J & Pietikäinen, M (1995) Page segmentation and classification using fast feature extraction and connectivity analysis. Proc. Third International Conference on Document Analysis and Recognition (ICDAR'95), August 14-18, Montreal, Canada, pp. 1127-1131.
- [SAU97] Sauvola J, Pietikäinen M & Koivusaari M (1997) Predictive coding for document layout characterization. Proc. Workshop on Document Image Analysis (DIA'97), June 20, San Juan, Puerto Rico, 44-50.
- [SEJ87] T. J. Sejnowski and C. R. Rosenberg, "Parallel Networks that learn to pronounce English text", *Complex Systems.*, Vol.1, p 145-168, 1987.
- [SID94] M. A. Sid-Ahmed, "Image Processing: Theory, Algorithms and Applications," McGraw-Hill, New York, 1994.
- [SRI92] S.N. Srihari, High-Performance Reading Machines, *Proceedings of the IEEE*, 80(7), July 1992, 1120-1132.
- [TAG94] K. Taghva, J. Borsack, and A. Condit. Expert system for automatically correcting OCR output. In *Proceedings of the SPIE - Document Recognition*, pages 270--278, 1994.

## Appendix

**Appendix A:**  
**Source Code for the six NN models**

## //Main Program

```
// stdafx.h : include file for standard system include files.
// or project specific include files that are used frequently, but
//   are changed infrequently
//

#ifndef AFX_STDAFX_H__D599BF28_8D75_11D4_89AC_00E0290F3618__INCLUDED_
#define AFX_STDAFX_H__D599BF28_8D75_11D4_89AC_00E0290F3618__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#define VC_EXTRALEAN           // Exclude rarely-used stuff from Windows headers

#include <afxwin.h>          // MFC core and standard components
#include <afxext.h>           // MFC extensions
#include <afxdisp.h>          // MFC OLE automation classes
#include <math.h>
#include <time.h>
#include <stdlib.h>
#ifndef _AFX_NO_AFXCMN_SUPPORT
#include <afxcmn.h>           // MFC support for Windows Common Controls
#endif // _AFX_NO_AFXCMN_SUPPORT

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_STDAFX_H__D599BF28_8D75_11D4_89AC_00E0290F3618__INCLUDED_)
```

## // stdafx.cpp :

```
//source file that includes just the standard includes
//      Neural.pch will be the pre-compiled header
//      stdafx.obj will contain the pre-compiled type information

#include "stdafx.h"
```

```

//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.

Used by Neural.rc

// 
#define IDD_ABOUTBOX      100
#define IDR_MAINFRAME     128
#define IDR_NEURALTYPE    129
#define IDD_DIALOG1        132
#define IDD_DIALOG2        133
#define IDD_DIALOG3        134
#define IDD_DIALOG4        135
#define IDD_DIALOG5        136
#define IDD_DIALOG6        137
#define IDD_DIALOG7        138
#define IDD_DIALOG8        139
#define IDC_EDITFileName   1005
#define IDC_EDITWidth       1006
#define IDC_EDITHeight      1007
#define IDC_EDITId          1008
#define IDC_EDITReadImageText 1009
#define IDC_EDITWFname     1010
#define IDC_RADIOTrain      1011
#define IDC_RADIOTest       1012
#define IDC_HamTrain        1013
#define IDC_HamTest         1014
#define IDC_CarpGrossTrain  1015
#define IDC_CarpGrossTest   1016
#define IDC_SLPTraining     1017
#define IDC_SLPTesting      1018
#define IDC_KohTrain        1019
#define IDC_KohTest         1020
#define IDC_MLPTrain        1021
#define IDC_MLPTest         1022
#define ID_BUTTONSAVEIMAGE  32771
#define ID_BUTTONHOPFIELD   32772
#define ID_BUTTONHAMMING    32773
#define ID_BUTTONCARPENTER  32774
#define ID_BUTTONPERCEPTRON 32775
#define ID_BUTTONMLAYER     32776
#define ID_BUTTONKOHONEN    32777

// Next default values for new objects
// 
#ifndef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_3D_CONTROLS    1
#define _APS_NEXT_RESOURCE_VALUE 140
#define _APS_NEXT_COMMAND_VALUE 32778
#define _APS_NEXT_CONTROL_VALUE 1023
#define _APS_NEXT_SYMED_VALUE 101
#endif
#endif

```

```

// MainFrm.h : interface of the CMainFrame class
//
//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#ifndef !defined(AFX_MAINFRM_H__D599BF2A_8D75_11D4_89AC_00E0290F3618__INCLUDED_)
#define AFX_MAINFRM_H__D599BF2A_8D75_11D4_89AC_00E0290F3618__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

class CMainFrame : public CFrameWnd
{
protected: // create from serialization only
    CMainFrame();
    DECLARE_DYNCREATE(CMainFrame)

// Attributes
public:

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CMainFrame)
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    //}}AFX_VIRTUAL

// Implementation
public:
    virtual ~CMainFrame();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected: // control bar embedded members
    CStatusBar m_wndStatusBar;
    CToolBar m_wndToolBar;

// Generated message map functions
protected:
    //{{AFX_MSG(CMainFrame)
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
        // NOTE - the ClassWizard will add and remove member functions here.
        // DO NOT EDIT what you see in these blocks of generated code!
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_MAINFRM_H__D599BF2A_8D75_11D4_89AC_00E0290F3618__INCLUDED_)

```

**//End - MainFrm.h**

```

// MainFrm.cpp : implementation of the CMainFrame class
//

#include "stdafx.h"
#include "Neural.h"

#include "MainFrm.h"

#ifndef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

///////////
// CMainFrame

IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
    //{{AFX_MSG_MAP(CMainFrame)
        // NOTE - the ClassWizard will add and remove mapping macros here.
        // DO NOT EDIT what you see in these blocks of generated code !
    ON_WM_CREATE()
   //}}AFX_MSG_MAP
END_MESSAGE_MAP()

static UINT indicators[] =
{
    ID_SEPARATOR,           // status line indicator
    ID_INDICATOR_CAPS,
    ID_INDICATOR_NUM,
    ID_INDICATOR_SCRL,
};

///////////
// CMainFrame construction/destruction

CMainFrame::CMainFrame()
{
    // TODO: add member initialization code here
}

CMainFrame::~CMainFrame()
{
}

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
        return -1;

    if (!m_wndToolBar.Create(this) ||
        !m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
    {
        TRACE0("Failed to create toolbar\n");
        return -1; // fail to create
    }

    if (!m_wndStatusBar.Create(this) ||
        !m_wndStatusBar.SetIndicators(indicators,
            sizeof(indicators)/sizeof(UINT)))
    {
        TRACE0("Failed to create status bar\n");
        return -1; // fail to create
    }
}

```

```

// TODO: Remove this if you don't want tool tips or a resizable toolbar
m_wndToolBar.SetBarStyle(m_wndToolBar.GetBarStyle() |
    CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_DYNAMIC);

// TODO: Delete these three lines if you don't want the toolbar to
// be dockable
m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
EnableDocking(CBRS_ALIGN_ANY);
DockControlBar(&m_wndToolBar);

return 0;
}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Modify the Window class or styles here by modifying
    // the CREATESTRUCT cs

    return CFrameWnd::PreCreateWindow(cs);
}

/////////////////////////////////////////////////////////////////////////
// CMainFrame diagnostics

#ifndef _DEBUG
void CMainFrame::AssertValid() const
{
    CFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
    CFrameWnd::Dump(dc);
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////
//End - MainFrm.cpp

```

```

// Neural.h : main header file for the NEURAL application
//

#ifndef AFX_NEURAL_H__D599BF26_8D75_11D4_89AC_00E0290F3618__INCLUDED_
#define AFX_NEURAL_H__D599BF26_8D75_11D4_89AC_00E0290F3618__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#ifndef __AFXWIN_H__
    #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"      // main symbols

///////////////////////////////
// CNeuralApp:
// See Neural.cpp for the implementation of this class
//


class CNeuralApp : public CWinApp
{
public:
    CNeuralApp();

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CNeuralApp)
    public:
        virtual BOOL InitInstance();
    //}}AFX_VIRTUAL

// Implementation

    //{{AFX_MSG(CNeuralApp)
    afx_msg void OnAppAbout();
        // NOTE - the ClassWizard will add and remove member functions here.
        // DO NOT EDIT what you see in these blocks of generated code !
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

/////////////////////////////
//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_NEURAL_H__D599BF26_8D75_11D4_89AC_00E0290F3618__INCLUDED_)

//End - Neural.h

```

```

// Neural.cpp : Defines the class behaviors for the application.

//
#include "stdafx.h"
#include "Neural.h"

#include "MainFrm.h"
#include "NeuralDoc.h"
#include "NeuralView.h"

#ifndef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

///////////
// CNeuralApp

BEGIN_MESSAGE_MAP(CNeuralApp, CWinApp)
    //{{AFX_MSG_MAP(CNeuralApp)
    ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
        // NOTE - the ClassWizard will add and remove mapping macros here.
        // DO NOT EDIT what you see in these blocks of generated code!
    //}}AFX_MSG_MAP
    // Standard file based document commands
    ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
    ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
END_MESSAGE_MAP()

///////////
// CNeuralApp construction

CNeuralApp::CNeuralApp()
{
    // TODO: add construction code here,
    // Place all significant initialization in InitInstance
}

///////////
// The one and only CNeuralApp object

CNeuralApp theApp;

///////////
// CNeuralApp initialization

BOOL CNeuralApp::InitInstance()
{
    AfxEnableControlContainer();

    // Standard initialization
    // If you are not using these features and wish to reduce the size
    // of your final executable, you should remove from the following
    // the specific initialization routines you do not need.

#ifndef _AFXDLL
    Enable3dControls();           // Call this when using MFC in a shared DLL
#else
    Enable3dControlsStatic();     // Call this when linking to MFC statically
#endif

    // Change the registry key under which our settings are stored.
    // You should modify this string to be something appropriate
    // such as the name of your company or organization.
    //SetRegistryKey(_T("Local AppWizard-Generated Applications"));
    SetRegistryKey("Neural Networks");
}

```

```

LoadStdProfileSettings(8); // Load standard INI file options (including MRU)

// Register the application's document templates. Document templates
// serve as the connection between documents, frame windows and views.

CSingleDocTemplate* pDocTemplate;
pDocTemplate = new CSingleDocTemplate(
    IDR_MAINFRAME,
    RUNTIME_CLASS(CNeuralDoc),
    RUNTIME_CLASS(CMainFrame),      // main SDI frame window
    RUNTIME_CLASS(CNeuralView));
AddDocTemplate(pDocTemplate);

// Enable DDE Execute open
EnableShellOpen();
RegisterShellFileTypes(TRUE);

// Parse command line for standard shell commands, DDE, file open
CCommandLineInfo cmdInfo;
ParseCommandLine(cmdInfo);

// Dispatch commands specified on the command line
if (!ProcessShellCommand(cmdInfo))
    return FALSE;

// The one and only window has been initialized, so show and update it.
m_pMainWnd->ShowWindow(SW_SHOW);
m_pMainWnd->UpdateWindow();

// Enable drag/drop open
m_pMainWnd->DragAcceptFiles();

return TRUE;
}

///////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
    CAboutDlg();

// Dialog Data
    //{{AFX_DATA(CAboutDlg)
    enum { IDD = IDD_ABOUTBOX };
    //}}AFX_DATA

    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CAboutDlg)
protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:
    //{{AFX_MSG(CAboutDlg)
        // No message handlers
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
    //{{AFX_DATA_INIT(CAboutDlg)
    //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{

```

```
CDialog::DoDataExchange(pDX);
//{{AFX_DATA_MAP(CAboutDlg)
//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
//{{AFX_MSG_MAP(CAboutDlg)
//  // No message handlers
//}}AFX_MSG_MAP
END_MESSAGE_MAP()

// App command to run the dialog
void CNeuralApp::OnAppAbout()
{
    CAboutDlg aboutDlg;
    aboutDlg.DoModal();
}

///////////
// CNeuralApp commands

////End - Neural.cpp
```

```

// NeuralDoc.h : interface of the CNeuralDoc class
//
////////////////////////////////////////////////////////////////////////

#ifndef AFX_NEURALDOC_H__D599BF2C_8D75_11D4_89AC_00E0290F3618__INCLUDED_
#define AFX_NEURALDOC_H__D599BF2C_8D75_11D4_89AC_00E0290F3618__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

class CNeuralDoc : public CDocument
{
protected: // create from serialization only
    CNeuralDoc();
    DECLARE_DYNCREATE(CNeuralDoc)

private:
    CBitmap m_bitmap;
// Attributes
public:

// Operations
public:

// Overrides
// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(CNeuralDoc)
public:
    virtual BOOL OnOpenDocument (LPCTSTR);
    virtual void DeleteContents();
    virtual BOOL OnNewDocument();
    virtual void Serialize(CArchive& ar);
//}}AFX_VIRTUAL

// Implementation
public:
    virtual ~CNeuralDoc();
    CBitmap* GetBitmap();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
    //{{AFX_MSG(CNeuralDoc)
        // NOTE - the ClassWizard will add and remove member functions here.
        // DO NOT EDIT what you see in these blocks of generated code !
    }}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_NEURALDOC_H__D599BF2C_8D75_11D4_89AC_00E0290F3618__INCLUDED_)

```

```

// NeuralDoc.cpp : implementation of the CNeuralDoc class
//

#include "stdafx.h"
#include "Neural.h"

#include "NeuralDoc.h"

#ifndef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

///////////
// CNeuralDoc

IMPLEMENT_DYNCREATE(CNeuralDoc, CDocument)

BEGIN_MESSAGE_MAP(CNeuralDoc, CDocument)
    //{{AFX_MSG_MAP(CNeuralDoc)
        // NOTE - the ClassWizard will add and remove mapping macros here.
        // DO NOT EDIT what you see in these blocks of generated code!
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

///////////
// CNeuralDoc construction/destruction

CNeuralDoc::CNeuralDoc()
{
    // TODO: add one-time construction code here
}

CNeuralDoc::~CNeuralDoc()
{
}

BOOL CNeuralDoc::OnNewDocument()
{
    if (!CDocument::OnNewDocument())
        return FALSE;

    // TODO: add reinitialization code here
    // (SDI documents will reuse this document)

    return TRUE;
}

///////////
// CNeuralDoc serialization

void CNeuralDoc::Serialize(CArchive& ar)
{
    if (ar.IsStoring())
    {
        // TODO: add storing code here
    }
    else
    {
        CString strFileName = ar.GetFile()->GetFilePath();
        HBITMAP hBitmap = (HBITMAP)::LoadImage(AfxGetInstanceHandle(),
            (LPCTSTR)strFileName, IMAGE_BITMAP, 0,0,LR_LOADFROMFILE |
            LR_CREATEDIBSECTION);

        if(hBitmap == NULL)

```

```

        AfxThrowArchiveException (CArchiveException::badIndex);

        m_bitmap.Attach (hBitmap);
        // TODO: add loading code here
    }

// CNeuralDoc diagnostics

#ifdef _DEBUG
void CNeuralDoc::AssertValid() const
{
    CDocument::AssertValid();
}

void CNeuralDoc::Dump(CDumpContext& dc) const
{
    CDocument::Dump(dc);
}
#endif //_DEBUG

// CNeuralDoc commands
BOOL CNeuralDoc::OnOpenDocument(LPCTSTR lpszPathName)
{
    if(!CDocument::OnOpenDocument (lpszPathName))
    {
        POSITION pos = GetFirstViewPosition();
        CView* pView = GetNextView (pos);
        pView->OnInitialUpdate();
        return FALSE;
    }
    return TRUE;
}

void CNeuralDoc::DeleteContents()
{
    if((HBITMAP) m_bitmap != NULL)
        m_bitmap.DeleteObject();
    CDocument::DeleteContents();
}

CBitmap* CNeuralDoc::GetBitmap()
{
    return ((HBITMAP) m_bitmap == NULL) ? NULL : &m_bitmap;
}

```

```

// NeuralView.h : interface of the CNeuralView class
//
//{{AFX_H_INCLUDES}}
#ifndef AFX_NEURALVIEW_H__D599BF2E_8D75_11D4_89AC_00E0290F3618__INCLUDED_
#define AFX_NEURALVIEW_H__D599BF2E_8D75_11D4_89AC_00E0290F3618__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

class CNeuralView : public CView
{
protected: // create from serialization only
    CNeuralView();
    DECLARE_DYNCREATE(CNeuralView)

// Attributes
public:
    CNeuralDoc* GetDocument();

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CNeuralView)
public:
    virtual void OnDraw(CDC* pDC); // overridden to draw this view
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
protected:
//}}AFX_VIRTUAL

// Implementation
public:
    virtual ~CNeuralView();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:
    CString Time1;
    CTime startTime;
    CTime endTime;
    CTimeSpan elapsedTime;

    int             Width_Bytes;
    int             Width;
    int             Height;
    int             BitsPerPixel;
    BYTE*          Buffer;
    int*           BitBuffer;
    int             ImageWidthPixel;
    int             ImageHightPixel;
    int             WLow;
    int             WHigh;
    FILE*          stream;
    int*           Id;
    int             TotId;
    char            fname[300];
    char            WFname[300];

//    void ReadTextFile(void);
//    void ReadImage(void);
//    void GetCharacterArea(void);
//    void DisplayTheCharacter(CDC* );
//    void ConverBitToByte(int,int);

```

```

void StoreImageInfo(int ,int ,int ,CString);
void ReadImageInfo(void);
void StoreImage(void);
void GetOutputFromMu(int*,int);
int RWweight(int*,int,int);
void ConvertForCarpGross(void);

//
// void CleanImage(void);
// void Moment(void);
// Generated message map functions
protected:
    //{{AFX_MSG(CNeuralView)
    afx_msg void OnButtonsaveimage();
    afx_msg void OnButtonhopfield();
    afx_msg void OnButtonhamming();
    afx_msg void OnButtoncarpenter();
    afx_msg void OnButtonperceptron();
    afx_msg void OnButtonkohonen();
    afx_msg void OnButtonmlayer();
    //}}AFX_MSG
DECLARE_MESSAGE_MAP()
};

#ifndef _DEBUG // debug version in NeuralView.cpp
inline CNeuralDoc* CNeuralView::GetDocument()
    { return (CNeuralDoc*)m_pDocument; }
#endif

///////////////////////////////
//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_NEURALVIEW_H__D599BF2E_8D75_11D4_89AC_00E0290F3618__INCLUDED_)

```

```

// NeuralView.cpp : implementation of the CNeuralView class
//

#include "stdafx.h"
#include "Neural.h"
#include "NeuralDoc.h"
#include "NeuralView.h"
#include "BmpInfo.h"
#include "ReadImageText.h"
#include "Hopfield.h"
#include "DlgHopfield.h"
#include "Hamming.h"
#include "HammingNetDlg.h"
#include "CarpGross.h"
#include "arpGrossDlg.h"
#include "SLPerceptron.h"
#include "SLPDlg.h"
#include "Hohonen.h"
#include "KohDig.h"
#include "MLPDlg.h"
#include "MLP.h"

#ifndef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

///////////////////////////////
// CNeuralView

CString Message;
int      WhatToDisplay=0;
int      testN=0;

IMPLEMENT_DYNCREATE(CNeuralView, CView)

BEGIN_MESSAGE_MAP(CNeuralView, CView)
    //{{AFX_MSG_MAP(CNeuralView)
    ON_COMMAND(ID_BUTTONSAVEIMAGE, OnButtonsaveimage)
    ON_COMMAND(ID_BUTTONHOPFIELD, OnButtonhopfield)
    ON_COMMAND(ID_BUTTONHAMMING, OnButtonhamming)
    ON_COMMAND(ID_BUTTONCARPENTER, OnButtoncarpenter)
    ON_COMMAND(ID_BUTTONPERCEPTRON, OnButtonperceptron)
    ON_COMMAND(ID_BUTTONKOHONEN, OnButtonkohonen)
    ON_COMMAND(ID_BUTTONMLAYER, OnButtonmlayer)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

///////////////////////////////
// CNeuralView construction/destruction

CNeuralView::CNeuralView()
{
    // TODO: add construction code here
}

CNeuralView::~CNeuralView()
{
}

BOOL CNeuralView::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Modify the Window class or styles here by modifying
    // the CREATESTRUCT cs
}

```

```

        return CView::PreCreateWindow(cs);
    }

///////////////////////////////
// CNeuralView drawing

void CNeuralView::OnDraw(CDC* pDC)
{
    CNeuralDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    CBitmap* pBitmap = pDoc->GetBitmap();

    if(pBitmap != NULL)
    {
        DIBSECTION ds;
        pBitmap->GetObject(sizeof(DIBSECTION),&ds);

        CDC memDC;
        memDC.CreateCompatibleDC (pDC);
        CBitmap* pOldBitmap= memDC.SelectObject(pBitmap);

        // 
        // 
        pDC->BitBlt(0,0,ds.dsBm.bmWidth, ds.dsBm.bmHeight, &memDC,
                      0,0,SRCCOPY);
        pDC->StretchBlt(100,200,100,100, &memDC,0,0,ds.dsBm.bmWidth, ds.dsBm.bmHeight,
                          SRCCOPY);

        memDC.SelectObject(pOldBitmap);
    }

    // TODO: add draw code for native data here
}

///////////////////////////////
// CNeuralView diagnostics

#ifndef _DEBUG
void CNeuralView::AssertValid() const
{
    CView::AssertValid();
}

void CNeuralView::Dump(CDumpContext& dc) const
{
    CView::Dump(dc);
}

CNeuralDoc* CNeuralView::GetDocument() // non-debug version is inline
{
    ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CNeuralDoc)));
    return (CNeuralDoc*)m_pDocument;
}
#endif // _DEBUG

///////////////////////////////
// CNeuralView message handlers

void CNeuralView::OnButtonsaveimage()
{
    // TODO: Add your command handler code here
    CClientDC aDC(this);
    ReadImage();
    DisplayTheCharacter(&aDC);
}

void CNeuralView::ReadImage()
{
    //DWORD A 32-bit unsigned integer or the address of a segment and its associated offset.
}

```

```

BmpInfo bmpInfoDlg;
if(bmpInfoDlg.DoModal()==IDOK)
{
    int WthPixel,HhtPixel,Idd;
    CString filename;
    DWORD length;
    DWORD Image_Length;
    HhtPixel=bmpInfoDlg.m_lHeight;
    WthPixel=bmpInfoDlg.m_lWidth;
    Idd=bmpInfoDlg.m_Id;
    filename=bmpInfoDlg.m_FileName;

    CNeuralDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    CBitmap* pBitmap = pDoc->GetBitmap ();
    if (pBitmap != NULL)
    {
        DIBSECTION ds;
        pBitmap->GetObject (sizeof (DIBSECTION), &ds);

        Width_Bytes=ds.dsBm.bmWidth;
        Width_Bytes=Width_Bytes/8;
        if((ds.dsBm.bmWidth%8)!=0)   Width_Bytes+=1;
        if((Width_Bytes%2)!=0)       Width_Bytes+=1;
        Height=ds.dsBm.bmHeight;
        BitsPerPixel=ds.dsBm.bmBitsPixel;
        length=ds.dsBm.bmWidthBytes*ds.dsBm.bmHeight;
        Width=Width_Bytes;

        //BYTE An 8-bit integer that is not signed.
        if((Buffer= new BYTE[length]) !=0)
        {
            Image_Length=(pBitmap->GetBitmapBits(length,Buffer));
            if(BitBuffer= new int[WthPixel*HhtPixel]) !=0
            {
                ConverBitToByte(HhtPixel,WthPixel);
                StoreImageInfo(HhtPixel,WthPixel,Idd,filename);
                delete[] BitBuffer;
            }
            delete[] Buffer;
        }
    }
}

void CNeuralView::DisplayTheCharacter(CDC* pDC)
{
    CNeuralDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    CBitmap* pBitmap = pDoc->GetBitmap();

    if(pBitmap != NULL)
    {
        DIBSECTION ds;
        pBitmap->GetObject(sizeof(DIBSECTION),&ds);

        CDC memDC;
        memDC.CreateCompatibleDC (pDC);
        CBitmap* pOldBitmap= memDC.SelectObject(pBitmap);

        pDC->BitBlt(100,50,ds.dsBm.bmWidth, ds.dsBm.bmHeight, &memDC,
                     0,0,SRCCOPY);
        memDC.SelectObject(pOldBitmap);
    }
}

```

```

void CNeuralView::ConverBitToByte(int HhtPixel,int WthPixel)
{
    int i,j,k,m=0,row;
    BYTE Data,Bt;
    m=0;
    for(i=0;i<Height;i++)
    {
        row=0;
        for(j=0;j<Width;j++)
        {
            Data=Buffer[i*Width+j];
            Bt=128;
            for(k=0;k<8;k++)
            {
                if(Data & Bt)
                    BitBuffer[m]=-1;
                else
                    BitBuffer[m]=1;
                m++;
                Bt=Bt>>1;
                row++;
                if(row==WthPixel)
                {
                    row=-1;
                    break;
                }
            }
            if(row== -1)
                break;
        }
    }
}

void CNeuralView::StoreImageInfo(int HhtPixel,int WthPixel,int &dd,CString filename)
{
    int i,j,Tot,W,H,len;
    Tot=0;
    strcpy(fname,filename);
    len=strlen(fname);
    if(!(fname[len-4]=='.'))
    {
        Message.Format("File name %s is not has extension",fname);
        AfxMessageBox(Message,MB_OK);
        return;
    }

    fname[len-3]='.';
    fname[len-2]='\n';
    fname[len-1]='\0';

    if((stream=fopen(fname,"r")))
    {
        fscanf(stream,"%d",&Tot);
        fscanf(stream,"%d",&H);
        fscanf(stream,"%d",&W);
        fclose(stream);
        if(!(H==HhtPixel && W==WthPixel))
        {
            Message.Format("Image dimension does not match");
            AfxMessageBox(Message,MB_OK);
            return;
        }
    }
    Tot+= HhtPixel*WthPixel;
}

```

```

if(stream=fopen(fname,"w"))
{
    sprintf(stream,"%d %d %d \n",Tot,HhtPixel,WthPixel);
    fclose(stream);
}

if(stream=fopen(filename,"a"))
{
    fprintf(stream,"n%d\n ",Id);
    for(i=0;i<HhtPixel;i++)
    {
        for(j=0;j<WthPixel;j++)
        {
            fprintf(stream,"%d ",BitBuffer[i*WthPixel+j]);
        }
        fprintf(stream,"\n");
    }
    fclose(stream);
}

void CNeuralView::ReadImageInfo()
{
    int i,j,Tot,W,H,len,pI,indexId,indexB;

    Tot=0;
    CReadImageText DlgRIText;
    if(DlgRIText.DoModal()==IDOK)
    {
        strcpy(fname,DlgRIText.m_Fname);
        strcpy(WFname,DlgRIText.m_WFname);
        len=strlen(fname);
        if(!(fname[len-4]=='.'))
        {
            Message.Format("File name %s' is not has extension",fname);
            AfxMessageBox(Message,MB_OK);
            return;
        }

        fname[len-3]='.';
        fname[len-2]='\n';
        fname[len-1]='\0';

        if((stream=fopen(fname,"r")))
        {
            fscanf(stream,"%d",&Tot);
            fscanf(stream,"%d",&H);
            fscanf(stream,"%d",&W);
            ImageWidthPixel=W;
            ImageHeightPixel=H;
            fclose(stream);
        }
        else
        {
            Message.Format("File %s not found",fname);
            AfxMessageBox(Message,MB_OK);
            return;
        }
        TotId=Tot/(W*H);

        if(!(Id= new int[TotId]))
        {
            Message.Format("Memory allocation problem");
            AfxMessageBox(Message,MB_OK);
            return;
        }

        if(!(BitBuffer= new int[Tot]))
        {

```

```

        Message.Format("Memory allocation problem");
        AfxMessageBox(Message, MB_OK);
        return;
    }

    strcpy(fname,DigRIText.m_Fname);
    if(stream=fopen(fname,"r"))
    {
        indexId=0;
        indexB=0;
        rewind(stream);
        for(i=0;i<TotId;i++)
        {
            fscanf(stream,"%d",&pI);
            Id[indexId++]=pI;
            for(j=0;j<(W*H);j++)
            {
                fscanf(stream,"%d ",&pI);
                BitBuffer[indexB++]=pI;
            }
        }
        fclose(stream);
    }
}

int CNcralView::RWweight(int* w,int length,int RW)
{
    int i,pv;
    length*=length;
    if(RW==0)
    {
        if(stream=fopen(WFname,"w"))
        {
            fprintf(stream,"%d ",length);
            for(i=0;i<length;i++)
            {
                fprintf(stream,"%d ",w[i]);
            }
            fclose(stream);
        }
        else
        {
            Message.Format("File %s not found",WFname);
            AfxMessageBox(Message, MB_OK);
            return 0;
        }
    }
    else if(RW==1)
    {
        if(stream=fopen(WFname,"r"))
        {
            rewind(stream);
            fscanf(stream,"%d",&pv);
            length=pv;
            for(i=0;i<length;i++)
            {
                fscanf(stream,"%d",&pv);
                w[i]=pv;
            }
        }
        fclose(stream);
    }
    else
    {
        Message.Format("File %s not found",WFname);
        AfxMessageBox(Message, MB_OK);
        return 0;
    }
}

```

```

        }
    else if(RW==2)
    {
        if(stream=fopen(WFname,"r"))
        {
            rewind(stream);
            fscanf(stream,"%d",&pv);
            length=pv;
        }
        else
        {
            Message.Format("File %s not found",WFname);
            AfxMessageBox(Message,MB_OK);
            return 0;
        }
    }
    else
    {
        Message.Format("RW flag not equal to 0, 1, or 2");
        AfxMessageBox(Message,MB_OK);
        return 0;
    }
    return length;
}

/*********************************************
*                                     Hopfield Neural Network
********************************************/


void CNeuralView::StoreImage()
{
    int j,k,indexB;
    if(stream=fopen("c:\\Nicholas\\Data\\Result\\HopResult.txt","a"))
    {
        //           indexId=0;
        //           indexB=0;
        //           fprintf(stream,"%d\n ",testN);
        //           for(i=0;i<TotId;i++)
        //           {
        //               fprintf(stream,"%d\n",Id[indexId++]);
        //               for(j=0;j<(ImageHighPixel);j++)
        //               {
        //                   for(k=0;k<(ImageWidthPixel);k++)
        //                   {
        //                       fprintf(stream,"%d ",BitBuffer[indexB++]);
        //                   }
        //                   fprintf(stream,"\n");
        //               }
        //               fprintf(stream,"\n\n");
        //           }
        //           fclose(stream);
    }
}

void CNeuralView::OnButtonhopfield()
{
    // TODO: Add your command handler code here
    int length,i=1,j,TN,indx,count,T_test,TotNode,indx1;
    int TotTData,k,NCR,EndTrainLevel;
    int* Orig;
    long int q,max,m,maxtrain,experiment;
    CHopfield HopF;
    DlgHopfield DlgHopF;
    experiment=0;/0 - no experiment, 1 - do experiment
    if(DlgHopF.DoModal()==IDOK)
    {
        if(DlgHopF.m_work==1)
        {
            ReadImageInfo();
            HopF.deleteall();

```

```

TotNode=ImageWidthPixel*ImageHeightPixel;
length=TotNode*TotNode;
HopF.AllMemory(length,TotId);
indx=0;
for(i=0;i<TotId*TotNode;i++)
{
    HopF.AllData[i]=BitBuffer[i];
}
if(!(Orig= new int[TotId]))
{
    Message.Format("Memory allocation problem");
    AfxMessageBox(Message,MB_OK);
    return;
}
q=0;
maxtrain=0;
if(stream=fopen("c:\\Nicholas\\Data\\Result\\HopTrainResult.txt","w"))
{
    EndTrainLevel=0;
    max=(long int)pow(2,TotId);// pow( base, exponent )
    if(experiment==0)
        max=1;
    while(q<max)
    {
        TotTData=HopF.SelectTrainSample(q,TotId,Orig);
        fprintf(stream,"n %d - Training Numbers are ",q);
        indx1=0;
        for(j=0;j<TotId;j++)
        {
            if(Orig[j]==1)
            {
                for(k=0;k<TotNode;k++)
                    BitBuffer[indx1*TotNode+k]=
                    HopF.AllData[j*TotNode+k];
                indx1++;
                fprintf(stream,"%d ",Id[j]);
            }
        }
        fprintf(stream,"n ");
        HopF.Train(BitBuffer,ImageWidthPixel*ImageHeightPixel,TotTData);
        NCR=0;
        for(j=0;j<TotId;j++)
        {
            if(Orig[j]==1)
            {
                for(k=0;k<TotNode;k++)
                    BitBuffer[k]=HopF.AllData[j*TotNode+k];
                HopF.TestInitialize(BitBuffer,0,TotNode);
                count=0;
                m=0;
                while(m<500)
                {
                    HopF.Test();
                    if(HopF.CV==0)
                    {
                        count++;
                    }
                    else
                    {
                        count=0;
                    }
                    if(count>2)
                        break;
                    m++;
                }
                T_test=0;
                for(k=0;k<TotNode;k++)
                {
                    if(!(HopF.Mu[k]==BitBuffer[k]))

```

```

        T_test=-1;
        //break;
    }
}
if(T_test==0)
{
    NCR++;
    fprintf(stream,"Number %d - %d\n ",Id[j].l);
}
else
{
    fprintf(stream,"Number %d - %s\n ",Id[j],"Not Recognized");
}
}
if(TotTData==NCR)
{
    EndTrainLevel=1;
}
fprintf(stream,"Total Recognition %d\n\n ",NCR);
if(maxtrain<NCR)
    maxtrain=NCR;
q++;
}
fprintf(stream,"Maximum Recognition %d\n\n ",maxtrain);
fclose(stream);
}

RWweight(HopF.t,ImageWidthPixel*ImageHightPixel,0);
delete[] BitBuffer;
delete[] Id;
delete[] Orig;
HopF.deleteall();
Message.Format("Training is completed");
AfxMessageBox(Message,MB_OK);
}
else if(DlgHopF.m_work==2)
{
    ReadImageInfo();
    length=RWweight(HopF.t,length,2);
    HopF.AllMemory(length,1);
    RWweight(HopF.t,length,1);
    HopF.TestInitialize(BitBuffer,0,ImageWidthPixel*ImageHightPixel);
    indx=0;
    i=0;
    TN=ImageWidthPixel*ImageHightPixel;
    if(!(Orig= new int[TN]))
    {
        Message.Format("Memory allocation problem");
        AfxMessageBox(Message,MB_OK);
        return;
    }
    for(j=0;j<TN;j++)
        Orig[j]=BitBuffer[j];
    count=0;
    testN=0;
    GetOutputFromMu(HopF.Mu,indx);
    StoreImage();
    while(i<5000)
    {
        testN=i+1;
        HopF.Test();
        GetOutputFromMu(HopF.Mu,indx);
        StoreImage();
        indx+=ImageWidthPixel*ImageHightPixel;
        HopF.TestInitialize(BitBuffer,indx,ImageWidthPixel*ImageHightPixel);
        HopF.Test();
        GetOutputFromMu(HopF.Mu,indx)
        i++;
    }
}

```





```

*****  

The Carpenter/Grossberg classifier Neural Network  

*****  

*****  

*****  

void CNeuralView::OnButtoncarpenter()  

{  

    // TODO: Add your command handler code here  

    int i=1;  

    CCarpGross CarpGross;  

    CarpGrossDig      DlgCarpGross;  

    DlgCarpGross.m_Work=0;  

    if(DlgCarpGross.DoModal()==IDOK)  

    {  

        if(DlgCarpGross.m_Work==1)  

        {  

            ReadImageInfo();  

            ConvertForCarpGross();  

            CarpGross.Initialization(ImageWidthPixel*ImageHightPixel,TotId);  

            CarpGross.Train(BitBuffer,Id);  

            CarpGross.RWCarpGross(WFname,0);  

            delete[] BitBuffer;  

            delete[] Id;  

            //elapsedTime;  

            //Time=Time/CarpGross.NEpoch;  

            Message.Format("Training is completed - Total epoch = %d and %s per  

epoch",CarpGross.NEpoch,CarpGross.Time);  

            AfxMessageBox(Message,MB_OK);  

            CarpGross.DeleteAll();  

        }  

        else if(DlgCarpGross.m_Work==2)  

        {  

            ReadImageInfo();  

            ConvertForCarpGross();  

            CarpGross.RWCarpGross(WFname,1);  

            CarpGross.Test(BitBuffer,Id,TotId,ImageWidthPixel*ImageHightPixel);  

            delete[] BitBuffer;  

            delete[] Id;  

            Message.Format("The tested Number is %d - Total iteration = %d and %s per  

epoch",CarpGross.RecogId,CarpGross.NEpoch,CarpGross.Time);  

            AfxMessageBox(Message,MB_OK);  

            CarpGross.DeleteAll();  

        }  

        else  

        {  

            Message.Format("Selecting Test or Trin option problem");  

            AfxMessageBox(Message,MB_OK);  

        }  

    }  

}  

void CNeuralView::ConvertForCarpGross()  

{  

    int i,j;  

    for(i=0;i<TotId;i++)  

    {  

        for(j=0;j<(ImageWidthPixel*ImageHightPixel);j++)  

        {  

            if(BitBuffer[i*(ImageWidthPixel*ImageHightPixel)+j]<0)  

            {  

                BitBuffer[i*(ImageWidthPixel*ImageHightPixel)+j]=0;  

            }  

        }  

    }  

}

```

```

}

/*$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
End of Carpenter/Grossberg classifier Neural Network

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$*/



***** *****
The Single perceptron Neural Network
***** *****,


void CNeuralView::OnButtonperceptron()
{
    // TODO: Add your command handler code here
    int i=1,epoach;
    CSLPerceptron SLP;
    SLPDlg DlgSLP;
    DlgSLP.m_Work=0;
    if(DlgSLP.DoModal()==IDOK)
    {
        if(DlgSLP.m_Work==1)
        {
            ReadImageInfo();
            SLP.Initialization(ImageWidthPixel*ImageHightPixel,TotId, Id);
            SLP.Error=99.0f;
            startTime = CTime::GetCurrentTime();
            SLP.TRecog=0;
            epoach=0;
            while(SLP.TRecog<TotId)//(SLP.Error>0)//
            {
                SLP.Train(BitBuffer);
                SLP.Test(BitBuffer);//TestConvergence(BitBuffer);
                epoach++;
            }
            SLP.RWSLP(WFname,0);
            endTime = CTime::GetCurrentTime();
            elapsedTime = (endTime - startTime);
            Time1 =elapsedTime.Format( "Total days: %D, hours: %H, mins: %M, secs: %S" );
            Message.Format("Training is completed - Total epoch = %d and %s per epoch",epoach,Time1);
            AfxMessageBox(Message,MB_OK);
            SLP.DeleteAll();
            delete[] BitBuffer;
            delete[] Id;
        }
        else if(DlgSLP.m_Work==2)
        {
            ReadImageInfo();
            SLP.TestInitial(TotId ,WFname);
            SLP.RWSLP(WFname,1);
            if((ImageWidthPixel*ImageHightPixel)!=SLP.N)
            {
                Message.Format("Number of input nodes does not match with trained NN nodes");
                AfxMessageBox(Message,MB_OK);
                return;
            }
            startTime = CTime::GetCurrentTime();
            SLP.Test(BitBuffer);
            endTime = CTime::GetCurrentTime();
            elapsedTime = (endTime - startTime);
            Time1 =elapsedTime.Format( "Total days: %D, hours: %H, mins: %M, secs: %S" );

            delete[] BitBuffer;
            delete[] Id;
        }
    }
}

```

```

        Message.Format("The tested Number is %d - Total iteration = 1 and %s per
epoch",SLP.ResultId[0],Time1);
        AfxMessageBox(Message,MB_OK);
        SLP.DeleteAll();
    }
else
{
    Message.Format("Selecting Test or Train option problem");
    AfxMessageBox(Message,MB_OK);
}
}
}

```

```

/*$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
End of single perceptron Neural Network
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$*/

```

```

***** * The Kohonen self organizing feature map Neural Network * *****

```

```

void CNeuralView::OnButtonkohonen()
{
    // TODO: Add your command handler code here
    int i=1,epoach;
    CHohenen Koh;
    KohDlg DlgKoh;
    DlgKoh.m_Work=0;
    if(DlgKoh.DoModal()==IDOK)
    {
        if(DlgKoh.m_Work==1)
        {
            ReadImageInfo();
            Koh.Initialization(ImageWidthPixel*ImageHightPixel,TotId.Id);
            Koh.TrainAgain=1;
            epoach=0;
            startTime = CTime::GetCurrentTime();
            while(epoach<5000)/(Koh.TrainAgain==1 && epoach<5000)
            {
                Koh.Train(BitBuffer);
                Koh.ClusterTest(BitBuffer);
                epoach++;
            }
            Koh.RWKoh(WFname,0);
            delete[] BitBuffer;
            delete[] Id;
            endTime = CTime::GetCurrentTime();
            elapsedTime = (endTime - startTime);
            Time1 =elapsedTime.Format( "Total days: %D, hours: %H, mins: %M, secs: %S" );
            Message.Format("Training is completed - Total epoch = %d and %s per epoch",epoach,Time1);
            AfxMessageBox(Message,MB_OK);
            Koh.DeleteAll();
        }
        else if(DlgKoh.m_Work==2)
        {
            int var;
            ReadImageInfo();
            Koh.TestInitial(TotId ,WFname);
            Koh.RWKoh(WFname,1);
            if((ImageWidthPixel*ImageHightPixel)!=Koh.N)

```

```

    {
        Message.Format("Number of input nodes does not match with trained NN nodes");
        AfxMessageBox(Message,MB_OK);
        return;
    }
    startTime = CTime::GetCurrentTime();
    Koh.Test(BitBuffer);
    endTime = CTime::GetCurrentTime();
    elapsedTime = (endTime - startTime);
    Time1 = elapsedTime.Format( "Total days: %D, hours: %H, mins: %M, secs: %S" );
    delete[] BitBuffer;
    delete[] Id;
    var=Koh.ResultId[0];
    var=Koh.ClasterInfo[Koh.Max_G * var+1];
    Message.Format("The tested Number is %d - Total iteration = 1 and %s per epoch",var,Time1);
    AfxMessageBox(Message,MB_OK);
    Koh.DeleteAll();
}
else
{
    Message.Format("Selecting Test or Train option problem");
    AfxMessageBox(Message,MB_OK);
}
}
}

```

# \*\*\*\*\* \* The Multi layer perceptron Neural Network \* \*\*\*\*\*

```

void CNeuralView::OnButtonmlayer()
{
    // TODO: Add your command handler code here
    int ij,Ti,TM=1;
    CDelta DTAAlg;
    CCClientDC aDC(this);
    int L=3;
    int NL[4];

    CMLPDialog DlgMLP;
    DlgMLP.m_Work=0;
    if(DlgMLP.DoModal()==IDOK)
    {
        NL[1]=15;
        NL[2]=10;
        if(DlgMLP.m_Work==1)
        {
            ReadImageInfo();
            NL[0]=ImageWidthPixel*ImageHeightPixel;
            startTime = CTime::GetCurrentTime();
            DTAAlg.Initial(TotId,L,BitBuffer,Id,NL);
            for(i=1,j=0;i<L;i++)
                j+=NL[i]*NL[i-1];
        }
        if(stream=fopen("c:\\Nicholas\\Data\\Error\\DeltaError.txt","w"))
        {
            i=0;
            DTAAlg.TotalE=99.0f;

```

```

DTAAlg.TRecogn=0;
while((DTAAlg.TotalE>0.001) && (i<5000) &&
      (DTAAlg.TRecogn<DTAAlg.M))
{
    DTAAlg.Train();
    fprintf(stream,"%d %d %f
\n",i,DTAAlg.TRecogn,DTAAlg.TotalE);

    i++;
}
fclose(stream);
}
Ti=i;
for(i=1,j=0;i<L;i++)
{
    j+=NL[i]*NL[i-1];
    DTAAlg.ReadWrite(0,j,WFname);
    endTime = CTime::GetCurrentTime();
    elapsedTime = (endTime - startTime);
    Time1 =elapsedTime.Format( "Total days: %D, hours: %H, mins: %M, secs: %S" );
    Message.Format("Trining is completed - Recognition %d out of %d, E = %f, i= %d,
                    Total Time = %s",DTAAlg.TRecogn,DTAAlg.M,DTAAlg.TotalE,Ti,Time1);
    AfxMessageBox(Message,MB_OK);
    delete[] Id;
    delete[] BitBuffer;
}
else if(DlgMLP.m_Work==2)
{
    ReadImageInfo();
    NL[0]=ImageWidthPixel*ImageHeightPixel;
    startTime = CTime::GetCurrentTime();
    DTAAlg.Initial(TotId,L,BitBuffer,Id,NL);
    for(i=1,j=0;i<L;i++)
    {
        j+=NL[i]*NL[i-1];
        DTAAlg.ReadWrite(1,j,WFname);
        DTAAlg.TotalE=DTAAlg.fdelta();
        endTime = CTime::GetCurrentTime();
        elapsedTime = (endTime - startTime);
        Time1 =elapsedTime.Format( "Total days: %D, hours: %H, mins: %M, secs: %S" );
        if(stream=fopen("c:\\Nicholas\\Data\\Error\\DeltaRecog.txt","w"))
        {
            fprintf(stream,"Tested character Recognized character result correct or
                           rejected or miss classified \n\n");
            for(i=0;i<TotId;i++)
            {
                fprintf(stream,"      %d
                           %d
\n",DTAAlg.RecogResult[i*3],DTAAlg.RecogResult[i*3+1],
                    DTAAlg.RecogResult[i*3+2]);
            }
            fclose(stream);
        }
        Message.Format("Testing is completed - Recognized character %d, Recognition %d out
                       of %d, E = %f, Total Time = %s",DTAAlg.RecogResult[1],
                         DTAAlg.TRecogn,DTAAlg.M,DTAAlg.TotalE,Time1);
        AfxMessageBox(Message,MB_OK);
        delete[] Id;
        delete[] BitBuffer;
    }
}
}
}

```

```

// ReadImageText.h : header file
#ifndef AFX_READIMAGETEXT_H_65DD1240_8EEC_11D4_89AC_00E0290F3618_INCLUDED_
#define AFX_READIMAGETEXT_H_65DD1240_8EEC_11D4_89AC_00E0290F3618_INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// ReadImageText.h : header file
//

///////////////////////////////
// CReadImageText dialog

class CReadImageText : public CDialog
{
// Construction
public:
    CReadImageText(CWnd* pParent = NULL); // standard constructor

// Dialog Data
    //{{AFX_DATA(CReadImageText)
    enum { IDD = IDD_DIALOG2 };
    CString m_Fname;
    CString m_WFname;
    //}}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CReadImageText)
protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
    //{{AFX_MSG(CReadImageText)
        // NOTE: the ClassWizard will add member functions here
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_READIMAGETEXT_H_65DD1240_8EEC_11D4_89AC_00E0290F3618_INCLUDED_)

```

```

// ReadImageText.cpp : implementation file
//

#include "stdafx.h"
#include "Neural.h"
#include "ReadImageText.h"

#ifndef _DEBUG
#define new DEBUG_NEW
#endif THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

///////////
// CReadImageText dialog

CReadImageText::CReadImageText(CWnd* pParent /*=NULL*/)
    : CDialog(CReadImageText::IDD, pParent)
{
    //{{AFX_DATA_INIT(CReadImageText)
    m_Fname = _T("c:\\Nicholas\\Data\\Image Info\\Mathy.txt");
    m_WFname = _T("c:\\Nicholas\\Data\\Weight Info\\HopWeight.txt");
    //    m_Fname = _T("c:\\Nick\\Temp\\mathy.txt");
    //    m_WFname = _T("c:\\Nick\\Temp\\KohWeight.txt");
    //}}AFX_DATA_INIT
}

void CReadImageText::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CReadImageText)
    DDX_Text(pDX, IDC_EDITReadImageText, m_Fname);
    DDX_Text(pDX, IDC_EDITWFname, m_WFname);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CReadImageText, CDialog)
    //{{AFX_MSG_MAP(CReadImageText)
        // NOTE: the ClassWizard will add message map macros here
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

///////////
// CReadImageText message handlers

```

## // DlgHopfield.h : header file

```
#if !defined(AFX_DLGHOPFIELD_H__B177E260_8FBA_11D4_89AC_00E0290F3618__INCLUDED_)
#define AFX_DLGHOPFIELD_H__B177E260_8FBA_11D4_89AC_00E0290F3618__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// DlgHopfield.h : header file
//

///////////////////////////////
// DlgHopfield dialog

class DlgHopfield : public CDialog
{
// Construction
public:
    DlgHopfield(CWnd* pParent = NULL); // standard constructor
    int         m_work;

// Dialog Data
    //{{AFX_DATA(DlgHopfield)
    enum { IDD = IDD_DIALOG3 };
        // NOTE: the ClassWizard will add data members here
    //}}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(DlgHopfield)
protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
    //{{AFX_MSG(DlgHopfield)
    afx_msg void OnRADIOTest();
    afx_msg void OnRADIOTrain();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_DLGHOPFIELD_H__B177E260_8FBA_11D4_89AC_00E0290F3618__INCLUDED_)
```

```

// DlgHopfield.cpp : implementation file
//

#include "stdafx.h"
#include "Neural.h"
#include "DlgHopfield.h"

#ifndef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

///////////
// DlgHopfield dialog

DlgHopfield::DlgHopfield(CWnd* pParent /*=NULL*/)
    : CDialog(DlgHopfield::IDD, pParent)
{
    //{{AFX_DATA_INIT(DlgHopfield)
        // NOTE: the ClassWizard will add member initialization here
    //}}AFX_DATA_INIT
}

void DlgHopfield::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(DlgHopfield)
        // NOTE: the ClassWizard will add DDX and DDV calls here
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(DlgHopfield, CDialog)
    //{{AFX_MSG_MAP(DlgHopfield)
    ON_BN_CLICKED(IDC_RADIOTest, OnRADIOTest)
    ON_BN_CLICKED(IDC_RADIOTrain, OnRADIOTrain)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

///////////
// DlgHopfield message handlers

void DlgHopfield::OnRADIOTest()
{
    // TODO: Add your control notification handler code here
    m_work=2;
}

void DlgHopfield::OnRADIOTrain()
{
    // TODO: Add your control notification handler code here
    m_work=1;
}

```

```
class CHopfield
{
protected:

public:
    CHopfield();
    virtual ~CHopfield();

public:
    int*      t;
    int          NNodes;
    int          NSample;
    int*      Mu;
    int*      Mul;
    int          deleteid;
    int          CV;
    int*      AllData;
    CString Message;

    void      deleteall();
    void      Train(int*,int,int);
    void      AllMemory(int,int);
    void      TestInitialize(int*,int,int);
    void      Test(void);
    int       SelectTrainSample(long int,int,int*);

};
```

```

#include "stdafx.h"
#include "Neural.h"
#include "Hopfield.h"

CHopfield::CHopfield()
{
    deleteid=0;
}

CHopfield::~CHopfield()
{
}

void CHopfield::deleteall()
{
    if(deleteid==0)
    {
        return;
    }
    else if(deleteid==1)
    {
        delete[] t;
        deleteid=0;
    }
    else if(deleteid==2)
    {
        delete[] t;
        delete[] Mu;
        deleteid=0;
    }
    else if(deleteid==3)
    {
        delete[] t;
        delete[] Mu;
        delete[] Mul;
        deleteid=0;
    }
    else if(deleteid==4)
    {
        delete[] t;
        delete[] Mu;
        delete[] Mul;
        delete[] AllData;
        deleteid=0;
    }
    else
    {
        Message.Format("Hopfield - delete Memory problem");
        AfxMessageBox(Message, MB_OK);
        return;
    }
}

void CHopfield::Train(int* x,int length,int Tid)
{
    int i,j,s,Tlen;
    NNodes=length;
    Tlen=length*length;

    for(i=0;i<Tlen;i++)
        t[i]=0;
    for(i=0;i<length;i++)
    {
        for(j=0;j<length;j++)
        {
            for(s=0;s<Tid;s++)
            {
                if(i==j)
                {

```

```

        t[i*length+j]+=0;
    }
    else
    {
        t[i*length+j]+=x[s*length+j]*x[s*length+i];
    }
}
}

void CHopfield::AllMemory(int length,int Tid)
{
    int Nodes;
    deleteall();
    Nodes=int(sqrt(length));
    if(!(t=new int[length]))
    {
        Message.Format("Hopfield - Memory allocation problem");
        AfxMessageBox(Message,MB_OK);
        return;
    }
    else
        deleteid=1;
    if(!(Mu=new int[Nodes]))
    {
        Message.Format("Hopfield - Memory allocation problem");
        AfxMessageBox(Message,MB_OK);
        return;
    }
    deleteid=2;
    if(!(Mu1=new int[Nodes]))
    {
        Message.Format("Hopfield - Memory allocation problem");
        AfxMessageBox(Message,MB_OK);
        return;
    }
    deleteid=3;
    if(!(AllData=new int[Nodes*Tid]))
    {
        Message.Format("Hopfield - Memory allocation problem");
        AfxMessageBox(Message,MB_OK);
        return;
    }
    deleteid=4;
}

void CHopfield::TestInitialize(int* x,int StartP,int length)
{
    int i=0;
    NNodes=length;
    for(i=0;i<length;i++)
    {
        Mu[i]=x[i+StartP];
    }
}

void CHopfield::Test(void)
{
    int i,j;
    int Sum;

    for(i=0;i<NNodes;i++)
    {
        Sum=0;
        for(j=0;j<NNodes;j++)
        {
            Sum+=((t[i*NNodes+j])*(Mu[j]));
        }
        if(Sum>=0)

```

```

        {
            MuI[i]=1;
        }
        else
        {
            MuI[i]=-1;
        }

// Sum=(float)exp(-(double)Sum);
// MuI[i]=(float)((2.0/(1.0+Sum))-1);
}
CV=0;
for(i=0;i<NNodes;i++)
{
    if(!(Mu[i]==MuI[i]))
    {
        CV=-1;
    }
    Mu[i]=MuI[i];
}
}

int CHopfield::SelectTrainSample(long int q,int TD,int* TData)
{
    int TNData;
    int i;
    long int x;

    TNData=TD;
    for(i=0;i<TD;i++)
    {
        TData[i]=1;
    }
    x=1;
    for(i=0;i<TD;i++)
    {
        if(x & q)
        {
            TData[i]=0;
            TNData--;
        }
        x=x<<1;
    }
    return TNData;
}

```

```

// HammingNetDlg.h : header file
#ifndef AFX_HAMMINGNETDLG_H_93AB63A0_916E_11D4_89AC_00E0290F3618_INCLUDED_
#define AFX_HAMMINGNETDLG_H_93AB63A0_916E_11D4_89AC_00E0290F3618_INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// HammingNetDlg.h : header file
//

///////////////////////////////
// HammingNetDlg dialog

class HammingNetDlg : public CDialog
{
// Construction
public:
    int         m_Work;
    HammingNetDlg(CWnd* pParent = NULL); // standard constructor

// Dialog Data
    //{{AFX_DATA(HammingNetDlg)
    enum { IDD = IDD_DIALOG4 };
        // NOTE: the ClassWizard will add data members here
    //}}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(HammingNetDlg)
protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
    //{{AFX_MSG(HammingNetDlg)
    afx_msg void OnHamTrain();
    afx_msg void OnHamTest();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_HAMMINGNETDLG_H_93AB63A0_916E_11D4_89AC_00E0290F3618_INCLUDED_)

```

```

// HammingNetDlg.cpp : implementation file
//

#include "stdafx.h"
#include "Neural.h"
#include "HammingNetDlg.h"

#ifndef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

///////////
// HammingNetDlg dialog

HammingNetDlg::HammingNetDlg(CWnd* pParent /*=NULL*/)
    : CDialog(HammingNetDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(HammingNetDlg)
        // NOTE: the ClassWizard will add member initialization here
    //}}AFX_DATA_INIT
}

void HammingNetDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(HammingNetDlg)
        // NOTE: the ClassWizard will add DDX and DDV calls here
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(HammingNetDlg, CDialog)
    //{{AFX_MSG_MAP(HammingNetDlg)
    ON_BN_CLICKED(IDC_HamTrain, OnHamTrain)
    ON_BN_CLICKED(IDC_HamTest, OnHamTest)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

///////////
// HammingNetDlg message handlers

void HammingNetDlg::OnHamTrain()
{
    // TODO: Add your control notification handler code here
    m_Work=1;
}

void HammingNetDlg::OnHamTest()
{
    // TODO: Add your control notification handler code here
    m_Work=2;
}

```

```

class CHamming
{
private:

public:
    int           Deleteindx;
    CString Message;
    FILE*        stream;
    CHamming(void);
    virtual ~CHamming(void);
    void DeleteAll(void);

public:
    float      *w;
    float      *theeta;
    float      *t;
    float      *mu;
    float      *mul;
    int         N;
    int         M;
    float      Eps;
    int         conv;
    int         RegId;

    void Train(int*,int,int);
    void InputPreprocess(int*);
    void RW(char*,int);
    void TestInit(int* x);
    void Test(void);

};


```

```

#include "stdafx.h"
#include "Hamming.h"

float Hamm_min=0,Hamm_max=60;

CHamming::CHamming()
{
    Deleteindx=0;
}

CHamming::~CHamming()
{
}

void CHamming::Train(int* x,int lenght,int nclass)
{
    int i,j;
    float var1;

    DeleteAll();
    N=lenght;
    M=nclass;
    if(!(w=new float[lenght*nclass]))
    {
        Message.Format("Hamming - Memory allocation problem");
        AfxMessageBox(Message,MB_OK);
        return;
    }

    Deleteindx=1;
    if!((theeta=new float[nclass]))
    {
        Message.Format("Hamming - Memory allocation problem");
        AfxMessageBox(Message,MB_OK);
        return;
    }
    Deleteindx=2;
    if!((t=new float[nclass*nclass]))
    {
        Message.Format("Hamming - Memory allocation problem");
        AfxMessageBox(Message,MB_OK);
        return;
    }
    Deleteindx=3;

    var1=lenght/((Hamm_max-Hamm_min)/2.0f);
    InputPreprocess(x);
    for(i=0;i<nclass;i++)
    {
        for(j=0;j<lenght;j++)
        {
            w[i*lenght+j]=x[i*lenght+j]/2.0f;
        }
        theeta[i]=var1;
    }
    Eps=0.8f/nclass;
    for(i=0;i<nclass;i++)
    {
        for(j=0;j<nclass;j++)
        {
            if(i==j)
            {
                t[i*nclass+j]=1;
            }
            else
            {
                t[i*nclass+j]=-Eps;
            }
        }
    }
}

```

```

        }

void CHamming::DeleteAll(void)
{
    if(Deleteindx==1)
    {
        delete[] w;
        Deleteindx=0;
    }
    else if(Deleteindx==2)
    {
        delete[] w;
        delete[] theeta;
        Deleteindx=0;
    }
    else if(Deleteindx==3)
    {
        delete[] w;
        delete[] theeta;
        delete[] t;
        Deleteindx=0;
    }
    else if(Deleteindx==4)
    {
        delete[] w;
        delete[] theeta;
        Deleteindx=0;
        delete[] t;
        delete[] mu;
    }
    else if(Deleteindx==5)
    {
        delete[] w;
        delete[] theeta;
        delete[] t;
        delete[] mu;
        delete[] mu1;
        Deleteindx=0;
    }
    else;
}

void CHamming::RW(char* WFname,int flag)
{
    int i;
    float fp;
    if(flag==0)
    {
        if(Deleteindx==3)
        {
            if(stream=fopen(WFname,"w"))
            {
                fprintf(stream,"%d %d %f\n",N,M,Eps);
                for(i=0;i<N*M;i++)
                {
                    fprintf(stream,"%f ",w[i]);
                }
                fprintf(stream,"\n");
                for(i=0;i<M;i++)
                {
                    fprintf(stream,"%f ",theeta[i]);
                }
                fprintf(stream,"\n");
                for(i=0;i<M*M;i++)
                {
                    fprintf(stream,"%f ",t[i]);
                }
                fclose(stream);
            }
        }
    }
}

```

```

        else
        {
            Message.Format("File %s is not found",WFname);
            AfxMessageBox(Message,MB_OK);
            return;
        }
    }
else
{
    Message.Format("Hamming - Weight file writting problem");
    AfxMessageBox(Message,MB_OK);
    return;
}
else if(flag==1)
{
    DeleteAll();
    if(stream=fopen(WFname,"r"))
    {
        fscanf(stream,"%d",&N);
        fscanf(stream,"%d",&M);
        fscanf(stream,"%f",&Eps);

        if(!(w=new float[N*M]))
        {
            Message.Format("Hamming - Memory allocation problem");
            AfxMessageBox(Message,MB_OK);
            return;
        }

        Deleteindx=1;
        if!((theeta=new float[M]))
        {
            Message.Format("Hamming - Memory allocation problem");
            AfxMessageBox(Message,MB_OK);
            return;
        }
        Deleteindx=2;

        if!((t=new float[M*M]))
        {
            Message.Format("Hamming - Memory allocation problem");
            AfxMessageBox(Message,MB_OK);
            return;
        }
        Deleteindx=3;
        if!((mu=new float[M]))
        {
            Message.Format("Hamming - Memory allocation problem");
            AfxMessageBox(Message,MB_OK);
            return;
        }
        Deleteindx=4;
        if!((mul=new float[M]))
        {
            Message.Format("Hamming - Memory allocation problem");
            AfxMessageBox(Message,MB_OK);
            return;
        }
        Deleteindx=5;
        for(i=0;i<N*M;i++)
        {
            fscanf(stream,"%f ",&fp);
            w[i]=fp;
        }
        for(i=0;i<M;i++)
        {
            fscanf(stream,"%f ",&fp);
            theeta[i]=fp;
        }
    }
}

```

```

        for(i=0;i<M*M;i++)
        {
            fscanf(stream,"%f ",&fp);
            t[i]=fp;
        }
        fclose(stream);
    }
    else
    {
        Message.Format("File %s is not found",WFname);
        AfxMessageBox(Message,MB_OK);
        return;
    }
}
else
{
    Message.Format("Hamming - Weight file writting flag problem");
    AfxMessageBox(Message,MB_OK);
    return;
}

void CHamming::InputPreprocess(int* x)
{
    int i;
    for(i=0;i<N;i++)
    {
        if(x[i]==-1)
            x[i]=0;
    }
}

void CHamming::TestInit(int* x)
{
    int i,j;
    float sum;
    InputPreprocess(x);
    for(i=0;i<M;i++)
    {
        sum=0.0f;
        for(j=0;j<N;j++)
        {
            sum+=(w[i*N+j] * x[j]);
        }
        sum-=theeta[i];
        if(sum>Hamm_max)
        {
            sum=Hamm_max;
        }
        else if(sum<Hamm_min)
        {
            sum=Hamm_min;
        }
        else;
        mu[i]=sum;
    }
}

void CHamming::Test(void)
{
    int i,j;
    float sum;

    for(i=0;i<M;i++)
    {
        sum=0.0f;
        for(j=0;j<M;j++)
        {

```

```

        if(i!=j)           sum+=Eps*mu[j];
    }
    sum=mu[i]-sum;
    if(sum>Hamm_max)
    {
        sum=Hamm_max;
    }
    else if(sum<Hamm_min)
    {
        sum=Hamm_min;
    }
    else;
    mul[i]=sum;
}
for(i=0;i<M;i++)
    mu[i]=mul[i];
conv=0;
RegId=-1;
for(i=0;i<M;i++)
{
    if(mu[i]>Hamm_min)
    {
        conv++;
        RegId=i;
    }
}
}

```

```

// arpGrossDlg.h : header file
#ifndef AFX_ARPGROSSDLG_H_544C3EC0_9739_11D4_A34C_0000C0825656_INCLUDED_
#define AFX_ARPGROSSDLG_H_544C3EC0_9739_11D4_A34C_0000C0825656_INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// arpGrossDlg.h : header file
//

///////////////////////////////
// CarpGrossDlg dialog

class CarpGrossDlg : public CDialog
{
// Construction
public:
    int             m_Work;
    CarpGrossDlg(CWnd* pParent = NULL); // standard constructor

// Dialog Data
    //{{AFX_DATA(CarpGrossDlg)
    enum { IDD = IDD_DIALOG5 };
        // NOTE: the ClassWizard will add data members here
    //}}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CarpGrossDlg)
protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
    //{{AFX_MSG(CarpGrossDlg)
    afx_msg void OnCarpGrossTest();
    afx_msg void OnCarpGrossTrain();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_ARPGROSSDLG_H_544C3EC0_9739_11D4_A34C_0000C0825656_INCLUDED_)

```

```

// arpGrossDlg.cpp : implementation file
//

#include "stdafx.h"
#include "Neural.h"
#include "arpGrossDlg.h"

#ifndef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

///////////
// CarpGrossDlg dialog

CarpGrossDlg::CarpGrossDlg(CWnd* pParent /*=NULL*/)
    : CDialog(CarpGrossDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CarpGrossDlg)
        // NOTE: the ClassWizard will add member initialization here
    //}}AFX_DATA_INIT
}

void CarpGrossDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CarpGrossDlg)
        // NOTE: the ClassWizard will add DDX and DDV calls here
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CarpGrossDlg, CDialog)
    //{{AFX_MSG_MAP(CarpGrossDlg)
    ON_BN_CLICKED(IDC_CarpGrossTest, OnCarpGrossTest)
    ON_BN_CLICKED(IDC_CarpGrossTrain, OnCarpGrossTrain)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

///////////
// CarpGrossDlg message handlers

void CarpGrossDlg::OnCarpGrossTest()
{
    m_Work=2;
    // TODO: Add your control notification handler code here
}

void CarpGrossDlg::OnCarpGrossTrain()
{
    m_Work=1;
    // TODO: Add your control notification handler code here
}

```

```

class CCarpGross
{
protected:

public:
    int           Deleteindx;
    CString Message;
    CString Time;
    FILE*        stream;
    CCarpGross(void);
    CTime startTime;
    CTime endTime;
    CTimeSpan elapsedTime;
    virtual ~CCarpGross(void);
    void DeleteAll(void);

public:
    int           N;
    int           M;
    int           NEpoach;
    int*          t;
    float*        b;
    float*        p;
    float*        mu;
    int*          Disable;
    int*          ResultId;
    //           int           BM;
    //           int           TSample;
    //           int           RecogId;

    void Initialization(int,int);
    void Train(int*,int*);
    void RWCarGross(char*,int);
    void Test(int*,int*,int,int);
};


```

```

#include "stdafx.h"
#include "CarpGross.h"

float VigilanceTh=0.7f/0.685f;
int TotalOutput=10;

CCarpGross::CCarpGross()
{
    Deleteindx=0;
}

CCarpGross::~CCarpGross()
{
}

void CCarpGross::DeleteAll()
{
    if(Deleteindx==1)
    {
        delete[] t;
    }
    else if(Deleteindx==2)
    {
        delete[] t;
        delete[] b;
    }
    else if(Deleteindx==3)
    {
        delete[] t;
        delete[] b;
        delete[] mu;
    }
    else if(Deleteindx==4)
    {
        delete[] t;
        delete[] b;
        delete[] mu;
        delete[] Disable;
    }
    else if(Deleteindx==5)
    {
        delete[] t;
        delete[] b;
        delete[] mu;
        delete[] Disable;
        delete[] ResultId;
    }
    else;
}

void CCarpGross::Initialization(int Nn,int Mm)
{
    int i;
    float var;

    N=Nn;
    TSample=Mm;
    M=TotalOutput;
    DeleteAll();

    if(!(t=new int[N*M]))
    {
        Message.Format("Carpenter|Grossberg - Memory allocation problem");
        AfxMessageBox(Message,MB_OK);
        return;
    }
    Deleteindx=1;
}

```

```

if(!(b=new float[N*M]))
{
    Message.Format("Carpenter|Grossberg - Memory allocation problem");
    AfxMessageBox(Message,MB_OK);
    return;
}
Deleteindx=2;
if(!(mu=new float[M]))
{
    Message.Format("Carpenter|Grossberg - Memory allocation problem");
    AfxMessageBox(Message,MB_OK);
    return;
}
Deleteindx=3;
if!(!Disable=new int[M])
{
    Message.Format("Carpenter|Grossberg - Memory allocation problem");
    AfxMessageBox(Message,MB_OK);
    return;
}
Deleteindx=4;
if!(!ResultId=new int[M])
{
    Message.Format("Carpenter|Grossberg - Memory allocation problem");
    AfxMessageBox(Message,MB_OK);
    return;
}
Deleteindx=5;

var=float(1.0f/(1+N));
for(i=0;i<N*M;i++)
{
    t[i]=1;
    b[i]=var;
}
p=VigilanceTh;
}

void CCarpGross::Train(int* x,int* id)
{
    int i,j,indx,loc,sum,count,OneMoreLoop;
    float max,var,CapX,CapTX;
    CTimeSpan elapsedTime( 0, 0, 0, 0 );

    if(!(stream=fopen("c:\\Nicholas\\Data\\CarpGrossError.txt","w")))
    {
        Message.Format("Carpenter|Grossberg - Cannot write error file");
        AfxMessageBox(Message,MB_OK);
        return;
    }
    for(i=0;i<M;i++)
    {
        Disable[i]=0;
        ResultId[i]=-1;
    }

    OneMoreLoop=1;
    NEpoach=0;
    elapsedTime=elapsedTime1;
    startTime = CTime::GetCurrentTime();
    while(OneMoreLoop && NEpoach<5000)
    {
        OneMoreLoop=0;
        indx=0;
        while(indx<TSample)
        {
            max=0.0f;
            loc=-1;
            for(j=0;j<M;j++)
            {

```

```

        mu[j]=0.0f;
        if(Disable[j]==0)
        {
            for(i=0;i<N;i++)
            {
                mu[j]+=b[j*N+i]*x[indx*N+i];
            }
            if(max<mu[j])
            {
                max=mu[j];
                loc=j;
            }
        }
    }

    CapX=0;
    CapTX=0;
    for(i=0;i<N;i++)
    {
        CapX+=x[indx*N+i];
        CapTX+=(t[loc*N+i]*x[indx*N+i]);
    }
    var=CapTX/CapX;
    if(p>=var)
    {
        Disable[loc]=1;
        count=0;
        for(i=0;i<M;i++)
        {
            if(Disable[i]==1)
                count++;
        }
        if(count==M)
        {
            fprintf(stream,"Input number %d cannot train\n",indx);
            for(i=0;i<M;i++)
            {
                Disable[i]=0;
            }
            indx++;
        }
    }
    else
    {
        sum=0;
        for(i=0;i<N;i++)
        {
            sum+=(t[loc*N+i]*x[indx*N+i]);
            t[loc*N+i]=t[loc*N+i]*x[indx*N+i];
        }
        for(i=0;i<N;i++)
        {
            b[loc*N+i]= float((t[loc*N+i])/(0.5f+sum));
        }
        if(ResultId[loc]!=-1)
            OneMoreLoop=1;
        ResultId[loc]=id[indx];
        indx++;
        for(i=0;i<M;i++)
        {
            Disable[i]=0;
        }
    }
}
if(OneMoreLoop==1)
{
    for(i=0;i<M;i++)
    {
        Disable[i]=0;
    }
}

```

```

        ResultId[i]=-1;
    }
}
NEpoach++;
}
endTime = CTime::GetCurrentTime();
elapsedTime += (endTime - startTime);
Time = elapsedTime.Format( "Total days: %D, hours: %H, mins: %M, secs: %S" );
fclose(stream);
}

//printf( "OS time:\t\t\t%s\n", tmpbuf );
void CCarpGross::RWCarGross(char* filename,int flag)
{
    int i,jp;
    float fp;

    if(flag==1)
    {
        if(stream=fopen(filename,"r"))
        {
            DeleteAll();
            rewind(stream);
            fscanf(stream,"%d %d %f",&N,&M,&p);

            if(!(t=new int[N*M]))
            {
                Message.Format("Carpenter|Grossberg - Memory allocation problem");
                AfxMessageBox(Message,MB_OK);
                return;
            }
            Deleteindx=1;
            if(!(b=new float[N*M]))
            {
                Message.Format("Carpenter|Grossberg - Memory allocation problem");
                AfxMessageBox(Message,MB_OK);
                return;
            }
            Deleteindx=2;
            if!((mu=new float[M]))
            {
                Message.Format("Carpenter|Grossberg - Memory allocation problem");
                AfxMessageBox(Message,MB_OK);
                return;
            }
            Deleteindx=3;
            if!((Disable=new int[M]))
            {
                Message.Format("Carpenter|Grossberg - Memory allocation problem");
                AfxMessageBox(Message,MB_OK);
                return;
            }
            Deleteindx=4;
            if!((ResultId=new int[M]))
            {
                Message.Format("Carpenter|Grossberg - Memory allocation problem");
                AfxMessageBox(Message,MB_OK);
                return;
            }
            Deleteindx=5;

            for(i=0;i<M*N;i++)
            {
                fscanf(stream,"%d",&ip);
                t[i]=ip;
            }
            for(i=0;i<M*N;i++)
            {
                fscanf(stream,"%f",&fp);
                b[i]=fp;
            }
        }
    }
}
```

```

        }
        for(i=0;i<M;i++)
        {
            fscanf(stream,"%d",&ip);
            ResultId[i]=ip;
        }
        fclose(stream);
    }
    else
    {
        Message.Format("Carpenter|Grossberg - %s does not exist",filename);
        AfxMessageBox(Message,MB_OK);
        return;
    }
}
else if(flag==0)
{
    if(stream=fopen(filename,"w"))
    {
        fprintf(stream,"%d %d %f\n",N,M,p);
        for(i=0;i<M*N;i++)
        {
            fprintf(stream,"%d ",t[i]);
        }
        fprintf(stream,"\n");
        for(i=0;i<M*N;i++)
        {
            fprintf(stream,"%f ",b[i]);
        }
        fprintf(stream,"\n");
        for(i=0;i<M;i++)
        {
            fprintf(stream,"%d ",ResultId[i]);
        }
        fclose(stream);
    }
    else
    {
        Message.Format("Carpenter|Grossberg - Cannot allow to write this %s location",filename);
        AfxMessageBox(Message,MB_OK);
        return;
    }
}
else
{
    Message.Format("Carpenter|Grossberg - wrog flag number for RW");
    AfxMessageBox(Message,MB_OK);
    return;
}
}

void CCarpGross::Test(int* x,int* id,int MM, int NN)
{
    int i,j,indx,loc,sum,count;
    float max,var,CapX,CapTX;
    TSample=MM;
    if(N!=NN)
    {
        Message.Format("Carpenter|Grossberg - Dimension of Trained and testing samples did not match");
        AfxMessageBox(Message,MB_OK);
        return;
    }
    if(!(stream=fopen("c:\\Nicholas\\Data\\CarpGrossError.txt","w")))
    {
        Message.Format("Carpenter|Grossberg - Cannot write error file");
        AfxMessageBox(Message,MB_OK);
        return;
    }
    for(i=0;i<M;i++)

```

```

{
    Disable[i]=0;
}
startTime = CTime::GetCurrentTime();
idx=0;
NEpoach=0;
while(idx<TSample)
{
    max=0.0f;
    loc=-1;
    for(j=0;j<M;j++)
    {
        mu[j]=0.0f;
        if(Disable[j]==0)
        {
            for(i=0;i<N;i++)
            {
                mu[j]+=b[j*N+i]*x[idx*N+i];
            }
            if(max<mu[j])
            {
                max=mu[j];
                loc=j;
            }
        }
    }
}

CapX=0;
CapTX=0;
for(i=0;i<N;i++)
{
    CapX+=x[idx*N+i];
    CapTX+=(t[loc*N+i]*x[idx*N+i]);
}
var=float(CapTX/CapX);
if(p>=var)
{
    Disable[loc]=1;
    count=0;
    for(i=0;i<M;i++)
    {
        if(Disable[i]==1)
            count++;
    }

    if(count==M)
    {
        fprintf(stream,"Input number %d cannot train\n",idx);
        for(i=0;i<M;i++)
        {
            Disable[i]=0;
        }
        RecogId=-1;
        idx++;
    }
}
else
{
    if(ResultId[loc]==0)
    {
        sum=0;
        for(i=0;i<N;i++)
        {
            t[loc*N+i]=t[loc*N+i]*x[idx*N+i];
            sum+=(t[loc*N+i]*x[idx*N+i]);
        }
        for(i=0;i<N;i++)
        {
            b[loc*N+i]= float((t[loc*N+i])/(0.5f+sum));
        }
    }
}

```

```

        ResultId[loc]=id[idx];
        RecogId=0;
        idx++;
        for(i=0;i<M;i++)
        {
            Disable[i]=0;
        }
    }
    else
    {
        RecogId=ResultId[loc];
        idx++;
    }
}
NEpoach++;

}

endTime = CTime::GetCurrentTime();
elapsedTime = (endTime - startTime);
Time =elapsedTime.Format( "Total days: %D, hours: %H, mins: %M, secs: %S" );
fclose(stream);
}

```

```

// SLPDIg dialog
#ifndef AFX_SLPDLG_H__BF016E45_97AF_11D4_A34C_0000C0825656_INCLUDED_
#define AFX_SLPDLG_H__BF016E45_97AF_11D4_A34C_0000C0825656_INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// SLPDIg.h : header file
//

///////////////////////////////
// SLPDIg dialog

class SLPDIg : public CDialog
{
// Construction
public:
    SLPDIg(CWnd* pParent = NULL); // standard constructor
    int m_Work;

// Dialog Data
    //{{AFX_DATA(SLPDIg)
    enum { IDD = IDD_DIALOG6 };
        // NOTE: the ClassWizard will add data members here
    //}}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(SLPDIg)
protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
    //{{AFX_MSG(SLPDIg)
    afx_msg void OnSLPTraining();
    afx_msg void OnSLPTesting();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_SLPDLG_H__BF016E45_97AF_11D4_A34C_0000C0825656_INCLUDED_)

```

```

// SLPDlg.cpp : implementation file
//

#include "stdafx.h"
#include "Neural.h"
#include "SLPDlg.h"

#ifndef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

///////////
// SLPDlg dialog

SLPDlg::SLPDlg(CWnd* pParent /*=NULL*/)
    : CDialog(SLPDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(SLPDlg)
        // NOTE: the ClassWizard will add member initialization here
    //}}AFX_DATA_INIT
}

void SLPDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(SLPDlg)
        // NOTE: the ClassWizard will add DDX and DDV calls here
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(SLPDlg, CDialog)
    //{{AFX_MSG_MAP(SLPDlg)
        ON_BN_CLICKED(IDC_SLPTraining, OnSLPTraining)
        ON_BN_CLICKED(IDC_SLPTesting, OnSLPTesting)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

///////////
// SLPDlg message handlers

void SLPDlg::OnSLPTraining()
{
    // TODO: Add your control notification handler code here
    m_Work=1;
}

void SLPDlg::OnSLPTesting()
{
    // TODO: Add your control notification handler code here
    m_Work=2;
}

```

```

class CSLPerceptron
{
protected:
public:
    int           Deleteindx;
    CString Message;
    FILE*        stream;
    CSLPerceptron(void);
    virtual ~CSLPerceptron(void);
    void DeleteAll();

public:
    int           N;
    int           NSample;
    int           M;
    float*       d;
    float*       w;
    float        theeta;
    float        eta;
    float*       y;
    float        Error;
    int*         ResultId;
    int          TRecog;

    void Initialization(int,int,int*);
    void Train(int* );
    void TestConvergence(int* );
    void RWSLP(char*,int);
    void TestInitial(int ,char* );
    void Test(int* );
};


```

```

#include "stdafx.h"
#include "SLPerceptron.h"

int NOutput=10;
float THEETA=0.1f;
float ETA=0.5f;

CSLPerceptron::CSLPerceptron()
{
    Deleteindx=0;
    Error=99.0f;
}

CSLPerceptron::~CSLPerceptron()
{
}

void CSLPerceptron::DeleteAll()
{
    if(Deleteindx==1)
    {
        delete[] w;
    }
    else if(Deleteindx==2)
    {
        delete[] w;
        delete[] d;
    }
    else if(Deleteindx==3)
    {
        delete[] w;
        delete[] d;
        delete[] y;
    }
    else if(Deleteindx==4)
    {
        delete[] w;
        delete[] d;
        delete[] y;
        delete[] ResultId;
    }
    else;
    Deleteindx=0;
}

void CSLPerceptron::Initialization(int Nn,int NSamp,int* Oid)
{
    int i,itn;
    float rtn;

    N=Nn;
    NSample=NSamp;
    M=NOutput;
    theeta=THEETA;
    eta=ETA;

    DeleteAll();
    if(!(w=new float[N*M]))
    {
        Message.Format("SLP - Memory allocation problem");
        AfxMessageBox(Message,MB_OK);
        return;
    }
    Deleteindx=1;
    if(!(d=new float[NSample*M]))
    {
        Message.Format("SLP - Memory allocation problem");

```

```

        AfxMessageBox(Message,MB_OK);
        return;
    }
    Deleteindx=2;
    if(!(y=new float[M]))
    {
        Message.Format("SLP - Memory allocation problem");
        AfxMessageBox(Message,MB_OK);
        return;
    }
    Deleteindx=3;
    if(!(ResultId=new int[NSample]))
    {
        Message.Format("SLP - Memory allocation problem");
        AfxMessageBox(Message,MB_OK);
        return;
    }
    Deleteindx=4;
    srand( (unsigned)time( NULL ) );
    itm=N*M;
    rtm=1.0f*itm;
    for(i=0;i<N*M;i++)
    {
        w[i]=((rand()%itm)/rtm)-0.5f;
    }

    for(i=0;i<NSample*M;i++)
    {
        d[i]=-1.0f;
    }
    for(i=0;i<NSample;i++)
    {
        d[i*M+Oid[i]]=1.0f;
    }
}

void CSLPerceptron::Train(int* x)
{
    int i,j,k,l;
    float sum;

    for(i=0;i<NSample;i++)
    {
        k=i;

        for(l=0;l<M;l++)
        {
            sum=0;
            for(j=0;j<N;j++)
            {
                sum+=((w[l*N+j])*(x[k*N+j]));
            }
            sum=theeta;
            if(sum>0)
            {
                sum=1.0f;
            }
            else if(sum<=0)
            {
                sum=-1.0f;
            }
            else;
            y[l]=sum;
        }
        for(j=0;j<M;j++)
        {
            for(l=0;l<N;l++)
            {
                w[j*N+l]+=(eta*(d[k*M+j]-y[j])*x[k*N+l]);
            }
        }
    }
}

```

```

        }
    }

void CSLPerceptron::TestConvergence(int* x)
{
    int i,j,l,k;
    float sum;
    Error=0.0f;
    for(i=0;i<NSample;i++)
    {
        k=i;

        for(l=0;l<M;l++)
        {
            sum=0;
            for(j=0;j<N;j++)
            {
                sum+=((w[l*N+j])*(x[k*N+j]));
            }
            sum=theeta;
            if(sum>0)
            {
                sum=1.0f;
            }
            else if(sum<=0)
            {
                sum=-1.0f;
            }
            else;
            y[l]=sum;
        }
        for(j=0;j<M;j++)
        {
            Error+=((d[k*M+j]-y[j])*(d[k*M+j]-y[j]));
        }
    }
}

void CSLPerceptron::RWSLP(char* weightname,int flag)
{
    int i;
    float fp;
    if(flag==0)
    {
        if(stream=fopen(weightname,"w"))
        {
            fprintf(stream,"%d %d %f %f\n",N,M,theeta,eta);
            for(i=0;i<M*N;i++)
            {
                fprintf(stream,"%f ",w[i]);
            }
            fclose(stream);
        }
        else
        {
            MessageFormat("SLP - Cannot write into %s file",weightname);
            AfxMessageBox(Message,MB_OK);
            return;
        }
    }
    else if(flag==1)
    {
        if(stream=fopen(weightname,"r"))
        {
            rewind(stream);
            fscanf(stream,"%d %d %f %f",&N,&M,&theeta,&eta);
        }
    }
}

```

```

        for(i=0;i<M*N;i++)
        {
            fscanf(stream,"%f",&fp);
            w[i]=fp;
        }
        fclose(stream);
    }
    else
    {
        Message.Format("SLP - %s file does not exist",weightname);
        AfxMessageBox(Message,MB_OK);
        return;
    }
}
else
{
    Message.Format("SLP - file read write flag number(%d) does not exist",flag);
    AfxMessageBox(Message,MB_OK);
    return;
}
}

void CSLPerceptron::TestInitial(int NSamp,char* weightname)
{
    NSample=NSamp;
    DeleteAll();
    if(stream=fopen(weightname,"r"))
    {
        rewind(stream);
        fscanf(stream,"%d %d",&N,&M);
        fclose(stream);
    }
    else
    {
        Message.Format("SLP - file %s does not exist",weightname);
        AfxMessageBox(Message,MB_OK);
        return;
    }

    if(!(w=new float[N*M]))
    {
        Message.Format("SLP - Memory allocation problem");
        AfxMessageBox(Message,MB_OK);
        return;
    }
    Deleteindx=1;
    if(!(d=new float[NSample*M]))
    {
        Message.Format("SLP - Memory allocation problem");
        AfxMessageBox(Message,MB_OK);
        return;
    }
    Deleteindx=2;
    if(!(y=new float[M]))
    {
        Message.Format("SLP - Memory allocation problem");
        AfxMessageBox(Message,MB_OK);
        return;
    }
    Deleteindx=3;
    if(!(ResultId=new int[NSample]))
    {
        Message.Format("SLP - Memory allocation problem");
        AfxMessageBox(Message,MB_OK);
        return;
    }
    Deleteindx=4;
}

```

```

void CSLPerceptron::Test(int* x)
{
    int i,j,l,k,count,loc;
    float sum;
    Error=0.0f;
    TRecog=0;
    for(i=0;i<NSample;i++)
    {
        k=i;

        for(l=0;l<M;l++)
        {
            sum=0;
            for(j=0;j<N;j++)
            {
                sum+=((w[l*N+j])*(x[k*N+j]));
            }
            sum-=theeta;
            if(sum>0)
            {
                sum=1.0f;
            }
            else if(sum<=0)
            {
                sum=-1.0f;
            }
            else;
            y[l]=sum;
        }
        count=0;
        for(j=0;j<M;j++)
        {
            if(y[j]>0)
            {
                count++;
                loc=j;
            }
            else
            {
            }
        }
        if(count==1)
        {
            ResultId[i]=loc;
            TRecog++;
        }
        else
        {
            ResultId[i]=-1;
        }
    }
}

```

```

// MLPDlg.h : header file
#ifndef AFX_MLPDLG_H__CF44EF42_C637_11D5_A34D_0000C0825656_INCLUDED_
#define AFX_MLPDLG_H__CF44EF42_C637_11D5_A34D_0000C0825656_INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
//

///////////////////////////////
// CMLPDLg dialog

class CMLPDLg : public CDlg
{
// Construction
public:
    int         m_Work;
    CMLPDLg(CWnd* pParent = NULL); // standard constructor

// Dialog Data
    //{{AFX_DATA(CMLPDLg)
    enum { IDD = IDD_DIALOG8 };
        // NOTE: the ClassWizard will add data members here
    //}}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CMLPDLg)
protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
    //{{AFX_MSG(CMLPDLg)
    afx_msg void OnMLPTest();
    afx_msg void OnMLPTrain();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_MLPDLG_H__CF44EF42_C637_11D5_A34D_0000C0825656_INCLUDED_)

```

```

// MLPDlg.cpp : implementation file
//

#include "stdafx.h"
#include "Neural.h"
#include "MLPDlg.h"

#ifndef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

///////////
// CMLPDlg dialog

CMLPDlg::CMLPDlg(CWnd* pParent /*=NULL*/)
    : CDialog(CMLPDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CMLPDlg)
        // NOTE: the ClassWizard will add member initialization here
    //}}AFX_DATA_INIT
}

void CMLPDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CMLPDlg)
        // NOTE: the ClassWizard will add DDX and DDV calls here
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CMLPDlg, CDialog)
    //{{AFX_MSG_MAP(CMLPDlg)
    ON_BN_CLICKED(IDC_MLPTest, OnMLPTest)
    ON_BN_CLICKED(IDC_MLPTrain, OnMLPTrain)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

///////////
// CMLPDlg message handlers

void CMLPDlg::OnMLPTest()
{
    // TODO: Add your control notification handler code here
    m_Work=2;
}

void CMLPDlg::OnMLPTrain()
{
    // TODO: Add your control notification handler code here
    m_Work=1;
}

```

```

class CDelta
{
private:

protected:

public:
    int      Deleteindx;
    CString Message;
    FILE*   stream;
    CDelta(void);
    virtual ~CDelta(void);
    void DeleteAll(void);

public:
    float    st;
    float    THold;
    int*    NS;
    int*    NL;
    int          L;
    float*   x;
    int*    xp;
    int          M;
    float*   df;
    float*   net;
    float*   delta;
    float*   y;
    float    Slope;
    float*   d;
    float    theta;
    int          TRecogn;
    float    TotalE;
    int*    RecogResult;

    void Initial(int,int,int*,int*,int*);
    void Train(void);
    void deltafun(void);
    float fdelta(void);
    void ReadWrite(int,int,char* );
};


```

```

#include "stdafx.h"
#include "MLP.h"
#define fd(i) (1.0-i*i)

CDelta::CDelta()
{
    Deleteindx=0;
    Slope=0.2f;
    theta=0.1f;
    THold=0.5f;
    st=0.2f;
}

CDelta::~CDelta()
{
}

void CDelta::DeleteAll()
{
    if(Deleteindx==1)
    {
        delete[] NS;
    }
    else if(Deleteindx==2)
    {
        delete[] NL;
        delete[] NS;
    }
    else if(Deleteindx==3)
    {
        delete[] NL;
        delete[] x;
        delete[] NS;
    }
    else if(Deleteindx==4)
    {
        delete[] NL;
        delete[] x;
        delete[] NS;
        delete[] df;
    }
    else if(Deleteindx==5)
    {
        delete[] NL;
        delete[] x;
        delete[] NS;
        delete[] df;
        delete[] net;
    }
    else if(Deleteindx==6)
    {
        delete[] NL;
        delete[] x;
        delete[] NS;
        delete[] df;
        delete[] net;
        delete[] delta;
    }
    else if(Deleteindx==7)
    {
        delete[] NL;
        delete[] x;
        delete[] NS;
        delete[] df;
        delete[] net;
        delete[] delta;
        delete[] y;
    }
    else if(Deleteindx==8)
}

```

```

    {
        delete[] NL;
        delete[] x;
        delete[] NS;
        delete[] df;
        delete[] net;
        delete[] delta;
        delete[] y;
        delete[] xp;
    }
    else if(Deleteindx==9)
    {
        delete[] NL;
        delete[] x;
        delete[] NS;
        delete[] df;
        delete[] net;
        delete[] delta;
        delete[] y;
        delete[] xp;
        delete[] d;
    }
    else if(Deleteindx==10)
    {
        delete[] NL;
        delete[] x;
        delete[] NS;
        delete[] df;
        delete[] net;
        delete[] delta;
        delete[] y;
        delete[] xp;
        delete[] d;
        delete[] RecogResult;
    }
    else
    {
    }
    Deleteindx=0;
}

void CDelta::Initial(int TS,int TL,int* xxp,int* iid,int* nNL)
{
    int i,j,N;
    M=TS;
    L=TL;

    DeleteAll();

    if(!(NS=new int[L]))
    {
        Message.Format("CDelta-Memory allocation problem");
        AfxMessageBox(Message,MB_OK);
        return;
    }
    Deleteindx=1;
    if(!(NL=new int[L]))
    {
        Message.Format("CDelta-Memory allocation problem");
        AfxMessageBox(Message,MB_OK);
        return;
    }
    Deleteindx=2;

    for(i=0;i<L;i++)
    {
        NL[i]=nNL[i];
    }
}

```

```

NS[0]=NL[0]*NL[1];
for(i=1;i<L-2;i++) NS[i]=NS[i-1]+NL[i]*NL[i+1];

for(i=0,N=0;i<L-1;i++) N+=NL[i]*NL[i+1];

if(!(x=new float[N]))
{
    Message.Format("CDelta-Memory allocation problem");
    AfxMessageBox(Message,MB_OK);
    return;
}
Deleteindx=3;
 srand( (unsigned)time( NULL ) );
for(i=0;i<N;i++)
    x[i]=(float)((rand()%N)/(float)N)-0.5f;

if(!(df=new float[N]))
{
    Message.Format("CDelta-Memory allocation problem");
    AfxMessageBox(Message,MB_OK);
    return;
}
Deleteindx=4;
if(!(net=new float[NL[L-1]]))
{
    Message.Format("CDelta-Memory allocation problem");
    AfxMessageBox(Message,MB_OK);
    return;
}
Deleteindx=5;
for(i=1,N=0;i<L;i++)
    N+=NL[i];
if(!(delta=new float[N]))
{
    Message.Format("CDelta-Memory allocation problem");
    AfxMessageBox(Message,MB_OK);
    return;
}
Deleteindx=6;
if(!(y=new float[N]))
{
    Message.Format("CDelta-Memory allocation problem");
    AfxMessageBox(Message,MB_OK);
    return;
}
Deleteindx=7;
if(!(xp=new int[M*NL[0]]))
{
    Message.Format("CDelta-Memory allocation problem");
    AfxMessageBox(Message,MB_OK);
    return;
}
Deleteindx=8;
for(i=0;i<M;i++)
{
    for(j=0;j<NL[0];j++)
    {
        xp[j*M+i]=x*xp[i*NL[0]+j];
    }
}
if(!(d=new float[NL[L-1]]))
{
    Message.Format("CDelta-Memory allocation problem");
    AfxMessageBox(Message,MB_OK);
    return;
}
Deleteindx=9;
if!(RecogResult=new int[M*3])

```

```

    {
        Message.Format("CDelta-Memory allocation problem");
        AfxMessageBox(Message, MB_OK);
        return;
    }
    Deleteindx=10;
    for(i=0;i<NL[L-1]*M;i++)
    {
        d[i]=-0.5f;
    }
    for(i=0;i<M;i++)
    {
        RecogResult[i*3]=iid[i];
        d[i+M*(iid[i])]=0.5f;
    }
}

void CDelta::Train()
{
    int i,N;

    for(i=0,N=0;i<L-1;i++) N+=NL[i]*NL[i+1];

    deltafun();
    for(i=0;i<N;i++)
        x[i]+=st*df[i];
    TotalE=fdelta();
}

void CDelta::deltafun(void)
{
    // Source: [SID94]
    int i,j,k,m,n,N,Nt1,Nt2,Nt3,ii,z;
    float E,error,sum,ne;

    for(i=0,N=0;i<L-1;i++) N+=NL[i]*NL[i+1];
    for(i=0;i<N;i++)
        df[i]=0.0f;
    for(i=0;i<M;i++)
    {
        for(z=0;z<NL[1];z++)
        {
            for(j=0,ne=theta;j<NL[0];j++)
                ne+=x[z+j*NL[1]]*xp[j*M+i];
            E=(float)exp(-(double)Slope*ne);
            y[z]=(float)((2.0/(1.0+E))-1);
        }
        Nt1=NL[1];
        Nt2=0;
        for(n=2;n<L-1;n++)
        {
            for(z=0;z<NL[n];z++)
            {
                m=Nt1+z;
                for(j=0,ne=theta;j<NL[n-1];j++)
                    ne+=x[Ns[n-2]+z+j*NL[n]]*y[j+Nt2];
                E=(float)exp(-(double)Slope*ne);
                y[m]=(float)((2.0/(1.0+E))-1);
            }
            Nt1+=NL[n];
            Nt2+=NL[n-1];
        }
        for(z=0;z<NL[L-1];z++)
    {
}

```

```

m=Nt1+z;
for(j=0,net[z]=theta;j<NL[L-2];j++)
    net[z] +=x[NS[L-3]+z+j*NL[L-1]]*y[j+Nt2];
E=(float)exp(-(double)Slope*net[z]);
y[m]=(float)((2.0/(1.0+E))-1);
}

for(z=1,Nt1=0;z<(L-1);z++)
    Nt1+=NL[z];

for(z=0;z<NL[L-1];z++)
{
    ii=Nt1+z;
    error=float(d[i+z*M]-y[ii]);
    delta[ii]=float(error*fd(y[ii])*Slope/2.0f);
}

for(m=0;m<(L-2);m++)
{
    Nt2=Nt1-NL[L-2-m];
    for(z=0;z<NL[L-2-m];z++)
    {
        ii=Nt2+z;
        sum=0.0;
        n=NS[L-3-m];
        for(j=0;j<NL[L-1-m];j++)
            sum+=delta[Nt1+j]*x[n+j+z*NL[L-1-m]];
        delta[ii]=float(sum*fd(y[ii])*Slope/2.0f);
    }
    Nt1=Nt2;
}
for(k=0;k<NL[1];k++)
{
    for(j=0;j<NL[0];j++)
    {
        df[k+j*NL[1]]+=delta[k]*xp[i+j*M];
    }
}
Nt1=NS[0]; Nt2=0;
Nt3=NL[1];
for(m=1;m<(L-1);m++)
{
    for(k=0;k<NL[m+1];k++)
        for(j=0;j<NL[m];j++)
            df[Nt1+k+j*NL[m+1]]+=delta[Nt3+k]*y[Nt2+j];
    Nt1=NS[m];
    Nt2+=NL[m];
    Nt3+=NL[m+1];
}
}

float CDelta::fdelta(void)
{
    // Source: [SID94]
    float net,error=0.0,q1=0.0f;
    float E;
    int k,j,i,n,m,Nt1,Nt2,code,count,IdN,check,ch,recgC;
    float* w;
    TRecogn=0;
    w=x;
    for(k=0;k<M;k++)
    {
        code=0;
        count=0;
        IdN=-101;
        check=0;
        for(i=0;i<NL[1];i++)
        {
            for(j=0,net=theta;j<NL[0];j++)

```

```

        net+=w[i+j*NL[1]]*xp[j*M+k];
        E=(float)exp(-(double)Slope*net);
        y[i]=(float)((2.0/(1.0+E))-1);
    }

    Nt1=NL[1];
    Nt2=0;

    for(n=2;n<L;n++)
    {
        for(i=0;i<NL[n];i++)
        {
            m=Nt1+i;
            for(j=0,net=theta;j<NL[n-1];j++)
                net+=w[NL[n-2]+i+j*NL[n]]*y[j+Nt2];
            E=(float)exp(-(double)Slope*net);
            y[m]=(float)((2.0/(1.0+E))-1);
        }
        Nt1+=NL[n];
        Nt2+=NL[n-1];
    }

    recgC=-1;
    ch=1;
    for(i=0;i<NL[L-1];i++)
    {
        error=float(d[k+i*M]-y[Nt2+i]);
        q1+=(error*error);
        if(fabs(error)>THold)
        {
            code=-101;
        }
        if(d[k+i*M]>0)
        {
            recgC=i;
            count++;
            IdN=i;
        }
        ch=ch<<1;
    }
    if(code==0)
    {
        if(count!=1)
        {
            IdN=-1;
            check=-1;
        }
        else
            check=1;
        TRecogn++;
    }
    else
        IdN=-101;
    RecogResult[k*3+1]=IdN;
    RecogResult[k*3+2]=check;
}//K-loop end here
q1/=2.0;
return q1;
}

void CDelta::ReadWrite(int flag,int tot,char* wname)
{
    FILE* stream;
    int i;
    float fp;

    if(flag==0)
    {
        if(stream=fopen(wname,"w"))

```

```

    {
        for(i=0;i<tot;i++)
        {
            fprintf(stream,"%f ",x[i]);
        }
        fclose(stream);
    }
    else
    {
        Message.Format("CDelta-File %s cannot open for write",wname);
        AfxMessageBox(Message,MB_OK);
        return;
    }
}
else if(flag==1)
{
    if(stream=fopen(wname,"r"))
    {
        for(i=0;i<tot;i++)
        {
            fscanf(stream,"%f",&fp);
            x[i]=fp;
        }
        fclose(stream);
    }
    else
    {
        Message.Format("CDelta-File %s cannot open",wname);
        AfxMessageBox(Message,MB_OK);
        return;
    }
}
else
{
    Message.Format("CDelta-It is not read or write flag number");
    AfxMessageBox(Message,MB_OK);
    return;
}
}

```

```

// KohDlg.h : header file
#ifndef AFX_KOHDLG_H_0912C680_A045_11D4_A34C_0000C0825656_INCLUDED_
#define AFX_KOHDLG_H_0912C680_A045_11D4_A34C_0000C0825656_INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// KohDlg.h : header file
//

///////////////////////////////
// KohDlg dialog

class KohDlg : public CDialog
{
// Construction
public:
    KohDlg(CWnd* pParent = NULL); // standard constructor
    int             m_Work;

// Dialog Data
//{{AFX_DATA(KohDlg)
enum { IDD = IDD_DIALOG7 };
// NOTE: the ClassWizard will add data members here
//}}AFX_DATA


// Overrides
// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(KohDlg)
protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
//}}AFX_VIRTUAL

// Implementation
protected:

// Generated message map functions
//{{AFX_MSG(KohDlg)
afx_msg void OnKohTest();
afx_msg void OnKohTrain();
//}}AFX_MSG
DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_KOHDLG_H_0912C680_A045_11D4_A34C_0000C0825656_INCLUDED_)

```

```

// KohDlg.cpp : implementation file
//

#include "stdafx.h"
#include "Neural.h"
#include "KohDlg.h"

#ifndef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

///////////
// KohDlg dialog

KohDlg::KohDlg(CWnd* pParent /*=NULL*/)
    : CDialog(KohDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(KohDlg)
        // NOTE: the ClassWizard will add member initialization here
    //}}AFX_DATA_INIT
}

void KohDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(KohDlg)
        // NOTE: the ClassWizard will add DDX and DDV calls here
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(KohDlg, CDialog)
    //{{AFX_MSG_MAP(KohDlg)
        ON_BN_CLICKED(IDC_KohTest, OnKohTest)
        ON_BN_CLICKED(IDC_KohTrain, OnKohTrain)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

///////////
// KohDlg message handlers

void KohDlg::OnKohTest()
{
    // TODO: Add your control notification handler code here
    m_Work=2;
}

void KohDlg::OnKohTrain()
{
    // TODO: Add your control notification handler code here
    m_Work=1;
}

```

```

class CHohonen
{
private:

public:    int           Deleteindx;
CString Message;
FILE*      stream;
CHohonen(void);
virtual ~CHohonen(void);
void DeleteAll(void);

public:    float*       w;
float*      d;
int          N;
int          NSample;
int          CountRad;
int          M;
float        eta;
int*        BufferId;
int          NE;
int          OW;
int          OH;
int          RadChange;
int          LRowCol;
float        Deta;
int*        ClasterInfo;
int*        ResultId;
int          Max_G;
int          TrainAgain;

void Initialization(int,int,int* );
void Train(int* );
void ClusterTest(int* );
void RWKoh(char* ,int);
void TestInitial(int,char* );
void Test(int* );

};


```

```

#include "stdafx.h"
#include "Hohonon.h"

int HNOutput=30;
float HETA=0.7f;
int      Max_Iter=4000;
int Radious=10;
int Max_Group=11;//Maximum overlap=11-1=10

CHohonon::CHohonon()
{
    Deleteindx=0;
    eta=HETA;
    CountRad=0;
}

CHohonon::~CHohonon()
{
}

void CHohonon::DeleteAll()
{
    if(Deleteindx==1)
    {
        delete[] w;
    }
    else if(Deleteindx==2)
    {
        delete[] w;
        delete[] d;
    }
    else if(Deleteindx==3)
    {
        delete[] w;
        delete[] d;
        delete[] BufferId;
    }
    else if(Deleteindx==4)
    {
        delete[] w;
        delete[] d;
        delete[] BufferId;
        delete[] ClasterInfo;
    }
    else if(Deleteindx==5)
    {
        delete[] w;
        delete[] d;
        delete[] BufferId;
        delete[] ClasterInfo;
        delete[] ResultId;
    }
    else;
}

void CHohonon::Initialization(int NinputNode,int TSsample,int* Id)
{
    int i,itn;
    float rtn;

    N=NinputNode;
    NSample=TSsample;
    M=HNOutput;

    DeleteAll();
}

```

```

if(!(w=new float[N*M]))
{
    Message.Format("Kohonen - Memory allocation problem");
    AfxMessageBox(Message,MB_OK);
    return;
}
Deleteindx=1;
if(!(d=new float[M]))
{
    Message.Format("Kohonen - Memory allocation problem");
    AfxMessageBox(Message,MB_OK);
    return;
}
Deleteindx=2;

if!(BufferId=new int[NSample])
{
    Message.Format("Kohonen- Memory allocation problem");
    AfxMessageBox(Message,MB_OK);
    return;
}
Deleteindx=3;
if!(ClasterInfo=new int[M*Max_Group])
{
    Message.Format("Kohonen- Memory allocation problem");
    AfxMessageBox(Message,MB_OK);
    return;
}
Deleteindx=4;
strand( (unsigned)time( NULL ) );
itn=N*M;
rtn=1.0f*itn;
for(i=0;i<N*M;i++)
{
    w[i]=((rand()%itn)/rtn)-0.5f;
    if(w[i]>0.5 || w[i]<-0.5)
    {
        w[i]=0.5f;
    }
}
for(i=0;i<NSample;i++)
{
    BufferId[i]=ld[i];
}
i=0;
while(1)
{
    if((i*i)>M)
        break;
    else
        i++;
}
OW=i-1;
if((M%OW)!=0)
    OH=OW+1;
else
    OH=OW;
LRowCol=M%OW;
NE=Radius;
if(NE>=OW)
    NE=OW;
if(NE<0)
    NE=0;
if((NE%2)!=0)
{
    NE=NE/2+1;
}
else
{

```

```

        NE=NE/2;
    }

    RadChange=Max_Iter/(NE+1);
    Deta=float((1.0f/Max_Iter)*(HETA-0.1));

    for(i=0;i<M*Max_Group;i++)
    {
        ClasterInfo[i]=0;
    }

}

void CHohonen::Train(int* x)
{
    int i,j,k,loc,Start,End,p;
    float sum,sum1,min;

    for(k=0;k<NSample;k++)
    {
        min=9999999.0f;
        loc=-1;
        for(i=0;i<M;i++)
        {
            sum=0.0f;
            for(j=0;j<N;j++)
            {
                sum1=(float)fabs(w[i*N+j]-x[k*N+j]);
                sum+=(sum1 * sum1);
            }
            d[i]=sum;
            if(min>sum)
            {
                min=sum;
                loc=i;
            }
        }

        Start=loc-NE;
        End=Start+2*NE+1;
        for(i=Start;i<End;i++)
        {
            if(i>=0 && i<M)
            {
                for(p=0;p<N;p++)
                {
                    w[i*N+p]+=(eta*(x[k*N+p]-w[i*N+p]));
                }
            }
        }
    }
    eta=Deta;
    if(CountRad>=RadChange)
    {
        if(NE>0)
        {
            NE-=1;
        }
        CountRad=0;
    }
    else
    {
        CountRad++;
    }
}

void CHohonen::ClusterTest(int* x)
{
    int k,i,j,loc,indx;
    float sum,sum1,min;
}

```

```

        for(i=0;i<M*Max_Group;i++)
        {
            ClasterInfo[i]=0;
        }
        TrainAgain=0;
        k=0;
        while(k<NSample)
        {
            min=999999.0;
            loc=-1;
            for(i=0;i<M;i++)
            {
                sum=0.0f;
                for(j=0;j<N;j++)
                {
                    sum1=(float)fabs(w[i*N+j]-x[k*N+j]);
                    sum+=(sum1 * sum1);
                }
                d[i]=(float)sum;
                if(fabs(sum)<0.00001)
                    sum=0.0f;
                if(min>=sum)
                {
                    min=sum;
                    loc=i;
                }
            }
            idx=ClasterInfo[loc*Max_Group];
            if(idx>(Max_Group-2))
            {
                idx=0;
                Message.Format("Kohonen- More than %d pattern in one group (reset the group)",(Max_Group-1));
                AfxMessageBox(Message,MB_OK);
            }
            idx++;
            ClasterInfo[loc*Max_Group+idx]=BufferId[k];
            ClasterInfo[loc*Max_Group]=idx;
            if(idx>1)
            {
                TrainAgain=1;
                break;
            }
            k++;
        }
    }

void CHohonen::RWKoh(char* weightname,int flag)
{
    int i,lp,j,num;
    float fp;
    if(flag==0)
    {
        if(stream=fopen(weightname,"w"))
        {
            fprintf(stream,"%d %d %d\n",N,M,Max_Group);
            for(i=0;i<M*N;i++)
            {
                fprintf(stream,"%f ",w[i]);
            }
            for(i=0;i<M;i++)
            {
                lp=ClasterInfo[i*Max_Group];
                fprintf(stream,"%d ",lp);
                for(j=1;j<=lp;j++)
                {
                    fprintf(stream,"%d ",ClasterInfo[i*Max_Group+j]);
                }
            }
            fclose(stream);
        }
    }
}

```

```

        }
        else
        {
            Message.Format("Kohonen - Cannot write into %s file",weightname);
            AfxMessageBox(Message,MB_OK);
            return;
        }
    }
    else if(flag==1)
    {
        if(stream=fopen(weightname,"r"))
        {
            rewind(stream);
            fscanf(stream,"%d %d %d",&N,&M,&Max_G);
            for(i=0;i<M*N;i++)
            {
                fscanf(stream,"%f",&fp);
                w[i]=fp;
            }
            for(i=0;i<M;i++)
            {
                fscanf(stream,"%d",&Ip);
                num=Ip;
                ClasterInfo[i*Max_G]=Ip;
                for(j=1;j<=num;j++)
                {
                    fscanf(stream,"%d",&Ip);
                    ClasterInfo[i*Max_G+j]=Ip;
                }
            }
            fclose(stream);
        }
        else
        {
            Message.Format("Kohonen - %s file does not exist",weightname);
            AfxMessageBox(Message,MB_OK);
            return;
        }
    }
    else
    {
        Message.Format("Kohonen - file read write flag number(%d) does not exist",flag);
        AfxMessageBox(Message,MB_OK);
        return;
    }
}

void CHohonen::TestInitial(int NSamp,char* weightname)
{
    NSample=NSamp;
    DeleteAll();
    if(stream=fopen(weightname,"r"))
    {
        rewind(stream);
        fscanf(stream,"%d %d %d",&N,&M,&Max_G);
        fclose(stream);
    }
    else
    {
        Message.Format("Kohonen - file %s does not exist",weightname);
        AfxMessageBox(Message,MB_OK);
        return;
    }

    if(!(w=new float[N*M]))
    {
        Message.Format("Kohonen - Memory allocation problem");
        AfxMessageBox(Message,MB_OK);
        return;
    }
}

```

```

Deleteindx=1;
if(!(d=new float[M]))
{
    Message.Format("Kohonen - Memory allocation problem");
    AfxMessageBox(Message,MB_OK);
    return;
}
Deleteindx=2;
if(!(BufferId=new int[Nsample]))
{
    Message.Format("Kohonen - Memory allocation problem");
    AfxMessageBox(Message,MB_OK);
    return;
}
Deleteindx=3;
if(!(ClusterInfo=new int[M*Max_G]))
{
    Message.Format("Kohonen - Memory allocation problem");
    AfxMessageBox(Message,MB_OK);
    return;
}
Deleteindx=4;
if(!(ResultId=new int[Nsample]))
{
    Message.Format("Kohonen - Memory allocation problem");
    AfxMessageBox(Message,MB_OK);
    return;
}
Deleteindx=5;
}

void CHohonen::Test(int* x)
{
    int k,i,j,loc;
    float sum,sum1,min;

    for(k=0;k<Nsample;k++)
    {
        min=999999.0;
        loc=-1;
        for(i=0;i<M;i++)
        {
            sum=0.0f;
            for(j=0;j<N;j++)
            {
                sum1=(float)fabs(w[i*N+j]-x[k*N+j]);
                sum+=(sum1 * sum1);
            }
            d[i]=(float)sum;
            if(fabs(sum)<0.00001)
                sum=0.0f;
            if(min>=sum)
            {
                min=sum;
                loc=i;
            }
        }
        ResultId[k]=loc;
    }
}

```

**Appendix B:**

**Source Code for the Fast training algorithm and the Delta  
Rule training algorithms**

```

// stdafx.h : include file for standard system include files,
// or project specific include files that are used frequently, but
// are changed infrequently
//

#ifndef AFX_STDAFX_H__4517CFE6_B742_11D4_A34F_0000C0825656_INCLUDED_
#define AFX_STDAFX_H__4517CFE6_B742_11D4_A34F_0000C0825656_INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#define VC_EXTRALEAN           // Exclude rarely-used stuff from Windows headers

#include <afxwin.h>      // MFC core and standard components
#include <afxext.h>        // MFC extensions
#include <afxdisp.h>        // MFC OLE automation classes
#include <math.h>
#ifndef _AFX_NO_AFXCMN_SUPPORT
#include <afxcmn.h>        // MFC support for Windows Common Controls
#endif // _AFX_NO_AFXCMN_SUPPORT

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_STDAFX_H__4517CFE6_B742_11D4_A34F_0000C0825656_INCLUDED_)

```

```

// stdafx.cpp : source file that includes just the standard includes
// FTAAlgorithm.pch will be the pre-compiled header
// stdafx.obj will contain the pre-compiled type information

#include "stdafx.h"

```

```

//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.

// Used by FTAlgorithm.rc
//

#define IDD_ABOUTBOX      100
#define IDR_MAINFRAME     128
#define IDR_FTALGOTYPE    129
#define IDD_DIALOG1        130
#define IDD_DIALOG2        131
#define IDD_DIALOG3        132
#define IDD_DIALOG4        134
#define IDC_Train          1000
#define IDC_Test            1001
#define IDC_SName           1002
#define IDC_WName           1003
#define IDC_FName           1004
#define IDC_IWidth          1005
#define IDC_IHeight         1006
#define IDC_IId              1007
#define IDC_TFile            1008
#define IDC_SResult          1009
#define IDC_PWeights         1010
#define IDC_SWeights         1012
#define ID_FTA               32772
#define ID_Save              32773
#define ID_BPA               32774
#define ID_DTA               32775
#define ID_ExtractData       32776

// Next default values for new objects
//

#ifndef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_3D_CONTROLS      1
#define _APS_NEXT_RESOURCE_VALUE 135
#define _APS_NEXT_COMMAND_VALUE 32777
#define _APS_NEXT_CONTROL_VALUE 1013
#define _APS_NEXT_SYMED_VALUE   101
#endif
#endif

```

```

// MainFrm.h : interface of the CMainFrame class
//
///////////////////////////////
#ifndef !defined(AFX_MAINFRM_H__4517CFE8_B742_11D4_A34F_0000C0825656_INCLUDED_)
#define AFX_MAINFRM_H__4517CFE8_B742_11D4_A34F_0000C0825656_INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

class CMainFrame : public CFrameWnd
{
protected: // create from serialization only
    CMainFrame();
    DECLARE_DYNCREATE(CMainFrame)

// Attributes
public:

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CMainFrame)
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    //}}AFX_VIRTUAL

// Implementation
public:
    virtual ~CMainFrame();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected: // control bar embedded members
    CStatusBar m_wndStatusBar;
    CToolBar m_wndToolBar;

// Generated message map functions
protected:
    //{{AFX_MSG(CMainFrame)
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
        // NOTE - the ClassWizard will add and remove member functions here.
        // DO NOT EDIT what you see in these blocks of generated code!
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

/////////////////////////////
//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_MAINFRM_H__4517CFE8_B742_11D4_A34F_0000C0825656_INCLUDED_)

```

```

// MainFrm.cpp : implementation of the CMainFrame class
//

#include "stdafx.h"
#include "FTAlgorithm.h"

#include "MainFrm.h"

#ifndef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

///////////
// CMainFrame

IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
    //{{AFX_MSG_MAP(CMainFrame)
        // NOTE - the ClassWizard will add and remove mapping macros here.
        // DO NOT EDIT what you see in these blocks of generated code !
    ON_WM_CREATE()
   //}}AFX_MSG_MAP
END_MESSAGE_MAP()

static UINT indicators[] =
{
    ID_SEPARATOR,           // status line indicator
    ID_INDICATOR_CAPS,
    ID_INDICATOR_NUM,
    ID_INDICATOR_SCRL,
};

///////////
// CMainFrame construction/destruction

CMainFrame::CMainFrame()
{
    // TODO: add member initialization code here
}

CMainFrame::~CMainFrame()
{
}

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
        return -1;

    if (!m_wndToolBar.Create(this) ||
        !m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
    {
        TRACE0("Failed to create toolbar\n");
        return -1;    // fail to create
    }

    if (!m_wndStatusBar.Create(this) ||
        !m_wndStatusBar.SetIndicators(indicators,
            sizeof(indicators)/sizeof(UINT)))
    {
        TRACE0("Failed to create status bar\n");
        return -1;    // fail to create
    }
}

```

```

// TODO: Remove this if you don't want tool tips or a resizable toolbar
m_wndToolBar.SetBarStyle(m_wndToolBar.GetBarStyle() |
    CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_DYNAMIC);

// TODO: Delete these three lines if you don't want the toolbar to
// be dockable
m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
EnableDocking(CBRS_ALIGN_ANY);
DockControlBar(&m_wndToolBar);

return 0;
}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Modify the Window class or styles here by modifying
    // the CREATESTRUCT cs

    return CFrameWnd::PreCreateWindow(cs);
}

///////////////////////////////
// CMainFrame diagnostics

#ifndef _DEBUG
void CMainFrame::AssertValid() const
{
    CFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
    CFrameWnd::Dump(dc);
}

#endif //_DEBUG

///////////////////////////////
// CMainFrame message handlers

```

```

// FTAlgorithm.h : main header file for the FTALGORITHM
// application
//

#ifndef AFX_FTALGORITHM_H_4517CFE4_B742_11D4_A34F_0000C0825656_INCLUDED_
#define AFX_FTALGORITHM_H_4517CFE4_B742_11D4_A34F_0000C0825656_INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

#ifndef __AFXWIN_H__
#error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h" // main symbols

///////////////////////////////
// CFTAlgorithmApp:
// See FTAlgorithm.cpp for the implementation of this class
//

class CFTAlgorithmApp : public CWinApp
{
public:
    CFTAlgorithmApp();

// Overrides
// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(CFTAlgorithmApp)
public:
    virtual BOOL InitInstance();
//}}AFX_VIRTUAL

// Implementation

//{{AFX_MSG(CFTAlgorithmApp)
afx_msg void OnAppAbout();
// NOTE - the ClassWizard will add and remove member functions here.
// DO NOT EDIT what you see in these blocks of generated code !
//}}AFX_MSG
DECLARE_MESSAGE_MAP()
};

///////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_FTALGORITHM_H_4517CFE4_B742_11D4_A34F_0000C0825656_INCLUDED_)

```

```

// FTAlgorithm.cpp : Defines the class behaviors for the
// application.
//
#include "stdafx.h"
#include "FTAlgorithm.h"
#include "MainFrm.h"
#include "FTAlgorithmDoc.h"
#include "FTAlgorithmView.h"
#ifndef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

///////////////////////////////
// CFTAlgorithmApp

BEGIN_MESSAGE_MAP(CFTAlgorithmApp, CWinApp)
    //{{AFX_MSG_MAP(CFTAlgorithmApp)
    ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
        // NOTE - the ClassWizard will add and remove mapping macros here.
        // DO NOT EDIT what you see in these blocks of generated code!
    //}}AFX_MSG_MAP
    // Standard file based document commands
    ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
    ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
END_MESSAGE_MAP()

///////////////////////////////
// CFTAlgorithmApp construction

CFTAlgorithmApp::CFTAlgorithmApp()
{
    // TODO: add construction code here,
    // Place all significant initialization in InitInstance
}

///////////////////////////////
// The one and only CFTAlgorithmApp object
CFTAlgorithmApp theApp;
///////////////////////////////
// CFTAlgorithmApp initialization

BOOL CFTAlgorithmApp::InitInstance()
{
    AfxEnableControlContainer();

    // Standard initialization
    // If you are not using these features and wish to reduce the size
    // of your final executable, you should remove from the following
    // the specific initialization routines you do not need.

#ifdef _AFXDLL
    Enable3dControls();           // Call this when using MFC in a shared DLL
#else
    Enable3dControlsStatic();     // Call this when linking to MFC statically
#endif

    // Change the registry key under which our settings are stored.
    // You should modify this string to be something appropriate
    // such as the name of your company or organization.
    SetRegistryKey(_T("Local AppWizard-Generated Applications"));
    LoadStdProfileSettings(); // Load standard INI file options (including MRU)
    // Register the application's document templates. Document templates
    // serve as the connection between documents, frame windows and views.
}

```

```

CSingleDocTemplate* pDocTemplate;
pDocTemplate = new CSingleDocTemplate(
    IDR_MAINFRAME,
    RUNTIME_CLASS(CFTAlgorithmDoc),
    RUNTIME_CLASS(CMainFrame),           // main SDI frame window
    RUNTIME_CLASS(CFTAlgorithmView));
AddDocTemplate(pDocTemplate);
// Parse command line for standard shell commands, DDE, file open
CCmdLineInfo cmdInfo;
ParseCommandLine(cmdInfo);
// Dispatch commands specified on the command line
if (!ProcessShellCommand(cmdInfo))
    return FALSE;
// The one and only window has been initialized, so show and update it.
m_pMainWnd->ShowWindow(SW_SHOW);
m_pMainWnd->UpdateWindow();

return TRUE;
}

///////////////////////////////
// CAboutDlg dialog used for App About
class CAboutDlg : public CDialog
{
public:
    CAboutDlg();
// Dialog Data
    //{{AFX_DATA(CAboutDlg)
    enum { IDD = IDD_ABOUTBOX };
    //}}AFX_DATA

    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CAboutDlg)
protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    //}}AFX_VIRTUAL
// Implementation
protected:
    //{{AFX_MSG(CAboutDlg)
        // No message handlers
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
    //{{AFX_DATA_INIT(CAboutDlg)
    //}}AFX_DATA_INIT
}
void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CAboutDlg)
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
    //{{AFX_MSG_MAP(CAboutDlg)
        // No message handlers
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

// App command to run the dialog
void CFTAlgorithmApp::OnAppAbout()
{
    CAboutDlg aboutDlg;
    aboutDlg.DoModal();
}

///////////////////////////////
// CFTAlgorithmApp commands

```

```

// FTAlgorithmDoc.h : interface of the CFTAlgorithmDoc class
//
////////////////////

#ifndef AFX_FTALGORITHMDOC_H_4517C7EA_B742_11D4_A34F_0000C0825656_INCLUDED_
#define AFX_FTALGORITHMDOC_H_4517C7EA_B742_11D4_A34F_0000C0825656_INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

class CFTAlgorithmDoc : public CDocument
{
protected: // create from serialization only
    CFTAlgorithmDoc();
    DECLARE_DYNCREATE(CFTAlgorithmDoc)

// Attributes
private:
    CBitmap m_bitmap;
public:

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CFTAlgorithmDoc)
public:
    virtual BOOL OnOpenDocument (LPCTSTR);
    virtual void DeleteContents();
    virtual BOOL OnNewDocument();
    virtual void Serialize(CArchive& ar);
//}}AFX_VIRTUAL

// Implementation
public:
    virtual ~CFTAlgorithmDoc();
    CBitmap* GetBitmap();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
    //{{AFX_MSG(CFTAlgorithmDoc)
        // NOTE - the ClassWizard will add and remove member functions here.
        // DO NOT EDIT what you see in these blocks of generated code !
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_FTALGORITHMDOC_H_4517C7EA_B742_11D4_A34F_0000C0825656_INCLUDED_)

```

```

// FTAlgorithmDoc.cpp : implementation of the
// CFTAlgorithmDoc class
//

#include "stdafx.h"
#include "FTAlgorithm.h"

#include "FTAlgorithmDoc.h"

#ifndef _DEBUG
#define new DEBUG_NEW
#endif THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

///////////////////////////////
// CFTAlgorithmDoc

IMPLEMENT_DYNCREATE(CFTAlgorithmDoc, CDocument)

BEGIN_MESSAGE_MAP(CFTAlgorithmDoc, CDocument)
    //{{AFX_MSG_MAP(CFTAlgorithmDoc)
        // NOTE - the ClassWizard will add and remove mapping macros here.
        // DO NOT EDIT what you see in these blocks of generated code!
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

///////////////////////////////
// CFTAlgorithmDoc construction/destruction

CFTAlgorithmDoc::CFTAlgorithmDoc()
{
    // TODO: add one-time construction code here
}

CFTAlgorithmDoc::~CFTAlgorithmDoc()
{
}

BOOL CFTAlgorithmDoc::OnNewDocument()
{
    if (!CDocument::OnNewDocument())
        return FALSE;

    // TODO: add reinitialization code here
    // (SDI documents will reuse this document)

    return TRUE;
}

///////////////////////////////
// CFTAlgorithmDoc serialization

void CFTAlgorithmDoc::Serialize(CArchive& ar)
{
    if (ar.IsStoring())
    {
        // TODO: add storing code here
    }
    else
    {
        CString strFileName = ar.GetFile()->GetFilePath();
        HBITMAP hBitmap = (HBITMAP)::LoadImage(AfxGetInstanceHandle(),
            (LPCTSTR)strFileName, IMAGE_BITMAP, 0, 0, LR_LOADFROMFILE |
            LR_CREATEDIBSECTION);
    }
}

```

```

        if(hBitmap == NULL)
            AfxThrowArchiveException (CArchiveException::badIndex);

        m_bitmap.Attach (hBitmap);
        // TODO: add loading code here
    }

///////////////////////////////
// CFTAlgorithmDoc diagnostics

#ifndef _DEBUG
void CFTAlgorithmDoc::AssertValid() const
{
    CDocument::AssertValid();
}

void CFTAlgorithmDoc::Dump(CDumpContext& dc) const
{
    CDocument::Dump(dc);
}
#endif // _DEBUG

///////////////////////////////
// CFTAlgorithmDoc commands

BOOL CFTAlgorithmDoc::OnOpenDocument(LPCTSTR lpszPathName)
{
    if(!(CDocument::OnOpenDocument (lpszPathName)))
    {
        POSITION pos = GetFirstViewPosition();
        CView* pView = GetNextView (pos);
        pView->OnInitialUpdate();
        return FALSE;
    }
    return TRUE;
}

void CFTAlgorithmDoc::DeleteContents()
{
    if((HBITMAP) m_bitmap != NULL)
        m_bitmap.DeleteObject();
    CDocument::DeleteContents();
}

CBitmap* CFTAlgorithmDoc::GetBitmap()
{
    return ((HBITMAP) m_bitmap == NULL) ? NULL : &m_bitmap;
}

```

```

// FTAlgorithmView.h : interface of the CFTAlgorithmView
//class
//
////////////////////////////////////////////////////////////////////////

#ifndef AFX_FTALGORITHMVIEW_H_4517CFEC_B742_11D4_A34F_0000C0825656_INCLUDED_
#define AFX_FTALGORITHMVIEW_H_4517CFEC_B742_11D4_A34F_0000C0825656_INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

class CFTAlgorithmView : public CView
{
protected: // create from serialization only
    CFTAlgorithmView();
    DECLARE_DYNCREATE(CFTAlgorithmView)

// Attributes
public:
    CFTAlgorithmDoc* GetDocument();

// Operations
public:
    CString Time1;
    CTime startTime;
    CTime endTime;
    CTimeSpan elapsedTime;
    CString Message;
    int Width_Bytes;
    int Width;
    int Height;
    int BitsPerPixel;
    BYTE* Buffer;
    int* BitBuffer;
    float* BitBufferF;
    int ImageWidthPixel;
    int ImageHeightPixel;
    int WLow;
    int WHigh;
    FILE* stream;
    int* Id;
    int TotId;
    char fn2me[300];
    char WFname[300];
    int begin;
    float DMError;
    float DCError;
    int DMEIter;
    int DCEIter;
    int DMRecog;
    int DCRecog;
    int DTChar;

    void ReadTextFile(void);
    void ReadImage(void);
    void GetCharacterArea(void);
    void DisplayTheCharacter(CDC* );
    void ConverBitToByte(int,int);
    void StoreImageInfo(int ,int ,int ,CString);
    void ReadImageInfo(void);
    int RWweight(int*,int,int);

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CFTAlgorithmView)
public:

```

```

virtual void OnDraw(CDC* pDC); // overridden to draw this view
virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
protected:
//}}AFX_VIRTUAL

// Implementation
public:
    virtual ~CFTAlgorithmView();
#ifndef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
    //{{AFX_MSG(CFTAlgorithmView)
    afx_msg void OnFta();
    afx_msg void OnISave();
    afx_msg void OnBpa();
    afx_msg void OnDta();
    afx_msg void OnExtractData();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

#ifndef _DEBUG // debug version in FTAlgorithmView.cpp
inline CFTAlgorithmDoc* CFTAlgorithmView::GetDocument()
    { return (CFTAlgorithmDoc*)m_pDocument; }
#endif

///////////
//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_FTALGORITHMVIEW_H__4517CFEC_B742_11D4_A34F_0000C0825656__INCLUDED_)

```

```

// FTAlgorithmView.cpp : implementation of the
// CFTAlgorithmView class
//

#include "stdafx.h"
#include "FTAlgorithm.h"

#include "FTAlgorithmDoc.h"
#include "FTAlgorithmView.h"
#include "FTADlg2.h"
#include "FTADlg.h"
#include "FastTA.h"
#include "BmpInfo.h"
#include "Delta.h"
#include "ImageTextInfo.h"
#include "ExtractImageInfo.h"

#ifndef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

///////////
// CFTAlgorithmView

IMPLEMENT_DYNCREATE(CFTAlgorithmView, CView)

BEGIN_MESSAGE_MAP(CFTAlgorithmView, CView)
    //{{AFX_MSG_MAP(CFTAlgorithmView)
    ON_COMMAND(ID_FTA, OnFta)
    ON_COMMAND(ID_ISave, OnISave)
    ON_COMMAND(ID_BPA, OnBpa)
    ON_COMMAND(ID_DTA, OnDta)
    ON_COMMAND(ID_ExtractData, OnExtractData)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

///////////
// CFTAlgorithmView construction/destruction

CFTAlgorithmView::CFTAlgorithmView()
{
    begin=-1;
    DMError=99999.0;
    DCError=99999.0;
    DMEIter=0;
    DCEIter=0;
    DMRecog=0;
    DCRecog=0;
    DTChar=0;
    // TODO: add construction code here
}

CFTAlgorithmView::~CFTAlgorithmView()
{
}

BOOL CFTAlgorithmView::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Modify the Window class or styles here by modifying
    // the CREATESTRUCT cs

    return CView::PreCreateWindow(cs);
}

///////////

```

```

// CFTAlgorithmView drawing

void CFTAlgorithmView::OnDraw(CDC* pDC)
{
    CFTAlgorithmDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    CBitmap* pBitmap = pDoc->GetBitmap();

    if(pBitmap != NULL)
    {
        DIBSECTION ds;
        pBitmap->GetObject(sizeof(DIBSECTION),&ds);

        CDC memDC;
        memDC.CreateCompatibleDC (pDC);
        CBitmap* pOldBitmap= memDC.SelectObject(pBitmap);

        pDC->StretchBlt(0,0,200,300, &memDC,
                          0,0,ds.dsBm.bmWidth, ds.dsBm.bmHeight,SRCCOPY);

        memDC.SelectObject(pOldBitmap);
    }
}

// CFTAlgorithmView diagnostics

#ifndef _DEBUG
void CFTAlgorithmView::AssertValid() const
{
    CView::AssertValid();
}
#endif

void CFTAlgorithmView::Dump(CDumpContext& dc) const
{
    CView::Dump(dc);
}

CFTAlgorithmDoc* CFTAlgorithmView::GetDocument() // non-debug version is inline
{
    ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CFTAlgorithmDoc)));
    return (CFTAlgorithmDoc*)m_pDocument;
}

#endif // _DEBUG

// CFTAlgorithmView message handlers
void CFTAlgorithmView::OnLSave()
{
    CClientDC aDC(this);
    ReadImage();
    DisplayTheCharacter(&aDC);
}

void CFTAlgorithmView::ReadImage()
{
    BmpInfo bmpInfoDlg;
    if(bmpInfoDlg.DoModal()==IDOK)
    {
        int WthPixel,HhtPixel,Idd;
        CString filename;
        DWORD length;
        DWORD Image_Length;
        HhtPixel=bmpInfoDlg.m_lHeight;
        WthPixel=bmpInfoDlg.m_lWidth;
    }
}

```

```

Idd=bmpInfoDlg.m_Id;
filename=bmpInfoDlg.m_FileName;

CFTAAlgorithmDoc* pDoc = GetDocument();
ASSERT_VALID(pDoc);

CBitmap* pBitmap = pDoc->GetBitmap ();
if (pBitmap != NULL)
{
    DIBSECTION ds;
    pBitmap->GetObject (sizeof (DIBSECTION), &ds);

    Width_Bytes=ds.dsBm.bmWidth;
    Width_Bytes=Width_Bytes/8;
    if((ds.dsBm.bmWidth%8)!=0)   Width_Bytes+=1;
    if((Width_Bytes%2)!=0)       Width_Bytes+=1;
    Height=ds.dsBm.bmHeight;
    BitsPerPixel=ds.dsBm.bmBitsPixel;
    length=ds.dsBm.bmWidthBytes*ds.dsBm.bmHeight;
    Width=Width_Bytes;
    if((Buffer= new BYTE[length]) !=0)
    {
        Image_Length=(pBitmap->GetBitmapBits(length,Buffer));
        if((BitBuffer= new int[WthPixel*HhtPixel]) !=0)
        {
            ConverBitToByte(HhtPixel,WthPixel);
            StoreImageInfo(HhtPixel,WthPixel,Idd,filename);
            delete[] BitBuffer;
        }
        delete[] Buffer;
    }
}
}

void CFTAAlgorithmView::DisplayTheCharacter(CDC* pDC)
{

}

void CFTAAlgorithmView::ConverBitToByte(int HhtPixel,int WthPixel)
{
    int i,j,k,m=0,row;
    BYTE Data,Bt;
    m=0;
    for(i=0;i<Height;i++)
    {
        row=0;
        for(j=0;j<Width;j++)
        {
            Data=Buffer[i*Width+j];
            Bt=128;
            for(k=0;k<8;k++)
            {
                if(Data & Bt)
                    BitBuffer[m]=-1;
                else
                    BitBuffer[m]=1;
                m++;
                Bt=Bt>>1;
                row++;
                if(row==WthPixel)
                {
                    row=-1;
                    break;
                }
            }
        }
    }
}

```

```

        }
        if(row== -1)
            break;
    }
}

void CFTAlgorithmView::StoreImageInfo(int HhtPixel,int WthPixel,int Idd,CString filename)
{
    int i,j,TotCh,W,H,len;
    long int Tot;

    Tot=0;
    TotCh=0;

    strcpy(fname,filename);
    len=strlen(fname);
    if(!(fname[len-4]=='.'))
    {
        Message.Format("File name %s is not has extension",fname);
        AfxMessageBox(Message,MB_OK);
        return;
    }

    fname[len-3]='.';
    fname[len-2]='n';
    fname[len-1]='';

    if((stream=fopen(fname,"r")))
    {
        fscanf(stream,"%d",&TotCh);
        fscanf(stream,"%d",&H);
        fscanf(stream,"%d",&W);
        fclose(stream);
        if !(H==HhtPixel && W==WthPixel))
        {
            Message.Format("Image dimension does not match");
            AfxMessageBox(Message,MB_OK);
            return;
        }
    }
    Tot = TotCh* HhtPixel*WthPixel;
    Tot+= HhtPixel*WthPixel;
    TotCh++;
    if(stream=fopen(fname,"w"))
    {
        fprintf(stream,"%d %d %d \n",TotCh,HhtPixel,WthPixel);
        fclose(stream);
    }

    if(stream=fopen(filename,"a"))
    {
        fprintf(stream,"\n%d\n ",Idd);
        for(i=0;i<HhtPixel;i++)
        {
            for(j=0;j<WthPixel;j++)
            {
                fprintf(stream,"%d ",BitBuffer[i*WthPixel+j]);
            }
            fprintf(stream,"\n");
        }
        fclose(stream);
    }
}

void CFTAlgorithmView::ReadImageInfo()
{
    int i,j,TotCh,W,H,len,pL,indexId,indexB;
}

```

```

float pF;
char temp[300];
long int Tot;

Tot=0;
TotCh=0;
strcpy(temp, fname);
len=strlen(fname);
if(!(fname[len-4]=='.'))
{
    Message.Format("File name %s' is not has extension", fname);
    AfxMessageBox(Message, MB_OK);
    return;
}

fname[len-3]='.';
fname[len-2]='n';
fname[len-1]='t';
if((stream=fopen(fname, "r")))
{
    fscanf(stream, "%d", &TotCh);
    fscanf(stream, "%d", &H);
    fscanf(stream, "%d", &W);
    ImageWidthPixel=W;
    ImageHeightPixel=H;
    fclose(stream);
}
else
{
    Message.Format("File %s not found", fname);
    AfxMessageBox(Message, MB_OK);
    return;
}
TotId=TotCh//Tot/(W*H);
Tot=TotCh*(W*H)//TotCh * W * H;
if(! (Id= new int [TotId]))
{
    Message.Format("Memory allocation problem");
    AfxMessageBox(Message, MB_OK);
    return;
}
if!(BitBufferF= new float[Tot])
{
    Message.Format("Memory allocation problem");
    AfxMessageBox(Message, MB_OK);
    return;
}

strcpy(fname,temp);
if(stream=fopen(fname, "r"))
{
    indexId=0;
    indexB=0;
    rewind(stream);
    for(i=0;i<TotId;i++)
    {
        fscanf(stream, "%d", &pI);
        Id[indexId++]=pI;
        for(j=0;j<(W*H);j++)
        {
            fscanf(stream, "%f", &pF);
            BitBufferF[indexB++]=pF;
        }
    }
    fclose(stream);
}
}

int CFTAlgorithmView::RWweight(int* w,int length,int RW)
{

```

```

int i,pv;
length*=length;
if(RW==0)
{
    if(stream=fopen(WFname,"w"))
    {
        fprintf(stream,"%d ",length);
        for(i=0;i<length;i++)
        {
            fprintf(stream,"%d ",w[i]);
        }
        fclose(stream);
    }
    else
    {
        Message.Format("File %s not found",WFname);
        AfxMessageBox(Message,MB_OK);
        return 0;
    }
}
else if(RW==1)
{
    if(stream=fopen(WFname,"r"))
    {
        rewind(stream);
        fscanf(stream,"%d",&pv);
        length=pv;
        for(i=0;i<length;i++)
        {
            fscanf(stream,"%d",&pv);
            w[i]=pv;
        }
        fclose(stream);
    }
    else
    {
        Message.Format("File %s not found",WFname);
        AfxMessageBox(Message,MB_OK);
        return 0;
    }
}
else if(RW==2)
{
    if(stream=fopen(WFname,"r"))
    {
        rewind(stream);
        fscanf(stream,"%d",&pv);
        length=pv;
    }
    else
    {
        Message.Format("File %s not found",WFname);
        AfxMessageBox(Message,MB_OK);
        return 0;
    }
}
else
{
    Message.Format("RW flag not equal to 0, 1, or 2");
    AfxMessageBox(Message,MB_OK);
    return 0;
}
return length;
}

*****

```

Fast Training Algorithm

```

*****/
```

```

void CFTAlgorithmView::OnFta()
{
    // TODO: Add your command handler code here
    int i,j,Ti;
    float pF;
    FILE* stream101;

    FTADlg DlgTrainTest;
    FTADlg2 DlgSWName;
    CFTA FTAlg;
    int L=4;
    int NL[4];
    CCClientDC aDC(this);

    if(DlgTrainTest.DoModal()==IDOK)
    {
        if(DlgSWName.DoModal()==IDOK)
        {
            NL[1]=17;
            NL[2]=21;
            NL[3]=8;
            strcpy(WFname,DlgSWName.m_WName);
            strcpy(fname,DlgSWName.m_SName);
            if(DlgTrainTest.m_Work==1)
            {
                ReadImageInfo();
                NL[0]=ImageWidthPixel*ImageHeightPixel;

                FTAlg.TrainInit(L,NL,TotId,BitBufferF,Id);
                for(i=1,j=0;i<L;i++)
                    j+=NL[i]*NL[i-1];
                if(DlgTrainTest.m_SWeight==0)
                {
                    FTAlg.ReadWrite(0,j,"c:\\Nicholas\\Data\\Weight
                        Info\\InitialMLPWeight.txt");
                    if(stream101=fopen("c:\\Nicholas\\Data\\Weight
                        Info\\InitialMLP_R.txt","w"))
                    {
                        for(i=0;i<j;i++)
                        {
                            fprintf(stream101,"%f ",FTAlg.R[i]);
                        }
                        fclose(stream101);
                    }
                    else
                        return;
                }
                else
                {
                    FTAlg.ReadWrite(1,j,"c:\\Nicholas\\Data\\Weight
                        Info\\InitialMLPWeight.txt");
                    if(stream101=fopen("c:\\Nicholas\\Data\\Weight Info\\InitialMLP_R.txt","r"))
                    {
                        for(i=0;i<j;i++)
                        {
                            fscanf(stream101,"%f",&pF);
                            FTAlg.R[i]=pF;
                        }
                        fclose(stream101);
                    }
                    else
                        return;
                }
                i=0;
                FTAlg.TotalE=99.0f;
                FTAlg.TotalR=0;
            }
        }
    }
}
```

```

DCError=FTAlg.TotalE;
DCRecog=FTAlg.TotalR;
DTChar=FTAlg.M;
strcpy(FTAlg.WFname,WFname);
startTime = CTime::GetCurrentTime();
if(stream=fopen("c:\\Nicholas\\Data\\Error\\FastTrainError.txt","w"))
{
    FTAlg.stream99=stream;
    FTAlg.Niter=FTAlg.M;
    while((FTAlg.TotalE>0.001) && (FTAlg.TotalR<FTAlg.M) && (i<50000))
    {
        FTAlg.FTATrain();
        DCError=FTAlg.TotalE;
        if(DMError>DCError)
        {
            DMError=DCError;
            DMEIter=i;
        }
        DCEIter=i;
        DCRecog=FTAlg.TotalR;
        if(DMRecog<DCRecog)
        {
            DMRecog=DCRecog;
        }
        i++;
        if(FTAlg.Niter<1)
            FTAlg.Niter=FTAlg.M;
    }
    fclose(stream);
}
Ti=i;
for(i=1,j=0;i<L;i++)
    j+=NL[i]*NL[i-1];

endTime = CTime::GetCurrentTime();
elapsedTime = (endTime - startTime);
Time1 =elapsedTime.Format( "Total days: %D, hours: %H, mins: %M, secs: %S" );
Message.Format("Trining is completed - Recognition %d out of %d, E = %f, i= %d Total
time = %s",FTAlg.TotalR,FTAlg.M,FTAlg.TotalE,Ti,Time1);
AfxMessageBox(Message,MB_OK);
delete[] Id;
delete[] BitBufferF;
}
else if(DlgTrainTest.m_Work==2)
{
    ReadImageInfo();
    NL[0]=ImageWidthPixel*ImageHeightPixel;
    startTime = CTime::GetCurrentTime();
    FTAlg.TrainInit(L,NL,TotId,BitBufferF,Id);
    for(i=1,j=0;i<L;i++)
        j+=NL[i]*NL[i-1];
    FTAlg.ReadWrite(1,j,WFname);
    FTAlg.Test();
    endTime = CTime::GetCurrentTime();
    elapsedTime = (endTime - startTime);
    Time1 =elapsedTime.Format( "Total days: %D, hours: %H, mins: %M, secs: %S" );
    if(stream=fopen("c:\\Nicholas\\Data\\Error\\FastTrainRecog.txt","w"))
    {
        fprintf(stream,"Tested character Recognized character result correct or
rejected or miss classified \n\n");
        for(i=0;i<TotId;i++)
        {
            fprintf(stream,"%d
%d
\n",FTAlg.RecogResult[i*3],FTAlg.RecogResult[i*3+1],
FTAlg.RecogResult[i*3+2]);
        }
    }
    fclose(stream);
}

```

```

        MessageFormat("Testing is completed - Recognized character = %d, Recognition %d
out of %d, Total time = %s",FTAlg.RecogResult[1],FTAlg.TotalR,
FTAlg.M,Time1);
AfxMessageBox(Message,MB_OK);
delete[] Id;
delete[] BitBufferF;
}
else
{
}
}
}

void CFTAlgorithmView::OnBpa()
{
}

void CFTAlgorithmView::OnDta()
{
// TODO: Add your command handler code here
int i,j,Ti,TM=1;
FILE* stream101;
float pF;
FTADlg DlgTrainTest;
FTADlg2 DlgSWName;
CDelta DTAAAlg;
CClientDC aDC(this);
int L=4;
int NL[4];

if(DlgTrainTest.DoModal()==IDOK)
{
    if(DlgSWName.DoModal()==IDOK)
    {
        NL[1]=17;
        NL[2]=21;
        NL[3]=8;
        strcpy(WFname,DlgSWName.m_WName);
        strcpy(fname,DlgSWName.m_SName);
        if(DlgTrainTest.m_Work==1)
        {
            ReadImageInfo();
            NL[0]=ImageWidthPixel*ImageHeightPixel;
            startTime = CTime::GetCurrentTime();
            DTAAAlg.Initial(TotId,L,BitBufferF,Id,NL);
            for(i=1,j=0;i<L;i++)
                j+=NL[i]*NL[i-1];
            if(DlgTrainTest.m_SWeight==1)
            {
                if(stream101=fopen("c:\\Nicholas\\Data\\Weight
Info\\InitialMLPWeight.txt","r"))
                {
                    for(i=0;i<j;i++)
                    {
                        fscanf(stream101,"%f",&pF);
                        DTAAAlg.x[i]=pF;
                    }
                    fclose(stream101);
                }
                else
                {
                    return;
                }
            }
        }
    }
}
}

```

```

if(stream=fopen("c:\\Nicholas\\Data\\Error\\DeltaError.txt","w"))
{
    i=0;
    DTAAlg.TotalE=99.0f;
    DTAAlg.TRecogn=0;
    DCError=DTAAlg.TotalE;
    DCRecog=DTAAlg.TRecogn;
    DTChar=DTAAlg.M;
    while((DTAAlg.TotalE>0.001) && (i<5000) &&
          (DTAAlg.TRecogn<DTAAlg.M))
    {
        DTAAlg.Train();
        fprintf(stream,"%d %d %f
                      \n",i,DTAAlg.TRecogn,DTAAlg.TotalE);

        DCError=DTAAlg.TotalE;
        if(DMError>DCError)
        {
            DMError=DCError;
            DMEIter=i;
        }
        DCEIter=i;
        DCRecog=DTAAlg.TRecogn;
        if(DMRecog<DCRecog)
        {
            DMRecog=DCRecog;
        }
        i++;
    }
    fclose(stream);
}
Ti=i;
for(i=1,j=0;i<L;i++)
{
    j+=NL[i]*NL[i-1];
    DTAAlg.ReadWrite(0,j,WFname);
    endTime = CTime::GetCurrentTime();
    elapsedTime = (endTime - startTime);
    Time1 = elapsedTime.Format( "Total days: %D, hours: %H, mins: %M, secs: %S" );
    Message.Format("Trining is completed - Recognition %d out of %d, E = %f, i= %d,
                    Total Time = %s",DTAAlg.TRecogn,DTAAlg.M,DTAAlg.TotalE,Ti,Time1);
    AfxMessageBox(Message,MB_OK);
    delete[] Id;
    delete[] BitBufferF;
}
else if(DlgTrainTest.m_Work==2)
{
    ReadImageInfo();
    NL[0]=ImageWidthPixel*ImageHeightPixel;
    startTime = CTime::GetCurrentTime();
    DTAAlg.Initial(TotId,L,BitBufferF,Id,NL);
    for(i=1,j=0;i<L;i++)
    {
        j+=NL[i]*NL[i-1];
        DTAAlg.ReadWrite(1,j,WFname);
        DTAAlg.TotalE=DTAAlg.fdelta();
        endTime = CTime::GetCurrentTime();
        elapsedTime = (endTime - startTime);
        Time1 = elapsedTime.Format( "Total days: %D, hours: %H, mins: %M, secs: %S" );
        if(stream=fopen("c:\\Nicholas\\Data\\Error\\DeltaRecog.txt","w"))
        {
            fprintf(stream,"Tested character Recognized character result correct or
                           rejected or miss classified \n\n");
            for(i=0;i<TotId;i++)
            {
                fprintf(stream,"      %d      %d
                               \n",DTAAlg.RecogResult[i*3],DTAAlg.RecogResult[i*3+1],
                               DTAAlg.RecogResult[i*3+2]);
            }
            fclose(stream);
        }
    }
}

```

```

        Message.Format("Testing is completed - Recognized character %d, Recognition %d out
of %d, E = %f, Total Time = %s",DTAAlg.RecogResult[1],
DTAAlg.TRecogn,DTAAlg.M,DTAAlg.TotalE,Time1);
AfxMessageBox(Message,MB_OK);
delete[] Id;
delete[] BitBufferF;
}
else
{
}
}
}

void CFTAlgorithmView::OnExtractData()
{
    // TODO: Add your command handler code here
    BYTE temp;

    // BitBuffer
    ImageTextInfo ImgDlg;
    CExtractImageInfo EInfo;

    if(ImgDlg.DoModal()==IDOK)
    {
        DWORD length;
        DWORD Image_Length;

        CFTAlgorithmDoc* pDoc = GetDocument();
        ASSERT_VALID(pDoc);

        CBitmap* pBitmap = pDoc->GetBitmap ();
        if(pBitmap != NULL)
        {
            DIBSECTION ds;
            pBitmap->GetObject (sizeof (DIBSECTION), &ds);

            Width_Bytes=ds.dsBm.bmWidth;
            Width_Bytes=Width_Bytes/8;
            if((ds.dsBm.bmWidth%8)!=0)   Width_Bytes+=1;
            if((Width_Bytes%2)!=0)       Width_Bytes+=1;
            Height=ds.dsBm.bmHeight;
            BitsPerPixel=ds.dsBm.bmBitsPixel;
            length=ds.dsBm.bmWidthBytes*ds.dsBm.bmHeight;
            Width=Width_Bytes;

            //BYTE An 8-bit integer that is not signed.
            if((Buffer= new BYTE[length]) !=0)
            {
                Image_Length=(pBitmap->GetBitmapBits(length,Buffer));
                temp=255;
                for(DWORD k=0;k<length;k++)
                {
                    Buffer[k]=Buffer[k] ^ temp;
                }
                EInfo.Initial(Buffer,Width_Bytes,Width,Height,length,
                            ImgDlg.m_StoreFName,ImgDlg.m_TextFile);
                EInfo.ReadTextFile();
                EInfo.ReadImage();
                delete[] Buffer;
            }
        }
    }
}
}

```

```

// ImageTextInfo.h : header file
#ifndef AFX_IMAGETEXTINFO_H__2CDD88E0_BE89_11D4_80AA_F26EABE46405__INCLUDED_
#define AFX_IMAGETEXTINFO_H__2CDD88E0_BE89_11D4_80AA_F26EABE46405__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// ImageTextInfo.h : header file
//

///////////////////////////////
// ImageTextInfo dialog

class ImageTextInfo : public CDialog
{
// Construction
public:
    ImageTextInfo(CWnd* pParent = NULL); // standard constructor

// Dialog Data
    //{{AFX_DATA(ImageTextInfo)
    enum { IDD = IDD_DIALOG4 };
    CString m_StoreFName;
    CString m_TextFile;
    //}}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(ImageTextInfo)
protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
    //{{AFX_MSG(ImageTextInfo)
        // NOTE: the ClassWizard will add member functions here
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_IMAGETEXTINFO_H__2CDD88E0_BE89_11D4_80AA_F26EABE46405__INCLUDED_)

```

```

// ImageTextInfo.cpp : implementation file
//

#include "stdafx.h"
#include "FTAAlgorithm.h"
#include "ImageTextInfo.h"

#ifndef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

///////////
// ImageTextInfo dialog

ImageTextInfo::ImageTextInfo(CWnd* pParent /*=NULL*/)
    : CDialog(ImageTextInfo::IDD, pParent)
{
    //{{AFX_DATA_INIT(ImageTextInfo)
    m_StoreFName = _T("c:\\Nicholas\\Data\\Image Info\\Mathy.txt");
    m_StoreFName = _T("c:\\Nicholas\\Training Set\\T21.txt");
    m_TextFile = _T("c:\\Nicholas\\Training Set\\T1.txt");
    //}}AFX_DATA_INIT
}

void ImageTextInfo::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(ImageTextInfo)
    DDX_Text(pDX, IDC_SResult, m_StoreFName);
    DDX_Text(pDX, IDC_TFile, m_TextFile);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(ImageTextInfo, CDialog)
    //{{AFX_MSG_MAP(ImageTextInfo)
        // NOTE: the ClassWizard will add message map macros here
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

///////////
// ImageTextInfo message handlers

```

```

// FTADlg.h : header file
#ifndef AFX_FTADLG_H__25C3C380_B745_11D4_A34F_0000C0825656_INCLUDED_
#define AFX_FTADLG_H__25C3C380_B745_11D4_A34F_0000C0825656_INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// FTADlg.h : header file
//

///////////////////////////////
// FTADlg dialog

class FTADlg : public CDialog
{
// Construction
public:
    int         m_Work;
    int m_PWeight;
    int m_SWeight;
    FTADlg(CWnd* pParent = NULL); // standard constructor

// Dialog Data
    //{{AFX_DATA(FTADlg)
    enum { IDD = IDD_DIALOG1 };
        // NOTE: the ClassWizard will add data members here
    //}}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(FTADlg)
protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
    //{{AFX_MSG(FTADlg)
    afx_msg void OnTest();
    afx_msg void OnTrain();
    afx_msg void OnPWeights();
    afx_msg void OnSWeights();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_FTADLG_H__25C3C380_B745_11D4_A34F_0000C0825656_INCLUDED_)

```

```

// FTADlg.cpp : implementation file
//

#include "stdafx.h"
#include "FTAAlgorithm.h"
#include "FTADlg.h"

#ifndef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

///////////
// FTADlg dialog

FTADlg::FTADlg(CWnd* pParent /*=NULL*/)
    : CDialog(FTADlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(FTADlg)
        // NOTE: the ClassWizard will add member initialization here
    m_Work=0;
    m_PWeight=0;
    m_SWeight=0;
    //}}AFX_DATA_INIT
}

void FTADlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(FTADlg)
        // NOTE: the ClassWizard will add DDX and DDV calls here
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(FTADlg, CDialog)
    //{{AFX_MSG_MAP(FTADlg)
    ON_BN_CLICKED(IDC_Test, OnTest)
    ON_BN_CLICKED(IDC_Train, OnTrain)
    ON_BN_CLICKED(IDC_PWeights, OnPWeights)
    ON_BN_CLICKED(IDC_SWeights, OnSWeights)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

///////////
// FTADlg message handlers

void FTADlg::OnTest()
{
    // TODO: Add your control notification handler code here
    m_Work=2;
}

void FTADlg::OnTrain()
{
    // TODO: Add your control notification handler code here
    m_Work=1;
}

void FTADlg::OnPWeights()
{
    // TODO: Add your control notification handler code here
    if(m_PWeight==0)
        m_PWeight=1;
}

```

```
    else
        m_PWeight=0;
}

void FTADlg::OnSWeights()
{
    // TODO: Add your control notification handler code here
    if(m_SWeight==0)
        m_SWeight=1;
    else
        m_SWeight=0;
}
```

```

// FTADlg2.h : header file
#ifndef AFX_FTADLG2_H__25C3C381_B745_11D4_A34F_0000C0825656_INCLUDED_
#define AFX_FTADLG2_H__25C3C381_B745_11D4_A34F_0000C0825656_INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// FTADlg2.h : header file
//

///////////////////////////////
// FTADlg2 dialog

class FTADlg2 : public CDialog
{
// Construction
public:
    FTADlg2(CWnd* pParent = NULL); // standard constructor

// Dialog Data
    //{{AFX_DATA(FTADlg2)
    enum { IDD = IDD_DIALOG2 };
    CString m_SName;
    CString m_WName;
    //}}AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(FTADlg2)
protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
    //{{AFX_MSG(FTADlg2)
        // NOTE: the ClassWizard will add member functions here
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_FTADLG2_H__25C3C381_B745_11D4_A34F_0000C0825656_INCLUDED_)

```

```

// FTADlg2.cpp : implementation file
//

#include "stdafx.h"
#include "FTAAlgorithm.h"
#include "FTADlg2.h"

#ifndef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

///////////
// FTADlg2 dialog

FTADlg2::FTADlg2(CWnd* pParent /*=NULL*/)
    : CDialog(FTADlg2::IDD, pParent)
{
    //{{AFX_DATA_INIT(FTADlg2)
    m_SName = _T("c:\\Nicholas\\Data\\Image Info\\Mathy.txt");
    m_WName = _T("c:\\Nicholas\\Data\\Weight Info\\MLPWeight.txt");
    //}}AFX_DATA_INIT
}

void FTADlg2::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(FTADlg2)
    DDX_Text(pDX, IDC_SName, m_SName);
    DDX_Text(pDX, IDC_WName, m_WName);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(FTADlg2, CDialog)
    //{{AFX_MSG_MAP(FTADlg2)
        // NOTE: the ClassWizard will add message map macros here
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

///////////
// FTADlg2 message handlers

```

```

// BmpInfo.h : header file
#ifndef AFX_BMPINFO_H__25C3C382_B745_11D4_A34F_0000C0825656__INCLUDED_
#define AFX_BMPINFO_H__25C3C382_B745_11D4_A34F_0000C0825656__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// BmpInfo.h : header file
//

///////////////////////////////
// BmpInfo dialog

class BmpInfo : public CDialog
{
// Construction
public:
    BmpInfo(CWnd* pParent = NULL); // standard constructor

// Dialog Data
//{{AFX_DATA(BmpInfo)
enum { IDD = IDD_DIALOG3 };
int         m_lHeight;
CString    m_FileName;
int         m_lWidth;
int         m_Id;
//}}AFX_DATA

// Overrides
// Class Wizard generated virtual function overrides
//{{AFX_VIRTUAL(BmpInfo)
protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
//}}AFX_VIRTUAL

// Implementation
protected:

// Generated message map functions
//{{AFX_MSG(BmpInfo)
    // NOTE: the ClassWizard will add member functions here
//}}AFX_MSG
DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_BMPINFO_H__25C3C382_B745_11D4_A34F_0000C0825656__INCLUDED_)

```

```

// BmpImfo.cpp : implementation file
//

#include "stdafx.h"
#include "FTAAlgorithm.h"
#include "BmpImfo.h"

#ifndef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

///////////
// BmpImfo dialog

BmpImfo::BmpImfo(CWnd* pParent /*=NULL*/)
    : CDialog(BmpImfo::IDD, pParent)
{
    //{{AFX_DATA_INIT(BmpImfo)
    m_lHeight = 12;
    m_FileName = _T("c:\\Nicholas\\Data\\Image Info\\Mathy.txt");
    m_lWidth = 10;
    m_Id = 1;
    //}}AFX_DATA_INIT
}

void BmpImfo::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(BmpImfo)
    DDX_Text(pDX, IDC_lHeight, m_lHeight);
    DDX_Text(pDX, IDC_FName, m_FileName);
    DDX_Text(pDX, IDC_lWidth, m_lWidth);
    DDX_Text(pDX, IDC_Id, m_Id);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(BmpImfo, CDialog)
    //{{AFX_MSG_MAP(BmpImfo)
        // NOTE: the ClassWizard will add message map macros here
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

///////////
// BmpImfo message handlers

```

```

class CExtractImageInfo
{
private:
public:
    int      Deleteindx;
    CString Message;
    FILE*   stream;
    FILE*   streaml;
    CExtractImageInfo(void);
    virtual ~CExtractImageInfo(void);
    void DeleteAll(void);

public:
    int          Width_Bytes;
    int          Height;
    int          Width;
    int          BitsPerPixel;
    BYTE*       Buffer;
    int          HLow;
    int          HHigh;
    int          WLow;
    int          WHigh;
    int*        ExtId;
    int          TotId;
    char        StoreInfo[300];
    char        ReadText[300];
    char        StoreInfoinf[300];
    int          TCh;
    int          TW;
    int          TH;

    void ReadImage(void);
    void GetCharacterArea(void);
    void DisplayTheCharacter(CDC* );
    void CleanImage(void);
    void ReadTextFile(void);
    void Initial(BYTE*,int,int,int,DWORD,CString,CString);
    void StoreTheCharacter(void);
    void ResetSize(int,int,BYTE* );
    void Moment(int,int,BYTE* );
};


```

```

#include "stdafx.h"
#include "ExtractImageInfo.h"

#define OffsetH      550
#define OffsetW     25
#define Noise       2
#define LineSpace   5
#define CharSpace   5
#define offsetCh 65
#define FWidth      20
#define FHeight     20
#define FeatWidth   7
#define FeatHeight  1

//char* TextFileName="ABCD.txt";
int GlobalInt;

CExtractImageInfo::CExtractImageInfo()
{
    Deleteindx=0;
}

CExtractImageInfo::~CExtractImageInfo()
{
}

void CExtractImageInfo::DeleteAll()
{
    if(Deleteindx==1)
    {
        delete[] Buffer;
    }
    else if(Deleteindx==2)
    {
        delete[] Buffer;
        delete[] ExtId;
    }
/*    else if(Deleteindx==3)
    {
        delete[] NL;
        delete[] x;
        delete[] NS;
    }
    else if(Deleteindx==4)
    {
        delete[] NL;
        delete[] x;
        delete[] NS;
        delete[] df;
    }
    else if(Deleteindx==5)
    {
        delete[] NL;
        delete[] x;
        delete[] NS;
        delete[] df;
        delete[] net;
    }
    else if(Deleteindx==6)
    {
        delete[] NL;
        delete[] x;
        delete[] NS;
        delete[] df;
        delete[] net;
        delete[] delta;
    }
}

```

```

        else if(Deleteindx==7)
        {
            delete[] NL;
            delete[] x;
            delete[] NS;
            delete[] df;
            delete[] net;
            delete[] delta;
            delete[] y;
        }
        else if(Deleteindx==8)
        {
            delete[] NL;
            delete[] x;
            delete[] NS;
            delete[] df;
            delete[] net;
            delete[] delta;
            delete[] y;
            delete[] xp;
        }
        else if(Deleteindx==9)
        {
            delete[] NL;
            delete[] x;
            delete[] NS;
            delete[] df;
            delete[] net;
            delete[] delta;
            delete[] y;
            delete[] xp;
            delete[] d;
        }
        else if(Deleteindx==10)
        {
            delete[] NL;
            delete[] x;
            delete[] NS;
            delete[] df;
            delete[] net;
            delete[] delta;
            delete[] y;
            delete[] xp;
            delete[] d;
            delete[] RecogResult;
        }*/
    else
    {
    }
    Deleteindx=0;
}

void CExtractImageInfo::Initial(BYTE* Bu,int WB,int Wi,int He,DWORD len,CString name1,CString name2)
{
    int Tot,H,W;
    DeleteAll();
    if(!(Buffer= new BYTE[len]))
    {
        Message.Format("CExtractImageInfo - Memory allocation problem");
        AfxMessageBox(Message,MB_OK);
        return;
    }
    Deleteindx=1;
    Width_Bytes=WB;
    Width=Wi;
    Height=He;
    strcpy(StoreInfo.name1);
    strcpy(ReadText.name2);
    strcpy(StoreInfoinf.name1);
    for(DWORD i=0;i<len;i++)
}

```

```

{
    Buffer[i]=Bu[i];
}

len=strlen(StoreInfoinf);
if(!(StoreInfoinf[len-4]=='.'))
{
    Message.Format("File name %s is not has extension",StoreInfoinf);
    AfxMessageBox(Message,MB_OK);
    return;
}

StoreInfoinf[len-3]='.';
StoreInfoinf[len-2]='n';
StoreInfoinf[len-1]='?';

if((stream=fopen(StoreInfoinf,"r")))
{
    fscanf(stream,"%d",&Tot);
    fscanf(stream,"%d",&H);
    fscanf(stream,"%d",&W);
    fclose(stream);
    TCh=Tot;
    TW=W;
    TH=H;
    if((W!=FeatWidth && H!=FeatHeight))
    {
        Message.Format("Image dimension does not match");
        AfxMessageBox(Message,MB_OK);
        TCh=0;
        TW=0;
        TH=0;
        return;
    }
}
else
{
    TCh=0;
    TW=0;
    TH=0;
}
}

void CExtractImageInfo::ReadImage()
{
//DWORD A 32-bit unsigned integer or the address of a segment and its associated offset.
int i,j,k,m,fileE=0;
BYTE temp,var,Trial=0;
int BlackPixel,IsHigh=0,FindWidthLimit=0;
int EmptyLine,IsEmptyLine;

stream=fopen(StoreInfo,"a");
stream1=fopen(StoreInfoinf,"w");
if(stream == NULL)
    fileE=1;
if(stream1 == NULL)
{
    if(fileE==0)
        fclose(stream);
    fileE=2;
}
if(fileE!=0)
{
    if(fileE==1)
        fclose(stream1);
    else;

    return;
}
}

```

```

for(i=0;i<OffsetH;i++)
{
    for(j=0;j<Width_Bytes;j++)
    {
        Buffer[i*Width_Bytes+j]=255;
    }
}
for(i=(Height-OffsetH);i<Height;i++)
{
    for(j=0;j<Width_Bytes;j++)
    {
        Buffer[i*Width_Bytes+j]=255;
    }
}
for(i=OffsetH;i<(Height-OffsetH);i++)
{
    for(j=0;j<OffsetW;j++)
    {
        Buffer[i*Width_Bytes+j]=255;
    }
}
for(i=OffsetH;i<(Height-OffsetH);i++)
{
    for(j=(Width_Bytes-OffsetW);j<Width_Bytes;j++)
    {
        Buffer[i*Width_Bytes+j]=255;
    }
}
CleanImage();
GlobalInt=0;
for(i=OffsetH;i<(Height-OffsetH);i++)
{
    j=OffsetW;
    IsEmptyLine=1;
    while(j<(Width_Bytes-OffsetW))
    {
        if(IsHigh==0)
        {
            if(Buffer[i*Width_Bytes+j] != 255)
            {
                BlackPixel=0;
                var=Buffer[i*Width_Bytes+j];
                temp=128;
                k=0;
                Trial=0;
                while(k<8)
                {
                    if((temp & var))
                    {
                        if(BlackPixel<=Noise && BlackPixel>0)
                        {
                            BlackPixel=0;
                            if(Trial==1)
                            {
                                Buffer[i*Width_Bytes+j]=255;
                                break;
                            }
                            else
                            {
                                for(m=0;m<=k;m++)
                                {
                                    var=(var | temp);
                                    temp=temp>>1;
                                }
                            }
                        }
                        else
                        {
                            temp=temp>>1;
                        }
                    }
                }
            }
        }
    }
}

```



```

        j++;
    }//j-loop
    if(IsEmptyLine==1 && IsHigh==1)
    {
        EmptyLine++;
        if(EmptyLine>LineSpace)
        {
            EmptyLine=0;
            HHigh=i-LineSpace;
            IsHigh=0;
            GetCharacterArea();
        }
    }
}//i-loop

TCh+=GlobalInt;
fprintf(stream1,"%d %d %d",TCh,TH,TW);
fclose(stream);
fclose(stream1);
Message.Format("All the characters are successfully extracted");
AfxMessageBox(Message,MB_OK);

//          pBitmap->SetBitmapBits( length,Buffer );
}

void CExtractImageInfo::GetCharacterArea()
{
    int i,j;
    int NextCharacter,IsLow;
    IsLow=1;
//    CClientDC aDC(this);

    for(i=OffsetW;i<(Width-OffsetW);i++)
    {
        NextCharacter=1;
        for(j=HLow;j<HHigh;j++)
        {
            if(IsLow==1)
            {
                if(Buffer[j*Width+i]!=255)
                {
                    IsLow=0;
                    NextCharacter=0;
                    WLow=i;
                    break;
                }
            }
            else
            {
                if(Buffer[j*Width+i] != 255)
                {
                    NextCharacter=0;
                    break;
                }
            }
        }
        if(NextCharacter==1 && IsLow==0)
        {
            IsLow=1;
            WHigh=i;
            if(GlobalInt>TotId)
                return;
            StoreTheCharacter();
            DisplayTheCharacter(&aDC);
            Moment();
        }
    }
}

```

```

}

void CExtractImageInfo::StoreTheCharacter()
{
    int i,j,IWidth,IHeight;
    int k,irow,jcolumn;
    BYTE* TBuf;
    BYTE Temp,value;

    IWidth=(WHigh-WLow)*8;
    IHeight=HHigh-HLow;

    if(IWidth<Noise || IHeight<Noise)
        return;
    if(TBuf=new BYTE[IWidth*IHeight])
    {
        irow=0;
        for(i=HLow;i<HHigh;i++)
        {
            jcolumn=0;
            for(j=WLow;j<WHigh;j++)
            {
                Temp=128;
                value=Buffer[i*Width+j];
                for(k=0;k<8;k++)
                {
                    if(value & Temp)
                    {
                        TBuf[irow*IWidth+jcolumn*8+k]=0;
                    }
                    else
                    {
                        TBuf[irow*IWidth+jcolumn*8+k]=1;
                    }
                    Temp=Temp>>1;
                }
                jcolumn++;
            }
            irow++;
        }
    }
    else
        return;

    TW=FeatWidth;
    TH=FeatHeight;

    // ResetSize(IHeight,IWidth,TBuf);
    Moment(IHeight,IWidth,TBuf);
    delete[] TBuf;
}

void CExtractImageInfo::ResetSize(int IHeight,int IWidth,BYTE* TBuf)
{
    int Xmin,Xmax,Ymin,Ymax,i,j,XW,YH,m,n,x,y;
    int xr,yr,minx,maxx,miny,maxy,trr,tyr,sum,x_y_;
    int Threshold;
    int* FImage;
    int* FeatImage;
    int count;

    Ymin=IHeight;
    Xmin=IWidth;
    Ymax=0;
    Xmax=0;
}

```

```

for(i=0;i<IHeight;i++)
{
    for(j=0;j<IWidth;j++)
    {
        if(TBuf[i*IWidth+j]==1)
        {
            if(Xmin>j) Xmin=j;
            if(Xmax<j) Xmax=j;
            if(Ymin>i) Ymin=i;
            if(Ymax<i) Ymax=i;
        }
    }
}

XW = Xmax - Xmin;
YH = Ymax - Ymin;

x = int(XW/FWidth);
xr= XW % FWidth;
y = int(YH/FHeight);
yr= YH % FHeight;

if(yr>0)
    y_=1;
else
    y_=0;

if((FImage==new int[FWidth*FHeight])&&(FeatImage==new int[FWidth+FHeight]))
{
    for(i=0;i<FWidth*FHeight;i++)
    {
        FImage[i]=-1;
    }
    if(FWidth>XW || FHeight>YH)
    {
        if(FWidth>XW && FHeight<=YH)
        {
            miny=maxy=0;
            tyr=yr-1;
            for(i=0;i<FHeight;i++)
            {
                miny=maxy;
                maxy+=y+y_;
                for(j=0;j<XW;j++)
                {
                    sum=0;
                    Threshold=(maxy-miny)/2;
                    for(m=miny;m<maxy;m++)
                    {
                        for(n=j;n<j+1;n++)
                        {
                            sum+=TBuf[(Ymin+m)*IWidth+Xmin+n];
                        }
                    }
                    if(sum>=Threshold)
                        FImage[i*FWidth+j]=1;
                }
                if(tyr>0)
                {
                    tyr--;
                    y_=1;
                }
                else
                    y_=0;
            }
        }
    }
}

```

```

        }
    }
    else if(FWidth<XW && FHeight>=YH)
    {
        for(i=0;i<YH;i++)
        {
            txr=xr;
            if(txr>0) x_=1; else x_=0;
            txr--;
            minx=maxx=0;
            for(j=0;j<FWidth;j++)
            {
                minx=maxx;
                maxx+=x+x_;
                sum=0;
                Threshold=(maxx-minx)/2;
                for(m=i;m<i+1;m++)
                {
                    for(n=minx;n<maxx;n++)
                    {
                        sum+=TBuf[(Ymin+m)*IWidth+Xmin+n];
                    }
                }
                if(txr>0)
                {
                    txr--;
                    x_=1;
                }
                else
                    x_=0;
                if(sum>=Threshold)
                    FImage[i*FWidth+j]=1;
            }
        }
    }
    else if(FWidth>XW && FHeight>YH)
    {
        for(i=Ymin;i<Ymax;i++)
        {
            for(j=Xmin;j<Xmax;j++)
            {
                if(TBuf[i*IWidth+j]==1)
                {
                    FImage[i*FWidth+j]=1;
                }
            }
        }
    }
    else;
}
else
{
    miny=maxy=0;
    tyr=yr-1;
    for(i=0;i<FHeight;i++)
    {
        txr=xr;
        if(txr>0) x_=1; else x_=0;
        txr--;
        minx=maxx=0;
        miny=maxy;
        maxy+=y+y_;
        for(j=0;j<FWidth;j++)
        {
            minx=maxx;
            maxx+=x+x_;
            sum=0;
            Threshold=(maxy-miny)*(maxx-minx)/2;
            for(m=miny;m<maxy;m++)
            {

```

```

        for(n=minx;n<maxx;n++)
        {
            sum+=TBuf[(Ymin+m)*FWidth+Xmin+n];
        }
    }
    if(txr>0)
    {
        txr--;
        x_=l;
    }
    else
        x_=0;
    if(sum>=Threshold)
        FImage[i*FWidth+j]=1;
}
if(tyr>0)
{
    tyr--;
    y_=l;
}
else
    y_=0;
}
}

fprintf(stream,"%d\n",ExtId[GlobalInt++]);

for(i=0;i<(FHeight+FWidth);i++)
    FeatImage[i]=0;

for(i=0;i<FHeight;i++)
{
    count=0;
    for(j=0;j<FWidth;j++)
    {
        if((FImage[i*FWidth+j])==1)
        {
            count+=1;
            FeatImage[FHeight+j]+=1;
        }
    }
    FeatImage[i]=count;
}

for(i=0;i<(FHeight + FWidth);i++)
{
    fprintf(stream,"%d ",FeatImage[i]);
}

/*
for(i=0;i<FHeight;i++)
{
    for(j=0;j<FWidth;j++)
    {
        fprintf(stream,"%d ",FImage[i*FWidth+j]);
    }
    fprintf(stream,"\n");
}
fprintf(stream,"\n");

delete[] FImage;
}

void CExtractImageInfo::DisplayTheCharacter(CDC* pDC)
{

```

```

}

void CExtractImageInfo::CleanImage()
{
    int i,j,k;
    BYTE Temp,value;
    BYTE* TBuf;
    int BPixel;

    if(TBuf=new BYTE[Width*8*Height])
    {
        for(i=0;i<Height;i++)
        {
            for(j=0;j<Width;j++)
            {
                Temp=128;
                value=Buffer[i*Width+j];
                for(k=0;k<8;k++)
                {
                    if(value & Temp)
                    {
                        TBuf[i*Width*8+j*8+k]=1;
                    }
                    else
                    {
                        TBuf[i*Width*8+j*8+k]=0;
                    }
                    Temp=Temp>>1;
                }
            }
        }

        for(i=0;i<Height;i++)
        {
            BPixel=0;
            for(j=0;j<(Width*8);j++)
            {
                if(!(TBuf[i*Width*8+j]))
                {
                    BPixel++;
                }
                else
                {
                    if(BPixel<=Noise)
                    {
                        for(k=1;k<=BPixel;k++)
                        {
                            TBuf[i*Width*8+j-k]=1;
                        }
                    }
                    BPixel=0;
                }
            }
        }

        for(i=0;i<(Width*8);i++)
        {
            BPixel=0;
            for(j=0;j<Height;j++)
            {
                if(!(TBuf[j*Width*8+i]))
                {
                    BPixel++;
                }
                else
                {

```

```

        if(BPixel<=Noise)
        {
            for(k=1;k<=BPixel;k++)
            {
                TBuf[(j-k)*Width*8+i]=l;
            }
            BPixel=0;
        }
    }

    for(i=0;i<Height;i++)
    {
        for(j=0;j<Width;j++)
        {
            Temp=128;
            value=0;
            for(k=0;k<8;k++)
            {
                if(TBuf[i*Width*8+j*8+k])
                {
                    value|=Temp;
                }
                Temp=Temp>>1;
            }
            Buffer[i*Width+j]=value;
        }
    }
}
else
{
    return;
}
delete[] TBuf;
}

void CExtractImageInfo::ReadTextFile()
{
    char ch;
    int Index,count;
    stream=fopen(ReadText,"r");
    TotId=0;
    if(stream != NULL)
    {
        ch=l;
        rewind(stream);
        count=0;
        while(ch != EOF && count < 10)
        {
            fscanf(stream,"%c",&ch);
            if(ch>64 && ch < 91)
            {
                TotId++;
                count=0;
            }
            else
                count++;
        }
        if(!(ExtId= new int[TotId]))
        {
            Message.Format("CExtractImageInfo - Memory allocation problem");
            AfxMessageBox(Message,MB_OK);
            return;
        }
        Deleteindx=2;
        rewind(stream);
        Index=0;
    }
}

```

```

        count=0;
        while(ch != EOF && count < 10)
        {
            fscanf(stream,"%c",&ch);
            if(ch>64 && ch < 91)
            {
                ExtId[Index++]=ch;-offsetCh;
                count=0;
            }
            else
                count++;
        }
        fclose(stream);
    }

void CExtractImageInfo::Moment(int lHeight1,int lWidth1,BYTE* TBuf1)
{
    // Source: [KHA88]
    double fabs(),sqrt(),log();
    double m00,m10,m20,m30,m03,m12,m21,m02,m11,m01;
    double a,b,yc,xc,u20,u02,u30,u03,u12,u21,u11,v,h;
    double c,d,e,r,g,n11,n20,n02,n12,n21,n30,n03;
    int a1,b1,Hori,Vert,indx;
    float f[7];
    int* TBuf;
    int      Ymin,Xmin,Ymax,Xmax,lWidth,lHeight,i,j;
    float
Offset[8]={0.673524462f,4.184239497f,5.07147142f,7.343663569f,14.33523014f,10.10236316f,14.20623164f,0.965387666f};

    Ymin=lHeight1;
    Xmin=lWidth1;
    Ymax=0;
    Xmax=0;

    for(i=0;i<lHeight1;i++)
    {
        for(j=0;j<lWidth1;j++)
        {
            if(TBuf1[i*lWidth1+j]==1)
            {
                if(Xmin>j)          Xmin=j;
                if(Xmax<j)          Xmax=j;
                if(Ymin>i)          Ymin=i;
                if(Ymax<i)          Ymax=i;
            }
        }
    }
    lWidth = Xmax - Xmin;
    lHeight = Ymax - Ymin;

    if(!(TBuf= new int[lWidth*lHeight]))
    {
        Message.Format("CExtractImageInfo - Memory allocation problem");
        AfxMessageBox(Message,MB_OK);
        return;
    }

    indx=0;
    for(i=Ymin;i<Ymax;i++)
    {
        for(j=Xmin;j<Xmax;j++)
        {
            TBuf[indx++]=TBuf1[i*lWidth1+j];
        }
    }
}

```

```

        }
        Hori=IHeight%2;
        Vert=IWidth%2;

        m00 = 0.0;
        m01 = 0.0;
        m10 = 0.0;
        m11 = 0.0;
        m21 = 0.0;
        m12 = 0.0;
        m02 = 0.0;
        m20 = 0.0;
        m30 = 0.0;
        m03 = 0.0;

        u11 = 0.0;
        u20 = 0.0;
        u02 = 0.0;
        u30 = 0.0;
        u03 = 0.0;
        u12 = 0.0;
        u21 = 0.0;

    for( a1=0; a1<IHeight; a1++)
        for(b1 = 0; b1<IWidth; b1++)
        {
            if(TBuf[a1*IWidth+b1]==1)
            {
                // Compute up to the third order of moments //
                m00 +=1.0;
                m10 = m10 + 1.0*b1;
                m01 = m01 + 1.0*a1;
                m11 = m11 + 1.0*a1*b1;
                m20 = m20 + (b1*1.0)*(b1*1.0);
                m02 = m02 + (a1*1.0)*(a1*1.0);
                m12 = m12 + b1*(a1*1.0)*(a1*1.0);
                m21 = m21 + a1*(b1*1.0)*(b1*1.0);
                m30 = m30 + b1*(b1*1.0)*(b1*1.0);
                m03 = m03 + a1*(a1*1.0)*(a1*1.0);

            }
        }

        yc = m01/m00;
        xc = m10/m00;

    // Compute central moments from ordinary moment //
        u11 = m11 - (yc * m10);
        u20 = m20 - xc * m10;
        u02 = m02 - (yc * m01);
        u30 = m30 - (3 * xc * m20) + 2 * (m10 * (xc * xc));
        u03 = m03 - (3 * yc * m02) + 2 * (m01 * (yc * yc));
        u12 = m12 - (2 * yc * m11) - (xc * m02) + 2 * (yc * yc * m10); // correction m01 -> m10
        u21 = m21 - (2 * xc * m11) - (yc * m20) + (2 * xc * xc * m01);

    // to normalize the central moments //

        a = (m00 * m00);
        b = (a * (:sqrt((m00))));

        n20 = (u20/a);

```

```

n02 = (u02/a);
n11 = (u11/a);
n12 = (u12/b);
n21 = (u21/b);
n30 = (u30/b);
n03 = (u03/b);

a = n20 - n02;
b = n30 - 3 * n12;
c = n30 + n12;
d = 3 * n21 - n03;
e = n21 + n03;
r = 3 * n12 - n30;
g = 3 * n12 - n03;
v = (d*c) * (c*c); // correction
h= v-3*(d*c)*(e*e)+(g*e)*(3*(c*c)-(e*e));

// Compute the invariant moment //
f[0] = (float) ::fabs(::log(::fabs(n20 + n02)));
f[1] = (float) ::fabs (::log(::fabs((a*a) + 4 *(n11*n11))));
f[2] = (float) ::fabs (::log(::fabs((b*b) + (d*d))));
f[3] = (float) ::fabs (::log(::fabs((c*c) + (e*e))));
f[4] = (float) ::fabs (::log(::fabs((b*c)*((c*c)-3*(e*e))+(d*e)*(3*(c*c)-(e*e))))); // correction bracket
f[5] = (float) ::fabs (::log(::fabs(a*((c*c)-(e*e)) + (4*n11*c*e))));// correction
f[6] = (float) ::fabs (::log(::fabs(h))); //correction

fprintf(stream,"%d\n",ExtId[GlobalInt++]);

for(a1=0;a1<7;a1++)
{
    fprintf(stream,"%f ",(f[a1]-Offset[a1]));
}
fprintf(stream," \n");
delete[] TBuf;
}
}

class CFTA

```

```

{
private:

protected:

public:
    int      Deleteindx;
    CString Message;
    CFTA(void);
    FILE* stream99;
    virtual ~CFTA(void);
    void DeleteAll(void);

public:
    int*      NL;
    int*      NS;
    int       L;
    int       M;
    float*   w;
    float*   wi;
    float*   w2;
    float*   y;
    float*   delta;
    float*   net;
    float*   dd;
    float*   d;
    float   Sp;
    float   Theeta;
    float   ne;
    float*   x;
    float*   kl;
    float*   v;
    float*   R;
    float   b;
    float   st;
    float   TotalE;
    int*     RecogResult;
    int       TotalR;
    float   THold;
    float   Momentum;
    char    WFname[300];
    int     TNW;
    float   GMError;
    int     GMRecog;
    int     Niter;

void TrainInit(int,int*,int,float*,int*);
void PropagateForward(int);
void PropagateBackward(int);
void KalmanGain(int);
void FTAUpdateWeight(int);
void ReadWrite(int,int,char* );
void FTATrain(void);
void Test(void);
void TotalError(void);
};


```

```
#include "stdafx.h"
```

```

#include "FastTA.h"

#define fd(i) (1.0-y[i]*y[i])

float PInf=200;
float NInf= -200;
int      TIt=0;

CFTA::CFTA()
{
    Deleteindx=0;
    Sp=0.2f;
    Theeta=0.1f;
    THold=0.5f;
    b=0.93f;
    st=70.0f;
    Momentum=0.6f;
    TNW=0;
    GMError=99999.9f;
    GMRecog=0;
}

CFTA::~CFTA()
{
}

void CFTA::DeleteAll()
{
    if(Deleteindx==1)
    {
        delete[] NL;
    }
    else if(Deleteindx==2)
    {
        delete[] NL;
        delete[] x;
    }
    else if(Deleteindx==3)
    {
        delete[] NL;
        delete[] x;
        delete[] d;
    }
    else if(Deleteindx==4)
    {
        delete[] NL;
        delete[] x;
        delete[] d;
        delete[] w;
    }
    else if(Deleteindx==5)
    {
        delete[] NL;
        delete[] x;
        delete[] d;
        delete[] w;
        delete[] net;
    }
    else if(Deleteindx==6)
    {
        delete[] NL;
        delete[] x;
        delete[] d;
        delete[] w;
        delete[] net;
        delete[] y;
    }
    else if(Deleteindx==7)
    {
        delete[] NL;
    }
}

```

```

        delete[] x;
        delete[] d;
        delete[] w;
        delete[] net;
        delete[] y;
        delete[] delta;
    }
    else if(Deleteindx==8)
    {
        delete[] NL;
        delete[] x;
        delete[] d;
        delete[] w;
        delete[] net;
        delete[] y;
        delete[] delta;
        delete[] kl;
    }
    else if(Deleteindx==9)
    {
        delete[] NL;
        delete[] x;
        delete[] d;
        delete[] w;
        delete[] net;
        delete[] y;
        delete[] delta;
        delete[] kl;
        delete[] v;
    }
    else if(Deleteindx==10)
    {
        delete[] NL;
        delete[] x;
        delete[] d;
        delete[] w;
        delete[] net;
        delete[] y;
        delete[] delta;
        delete[] kl;
        delete[] v;
        delete[] R;
    }
    else if(Deleteindx==11)
    {
        delete[] NL;
        delete[] x;
        delete[] d;
        delete[] w;
        delete[] net;
        delete[] y;
        delete[] delta;
        delete[] kl;
        delete[] v;
        delete[] R;
        delete[] dd;
    }
    else if(Deleteindx==12)
    {
        delete[] NL;
        delete[] x;
        delete[] d;
        delete[] w;
        delete[] net;
        delete[] y;
        delete[] delta;
        delete[] kl;
        delete[] v;
        delete[] R;
        delete[] dd;
    }
}

```

```

        delete[] RecogResult;
    }
    else if(Deleteindx==13)
    {
        delete[] NL;
        delete[] x;
        delete[] d;
        delete[] w;
        delete[] net;
        delete[] y;
        delete[] delta;
        delete[] kl;
        delete[] v;
        delete[] R;
        delete[] dd;
        delete[] RecogResult;
        delete[] NS;
    }
    else
    {
    }
    Deleteindx=0;
}

void CFTA::TrainInit(int l,int* nl,int mnm,float* xx,int* idid)
{
    int i,N,NN1,Nt,Ntt,Ntt1,NNN,N1,j;

    L=l;
    M=mnm;

    DeleteAll();
    if(!(NL=new int[L]))
    {
        Message.Format("CFTA-Memory allocation problem");
        AfxMessageBox(Message,MB_OK);
        return;
    }
    Deleteindx=1;

    if(!(x=new float[M*nl[0]]))
    {
        Message.Format("CFTA-Memory allocation problem");
        AfxMessageBox(Message,MB_OK);
        return;
    }
    Deleteindx=2;

    if(!(d=new float[M*nl[L-1]]))
    {
        Message.Format("CFTA-Memory allocation problem");
        AfxMessageBox(Message,MB_OK);
        return;
    }
    Deleteindx=3;

    for(i=0;i<l;i++)
    {
        NL[i]=nl[i];
    }

    for(i=0,NN1=0;i<L-1;i++)
    {
        NN1+=NL[i]*NL[i+1];
    }
    TNW=N=NN1;

    if!(w=new float[N])

```

```

{
    Message.Format("CFTA-Memory allocation problem");
    AfxMessageBox(Message,MB_OK);
    return;
}
Deleteindx=4;

strand( (unsigned)time( NULL ) );
for(i=0;i<N;i++)
    w[i]=(float)((rand()%N)/(float)N)-0.5f;

for(i=1,Nt=0;i<L;i++)
    Nt+=NL[i];

for(i=1,Ntt=0;i<(L-1);i++)
    Ntt+=NL[i];
Ntt1=NL[0]*NL[0];

if(!(net=new float[NL[L-1]]))
{
    Message.Format("CFTA-Memory allocation problem");
    AfxMessageBox(Message,MB_OK);
    return;
}
Deleteindx=5;
if(!(y=new float[Nt]))
{
    Message.Format("CFTA-Memory allocation problem");
    AfxMessageBox(Message,MB_OK);
    return;
}
Deleteindx=6;
if(!(delta=new float[Nt]))
{
    Message.Format("CFTA-Memory allocation problem");
    AfxMessageBox(Message,MB_OK);
    return;
}
Deleteindx=7;

Nt+=(NL[0]-NL[L-1]);

if(!(kl=new float[Nt]))
{
    Message.Format("CFTA-Memory allocation problem");
    AfxMessageBox(Message,MB_OK);
    return;
}
Deleteindx=8;
if(!(v=new float[Nt]))
{
    Message.Format("CFTA-Memory allocation problem");
    AfxMessageBox(Message,MB_OK);
    return;
}
Deleteindx=9;

for(i=0,N1=0;i<L-1;i++)
    N1+= NL[i]*NL[i];

if(!(R=new float[N1]))
{
    Message.Format("CFTA-Memory allocation problem");
    AfxMessageBox(Message,MB_OK);
    return;
}
Deleteindx=10;

NNN=N1;

```

```

strand( (unsigned)time( NULL ) );
for(i=0;i<N1;i++)
    R[i]=(float)((rand()%N1)/(float)N1)-0.5f;

if(!(dd=new float[M*NL[L-1]]))
{
    Message.Format("CFTA-Memory allocation problem");
    AfxMessageBox(Message,MB_OK);
    return;
}
Deleteindx=11;
if(!(RecogResult=new int[M*3]))
{
    Message.Format("CFTA-Memory allocation problem");
    AfxMessageBox(Message,MB_OK);
    return;
}
Deleteindx=12;
if!(NS=new int[L])
{
    Message.Format("CFTA-Memory allocation problem");
    AfxMessageBox(Message,MB_OK);
    return;
}
Deleteindx=13;
for(i=0;i<M;i++)
{
    for(j=0;j<NL[0];j++)
    {
        x[j*M+i]=xx[i*NL[0]+j];
    }
}
NS[0]=NL[0]*NL[1];
for(i=1;i<L-2;i++) NS[i]=NS[i-1]+NL[i]*NL[i+1];

/*
for(i=0;i<NL[L-1]*M;i++)
{
    d[i]=-0.5f;
}
for(i=0;i<M;i++)
{
    d[i+M*(idid[i])]=0.5f;
}
*/
int ch;

for(i=0;i<M;i++)
{
    ch=1;
    RecogResult[i*3+0]=idid[i];
    for(j=0;j<8;j++)
    {
        if(idid[i] & ch)
            d[i+M*j]=0.5f;
        else
            d[i+M*j]=-0.5f;
        ch=ch<<1;
    }
}

for(i=0;i<(M*NL[L-1]);i++)
{
    dd[i]=(float)(log((1.0f+d[i])/(1.0f-d[i]))/Sp);
}
}

```

```

void CFTA::PropagateForward(int qq)
{
    // Source: [OSA94]
    float E;
    int j,z,Nt1,Nt2,n,m;

    wi=w;
    for(z=0;z<NL[1];z++)
    {
        for(j=0,ne=Theeta;j<NL[0];j++)
            ne+=(*wi++)*(x[j]*M+qq));
        if(ne> PInf)
        {
            ne=(float)PInf;
        }
        else if(ne<NInf)
        {
            ne=(float)NInf;
        }
        else;
        E=(float)exp(-(double)Sp*ne);
        y[z]=(float)((2.0/(1.0+E))-1);
    }
    Nt1=NL[1];
    Nt2=0;

    for(n=2;n<L-1;n++)
    {
        for(z=0;z<NL[n];z++)
        {
            m=Nt1+z;
            for(j=0,ne=Theeta;j<NL[n-1];j++)
                ne+=*(wi++)*y[j+Nt2];
            if(ne>PInf)
            {
                ne=(float)PInf;
            }
            else if(ne<NInf)
            {
                ne=(float)NInf;
            }
            else;
            E=(float)exp(-(double)Sp*ne);
            y[m]=(float)((2.0/(1.0+E))-1);
        }
        Nt1+=NL[n];
        Nt2+=NL[n-1];
    }

    for(z=0;z<NL[L-1];z++)
    {
        m=Nt1+z;
        for(j=0,net[z]=Theeta;j<NL[L-2];j++)
            net[z]+=(wi++)*y[j+Nt2];
        E=(float)exp(-(double)Sp*net[z]);
        if(ne>PInf)
        {
            ne=(float)PInf;
        }
        else if(ne<NInf)
        {
            ne=(float)NInf;
        }
        else;
        y[m]=(float)((2.0/(1.0+E))-1);
    }
}

```

```

void CFTA::PropagateBackward(int qq)
{
    // Source: [OSA94]
    int j,z,Nt1,ii,Nt2,n,m;
    float error,sum;
    for(z=1,Nt1=0;z<(L-1);z++)
        Nt1+=NL[z];

    for(z=0;z<NL[L-1];z++)
    {
        ii=Nt1+z;
        error=float(d[qq+z*M]-y[ii]);
        delta[ii]=float(error*fd(ii)*Sp/2.0f);
    }

    for(m=0;m<(L-2);m++)
    {
        Nt2=Nt1-NL[L-2-m];
        for(z=0;z<NL[L-2-m];z++)
        {
            ii=Nt2+z;
            sum=0.0f;
            n=NS[L-3-m]+z;
            for(j=0;j<NL[L-1-m];j++)
                sum+=delta[Nt1+j]*w[n+j*NL[L-2-m]];
            delta[ii]=float(sum*fd(ii)*Sp/2.0f);
        }
        Nt1=Nt2;
    }
}

void CFTA::KalmanGain(int qq)
{
    // Source: [OSA94]
    int j,z,Nt1,Nt2,Nt3,n,m;
    float xrx;

    Nt1=NL[0]*NL[0];

    for(z=0,xrx=b;z<NL[0];z++)
    {
        for(j=0,v[z]=0.0;j<NL[0];j++)
            v[z]+=R[z+j]*NL[0]*x[qq+j*M];
        xrx+=v[z]*v[z];
    }
    xrx=1/xrx;
    xrx/=float(1+sqrt(xrx));

    for(z=0;z<NL[0];z++)
    {
        for(j=0,kl[z]=0;j<NL[0];j++)
            kl[z]+=R[j+z*NL[0]]*v[j];
        kl[z]*=xrx;
        for(j=0,n=z*NL[0];j<NL[0];j++)
            R[j+n]=(R[j+n]-v[j]*kl[z])/b;
    }
    Nt2=NL[0];
    Nt3=0;

    for(m=1;m<(L-1);m++)
    {
        for(z=0,xrx=b;z<NL[m];z++)
        {
            for(j=0,v[z+Nt2]=0.0;j<NL[m];j++)
                v[z+Nt2]+=R[z+Nt1+j*NL[m]]*y[Nt3+j];
            xrx+=v[z+Nt2]*v[z+Nt2];
        }
    }
}

```

```

xrx=1/xrx;
xrx=float(1+sqrt(xrx));

for(z=0;z<NL[m];z++)
{
    for(j=0,kl[z+Nt2]=0,n=Nt1+z*NL[m];j<NL[m];j++)
        kl[z+Nt2]+=R[n+j]*v[j+Nt2];
    kl[z+Nt2]*=xrx;
    for(j=0,n=Nt1+z*NL[m];j<NL[m];j++)
        R[n+j]=((R[n+j]-v[j+Nt2]*kl[z+Nt2])/b);
}
Nt1+=NL[m]*NL[m];
Nt2+=NL[m];
Nt3+=NL[m];
}

void CFTA::FTAUpdateWeight(int qq)
{
    // Source: [OSA94]
    int Nt1,Nt2,z,j,n;
    wi=w;
    for(z=0;z<NL[1];z++)
        for(j=0;j<NL[0];j++)
            *(wi++)+=kl[j]*delta[z]*st*0.1f;

    Nt1=NL[0];
    Nt2=NL[1];

    for(n=2;n<L-1;n++)
    {
        for(z=0;z<NL[n];z++)
            for(j=0;j<NL[n-1];j++)
                *(wi++)+=kl[Nt1+j]*delta[z+Nt2]*st;
        Nt1+=NL[n-1];
        Nt2+=NL[n];
    }

    for(z=0;z<NL[L-1];z++)
        for(j=0;j<NL[L-2];j++)
            *(wi++)+=kl[Nt1+j]*(dd[qq+z*M]-net[z])*0.1f;

    if(b<0.99)
        b+=0.000001f;
}

void CFTA::ReadWrite(int flag,int tot,char* wname)
{
    FILE* stream;
    int i;
    float fp;

    if(flag==0)
    {
        if(stream=fopen(wname,"w"))
        {
            for(i=0;i<tot;i++)
            {
                fprintf(stream,"%f ",w[i]);
            }
            fclose(stream);
        }
        else
        {
            Message.Format("CFTA-File %s cannot open for write",wname);
            AfxMessageBox(Message,MB_OK);
            return;
        }
    }
}

```

```

else if(flag==1)
{
    if(stream=fopen(wname,"r"))
    {
        for(i=0;i<tot;i++)
        {
            fscanf(stream,"%f",&fp);
            w[i]=fp;
        }
        fclose(stream);
    }
    else
    {
        Message.Format("CFTA-File %s cannot open",wname);
        AfxMessageBox(Message,MB_OK);
        return;
    }
}
else
{
    Message.Format("CFTA-It is not read or write flag number");
    AfxMessageBox(Message,MB_OK);
    return;
}

void CFTA::FTATrain(void)
{
    int i,qq;
    float var,ivar;
    srand((unsigned)time( NULL ) );
    qq=(rand()%Niter);
    PropagateForward(qq);
    PropagateBackward(qq);
    KalmanGain(qq);
    FTAUpdateWeight(qq);
    Niter--;
    TotalError();
    Test();
    fprintf(stream99,"%d %d %f\n",TIt,TotalR,TotalE);

    if(GMError>TotalE)
    {
        GMError=TotalE;
    }
    if(GMRecog<TotalR)
    {
        GMRecog=TotalR;
        ReadWrite(0,TNW,WFname);
    }
    for(i=0;i<NL[0];i++)
    {
        ivar=x[i*M+Niter];
        x[i*M+Niter]=x[i*M+qq];
        x[i*M+qq]=ivar;
    }
    for(i=0;i<NL[L-1];i++)
    {
        var=d[i*M+Niter];
        d[i*M+Niter]=d[i*M+qq];
        d[i*M+qq]=var;
        var=dd[i*M+Niter];
        dd[i*M+Niter]=dd[i*M+qq];
        dd[i*M+qq]=var;
    }
    TIt++;
}

void CFTA::Test(void)
{

```

```

int i,j,indy,Count,recgC,ch;
float var,er;

for(i=1,indy=0;i<(L-1);i++)
{
    indy+=NL[i];
}
TotalR=0;
for(i=0;i<M;i++)
{
    PropagateForward(i);
    Count=0;
    er=0;
    recgC=0;
    ch=1;
    for(j=0;j<NL[L-1];j++)
    {
        if(d[i+j*M]>0)
        {
            Count+=1;
            recgC=recgC | ch;
        }
        ch=ch<<1;
        var=d[i+j*M]-y[indy+j];
        if(fabs(var)>THold)
        {
            er+=10;
        }
    }
    if(Count==1 && er==0)
    {
        RecogResult[i*3+1]=recgC;
        RecogResult[i*3+2]=1;
        TotalR++;
    }
    else
    {
        RecogResult[i*3+1]=Count;
        RecogResult[i*3+2]=-1;
    }
}
}

void CFTA::TotalError(void)
{
    int i,j,indy;
    float er;

    for(i=1,indy=0;i<(L-1);i++)
    {
        indy+=NL[i];
    }
    TotalE=0.0f;
    for(i=0;i<M;i++)
    {
        PropagateForward(i);
        er=0;
        for(j=0;j<NL[L-1];j++)
        {
            er+=d[i+j*M]-y[indy+j];
            TotalE+=(er*er);
        }
    }
    TotalE=TotalE/2.0f;
}
}

```

class CDelta

```

{
private:
protected:
public:
    int      Deleteindx;
    CString Message;
    FILE*   stream;
    CDelta(void);
    virtual ~CDelta(void);
    void DeleteAll(void);

public:
    float    st;
    float    THold;
    int*    NS;
    int*    NL;
    int          L;
    float*   x;
    float*   xp;
    int          M;
    float*   df;
    float*   net;
    float*   delta;
    float*   y;
    float    Sp;
    float*   d;
    float    theta;
    int          TRecogn;
    float    TotalE;
    int*    RecogResult;

    void Initial(int,int,float*,int*,int* );
    void Train(void);
    void deltafun(void);
    float fdelta(void);
    void ReadWrite(int,int,char* );
};


```

```

#include "stdafx.h"
#include "Delta.h"

#define fd(i) (1.0-i*i)

```

```

CDelta::CDelta()
{
    Deleteindx=0;
    Sp=0.2f;
    theta=0.1f;
    THold=0.5f;
    st=0.2f;
}

CDelta::~CDelta()
{
}

void CDelta::DeleteAll()
{
    if(Deleteindx==1)
    {
        delete[] NS;
    }
    else if(Deleteindx==2)
    {
        delete[] NL;
        delete[] NS;
    }
    else if(Deleteindx==3)
    {
        delete[] NL;
        delete[] x;
        delete[] NS;
    }
    else if(Deleteindx==4)
    {
        delete[] NL;
        delete[] x;
        delete[] NS;
        delete[] df;
    }
    else if(Deleteindx==5)
    {
        delete[] NL;
        delete[] x;
        delete[] NS;
        delete[] df;
        delete[] net;
    }
    else if(Deleteindx==6)
    {
        delete[] NL;
        delete[] x;
        delete[] NS;
        delete[] df;
        delete[] net;
        delete[] delta;
    }
    else if(Deleteindx==7)
    {
        delete[] NL;
        delete[] x;
        delete[] NS;
        delete[] df;
        delete[] net;
        delete[] delta;
        delete[] y;
    }
    else if(Deleteindx==8)
    {
        delete[] NL;
        delete[] x;
        delete[] NS;
    }
}

```

```

        delete[] df;
        delete[] net;
        delete[] delta;
        delete[] y;
        delete[] xp;
    }
    else if(Deleteindx==9)
    {
        delete[] NL;
        delete[] x;
        delete[] NS;
        delete[] df;
        delete[] net;
        delete[] delta;
        delete[] y;
        delete[] xp;
        delete[] d;
    }
    else if(Deleteindx==10)
    {
        delete[] NL;
        delete[] x;
        delete[] NS;
        delete[] df;
        delete[] net;
        delete[] delta;
        delete[] y;
        delete[] xp;
        delete[] d;
        delete[] RecogResult;
    }
    else
    {
    }
    Deleteindx=0;
}

void CDelta::Initial(int TS,int TL,float* xxp,int* iid,int* nNL)
{
    int i,j,N;
    M=TS;
    L=TL;

    DeleteAll();

    if(!(NS=new int[L]))
    {
        Message.Format("CDelta-Memory allocation problem");
        AfxMessageBox(Message,MB_OK);
        return;
    }
    Deleteindx=1;
    if(!(NL=new int[L]))
    {
        Message.Format("CDelta-Memory allocation problem");
        AfxMessageBox(Message,MB_OK);
        return;
    }
    Deleteindx=2;

    for(i=0;i<L;i++)
    {
        NL[i]=nNL[i];
    }

    NS[0]=NL[0]*NL[1];
    for(i=1;i<L-2;i++) NS[i]=NS[i-1]+NL[i]*NL[i+1];
}

```

```

for(i=0,N=0;i<L-1;i++) N+=NL[i]*NL[i+1];

if(!(x=new float[N]))
{
    Message.Format("CDelta-Memory allocation problem");
    AfxMessageBox(Message,MB_OK);
    return;
}
Deleteindx=3;
strand( (unsigned)time( NULL ) );
for(i=0;i<N;i++)
    x[i]=(float)((rand()%N)/(float)N)-0.5f;

if(!(df=new float[N]))
{
    Message.Format("CDelta-Memory allocation problem");
    AfxMessageBox(Message,MB_OK);
    return;
}
Deleteindx=4;
if(!(net=new float[NL[L-1]]))
{
    Message.Format("CDelta-Memory allocation problem");
    AfxMessageBox(Message,MB_OK);
    return;
}
Deleteindx=5;
for(i=1,N=0;i<L;i++)
    N+=NL[i];
if(!(delta=new float[N]))
{
    Message.Format("CDelta-Memory allocation problem");
    AfxMessageBox(Message,MB_OK);
    return;
}
Deleteindx=6;
if(!(y=new float[N]))
{
    Message.Format("CDelta-Memory allocation problem");
    AfxMessageBox(Message,MB_OK);
    return;
}
Deleteindx=7;
if(!(xp=new float[M*NL[0]]))
{
    Message.Format("CDelta-Memory allocation problem");
    AfxMessageBox(Message,MB_OK);
    return;
}
Deleteindx=8;
for(i=0;i<M;i++)
{
    for(j=0;j<NL[0];j++)
    {
        xp[j*M+i]=xxp[i*NL[0]+j];
    }
}
if(!(d=new float[M*NL[L-1]]))
{
    Message.Format("CDelta-Memory allocation problem");
    AfxMessageBox(Message,MB_OK);
    return;
}
Deleteindx=9;
if(!(RecogResult=new int[M*3]))
{
    Message.Format("CDelta-Memory allocation problem");
    AfxMessageBox(Message,MB_OK);
    return;
}

```

```

        }
    Deleteindx=10;
/*     for(i=0;i<NL[L-1]*M;i++)
    {
        d[i]=-0.5f;
    }
    for(i=0;i<M;i++)
    {
        RecogResult[i*3]=iid[i];
        d[i+M*(iid[i])]=0.5f;
    }
*/
int ch;

for(i=0;i<M;i++)
{
    RecogResult[i*3]=iid[i];
    ch=1;
    for(j=0;j<8;j++)
    {
        if(iid[i] & ch)
            d[i+M*j]=0.5f;
        else
            d[i+M*j]=-0.5f;
        ch=ch<<1;
    }
}

void CDelta::Train()
{
    int i,N;

    for(i=0,N=0;i<L-1;i++) N+=NL[i]*NL[i+1];

    deltafun();
    for(i=0;i<N;i++)
        x[i]+=st*df[i];
    TotalE=fdelta();
}

void CDelta::deltafun(void)
{
    // Source: [SID94]
    int i,j,k,m,n,N,Nt1,Nt2,Nt3,ii,z;
    float E,error,sum,ne;
    for(i=0,N=0;i<L-1;i++) N+=NL[i]*NL[i+1];
    for(i=0;i<N;i++)
        df[i]=0.0f;
    for(i=0;i<M;i++)
    {
        for(z=0;z<NL[1];z++)
        {
            for(j=0,ne=theta;j<NL[0];j++)
                ne+=x[z+j*NL[1]]*xp[j*M+i];
            E=(float)exp(-(double)Sp*ne);
            y[z]=(float)((2.0/(1.0+E))-1);
        }
        Nt1=NL[1];
        Nt2=0;
        for(n=2;n<L-1;n++)
        {
            for(z=0;z<NL[n];z++)

```

```

    {
        m=Nt1+z;
        for(j=0,ne=theta;j<NL[n-1];j++)
            ne+=x[NS[n-2]+z+j*NL[n]]*y[j+Nt2];
        E=(float)exp(-(double)Sp*ne);
        y[m]=(float)((2.0/(1.0+E))-1);
    }
    Nt1+=NL[n];
    Nt2+=NL[n-1];
}
for(z=0;z<NL[L-1];z++)
{
    m=Nt1+z;
    for(j=0,net[z]=theta;j<NL[L-2];j++)
        net[z] +=x[NS[L-3]+z+j*NL[L-1]]*y[j+Nt2];
    E=(float)exp(-(double)Sp*net[z]);
    y[m]=(float)((2.0/(1.0+E))-1);
}
for(z=1,Nt1=0;z<(L-1);z++)
    Nt1+=NL[z];

for(z=0;z<NL[L-1];z++)
{
    ii=Nt1+z;
    error=float(d[i+z*M]-y[ii]);
    delta[ii]=float(error*fd(y[ii])*Sp/2.0f);
}
for(m=0;m<(L-2);m++)
{
    Nt2=Nt1-NL[L-2-m];
    for(z=0;z<NL[L-2-m];z++)
    {
        ii=Nt2+z;
        sum=0.0;
        n=NS[L-3-m];
        for(j=0;j<NL[L-1-m];j++)
            sum+=delta[Nt1+j]*x[n+j+z*NL[L-1-m]];
        delta[ii]=float(sum*fd(y[ii])*Sp/2.0f);
    }
    Nt1=Nt2;
}
for(k=0;k<NL[1];k++)
{
    for(j=0;j<NL[0];j++)
    {
        df[k+j*NL[1]]+=delta[k]*xp[i+j*M];
    }
}
Nt1=NS[0]; Nt2=0;
Nt3=NL[1];
for(m=1;m<(L-1);m++)
{
    for(k=0;k<NL[m+1];k++)
        for(j=0;j<NL[m];j++)
            df[Nt1+k+j*NL[m+1]]+=delta[Nt3+k]*y[Nt2+j];
    Nt1=NS[m];
    Nt2+=NL[m];
    Nt3+=NL[m+1];
}
}

float CDelta::fdelta(void)
{
    // Source: [SID94]
    float net,error=0.0,q1=0.0f;
    float E;
    int k,j,i,n,m,Nt1,Nt2,code,count,IdN,check,ch,recgC;
    float* w;
    TRecogn=0;
}

```

```

w=x;
for(k=0;k<M;k++)
{
    code=0;
    count=0;
    IdN=-101;
    check=0;
    for(i=0;i<NL[1];i++)
    {
        for(j=0,net=theta;j<NL[0];j++)
            net+=w[i+j*NL[1]]*xp[j*M+k];
        E=(float)exp(-(double)Sp*net);
        y[i]=(float)((2.0/(1.0+E))-1);
    }

    Nt1=NL[1];
    Nt2=0;
    for(n=2;n<L;n++)
    {
        for(i=0;i<NL[n];i++)
        {
            m=Nt1+i;
            for(j=0,net=theta;j<NL[n-1];j++)
                net+=w[NS[n-2]+i+j*NL[n]]*y[j+Nt2];
            E=(float)exp(-(double)Sp*net);
            y[m]=(float)((2.0/(1.0+E))-1);
        }
        Nt1+=NL[n];
        Nt2+=NL[n-1];
    }
    recgC=0;
    ch=1;
    for(i=0;i<NL[L-1];i++)
    {
        error=float(d[k+i*M]-y[Nt2+i]);
        q1+=(error*error);
        if(fabs(error)>THold)
        {
            code=-101;
        }
        if(d[k+i*M]>0)
        {
            recgC=recgC | ch;
            count=1;
            IdN=i;
        }
        ch=ch<<1;
    }
    if(code==0)
    {
        if(count !=1)
        {
            IdN=-1;
            check=-1;
        }
        else
            check=1;
        TRecogn++;
    }
    else
        IdN=-101;
    RecogResult[k*3+1]=recgC;
    RecogResult[k*3+2]=check;
}
q1/=2.0;
return q1;
}

void CDelta::ReadWrite(int flag,int tot,char* wname)

```

```

{
    FILE* stream;
    int i;
    float fp;

    if(flag==0)
    {
        if(stream=fopen(wname,"w"))
        {
            for(i=0;i<tot;i++)
            {
                fprintf(stream,"%f ",x[i]);
            }
            fclose(stream);
        }
        else
        {
            Message.Format("CDelta-File %s cannot open for write",wname);
            AfxMessageBox(Message,MB_OK);
            return;
        }
    }
    else if(flag==1)
    {
        if(stream=fopen(wname,"r"))
        {
            for(i=0;i<tot;i++)
            {
                fscanf(stream,"%f",&fp);
                x[i]=fp;
            }
            fclose(stream);
        }
        else
        {
            Message.Format("CDelta-File %s cannot open",wname);
            AfxMessageBox(Message,MB_OK);
            return;
        }
    }
    else
    {
        Message.Format("CDelta-It is not read or write flag number");
        AfxMessageBox(Message,MB_OK);
        return;
    }
}

```

## **VITA AUCTORIS**

Nicholas M. Sandirasegaram was born in 1971 in Jaffna, Sri Lanka. He graduated from Sir John A. Macdonald Higher Secondary School in 1993 in Hamilton, Ontario. From there he went on to the University of Windsor where he obtained a B.A.Sc in Electrical Engineering in 1997. He is currently a candidate for the Master's degree in Electrical & Computer Engineering at the University of Windsor and hopes to graduate in Fall 2000.