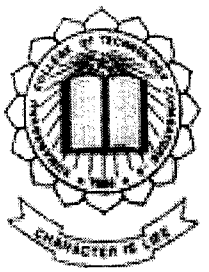
HANDWRITTEN SIGNATURE VERIFICATION



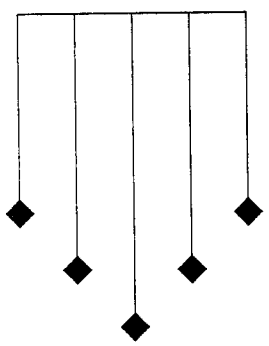
Project Report

Submitted by,

Prasanth Nair P
Priyalata B S
Suguna M

Under the guidance of,

Mrs. V.Vanitha M.E



March - 2002

Submitted in partial fulfillment of the requirements for the award of the degree of Bachelor of Engineering in Computer Science and Engineering of Bharathiar University, Coimbatore

Department of Computer Science and Engineering
Kumaraguru College of Technology
Coimbatore - 641006

Declaration

We, **Prasanth Nair P, Priyalata B.S, Suguna M** hereby declare that this project work entitled **"HANDWRITTEN SIGNATURE VERIFICATION"** submitted to Kumaraguru College of Technology, Coimbatore (Affiliated to Bharathiar University) is a record of original work done by us under the supervision and guidance of **Mrs.V.Vanitha M.E**, Department of Computer Science and Engineering.

Name of the Candidate	Register Number	Signature of the Candidate
Prasanth Nair P	**9827K0196**	
Priyalata B S	**9827K0201**	
Suguna M	**9827K0721**	

Countersigned by:

12/3/2002

Staff in Charge

Mrs. V.Vanitha M.E

Lecturer

Department of Computer Science and Engineering

Kumaraguru College of Technology

Coimbatore – 641006

Place : Coimbatore

Date : 12.03.2002

ACKNOWLEDGEMENT

We are bound to express our deep sense of gratitude to Dr.K.K.Padmanaban PhD, Principal, Kumaraguru College of Technology, Coimbatore, for providing permission to carry out this project work.

With profound sense of gratitude, we acknowledge with great pleasure the guidance and support extended by Prof. Dr. S.Thangaswamy PhD, Head of the Department of Computer Science and Engineering, Kumaraguru College of Technology, Coimbatore, for his valuable and continuous guidance, suggestions, constructive criticisms and persistent encouragement.

We express our deep sense of respectful gratitude to our project guide Mrs.V.Vanitha M.E., Lecturer, Department of Computer Science and Engineering for her valuable guidance, keen suggestions, innovative ideas, inspiration, discussions, helpful criticisms and kind encouragement in all the phases of this project. It had been indeed a great pleasure to work under her guidance.

It is our duty to express our thanks to Mrs. D Chandrakala M.C.A M.E, Senior Lecturer, Department of Computer Science and Engineering who has been an all time encouragement not only throughout the project but also during the entire course.

We also extend our heartiest thanks to Mrs. S Devaki B.E M.S, Assistant Professor, Department of Computer Science and Engineering, Kumaraguru College of Technology for providing us her support which really helped us come out with the project successfully.

We like to express our special thanks to all persons in the programming laboratory who helped us for the successful completion of the project. Finally we express our deep sense of gratitude to our parents, friends, and all other persons who were directly or indirectly involved with this project, for their invaluable help and consideration towards us.

SYNOPSIS

Authentication has become an essential part of highly computerized services and security-sensitive installations in the modern society. A number of techniques researched over the past decade checks on the person's face, voice, iris or fingerprint. For optimal security, practical systems usually apply at least two different authentication techniques. However, the techniques tend to require additional measurements to be usable or inoffensive to the person involved.

A popular means of authentication historically has been the Handwritten Signature. Handwritten Signature becomes one of the most common ways to authorize transactions and authenticate human identity. Handwritten signature recognition using neural networks provides a new feature in the transactions security system. It is obvious that no two persons would have similar signature in the world. Hence, the signatures of the persons are distinctive in nature.

The neural system is trained to understand the handwritten signatures of different persons for the task of signature recognition. The trained neural network recognizes the name of the person to whom the given signature belongs and also detects untrained signatures of the individual. The network is a Multilayer Feed Forward Network. The weights of the network are adjusted according to the Back propagation algorithm.

The Signatures are entered into the computer by a scanner in a fixed size and then preprocessed to feed the neural network. Neural network structures are constructed and simulated using java due to its portability feature. The performance of the system is examined and verified. The trained network is tested by giving the known and unknown signatures.

Signatures are used everyday to authorize the transfer of funds for bank cheque, credit cards, legal documents and others. This system can be implemented as an additional security feature for all kinds of business transactions and also for human identification.

Table of Contents

4. Design Document

5. Product Testing

Appendix

1.1 Handwritten Signature Verification

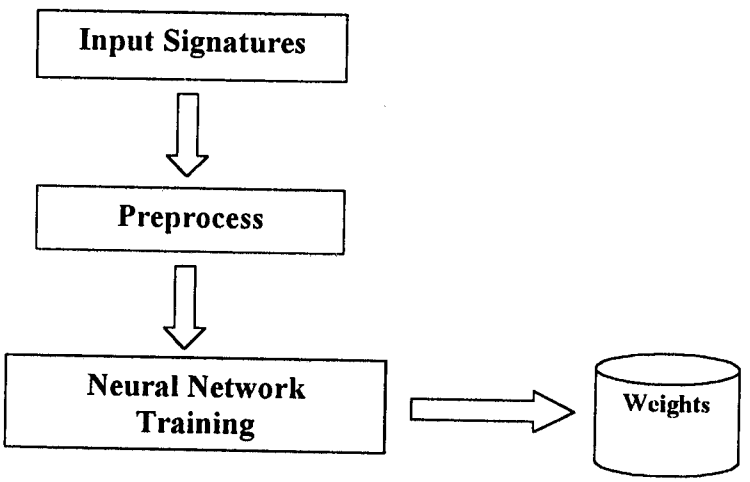
Our society is increasingly dependent on electronic storage and transmission of information, this has created a need for electronically verifying a person's identity. Nowadays, the bulk of monetary transactions are credit transactions not involving the exchange of cash or checks. The magnitude of this problem increases as the percentage of transactions rises. Also in an Industrial Security Control System, the security personnel find it difficult to identify a person using his Identity Card. One common form of identification is the signature.

Handwritten signatures have been the normal and customary way for identity verification. As an increasing number of transactions, especially financial, are being authorized via signature, methods of automatic signature verification must be developed, if authenticity is to be verified on a regular basis. Approaches to signature verification fall into two categories according to the acquisition of the data: *On-line* and *Off-line*. On-line data records the motion of the stylus while the signature is produced, and includes location, and possibly velocity, acceleration and pen pressure, as functions of time. Off-line data is an image of the signature. Although On-line verification is of great interest for "point of sale" applications, our project focuses primarily on Off-line verification which is of interest to check clearing houses, tax processing centers and other locations where a hard copy of the signature is obtained. The advantage of these systems is that they do not need specialised hardware to capture signature information at the point of signing.

In the Handwritten Signature Verification System developed, eight signatures are collected from each person. Out of these, five are used for training the network and the remaining are used for testing.

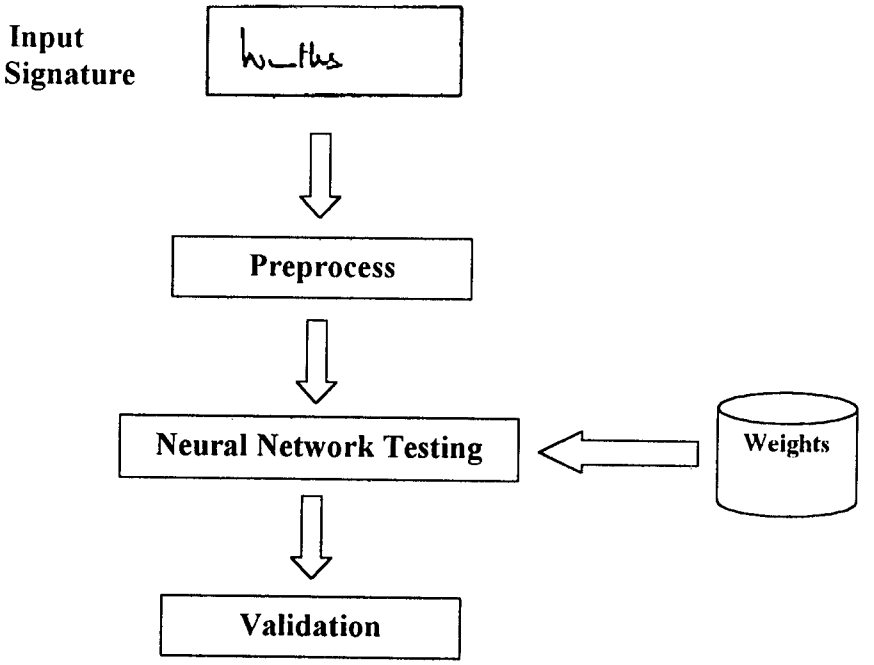
Signatures are scanned in a fixed resolution at 100dpi by a scanner to produce a black and white GIF image. The signatures are preprocessed to determine the input vectors for training and testing phases of the neural network.

Training Phase



During the training of the neural network, the sample set of signature images are preprocessed and the preprocessed data is given as input to the neural network. The neural network weights are adjusted using backpropagation algorithm and the final weights are stored.

Testing Phase



During testing, the signature of a person is first preprocessed and the preprocessed data is fed to the neural system. The neural network weights are initialized based on the weights obtained during the training phase. The trained system will now be able to recognize the signature and check whether the signature is valid or not.

If the signature is true, then the system display's the name of the person, otherwise the system indicates that the signature given doesn't match with any of the trained signatures.

Handwritten Signatre Verification has important areas of application, for example, in automatic cheque clearing, credit cards, legal documents and others. This system can be implemented as an additional security feature for all kinds of business transactions and also for human identification, which is of interest to check clearing houses, tax processing centers and other locations.

1.2 Introduction to Neural Networks

1.2.1 Neural Networks

a) Introduction

The human nervous system consists of very large number of neurons, which are connected to each other at staggered complexity. An estimated 10^{11} neurons participate in 10^{15} interconnections over transmission paths that may range for a meter or more. Each neuron shares many characteristics with other cells in the body but has unique capabilities to receive, process and transmit electrochemical signals over the neural pathway that comprise the brains communication system these neurons are used to sense impulses form various parts of our body. The brain in turn processes these activities and sends the result to the concerned part of the body to the neurons themselves.

Artificial neural network is the mimic of the human brain. Artificial neural network's (ANN) are composed of elements that perform in the manner that is analogous to the most elementary functions of the biological neuron. The connected networks of simple computational elements of neural networks are used to carry out complex cognitive and computational task. The word neural is nearly indicative of the original inspiration for ANN as abstract models of brain function. ANN employ high stylized units (originally called neurons) and connection's (originally called synapses).

Activations are computed locally and concurrently, therefore ANN are also called as "parallel distributed processing models" as each input pattern activates multiple hidden unit's, and each hidden unit may be activated by multiple input patterns is called distributed representation.

ANN is a network of many simple processes (units) ,each possibly having a small amount of local memory the units are connected by communication channels (connections) which usually carry numeric (as opposed to symbolic) data, encoded by any of various means. The units operate on the local data and on the input's they receive via the connections. They are also called as non-programmed adaptive information processing systems or connectionist models or neuromorphic systems.

ANN resembles the human brain in two ways:

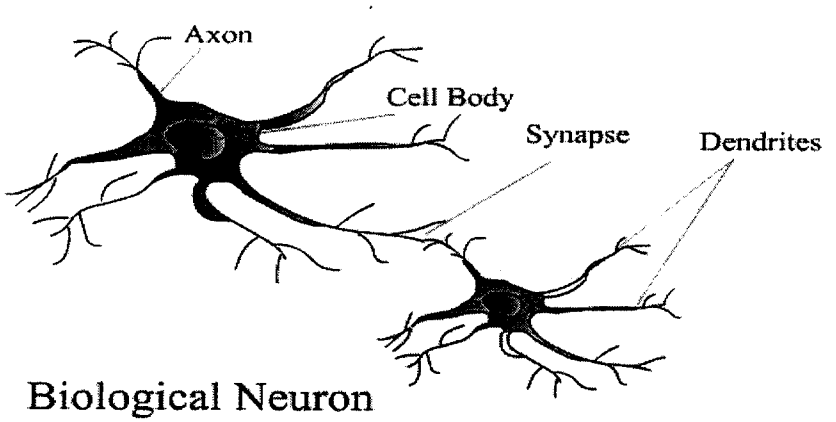
- Knowledge is acquired by the network through a learning process.
- Interneuron connection strengths known as synaptic weights are used to store the knowledge.

b) Model of Biological and Artificial Neurons

Biological Neurons

The elementary nerve cell, called neuron, is the fundamental building block of the biological neural network (human brain).

The schematic diagram of a neuron is shown in the following figure



Biological Neuron

A typical cell has 3 major regions:

- The cell body which is also called the soma
- The axon
- The dendrites

Dendrites form a dendric tree which is a fine bush of thin fibers around neurons body. Dendrites receive information from neurons through axons – long fibers that serve as transmission lines. An axon is a long cylindrical connection that carries impulses from the neuron. The end part of an axon split's into a fine arborization; each branch of it terminates in a small end bulb almost touching the dendrite of the neighboring neurons. The axon dendrite contact organ is called synapse. The synapse is where the neuron introduces its signal to the neighboring neuron.

The signals reaching a synapse and received by dendrites are electrical impulses. The inter-neuronal transmission is sometimes electrical but is usually effected by the release of the chemical transmitters at the synapse.

The neuron is able to respond to the total of its inputs aggregated within a short time interval called the period of latent summation. The neuron's response is generated if the total potential of its membrane reaches a certain level. Specifically the neuron generates a pulse response and sends it to its axon only if the conditions necessary for firing are fulfilled.

Artificial Neuron

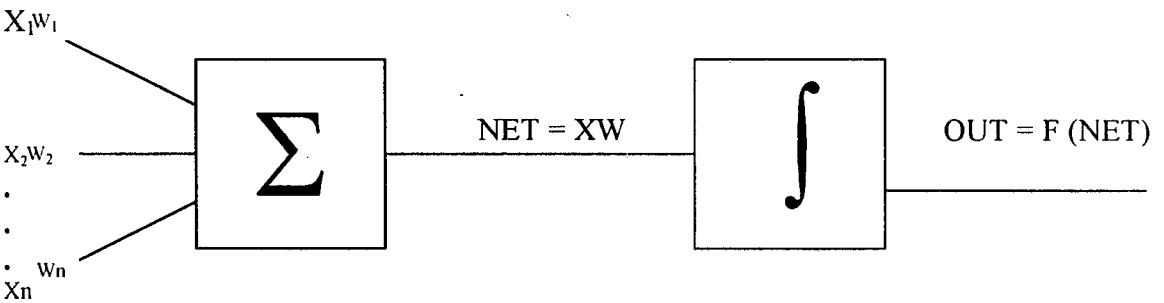
The artificial was designed to mimic the first order characteristics of the biological neuron. An artificial neuron is an information processing unit that is fundamental to the operation of the neural network. The basic elements of the artificial neuron are described below:

- A set of *synapses* or connecting links, each of which is characterized by a weight or the strength of its own (analogous to synaptic strength).
- An *adder* for summing the input signals weighted by the respective synapses of the neuron (analogous to the biological cell body).
- *Activation functions* for limiting the amplitude of the output neuron.

In essence a set of the inputs are applied, each representing the output of another neuron. Each is multiplied by a corresponding weight and all of the weighted inputs are then summed to determine the activation level of the neuron.

A bias term with input as +1 or –1 is given to the neural networks so as the system converges faster. Connections with the positive weights are called "excitatory" and connections with negative weights are called as "inhibitory".

The model of an artificial neuron is shown in the following figure



Model of a Neuron

X - Input Vector W – Synaptic Weights Vector

$X_1, X_2, X_3 \ldots Xn$ represent input values and each signal is multiplied by an associated weight $W_1, W_2, W_3 \ldots\ldots\ldots Wn$ before it is applied to the summation block produces an output called net. This may be stated in vector notation as follows

$$NET = XW$$

c) Benefits of Neural Networks

- **They are *adaptive*:** They can take data and learn from them .thus they infer solutions from the data presented to them, often capturing quite subtle relationships. This ability differs radically from standard software techniques because it does not depend on the programmer's prior knowledge of rules. Neural networks can reduce development time by learning underlying relationship even if they are difficult to describe. They can solve problems that lack existing solutions.

- **They can *generalize*:** They can correctly process data that only broadly resemble data they were trained originally. Similarly, they can handle imperfect or incomplete data, providing a measure of fault tolerance. Generalization is useful in practical applications because real world data is noisy.

- **They are *non-linear*:** They can capture complex interactions among the input variables in a system. In a linear system changing the single input produces a proportional change in the output, and the effect of the inputs depends on it's own value. In the non-linear system, the effect depends on the values of the other inputs.

- **They are *highly parallel*:** There are numerous identical, independent operations which can be executed simultaneously. Parallel hardware executes hundred's or thousand's times faster than the conventional microprocessor and digital signal processors.

- ***Simple processors*:** Each processor simply sums incoming signals and fires when these signals reach threshold so no complex computations are needed.

- ***Collective computations*:** A neural network does not execute individual instructions rather; all the nodes in the network are trained to solve a particular problem collectively.

d) Activation Functions

The NET signal form the network, is usually further processed by an activation function F to produce the neurons output signal OUT. The activation functions are used to compress the range of NET so that OUT lies between certain limits.

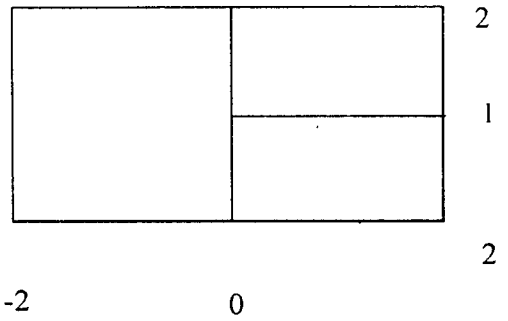
The following are some of the activation functions used in neural network system.

- Hard limiter
- Sigmoid function.

1. Hard Limiter

This is of the type OUT = $\begin{cases} 1 \text{ if NET} > 0 \\ 0 \text{ if NET} <= 0 \end{cases}$

where NET is the net sum from the summation unit.



(a) Hard Limiter Function

2. Sigmoid function

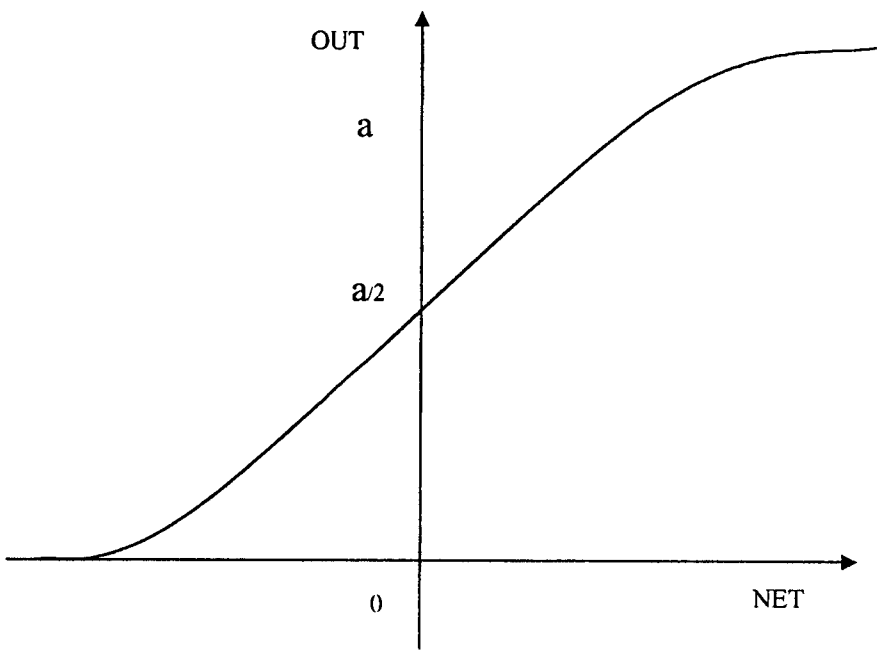
Logistic function

$OUT = 1/ (1+exp^{(-NET)})$

$OUT = 1/(1+0) = 1$

$OUT = 1/(1+ \alpha) = 0$

The values varies from 0 to 1



(b) Sigmoidal Logistic Function

e) Classification of Neural Networks

There are mainly two types of architecture of ANN: Feed Forward neural network (FFNN) and recurrent neural networks (RNN) as shown below:

1. Feed Forward Neural Network

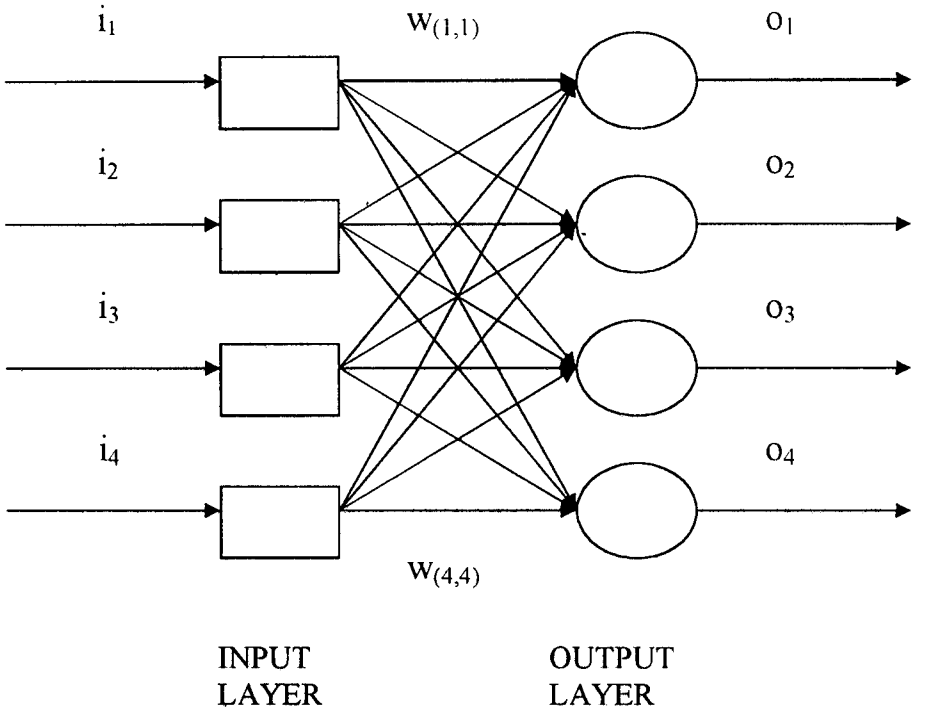
- Single layer feed forward network.
- Multi layer feed forward networks.

2. Recurrent networks.

In FFNN, there is no feed back loop. The flow of signal is only in the forward direction. The behavior of FFNN does not depend on past input. The network responds only to its present input. In RNN, there are feedback loops (essentially FFNN with output fed back to input).

Single Layer Feed Forward Networks

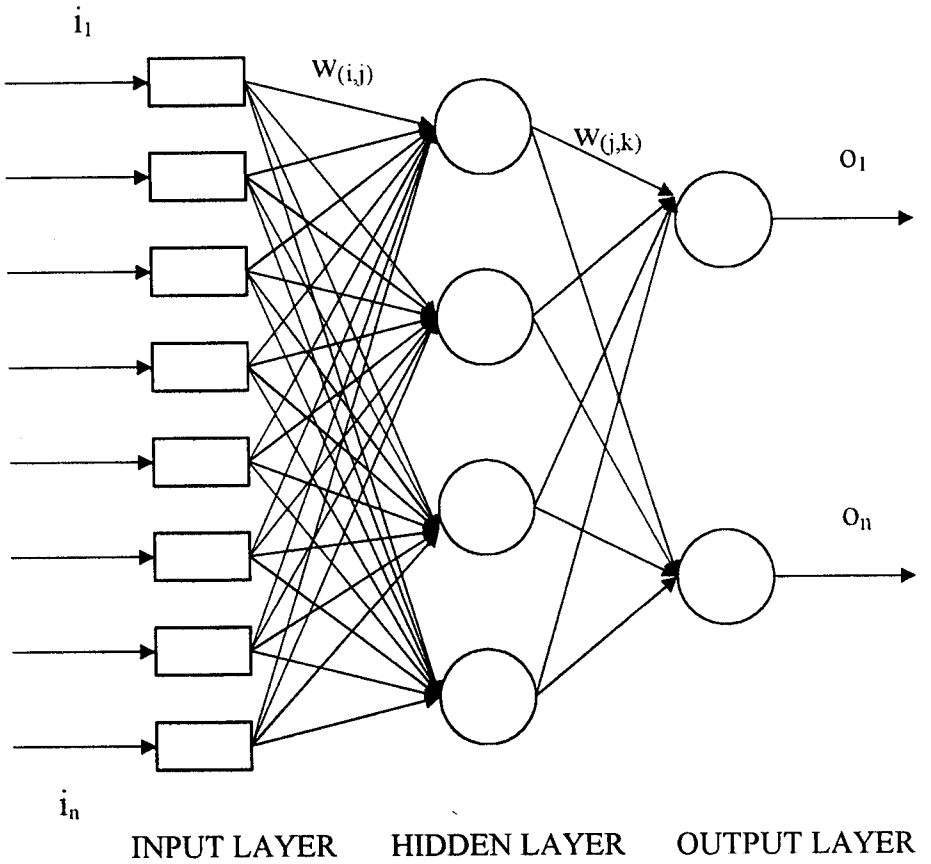
In the layered neural network, the neurons are organized in the form of layers. In the simple form of layered network we have an input layer of source nodes that projects on to the output layer of neurons. The single layer here represents the output layer of neurons.



Single Layer Feed Forward Network

Multilayer Feed Forward Networks

It is a feed forward network with one or more hidden layer. The source nodes in the input layer supply inputs to the neurons of the first hidden layer. The outputs of the first neurons are applied as inputs to the neurons of the second hidden layer and so on. Of every node in each layer of the network is connected to every other adjacent forward layer, and then the network is called fully connected. If however some of the links are missing the network is said to be partially connected. These networks can be used to realize complex input /output mapping.



Multi-Layer Feed Forward Network

Recurrent Networks

A recurrent neural network is one of which is at least one feed back loop there are different kind's of recurrent networks, depending on the way in which the feedback is used. In a typical case it has single layer of neurons. Other kinds of recurrent networks may have self-feed forward loops and also hidden neurons.

f) Training and Testing of a Network

i) Training

A neural network is trained so that the application of a set of inputs produces the desired (or at least consistent) set of outputs. Each such input (or output) set is referred to as a vector. Training is accomplished by sequentially applying input vectors.

While adjusting network weights according to the predetermined procedure. During training the network weights gradually converges to various values that each input vector produces the desired output vector.

The process of training a neural system is like studying a new subject. Both the process and the outcome of any operation will be studied while learning a new subject. This is a time consuming process, likewise while training a system both the input and its corresponding output are given supervised mode.

The training of neural network is classified into two ways they are

- Supervised training
- Unsupervised training

Supervised Training

Supervised training requires the paring of each input vector with a target vector representing the desired output, together these are called "training pair". A network is usually trained over a number of such training pair's. An input vector is applied and the output of the network is calculated and compared to the corresponding target vector, and the error is fed back through the network and weights are changed according to an algorithm that tends to minimize the error.

Unsupervised Training

Unsupervised training requires no target vector for the outputs and hence no comparisons to predetermined ideal responses. The training set consists only of the input vectors. The training algorithm modifies network weights to produce output vectors that are consistent, that is either application of one of these training vectors or application of a vector that is sufficiently similar to it will produce the same pattern of outputs. The training process therefore, extracts the statistical properties of the training set and group's similar vectors into classes. Applying a vector from the given class to the input will produce a specific output vector, but there is no way to determine prior to training which specific output pattern will be produced by a given input vector class. Hence the outputs of such a network must generally be transformed into a comprehensible form subsequent to the training process.

Ways of Training Neutral Network

It is broadly classified into two types. Along the types given below any one of the training methods is used to train a system.

The two types are:

1. Online or single pattern training
2. Batch or Epoch training

Online Training

In single pattern training, the error is back propagated and the weights are adjusted for every testing pattern. The network just learns to generate an output close to the output for the current pattern without actually learning anything about the entire training set. This type of training is more expensive because the weights are updated for each and every pattern individually.

Batch Training

In Epoch learning, weights are updated only after all the samples are presented to the network. An epoch consists of such a presentation of the entire set of the training samples. Weight changes suggested by different training samples are accumulated together into a single change to occur at the end of each epoch.

This epoch training can be implemented using parallelism in order to reduce the mount of training time of the network.

ii) Testing

The process of testing a neural network is like answering questions after the subject is through. If a person is through in that subject then he will take a small time to answer a question poised to him. Similarly the neural network will also take only a short time to answer a question poised to it. It is to be noted that only the questions are poised to the person and not the answer. Similarly, during testing only the input to the neural system is given.

1.2.2 Backpropagation Networks

a) Introduction

For many years there was no theoretically sound algorithm for training multilayer artificial neural network. Since single layer networks provided severely limited in what they could represent the entire field went into virtual eclipse.

Backpropagation is a systematic method for training an artificial neural network and it is a very popular model. It does not have a feedback connection, but errors are back propagated during training. It has a mathematical foundation that is strong if not highly practical. Despite its limitations, backpropagation has dramatically expanded the range of applications to which neural networks can be applied and it has generated many successful demonstrations of its power.

Backpropagation is useful in problems requiring recognition of complex patterns and performing nontrivial mapping functions. Backpropagation networks are formalized first by Werbos (1974). This network is designed to operate as a multilayer, feedforward network using the supervised mode of training.

b) Architecture of Backpropagation Network

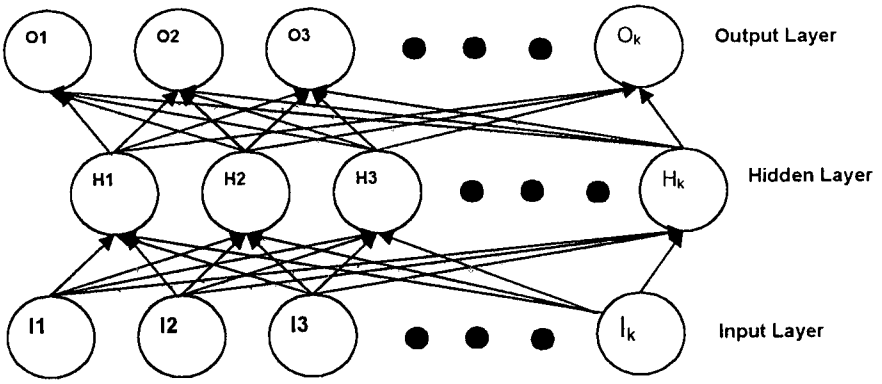
A Backpropagation network is made from interconnected nodes arranged in at least three layers. The three layers are called as input layer, hidden layer and output layer. The input layer is passive; it merely receives the data patterns passing into the network. The number of input nodes consequently equals the number of measured data values (vector components) presented to the network. Unlike the input layer, the output and the hidden layer both process data actively. The output layer as its name suggests produces the network result. In the back propagation network, the result is a set of continuously variable values (output vectors) one value per output node.

A single node has many inputs but only one output .Each input is a single data value presented to the node, usually thought a connection from a previous layer. An extra input called the threshold input acts as the reference level or basis for the processing element.

Associated with each connection is an adjustable value called weight. Basically a node calculates the weighted sum of its inputs, and then passes the sum through an activation function to produce a result.

The Backpropagation algorithm's strength is in its ability to change the value of its weights in response to errors. It does this automatically during training: hence, training requires a series of input patterns tagged with their desired output patterns.

A typical structure of back propagation network



c) Backpropagation Algorithm

The following is the pseudo code of the algorithm:

```
BPN( )
{
    ALLOCATED SPACE
    READ INITIAL WEIGHTS AND INPUT PATTERNS
    for(q=0; q<Number of iterations; q++)
     {
      for(p=0; p< Number of patterns; p++)
        {
        COMPUTE ACTIVATIONS
        PROPAGATE ERROR SIGNALS
        }
```

```
ADAPT WEIGHTS
        If (mean_squared_error < minimum_error)
        break;
}
WRITE FINAL WEIGHTS
WRITE OUTPUT VALUES
FREE STORAGE
}
```

Algorithm:

The following is the algorithm that can be used to train a back propagation network.

Let A be the number of units in the input layer, B be the number of units in the hidden layer and C be the number of units in the output layer. The activation levels of the input layer are denoted by X(j), in the hidden layer by H(j) and in the output layer by O(j).

Weights connecting the input layer to the hidden layer are denoted by W1(i,j) where the subscript 'i' indexes the input units and 'j indexes the hidden units. Likewise, weights connecting the hidden layer to the output layer are denoted by W2(i,j) with 'i' indexing the hidden units and 'j' indexing the output units.

1. Initialize the weights in the network. Each weight should be set randomly to a number between –1 and +1.

   W1(i,j) = random(-1,+1) for all i = 0 to A, j = 0 to B

   W2(i,j) = random(-1,+1) for all i = 0 to A, j = 0 to B

2. Initialize the activation of threshold units. The values of the threshold units should never change.

   $X(0) = 1.0$

   $H(0) = 1.0$

3. Choose an input-output pair, the input vector X(i) and the target output vector Y(i)

4. Propagate the activations from the units in the input layer to the units in the hidden layer using the activation function

   $$H(j) = 1/ (1 + e^{(-W1(i,j) * X(i))})$$

   where sum ranges from i = 0 to A for all j = 1 to B

5. Propagate the activation from the units in the hidden layer to the units in the output layer using the activation function

   $$O(j) = 1/ (1 + e^{(-W2(i,j) * H(i))})$$

   where sum ranges from i = 0 to A for all j = 1 to C

6. Compute the errors of the units in the output layer, denoted by d2(j). Errors are based on the network's actual output O(j) and the target output Y(j)

   $d2(j) = O(j) * (1-O(j)) * (Y(j) - O(j))$ for all $j = 1,2,\ldots,C$

7. Compute the errors of the units in the hidden layer, denoted by d1(j).

   $$d1(j) = H(j) * (1-H(j)) * \Sigma (d1(i) * W2(j,i))$$

   where sum ranges from $i = 1,2,\ldots,C$ and $j = 1,2,\ldots,B$

8. Adjust the weights between the hidden layer and the output layer

   $dW2(i,j) = h * d2(j) * H(i)$ for all $j = 1,2, \ldots,C$

9. Adjust the weights between the input layer and the hidden layer

   $dW1(i,j) = h * d1(j) * X(i)$ for all $j = 1,2, \ldots,B$

10. Go to step 3 and repeat

When all the input-output pairs have been presented to the network, one epoch has been completed. Repeat steps 4 thru 10 for as many epochs as possible.

d) Training of Backpropagation Network

The type of training used is supervised training where the system is trained with both the input and their corresponding outputs.

Before starting the training process, all of the weights must be initialized to small random numbers. This ensures that the network is not saturated by large values of the weights and prevents certain other training pathologies. For example, if all the weights start at equal values and the desired performance requires unequal values, the network will not learn.

There are two phases present in the training of backpropagation network.

Forward Phase

Here, the input vector X is applied to the input nodes of the network. It propagates through the network, layer by layer and an output vector Y is produced. The input vector X and the target vector T are coming from the training set. During this phase, the synaptic weights of the network are all fixed.

Reverse Phase

In this phase, the output pattern is compared to the desired output and an error signal is compared for each output unit.

The error signals are then transmitted backwards from the output layer to each node in the intermediate layer that contributes directly to the output. However, each unit in the intermediate layer receives only a portion of the total error signal, based roughly on the relative contribution the unit made to the original input. This process repeats layer by layer, until each node in the network has received an error signal that describes its relative contribution to the total error. Based on the error signal received, connection weights are then updated by each unit to cause the network to converge toward a state that allows all the training patterns to be encoded.

The significance of this process is that as the network trains, the nodes in the intermediate layers organize themselves such that different nodes learn to recognize different features of the total input space. After training, when presented with an arbitrary input pattern that is noisy or incomplete, the units in the hidden layers of the network will respond with an active output if the new input contains a pattern that resembles the feature the individual units learn to recognize during training. Conversely, hidden layer units have a tendency to inhibit their outputs if the input pattern does not contain the feature that they were trained to recognize.

As the signals propagate through the different layers in the network, the activity pattern present at each upper layer can be thought of as a pattern with features that can be recognized by units in the subsequent layer. The output pattern generated can be thought of as a feature map that provides an indication of the presence or absence of many different feature combinations at the input. The total effect of this behavior is that the backpropagation provides an effective means of allowing a computer system to examine data patterns that may be incomplete or noisy and to recognize subtle patterns from the partial input.

The equation that changes the weights – called the *delta rule* is designed to minimize the network's mean-squared error. The network's overall accuracy is improved by the aggregate corrections during training. When the network can process input patterns with sufficient accuracy, the weights are saved to preserve what it has learned.

Activation Function Used

Sigmoid function is used for backpropagation network. Sometimes called a logistic or simply squashing function, the sigmoid compresses the range of NET so that OUT lies between 0 and 1. Multilayer network have greater representational power than single layer network only if a non linearity is introduced. The squashing function produces the needed non-linearity.

The Backpropagation Algorithm requires only that the function be every where differential. The sigmoid function satisfies this requirement. Sigmoid function has the additional advantage of providing a form of automatic gain control. For small signals (NET near zero) the slope of the input/output curve is steep, producing high gain. As the magnitude of the signal becomes greater, the gain decreases. In this way lot of signals can be accommodated by the network without saturation, while signals are allowed to pass through without excessive attenuation.

Factors Affecting Backpropagation network

1) The main disadvantage of backpropagation network is the likely occurrence of local minima. Local minima can be avoided in the following methods.
   i. Reduce the gain term or learning co-efficient.
   ii. Usage of extra hidden units.
   iii. Increase of momentum factor.
   iv. Start training from different weights.

2) Selection of number of hidden neurons is still a trial and error method. Much number of hidden nodes may produce better outputs for the trained examples then the network with less hidden nodes, but it may produce worse outputs for inexperienced inputs.

3) Number of training patterns used must be high.

4) The initialization of weights strongly affects the ultimate solution. If all the weights are equal and the desired performance requires unequal values, the network will not train. If all the weights are equal to zero, the network will not be trained.

5) If learning rates are very small, the change in weights will be very small and it will need more iterations to converge. If learning rates are high, it speeds up learning and weights will assume a form that will make the network unstable i.e. oscillatory.

6) Choice of momentum: Momentum is used for improving the training time of the backpropagation algorithm, while enhancing the stability of the process. This method involves adding a term to the weights adjustment that is proportional to the amount of the previous weight change. Once an adjustment is made its "remembered" and serves to modify all subsequent weight adjustments.

System Requirements

2.1 Product Definition

2.1.1 Problem Statement

The product is concerned with the recognition of handwritten signatures of individuals for the purpose of authenticating human identity. The product should have the capability to recognize the untrained signature of a person based on trained samples of the persons signature.

2.1.2 Processing Environment

Hardware Specifications

Processor	Pentium III 550 MHz
Floppy Disk Drive	1.44 MB
CD ROM Drive	48X
Hard Disk	6.4 GB
Mouse	Microsoft compatible PS/2 mouse
Keyboard	Windows Keyboard
RAM	128MB
Display Adapter	On board SVGA card with 8 MB VRAM
Scanner	USB compatible 600 by 600 dpi resolution

Software Specifications

Language Used	Java
Tools Used	JCreator (Java IDE)
RDBMS	Oracle 8.0
Operating System	Windows 98/Me

2.1.3 User Characteristics

The product is mainly used by the administrator of the organization using it as a security measure, but it can also be used by the clients in a restricted environment. The software is designed to be a user friendly interface product. Necessary level of help features are provided at each interface screen of the product.

2.1.4 Solution Strategy

The problem was approached in a step by step fashion. The first and foremost step was to analyze and draft the modules of the project based on the problem definition. The solution ends up in breaking down the problem into three distinct modules namely preprocessing, neural network training and neural network testing.

2.1.5 Product Features

The product is developed both for client and the administrator and its display is developed to be user friendly. The product has its own features like accessing the dialog boxes for data entry.

The product has its own security feature. The console client/administrator can use the product only after entering the correct password. Each interface in the software is provided with the option to exit the software. Additionally each interface also has an associated help documentation which can be accessed at the click of a button.

2.1.6 Acceptance Criteria

The product in the training phase has to accept the sample set of signatures of an individual or individuals, preprocess and pass it to the neural network for training the network. In the testing phase it has to accept the signature of an individual, preprocess it and test it using the neural network.

2.2 Project Plan

2.2.1 Development Schedule

In order to complete the project in a given time, the development schedule is framed and based on the time slots, the product is developed. The preprocessing of the signature is the first step in the coding process. The very next stage is the creation of neural network structure and employing back propagation algorithm in the network. Finally the user interface for the product is developed. Finally, all modules are properly interlinked together.

Software Requirements Specification

3.1 Introduction

a) Purpose

The purpose of this SRS is to elaborate the requirements of Handwritten Signature Verification System. The system verifies the handwritten signature of individuals. The scanned images of individual person serve as the object to be verified.

b) Scope

The scope of the SRS is mainly focused on the requirements of the Handwritten Signature Verification system. The well structured and descriptive nature of the SRS is aimed at aiding the developers in developing an efficient Signature Verification System.

c) Overview

The further exploration of the Software Requirement Specification gives information about specific requirements such as functionality, usability and supportability of the system.

3.2 Overall Description

In general perception Handwritten Signature of each person is unique. This product Handwritten Signature verification system provides a way of authenticating a person based on his/her signature. The product is to act as a secondary authentication system in establishments.

A sample application where a bank client could access his account information with handwritten signature verification as an authentication measure is to be implemented. The product basically should employ an appropriate neural network model to identify the signature of an individual. The network should be capable of effectively recognizing the signatures of individuals.

The input to the system is a scanned image of signature of an individual. Typical areas of application are organizations where there is a need for added security such as bank, research establishments, business centers etc.

3.3 Specific Requirements

a) Functional Specification

The 'Handwritten signature verification' comprises of three main modules based on the function they perform. The three main modules are as follows:

- Preprocessing
- Neural Network Training
- Neural Network Testing

i) Preprocessing

The scanned gif image acts as input to preprocessing. The signature alone is first extracted from the image. The signature is translated both on the left boundary as well as top. The image is then transformed to a standard size of 200 by 75 pixels. The image is further broken down into 30 sub images of fixed size. The black pixel ratio in the sub images are computed.

ii) Neural Network Training

The data from the preprocessing stage serves as input to the neural network. The network first trains the signature of individuals from a sample set of signatures of each person. The network weights are balanced according to the input patterns. The weights obtained from training the network are stored for the testing module.

iii) Neural Network Testing

The testing constitutes initializing the neural network with the computed weights from the training module and obtaining the appropriate identification code for an individual to authenticate him.

b) Performance Requirement

The performance of the system depends on optimal training cycles for the train set of signatures. The performance deteriorates when the network is under trained or over trained. Optimal training cycles have to be initiated for better performance.

c) Exception Handling

During programming there is a chance to come across some exception condition. These exception conditions should be handled with care otherwise the whole software will crash.

d) External Interfaces

User interface is Graphical based for better interpretation. Ample dialog boxes and help features are provided at each user interface.

Design Document

4.1 External Design Specifications

The handwritten signature verification system is implemented along with a sample banking application. The external design specifications of the application and the signature verification system are discussed.

The user interface has a series of user interface screens leading to the training and testing modules of the project. There are two types of users in the application developed namely, client (a bank customer) and administrator (bank official). There are two different sets of interface one each for the client and the administrator.

The user interfaces for the client are:

- Account verification
- Signature testing
- Access account

i) Account Verification

The client inputs his name and account number through this interface. Only if both match he is granted access to the Signature testing interface.

ii) Signature testing

The test signature of the person is input here. The signature is tested and if authenticated, access is granted to view his account information.

iii) Access account

The user is given access in to the details of this bank account.

The user interfaces for the administrator are:

- Password Verification
- Administrative Tools
- Training Signatures
- Testing Signatures

i) Password Verification

The administrator inputs his name and password through this interface. Only if both match he is granted access to the administrators area.

ii) Administrative Tools

There are options to choose like training, testing, adding/deleting administrators, deleting clients etc., in this interface.

iii) Training Signatures

The sample set of signatures of each individual are fed here and the network trained for those signatures. The weights are stored in the file.

iv) Testing Signatures

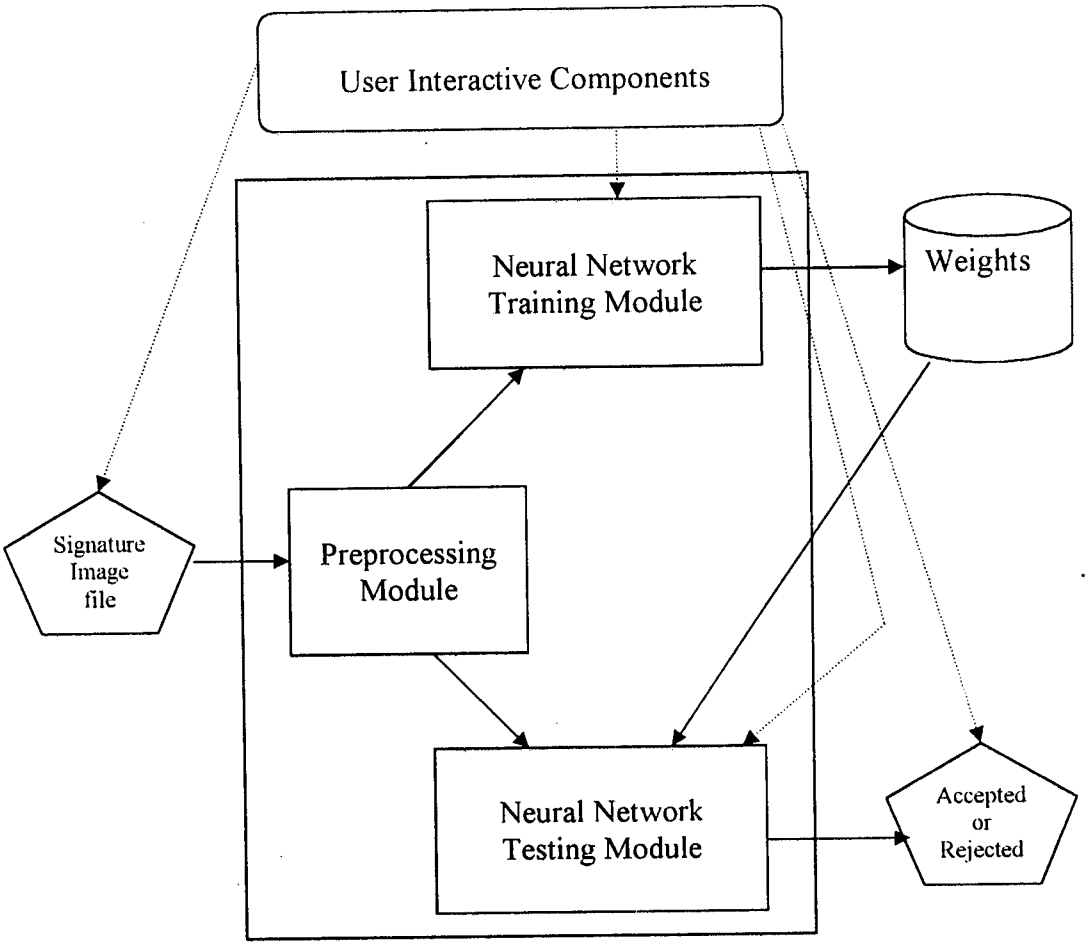
The signature of any existing client can be tested here.

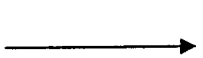
4.2 Architectural Design Specifications

4.2.1 Structure Diagrams

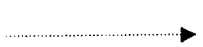
Structure diagram explain the details about the product structure. What are the inputs and the outputs to the structure, how the product is comprised into three modules and its name is also mentioned.

Structure Diagram

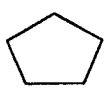




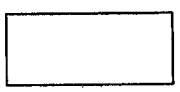
Data flow



User interactive component



Input/output



Processing module

4.3 Detailed Design Specification

4.3.1 Pseudo code for each routine

i) Preprocessing Module

Step 1: The gif image is converted into an array of pixels.

Step 2: The enclosing rectangular boundary is eliminated and a new array containing the signature alone is created.

Step 3: Both top and left translation is done on the new pixel array.

Step 4: The image is transformed to a standard size.

Step 5: The image is split into 30 sub image arrays.

Step 6: The percentage of black pixels in each of these array is computed.

ii) Neural Network Training

Step 1: The network structure is created and the weights randomly initialized.

Step 2: The output pattern is initialized and the test pattern is fed to the network.

Step 3: The network weights are balanced for the optimal number of cycles or till the mean square error limit is achieved.

Step 4: The weights finally are stored in a file.

Step 5: The network structure configuration information also is stored.

iii) Neural Network Testing

Step 1: The network is configured based on the configuration information from the train module.

Step 2: The weights are also initialized from the weights of the training module.

Step 3: The input as well as the output patterns are initialized and the input pattern is tested.

Step 4: The result obtained from the output neurons, is checked with the database to validate the person.

Product Testing

5.1 Unit Testing

Unit testing comprises the set of tests performed by an individual programmer prior to integration of the unit into a larger system. In this testing, individual components are tested to ensure that they operate correctly. Each component is tested independently, without the interference of other system components. Unit testing comprises functional testing as well as performance testing which have been carried out on the system.

i) **Type of test**: Functional Test

Test assumption: Empty Client name

Requirements being tested: Signature Testing

Expected outcome:

If the user does not enter client name the system should respond with the message box prompting the user to fill the client name.

Actual outcome:

The system displays the message box prompting the user to fill the client name.

ii) **Type of test**: Functional Test

Test assumption: Incorrect Client name

Requirements being tested: Signature Testing

Expected outcome:

If the user enters incorrect client name the system should respond with the authentication failed message.

Actual outcome:

The system displays the message box stating that the authentication process has failed.

iii) **Type of test**: Functional Test

Test assumption: Signature file not present

Requirements being tested: Signature Testing/Training

Expected Outcome:

If the user enters a file that is not present, system should respond with a message that file is not present or invalid filename.

Actual outcome:

The system displays the message box stating that the file is not present or invalid filename.

iv) **Type of test**: Performance Test

Test assumption: Time taken for authentication within 15 seconds.

Requirements being tested: Signature testing

Expected outcome:

A signature is tested on the already trained network and the time taken for authentication is noted.

Actual outcome:

The time taken for testing a signature is usually less than 15 seconds.

5.2 System Testing

Software system testing can be looked upon as one of the many processes. This is the last opportunity to correct any flaws in the developed system. System testing involves two kinds of activities. They are: integration testing and acceptance testing.

Each module of the system is tested, each subsystem is tested and the entire system is finally tested in integration testing. During each stage of implementation the tests are carried out. The modules are tested and integrated as they are developed.

Acceptance testing involves planning and execution of functional tests, and performance tests to verify that the implemented system satisfies its requirements.

Future Enhancements

6. Future Enhancement

- **Signature evolution** – A real system should be able to cope up with the signatures that change over time. The system must learn new genuine signatures to follow the evolution of human signatures over time.

- **Size of the training and test sets** – Methods are required to generate variations of a person's authentic signature within the expected percentage of variation for that person.

- **Investigating** other network structures and learning algorithms.

- **Collection of signatures** – It should deal with different size of signature images produced with any writing tool on any kind of paper (plain, patterned, colored) as found in different kinds of checks and documents.

7. Conclusion

The main objective of this project is to recognize the handwritten signature in an offline environment. To do so, an artificial neural network employing backpropagation learning algorithm has been constructed and trained on a training set of signatures.

The neural network has been tested for known and unknown signatures. The results have been quite good.

It could be concluded that

- Backpropagation networks easily learn the overall shape of the signature and hence can verify them.
- Increasing the learning rates decreases the learning time of the network, but the network retaining power also decreases.
- A limited number of signatures were sufficient to train the network.
- Backpropagation networks are sensitive to various shapes of signatures.

8. References

- Carl.G.Looney, *Pattern Recognition using Neural Networks: Theory and algorithms for Engineers and Scientists*, Oxford university press, 1997 Ed.
- Simon Haykin, *Neural Networks: A Comprehensive Foundation*, Pearson Education, 1999 Ed.
- S.K.Hungenahally and L.C. Jain, *Neuro-Intelligent Systems: An introduction to Vision Systems and Artificial Neural networks*, BPB Publications, 1994 Ed.
- Hagan, Demuth and Beale, *Neural Network Design*, Thomson Learning series, 1996 – first edition.
- David Geary, *Graphic Java 2, Volume II Swing*, Sun Microsystems press, Indian reprint 1999 Ed.
- Herbert Schildt, *The Complete Reference, Java 2*, Fourth Edition, Tata McGraw Hill, 2001 Ed.
- John Adolph Palinski, *Oracle Database Construction Kit*, Indian Reprint 1999.

Sample Code

```
// Neural Network Cycle Interface

import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.util.*;
import javax.swing.*;


class neural extends JFrame implements ActionListener
{
        JTextField text = new JTextField(10);
        JButton but = new JButton("OK");
        int counter;
        private String mast=null;
        public neural(String master,int temp)
        {
                JPanel p=new JPanel();
                p.setLayout(null);
                mast=master;
                counter=temp;
                but.setBounds(100,100,100,20);
                text.setBounds(100,200,100,20);
                p.add(but);p.add(text);
                getContentPane().add(p);

                addWindowListener(new WindowAdapter()
                {
                        public void windowClosing(WindowEvent e)
                        {
                                dispose();
                                System.exit(0);
                        }
                });
                but.addActionListener(this);

        }
```

```
public void actionPerformed(ActionEvent ae)
{
        if(ae.getSource() == but)
        {
                int cycles = 0;
                String str = text.getText();
                if(!str.equals(""))
                {
                        cycles = Integer.parseInt(text.getText());
                        System.out.println(cycles);
                }
                if(cycles != 0)
                {
                try{
                int bob=5;
          Ntraining trn = new Ntraining(counter,bob);
                trn.startTraining(cycles);
        JOptionPane jo1=new JOptionPane("Training
        Complete!",JOptionPane.INFORMATION_MESSAGE);
        JDialog d1= jo1.createDialog(null,"Completed!");
        d1.show();
                question frame=new question(mast);
                dispose();
                frame.setSize(450, 450);
                frame.setTitle("Question");
                frame.setVisible(true);
                }catch(Exception ioe){}
                }
        }

 }

}
```

```
// Neural Network Testing Interface

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.plaf.metal.MetalLookAndFeel;
import java.io.File;
import java.io.*;
import java.beans.*;
import java.sql.*;


class mtest extends JFrame
{
        private JLabel name,name1;
        private JLabel cname;
        private JTextField cname1;
        private JLabel selectimage;
        private JButton browse;
        private JTextField path;
        private JButton test;
        private JTextField field;
        private JButton bhelp;
        private JButton bexit;
        private JPanel jp;
        private JButton back;

        private File file;
        private JFileChooser chooser;
        private ImagePreviewer2 previewer;
        private PreviewPanel2 previewpane;
         String str;

        private int result;
        private String clname;
        private boolean validated=false;
        private String master;
        public mtest(String temp)
        {
```

```
jp=new JPanel();
jp.setLayout(null);

master=temp;

name=new JLabel("Administrator");
name1=new JLabel(temp);
cname=new JLabel("Client name");
cname1=new JTextField(30);

selectimage=new JLabel("Select Test Image");
browse=new JButton("Browse");
field=new JTextField();
test=new JButton("Test");
bhelp=new JButton("Help");
bexit=new JButton("Exit");
back=new JButton("Back");

chooser=new JFileChooser();
previewer =new ImagePreviewer2();
previewpane =new PreviewPanel2();
chooser.setAccessory(previewpane);

chooser.setFileFilter(new javax.swing.filechooser.FileFilter()
{
        public boolean accept(File f)
        {
                String name=f.getName().toLowerCase();
                return name.endsWith(".gif") || f.isDirectory();
        }
        public String getDescription()
        {
                return "gif files";
        }
});
name.setBounds(50,50,120,25);
name1.setBounds(190,50,100,25);
selectimage.setBounds(50,100,180,25);
cname.setBounds(50,140,80,25);
```

```
cname1.setBounds(170,140,100,25);

browse.setBounds(50,190,100,30);
field.setBounds(170,190,200,28);


test.setBounds(80,245,250,30);
back.setBounds(50,310,100,30);
bhelp.setBounds(170,310,100,30);
bexit.setBounds(290,310,100,30);

jp.add(name);
jp.add(name1);
jp.add(selectimage);
jp.add(browse);
jp.add(field);
jp.add(test);
jp.add(bhelp);
jp.add(bexit);
jp.add(back);
jp.add(cname);
jp.add(cname1);
getContentPane().add(jp);

browse.addActionListener(new ActionListener()
{
        public void actionPerformed(ActionEvent e)
        {
                int state =chooser.showOpenDialog(null);
                if(state==JFileChooser.APPROVE_OPTION)
                {
                        file=chooser.getSelectedFile();
                        field.setText(file.getPath());
                }
        }
});


back.addActionListener(new ActionListener()
```

```
{
        public void actionPerformed(ActionEvent e)
        {
        question frame=new question(master);
        dispose();
        frame.setSize(450,450);
        frame.setLocation(125,100);
        frame.setTitle("Question");
        frame.setVisible(true);
        }
});


test.addActionListener(new ActionListener()
{
        public void actionPerformed(ActionEvent e)
        {
                if(field.getText().equals("") ||
                cname1.getText().equals(""))
                {
                JOptionPane jo=new JOptionPane("Fields
                Empty!",JOptionPane.WARNING_MESSAGE);
                JDialog d= jo.createDialog(jp,"Retry");
                d.show();
            }
            else
            {
                clname=cname1.getText();
                str=field.getText();
                System.out.println(str);
                str=str.replace('\\','/');
                preprocess x=new preprocess(str,"TEST");
                Ntesting cc =new Ntesting();

                        Ntesting ccc =new Ntesting();
                result=ccc.getresult();
                result++;
                System.out.println("result"+result);
```

```
try{
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

        }
catch(Exception fggfe)
{
        System.out.println(fggfe);
}


try{
Connection cn=DriverManager.getConnection
("jdbc:odbc:prasanth","scott","tiger");
System.out.println("Connecting to database for
testing....");
java.sql.Statement st=cn.createStatement();
ResultSet rs=st.executeQuery("select * from
authenticate order by sno");
validated=false;
while(rs.next())
{

        if(result==rs.getInt(1))
        {
                if(clname.equals(rs.getString(3)))
                validated=true;
        }
}
System.out.println("boolean"+validated);
if (validated)
{
JOptionPane jo=new JOptionPane("Authentication
Positive "+clname+ "!",
JOptionPane.INFORMATION_MESSAGE);
JDialog d= jo.createDialog(null,"Authentication
Process");
d.show();
}
```

```
                if(!validated)
                {
                JOptionPane jo=new JOptionPane("Authentication
                FAILED !"
                ,JOptionPane.INFORMATION_MESSAGE);
                JDialog d= jo.createDialog(null,"Authentication
                Process");
                d.show();

                }
        }
        catch(Exception eeee){System.out.println(eeee);}

        }
        question fra=new question(null);
        dispose();
        fra.setLocation(50,30);
        fra.setSize(450, 450);
        fra.setTitle("Question");
        fra.setVisible(true);
        }
});


bexit.addActionListener(new ActionListener()
{
        public void actionPerformed(ActionEvent e)
        {
                dispose();
                System.exit(0);
        }
});



chooser.addPropertyChangeListener(new PropertyChangeListener()
{
public void propertyChange(PropertyChangeEvent e)
```

```
        {
            try
            {

    if(e.getPropertyName().equals(JFileChooser.SELECTED_FILE_CHANGED_
PROPERTY))
            {
                    File f=(File)e.getNewValue();
                    String s=f.getPath();
                    String suffix=null;

                    int i=s.lastIndexOf('.');

                    if(i>0 && i<s.length()-1)
                    suffix=s.substring(i+1).toLowerCase();

                    if(suffix.equals("gif") || suffix.equals("jpg"))
                    previewer.configure(f);
            }
            }

        catch(Exception ba)
        {}
        }

        });


        addWindowListener(new WindowAdapter()
        {
            public void windowClosing(WindowEvent e)
            {
                dispose();
                System.exit(0);
            }
        });
    }
```

```
class PreviewPanel2 extends JPanel{
        public PreviewPanel2()
        {  try
           {
                JLabel label=new JLabel("Image
                Previewer",SwingConstants.CENTER);
                setPreferredSize(new Dimension(150,0));
                setBorder(BorderFactory.createEtchedBorder());
                setLayout(new BorderLayout());

                label.setBorder(BorderFactory.createEtchedBorder());
                add(label,BorderLayout.NORTH);
                add(previewer,BorderLayout.CENTER);
                }
                catch(Exception e)
                {}
        }
}
class ImagePreviewer2 extends JLabel{
        public void configure(File f)
        {
                try{
                Dimension size =getSize();
                Insets insets =getInsets();
                ImageIcon icon =new ImageIcon(f.getPath());
                                setIcon(new
                ImageIcon(icon.getImage().getScaledInstance(size.width-insets.left-
                insets.right,
                size.height-insets.top-insets.bottom,Image.SCALE_SMOOTH)));
                }
                catch(Exception e)
                {}
                }
        }
}
```

```
// Neural Network Testing

import java.io.*;
import java.util.*;

class Ntesting
{
        double testpat[] = new double[30];
        double wt1[][];
        double wt2[][];
        double act1[];
        double act2[];
        double out1[];
        double out2[];
        double tresh1[];
        double tresh2[];
        int noOfInputNeurons;
        int noOfOutputNeurons;
        int noOfHiddenNeurons;
        public Ntesting()
        {
                //Reading the test data
                BufferedReader bf = null;
                try
                {
                        bf = new BufferedReader(new FileReader("test.nn"));
                }catch(FileNotFoundException fo){}
                String str = null;
                try{str = bf.readLine();}
                catch(IOException ioe){}
                StringTokenizer strtok = new StringTokenizer(str,",");

                int index = 0;
                while(strtok.hasMoreTokens())
                {
                        testpat[index] = Double.parseDouble(strtok.nextToken());
                        index++;
                }
```

```
        //reading the configuration details
        try{
                BufferedReader buf = new BufferedReader(new
                FileReader("config.nn"));
                StringTokenizer strtk = new
                StringTokenizer(buf.readLine(),",");
                noOfInputNeurons = Integer.parseInt(strtk.nextToken());
                noOfHiddenNeurons = Integer.parseInt(strtk.nextToken());
                noOfOutputNeurons = Integer.parseInt(strtk.nextToken());
                System.out.println(noOfHiddenNeurons);
                act1 = new double[noOfHiddenNeurons];
                act2 = new double[noOfOutputNeurons];
                out1 = new double[noOfHiddenNeurons];
                out2 = new double[noOfOutputNeurons];

        }catch(IOException ioe){}


        //reading weights and thresholds
        try{
        FileInputStream fin = new FileInputStream("weights.nn");
        ObjectInputStream oin = new ObjectInputStream(fin);
        wt1 = (double[][])oin.readObject();
        wt2 = (double[][])oin.readObject();
        tresh1 = (double[])oin.readObject();
        tresh2 = (double[])oin.readObject();
        displayAll();
        System.out.println();
        System.out.println("-----------------------------------");
        System.out.println();
        test();
        }catch(IOException ioe){System.out.println("Cannot read weights");}
        catch(ClassNotFoundException coe){System.out.println("Class Not
Found");}
    }

    void test()
    {
            for(int i = 0;i<noOfHiddenNeurons;i++)
```

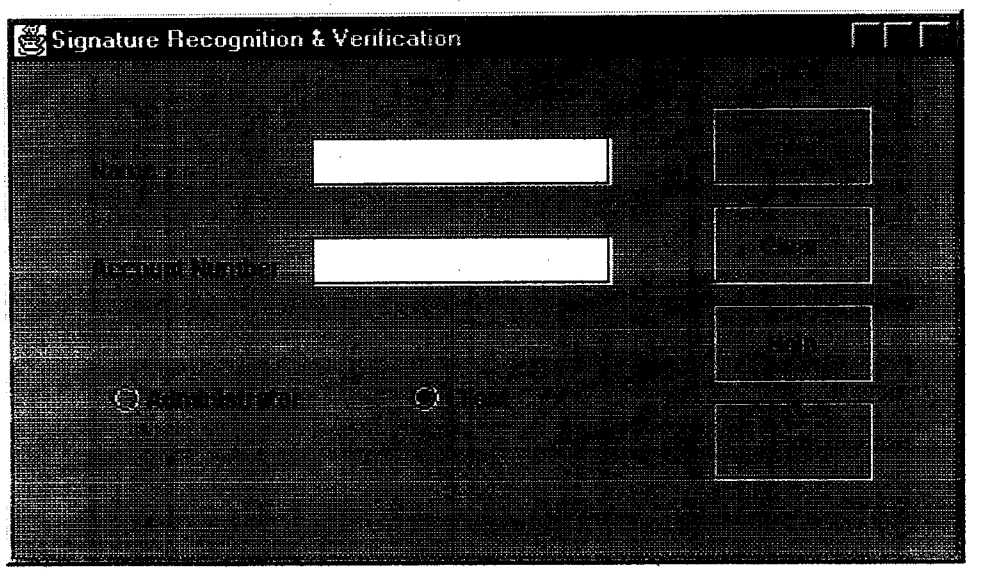
```
        {
                act1[i] = 0;
                for(int j = 0;j<noOfInputNeurons;j++)
                {
                        act1[i]+= wt1[j][i]*testpat[j];
                }
                act1[i]+=tresh1[i];
                out1[i] = (1.0f)/(1.0f + (double)Math.exp(-act1[i]));

        }

        for(int i = 0;i<noOfOutputNeurons;i++)
        {
                act2[i] = 0;
                for(int j = 0;j<noOfHiddenNeurons;j++)
                {
                        act2[i]+= wt2[j][i]*out1[j];
                }
                act2[i]+=tresh2[i];
                out2[i] = (1.0f)/(1.0f + (double)Math.exp(-act2[i]));

        }

        for(int i = 0;i<noOfOutputNeurons;i++)
        {
                if(out2[i]>=0.5)
                System.out.print("1");
                else
                System.out.print("0");


        }
        System.out.println("");

}


public int getresult()
{
```

```
        int count=0;
        int result=0;
        int temp=0;
        for(int i=noOfOutputNeurons-1;i>=0;i--)
        {
                if(out2[i]>=0.5)
                temp=1;
                else
                temp=0;

                result=result+(int)(temp*Math.pow(2,count));
                count++;
        }

        return(result);
        }
//Displays all the parameters read from files
void displayAll()
{
        System.out.println("Input Neurons:"+noOfInputNeurons);
        System.out.println("Hidden Neurons:"+noOfHiddenNeurons);
        System.out.println("Output Neurons:"+noOfOutputNeurons);
        System.out.println("weights 1");
        for(int i = 0;i<wt1.length;i++)
        {
                for(int j = 0;j<wt1[i].length;j++)
                {
                        System.out.print(wt1[i][j]+",");
                }
                System.out.println("");
        }
        System.out.println("weights 2");
        for(int i = 0;i<wt2.length;i++)
        {
                for(int j = 0;j<wt2[i].length;j++)
                {
                        System.out.print(wt2[i][j]+",");
                }
                System.out.println("");
```

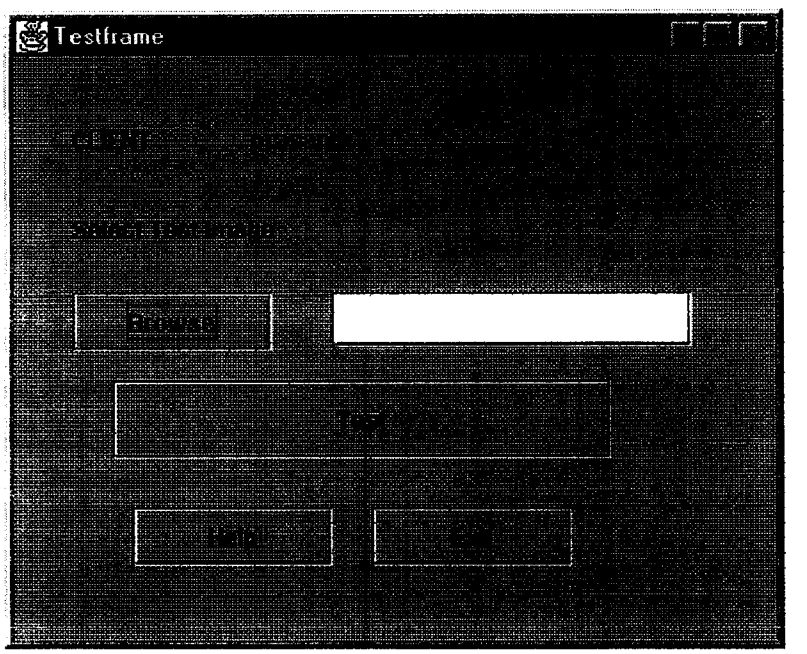
```
            }
            System.out.println("Thershold 1");
            for(int i = 0;i<tresh1.length;i++)
            {
                    System.out.print(tresh1[i]+",");
            }
            System.out.println("");
            System.out.println("Thershold 2");
            for(int i = 0;i<tresh2.length;i++)
            {
                    System.out.print(tresh2[i]+",");
            }
            System.out.println("Test pattern");
            for(int i = 0;i<30;i++)
            {
                    System.out.print(testpat[i]+",");
            }
        }
}
```

Input/Output Screen

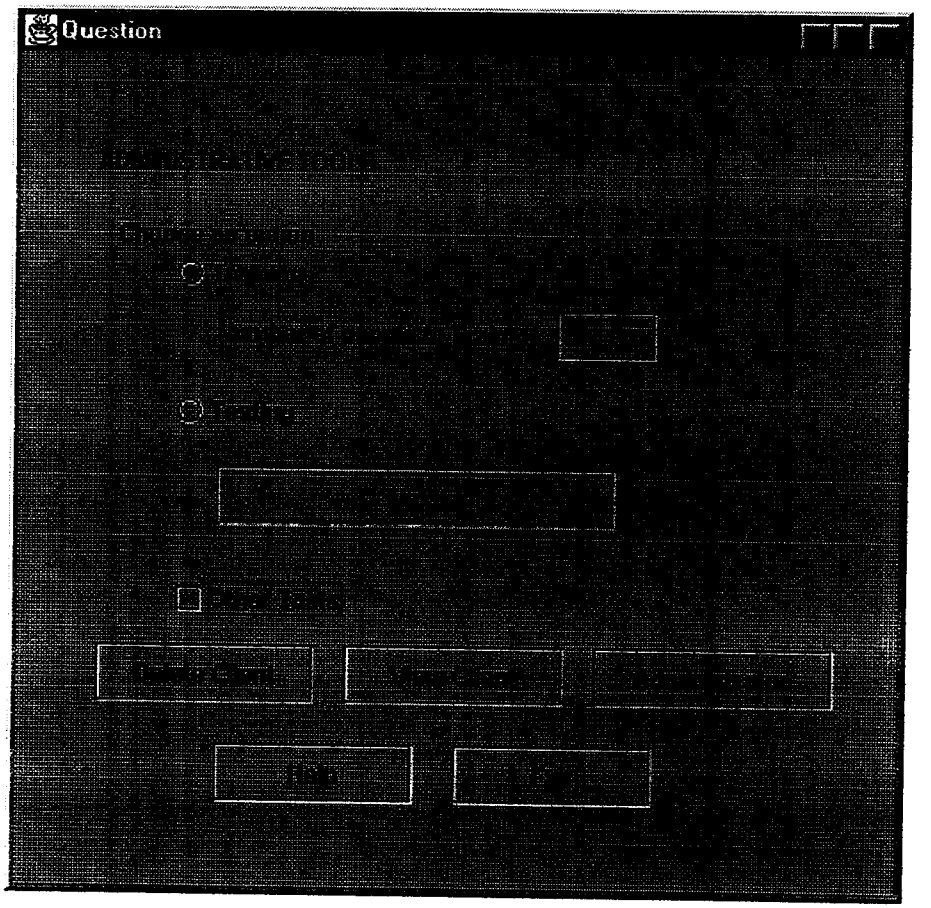
Input Screen



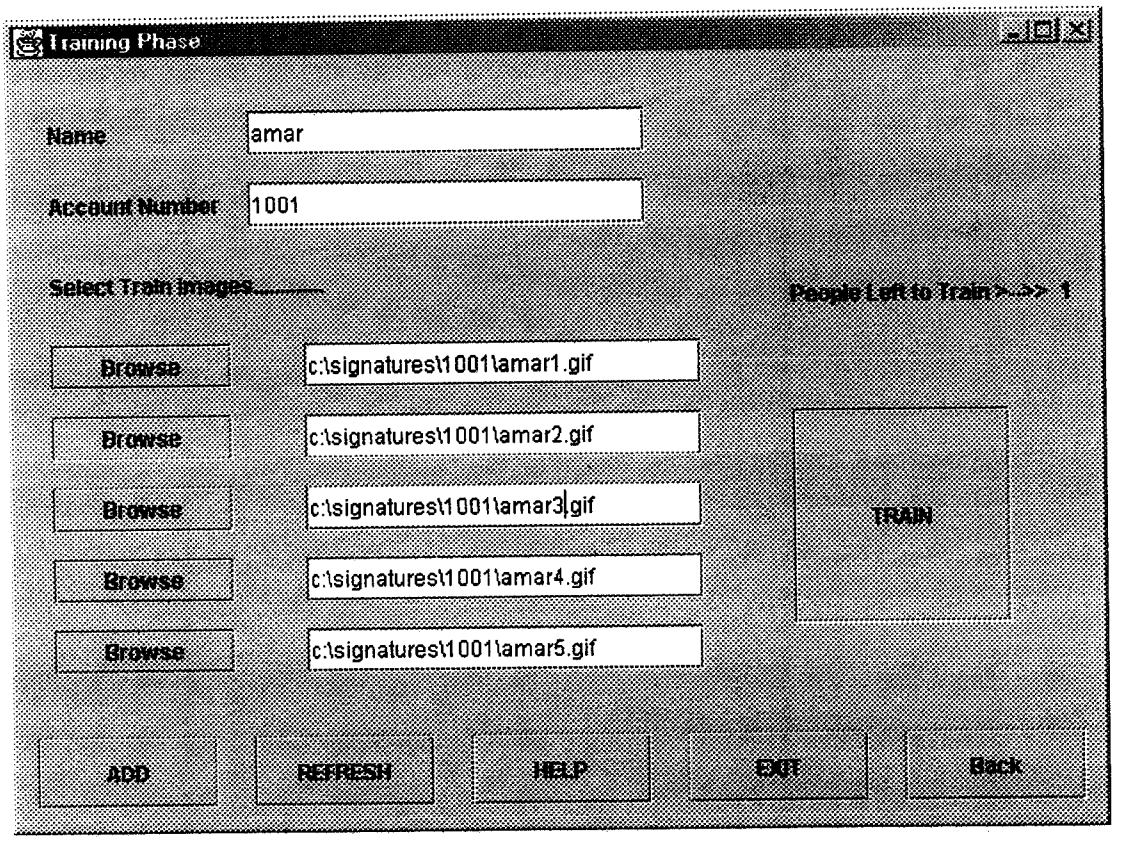
Signature Testing Interface



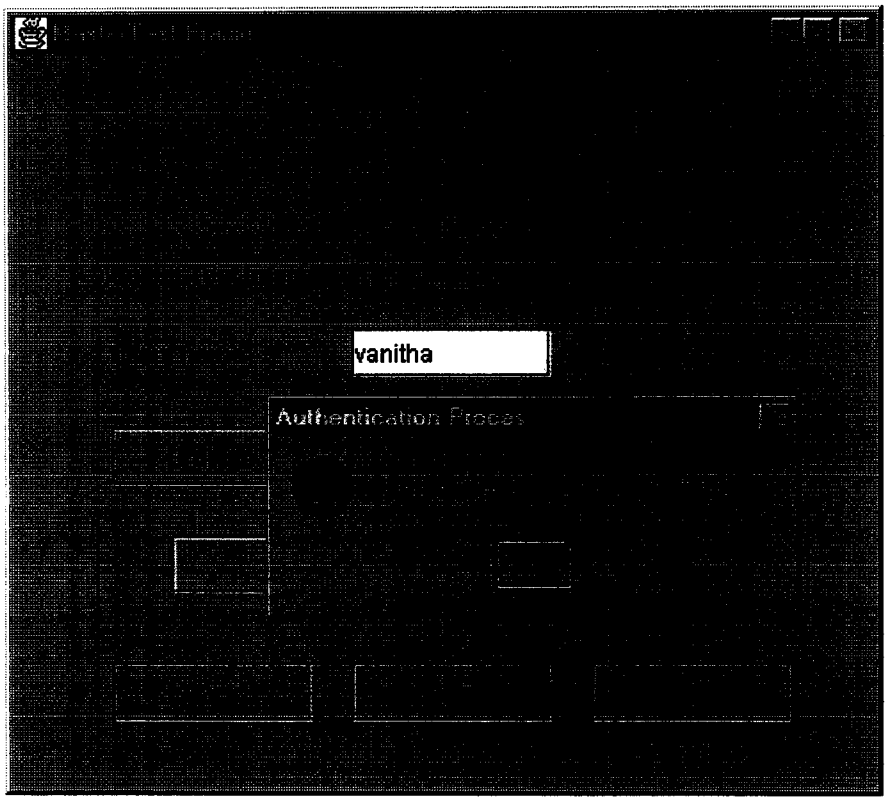
Administrator Section



Training Interface



Test Result



Neural Networks weights & cycles