# HANDWRITTEN CHARACTER RECOGNITION USING NEURAL NETWORKS



PROJECT REPORT

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE AWARD OF THE DEGREE OF

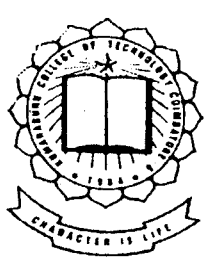
**MASTER OF ENGINEERING**

OF THE BHARATHIAR UNIVERSITY, COIMBATORE.

SUBMITTED BY
**M.DURAIPANDIAN**
Register No.0037K0005

GUIDED BY

**Mrs.D.CHANDRAKALA, M.E**




DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
**KUMARAGURU COLLEGE OF TECHNOLOGY**
**Coimbatore – 641 006**
December – 2001

KUMARAGURU COLLEGE OF TECHNOLOGY
(Affiliated to the Bharathiar University)
Coimbatore – 641 006

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

# CERTIFICATE

*This is to certify that the project work entitled*

## HANDWRITTEN CHARACTER RECOGNITION USING NEURAL NETWORKS

*Has been submitted by*

**M.DURAIPANDIAN**
**Register Number –0037K0005**

*Submitted in partial fulfillment of the requirements for the award of the Degree of*

**Master of Engineering** of **Bharathiar University.**

D. Chandrakumar 18/12/2001
Guide

S. Thangasamy 20/12/01
Head of the Department

*Submitted for the University Examination held on* 20/12/2001

D. Chandrakumar 20/12/2001
Internal Examiner

External Examiner

# DECLARATION

I hereby declare that the project entitled '**HANDWRITTEN CHARACTER RECOGNITION USING NEURAL NETWORKS**', submitted to **Bharathiar University** as the project work of Master of Engineering (COMPUTER SCIENCE AND ENGINEERING) Degree, is a record of original work done my me under the supervision and guidance of **Mrs.D.Chandrakala M.E, Kumaraguru College of Technology, Coimbatore** and this project work has not found the basis for the award of any Degree/ Diploma/ Associateship/ Fellowship or similar title to any candidate of any university.

M. Dusaipandian

Signature of the Student

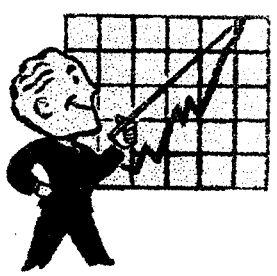
Place: Coimbatore

Date: 18/12/2001

Countersigned by

D Chandra 18/12/2001

Internal Guide



# Acknowledgement

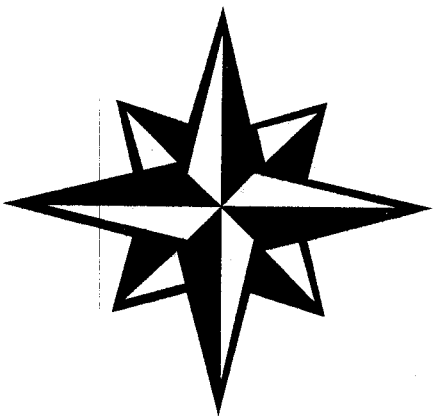
## ACKNOWLEDGEMENTS

I am greatly indebted to my revered Principal **Dr.K.K.Padmanabhan**, B.Sc (Engg), M.Tech, and PhD who has been leading the institution in all aspects of potential growth.

I earnestly express my sincere thanks to our beloved Head of the Department **Prof.Dr.S.Thangasamy**, B.E. (Hon's), Ph.D, for his immense encouragement and help for being a source of inspiration all through our course of study.

I am very much obliged to express my sincere thanks and gratitude to my guide **Mrs.D.Chandrakala,** M.E for her guidance in project identification, valuable suggestions and encouragement, which has enabled me to complete my project successfully.

I also thank all the staff members of the Department of Computer Science and Engineering for all their encouragement and moral support.



# Synopsis

# SYNOPSIS

Character recognition is a trivial task for humans, but to structure a computer program that does character recognition is extremely difficult. Recognizing patterns is just one of those things a human does well and computers don't. The main reason for this is that it accommodates many sources of variability. Noise for example, consists of random changes to a pattern, particularly near the edges, which may be interpreted as a completely different character by a computer program. Another source of confusion is the high level of abstraction.

There exist several different techniques for recognizing handwritten English uppercase alphabets and Digits. One distinguishes characters by the number of loops in a character and the direction of their concavities. Another common technique uses a neural network to recognize the characters. The objective of this project is to investigate how good neural networks solves the handwritten character recognition problem.

The sole aim of this project is to create an easy to use environment in which the user can draw characters and then let the program try to interpret the characters. It also demonstrates how effective the Backpropagation algorithm is used to recognize the characters. The proposed project is divided into three phases.
The three phases are

1) Data capture phase
2) Feature extraction phase
3) Training and Recognition phase

Data capture phase:

In the data capture stage inputs are given to the system by means of drawing a picture frame with the help of a mouse.The picture frame is divided into 30x30 grid of pixels.

Feature extraction phase:

This process is necessary because not all information of the bitmap character can be used as a input to the neural network. Typically the dimension of the area of the bitmap character may be 30x30 pixels.This would mean that if every pixel information was input to the neural network,900 input nodes would be required.This is computationally not feasible in a PC environment.

Some features can be extracted from the bitmap character. One simple feature extraction method used in the project, associates each pixel of the bitmap to one node in the neural network input grid. The number of pixels associated to each cell is then normalized as a proportion of the maximum number of pixels in all the cells.The input pattern is normalized from a resolution of 30x30 to 6x6 grid of pixels.This will determine the size of the neural network.

Training and Recognition phase:

The input patterns are then propagated through the network and trained. The character that is closely related to the stored character is recognized.

The conventional backpropagation algorithm is replaced by the fastlearning backpropagation algorithm due to its inherent disadvantages such as local minima, network paralysis and slow convergence. The proposed Backpropagation algorithm is implemented in Turbo C++ 3.0.

# Contents

## CONTENTS

# Introduction

## 1.1 The current status of the problem taken up

Handwritten character recognition systems have been proposed and implemented in a number of different ways. However for the recognition of unconsidered handwritten characters, no simple scheme is likely to achieve high recognition and reliability rates, In order to maximize the performance of unconstrained handwritten character recognition, the following approach is considered in this project. That is to design a feature extractor that does not miss important features while minimizing the number of meaningless pixels.

This scheme involves a feature extraction method associates each pixel of the bitmap to one node in the neural network input grid. The number of pixels associates to each cell is the normalized as a proportion of the maximum number of pixels in all the cells and a classification stage for recognizing characters with a simple feedforward backpropagation network trained with the backpropagation algorithm .

The neural networks designed implemented and tested for recognition of handwritten characters is multi-layer perceptron (MLP) networks is based on the backpropagation network and apply the backpropagation algorithm.

The conventional backpropagation algorithm a simple supervised learning algorithm has a number of limitations such as performance dependence on constant factors (learning rate, momentum factor) and slow convergence.An efficient fast learning algorithm has been incorporated to overcome these problems.

## 1.2 Relevance and Importance of the topic

The constant development of computer tools leads to a requirement of easier interfaces between the man and the computer. Handwritten Character Recognition may for instance be applied to Zip-Code recognition, automatic printed form acquisition, or checks reading. The importance of these applications has led to intense research for several years in the field of Off-Line handwritten character recognition.

For some years, Artificial Neural Networks, and especially Multilayer Perceptrons, have shown good capabilities in performing classification tasks. This is due to the non-linearity's that are included in these connectionist systems, and to the discriminate training phase that they are submitted to. However, their performance is strongly affected by the quality of the representation of the characters. This may require a large number of parameters to represent the character, which then results in difficulty in establishing the rules for recognition. In other words the MLPs become difficult to train. Moreover, the greater the size of the network, the greater is the computation time. This can greatly restrict their practical use.

So, it is necessary to perform efficient features extraction on the one hand, and to optimize the layout of the artificial neural network on the other hand.

There are four interesting and recent applications/products that use character recognition technologies in order to show the usefulness of such technologies.

### Real Time Character Recognition in Palm Top Computing Devices

3Com Palm Pilot uses a Graffiti-based writing pad for touch screen input in place of a keyboard. Each character is input as a continuous stroke. A stroke is completed when the user lifts up the pen from the scratch pad. Metz gives a review of the various features of 3Com Palm Pilot III. Since the introduction of Palm Pilot III, many other similar palm top computing devices have appeared in the market. Of these are those that run on Microsoft Windows CE Operating System for example Nino by Philips and the Aero series by Compaq. All these devices have real time handwritten character recognition capability.

### Light Pen OCR Dictionary

A Canadian company has come up with mobile scanners that are able to recognize printed characters and words, search up their meaning and even read the meaning out. These devices also come with the ability to translate words from one language to another. A point to note is that Quicktionary still uses OCR and not ICR technologies as it could not recognize hand written characters. However, the interesting point to note is that it manages to recognize printed words and also to pronoun them and even translate them to a different language. Many other technologies such as natural language and voice processing must have been used together with OCR to come up with such a unique product.

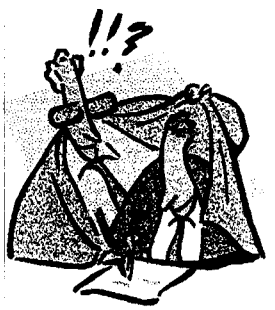
### Mail-sorting Machine

Basically modern mail sorting machines are quite commonly used in Post Offices around the world that handle large volume of mail daily. Sorting mail manually is tedious and error prone. Mail sorter machine uses ICR technologies to recognize handwritten postal code and sort the mail according based on the postal code recognized. The marvelous aspect of mail sorting machine is that they are very fast. A mail-sorting machine

with a sort speed of 30,000-50,000 letters and postcards per hour has been developed by Hitachi.

**Form Processing**

Electronic form processing basically uses ICR technologies to recognize handwritten characters in pre-designed forms. The recognized characters can then be directly entered into a database. This increases productivity by cutting down on manual keyboard entry. Garris mentioned that the National Institute of Standards and Technology (NIST) has developed a standard reference form-based handprint recognition system for evaluating optical character recognition. NIST is making this recognition system freely available to the general public.




# Literature Survey

# LITERATURE SURVEY

An introduction to Neural networks is given in the book titled ' Neural networks in Computer Intelligence' authored by LiMin Fu.It is a pleasant introduction to a student who is interested in projects dealing with neural networks. The book gives an insight into the concept of a biological neuron and the similarity with the artificial neuron simulated in the computer. The various computational models and types of neural networks are discussed in detail. The book gives an overview of the different training algorithms used to train the computational models. The book gives us sufficient information to decide on which computational model and training algorithm should be implemented in view of the problem at hand. For the implementation of the project, the classification computational model is selected to recognize the various hand written characters.

The two popular training techniques, the supervisory method and the unsupervised method of training have been discussed. They differ depending on the input and output patterns present to them. The book also discusses about the back propagation algorithm that is used to train the given neural network. The book also gives an idea about the construction of a neural network, and the selection of the number of nodes in each layer.

For the project at hand, the implementation of the supervised form of training using the back propagation algorithm was found as the best solution to solve the problem.

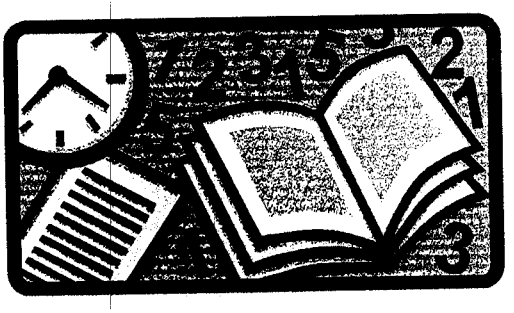
Automatic recognition of hand written character is one of t he benchmarks in pattern recognition research detailed insight in the recognition of hand written characters is authored by N.K.Bose and P.Liang namely 'Neural network Fundamentals with Graphs algorithms and applications'. The book mainly deals with the different applicative prospects of neural networks. It describes how the concept of a neural network can be used to solve difficult problems by giving many instances from real life situations.

In the context of hand written character recognition, the book describes that there are two algorithms to solve the hand written recognition problem. They are the backpropagation algorithm and the growth algorithm. It looks into the advantages and disadvantages and compares them from the implementation point of view. It narrates the typical method of constructing a neural network for recognition of hand written characters. It tells about the number of layers and the number of neurons in each layer to be selected based on the problem or the number of characters the neural network has to be trained to identify.

The second edition of the book' Neural networks and fuzzy logic by Dr.Valluru B.Rao and Hayagriva V. Rao serves as a guide into the implementation of the project. It firstly introduces a beginner into the concept of neural networks and fuzzy logic. It mainly concentrates on solving problems using C++ having an object-oriented concept. It describes the construction of neural networks and the application of neural networks to pattern recognition, which is of interest to the project. It gives us an insight to programming demonstrating a C++ implementation of a back propagation simulator.

The book also surveys the problems of the conventional backpropagation algorithm and looks into certain modifications leading to an algorithm having better efficiency and training speeds. The conventional backpropagation training algorithm is replaced by the fast learning algorithm eradicating the problems such as slow convergence and local minima. The fast learning algorithm restores the problem of local minima by adding a momentum term thereby preventing the network from settling in any local minima. This algorithm greatly reduces the time required to train a neural network.

Websites give us various feature extraction methods and the preprocessing phases in detail discussing their advantages and disadvantages from the problem point of view.



# Proposed Line of Attack

Creating software that can recognize handwritten characters (English Uppercase Alphabets & Digits) using feed forward backpropagation neural network.

The project work comprises of three phases, they are

1. Data Capture
2. Feature Extraction
3. Training & Recognition.

Data Capture phase

Inputs are given to the system by means of drawing into a picture_frame with the help of a mouse. The picture frame is divided into 30 X 30 grid of pixels . The drawing in the picture_frame will be then scanned by the system.

The input pattern to the system must be digitized into a rectangular picture_frame array P={p=(i, j) / 1<=i<=n, 1<=j=m; n, m belongs to N}. The pattern is a binary picture, i.e., the points on the pattern assume the value of one while the other points of the frame take the value of zero. For our recognition system the picture_frame is a 30 x 30 array.

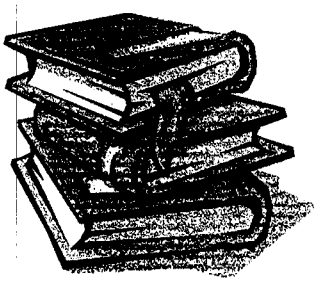
Feature Extraction

This process is necessary because not all information of the bitmap character can be used as input to the neural network. Typically the dimension of the area of a segregated character may be 30 by 30 pixels. This would mean that if every pixel information was input into the neural network, 900 input nodes would be required. This is computationally not feasible in a PC environment. Thus we go for the feature extraction stage which normalizes the given input pattern into a pattern of lower resolution.

This reduces the number of input neurons to the neural network In this project the input pattern is normalized from a resolution of 30x30 to a resolution of 6x6 grid of pixels.This facilitates design of a neural network with 36 input nodes.

Training & Recognition

The training and recognition phase is the most trivial phase in the implementation of a neural network. There are different computational models in neural networks. The project uses the classification computational model in identifying the hand written characters. The neural networks designed, implemented and tested for recognition of handwritten characters, is a feedforward network trained using the Backpropagation algorithm.



# Details of Proposed Methodology

**Introduction to Artificial Neural Networks**

Modeling of Human Recognition by Machine

One of the main differences between the way human beings perceive the world and how the machine does it is that human beings are less 'exact' than a machine.

To enable the machine to simulate human cognitive behavior, soft computing is introduced. Soft computing attempts to remove the rigidity of traditional computing. Currently the main areas in soft computing are:

1) Fuzzy Logic

2) Neural network

3) Genetic Algorithm

All of these are inspired by related characteristics found in human being and other biological organisms.

Fuzzy logic is motivated by the way human beings describe the world. The attributes that human beings assign to an entity are normally discrete while a more continuous treatment would have been more accurate. For example, words like cold warm and hot are used to describe temperature. The measure of temperature should have been continuous with a range from a fixed minimum to a fixed maximum. However, human beings are more comfortable with working with finite discrete classes than with a continuous infinite quantum. Fuzzy logic introduces discrete fuzziness into computer by making it work and reason with fuzzy components. A temperate of 35.7° C would be considered as 25% cold, 75% warm and 10 % hot.

Neural network is motivated by the findings of the working of neurons in the human brain. It is believed that knowledge is stored in the brain's neurons as complex and intricate network of connections (synapses) between neurons. As more knowledge is acquired, the arrangement of the synapses changes. A single piece of information is stored by not a single neuron but by probably thousands of them, each with a small contribution. This makes it possible for a brain tumor patient to continue to function even after a brain surgery that removes some part of his brain.

Implementation of neural computing in computer is by the use of data structure that simulates the network of neurons. The synapses are simulated as small weights in 2D arrays. The effect of training is to alter the content of such arrays of weights so that it will eventually reflect the characteristics of the trained input patterns. This approach is in contrast with traditional computing where information is normally stored in a knowledge database file. Information retrieval in a database approach is based on record searching. The system can only retrieve data that has been previously stored. In the neural network approach, the system can approximate a result even when the input is not one of those trained. This is a typical human characteristic and is very useful for pattern recognition where it is not possible to train all variations of the input.

Genetic algorithm takes the machine one step further. Inspired by the Darwinian evolution process, a population of structures is maintained. As the system interacts with the outside world, not only learning takes place, but there will also be a change in the data structure to best facilitate the learning process.

## Modeling of the Brain's Neuron

Since its introduction in the 60's, the basis for modeling the brain's neurons in a computer simulation has not changed significantly. The below figure shows the structure of biological neurons and the Table shows how each of the neuron's components are being modeled in a computer.

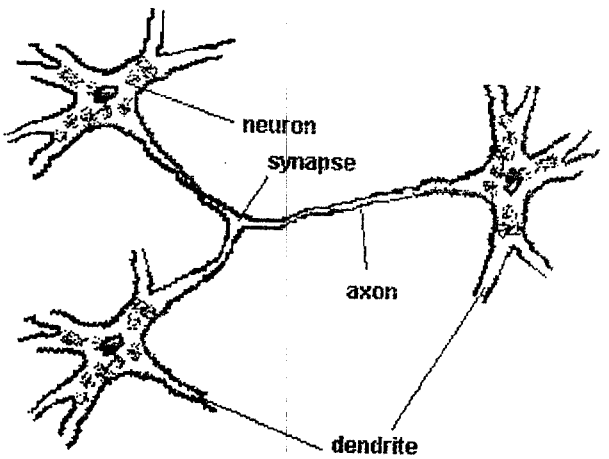
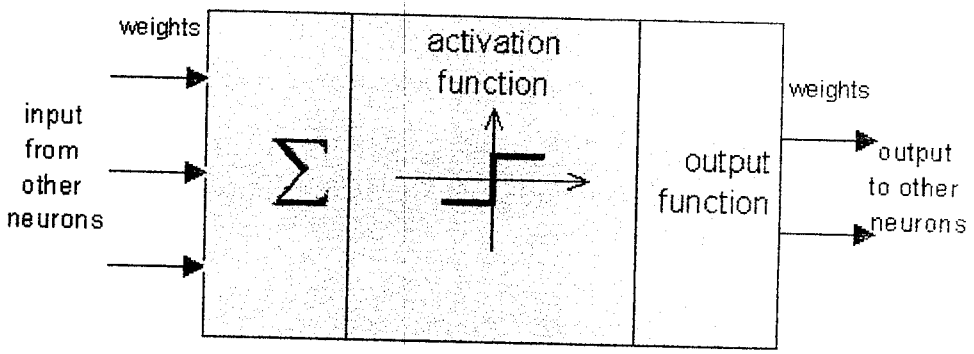



Figure : Neuron

Generally spoken, there are many different types of neural nets, but they all have nearly the same components. If one wants to simulate the human brain using a neural net, it is obviously that some drastic simplifications have to be made:

First of all, it is impossible to "copy" the true parallel processing of all neural cells. Although there are computers that have the ability of parallel processing, the large number of processors that would be necessary to realize it can't be afforded by today's hardware. Another limitation is that a computer's internal structure can't change while performing any tasks.

The following figure shows an idealized neuron of a neural net.



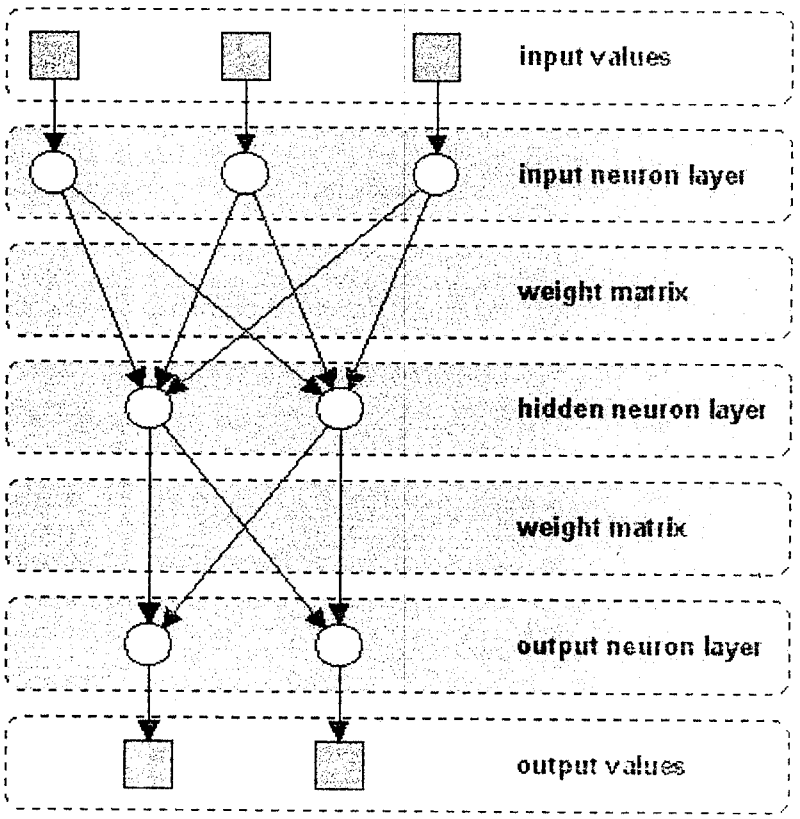

Structure of a neuron in a neural network

An artificial neuron looks similar to a biological neural cell. And it works in the same way. Information (called the input) is sent to the neuron on its incoming weights. This input is processed by a propagation function that adds up the values of all incoming weights.

The resulting value is compared with a certain threshold value by the neuron's activation function. If the input exceeds the threshold value, the neuron will be activated, otherwise it will be inhibited. If activated, the neuron sends an output on its outgoing weights to all connected neurons and so on.

In a neural net, the neurons are grouped in layers, called neuron layers. Usually each neuron of one layer is connected to all neurons of the preceding and the following layer (except the input layer and the output layer of the net). The information given to a neural net is propagated layer-by-layer from input layer to output layer through none, one or more hidden layers. Depending on the learning algorithm, it is also possible that information is propagated backwards through the net.

The following figure shows a neural net with three neuron layers.




Neural net with three neuron layers

Note that this is not the general structure of a neural net. For example, some neural net types have no hidden layers or the neurons in a layer are arranged as a matrix. What's common to all neural net types is the presence of at least one weight matrix, the connections between two neuron layers.

| Biological Component | Computer Model |
|---|---|
| Neuron | Node |
| Synapse | 2D matrix of weight |
| Dendrite | Inputs |
| Axon | Output |
| Arrangement of neurons | Layer of nodes |

Table : Computer modeling of neurons

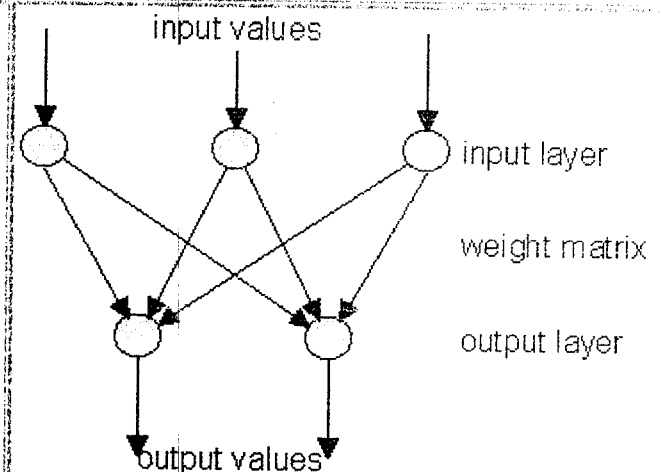
**Use of Neural Network for Pattern Classification**

One of the primary uses of neural network is for the purpose of input classification and approximation. This makes it quite suitable for the purpose of pattern classification.

There are many neural network models that can be used for this purpose. Some of these are the simple single layer perceptron network, multi-layer perceptron network with back propagation, counter propagation network, neo-cognitive network, Hop field's network and ART1 (Adaptive Resonance Theory 1).

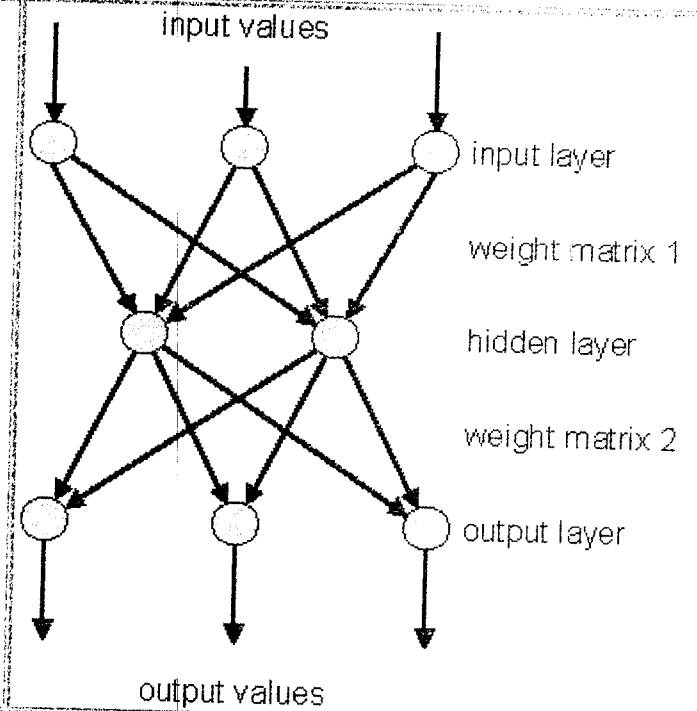
**Single Layer Perceptron**

The Perceptron was first introduced by F. Rosenblatt in 1958. It is a very simple neural net type with two neuron layers that accepts only binary input and output values (0 or 1). The learning process is supervised and the net is able to solve basic logical operations like AND or OR. It is also used for pattern classification purposes. More complicated logical operations (like the XOR problem) cannot be solved by a Perceptron.

| Perceptron characteristics | |
|---|---|
| sample structure |  |
| Type | Feedforward |
| neuron layers | 1 input layer 1 output layer |
| Input value types | binary |
| activation function | hard limiter |
| learning method | Supervised |
| learning algorithm | Hebb learning rule |
| mainly used in | Simple logical operations<br>pattern classification |

**Multi-Layer-Perceptron**

The Multi-Layer-Perceptron was first introduced by M. Minsky and S. Papert in 1969. It is an extended Perceptron and has one ore more hidden neuron layers between its input and output layers. Due to its extended structure, a Multi-Layer-Perceptron is able to solve every logical operation, including the XOR problem.

| Multi-Layer-Perceptron characteristics | |
|---|---|
| **sample structure** |  |
| **Type** | Feedforward |
| **neuron layers** | 1 input layer<br>1 or more hidden layers<br>1 output layer |
| **input value types** | Binary |
| **activation function** | hard limiter / sigmoid |
| **learning method** | Supervised |
| **learning algorithm** | delta learning rule<br>backpropagation (mostly used) |
| **mainly used in** | complex logical operations<br>pattern classification |

## The Feed Forward and Back propagation Network

The feed forward and back propagation network is one of the most popular neural networks currently being used for implementing pattern recognition systems. It is one of the simplest and yet yields surprising good results.

The only problem is that network training would normally be quite long with no promise that the network can be trained at all.

## Network Training

The training for a neural network effectively alters the weights in the weight matrix. Network training can be supervised or unsupervised.

A supervised network makes use of training input/output pairs. An input vector is matched with a desired output vector. During training, the weights in the weight matrix are changed to minimize the network training error (the difference between the desired output and the actual output).

Unsupervised training does not make use of any desired outputs for comparison. Learning is accomplished based on observations of responses to input. It can be said that changes in the weights can be made to improve on the classification of input patterns into classes which has gradually evolved as training progressed.

The multi-layer back propagation method uses supervised training. A self -organizing map network uses unsupervised learning while the counter propagation network uses a combination of supervised and unsupervised training.

## Back propagation Network Implementation

Back propagation Neural Network Theory

The Back propagation Net was first introduced by G.E. Hinton, E. Rumelhart and R.J. Williams in 1986 and is one of the most powerful neural net types. It is a supervised algorithm that learns by first computing the output using feed forward network, then calculating the error signal and propagating the error backwards through the network.

It has the same structure as the Multi-Layer-Perceptron and uses the back propagation-learning algorithm. The back propagation network is probably the most well known and widely used among the current types of neural network systems available.

The back propagation network is a multiplayer feed forward network with a different transfer function in the artificial neuron and more powerful learning rule. The learning rule is known as back propagation, which is a kind of gradient descent technique with backward error propagation, as shown in figure .

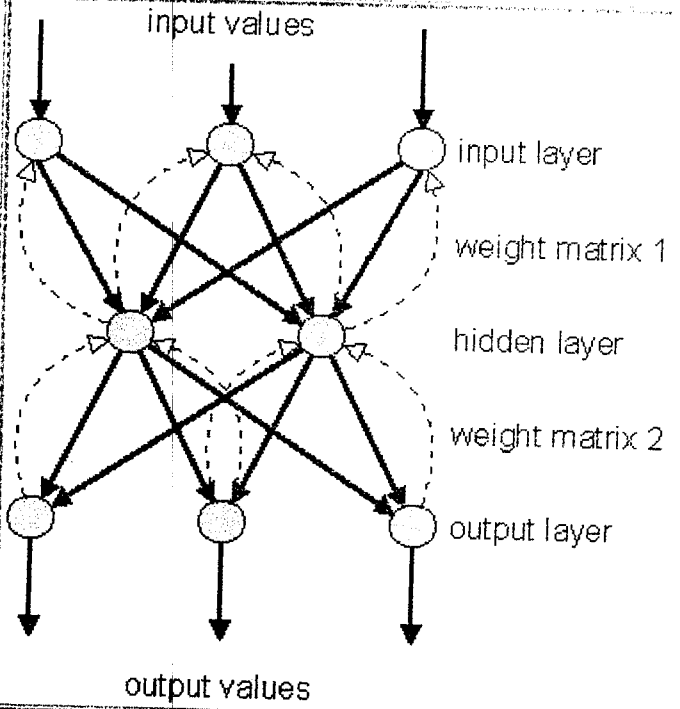
| Backpropagation Net characteristics | |
|---|---|
| **sample structure** |  |
| **Type** | feedforward |
| **neuron layers** | 1 input layer<br>1 or more hidden layers<br>1 output layer |
| **input value types** | Binary |
| **activation function** | Sigmoid |
| **Learning method** | Supervised |
| **Learning algorithm** | Backpropagation |
| **mainly used in** | complex logical operations<br>pattern classification<br>speech analysis |

Figure . The backpropagation network

The back propagation network in essence learns a mapping from a set input patterns to a set of input patterns .the network can be designed and trained to accomplish a wide variety of mappings. This ability comes from the nodes in the hidden layer or layers or layers of the network which learn to respond to features found in the input patterns. The features recognized or extracted by the hidden units correspond to the correlation of activity among different input units. As the network is trained with different examples, the network has the ability to generalize over similar features found in different patterns. The key issue is that the hidden units must be trained to extract a sufficient set of general features applicable to both seen and unseen instance. To achieve this goal, at first, the network must not be trained to extract a sufficient set of general features applicable to both seen and unseen instances.

**Benefits/Advantages of Backpropagation**

Backpropagation is a supervised learning algorithm that can be applied to multilayer feedforward networks. The standard algorithm is straightforward and easy to understand. The various enhancements that have been introduced over time aim to increase the efficiency of the standard algorithm.

**Limitations of Back propagation**

The conventional back propagation algorithm is a simple algorithm which is found to satisfactorily for simple applications. The convergence is based on the initial random weight of the network and various constant parameters like learning rate and momentum factor. A wrong choice for the parameters lead to slow convergence, stuck at local minimum. Even a slight variation in the parameters affects the performance. The limitations of the conventional algorithm are as follows.

Network Paralysis

During the training process the weights in the network get adjusted to large values. As a result the neurons operate at large values of out, in a region where the derivative of the squashing function becomes small. The error signal sent back is proportional to the derivative and training becomes virtually standstill. This is called as network paralysis.

Local Minima

Back propagation is gradient descent technique which constantly adjust its weights towards a minimum. The error surface of a network is highly convoluted-full of hills, valley, folds and gullies. The network can get trapped in a local minima (shallow valley), and may not be able to escape.

Step Size

A proper step size is to be selected. A smaller step size leads to slower convergence and a large value leads to paralysis or continuous instability.

Temporal Instability

During the process of learning to recognize the input, the network may forget the previously learned one. This is termed as Temporal Instability.

## Approaches to Handwriting Recognition

In a typical OCR application for printed fonts, the traditional approach is the use of image processing techniques using character models for matching and probing into the input characters after the input characters have been segregated. However, this approach is sensitive to the size of the fonts and the font type. Although much research has been done on omni font recognition, the task has been uphill. For handwritten input, the task becomes even more formidable.

Soft computing has been adopted into the process of character recognition for its ability to create input output mapping with good approximation. The alternative for input/output mapping may be the use of a lookup table that is totally rigid with no room for input variations.

Figure shows the processes used by many character recognition systems. It shows how neural network has been incorporated into character recognition system together with the traditional tasks of segregating the visual inputs into the character components and feature extraction.

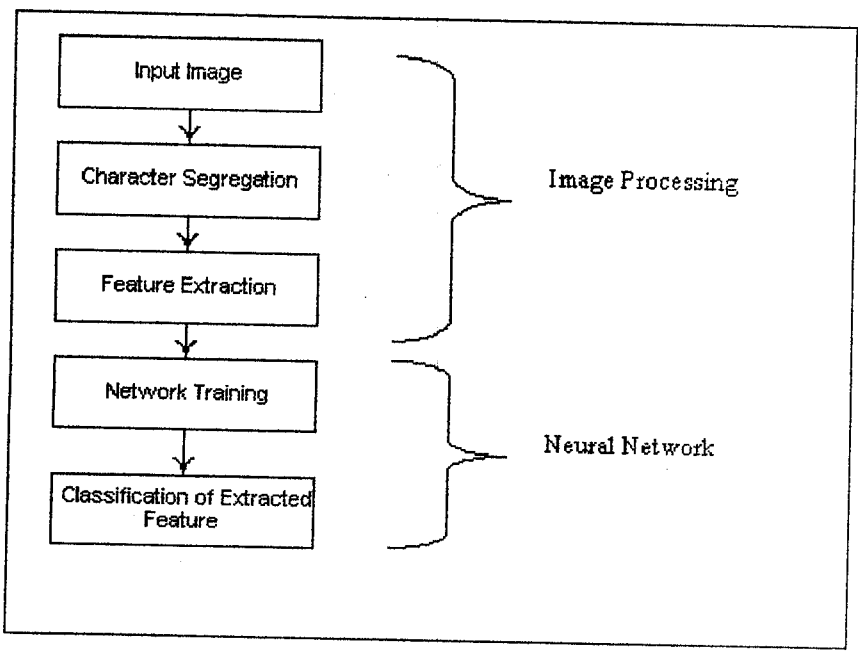



Figure : Processes in Character Recognition

**Handwritten Character Recognition System**

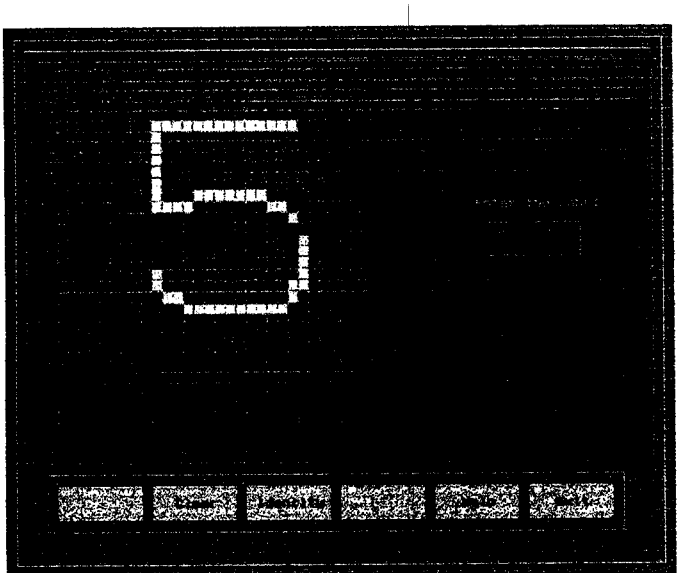
The project work comprises of three phases , they are

1. Data capture phase
2. Feature Extraction phase
3. Training & Recognition phase.

**Data Capture phase**

Inputs to the neural network are given by means of drawing into a picture frame with the help of a mouse. The picture frame is divided into 30 X 30 grid of pixels. The drawing in the picture frame will be scanned by the system.

The input pattern of the system must be digitized into a rectangular picture frame array P={p=(i, j) / 1<=i<=n, 1<=j=m; n, m belongs to N }. The pattern is a binary picture, i.e., the points on the pattern assume the value of one while the other points of the frame take the value of zero. For our recognition system the picture_frame is 30 x 30 array.

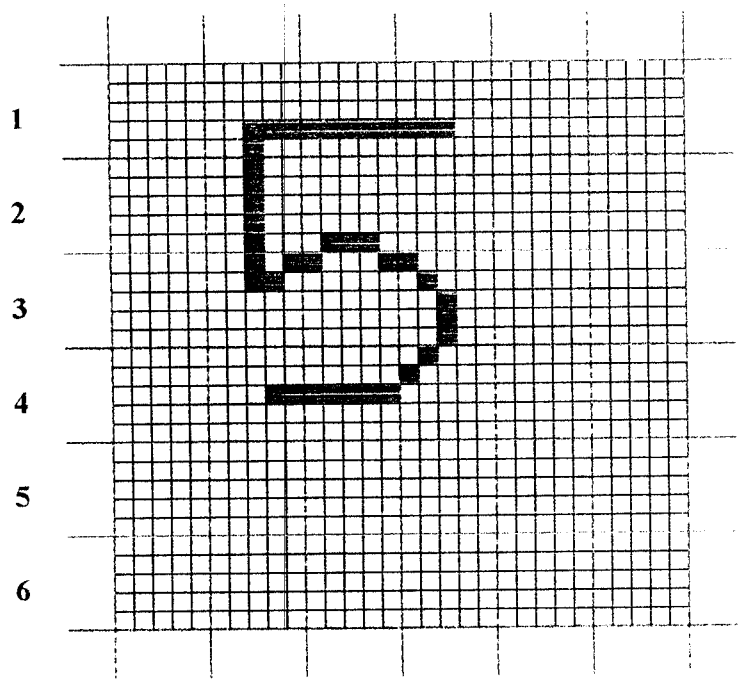


**Feature Extraction**

This process is necessary because not all information of the segregated bitmap character can be used as input to the neural network. Typically the dimension of the area of a segregated character may be 30 by 30 pixels. This would mean that if every pixel information was input into the neural network, 900 input nodes would be required. This is computationally not feasible in a PC environment.

Thus some features must be extracted from the segregated character. One simple feature extraction method used in this project, associates each pixel of the bitmap to one node in the neural network input grid. The number of pixels associated to each cell is then normalized as a proportion of the maximum number of pixels in all the cells.

Figure  shows the feature extraction process. The first diagram on the right shows the number of pixels in each corresponding cell. The second diagram on the right is the normalized results after dividing by the largest number of pixels.




| | | | | | |
|---|---|---|---|---|---|
| 0 | 4 | 5 | 3 | 0 | 0 |
| 0 | 5 | 3 | 0 | 0 | 0 |
| 0 | 4 | 2 | 5 | 0 | 0 |
| 0 | 2 | 5 | 2 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |

| | | | | | |
|---|---|---|---|---|---|
| 0 | 0.8 | 1 | 0.6 | 0 | 0 |
| 0 | 1 | 0.6 | 0 | 0 | 0 |
| 0 | 0.8 | 0.4 | 1 | 0 | 0 |
| 0 | 0.4 | 1 | 0.4 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |

Figure -Feature Extraction

**Training and Recognition phase**

Construction of Neural Network

Input layer

As already mentioned, each character is represented by 6 x 6 pixels, forming an array of 36 input stimuli. Even though the resolution seem, it should be more than adequate to learn the program 36 distinct characters (0-9 & A-Z).

Hidden layer

There doesn't seem to exist a good rule for selecting the number of hidden units needed to perform back propagate successfully. Instead the user has to experiment with hidden number of units to see what gives the most satisfactory result and best speed.

The program uses only one hidden layer since any advantage using more hidden layers could not be observed with the learning and test-data in this project.

Output layer

Since we are particularly interested in the categorization of the characters, we use grandmother cells where one and only one output unit responds to a certain character. We are only interested in digits & upper-case letters and use 36 response units , one for each character. The output unit with highest activity will be selected as the character best matching the input data.

Training neural network

Building a neuron network for any but the most trivial application requires that several network are built, each of different complexity, or stopped at different points during training, or simply started from different random weight configuration. Each network should be saved, tested and analyzed and the most appropriate finally chosen.

When to stop training:

When one or more of the following criteria have been satisfied:

- The average error per pattern for the latest cycle is less than the error tolerance specified by the user.
- The maximum cycle count specified by the user is exceeded.

## CONVENTIONAL BACKPROPAGATION ALGORITHM

Learning algorithms are procedures used for modifying synaptic weights in an orderly fashion. Learning in Neural Network may be classified as supervised / unsupervised. Supervised learning assumes the availability of a teacher who classifies the training patterns into output class, whereas an unsupervised learning does not have pattern classification information. The network architecture and the learning algorithms are linked. The commonly used supervised learning algorithm for feed forward network has certain limitations. To overcome these limitations, certain learning algorithms are proposed.

The popular and commonly used learning algorithm for multilayer feed forward network is Backpropagation Algorithm. This is a gradient descent technique with backward error propagation.

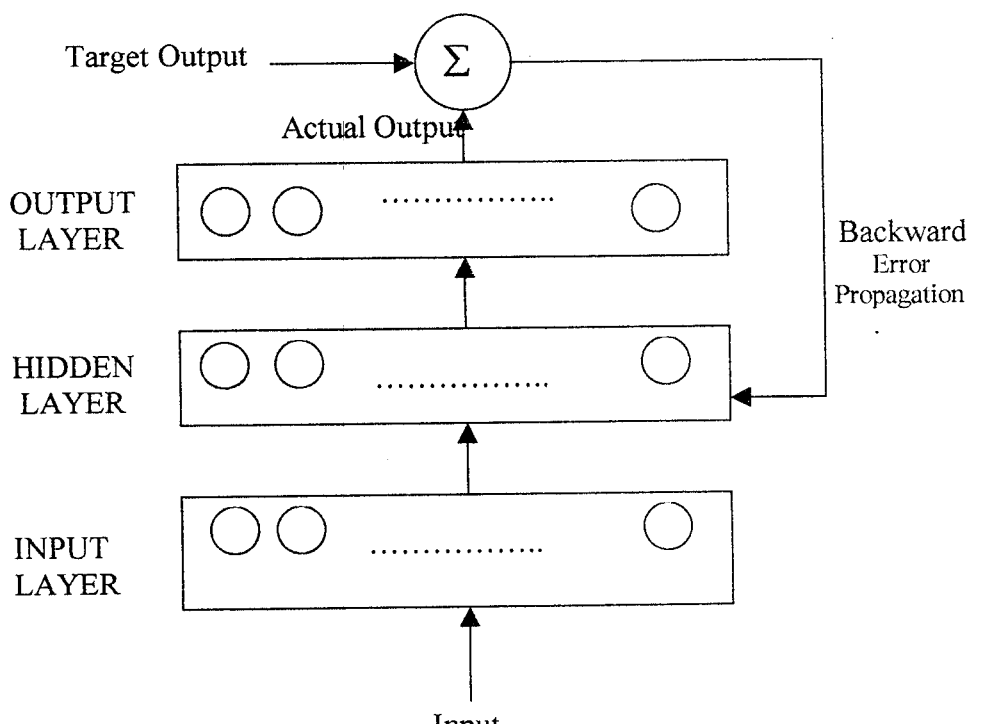



Figure: Backpropagation Network

Backpropagation Algorithm

Step 1 : Weight Initialization

Set all weights and node thresholds to small random numbers. The node threshold is negative of the weight from bias unit ( whose activation level is fixed at 1)

Step 2 : Calculation of Activation

2a. The activation level of an input unit is determined by the instance presented to the network.

2b. The activation level of a neuron in hidden / output layer,

Oj is given by

$$Oj = F(\sum W_{ji}Oi-\theta_j)$$

Where $W_{ji}$ is the weight from an input $O_p$

$\theta_j$ is the node threshold,

and F is the sigmoid function.

$F(a) = 1 / 1 + e^{-a}$

Step 3 : Weight Training

3a. Start at the output units and work backward to the hidden layer recursively. Adjust weights by

$$W_{ji}(t+1) = W_{ji}(t) + \Delta W$$

Where $W_{ji}(t)$ is the weight from unit *i* to unit *j* at time *t*

And $\Delta W_{ji}$ is the weight update.

3b. The weight update is computed by

$$\Delta W_{ji} = \eta\delta_j O_j$$

where $\eta$ is the learning rate.

3c. The error gradient is calculated as :

For output units :

$$\delta^{o}_{j} = O_j(1 - O_j)(T_j - O_j)$$

where $T_j$ is the desired (target) output activation and $O_j$ is the actual output activation at output unit *j*.

$$\delta^{h}_{j} = O_j(1 - O_j) \sum_{k} \delta_k W_k$$

where $\delta_k$ is the error gradient at unit *k* to which a connection points from hidden unit *j*.

3d. Repeat step 2 until convergence.

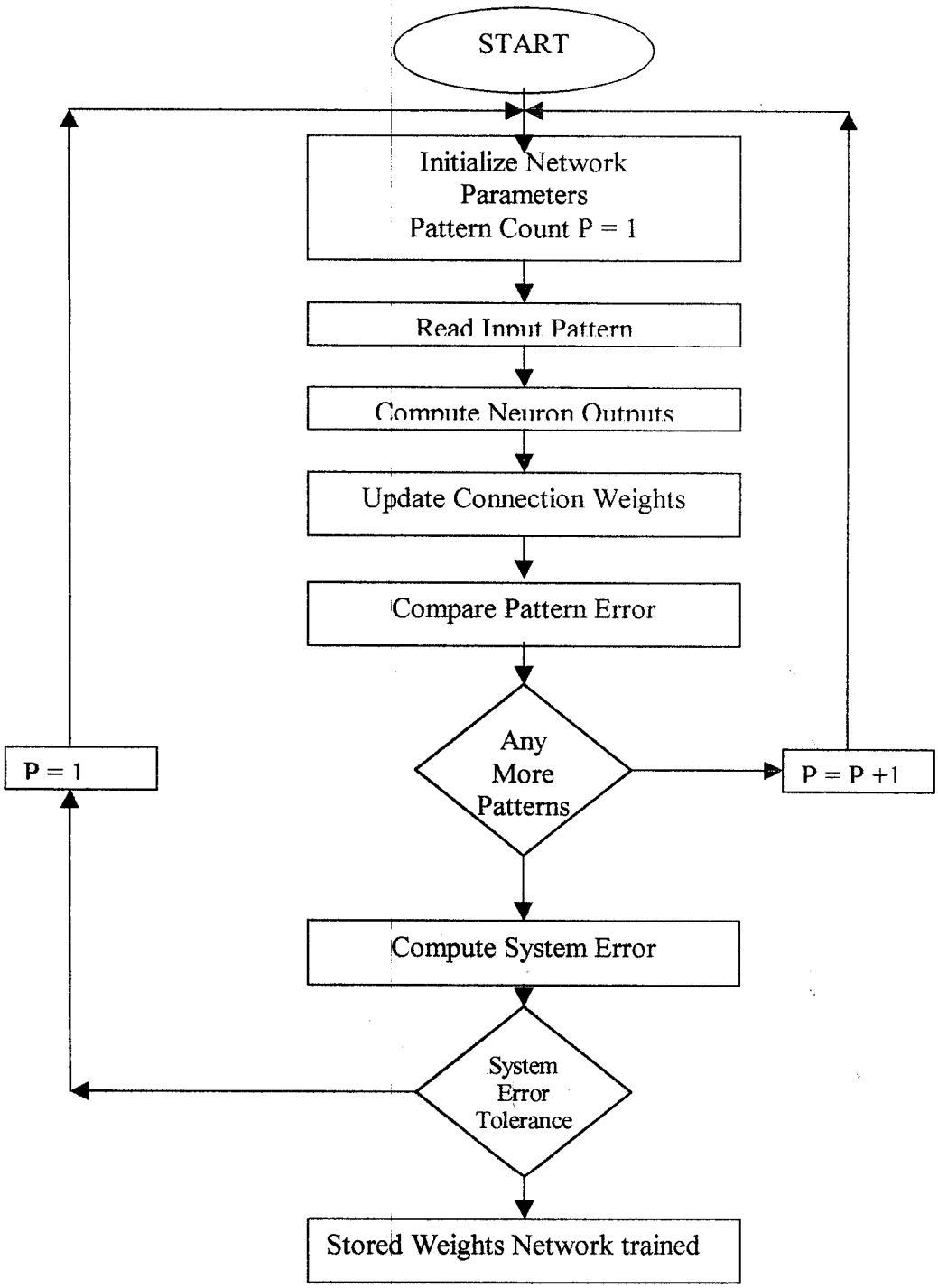



Figure: Flow Chart for Backpropagation

Enhancing the Backpropagation Algorithm

Due to the above limitations of conventional Backpropagation algorithm we add some new features to the simulator : a term called Momentum, and the capability of adding noise to the inputs during simulation. There are many variations of the algorithm that try to alleviate two problems with back propagation. First like other neural networks there is strong possibility that the solution found with back propagation is not a global error minimum, but a local one. You may need to shake the weights a little by some means to get out of the local minimum and possibly arrive at a lower minimum. The second problem with back propagation is speed. The algorithm is very slow at learning. There are many proposals for speeding up the search process. Neural networks are inherently parallel processing architectures and are suited for simulation on parallel processing hardware. While there are a few plug-in neural net or digital signal processing boards available in the market, the low-cost simulation platform of choice remains the personal computer. Speed enhancements to the training algorithm are therefore very necessary.

Adding the Momentum term

A simple change to the training law that sometimes results in much faster training is the addition of a momentum term. The training law for back propagation as implemented in the simulator is

Weight change = Beta * output_error * input

Now we add a term to the weight change equation as follows

Weight change = Beta * output_error * input + Alpha * previous_weight_change

Beta – Learning Parameter , Alpha – Momentum Parameter

The second term in this equation is the momentum term. The weight change in the absence of error , would be constant multiple by the previous weight change. In other words the weight change continues in the direction it was heading. The momentum term is an attempt to try to keep the weight change process moving and thereby not get stuck in local minimas.

Adding Noise During Training

Another approach to breaking out of local minima as well as to enhance generalization ability is to introduce some noise in the inputs during training. A random number is added to each input component of the input vector as it is applied to the network. This is scaled by an overall noise factor NF, which has a 0 to 1 range. You can add as much noise to the simulation as you want, or not any at all, by choosing NF = 0. when you are close to a solution and have reached a satisfactory minimum, you don't want noise at that time to interfere with convergence to the minimum. We implement a noise factor that decreases with number of cycles.

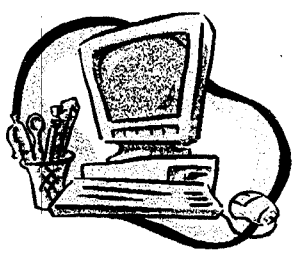
The noise factor is reduced at regular intervals. The new noise factor is updated with the network class function called set_NF(float). There is a member variable in the network class called NF that holds the current value for the noise factor. The noise is added to the inputs in the input_layer member function calc_out().

Another reason for using noise is to prevent memorization by the network. You are effectively presenting a different input pattern with each cycle so it becomes hard for the network to memorize patterns.

# Results

The project was successfully implemented to identify all the handwritten digits with an error tolerance of 0.001. The neural network was constructed and the backpropagation algorithm was used as the training algorithm to train the given neural network. The input to the neural network was normalized from 30x30 grid of pixels to 6x6 grid of pixels. This reduces the number of neurons or input nodes of the neural network

The project identifies handwritten alphabets to a certain extent. Some problems that came across is in the identification of the ambiguous digits 3 and 8 that at times look correspondingly the same. Digits like 0 and alphabet O posed a specific problem in recognition. The same was for the digit 8 and alphabet B and the digit 2 and alphabet Z.The above problems have been incorporated into the recognition system due to the feature extraction method used.



# Conclusion

The project deals with the identification of hand written characters using feed forward backpropagation neural network .A Neural network of the classification model trained using the Backpropagation algorithm seems to be well suited for handwritten character recognition. Many of the problems that arise when using Backpropagation to recognize characters can easily be eliminated or reduced by adding routines that scales, centers the given input pattern.

Several preprocessing techniques such as character thinning, scaling and centering can be incorporated into the project, which would enhance the accuracy of identification of the given hand written characters.

Other problem such as noise and the problem of separating characters exists and there is not much one can do about those problems except improving the quality of the equipment used when scanning/reading characters or using a database or artificial intelligence to give more accurate interpretations.

Detection of network paralysis poses a particular challenge while using the conventional backpropagation algorithm. The conventional backpropagation training algorithm is replaced by the fast learning back propagation training algorithm due to its inherent disadvantages such as network paralysis, local minima and slow convergence.

The project can be extended to convert handwritten material directly into electronic format .It can also be upgraded to allow easy conversion of printed material such as textbooks to electronic format. The project can also be used for signature verification, authentication and for automatic mail sorting facilities in the post offices and for form processing where a large number of a forms should be handled.

# References

Reference Books

1. LiMin Fu, "Neural Networks in Computer Intelligence" McGraw Hill, Inc, 1994

2. Philip D Wasserman, "Neural Computing", Van Nostrand Reinhold, New York, 1989

3. Valluru Rao and Hayagriva Rao, "C ++ Neural Networks and Fuzzy Logic", BPB Publications, 1996

4. J.R. Ullmann, M.A.Ph.D "Pattern Recognition Techniques "

   The Butterworth group ,1973.

5. Nils J.Nilsson, "Principles of Artificial Intelligence", N.K.Mehra for Narosa Publishing House, NewDelhi – 17

6. Bart Kosko – University of Southern California, "Neural Networks & Fuzzy Systems",Prentice Hall of India Private Limited NewDelhi – 1, Fourth Indian Reprint N.K.bose, P.Liang,

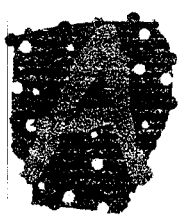
Websites referenced

http://www.eng.buffalo.edu/Calendar/99/CEDAR.html

http://www.cis.hut.fi/projects/hcr/

http://www.cs.wisc.edu/~dyer/cs540/hw/hw4/hw4.html

http://www.starhub.net.sg/~kwyang/FinalReport1.htm



# Appendix

**Main program for handwritten character recognition:**

```
#include<graphics.h>
#include<iostream.h>
#include<conio.h>
#include<dos.h>
#include<stdlib.h>
#include<stdio.h>
#include<string.h>
#include<fstream.h>
#include<alloc.h>
#include<dir.h>

#include "Graphics.cpp"
#include "Button.cpp"
#include "Mouse.cpp"
#include "backprop.cpp"

#define INPUT_FILE    "input.dat"      // WRITE
#define FEATURE_FILE        "features.dat"  // APPEND

class Main
{
Mouse ms;
Graphics g;
Button bDraw,bClear,bIdentify,bTrain,bHelp,bExit;
int blDraw;
public:
  void start();
  void process();
  void end();
};

void Main::start()
{
blDraw = 0;
g.initGraphics();
g.drawRect(10,10,getmaxx()-10,getmaxy()-10);
g.drawRect(20,20,getmaxx()-20,380);
g.drawRect(40,390,590,440);

ms.initMouse();
ms.showMouse();

bDraw.setPos(50,400,130,430);
bDraw.setText("Draw");
```

```
bDraw.add();

bClear.setPos(140,400,220,430);
bClear.setText("Clear");
bClear.add();

bIdentify.setPos(230,400,310,430);
bIdentify.setText("Identify");
bIdentify.add();

bTrain.setPos(320,400,400,430);
bTrain.setText("Train");
bTrain.add();

bHelp.setPos(410,400,490,430);
bHelp.setText("Help");
bHelp.add();

bExit.setPos(500,400,580,430);
bExit.setText("Exit");
bExit.add();
}
void Main::process()
{
int xpos,ypos;
int trainFlag;

while(1)
{
ms.getPos();
xpos = ms.x;
ypos = ms.y;
if(ms.button == 1)
{
if(bExit.ptInRect(xpos,ypos))
{
  ms.hideMouse();
  bExit.click();
  ms.showMouse();
  end();
  exit(0);
}
else if(bDraw.ptInRect(xpos,ypos))
{
  blDraw = 1;
  ms.hideMouse();
```

```
    bDraw.click();
    g.drawGrid();
    ms.showMouse();
  }
  else if(bClear.ptInRect(xpos,ypos))
  {
    ms.hideMouse();
    bClear.click();
    g.clearGrid();
    ms.showMouse();
  }
  else if(bIdentify.ptInRect(xpos,ypos))
  {
    ms.hideMouse();
    bIdentify.click();
    g.convert(INPUT_FILE);
    g.initValues();
  /*        g.thinning();
    g.clearGrid();
    g.fillGrid();
    g.drawString("After thinning and noise removal",360,150);
    getch();*/
    g.fExtract(TEST_FILE,0);
    trainFlag = 0;
    g.clrscr(25,25,getmaxx()-25,375);
    BPNetwork bp;
    bp.BPProcess(trainFlag);
    ms.showMouse();

  }
  else if(bTrain.ptInRect(xpos,ypos))
  {
    ms.hideMouse();
    bTrain.click();

    g.convert(INPUT_FILE);
    g.initValues();
  /*        g.thinning();
    g.clearGrid();
    g.fillGrid();

    g.drawString("After thinning and noise removal",360,150);
    getch();*/
    g.clrscr(360,25,getmaxx()-25,375);
    g.getTarget(); // only when train the network
    g.fExtract(FEATURE_FILE,1);
```

```
  trainFlag = 1;
  g.clrscr(25,25,getmaxx()-25,375);



    BPNetwork bp;
  bp.BPProcess(trainFlag);

  ms.showMouse();
}
else if(bHelp.ptInRect(xpos,ypos))
{
  ms.hideMouse();
  bHelp.click();
  ms.showMouse();
}

else if(g.ptInGrid(xpos,ypos) && blDraw)
{
  ms.hideMouse();
  g.fillRect(xpos,ypos);
  ms.showMouse();
}
}
}
}
void Main::end()
{
g.closeGraphics();
}
int main()
{
Main m;
m.start();
m.process();
m.end();
return 0;
}
```

**Header file for graphical user interface program:**

```
#define chk_arr 8
#define TRAIN 1
#define TEST  0
class Graphics
{
int gd,gm;
int pic_array[50][50];
int t_pics[50][50];
int *mark_fil[110];
int *mark_del[610];

int values[8][8];

int del_count;
int fil_count;
int read_flag;

int del_size,fil_size;
float target[36];

public:
        Graphics(){ gd =DETECT;del_size=600;fil_size=100;}
        void initGraphics();
        void closeGraphics();
        void drawRect(int l,int t,int r,int b);
        void fillRect(int xpos,int ypos);
        void clrscr(int l,int t,int r,int b);
        void clearGrid();
        void drawGrid();
        void drawString(char *string,int x,int y);
        int ptInGrid(int x,int y);
        void convert(char* filename);
        void thinning();
        void fExtract(char *filename,int mode);
        void setWindow(int l,int t,int r,int b,int clip);
        void setFont(int,int,int);
        void getTarget();
        void initValues();
        int validrange(int,int,int,int);
        int readvalues(int,int,int,int);
        int compare(int,int,int[][8]);
        int neighbours(int,int);
        int connectivity();
        void bigholes(int,int,int);
```

```
        void holes();
        void delfill();
        void removestrays();
        void updowndelete();
        void thinem();
        void fillGrid();
        void saveresult();
};
```

## C++ Program for implementing Graphical User Interface

```
#include "Graphics.h"

#define GLEFT          50
#define GTOP    50
#define GRIGHT  350
#define GBOTTOM 350
#define GRAY  8

void Graphics::initValues()
{
read_flag = 1;
for(int i=0;i<36;i++)
target[i]=0.0;
}
void Graphics::initGraphics()
{
initgraph(&gd,&gm,"");
}
void Graphics::closeGraphics()
{
closegraph();
}
void Graphics::drawRect(int l,int t,int r,int b)
{
rectangle(l,t,r,b);
}

void Graphics::drawGrid()
{
/* grid size : 30*30 */
clrscr(25,25,getmaxx()-25,375);
setWindow(0,0,getmaxx(),getmaxy(),1);

int row,col;
int i,j;
for(i=0;i<30;i++)
for(j=0;j<30;j++)
        pic_array[i][j]=0;

settextjustify(0,0);
setcolor(GRAY);

for(col = GLEFT;col<GRIGHT;col+=10)
{
```

```
        for(row = GTOP;row<GBOTTOM;row+=10)
        {
                rectangle(col,row,col+10,row+10);
        }
        }
        setcolor(WHITE);
        }
        void Graphics::fillRect(int xpos,int ypos)
        {
        //if(getpixel(xpos+5,ypos+5) == BLACK)
        //{
        setfillstyle(SOLID_FILL,WHITE);
        floodfill(xpos,ypos,GRAY);
        //}
        /*else if(getpixel(xpos+5,ypos+5) == WHITE)
        {
        setfillstyle(SOLID_FILL,BLACK);
        floodfill(xpos,ypos,GRAY);
        }*/
        }
        void Graphics::clearGrid()
        {
        int row,col;

        setfillstyle(SOLID_FILL,BLACK);

        for(col = GLEFT;col<GRIGHT;col+=10)
        {
        for(row = GTOP;row<GBOTTOM;row+=10)
        {
        if(getpixel(col+5,row+5) == WHITE)
                floodfill(col+5,row+5,GRAY);
        }
        }
        }
        int Graphics::ptInGrid(int x,int y)
        {
                if(x >=GLEFT && x <=GRIGHT && y >=GTOP && y
<=GBOTTOM)
                        return 1;
                else
                        return 0;
        }
        void Graphics::convert(char *filename)
        {
        int row,col;
```

```
int i,j;
i=j=0;

ofstream of(filename,ios::app);
for(row = GTOP;row<GBOTTOM;row+=10)
{
for(col = GLEFT;col<GRIGHT;col+=10)
{
if(getpixel(col+5,row+5) == WHITE)
{
        pic_array[i][j] = 1;
/* make 8 directions to 1 */

/*      if(pic_array[i][j-1] == 0)
                pic_array[i][j-1] = 1;
        if(pic_array[i][j+1] == 0)
                pic_array[i][j+1] = 1; // right
        if(pic_array[i-1][j] == 0)
                pic_array[i-1][j] = 1; // top
        if(pic_array[i+1][j] == 0)
                pic_array[i+1][j] = 1;*/
/*      pic_array[i-1][j-1] = 1;
        pic_array[i+1][j-1] = 1;
        pic_array[i-1][j+1] = 1;
        pic_array[i+1][j+1] = 1;*/

        of<<"1 ";
}
else
{
        pic_array[i][j] = 0;
        of<<"0 ";
}
j++;
}
i++;
j=0;
of<<"\n";
}
of.close();
}

void Graphics::fExtract(char *filename,int mode)
{

int flag=0;
```

```
for(int i=0;i<30;i++)
{
for(int j=0;j<30;j++)
{
        if(pic_array[i][j] == 1)
        {
                flag = 1;
                break;
        }
}
}
if(!flag)
{
cout<<"no pattern is present";
exit(0);
}


int row,col;

row = GTOP;
col = GLEFT;
int count = 0;
int pass=0;
int cnt = 0;

ofstream of;
/* extract into 6*6 array */

/*while(1)
{
count = 0;
if((row >= GBOTTOM && col >= GRIGHT))
{
        cout<<"end";
        break;
}

if(row >= GBOTTOM)          // reset row
{
//      of<<"\n";
        if(pass == 5)
                break;
        pass++;
        row = GTOP;
```

```
            col = GLEFT +((pass*5)*10);          // reset col
    }
    else
            col = GLEFT;

    for(int i=1;i<=5;i++)
    {
    for(int j=1;j<=5;j++)
    {
    if(getpixel(col+5,row+5) == WHITE)
            count++;
    col+=10;
    }
    row+=10;
    col = GLEFT +((pass*5)*10);          // reset col
    }
    of<<count<<" ";
    }*/
    int pixarr[10][10]={0};
    float cntarr[10][10]={0};

    int r,c;
    r=c=0;


    while(1)
    {
    count = 0;
    if((row >= GBOTTOM && col >= GRIGHT))
    {
            break;
    }

    if(col >= GRIGHT)    // reset row
    {
            r+=1;
            if(pass == 5)
                    break;
            cnt=0;
            pass++;
            col = GLEFT;
    }

    row = GTOP +((pass*5)*10);// reset col

    for(int i=1;i<=5;i++)
```

```
{
col = GLEFT +((cnt*5)*10); // reset col
for(int j=1;j<=5;j++)
{
if(getpixel(col+5,row+5) == WHITE)
        count++;
col+=10;
}
row+=10;
}
cnt++;
//pixarr[r][c] = count*1.0f;
pixarr[r][c] = count;
c++;
}
float max = pixarr[0][0];

for(i=0;i<6;i++)
{
for(int j=0;j<6;j++)
{
        if(pixarr[i][j] > max)
                max = pixarr[i][j];
}
}
if(mode == 1)
        of.open(filename,ios::app);
else
        of.open(filename,ios::out);
for(i=0;i<6;i++)
{
for(int j=0;j<6;j++)
{
cntarr[i][j] = ((pixarr[i][j])*1.0f)/ (max*1.0f);
of<<cntarr[i][j]<<" ";
//fprintf(of,"%f ",pixarr[i][j]);
}
of<<"\n";
//fprintf(of,"%c",'\n');
}
if(mode == 1)
{
for(i=0;i<36;i++)
of<<target[i]<<" ";
of<<"\n";
}
```

```
of.close();

//fclose(of);
}
void Graphics::clrscr(int l,int t,int r,int b)
{
setfillstyle(SOLID_FILL,BLACK);
bar(l,t,r,b);
//bar(25,25,getmaxx()-25,375);
}
void Graphics::drawString(char string[150],int x,int y)
{
settextjustify(LEFT_TEXT,TOP_TEXT);
outtextxy(x,y,string);
}
void Graphics::setWindow(int l,int t,int r,int b,int clip)
{
setviewport(l,t,r,b,clip);
}
void Graphics::getTarget()
{
        char temp;

        drawString("Enter the Label",450,150);
        drawRect(450,200,550,170);
        gotoxy(60,12);
        cin>>temp;
        if(temp >='0' && temp <= '9')
                target[temp-48]=1.0;
        else if(temp >='A' && temp <= 'Z')
                target[temp-55]=1.0;
        else
        {
                cout<<"Invalid character(only digits and capital letters are
recognized by this system";
                exit(1);
        }
}
void Graphics::thinning()
{
register int i;
del_count = -1;
fil_count = -1;
for(i=0;i<del_size;i++)
        mark_del[i]=(int*)farmalloc(2*sizeof(int));
for(i=0;i<fil_size;i++)
```

```
        mark_fil[i]=(int*)farmalloc(2*sizeof(int));
holes();
delfill();
removestrays();
updowndelete();
thinem();
for(i=0;i<del_size;i++)
farfree(mark_del[i]);
for(i=0;i<fil_size;i++)
farfree(mark_fil[i]);
saveresult();
}
void Graphics::saveresult()
{
ofstream ofs("thinning.dat",ios::out);
for(int i=0;i<30;i++)
{
for(int j=0;j<30;j++)
        ofs<<pic_array[i][j]<<" ";
ofs<<"\n";
}
ofs.close();
}
int Graphics::validrange(int sy,int sx,int uy,int ux)
{
/* check if the reading range of values is valid */
if((sx >= 0 ) && (sy >=0) && ((sx+ux)< 30) && ((sy+uy) < 30))
        return 1;
return 0;
}
int Graphics::readvalues(int sy,int sx,int uy,int ux)
{
/* read a (uy-ux) x (ux-sx) matrix from pic_array to values */
int flag=0;
register int i,j;
if(validrange(sy,sx,uy,ux))
{
        flag= 1;
        for(i=sy;i<(sy+uy);i++)
        for(j=sx;j<(sx+ux);j++)
                if(read_flag)
                        values[i-sy][j-sx]=pic_array[i][j];
                else
                        values[i-sy][j-sx]=t_pics[i][j];
}
return flag;
```

```
}
int Graphics::compare(int ry,int rx,int mat[][chk_arr])
{
/* used for comparing ry x rx elements of values with mat */
register int i,j;
int flag=1;
for(i=0;i<ry;i++)
for(j=0;j<rx;j++)
        if((mat[i][j] != -1) && (mat[i][j] != values[i][j]))
        {
                flag=0;
                break;
        }
return flag;
}
int Graphics::neighbours(int y,int x)
{
/* finds the no. of neighbours of pixel y,x */
int nb=0;
register int i,j;
readvalues(y-1,x-1,3,3);
for(i=0;i<3;i++)
for(j=0;j<3;j++)
        if(values[i][j] == 1) nb++;
nb -= values[1][1];
return nb;
}
int Graphics::connectivity()
{
/* computer the connectivity of the center pixel of values */
int conn=0;
int mat[10];
int i;
mat[4]=1-values[0][0];
mat[3]=1-values[0][1];
mat[2]=1-values[0][2];
mat[5]=1-values[1][0];
mat[0]=1-values[1][1];
mat[1]=1-values[1][2];
mat[6]=1-values[2][0];
mat[7]=1-values[2][1];
mat[8]=1-values[2][2];
mat[9]=mat[1];

for(i=1;i<=7;i+=2)
        conn+=mat[i] - (mat[i] * mat[i+1] * mat[i+2]);
```

```
return conn;
}
void Graphics::bigholes(int y,int x,int type)
{
int fill_flag=1;
int i_1[7][chk_arr]={ {-1,-1,1,1,1,-1,-1},
                      {-1,1,1,1,1,1,-1},
                      {-1,1,1,1,1,1,-1},
                      {1,1,1,0,1,1,1},
                      {-1,1,1,1,1,1,-1},
                      {-1,1,1,1,1,1,-1},
                      {-1,-1,1,1,1,-1,-1}
                     };

int i_2[8][chk_arr]={ {-1,-1,1,1,1,-1,-1},
                      {-1,1,1,1,1,1,-1},
                      {-1,1,1,1,1,1,-1},
                      {1,1,1,0,1,1,1},
                      {1,1,1,0,1,1,1},
                      {-1,1,1,1,1,1,-1},
                      {-1,1,1,1,1,1,-1},
                      {-1,-1,1,1,1,-1,-1}
                     };
int i_3[7][chk_arr]={ {-1,-1,1,1,1,1,-1,-1},
                      {-1,-1,1,1,1,1,1,-1,-1},
                      {-1,1,1,1,1,1,1,1,-1},
                      {1,1,1,0,0,1,1,1},
                      {-1,1,1,1,1,1,1,1,-1},
                      {-1,-1,1,1,1,1,1,-1,-1},
                      {-1,-1,1,1,1,1,1,-1,-1}
                     };
if(readvalues(y-3,x-3,7,7))
        if(compare(7,7,i_1))
                fill_flag=0;

if(readvalues(y-3,x-3,8,7))
        if(compare(8,7,i_2))
                fill_flag=0;
if(readvalues(y-3,x-3,7,8))
        if(compare(7,8,i_3))
                fill_flag=0;
if(fill_flag)
        switch(type)
        {
        case 1: mark_fil[++fil_count][0]=y;
                        mark_fil[fil_count][1]=x;break;
```

```
        case 2:
                        mark_fil[++fil_count][0]=y;
                        mark_fil[fil_count][1]=x;
                        mark_fil[++fil_count][0]=y+1;
                        mark_fil[fil_count][1]=x;
                        break;
        case 3:
                        mark_fil[++fil_count][0]=y;
                        mark_fil[fil_count][1]=x;
                        mark_fil[++fil_count][0]=y;
                        mark_fil[fil_count][1]=x+1;
                        break;
        }
}

void Graphics::holes()
{
/* used to find holes */

register int i,j;
int h1[3][chk_arr] = { {-1,1,-1},
                                        {1,0,1},
                                        {-1,1,-1},
                                       };
int h2[3][chk_arr] = { {-1,1,1,-1},
                                        {1,0,0,1},
                                        {-1,1,1,-1},
                        };
int h3[4][chk_arr] = { {-1,1,-1},
                                        {1,0,1},
                                        {1,0,1},
                                        {-1,1,-1}
                                       };
for(i=0;i<30;i++)
for(j=0;j<30;j++)
        if(pic_array[i][j] == 0)
        {
                if(readvalues(i-1,j-1,3,3))
                        if(compare(3,3,h1))
                                if(pic_array[i-1][j-1] == 0)
                                {
                                        mark_del[++del_count][0]=i-1;
                                        mark_del[del_count][1]=j;
                                        mark_del[++del_count][0]=i;
                                        mark_del[del_count][1]=j-1;
                                }
```

```
else if(pic_array[i-1][j+1] == 0)
                {
                        mark_del[++del_count][0]=i-1;
                        mark_del[del_count][1]=j;
                        mark_del[++del_count][0]=i;
                        mark_del[del_count][1]=j+1;
                }
                else if(pic_array[i+1][j+1] == 0)
                {
                        mark_del[++del_count][0]=i;
                        mark_del[del_count][1]=j+1;
                        mark_del[++del_count][0]=i+1;
                        mark_del[del_count][1]=j;
                }
                else if(pic_array[i+1][j-1] == 0)
                {
                        mark_del[++del_count][0]=i;
                        mark_del[del_count][1]=j-1;
                        mark_del[++del_count][0]=i+1;
                        mark_del[del_count][1]=j;
                }
                else
                        bigholes(i,j,1);
                if(readvalues(i-1,j-1,3,4))
                        if(compare(3,4,h2))
                                if(pic_array[i-1][j-1] == 0)
                                {
                                mark_del[++del_count][0] = i-1;
                                mark_del[del_count][1] = j;
                                mark_del[++del_count][0]=i;
                                mark_del[del_count][1]=j-1;
                                }
                        else if(pic_array[i-1][j+2] == 0)
                                {
                                mark_del[++del_count][0] = i-1;
                                mark_del[del_count][1] = j+1;
                                mark_del[++del_count][0]=i;
                                mark_del[del_count][1]=j+2;
                                }
                        else if(pic_array[i+1][j+2] == 0)
                                {
                                mark_del[++del_count][0] = i;
                                mark_del[del_count][1] = j+2;
                                mark_del[++del_count][0]=i+1;
                                mark_del[del_count][1]=j+1;
                                        }
```

```
else if(pic_array[i+1][j-1] == 0)
{
mark_del[++del_count][0] = i;
mark_del[del_count][1] = j-1;
mark_del[++del_count][0]=i+1;
mark_del[del_count][1]=j;
}
else
bigholes(i,j,2);


if(readvalues(i-1,j-1,4,3))
if(compare(4,3,h3))
if(pic_array[i-1][j-1] == 0)
{
mark_del[++del_count][0] = i-1;
mark_del[del_count][1] = j;
mark_del[++del_count][0]=i;
mark_del[del_count][1]=j-1;
}
else if(pic_array[i-1][j+1] == 0)
{
mark_del[++del_count][0] = i-1;
mark_del[del_count][1] = j;
mark_del[++del_count][0]=i;
mark_del[del_count][1]=j+1;
}
else if(pic_array[i+2][j+1] == 0)
{
mark_del[++del_count][0] = i+1;
mark_del[del_count][1] = j+1;
mark_del[++del_count][0]=i+2;
mark_del[del_count][1]=j;
}
else if(pic_array[i+2][j-1] == 0)
{
mark_del[++del_count][0] = i+1;
mark_del[del_count][1] = j-1;
mark_del[++del_count][0]=i+2;
mark_del[del_count][1]=j;
}
else
        bigholes(i,j,3);
}
```

```
}
void Graphics::delfill()
{
/* deletes and fills marked elements */
int i;
for(i=0;i<=del_count;i++)
        pic_array[mark_del[i][1]][mark_del[i][0]]=0;
for(i=0;i<=fil_count;i++)
        pic_array[mark_fil[i][1]][mark_fil[i][0]]=1;
}
void Graphics::removestrays()
{
/* remove isolated pixels */

register int i,j;
for(i=0;i<30;i++)
for(j=0;j<30;j++)
if(pic_array[i][j] == 1)
        if(neighbours(i,j) < 3)
                if(connectivity() < 2)
                        pic_array[i][j]=0;
}
void Graphics::updowndelete()
{
/* acute angle emphasis*/

register int i,j;
int del=0,any_del=0;

int d1[5][chk_arr]={ {1,1,0,1,1},
                     {1,1,0,1,1},
                     {1,1,1,1,1},
                     {1,1,1,1,1},
                     {-1,1,1,1,-1}
                   };

int d2[5][chk_arr]={ {1,0,0,1,1},
                     {1,1,0,1,1},
                     {1,1,1,1,1},
                     {1,1,1,1,1},
                     {-1,1,1,1,-1}
                   };
```

```
int d3[5][chk_arr]={ {1,1,0,0,1},
                     {1,1,0,1,1},
                     {1,1,1,1,1},
                    {1,1,1,1,1},
                    {-1,1,1,1,-1}
                   };
int d4[5][chk_arr]={ {1,0,0,1,1},
                    {1,0,0,1,1},
                    {1,1,1,1,1},
                    {1,1,1,1,1},
                    {-1,1,1,1,-1}
                   };
int d5[5][chk_arr]={ {1,1,0,0,1},
                    {1,1,0,0,1},
                    {1,1,1,1,1},
                    {1,1,1,1,1},
                    {-1,1,1,1,-1}
                   };
int u1[5][chk_arr]={ {-1,1,1,1,-1},
                    {1,1,1,1,1},
                        {1,1,1,1,1},
                    {1,1,0,1,1},
                        {1,1,0,1,1}
                   };

int u2[5][chk_arr]={ {-1,1,1,1,-1},
                    {1,1,1,1,1},
                    {1,1,1,1,1},
                    {1,1,0,1,1},
                    {1,1,0,0,1}
                   };
int u3[5][chk_arr]={ {-1,1,1,1,-1},
                    {1,1,1,1,1},
                    {1,1,1,1,1},
                    {1,1,0,1,1},
                    {1,0,0,1,1}
                   };
int u4[5][chk_arr]={ {-1,1,1,1,-1},
                    {1,1,1,1,1},
                    {1,1,1,1,1},
                        {1,0,0,1,1},
                    {1,0,0,1,1}
                   };
```

```
int u5[5][chk_arr]={ {-1,1,1,1,-1},
                     {1,1,1,1,1},
                     {1,1,1,1,1},
                     {1,1,0,0,1},
                     {1,1,0,0,1}
                    };
del_count=-1;
fil_count=-1;
for(i=0;i<30;i++)
for(j=0;j<30;j++)
        if(pic_array[i][j] == 1)
                if(readvalues(i-2,j-2,5,5))
                {
                        del=0;
                        if(compare(5,5,d1)) del=1;
                        if(compare(5,5,d2)) del=1;
                        if(compare(5,5,d3)) del=1;
                        if(compare(5,5,d4)) del=1;
                        if(compare(5,5,d5)) del=1;

                        if(compare(5,5,u1)) del=1;
                        if(compare(5,5,u2)) del=1;
                        if(compare(5,5,u3)) del=1;
                        if(compare(5,5,u4)) del=1;
                        if(compare(5,5,u5)) del=1;

                        if(del)
                        {
                                any_del = 1;
                                mark_del[++del_count][0] = j;
                                mark_del[del_count][1] = i;
                        }
                        }
                        delfill();

                        if(any_del)
                        for(i=0;i<30;i++)
                        for(j=0;j<30;j++)
                                if(pic_array[i][j] == 1)
                                        if(readvalues(i-2,j-2,5,5))
                                        {
                                                del=0;
                                                any_del=0;
                                                if(compare(5,5,d1)) del = 1;
                                                if(compare(5,5,d2)) del = 1;
                                                if(compare(5,5,d3)) del = 1;
```

```
if(compare(5,5,u1)) del = 1;
if(compare(5,5,u2)) del = 1;
if(compare(5,5,u3)) del = 1;

if(del)
{
any_del = 1;
mark_del[++del_count][0]=j;
mark_del[del_count][1]=i;
}
}
delfill();


if(any_del)
for(i=0;i<30;i++)
for(j=0;j<30;j++)
if(pic_array[i][j] == 1)
if(readvalues(i-2,j-2,5,5))
{
del=0;
any_del=0;
if(compare(5,5,d1)) del = 1;
if(compare(5,5,u1)) del = 1;

if(del)
{
any_del = 1;
mark_del[++del_count][0]=j;
mark_del[del_count][1]=i;
}
}
delfill();
}

void Graphics::thinem()
{
register int i,j;
int no_del=0;
int m1[3][chk_arr] = { {-1,0,-1},
{-1,1,1},
{-1,1,1}
};
```

```
int m2[3][chk_arr] = { {-1,-1,-1},
{0,1,1},
                       {-1,-1,-1}
                      };
int m3[3][chk_arr] = { {-1,1,-1},
                       {-1,1,-1},
                       {-1,0,-1}
                      };
int m4[3][chk_arr] = { {-1,-1,-1},
                       {1,1,0},
                       {-1,-1,-1}
                    };

del_count=-1;
while(!no_del)
{
no_del = 1;
for(i=0;i<30;i++)
for(j=0;j<30;j++)
        t_pics[i][j] = pic_array[i][j];

for(i=0;i<30;i++)
for(j=0;j<30;j++)
        if(pic_array[i][j] == 1)
                if(readvalues(i-1,j-1,3,3))
                        if(compare(3,3,m1))
                        {
                                read_flag=0;
                                if(neighbours(i,j) >1)
                                if(connectivity() == 1)
                                {
                                        no_del=0;
                                        mark_del[++del_count][0]=j;
                                        mark_del[del_count][1]=i;
                                        t_pics[i][j] = 0;
                                }
                                read_flag = 1;
                        }

 for(j=0;j<30;j++)
 for(i=29;i>=0;i--)
         if(pic_array[i][j] == 1)
                 if(readvalues(i-1,j-1,3,3))
                         if(compare(3,3,m2))
                         {
                                 read_flag=0;
```

```
if(neighbours(i,j) >1)
if(connectivity() == 1)
{
no_del=0;
mark_del[++del_count][0]=j;
mark_del[del_count][1]=i;
t_pics[i][j] = 0;
}
read_flag = 1;
}

for(i=29;i>=0;i--)
for(j=29;j>=0;j--)
        if(pic_array[i][j] == 1)
                if(readvalues(i-1,j-1,3,3))
                        if(compare(3,3,m3))
                        {
                                read_flag=0;
                                if(neighbours(i,j) >1)
                                if(connectivity() == 1)
                                {
                                        no_del=0;
                                        mark_del[++del_count][0]=j;
                                        mark_del[del_count][1]=i;
                                        t_pics[i][j] = 0;
                                }
                                read_flag = 1;
                        }

for(j=29;j>=0;j--)
for(i=0;i<30;i++)
        if(pic_array[i][j] == 1)
                if(readvalues(i-1,j-1,3,3))
                        if(compare(3,3,m4))
                        {
                                read_flag=0;
                                if(neighbours(i,j) >1)
                                if(connectivity() == 1)
                                {
                                        no_del=0;
                                        mark_del[++del_count][0]=j;
                                        mark_del[del_count][1]=i;
                                        t_pics[i][j] = 0;
                                }
                                read_flag = 1;
                        }
```

```
delfill();
}
}
void Graphics::fillGrid()
{
int row,col;
int i,j;
i=j=0;

for(row = GTOP;row<GBOTTOM;row+=10)
{
for(col = GLEFT;col<GRIGHT;col+=10)
{
        if(pic_array[i][j] == 1)
        {
                setfillstyle(SOLID_FILL,WHITE);
                floodfill(col+5,row+5,GRAY);
        }
        else
        {
                setfillstyle(SOLID_FILL,BLACK);
                floodfill(col+5,row+5,GRAY);
        }
j++;
}
i++;
j=0;
}
}
void Graphics::setFont(int i,int j,int k)
{
settextstyle(i,j,k);
}
```