COM
1496
c-2.

Trainer, Classifier & Post-Processor for Urdu OCR SDK

QUAID-I-AZAM UNIVERSITY
ISLAMABAD

By

Muhammad Ali

A report submitted to Quaid-i-Azam University Islamabad
As a partial fulfillment of the requirements for the
Degree of M.Sc. in Computer Science

February 2004

QUAID-I-AZAM UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

Dated: __________, 2004

FINAL APPROVAL

This is to certify that we have read the project report submitted by Mr. Muhammad Ali and it is our judgment that this report is of sufficient standard to warrant its acceptance by the Quaid-i-Azam University, Islamabad for the degree of Master of Science in Computer Science.

COMMITTEE:

1. External Examiner

Dr. Abdul Qadar [illegible]
Muhammad Ali Jinnah University, Islamabad Campus
74-E, Blue Area, Jinnah Ave.
Islamabad

2. Supervisor

Mr. Asif Qumer [illegible]
Lecturer
Department of Computer Science
Quaid-i-Azam University
Islamabad

3. Chairperson

Professor Dr. Farhana Shah [illegible]
Department of Computer Science
Quaid-i-Azam University
Islamabad

To my loving Parents.

Project Brief

Project Title:	Urdu OCR SDK (Trainer, Classifier and Post-Processor)
Undertaken By:	Muhammad Ali
Supervised By:	Sir Asif Qumar Lecturer Department of Computer Science Quaid-i-Azam University, Islamabad
Starting Date:	September 2003.
Completion Date:	February 2004.
Tool Used:	Microsoft Visual C++ 6.0
Operating System:	Microsoft Windows 2000 Professional.
System Used:	Intel® Pentium® 4 CPU 1.70GHz.

Abstract

Optical Character recognition (OCR) is one of the most successful applications of automatic pattern recognition. OCR technology is used to extract from filling form, and to automate data entry processes. It is prerequisite for speech production systems to aid the blind for reading.

Here we restrict our problem to handwritten isolated characters of Urdu language to be considered for recognition. A gray scale image of the character to be recognized is obtained by scanning the character using a standard scanner having a resolution of 300 DPI. The Classifier or Recognizer for images is based upon theory of Artificial Neural Networks.

The objective of proposed software system "Urdu OCR SDK" is to develop a Software Development Kit for Windows based system which can be integrated with Microsoft Visual C++ 6.0. Initially the system is designed and implemented as research project to investigate different techniques and investigate which is best and reliable for handwritten isolated Urdu characters.

Preface

This report represents description of Analysis, Design, Development, Implementation and Testing of "Trainer and Classifier for Urdu OCR SDK". The entire work is summarized in chapters and some Appendixes.

Chapter-1	This chapter contains the material about the problem domain.
Chapter-2	This chapter describes problem to be solved, scope objective and feasibility of problem to be solved.
Chapter-3	This chapter concerns about requirement engineering, project plan resources to be used.
Chapter-4	This chapter contains items regarding analysis of system.
Chapter-5	This chapter has items regarding the design of system.
Chapter-6	This chapter includes documentation about implementation of system.
Chapter-7	This chapter has documentation about testing of the system.
Appendix-A	It contains the classes' description.
Appendix-B	This appendix contains Gantt Chart.
References	Reference of books used in this documentation.
Bibliography	Bibliography of books used in project development process.
Webliography	URL's of web sites, those were useful in development process.
Glossary	Glossary of terms used in this documentation.

Acknowledgement

First of all I would like to extend my sincere and humble gratitude to Almighty Allah whose blessings and guidance has been a real source of all my achievements in my life.

I am grateful to my supervisor **Sir Asif Qumer** for his sincere help and guidance during my studies and project work. He always remained a source of Learning for me.

I will be failing in my duties if I don't express my warmest thanks to my external supervisor **Chaudhary Waseem Fazal** (*Director Technical AKSA-SDS*) for their help, guidance and valuable suggestions in algorithm design, reading materials and other difficulties of the project.

It is my bounden duty to pay tributes to my worthy teachers and staff members of Department of Computer Science Quaid-i-Azam University Islamabad. Greatest thanks go to my favorite teachers Dr. Masud Ahmed Malik, Dr. Farhana Shaah and Mr. Asif Qumer for their kind help and guidance.

I am indebted to my parents and other family members for their prayers and moral & financial support during my studies.

How Can I forget to mention the names of my best friends Shahzad Saleem, Muhammad Afzaal, Azam bhai, Muhammad Iqbal and Jaffar Hussain for their sincere help, encouragement and enjoyable company during my student life. They are really great source of inspiration for me in my difficulties. QAU not only gave me valuable knowledge and skills but also very good friends like Naveed Ahmed, Kashif Ali, Hassan Jamil, Sohail Yousaf, Aamir Shahzad and many more. Their enjoyable company was really a great source of pleasure for me at university campus and I always feel the presence of their well wishes for me.

At the end I would like to thank all of my class fellows, seniors and juniors for their cooperation and help.

Muhammad Ali

Contents — Page Number

Overview of System

1.1 Introduction to Organization

AKSA Solutions Development Services (Pvt.) Ltd. Is a privately held company headquartered in Islamabad, Pakistan.

AKSA offers comprehensive Software/Hardware design, development, integration and testing Services. AKSA's vision is to become a leading Product Development Company. AKSA build on the product concept to help companies realize a commercial product with a focus on their valued proposition consisting of low cost, high quality and on time delivery.

1.1.1 Application Areas

Digital Preservation: Design, development of middle ware products related to Digital Preservation. Digital Image Archiving Lab (DIAL) at Islamabad is utilized to offer services related to Digital Preservation needs.

Image Processing: Image compression, Image Comparison, forms processing and Optical Character Recognition.

Web Technologies: Static and dynamic websites development, multi-tier applications with the database at the backend. ASP, JSP and Servlets as the middle-tier and front end in HTML.

Database: Payroll System, Inventory Control System, Customer Resource Management, Human Resource Management System, Office Information System, Management Information System, Accounting System, Time & Attendance System.

1.2 Existing Systems

Following is a list of existing systems, libraries and tools, which I studied before start our work. These systems are useful for "English" Optical Character Recognition and there is not any available for "Urdu". So there is need to develop one for "Urdu" and my work is a step toward that.

1.2.1 LEADS tool

It is an SDK for English OCR, but can also recognize some European languages as well. Purpose of its study was to get some domain knowledge and get ideas for our own SDK design.

1.2.2 Pegasus

It is an SDK for English OCR and it can also recognize some special characters as well. Purpose of its study was to get some domain knowledge and get ideas for our own SDK design.

1.2.3 Image Magic

This SDK is a library for writing programs to perform image manipulation and analysis and to develop advanced applications, including real-time image-processing applications.

1.2.4 I.R.I.S Toolkit

I.R.I.S. offers toolkits to read printed information (OCR), to read numeric and alphabetic hand printing (ICR), to read bar codes, forms and to read the highly specialized banking fonts.

1.3 Problem Definition

A SDK* has to be developed which will recognize Urdu characters from scanned images. The scanned image may contain isolated Urdu alphabets filled in boxes. Image may contain different form of noise, there is need to apply filters on image to remove noise for better performance. After recognition of the characters the specific Uni-code will be stored. After conversion the characters of stored Uni-code will display in different styles and fonts. The characters will be hand-written, some are as follows:

				د	م	ح	م
					ی	ل	ع
			ل	ا	ض	ف	ا

The scanned image may consist of multiple pages. The Image format could be TIFF*, JPG* or BMP.

1.4 Scope

There are three major scenarios to discuss the scope of Urdu OCR SDK. Those are discussed below.

1.4.1 Context

Urdu OCR SDK will be used in an environment in which the dominating and frequently used operating system is Microsoft Windows. Urdu OCR must be built fit into a larger system or business context. So the constraint imposed as a result of context on Urdu OCR is that, OCR must be compatible with Microsoft Windows and operating systems, which are compatible with Microsoft Windows. Due to this reason we are developing an Urdu OCR SDK which will be reused in other applications like form processing etc.

1.4.2 Information Objectives

In information objective the data objects produced as outputs of system and data objects, which are required as inputs of system are discussed.

1.4.2.1 Input

Input of the system will be scanned papers images of Urdu text. The Text characters will be isolated. The images may of bmp, tiff or jpeg format.

1.4.2.2 Output

Output of the system will consist of the Unicode based text of Urdu language of corresponding image. This text will be passed to application, which will use our OCR SDK.

1.4.3 Functions and Performance

Functions are those tasks, which will be performed by our system in order to transform the input data to output. General functions that can be performed with our OCR are following.

- Training of Neural nets through training algorithm
- Recognition of characters
- Storing of corresponding Uni-code values or text

1.5 Objectives

The objectives are those functionalities, which are required from the system or in other words those features in the system, which are required to fulfill the problem definition. In order to fulfill requirements of the SDK we have the following objectives to achieve.

- Creation of a new ANN for Training of desired characters.
- Training of ANN through training algorithm
- Recognition of characters through some classifier
- Post-Recognition of the image manually by user in order to verify that the results are correct.
- Extension of dictionary by Training of pattern, which are not recognized correctly. This will be done after post-recognition.
- Storing of corresponding Uni-code of the recognized characters

1.6 Recourses Identification

There is need of resources for doing any kind of job. Following resources are required to accomplish the development of this system.

1.6.1 Human Resources

The teams working on English OCR in AKSA SDS (Pvt.) Ltd. Has maintained some software development metrics. These metrics indicates size, complexity and feasibility of the project. According to these metrics, complexity and size of the project indicates that effort required for this project is 15 person months.

1.6.2 Hardware Resources

Hardware resources required for the development are as follow.

Resource Name	**Minimal**	**Recommended**	**Description**
Processor	233 MHz	850 MHz	To run software required for development
RAM*	128 MB	256 MB	To run software required for development
Hard Drive	5GB	10GB	To run software required for development and store data.
Printer	Dot Matrix	HP DeskJet 600	To get hard copy of documentation for end user.
Keyboard	Standard	Standard	To give input to the computer
Mouse	Standard	Standard	To give input to the computer
Scanner	Kodak 3500	Kodak 3500	To get images of documents for conversion.
Color Monitor	With resolution 800x600	With resolution 1024x768	To view out put.

1.6.3 Software Resources

Software resources required for the development are as follow.

Resource Name	Minimal	Recommended	Description
Operating System	MS Windows 98	MS Windows 2000 professional	To run the hardware.
Development Tool	Visual C++ 6.0	Visual C++ 6.0	To develop the system.
Documentation Tool	MS Office 97	MS Office XP or Star Office TM 5.2	To prepare the documentation.
Management Tool	MS Project 98	MS Project 2000	For analysis & Design
CASE Tool	Rational Rose 2.0	Rational Rose 5.0	For analysis & Design
Help Tool	MSDN library	MSDN library	For development help

1.7 Feasibility Study

I study the feasibility of the system in following four ways:

(1) Technical Feasibility

(2) Economical Feasibility

(3) Operational Feasibility

(4) Legal Feasibility

1.7.1 Technical Feasibility

The Urdu OCR SDK will be technically feasible because of following

- Using VC ++ 6.0, which is efficient and reliable tool, we will develop the system.
- By proposed solution it is to enhance it in future.

1.7.2 Economical Feasibility

Economic Feasibility is directly related to cost of software. Cost of software depends on resources that are used. The system will be economically feasible because of following

- As the system is developed for academic purposes in order to complete partial fulfillment of M.Sc degree so the evaluation versions of software can be used and there is no need to spend extra money to purchase software resources.

1.7.3 Operational Feasibility

The system will be operationally feasible because of following

- As we have studied the system very well and by using the technique of brainstorming so the end product will meet all the requirements of the user such as recognition of characters with satisfactory reliability.

1.7.4 Legally Feasibility

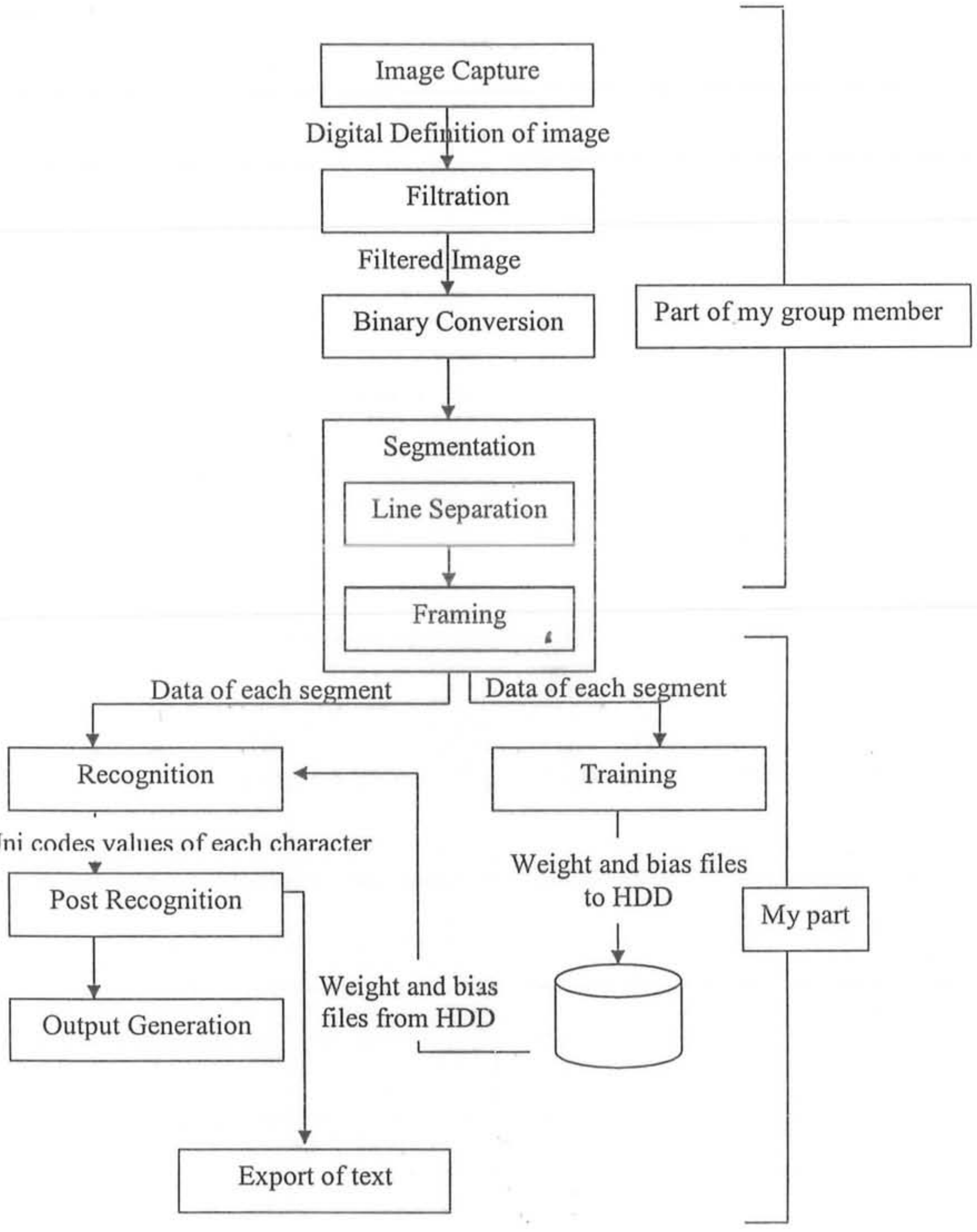
The system will be legally feasible because Legal feasibility is related with copyrights laws of softwares that are used in development. The development is involved with evaluation versions of softwares for academic purposes not for commercial purposes.

1.8 Paradigm Selection

The factors involved in the selection of software paradigm are project size, complexity, identification of requirements and time. Keeping in mind all these factors development has decided to follow **The Incremental Model**. It is very suitable when developing research project because team can analyze, design, code and test step by step, which is an excellent path to explore new domains.

1.9 Modular Interaction

As this project is going to make part of a bigger system so it is better to clearly distinguish here the parts of it. Modular interaction of the system at abstract level can be understood through following diagram



Requirement Engineering

The hardest single part of building a software system is deciding what to build. No other part of the work so cripples the resulting system if done wrong.

2.1 Requirement Engineering

It is the set of activities that leads to the production of requirements definitions and specifications of system.

System requirement include the following sub headings

- Requirements Definitions
    - Functional Requirements
    - Non Functional Requirements
- Requirements Specifications.

2.1.1 Requirement Definition

Requirements definitions of MIS are, what services the system is expected to provide and the constraints under which it must operate. Some of them are functional and some are non-functional which are defined separately in the subsequent sections.

2.1.1.1 Functional Requirements

- Creation of a new ANN for Training of desired characters.
- Training of ANN through training algorithm
- Recognition of characters through some classifier
- Post-Recognition of the image manually by user in order to verify that the results are correct.
- Storage and display of corresponding Uni-code of the recognized characters

2.1.1.2 Non-Functional Requirements

- The system should provide an attractive user interface for doing tasks like creating ANN, training ANN, classifying characters and for displaying recognized characters.

2.1.2 Requirement Specification

In it we have specified each and every functional requirement defined above in a precise way using *From Based Method**.

Function:	Creation of New ANN.
Description:	Creation of a new ANN is establishing a new Neural Net Work architecture with desired number of inputs, outputs and this can be trained for desired set of images.
Inputs:	Size of input layer, size of output layer and number of images going to be used for training it.
Source:	The information will be entered by user.
Outputs:	A desired network will be created ready for training.
Destination:	Inputs create a network and after creation it can be stored at HDD for further use.
Requires:	It requires number of input, outputs number of training images and specification of inputs and outputs at corresponding neurons.
Pre-conditions:	Nil
Post-conditions:	Default Network should inactive.

Function:	Training of New ANN.
Description:	Training of ANN train weight vectors for data set of specified input and output.
Inputs:	It requires the name and path of Network with which it would be stored at HDD after training.
Source:	The information will be entered by user.
Outputs:	A desired network will be trained for specified set of input classes.
Destination:	It can be stored at HDD for further use.
Requires:	It requires name and path of Network.
Pre-conditions:	Network should be created first.
Post-conditions:	Default Network should inactive and further recognition should be upon based of new trained Network

Function:	Classification of Characters
Description:	Classifier of ANN is responsible of recognizing an unknown character. It uses weights to calculate the output and recognize the characters.
Inputs:	It requires the input image to recognize.
Source:	The information will be entered by user.
Outputs:	A desired output will be a character index set at corresponding neuron of output layer of the ANN.
Destination:	Nil.
Requires:	It requires name and path of Network and input images to recognize.
Pre-conditions:	Network should be trained first or default network option is enabled
Post-conditions:	Nil

Function:	Post-Recognition
Description:	After recognition of ANN if there are some errors in decision of classifiers then that particular image can be trained through this option for making the Network more precise.
Inputs:	It requires the input image to train.
Source:	The information will be entered by user.
Outputs:	A desired network will correctly update its weights.
Destination:	Nil.
Requires:	It requires input image to train.
Pre-conditions:	Nil
Post-conditions:	Nil

Function:	Storage and Display of Characters
Description:	After Recognition of images the corresponding Unicode characters will be displayed and which can be stored for further reference.
Inputs:	It requires the input of index of characters recognized by ANN.
Source:	The information will be entered by provided by ANN.
Outputs:	A desired output will be Unicode based characters and will be displayed.
Destination:	Nil.
Requires:	It requires the characters Unicode and recognized characters indexes to assign that Unicode.
Pre-conditions:	Characters should be recognized first.
Post-conditions:	Nil

System Analysis

For Analysis and Design purposes the Unified Modeling Language (UML) is used. In UML, a system is represented using five different "views" that describe the system from distinctly different perspective. Each view is defined by a set of diagrams. The following views are present in UML.

User Model View

The use-case is the modeling approach of choice for the user model view.

Structural model view

Data and functionality are viewed from inside the system. That is, static structure (classes, objects and relationship) is modeled.

Behavioral Model

This model represents the dynamic or behavioral aspects of the system. It also depicts the interactions or collaborations between various structural elements described in the user model and structural models view.

3.1 User Model View

3.1.1 Identification of Actor

Actor of this systems are (1) Developers who is using this SDK in their applications (2) Indirect users who are using applications based on this SDK.

3.1.2 Use Case Identification

Use cases are represented graphically in a use case diagram to allow the analyst to visualize each use case in the context of other use cases in the system or subsystem to show its relationship with actors and other use cases. Use case names are text strings that contain letters, numbers and most punctuation marks except for colon, which is used to separate use case names from the name of packages, and it is good idea to keep them short. Use case names are normally made up of an active verb and a noun or noun phrase that concisely describe the behavior of the system that you are modeling. There is only one actor of the system, which is the user of OCR SDK.

3.1.3 Use Case Diagram

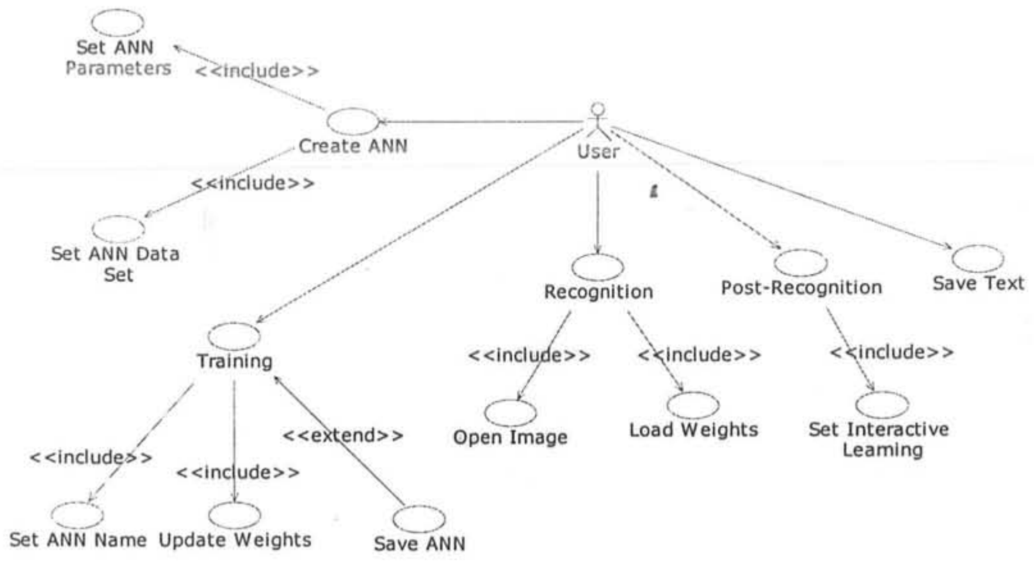
In use case diagram use case are drawn as an ellipse, the name of the use case usually written inside the ellipse, but can be placed beneath it. Do not mix these two styles in the same model.

3.1.4 Use Case

Following are the possible identified use cases of the system.

- Creation of an ANN
- Training
- Recognition
- Post-Recognition
- Storage of character Uni-code
- Generation of Output
- Export of Text

3.1.5 Use Diagram



Use Case Diagram (Figure 3.1)

3.1.6 Use Case Description (Use-Case Template)

Here is the description of the few use cases identified above.

3.1.6.1 Creation of ANN

Use case:	**Creation of ANN**
Actors	User of application
Trigger	User select the "create ANN option from menu"
Flow of events	
1	User will click the create ANN option from main menu of application.
2	A dialog box will appear for taking inputs of Number of input layer, hidden layer and output layer neuron and number of training inputs for the ANN
3	An ANN will be created and will be ready for setting input and output vector at the input and output layer of the ANN.
4	Default ANN will become inactive.

3.1.6.2 Training

Use case:	**Training of ANN**
Actors	User of application
Trigger	User select the "Train ANN" option from menu
Flow of events	
1	User will click the train ANN option from main menu of application.
2	A dialog box will appear for taking inputs of Name of ÁNN for saving it to HDD.
3	An ANN will be trained and will be ready for recognizing.

3.1.6.3 Recognition

Use case:	**Recognition of Images**
Actors	User of application
Trigger	User select the "Recognize" option from menu
Flow of events	
1	User will click the Recognize option from main menu of application.
2	The Unicode based text will appear in another window after recognition.

3.1.6.4 Post-Recognition

Use case:	**Post-Recognition**
Actors	User of application

Trigger	User select the "Recognize" option from menu after setting mode "interactive learning"
Flow of events	
1	User will click the Recognize option from main menu of application.
2	The Unicode based text will appear in another window after recognition.
3	Post-recognition of images will include training of those images which will not correctly recognized in first attempt and this process will include specification of data set again.

3.1.6.5 Storage of Text

Use case:	**Storage of Text**
Actors	User of application
Trigger	User select the "Save" option from menu after setting mode "interactive learning"
Flow of events	
1	User will click the Save option from main menu of application.
2	The Dialog box will open prompting for file name of the text.
3	File of text will be stored.

3.2 Problem Definition

A paperless office may have some advantages, but simply going paperless does not necessarily give you more power and versatility. A paper report and a scan of a report are equally useless when it comes to working with intelligent documents like word processing files. To get past this limitation and make the raster images truly useful, many businesses convert their scanned documents into intelligent files like text files (which can be edited in a word processor) or vector drawing files (which can be edited in a drawing program). This conversion is done using programs called optical recognition software that recognize shapes within the raster images and turns those shapes into actual words or vector-defined graphics.

OCR softwares examine raster images of letters and converts the images into actual ASCII or Unicode characters. Once converted into real text, the document can be edited within a word processor, spreadsheet, database, or any other program that can work with text-based documents.

3.3 Static Model

Object oriented methodology provides information hiding, abstraction and modularity, by viewing a software system as a set of interconnecting data objects. These data objects have their own private status represented by a set of attributes (data members) and a set of member functions to change their state. The data objects communicate with one another through messages and by calling each other's member functions directly.

3.3.1 Identification of Classes

The most important classes of this system are as follow:

- CBackProp
- CBackPropTrain
- CBackPropTest
- COutputList
- CUrduString
- CWeight
- CLayer
- CLayerList
- CNet
- CPerceptron
- CPerceptronTrain
- CPerceptronTest

3.3.1.1 Representation of Classes

A CRC model is used for the organized representation of the classes. The CRC cards are divided in three sections. Along the top of the card the name of the class is specified. In the body of the card responsibilities of class are listed on the left and the collaborators on the right.

Class Name: CbackProp	
Class Type: Property Class	
Class Characteristic: Aggregate, Concurrent	
Responsibilities	**Collaboration**

• It creates a multiplayer back-propagation Neural Network architecture. • Set the parameters of the Network. • Read Image Input. • Set the image as input of Neural Network at its input layer. • Reads and writes 'weights' dictionary files.	CBackPropTrain CBackPropTest Cweight

Class Name: CbackPropTrain	
Class Type: Property Class	
Class Characteristic: Aggregate, Concurrent	
Responsibilities	**Collaboration**
Set source to input layer of the ANN. Set target to the output layer of the ANN. Set the learning rate of ANN. Train the Network and updates the weight files of both layers.	CBackProp CBackPropTest Cweight

Class Name: CbackPropTest	
Class Type: Property Class	
Class Characteristic: Aggregate, Concurrent	
Responsibilities	**Collaboration**
Load the weights dictionary file. Recognize the input image pattern.	CBackProp CBackPropTrain Cweight

Class Name: CoutputList	
Class Type: Property Class	
Class Characteristic: Aggregate, Concurrent	
Responsibilities	**Collaboration**
Set the Uni-code values to the image patterns by using the information received during recognition.	CUrduString CBackPropTest CWeight

Set the output file for the text. Write the output file in a Unicode text format.	

Class Name: Cweight	
Class Type: Property Class	
Class Characteristic: Aggregate, Concurrent	
Responsibilities	**Collaboration**
It is responsible of encapsulating weights and bias of both layers of ANN. Set the architecture of weight files. Write the trained weights dictionary of the ANN. Reads the trained weights dictionary of ANN.	CUrduString CBackPropTest CBackProp

Class Name: Cperceptron	
Class Type: Property Class	
Class Characteristic: Aggregate, Concurrent	
Responsibilities	**Collaboration**
It creates a Perceptron Neural Network architecture. Set the parameters of the Network. Read Image Input. Set the image as input of Neural Network at its input layer. Reads and writes 'weights' dictionary files.	CPerceptronTrain CPerceptronTest CWeight CLayer

Class Name: CPerceptronTrain	
Class Type: Property Class	
Class Characteristic: Aggregate, Concurrent	
Responsibilities	**Collaboration**
Set source to input layer of the ANN. Set target to the output layer of the ANN. Set the Thresh hold of ANN.	CPerceptron CPerceptronTest CWeight

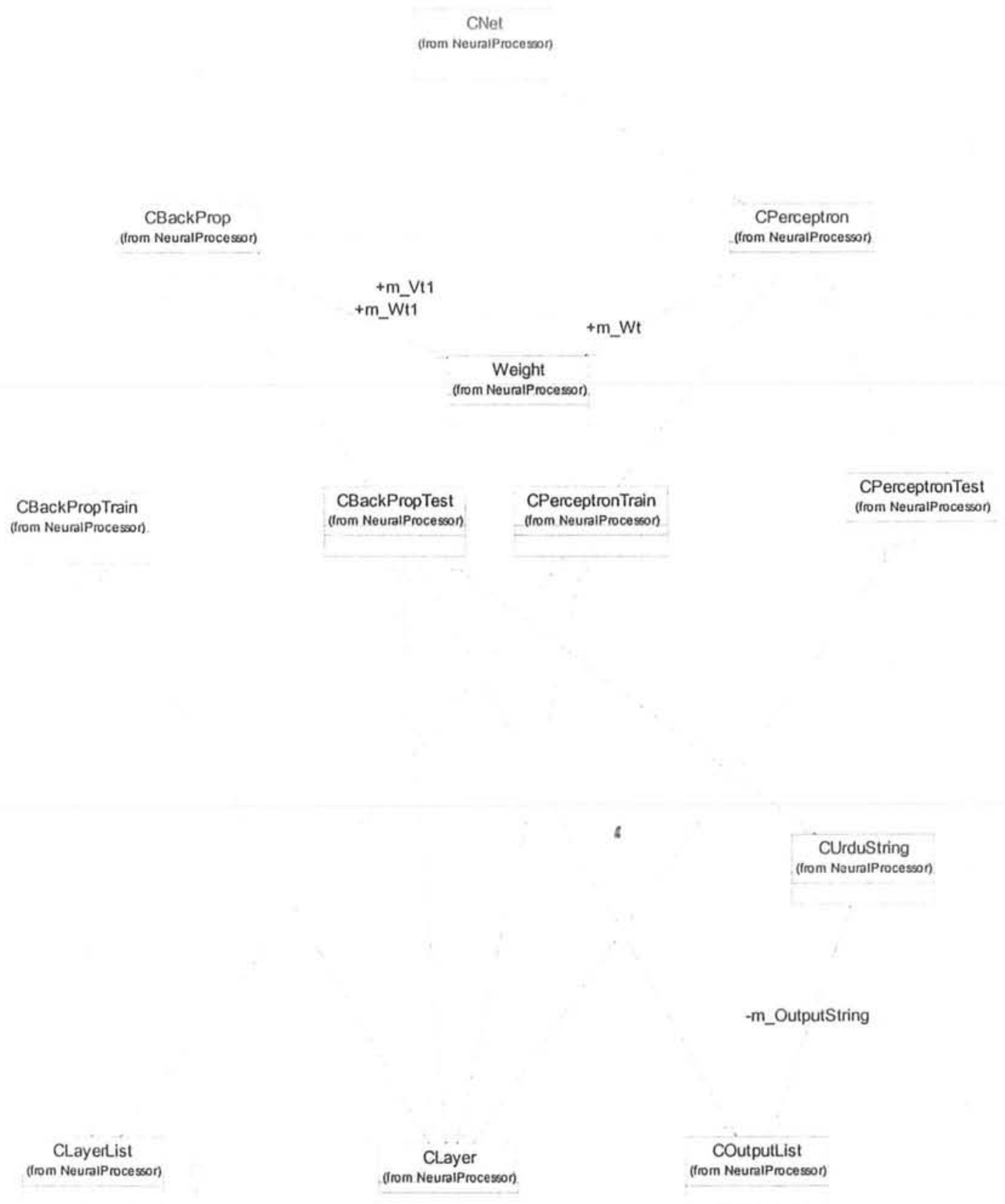
Train the Network and updates the weight files of both layers.	CLayer

Class Name: CperceptronTest	
Class Type: Property Class	
Class Characteristic: Aggregate, Concurrent	
Responsibilities	**Collaboration**
Load the weights dictionary file. Recognize the input image pattern.	CPerceptron CPerceptronTrain CWeight CLayer

Class Name: Clayer	
Class Type: Property Class	
Class Characteristic: Aggregate, Concurrent	
Responsibilities	**Collaboration**
Represents one layer of ANN. It is responsible of converting digitized image into source vector. Store the input and output vectors during processing.	CUrduString CBackPropTest CBackPropTrain CPerceptronTest CPerceptronTrain

Class Name: CurduString	
Class Type: Property Class	
Class Characteristic: Aggregate, Concurrent	
Responsibilities	**Collaboration**
It is responsible of encapsulating uni-code characters of recognized images. It assign the uni-code values to recognized output list of COutputList object.	COutputList

3.3.1.2 Class Diagram



Class Diagram (Figure 3.2)

3.3.1.3 Identification of Attributes and Methods

The following paragraphs briefly describe the specifications of the above-mentioned classes:

CNet

Class:	Cnet	
Parent Class:	None	
Operations:	Abstrsct ActivationFunction()	Integer (return type)
	Abstrsct GetDictionaryName ()	String (return type)
	Abstrsct ReadDictionary (string)	Bool (return type)
	Abstrsct WriteDictionary (string)	Bool (return type)
	Abstrsct SetDictionaryName (string)	Void (return type)

CBackPropTrain

Class:	CbackPropTrain	
Parent Class:	CbackProp	
Attributes:	m_LearningRate	float (attribute type)
	m_Source	List< Weight > (attribute type)
	m_Target	List < vector <float>> (attribute type)
Operations:	BackPropNewTrain ();	Void (return type)
	BackPropUpdateTrain ();	Void (return type)
	SetLearningRate (float);	void (return type)
	SetSource(string, int)	void (return type)
	SetSource (list <string> ,int)	void (return type)
	SetSource (list <string>, list <int>)	void (return type)

CbackPropTest

Class:	CbackPropTest	
Parent Class:	CbackProp	
Operations:	BackPropRecog (CvisByteImage)	Void (return type)
	CBackPropTest (list <CVisByteImage>)	void (return type)
	CBackPropTest int, int, int, int)	void (return type)

COutputList

Class:	CoutputList	
Parent Class:		
Attributes:	m_CharacterList	list <int> (attribute type)
	m_OutputFileName	CurduString (attribute type)
	m_OutputString	string (attribute type)
Operations:	GetCharList ();	List <int > (return type)
	GetOutputFileName ();	string (return type)
	SetCharList (int)	void (return type)
	SetOutputFileName(string)	void (return type)
	GetOutputString()	string (return type)
	SetOutputString()	void (return type)
	WriteOutputFile (string, CurduString)	

CLayer

Class:	Clayer	
Parent Class:		
Attributes:	m_Matrix	vector < vector <float> > (attribute type)
	m_Col	int < float > (attribute type)
	m_Row	int (attribute type)
Operations:	At(int, int);	float & (return type)
	empty ();	void (return type)
	FillRandom ()	void (return type)
	FillZero ()	void (return type)
	GetCols ()	int (return type)
	GetRows ()	int (return type)
	Operator [] (int)	float & (return type)
	Random()	float (return type)
	ReSize (int, int)	Void (return type)

CLayerList

Class:	ClayerList	
Parent Class:	None	
Operations:	Empty ();	Void (return type)
	Clear ();	void (return type)
	ReSize(int)	void (return type)
	At (type_size)	void (return type)
	Last()	CLayer* (return type)
	RemoveLast()	void (return type)
	Append()	void (return type)
	GetSize()	type_size (return type)
	First ()	CLayer* (return type)

CUrduString

Class:	CUrduString	
Parent Class:	None	
Operations:	Empty ();	Void (return type)
	Clear ();	void (return type)
	ReSize(int)	void (return type)
	At (type_size)	void (return type)
	Last()	CLayer* (return type)
	RemoveLast()	void (return type)
	Append()	void (return type)
	GetSize()	type_size (return type)
	First ()	CLayer* (return type)

CBackProp

Class:	CBbackProp	
Parent Class:	CNet	
Attributes:	m_Bvs	vector < float > (attribute type)
	m_Bvs	vector < float > (attribute type)
	m_DictionaryFileName	String (attribute type)
	m_Wt	CWeight (attribute type)
	m_Wt	CWeight (attribute type)
	m_Input	Integer (attribute type)
	m_Inner	Integer (attribute type)
	m_Output	Integer (attribute type)
	m_TrainOutput	Integer (attribute type)
Operations:	Sigmoid (float);	float (return type)
	D_Sigmoid (float);	float (return type)
	IsLayerAdjusted()	bool (return type)
	GetInputNeuron()	Integer (attribute type)
	GetInnerNeuron()	Integer (attribute type)
	GetOutputNeuron()	Integer (attribute type)
	ReadDictionary(char*FileName)	bool (return type)
	WriteDictionary(char*FileName)	bool (return type)
	GetDictionaryName()	string (return type)
	GetTrainOutput()	Integer (attribute type)
	SetParameters (int, int, int, int)	Void (return type)
	SetDictionaryName (string)	Void (return type)

CPerceptron

Class:	Cperceptron	
Parent Class:	CNet	
	m_Wt	CWeight (attribute type)
	m_Bws	vector < float > (attribute type)
	m_dThreshHold	float (attribute type)
	m_Input	Integer (attribute type)
	m_Output	Integer (attribute type)

	m_DictionaryName	string (attribute type)
Operations:	SetParameters (int, int, int, int)	Void (return type)
	ReadDictionary(char*FileName)	bool (return type)
	WriteDictionary(char*FileName)	bool (return type)
	SetDictionaryName (string)	Void (return type)

CPerceptronTrain

Class:	CPerceptronTrain	
Parent Class:	CPerceptron	
Attributes:	m_Source	CLayer (attribute type)
	m_Target	List <vector < float > >(attribute type)
Operations:	PTrain( );	void (return type)
	SetSource (list<CLayer>, int)	void (return type)
	SetSource (CLayer, int)	Void (return type)
	SetSource (list<CLayer>, list<int>)	Void (return type)

CPerceptronTest

Class:	CperceptronTest	
Parent Class:	CNet	
Operations:	PerceptronRecog(CLayer)	Integer (return type)
	PerceptronRecog(list <CLayer>)	List <int> (return type)

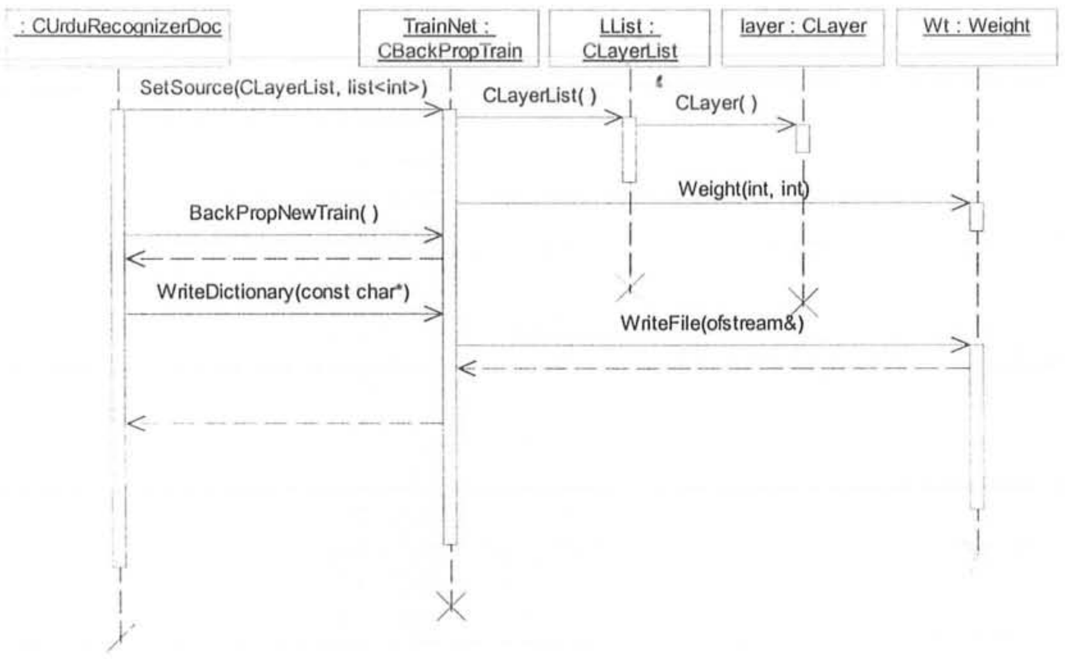
CWeight

Class:	CWeight	
Parent Class:		
Attributes:	m_Bias	int < float > (attribute type)
	m_Col	int < float > (attribute type)
	m_Row	int (attribute type)
	m_Matrix	vector < vector <float> > (attribute type)
Operations:	At(int, int);	float & (return type)
	empty ();	void (return type)
	FillRandom ()	void (return type)
	FillZero ()	void (return type)

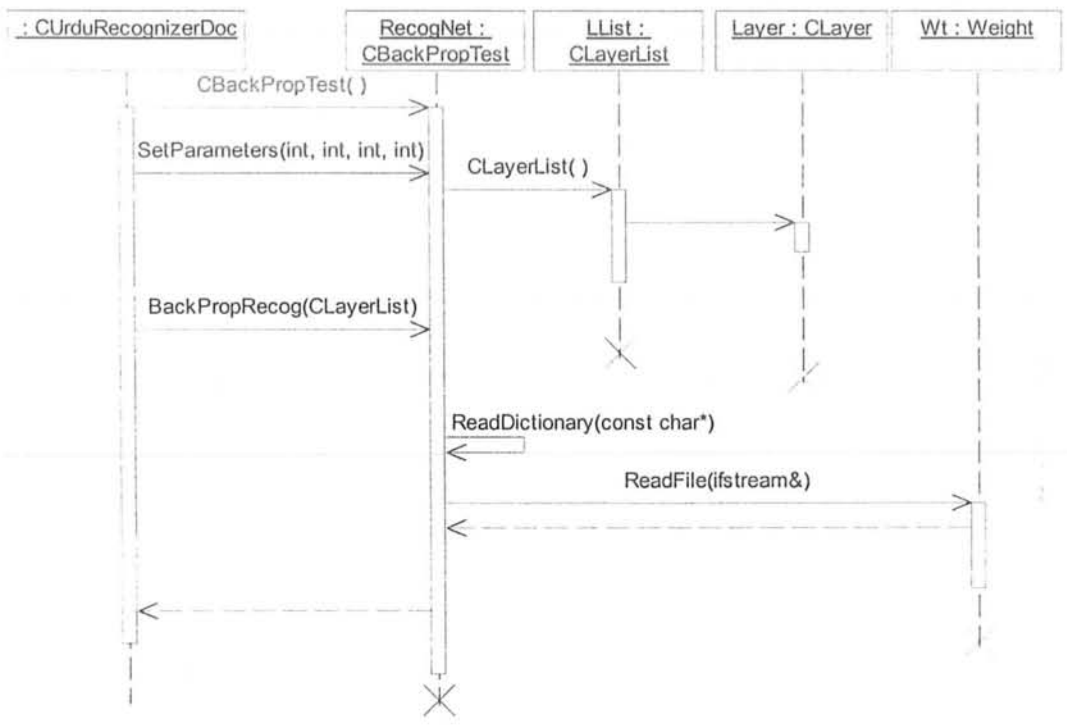
	GetBias ()	int (return type)
	GetCols ()	int (return type)
	GetRows ()	int (return type)
	Operator [] (int)	float & (return type)
	Random()	float (return type)
	ReadFile (string)	bool (return type)
	ReSize (int, int)	Void (return type)
	SetBias(int)	Void (return type)
	WriteFile(string)	bool (return type)
	CWeight()	
	CWeight(int, int)	
	CWeight (const Cweight &)	

3.4 Behavioral Model

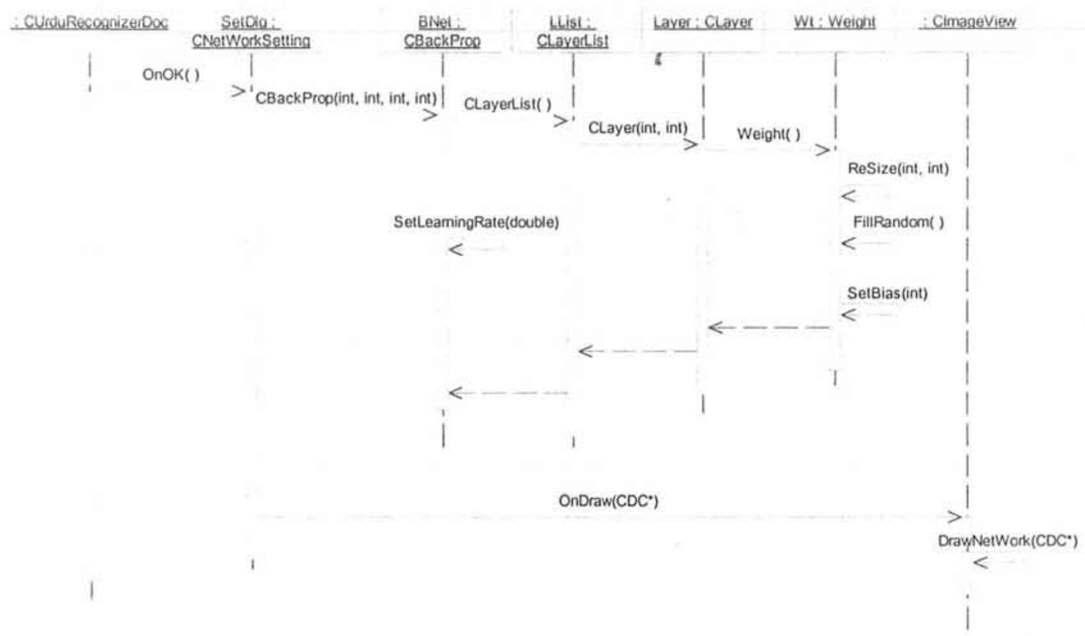
This model represents the dynamic or behavioral aspects of the system. It also depicts the interactions or collaborations between various structural elements described in the user model and structural models view.



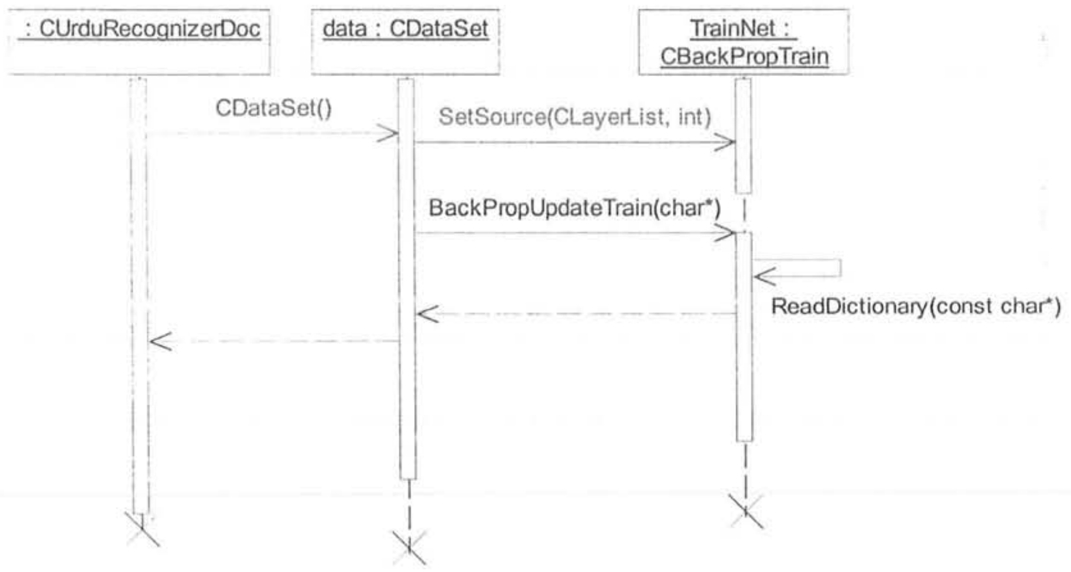
Train Network Sequence (Figure 3.3)



Recognize through Network Sequence (Figure 3.4)



Create Network Sequence (Figure 3.5)


: CUrduRecognizerDoc
data : CDataSet
TrainNet : CBackPropTrain
CDataSet()
SetSource(CLayerList, int)
BackPropUpdateTrain(char*)
ReadDictionary(const char*)

Post-Recognition Sequence (Figure 3.6)

System Design

4.1 GUI Classes Identification

- CMainFrame
- CUrduRecognizerDoc
- CUrduRecognizerApp
- CTextView
- CImageView
- CNetWorkSetting
- CNetWork
- CInteractive

4.2 Pseudo Code of Important Methods

The one of the important algorithm used is for training of ANN. The two training techniques are used in the system

1. Training through learning algorithm Perceptron
2. Training through Back-propagation

The testing of different characters after training through Perceptron showed that it is not suitable for the problem, so in next iteration the training technique is changed and a multilayer ANN was trained through back-propagation algorithm.

Here is the pseudo code of back-propagation training algorithm.

4.2.1 Important Considerations of Back propagation Algorithm

4.2.1.1 Activation Function

The basic function of an artificial neuron involves summing its weighted input signal and applying an output, or an activation function. For the input units it is identity function. Typically the same activation function is used for all neurons in any particular

layer of neural network, although this is not required. In most cases a non-linear activation function is used. In order to obtain advantages of the multilayer nets compared with the limited capabilities of the single layer nets, non-linear function is required.
Some Common Activation Functions are:

- Identity Function :

  $f(x) = x \qquad \text{for all } x.$

- Binary Step Function :

  $$f(x) = \begin{cases} 1 & \text{if } x >= 0 \\ 0 & \text{if } x < 0 \end{cases}$$

- Binary Sigmoid Function :

  $f(x) = 1/(1+\exp(-x));$

- Bipolar Sigmoid :

  $f(x)= 2/(1+\exp(-x))-1$

4.2.1.2 Initialization of Network

The choice of the initial weights will influence whether the network reaches the global (or only a local) minimum of error and, if so, how quickly it converges. The updation of the weights between two units depends on both the derivative of the upper unit's activation and the activation of the lower unit. For this reason, it is important to avoid choices of initial weights that would make it likely that either activation or derivatives of activation are zero. A common procedure is to initialize the weights (and biases) to random values between -0.5 and 0.5 (or between -1 and 1 or some other suitable interval).

4.2.1.3 Learning Rate

The smaller we make the learning rate parameter, the smaller will be the change to synaptic weights in the network from one iteration to next, and the smoother will be the trajectory in weight space. This improvement, however, is attained at the cost of a slower rate of learning. If, on the other hand, we make the learning rate parameter large so as to speed up the rate of learning, the resulting large changes in the synaptic weights assume such a form that the network may become unstable (i-e oscillatory).

4.2.1.4 Stopping Criteria

After each learning iteration, the network is tested for its generalization performance. The learning process is stopped when the generalization performance is adequate, or when it is apparent that the generalization performance has peaked.

4.2.2 Nomenclature

1. **TotalHLayers:** It specifies total number of hidden layers in the network.
2. **NeuronsCount:** It is a one dimensional array having locations equal to TotalHLayers+2 and holds number of neurons in each layer. Location 0 holds neurons in Input layer and location TotalHLayers+1 holds neurons in output layer.
3. **InputSet:** It is a 2-dimensional array holds input patterns for training. One row represents one input training pattern set.
4. **OutputSet:** It is a 2-dimensional array holds target patterns for training. One row represents one target training pattern set.
5. **WtsTable:** WtsTable is a 3-dimensional array; it holds synaptic weight values of neurons. First dimension points to layer number, second dimension points to neuron in layer, third dimension points to synapse for neuron in next layer.
6. **InputSignal:** It is a 2-dimensional array holds sum of signals of neurons in different layers.
7. **DeltaError:** It is a 2-dimensional array holds the error term which has to be backpropagated layer by layer.

4.2.3 Algorithm

```
Step 1. Initialize the weights (i-e WtsTable) to random values.
Step 2. For each Training set (current set = 0 to total number of training sets)
                Do step 3 to 7
Step 3. While Stopping condition is False  Do step 4 to 7
        Step 4. Feed-Forward
                For  Synapse=0 to NeuronsCount[1]-1 Do
```

```
Begin
        temp=0;
        For neuron=1 to NeuronsCount[0] Do
        temp= temp + InputSet[currentset][neuron-1] * WtsTable[0]
        [neuron] [synapse].
        InputSignal[0][synapse] = temp + WtsTable[0][0][synapse].
End.
For layer=1 to TotalHLayers Do
        For synapse=0 to NeuronsCount[layer+1] -1 Do
        Begin
                temp=0;
                For neuron=1 to NeuronsCount[layer] Do
                        temp+=WtsTable[layer][neuron][synapse] *
                        ActivationFunction (InputSignal[layer-1][neuron-
                        1]).
                InputSignal[0][synapse]=WtsTable[layer][0][synapse]+tem
                p
        End.
Step 5. Error BackPropagation
        For synapse=0 to NeuronsCount[TotalHLayers+1] -1 Do
        Begin
        output=ActivationFunction(InputSignal[TotalHLayers] [synapse]).
        DeltaError[TotalHLayers]
        [synapse]=TargetSet[currentset][synapse] *
        DerivativeAF(InputSignal[TotalHLayers] [synapse]).
        If  abs(Targetset[currentset][synapse]-output) < Tolerance
                ThenSet Stopping condition TRUE.
        End.
        For layer= TotalHLayers Downto 1 Do
          For synapse=1 to NeuronsCount[layer] Do
          Begin
```

```
                temp=0;
                For neuron=0 to NeuronsCount[layer+1]-1 Do
                    temp= temp + WtsTable[layer][synapse][neurons] +
                    DeltaError[layer][neuron].
                DeltaError[layer-1][synapse-1]= temp * DerivativeAF(
                InputSignal[layer-1][synapse-1] ).
            End.
Step 7. Update Weights
    For layer= TotalHL:ayers Downto 1 Do
    Begin
        For synapse=0 to NeuronsCount[layer+1]-1 Do
            WtsTable[layer][0][synapse] = WtsTable[layer][0][synapse]
                * LearningRate * DeltaError[layer][synapse].
        For neuron=1 to NeuronsCount[layer] Do
            For synapse=0 to NeuronsCount[layer+1]-1 Do
                WtsTable[layer][neurons][synapse]+=LearningRate*
                DeltaError[layer][synapse] *
                ActivationFunction(nputSignal [layer-1][neurons-1] )
        For synapse=0 to NeuronsCount[1]-1 Do
            WtsTable[0][0][synapse] = WtsTable[layer][0][synapse]
                                * LearningRate * DeltaError[0][synapse].
        For neuron=1 to NeuronsCount[0] Do
            For synapse=0 to NeuronsCount[1]-1 Do
                WtsTable[0][neurons][synapse] += LearningRate*
                DeltaError[0][synapse] * InputSet[currentset][neurons-1] ).
    End.
Step 8. Check Training of Net For Generalization of All cases.
    IF Required Generalization is achieved THEN
        Stop the Process
    ELSE
        Do Step 2 to 8.
```

System Implementation

5.1 Platform, Programming language and Tool Selection

Platform, programming language and related tools are selected according to nature and requirement of this project.

5.1.1 Platform Selection

Platform chosen for development of this system is Microsoft windows 2000, the objectives and reasons selected as development platform as it is organization's standard and later the system will be implemented on the same operating system. So, it is better to use it at development time for easy enhancement and later compatibility.

5.1.2 Programming Language Selection

Language selection is very difficult job and it depends upon the requirements of the system to be developed. Visual languages provide a visual interface and many other facilities, which help the developer making an application, which are more users friendly and attractive. Programming language chosen for development of this project is VC++. This language is ultimate choice for this type of operating system related project. The selection is based on number of reasons.

- Visual C++ provides simple way to build user interface.
- Many constructs and utility classes used in System are already available in Visual C++.
- It is requirement of the AKSA to follow the Microsoft standard because they are solution provider of Microsoft products; Visual C++ supports Microsoft standards so it was selected as implementation tool.
- Visual C++ is object oriented language. As object oriented approach is used during analysis and design phase of software development so it was necessary to select an object oriented language.

Its speed is reasonable as compared to other languages like JAVA and VB.NET.

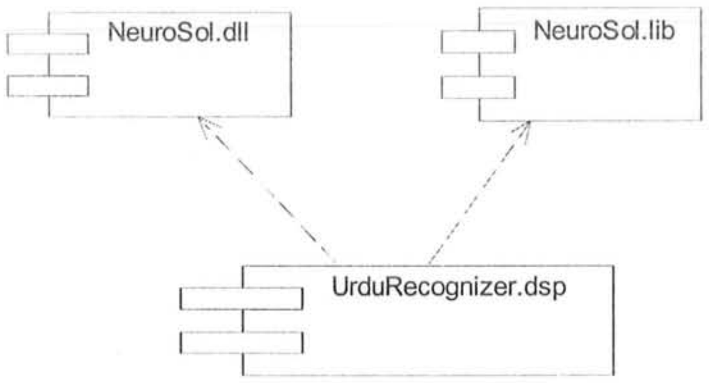
5.1.3 Code Documentation Standard

Code of project has been documented well with coding standard as told by external project in charge. These standards are described below

- Each Class name begin with capital letter C and first letter of class name is also capitalized e.g. CBackProp class.
- Fully object oriented design is followed in implementation of this system. Some Classes are pure virtual classes that are only used as base classes for others e.g. CNet class.
- A general Design structure is used for the system, this design generalization will enable future enhancements, and other types of network interfaces can be easily incorporated in current system without much difficulty.
- Each Class is well documented in form of class documentation. Documentation of these classes is provided already.

5.2 Component Diagram

A component represents a physical piece of program code, either as source code or DLL. Following is a component diagram which shows the interrelations between components. "NeuroSol.dll" and "NeuroSol.lib" wraps the functionality of classes CNet, CBackProp, CBackPropTrain, CBackPropTest, CPerceptron, CPeceptronTrain, CPerceptronTest, CWeight, CUrduString, COutputList, CLayer and CLayerList.



Component Diagram (Figure 5.1)

System Testing

6.1 Objectives of Testing

Testing of the application is done to achieve following objectives.

- Execution of the program is done with intent to find the errors in the program.
- Test Cases are designed so that these have high probability of finding an as-yet-undiscovered error.

6.2 Boundary Value Analysis

Boundary value analysis is a test case design technique that complements equivalence partitioning. Rather than selecting any element of an equivalence class, BVA leads to the selection of test cases at the "edge" of the class.

The reason to use this technique is that a greater number of errors tend to occur at the boundaries of the input domain rather than in the "center". The test cases are designed so that these test cases satisfy the requirements of BVA also.

6.3 Test Cases

Test cases are designed so that these maps with following functional requirements of the system.

R.1. Creation of a new ANN.

R.2. Training of ANN.

R.3. Classification or Recognition of Images

R.4. Post-Recognition

R.5. Storage and Display

Req. No.	R.1
Date of Test	8/12/2003
Program	UrduRecognizer.dsp
Description	Creation of a new ANN is tested by triggering the control input for it and objective of test is to verify that an ANN with desired parameters is being created.

Input	Selection of menu item "Create ANN" and the parameter of new ANN are filled in a Dialog box made for it.
Expected Output	Corresponding ANN is created and simulation Diagram has shown.
Actual Output	Corresponding ANN is created and simulation Diagram has shown.
Test Conductor	Muhammad Ali

Result of Test: Test showed that desired result is being shown.

Req. No.	R.2
Date of Test	15/12/2003
Program	UrduRecognizer.dsp
Description	Training of a new ANN is tested by triggering the control input for it and objective of test is to verify that an ANN is being trained correctly.
Input	Selection of menu item "Train ANN" and the parameter of new ANN were already set in creation of ANN and data set must also be applied at input layer and output layer.
Expected Output	Corresponding ANN is trained and simulation has shown, and a weight file has saved at HDD.
Actual Output	Corresponding ANN is trained and simulation has shown, and a weight file has saved at HDD.
Test Conductor	Muhammad Ali

Result of Test: Test showed that desired result is being shown.

Req. No.	R.3
Date of Test	18/12/2003
Program	UrduRecognizer.dsp
Description	Recognition of different images through classifier is done in order to check that its classifier is working right and training

	was correct.
Input	Selection of menu item "Recognize" and the images that have to recognize must already opened for recognition.
Expected Output	Corresponding indexes of alphabets must set at the output layer of the ANN.
Actual Output	Corresponding indexes of alphabets must set at the output layer of the ANN.
Test Conductor	Muhammad Ali

Result of Test: Test showed that desired result is being shown.

Req. No.	R.4
Date of Test	25/12/2003
Program	UrduRecognizer.dsp
Description	Post-Recognition done by the user of the application.
Input	Selection of menu item "Interactie Learning" and the images that have to recognize must already opened for recognition and then Recognize the image.
Expected Output	If there are errors then that images can be used for training in user mode and ANN weights must updated so that it recognize the error prone image.
Actual Output	If there are errors then that images can be used for training in user mode and ANN weights must updated so that it recognize the error prone image.
Test Conductor	Muhammad Ali

Result of Test: Test showed that desired result is being attained

Req. No.	R.5
Date of Test	1/1/2004
Program	UrduRecognizer.dsp
Description	The Uni-code based characters of corresponding images must

	display and can be stored at user will.
Input	Its input is same as for Recognition.
Expected Output	The Uni-code based characters of corresponding images must display and can be stored at user will.
Actual Output	The Uni-code based characters of corresponding images must display and can be stored at user will.
Test Conductor	Muhammad Ali

Result of Test: Test showed that desired result is being attained

Appendix-A

A.1 What is Optical Character Recognition?

OCR stands for Optical Character Recognition. In OCR processing, a scanned-in image or bitmap is analyzed for light and dark areas in order to identify each alphabetic letter or numeric digit. When a character is recognized, it is converted into an ASCII or Unicode text. Users can then save the conversion in a usable word processing file. When a text document is scanned, the computer can only see this text as graphical figures on a page. That text in form of image is not usable, searchable, or editable. That is, a scan file is nothing but a bunch of unrelated dots (called a raster image), not a collection of letters, words, or geometrically defined shapes. For example, the dot in a raster image that marks the peak of a letter ع is not connected to the dot that marks the bottom, nor does the scan have any way of knowing that the dots collectively make and ع. Because of this, you cannot work with a raster image as if it were an intelligent document. You cannot, for instance, spell-check a raster image, edit or reformat its text, search for words, or copy and paste a paragraph of text into a word processing document.

Automatic handwriting recognition has several applications. In most of them the system must recognize a phrase or a set of isolated words (form processing). However, the recognition is usually performed at the word level with the help of a lexicon, and then a post-processing phase is carried out to validate the combination of words.

Among all handwriting recognition systems some classifications are usually made. The technique to perform the recognition can be chosen from several. Recently Artificial Neural Networks (ANN) has become one of the popular techniques being used. Systems that recognize handwriting are referred to as off-line or on-line systems depending on whether ordinary handwriting on paper is scanned and digitized or a special stylus and a pressure sensitive tablet are used.

A.1.1 Off-line and On-line Handwriting

The fundamental difference is in the nature of the handwriting sample used in each system. In off-line systems, the samples are static. The system analyzes a digital image using either the raw pixel data, or some sort of representation of the pixel data.

Only data that can be obtained directly from the image is used, there is no additional information given to it.

This differs from an online system, where the samples are analyzed in a dynamic environment. This allows on-line systems to collect detailed real time information. This information may include the pressure and speed that a sample is written with, and the specific order in which the different alphanumeric characters are placed. In a signature verification system, this extra information can aid in writer identification, because individuals generally follow the same procedure every time they sign their name. This ability to collect peripheral information not only makes on-line systems easier to implement, but it also gives them higher success rates than their off-line counterparts.

Though off-line and on-line handwriting recognition systems share many similarities, to allow for full investigation of the usefulness of using ANNs in the field of handwriting recognition, but we will discuss only off-line systems and techniques.

A.1.2 Modules of an OCR

A typical handwriting recognition system consists of following modules in general:

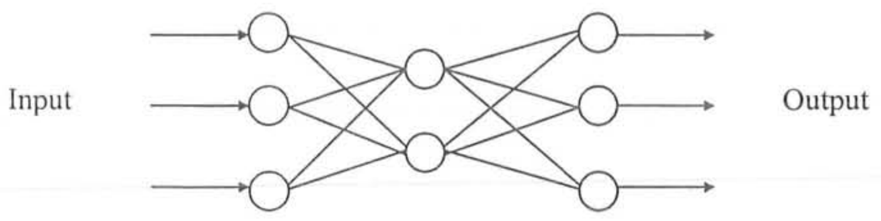
- Preprocessing
- Segmentation
- Classification
- Post-processing

"Classifier" is based on concepts of Artificial Neural Networks, so here it is important to give a brief overview of it

A.2 Introduction to Artificial Neural Network

The vast processing power inherent in biological neural structures has inspired the study of the structure itself for hints on organizing human made computing structures. Artificial Neural Networks cover the way to organize synthetics neurons to solve the same kind of difficult, complex problems in similar manners as we think the human brain may.

The brain evolved in the face of many constraints other then those of simply processing signals. We therefore look at neural networks as non-linear dynamic systems which are capable of adjusting themselves according to their environment and experience and which can satisfy certain constraints, which make them useful computational tools.



A Typical Figure of Artificial Neural Network (Figure A.1)

A.2.1 Artificial Neural Networks

A neural network is a computational structure inspired by the study of biological neural processing. There are many different types of neural networks; from relatively simple to very complex, just as there are many theories on how biological neural processing works.

A layered feed-forward neural network has layers, or subgroups of processing elements. A layer of processing elements makes independent computations on data that it receives and passes the results to another layer. The next layer may in turn make its independent computations and pass on the results to yet another layer. Finally, a subgroup of one or more processing elements determines the output from the network. Each processing element makes its computation based upon a weighted sum of its inputs. The first layer is the input layer and the last the output layer. The layers that are placed between the first and the last layers are the hidden layers. The processing elements are seen as units that are similar to the neurons in a human brain, and hence, they are referred to as cells, neuromimes, or artificial neurons. A threshold function is sometimes used to qualify the output of a neuron in the output layer. Even though our subject matter deals with artificial neurons, we will simply refer to them as neurons. Synapses between neurons are referred to as connections, which are represented by edges of a directed graph in which the nodes are the artificial neurons.

Artificial neural networks have been developed as a generalization of mathematical models of human cognition on the assumptions that:

1. Information processing occurs at many simple elements i-e artificial neurons.
2. Signals are passed between artificial neurons over connection links.
3. Each connection link has an associated weight, which in a typical neural network is multiplied to the signal transmitted.
4. Each artificial neuron applies an activation function (usually non-linear) to its input (sum of weighted input signals) to determine its output signal.

A.2.2 Model of Artificial Neurons

An artificial neuron is an information-processing unit that is fundamental to the operation of an artificial neural network. We can identify the following basic elements of an artificial neuron model:

A.2.2.1 A set of synapses or connecting links

Each of these links is characterized by a weight or strength of its own. Specifically a signal Xj at the input of link j connected to neuron k is multiplied by link weight W_{kj}. It is important to make a note of the manner in which the subscript of the link weight W_{kj} is written. The first subscript refers to the neuron in input layer and the second subscript refers to output end of link to which the weight refers. The reverse of this notation is also used in the literature.

A.2.2.2 An Adder for Summing the Input Signals

All input signals are added together to constitute a linear combination.

A.2.2.3 Activation Function to Limit the Amplitude of output of a neuron

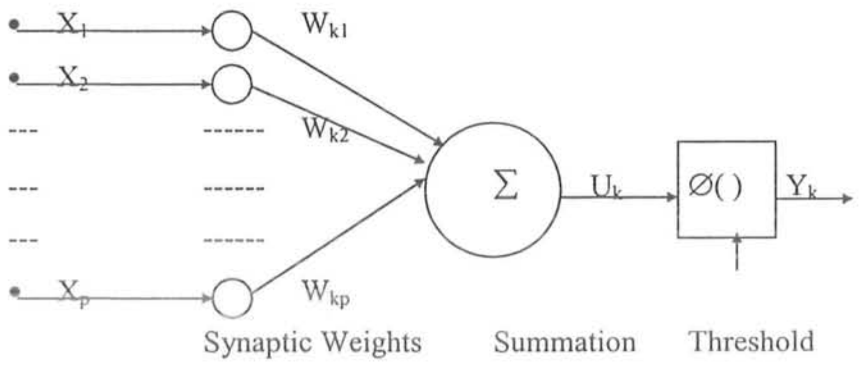
The activation function is also referred to in the artificial neural network as squashing function in that it squashes the permissible amplitude range of output signal to some finite value. Typically the normalized amplitude range of output of a neuron is written as closed unit interval [0, 1] or alternatively [-1, 1].

A.2.3 Threshold

The output of a neuron is also affected by an element called threshold. Threshold is some times set for the sake of scaling down the activation and mapping it into a meaningful output for the problem. The most often-used threshold function is the sigmoid function like tanh( ).

A.2.4 Bias

If for some purpose net input of the activation function has to be increased, a bias term is employed rather than threshold. Bias is negative of the threshold.



Non-Linear Model of Neuron (Figure A.2)

In mathematical terms we may describe a neuron k by writing the following pair of equation:

$$U_k = \sum_{j=1}^{P} WkjXj \qquad (1.1)$$

and

$$Y_k = \varnothing(U_k - \theta_k) \qquad (1.2)$$

Where X_1, X_2, ..., Xp are input signals,

W_{k1}, W_{k2}, ..., W_{kj} are link weights,

Uk is linear combiner output, θ_k is threshold, $\varnothing$ is activation function and Y_k is output signal of k^{th} neuron.

The use of threshold θ_k has the effect of applying an affine transformation to output U_k of the linear combiner in the model so

$$Y_k = (U_k - \theta_k) \qquad (1.3)$$

The threshold θk is an external parameter of artificial neuron k. Equation 1.1 and 1.2 can be formulated as

$$V_k = \sum_{j=1}^{P} Wkj\, Xj \qquad (1.4)$$

and

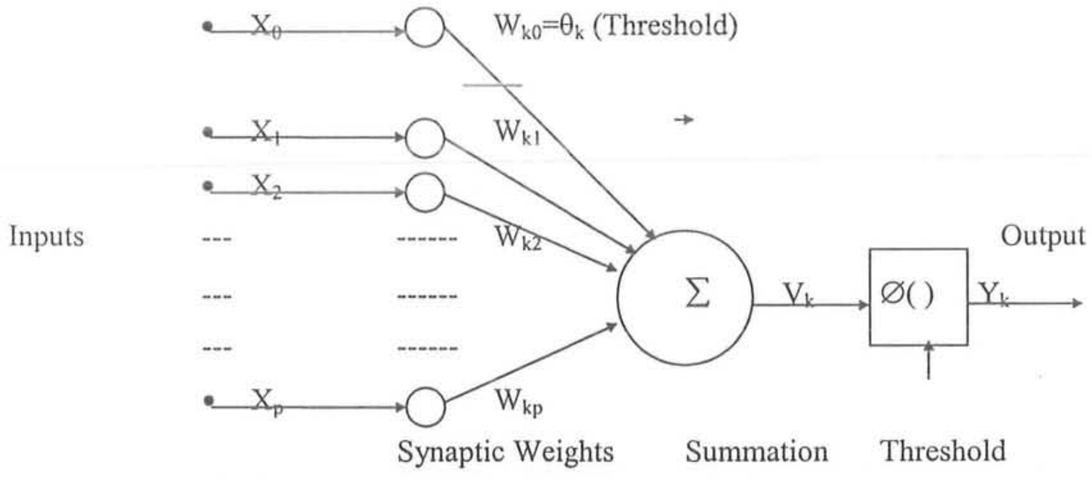
$$Y_k = \varnothing(V_k) \qquad (1.5)$$

In equation 1.4 a new synapse is added whose input is $X_0 = -1$ and whose weight is $W_{k0}=\theta_k$ and figure 1.3 illustrates it.

Figure 1.3 illustrates the effect of threshold by adding two factors in previous model.

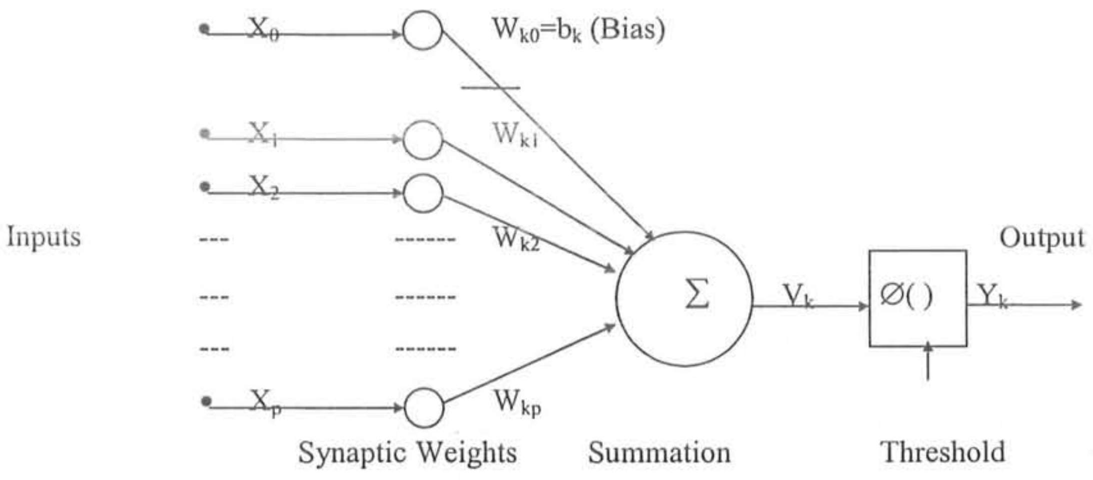
Adding new input signal fixed at $X_0 = -1$.

Adding a new synaptic weight equal to threshold θ_k.



Non-Linear Model Of Neuron With Threshold As Fixed Input (Figure A.3)

Alternatively we may model the neuron as in figure 1.4 where the combination of fixed input $X_0=1$ and weight $W_{k0} = b_k$ (bias b_k).



Non-Linear Model Of Neuron With Bias As Fixed Input (FigureA.4)

Although the models of figure 1.3 and figure 1.4 are different in appearance but mathematically they are equivalent.

A.2.5 Weights

The weights used on the connections between different layers have much significance in the working of the artificial neural networks and the characterization of a network. The following actions are possible in an artificial neural network:

- Start with one set of weights and run the network (no training).
- Start with one set of weights, run the network and modify some or all the weights, and run the network again with new set of weights. Repeat the process until some predetermined goal is met (training).

A.2.6 Learning and Training

Learning and training are important issues in applying artificial neural networks, as output of artificial neural networks heavily depends upon connection weights. Since output(s) may not be what is expected, the weights may need to be altered. Some rules then need to determine how to alter the weights. There should be a criterion to specify when the process of successive modifications ceases. This process of changing the weights, or rather, updating the weights is called training. Training is an external process and learning is a desired process that takes place internal to the network.

A.2.6.1Objective of Learning

An artificial neural network maps a set of inputs to a set of outputs. This non-linear mapping can be performed as a multidimensional mapping surface, the objective of learning is to mould the mapping surface according to a desired response, either with or with out an explicit training process. There are two broad categories of learning:

Supervised Learning

In supervised learning, external prototypes are used as target outputs for specific inputs and the network is given a learning algorithm to follow and calculate new connection weights that bring output close to the target output.

Unsupervised Learning

Unsupervised learning is a sort of learning that takes place without a teacher, so in this type of learning, a learning algorithm is given but target outputs are not given. In such a case similar input stimuli cause similar responses. Unsupervised learning is also called self-organized learning.

When artificial neural networks are developed and an appropriate learning algorithm is proposed, it would be based on the theory supporting the model. The learning equations are initially formulated in terms of differential equations, and using initial conditions that are available, the algorithm can be simplified to consist of an algebraic equation for the changes in weights. A large number of learning algorithms have been developed so for.

A.2.6.2 Learning, Stability and Convergence

Stability and convergence are of much interest in the learning of artificial neural networks. Suppose learning process is going to halt at some appropriate point which is a question of convergence. Whether learning is stable, or will the network have to learn all over again. Suppose we have a criterion such as energy to be minimized or cost to be decreased and we know the optimum level for this criterion. If the network achieves optimum value in a finite number of steps, we have convergence for the operation of network.

It is possible that convergence is slow, so much so that it may seem to take forever to achieve the convergence state. In that case we have to specify a tolerance value and require that the criterion be achieved with in that tolerance, avoiding a lot of computing time. We may also introduce a momentum parameter to further change the weight and thereby speed up the convergence.

Instead of convergence, the operation may result in oscillations. The weight structure may help changing back and forth, learning will never cease. So learning algorithms need to be analyzed in terms of convergence and stability of network learning as an essential algorithm property.

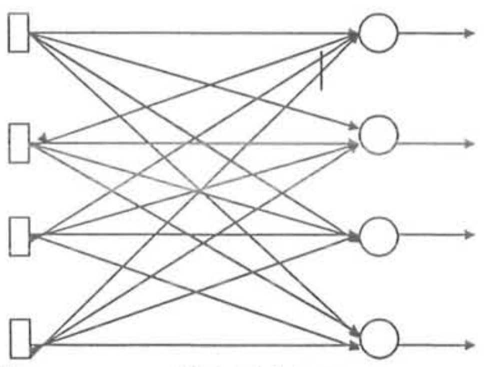
Learning time is an important consideration. This time could be long but should not be prohibitive. It is important to understand how network behaves with new inputs; some networks may need to be trained all over again, but some distortion (threshold) input is desirable. Restrictions on the format of input should also be known.

A.2.7 Architecture of Artificial Neural Network

The manner in which neurons of an artificial neural network are structured is called artificial neural networks architecture. It affects the training of artificial neural networks. In general we can identify four different classes of artificial neural networks.

A.2.7.1 Single Layer Feed Forward Network

A layered network is a network of neurons organized in the form of layers. In the simplest form of a layered network, we just have an input layer of source nodes that projects onto an output layer of neurons, but not vice versa. In other words this network is strictly feed forward type. It is illustrated in figure 1.5 for the case of four nodes in both the input and output layers. Such a network is called single layer network, with the designation "Single Layer" referring to the output layer of computation nodes, because no computation is performed there.

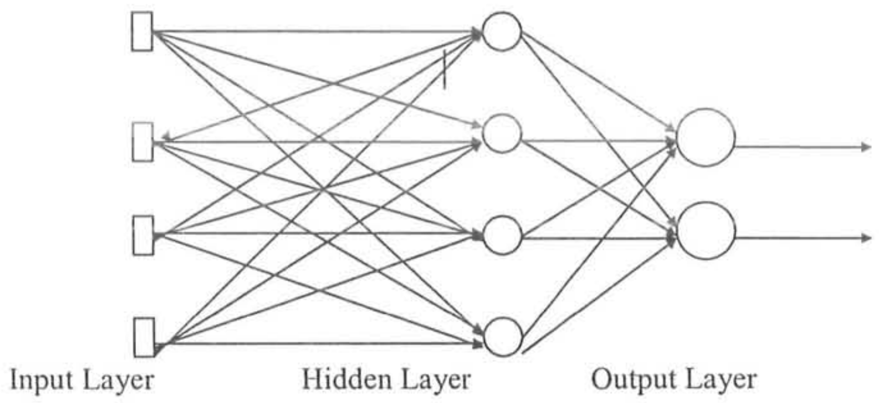


Feed Forward Network With Single Layer of Neurons (Figure A.5)

A.2.7.2 Multilayer Feed Forward Network

The second class of feed forward artificial neural networks distinguishes itself by the presence of one or more hidden layers, whose neurons are correspondingly called hidden neurons or hidden units. The function of the hidden neurons is to intervene between the external input and the network output. By adding one or more hidden layers the network is enabled to extract high-order statistics, for (in a rather loose sense) network acquires a global perspective despite its local connectivity by virtue of extra set of synaptic connections and extra dimension of neural interactions. The ability of hidden neurons to extract high-order statistics is particularly valuable when the size of input layer is large.

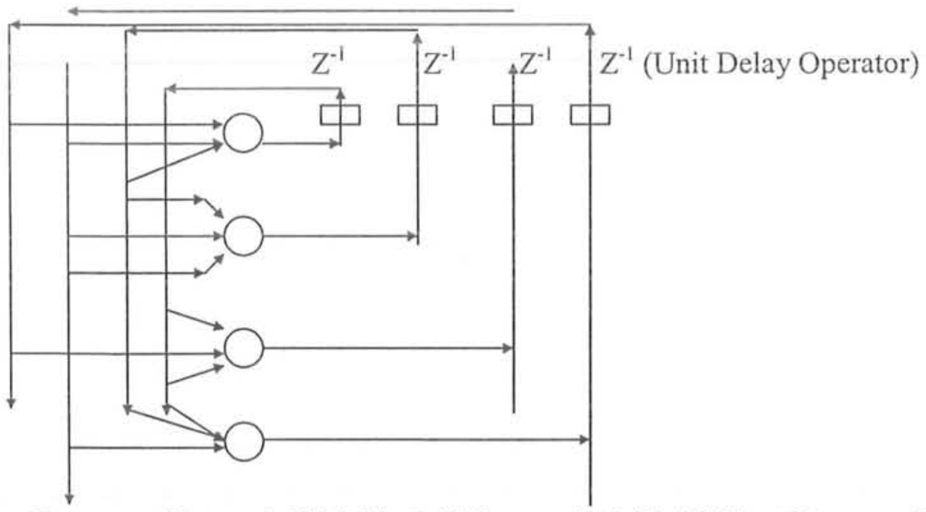
The source nodes in the input layer of the network supply respective elements of the activation pattern (input vector), which constitutes the input signals applied to the neurons in the second layer. The output signals of the second layer are used as inputs to the third layer, and so on for the rest of network. The set of output signals of the neurons in the final layer of the network constitute the overall response of the network to the activation pattern supplied by the source nodes in the first layer. The architectural graph of figure 1.6 illustrates the layout of multilayer feed forward neural networks for the case of single hidden layer.



Feed Forward Network With One Hidden Layer of Neurons (Figure A.6)

A.2.7.3 Recurrent Network

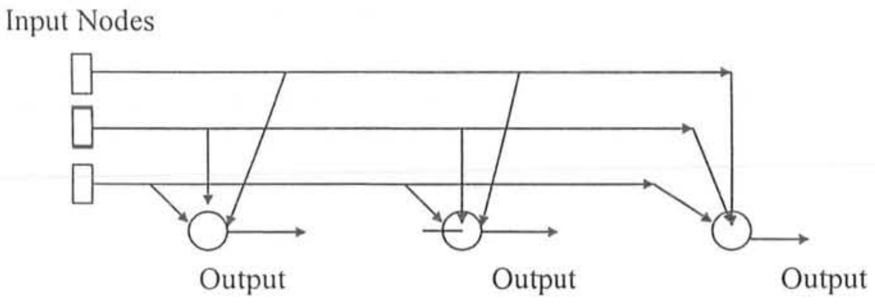
A recurrent artificial neural network distinguishes itself from feed forward artificial neural networks in that it has at least one feedback loop. For example, a recurrent network may consists of a single layer of neurons as illustrated in the graph figure of 1.7. In the structure depicted in this figure there are no self feed loops in the network; self feed loop refers to a situation where the output of a neuron is fed back to its own input. The recurrent network illustrated in figure 1.7 also has no hidden neuron.



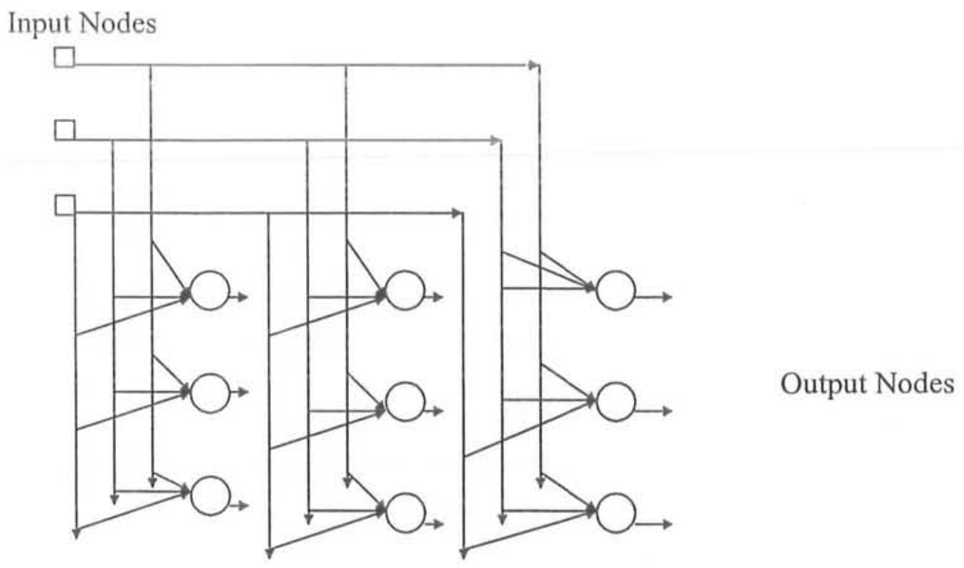
Recurrent Network With No Self Loops And No Hidden Neurons (Figure A.7)

A.2.7.4 Lattice Structures Artificial Neural Networks

A lattice consists of a one-dimensional, two-dimensional, or higher-dimensional array of neurons with a corresponding set of source nodes that supply the input signals to the array; the dimension of the array refers to the number of the dimensions of the space in which the graph lies. The architectural graph of the figure 1.8a depicts a one dimensional lattice of three neurons fed from a layer of three source nodes, whereas the architectural graph of figure 1.8b depicts a two dimensional lattice of 3 by 3 neurons fed from a layer of three source nodes. Note that in both cases each source node is connected to every neuron in the lattice. A lattice network is really a feed forward network with the output neurons arranged in rows and columns.



One Dimensional Lattice of 3 Neurons (Figure A.8 a)



Three Dimensional Lattices of 3-by-3 Neurons (Figure A.8 b)

A.2.8 Properties of Artificial Neural Networks

- They require no specific programming. In principle, when artificial neural networks are realized in hardware, they will simply be self-modifying machines with an input and an output, which will learn from multiple presentations of pattern from their environment.

- They can recognize easily and rapidly.
- They are also robust with regard to corrupt or incomplete data. Small changes do not affect the recognition of correct result.
- They are fault tolerant and resist damage. The death of a few thousand brain cells per day does not noticeably impair our memories.
- They respond quickly and correctly to previously unseen patterns. They are able to generalize to situations, which they have not previously experienced.

A.3 ANN and Character Recognition

Though some basic training techniques have already been discussed, its beneficial to formally introduce the processes used to train an artificial neural network.

There are two distinct stages when using ANNs to process any kind of data,

- A training stage and
- A classification stage.

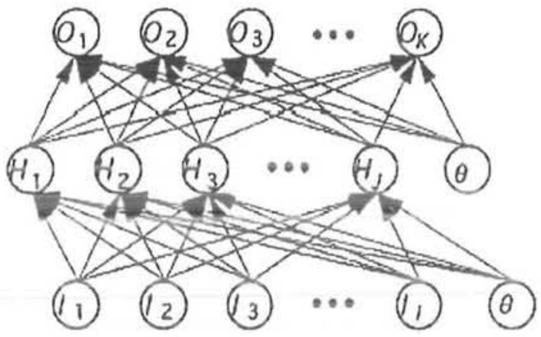
These two stages can be framed in terms of *handwriting recognition*.

In the ***classification stage***, samples are passed as inputs into the ANN, resulting in an output representing what the ANN believes to be the most correct output. However, to be successful, classification must be preceded by a training stage in which the ANN is given a set of inputs and the corresponding set of correct outputs. In this way, the ANN is able to adapt by adjusting its weights. In other words, the ANN "learns" to associate the correct output with each corresponding input. It's important to keep in mind that an acceptable training set will not only be representative of the data, it will also contain a large amount of both positive and negative samples. Though positive samples are often easier to obtain than negative samples, an insufficient amount of either kind will result in an ANN that performs poorly, either by accepting too many incorrect inputs, or by rejecting too many correct inputs. In the following section, a specific approach to handwriting recognition using neural networks will be discussed.

A.3.1 Back propagation

In the context of handwriting recognition, combining artificial neurons into a structure containing three layers can create an ANN.

- The first layer consists of neurons that are responsible for inputting a handwriting sample into the ANN.
- The second layer is a hidden layer. This layer allows an ANN to perform the error reduction necessary to successfully achieve the desired output.
- The final layer is the output layer. Typically, the number of neurons in this layer is determined by the size of the set of desired outputs, with each possible output being presented by a separate neuron. The structure of this particular ANN is illustrated in Figure 2.



The typical structure of a back propagation network(Figure A.9)

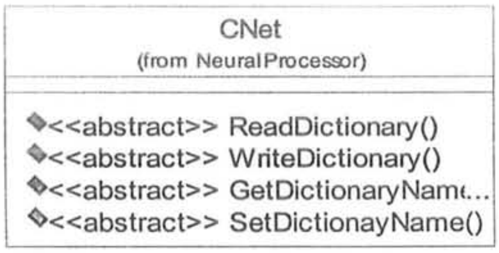
The structure of the ANN shown in Figure above is that of a back propagation network (BPN). This type of ANN is often used due to its general-purpose nature. In addition to their layered structure, BPNs are also characterized as being feed-forward, which means that all the links in the network are unidirectional, and the network is acyclic. Examination of Figure 2 also reveals that the neurons of the input layer are fully connected to the hidden layer, and the inputs of the hidden layer are fully connected to the output layer. This means that the error rate of one neuron affects the entire network. ***The process of determining how the error rates of each neuron impact the output is called back propagation.*** This relatively simple procedure is crucial to the error minimization process, a process that will eventually lead to identification of the desired output.

- In back propagation, a handwriting sample is propagated through the BPN, producing an output. This output is compared to the desired output, giving an error rate for the output layer. Because the error rate of a neuron is a function of the error rates of all the units that use its output, the error rates of the layer directly below the output layer can now be found.
- Subsequently, this allows the error rates of the layer two levels below the output layer to be found. These error rate calculations will continue to propagate through the BPN in a backward fashion, until the error rates for all the neurons have been found.
- Each neuron will then make slight weight adjustments in order to minimize its error signal. This pattern is repeated until the error rate of the output layer reaches a minimum value. This process is then repeated for the next input value, until all of the input values have been processed.

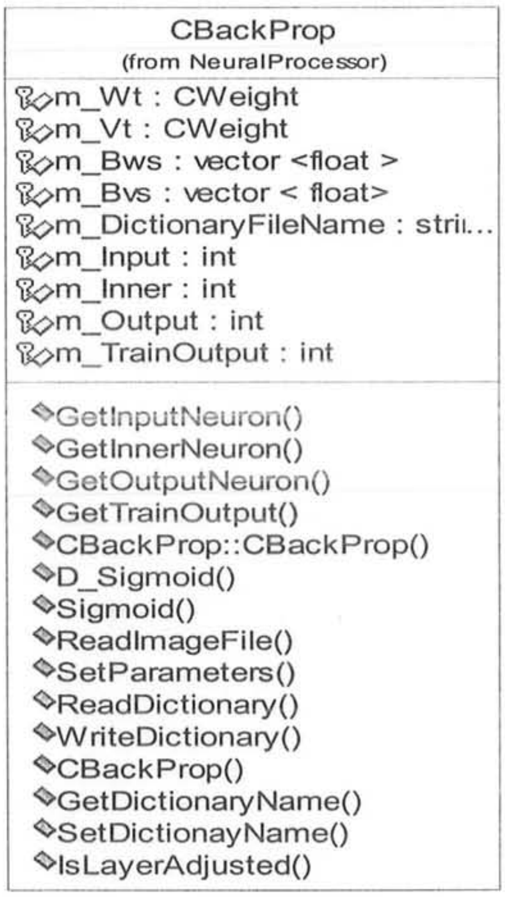
Appendix-B

1 Class Description

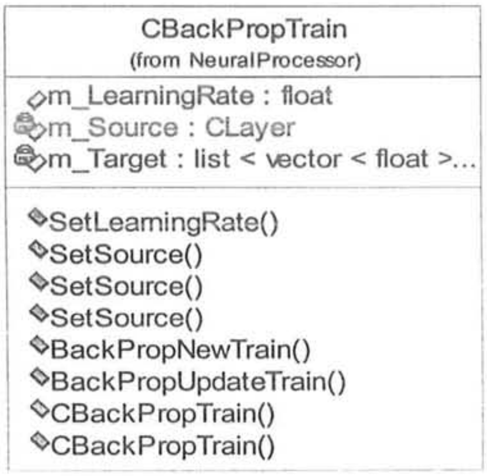
Class CNet


CNet
(from NeuralProcessor)
<<abstract>> ReadDictionary()
<<abstract>> WriteDictionary()
<<abstract>> GetDictionaryNam...
<<abstract>> SetDictionayName()

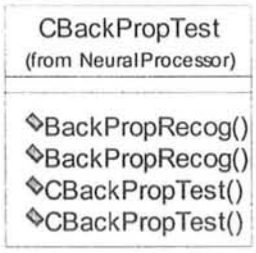
Class CBackProp



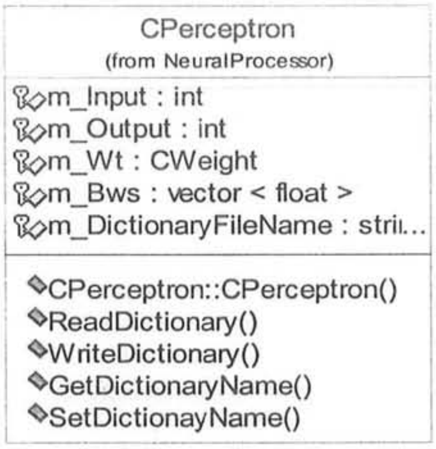
Class CBackPropTrain



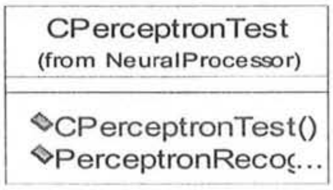
Class CBackPropTest



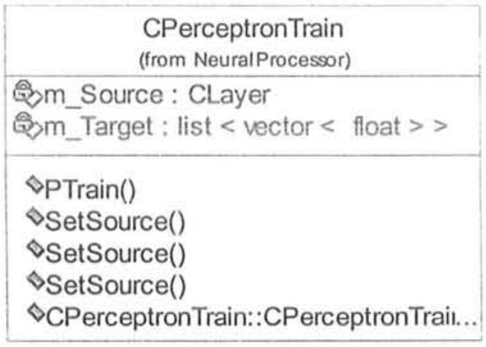
Class CPerceptron



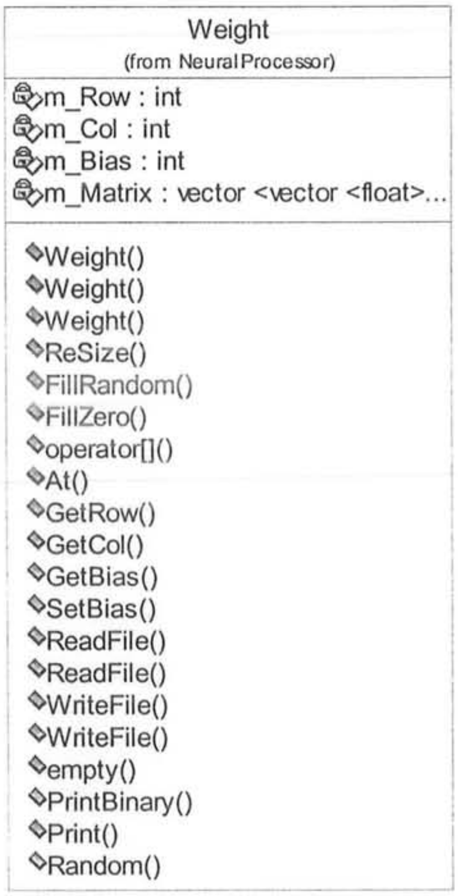
Class CPerceptronTest



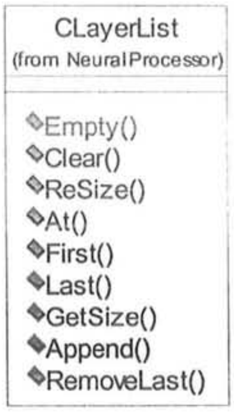
Class CPerceptronTrain



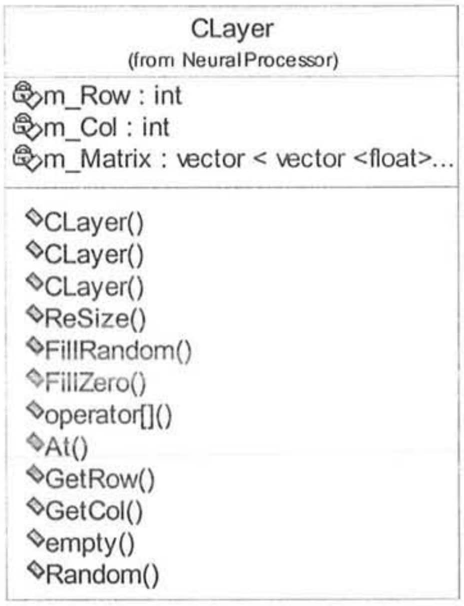
Class CWeight



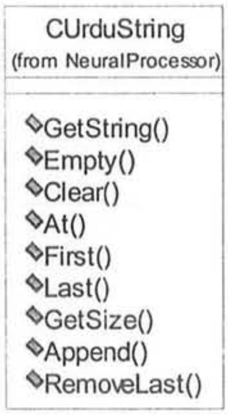
Class CLayerList



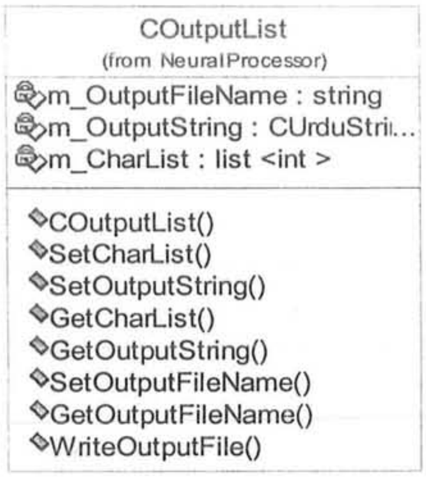
Class CLayer



Class CUrduString

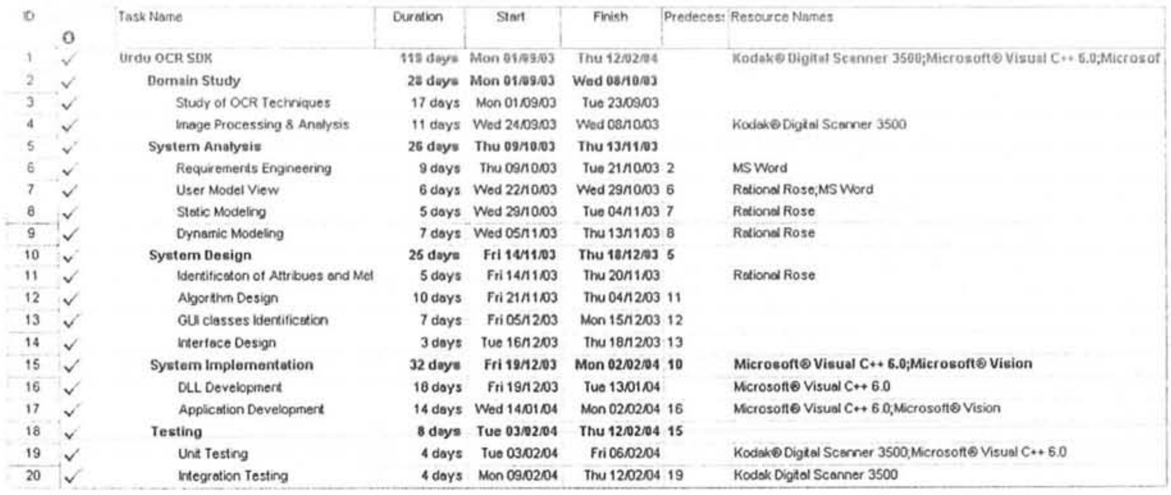


Class COutputList

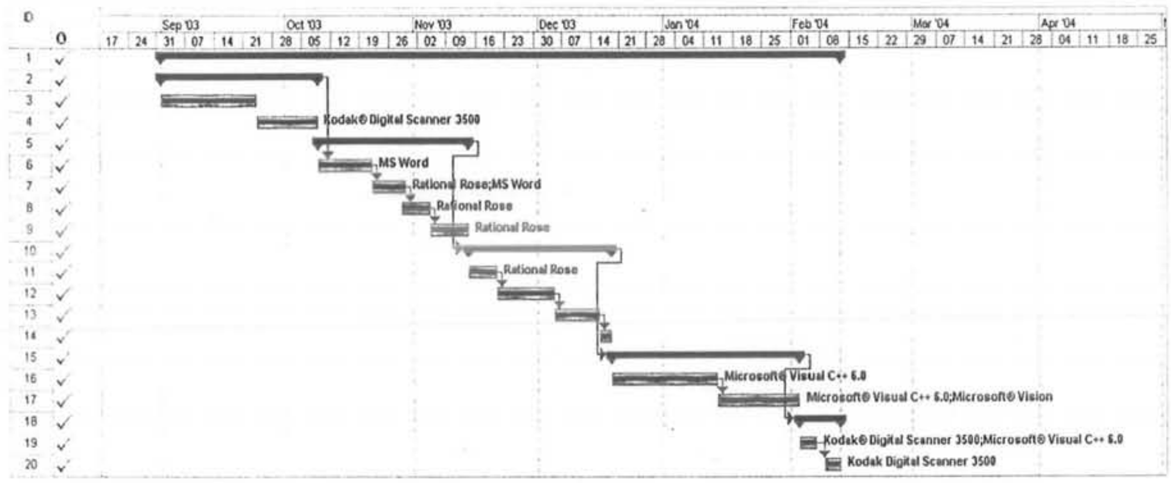


Appendix-C

1 Project Gantt Chart



ID	O	Task Name	Duration	Start	Finish	Predeces:	Resource Names
1	✓	Urdu OCR SDK	119 days	Mon 01/09/03	Thu 12/02/04		Kodak® Digital Scanner 3500;Microsoft® Visual C++ 6.0;Microsof
2	✓	**Domain Study**	**28 days**	**Mon 01/09/03**	**Wed 08/10/03**		
3	✓	Study of OCR Techniques	17 days	Mon 01/09/03	Tue 23/09/03		
4	✓	Image Processing & Analysis	11 days	Wed 24/09/03	Wed 08/10/03		Kodak® Digital Scanner 3500
5	✓	**System Analysis**	**26 days**	**Thu 09/10/03**	**Thu 13/11/03**		
6	✓	Requirements Engineering	9 days	Thu 09/10/03	Tue 21/10/03	2	MS Word
7	✓	User Model View	6 days	Wed 22/10/03	Wed 29/10/03	6	Rational Rose;MS Word
8	✓	Static Modeling	5 days	Wed 29/10/03	Tue 04/11/03	7	Rational Rose
9	✓	Dynamic Modeling	7 days	Wed 05/11/03	Thu 13/11/03	8	Rational Rose
10	✓	**System Design**	**25 days**	**Fri 14/11/03**	**Thu 18/12/03**	**5**	
11	✓	Identificaton of Attribues and Met	5 days	Fri 14/11/03	Thu 20/11/03		Rational Rose
12	✓	Algorithm Design	10 days	Fri 21/11/03	Thu 04/12/03	11	
13	✓	GUI classes Identification	7 days	Fri 05/12/03	Mon 15/12/03	12	
14	✓	Interface Design	3 days	Tue 16/12/03	Thu 18/12/03	13	
15	✓	**System Implementation**	**32 days**	**Fri 19/12/03**	**Mon 02/02/04**	**10**	**Microsoft® Visual C++ 6.0;Microsoft® Vision**
16	✓	DLL Development	18 days	Fri 19/12/03	Tue 13/01/04		Microsoft® Visual C++ 6.0
17	✓	Application Development	14 days	Wed 14/01/04	Mon 02/02/04	16	Microsoft® Visual C++ 6.0;Microsoft® Vision
18	✓	**Testing**	**8 days**	**Tue 03/02/04**	**Thu 12/02/04**	**15**	
19	✓	Unit Testing	4 days	Tue 03/02/04	Fri 06/02/04		Kodak® Digital Scanner 3500;Microsoft® Visual C++ 6.0
20	✓	Integration Testing	4 days	Mon 09/02/04	Thu 12/02/04	19	Kodak Digital Scanner 3500



References

[ROG00] Pressman Roger S., "Software Engineering, A practitioner's Approach", IEEE Software, 2000.

[IAN98] Sommervillie Ian, "Software Engineering", Addison Wesley, 1998.

[BER99] Oestereich Bernd, "Developing Software with UML", Addison Wesley, 1999.

Bibliography

[ROG00] Pressman Roger S., "Software Engineering, A practitioner's Approach", IEEE Software, 2000.

[IAN98] Sommervillie Ian, "Software Engineering", Addison Wesley, 1998.

[BER99] Oestereich Bernd, "Developing Software with UML", Addison Wesley, 1999.

[CHA98] Chapman Davis, "Teach Yourself Visual C++ 6 in 21 Days", SAMS, 1998.

[TOT98] Toth Viktor, "Programming Windows 98/NT Unleashed", SAMS, 1998.

[STR99] Eric Stroo, "Desktop Applications with Microsoft Visual C++ 6.0", Microsoft Press, 1999.

[WOO92] Gonzalez, R. C. and Woods, R. E. *"Digital Image Processing"*, Addison-Wesley, 1992.

[TG74] Tou, J. T. and Gonzalez, R. C., Pattern Recognition Principles, Addison-Wesley.

[UJR73]Ullman, J. R., Pattern Recognition Techniques, Butterworths, London

[DJ90] Dayhoff, J., Neural Network Architectures: An Introduction, Van Nostrand Reinhold, New York

Webliography

http://msdn.microsoft.com/

http://www.dacs.dtic.mil/techs/neural

http://citeseer.nj.nec.com

http://cs.felk.cvut.cz/~neurony/neocog/en

http://ct.radiology.uiowa.edu/~jiangm/courses/dip/html/

http://ct.radiology.uiowa.edu/~jiangm/courses/mm-cv-ip/

http://www.citeseer.org

Terms	Explanation
ANN	Artificial Neural Network
Back Propagation	A learning algorithm used to train an ANN.
Perceptron	A learning algorithm used to train an ANN.
OCR	Acronym for Optical Character Recognition.
Preprocessing	There is need of some tasks before recognition like Noise removal, Segmentation, Re-sampling and Skeltonization etc; and these are collectively called pre-processing.
SDK	Acronym for Software Development Kit
TIFF	Acronym for Tagged Image File Format, a famous raster image file format.
UML	Unified Modeling Language, These are notation used to describe system models of OOA & OOD etc.
Use Case	A scenario based methodology for election of requirements.
DLL	Dynamic Linked Library, used to wrap some functionality of code, which is dynamically linked during execution of software.
Test Cases	Test Cases are a set of inputs, which are used to check the system for its reliability and verification of code and validation of requirements, is done with the help of these.