Háskólinn á Akureyri

Viðskipta og raunvísindadeild

Final Year Project Dissertation

2007

Jón Orri Kristjánsson

Glyph Identification Using Neural Network Techniques

HORUS Project

Final Year Dissertation

Jón Orri Kristjánsson.

Supervisor: Dr. Nicola Whitehead.

School of Computing

Faculty of Business and Natural Sciences,

University of Akureyri.

Submitted April 2007, in partial fulfilment of the conditions of the award of the degree BSc.

I hereby declare that this interim report is all my own work, except as indicated in the text:

Signature ____________________________________

Date: 13/04/2007

I dedicate this project to all of those who have enough courage(stupidity?) to take on a neural network final year project without having had one single course in artificial intelligence nor C-Programming.

May God be with you !

Abstract

This document describes the work on the development of a semi-automatic hieroglyphic recognition system which uses neural network techniques. This system is developed for the HORUS project which is a cooperation between Nicola Whitehead, Nick Capanni and Stuart Watt. The necessary steps to create this system was to take an image in converting it to some grid and sending that on to a neural network which is the recognition part of the system. It is anticipated that this work will contribute towards the development of the HORUS project.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Project Description

This project's aim was to produce a system that was able to take in an image of up to 60 hieroglyphs aligned vertically and output the names of the glyphs. This process should be autonomous and the user shouldn't have to input a lot of parameters if the image imported was good.

1.2 Project Objectives

This project will work with a subset of 29 common glyphs in order to develop an identification system that can then be extended and generalized to a larger number of glyphs. Before they can be fed to the neural net, the individual glyphs must be extracted from an image containing a number of these glyphs, although it may not be necessary to use a neural net for this task.
A common problem with images of hieroglyphics is that they may be incomplete due to damage or variations in lighting. This is likely to affect the neural net's ability to recognize glyphs and will also be investigated.
(Whitehead et al. 2007)

1.3 Motivation for the work

Today the steps to translate a hieroglyph from a wall or a monument are the following:

- Take a picture of it.
- Manually input all the glyphs from the picture into a complex word processor.
- Translate all the glyphs into some language.

However what I would like to succeed with this project is to take out the 2^{nd} step in the preceding example. That is to try and be able to translate any glyph from a picture straight by inputting the picture. So the steps to translate the glyphs from the a wall or a monument would be as follows:

- Take a picture of it.
- Feed the picture into the system and get as an output the translation straightaway.

The main problem about image recognition like the problem at hand, is that images of the same thing can look really different according to the lighting in the picture and other variations in colour and etc. One approach to try and solve this problem is to try and build a neural network to identify the glyphs no matter how the lighting and colour changes in the pictures.
The project that I worked was aimed on a subset of 29 common glyphs but the system is also extendable so it can recognize all the other glyphs.
The things I had to do to complete this task were to:

- Recognize individual glyphs from a picture with a couple of glyphs on top of each other.
- Create a system that is able to identify the 29 common glyphs.
- Figure out a good learning mechanism so the neural network can recognize images with "noise"(image distortion, different lighting or variations in color).

1.4 Related Work

There are many systems available to recognize characters from images either handwritten or scanned in. However none of them can recognize either handwritten characters with noise nor hieroglyphics so none of them can actually be compared to the system that I am trying to build. I am going to talk about one system in particular. That is a character recognition system which was made by Andres Perez-Uribe. That is a character recognition system that Andres Perez-Uribe made to recognize characters and learn from mistakes made in the recognition.

1.4.1 Backpropagation with momentum

The system I found out to be the mose likemine is from a neural networks tutorial made by Andres Perez-Uribe in september 1993. That system uses the backpropagation algorithm with momentum to recognize numerical characters.

It gets as an input a 5x7 matrix of 0ťs and 1ťs. And then the network goes through the learning stage of the process. The networks performance is measured by calculating the outputs and the desired ones. The learning algorithm modifies the weight vectors accordingly to have the errors as low as possible. When the error goes under a certain threshold the learning is considered to be done and the system can be used to recognize the numbers. (Perez-Uribe 2002)

Advantages of my system over the Backpropagation with momentum system

Although the my system and the character recognition system that Andres Perez made arenŠt trying to do the same thing(one is recognizing numerical characters, other is recognizing hieroglyphs) they can be compared.
I would say the major advantages of my system are as follows:

- My system will be able to recognize more than one character at a time.
- My system takes as an input an image instead of the user inputting all the data.

1.4.2 Visual Character Recognition using Artificial Neural Networks

Shashank Araokar wrote a paper on how to recognize visual characters using artificial neural networks. That paper describes in some detail the steps that are needed to be taken when trying to build a system that is supposed to recognize visual characters with “noise".

He starts of by giving an example of an image digitization by projecting a letter onto a 6x8 grid. Then he goes on to explain a type of learning mechanism. After that he explains the network architecture. The network architecture consists of Candidate Scorre, Ideal Weight-Model Score and Recognition Quoitent. After that he talks a little about Performance issues. (Araokar n.d.)

1.5 Project Overview

The remaining of the document is structured as follows.
Chapter 1 provides the project description, project objectives, the importance of my project and my contribution to the project. Also It gives motivation for this work and a brief introduction to some related work.
Chapter 2 gives background information. Chapter 3 gives information about the system design, the architecture used and what kind the inputs and the outputs are in the system.
Chapter 4 gives information about the implementation of the system, also gives information about the technologies used in this project and also gives information about the time complexity of the system and the issues I landed in when implementing the system.
Chapter 5 gives information about the evaluation process of the system. In Chapter 6 you can read about how I can further this project any more and if I fulfilled all the objectives of the program and also whether or not this project is important.

Chapter 2

Background Readings

2.1 The HORUS Project

The HORUS project is a collaboration project between Dr. Nicola Whitehead from the University of Akureyri, Nick Capanni and Stuart Watt both from the School of Computing at the Robert Gordon University in Aberdeen, Scotland.

The problem as it is today is that to transcribe hieroglyphs you have to input all of the glyphs into a complex word processor and because there are over 4500 known glyphs thereof almost 800 that are known as common glyphs so it is a long and tedious process.

Hieroglyphs may both be drawn from left to right and top to bottom. So it is one of the problems to identify each of the glyphs in the image. There is most of the time a “divider" which is a whole line of white between the glyphs.

What HORUS strives to succeed is to make this process as automatic as possible.The best solution would of course be if the potential user would be able to throw into a system an image of glyphs and the system would crunct it down and output the transcription of the glyphs.

Potential users of the system might be for starters Egyptiologists but a further application might be used by tourists when travelling through Egypt and taking images of temples, walls or other incriptions. (Whitehead 2006)

2.2 Neural Networks

2.2.1 Neurons

A neuron consists of a certain number of inputs each of which has a certain "weight" assigned to it. The weights are simply an indication of how "important" that particular input is to the neuron. The "net" value of the neuron is then calculated. The net value is only a summed weight, which means that all the input neurons multiplied by their weights are summed together and if the net value goes over a certain threshold then the neuron fires(outputs 1) else it does nothing(outputs 0). The output is then fed to all the neurons that the neuron is connected to.(Generation5 2007*b*)

2.2.2 Learning

There exists a lot of options for neural networks to learn, for example the Kohonen, Delta and Back-Propagation learning algorithms. All of those go for the same end result, that is the neural network is always supposed to be "smarter" than in the previous run.(Generation5 2007*b*)
Most learning methods can be categorized in to two ways of learning, supervised and unsupervised. Supervised learning(for example the back-propagation) require a "teacher" to tell the neural network what the output of the net when presented with some input should be. The learning methods then convert all of the weights between the neurons. This process then loops until the network is able to recognize the input correctly. Unsupervised rules do not require a "teacher" because they just produce their output which is then further evaluated.(Generation5 2007*b*)
I am going to work with supervised learning in the back-propagation algorithm in this project.

2.2.3 Architecture

There are many types of architectures in neural networks(for example simple boolean networks(perceptrons) or self-organizing networks(Kohonen)). There is though one standard architecture.(Generation5 2007*b*)
All networks consist of several "layers" of neurons. (Rao 1995) In the image above you can see a

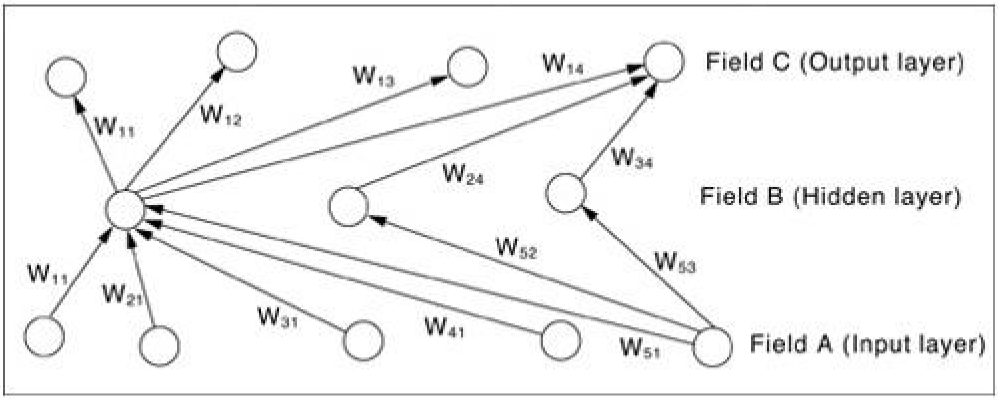


Figure 2.1: A Neural Network Structure with 3 Layers

neural network with 3 layers of neurons. The input layer takes the input and feeds that into the hidden layer which does all of the computations on the neurons. Then the hidden layer feeds the information into each of the output neurons and the output neuron that has the highest activation then either "fires"(outputs 1) or doesn't "fire"(outputs 0).(Generation5 2007*b*)

2.3 Bitmap File Structure

A .bmp file consists of 3 "elements":

- Bitmap file header (Size = 15 bytes)
- Bitmap info header (Size = 40 bytes)
- Pixel Colors

(Hetzl 1998)

Sometimes there is a RGB table between the bitmap info header and the pixel color. But in 24 bit bitmap(which is the one I am using in this project) there isn't any RGB table.

2.3.1 Bitmap File Header

start	size	name	stdvalue	purpose
1	2	bfType	19778	must always be set to 'BM' to declare that this is a .bmp-file.
3	4	bfSize	??	specifies the size of the file in bytes
7	2	bfReserved1	0	must always be set to zero.
9	2	bfReserved2	0	must always be set to zero.
11	4	bfOffBits	1078	specifies the offset from the beginning of the file to the bitmap data.

Table 2.1: Bitmap File Header(Hetzl 1998)

As you can see in this table the Bitmap File Header is 40 bytes large and contains some information that I have had to work with in this project. For example the bfSize and the bfOffBits values.

2.3.2 Bitmap Info Header

start	size	name	stdvalue	purpose
15	4	biSize	40	specifies the size of the BITMAPINFOHEADER structure, in bytes.
19	4	biWidth	100	specifies the width of the image, in pixels.
23	4	biHeight	100	specifies the height of the image, in pixels.
27	2	biPlanes	1	specifies the number of planes of the target device, must be set to zero.
29	2	biBitCount	8	specifies the number of bits per pixel.
31	4	biCompression	0	Specifies the type of compression, usually set to zero (no compression).
35	4	biSizeImage	0	specifies the size of the image data, in bytes. If there is no compression, it is valid to set this member to zero.
39	4	biXPelsPerMeter	0	specifies the the horizontal pixels per meter on the designated targer device, usually set to zero.
43	4	biYPelsPerMeter	0	specifies the the vertical pixels per meter on the designated targer device, usually set to zero.
47	4	biClrUsed	0	specifies the number of colors used in the bitmap, if set to zero the number of colors is calculated using the biBitCount member.
51	4	biClrImportant	0	specifies the number of color that are 'important' for the bitmap, if set to zero, all colors are important.

Table 2.2: Bitmap Info Header(Hetzl 1998)

In the bitmap info header I used the height, width and biBitCount in my system.

2.3.3 Pixel Colors

The color table is not present in the 24 bit bitmap because then each pixel is just presented with 24 bits of RGB colors.(DigiCamSoft 2007)
For example:

- FF FF FF = Full White or Full Light
- 00 00 00 = Full Black or No Light

2.4 Summary

This chapter talked about the 3 biggest parts I had to read up on and also gave a short introduction to each of them.

Chapter 3

System Design

The design part has changed a lot since i began to implement the system. I am going to show you the original design of the system and then go on in the next part to show you how I changed the design.

I can divide the overall task of the project into 5 smaller tasks so it will be clearer which steps need to be taken to fulfill the requirements of the project.

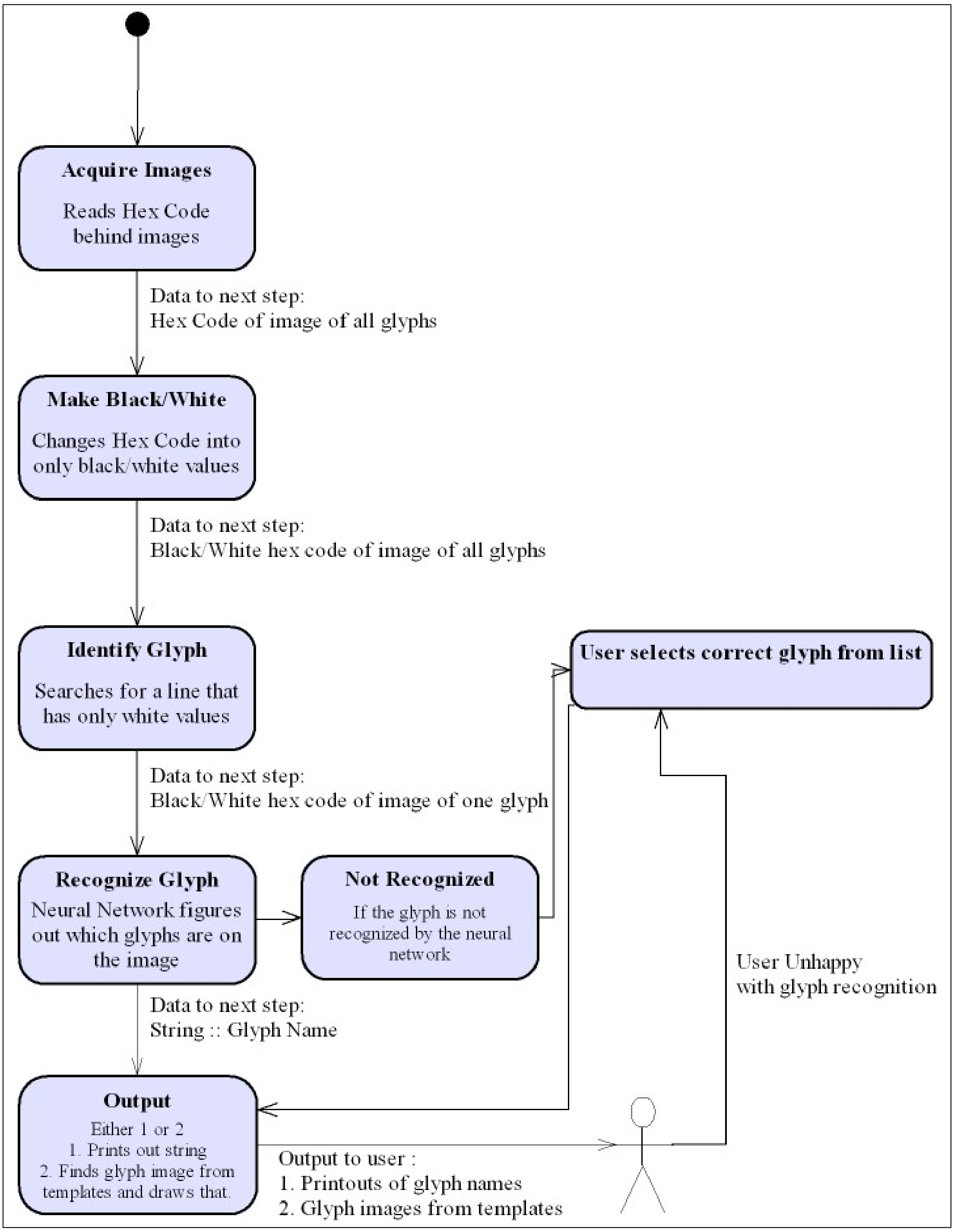


Figure 3.1: Data Flow Diagram - Original Full System

3.1 Acquire Images

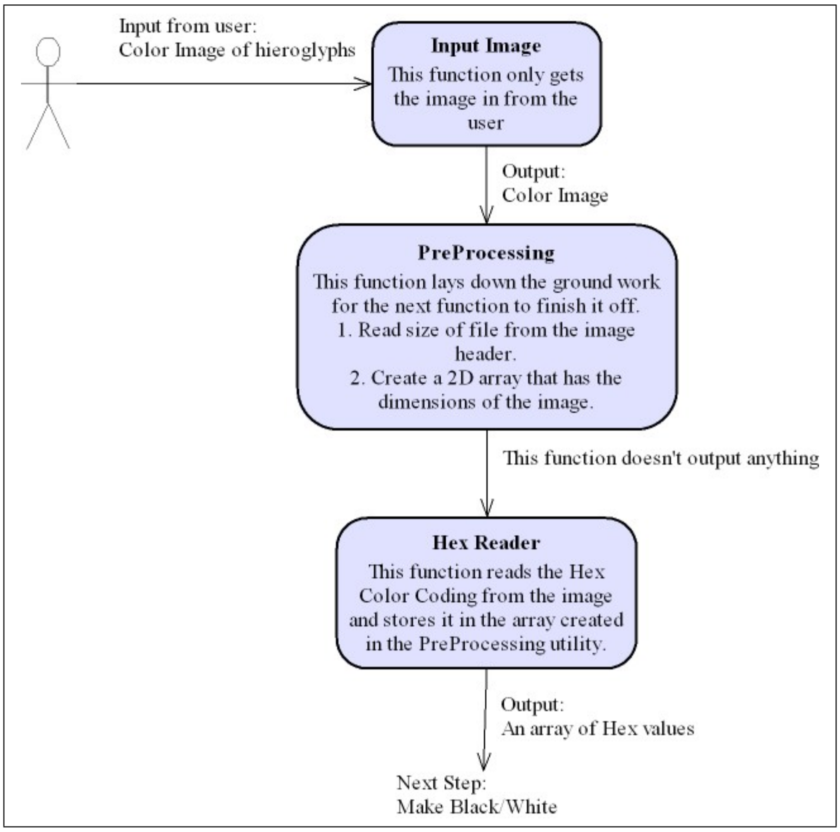


Figure 3.2: Data Flow Diagram - Acquire Images

This step has to take in the image input and return that image as an array to the next step.

3.1.1 Input Image

This function takes the image in from the user. Either through a command line argument or through some sort of a user interface.

Output: It sends the image through to the PreProcessing utility.

3.1.2 Preprocessing

This function lays down the ground work for the Hex Reader utility. It takes the input from the previous step and reads the size of the image from the header of the file and creates a 2-

Dimensional array for the Hex Reader utility.
Output: It doesnŠt output anything it only makes it sure the Hex Reader utility can go straight to work.

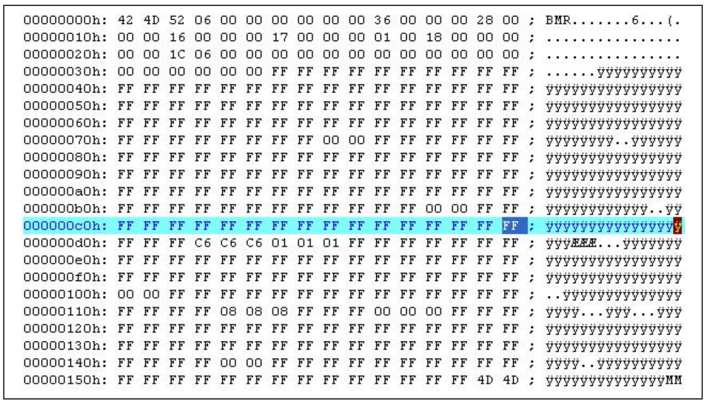
3.1.3 Hex Reader

This function reads the hexadecimal color coding in from the file and stores all the values in the array created by the PreProcessing utility.
Output: A 2-dimensional array of all the color codes of the image. It outputs to the next step which is the Make Black/White step.

3.1.4 Example

Figure 3.3: Example Hieroglyph

This is a sample picture and what I need to do in the next step is to read the hex code from the file. This particular file has hex code like the following:
These are the first lines of the hex code of the previous image. What I need to do in this step is to take all the color coding of the image and send that on to the next step.



```
00000000h: 42 4D 52 06 00 00 00 00 00 00 36 00 00 00 28 00 ; BMR.......6...(.
00000010h: 00 00 16 00 00 00 17 00 00 00 01 00 18 00 00 00 ; ................
00000020h: 00 00 1C 06 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000030h: 00 00 00 00 00 00 FF FF FF FF FF FF FF FF FF FF ; ......ÿÿÿÿÿÿÿÿÿÿ
00000040h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ; ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000050h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ; ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000060h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ; ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000070h: FF FF FF FF FF FF FF FF 00 00 FF FF FF FF FF FF ; ÿÿÿÿÿÿÿÿ..ÿÿÿÿÿÿ
00000080h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ; ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000090h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ; ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000000a0h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ; ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000000b0h: FF FF FF FF FF FF FF FF FF FF FF FF 00 00 FF FF ; ÿÿÿÿÿÿÿÿÿÿÿÿ..ÿÿ
000000c0h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ; ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000000d0h: FF FF FF C6 C6 C6 01 01 01 FF FF FF FF FF FF FF ; ÿÿÿÆÆÆ...ÿÿÿÿÿÿÿ
000000e0h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ; ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
000000f0h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ; ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000100h: 00 00 FF FF FF FF FF FF FF FF FF FF FF FF FF FF ; ..ÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000110h: FF FF FF FF 08 08 08 FF FF FF 00 00 00 FF FF FF ; ÿÿÿÿ...ÿÿÿ...ÿÿÿ
00000120h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ; ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000130h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF ; ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00000140h: FF FF FF FF 00 00 FF FF FF FF FF FF FF FF FF FF ; ÿÿÿÿ..ÿÿÿÿÿÿÿÿÿÿ
00000150h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF 4D 4D ; ÿÿÿÿÿÿÿÿÿÿÿÿÿÿMM
```

Figure 3.4: Example Hex Code

3.2 Make Black-White

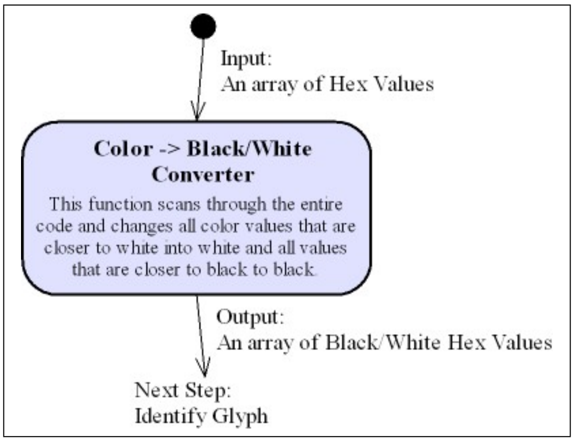


Figure 3.5: Data Flow Diagram - Make Black-White

This step of the system has to change the color codes into black or white codes. And return a black/white code array to the next step.

3.2.1 Color -> Black/White Converter

This utility scans through the entire code and changes the color values to only black or white color values. It chooses to change the values that are closer to white into white and the same thing for black.

Output: An array of Black/White hexadecimal values. It outputs to the next step which is the Identify Glyph step.

3.2.2 Example

A line of color hexadecimal codes:

FF D8 FF E0 00 10 4A 46 49 46 00 01 01 00 00

The line after it has gone through the Color -> Black/White Converter:

FF FF FF 00 00 00 FF FF FF 00 00 00 00 00 00

3.3 Identify Glyph

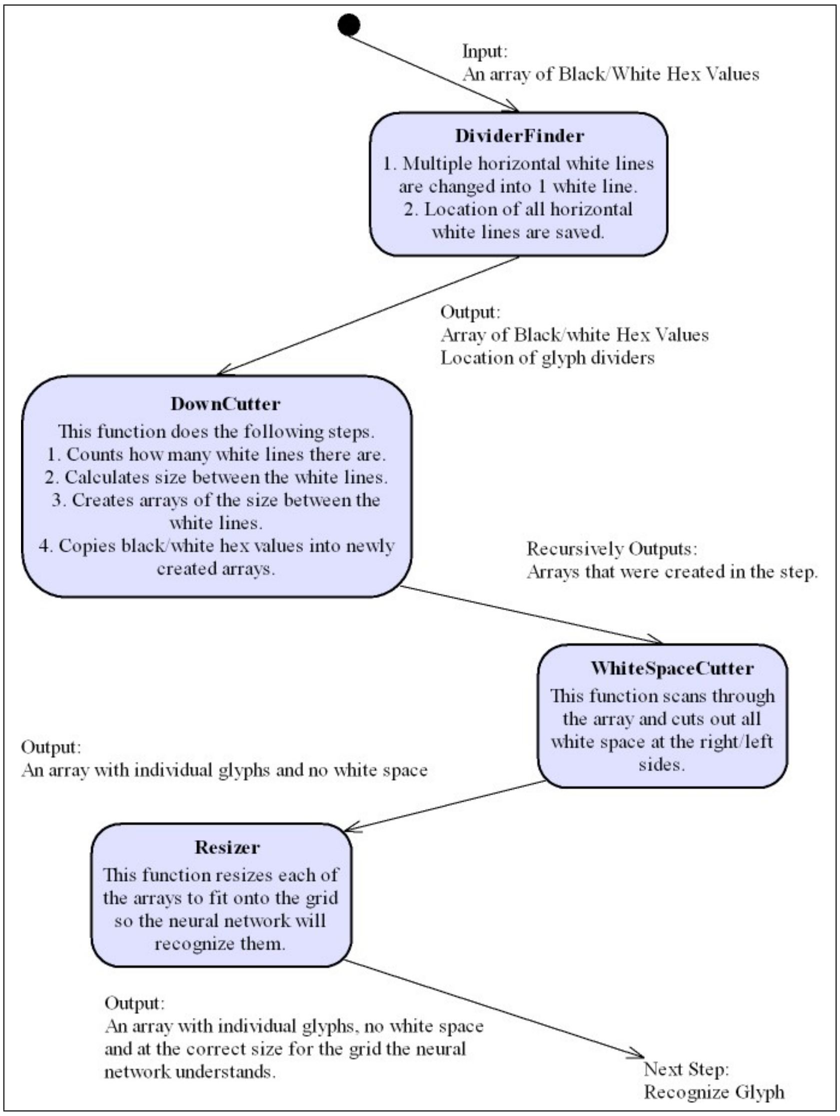


Figure 3.6: Data Flow Diagram - Identify Glyph

This part of the system has to identify an individual glyph and return it in as good a shape as it can.

3.3.1 DividerFinder

This function gets as an input an array of black/white hexadecimal values. It has to scan through the whole array and find lines that only contain whole white lines. It will then look above and under the line if it has more than a single white line. If it has more than one white line it will cut out the other lines. It will also save the position of all the white lines.
Output: Array of Black/White hexadecimal values. Location of the glyph dividers(white lines).

3.3.2 DownCutter

This function starts of with counting how many white lines there are(i.e. how many glyphs there are). It then calculates the size between the white lines. It then creates arrays(nr. of glyphs) of the size of the glyph. And then copies black/white values into the arrays.
Output: Recursively outputs the arrays that were created in the step.

3.3.3 White Space Cutter

This function scans through the code and cuts out all whole lines at the left and right sides of the glyphs.
Outputs: An array with individual glyphs and no white space on sides.

3.3.4 Resizer

This function resizes the arrays so it can fit onto the grid that is the default in the neural network.
Outputs: An array with individual glyphs, no white space and at the correct size for the neural network to recognize. Outputs to the Recognize Glyph step.

3.4 Recognize Glyph

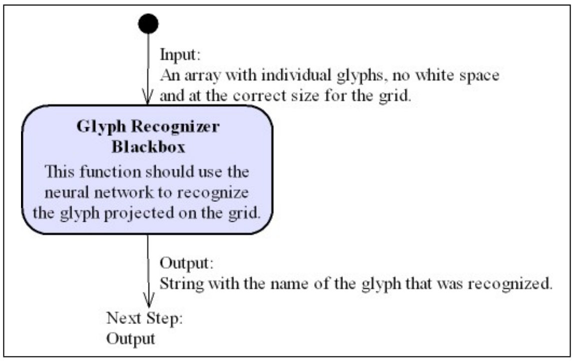


Figure 3.7: Data Flow Diagram - Recognize Glyph

This is the part where I modify the Backpropagation with momentum system. Some of the steps that I have to take to modify that system to fit my needs are the following:

- Enlarge the grid so it can fit a full size glyph.
- Make a training file where I input the training set of the algorithm. The system will enlarge the training set as it learns how to identify more and more things by user input.
- Change the number of inputs and outputs.

There are a lot of other modifications that I have to take so the system will fit my needs that I havenŠt identified yet.
This step returns the name of the glyph that was recognized.
If the neural net doesn't recognize the glyph, the user will have to choose the glyph from a template of glyphs.

3.5 Output

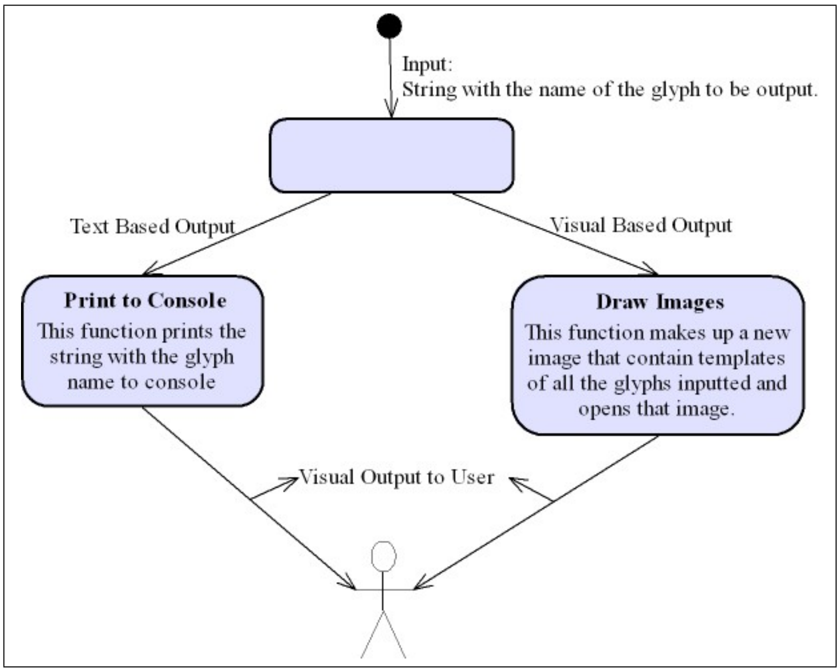


Figure 3.8: Data Flow Diagram - Output

This is where the user gets back the name or image of the glyphs he gave as an input in the first step.

3.5.1 Print to console

This step only prints out the name of the glyph that was taken in from the neural network.

3.5.2 Draw Images

This function draws the image taken from a template of all the 29 glyphs of the recognized glyph.

3.6 Input and Output

The system as it is today is divided into 3 parts.

- The image processing part.

- The neural network part.
- Test train file creator.

3.6.1 Image Processing

The image processing part takes in one image at a time with one glyph in it. It assumes that the glyph has been "traced"(that is filled in by the user with black) and it outputs back to the user a glyph with now whitespace and in perfect black-white "coloring".

3.6.2 Neural Network

The neural network takes in the test.dat and training.dat from the "Test train file creators.The image should have been sent from the image processing part and then the user should have resized it to a 40x40 image. It outputs the "odds" of the glyph to be one of the glyph the network has been tought.

3.6.3 Test train file creator

The part that creates the test and train files takes as a input an array of filenames that it is supposed to transform into test or training sets. It outputs back to the user "training.dat" and "test.dat" which hold all of the glyphs transformed.

3.7 Summary

This chapter told what the original system design was. Gave a detailed description of all the aspects of the system. But the design was changed a lot when the implementation phase began. It also gave a description of the input and output of the system.

Chapter 4

System Implementation

4.1 Technologies

When working with this project I was using the C-Programming language. I was also using the back-propagation neural network training algorithm.

4.1.1 C

Brief history

C was created by Dennis Ritchie at the Bell Telephone Laboratories in 1972 it was created to build the UNIX operating system. C was intended to be useful.
Because C was so flexible and a powerful language it was pretty quickly spread around a lot and because of that American National Standards Institute (ANSI) decided to make a standard which became known as ANSI Standard C. C was named C because its predecessor was named B. (Jones & Aitken 2002)

Why use C?

- C is a powerful and flexible language. What you can accomplish with C is limited only by your imagination. The language itself places no constraints on you. C is used for projects as diverse as operating systems, word processors, graphics, spreadsheets, and even compilers for other languages.

- C is a popular language preferred by professional programmers. As a result, a wide variety of C compilers and helpful accessories are available.

- C is a portable language. Portable means that a C program written for one computer system (an IBM PC, for example) can be compiled and run on another system (a DEC VAX system, perhaps) with little or no modification. Portability is enhanced by the ANSI standard for C, the set of rules for C compilers.

- C is a language of few words, containing only a handful of terms, called keywords, which serve as the base on which the language's functionality is built. You might think that a language with more keywords (sometimes called reserved words) would be more powerful. This isn't true. As you program with C, you will find that it can be programmed to do any task.

- C is modular. C code can (and should) be written in routines called functions. These functions can be reused in other applications or programs. By passing pieces of information to the functions, you can create useful, reusable code.

(Jones & Aitken 2002)

4.1.2 Feed-Forward Back-Propagation Neural Network

The feedforward backpropagation network is the most widely spread neural network training algorithm now. It does not have feedback questions but errors are backpropagated during training. Errors in the output determine measures of hidden layer output errors, which are used as a basis for adjustment of connection weights between the input and hidden layers. Adjusting the two sets of weights between the pairs of layers and recalculating the outputs is an iterative process that is carried on until the errors fall below a tolerance level. Learning rate parameters scale the adjustments to weights. A momentum parameter can also be used in scaling the adjustments from a previous iteration and adding to the adjustments in the current iteration.(Rao 1995)

Mapping

The feedforward backpropagation network maps the input vectors to output vectors. Pairs of input and output vectors are chosen to train the network first. Once training is completed, the weights are set and the network can be used to find outputs for new inputs. The dimension of the input vector determines the number of neurons in the input layer, and the number of neurons in the output layer is determined by the dimension of the outputs. If there are k neurons in the input layer and m neurons in the output layer, then this network can make a mapping from k-dimensional space to an m-dimensional space. Of course, what that mapping is depends on what pair of patterns or vectors are used as exemplars to train the network, which determine the network weights. Once trained, the network gives you the image of a new input vector under this mapping. Knowing what mapping you want the feedforward backpropagation network to be trained for implies the dimensions of the input space and the output space, so that you can determine the numbers of neurons to have in the input and output layers.(Rao 1995)

Training

The feedforward backpropagation network undergoes supervised training, with a finite number of pattern pairs consisting of an input pattern and a desired or target output pattern. An input pattern is presented at the input layer. The neurons here pass the pattern activations to the next layer neurons, which are in a hidden layer. The outputs of the hidden layer are obtained using a threshold function with the activations determined by the weights and the inputs. These hidden layer outputs become inputs to the output neurons, which process the inputs using a threshold function. The final output of the network is determined by the activations from the output layer.(Rao 1995)

Why use backpropagation?

Backpropagation neural network is a very good training algorithm. It is also good that it isn't really hard to implement and that it produces good results most often. Also I decided on using the backpropagation because it is so widely used and therefore a lot of resources available for that but not for example the Kohonen network.

Notation & Equations Used

The backpropagation uses a lot of mathematics to derive all the weight changes in the system. I am going to give a brief explanation about the equations and notation used and also where in my(Andres's) system they are used.

M1 = Interface between the input and the hidden layer

M2 = Interface between the hidden and the output layer

x[i] = Output of the ith input neuron

y[i] = Output of the ith hidden neuron

z[i] = Output of the ith output neuron

P = Desired output pattern

m = Number of input neurons

β_h = Learning rate

Δ = Change in a parameter

$e_j's$ = Error in output at the output layer

$t_i's$ = Error in output at the hidden layer

α = Momentum

$$y_j = f((\Sigma_i x_i M_1[i][j])) \tag{4.1}$$

Output of jth hidden layer neuron - function answerFromNet

$$z_j = f((\Sigma_i y_i M_2[i][j])) \tag{4.2}$$

Output of jth output layer neuron - function answerFromNet

$$desiredvalue - computedvalue = P_i - z_i \tag{4.3}$$

ith component of vector of output differences - function betaErrorOutput

$$e_i = (P_i - z_i) \tag{4.4}$$

ith component of output error at the output layer - function betaErrorOutput

$$t_i = y_i(1 - y_1)(\Sigma_j M_2[i][j]e_j) \tag{4.5}$$

ith component of output error at the hidden layer - function betaHiddenOutput

$$\Delta M_2[i][j](t) = \beta y_i e_j + \alpha \Delta M_2[i][j](t-1)] \tag{4.6}$$

Adjustment for weight between ith neuron in hidden layer and jth output neuron - function backpropagation

$$\Delta M_1[i][j](t) = \beta x_i t_j + \alpha \Delta M_1[i][j](t-1) \tag{4.7}$$

Adjustment for weight between ith neuron in input layer and jth hidden neuron - function backpropagation

$$f(x) = \frac{1}{1 + e^{-x}} \tag{4.8}$$

Sigmoid Function

(Rao 1995)

4.2 Implementation Issues

There were a lot of issues I landed on when implementing the system.

- Maximum Array Size
- Delimiter Problem
- Image Noise
- Resizing

4.2.1 Maximum Array Size

In the original system design I was hoping that I would be able to keep the image in the system at all times. I was going to keep all the color codes in an array but soon I found out that when

I tried to do that the arrays didn't give the same values at all times. So I read up on arrays in C and found out that C sometimes a threshold of 64 KB on arrays and that it wasn't advised to have arrays bigger than that.(Jones & Aitken 2002)

4.2.2 Delimiter Problem

When working with the bitmaps I found out that sometimes there were some extra 00 bytes at the end of each line in the image. I had a lot of problem trying to find out why those problems were and it wasn't until I was reading about image file formats trying to find out how the compression in jpeg's images was that I found out that those bytes are called junk bytes and they are added when each line in the system isn't dividible by 4. So I added to my system a "junk byte" finder. I added this equation to my system and found out that it worked.

$$4 - ((pixelPerColumn * 3)\%4) = junkbytes \tag{4.9}$$

4.2.3 Image Noise

When I was working throughout this project I took a lot of images of glyphs that I was going to feed through my system. But almost all the time I found out that the images were too dark to be able to feed through the image processing part and get something that made sense out of it. I tried out a lot of different cameras and stamp colors but at the end I decided on concentrating on other thins and decided that I wanted the user to "trace" the glyph with black color so that problem would be out of the history.

4.2.4 Resizing

This was one of the biggest problems I landed in because I had images that were in many different sizes and I wanted all of them to fit onto a 40x40 grid. If a person wants that procedure to work well it is a really hard to implement that. So I decided that I wanted the user to feed the image to some image processing tool and resize the image onto a 40x40 grid preferrably with the nearest neighbor technique.

4.3 The System

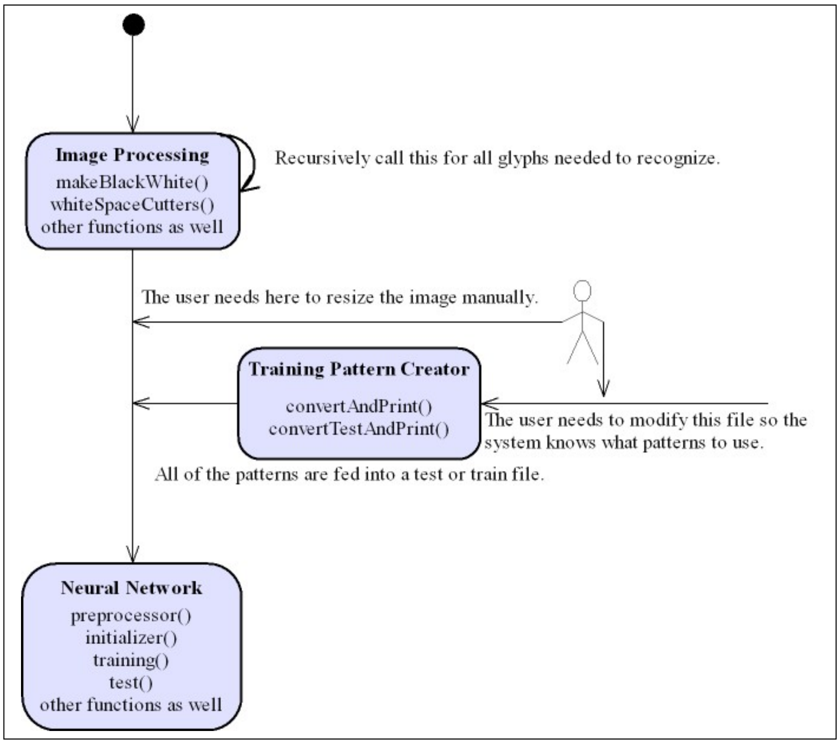


Figure 4.1: Data Flow Diagram - Whole Complete System

This is a flow chart of the whole system. The user throws all of the images he wants to recognize into the image processing part. Then he resizes all of the images onto a 40x40 grid. Then he runs the training pattern creator which creates both the test.dat and training.dat. All of the glyphs the user wants to recognize are in the test.dat file after he modifies the training pattern creator accordingly. But in the training.dat all of the files used for training reside. Then the user needs to run the neural network part of the system and that prints out what glyph is in what image.

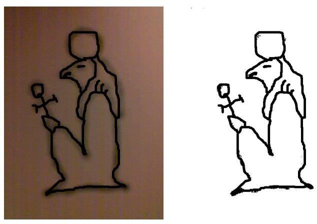
4.3.1 Image Processing

Manual Operations #1

The user has to "trace" the image so the recognition part can go more smoothly, i.e. the user has to color the glyph in the image black.

Make Black-White

I have got 2 versions of the black-white converter. One of them uses a threshold method that searches the darkest spot in the image while the other one assumes the user has "traced" the glyph with black color, because it "whitens" all the other pixels that aren't black. Currently I am working on the later one because the first one wasn't working as well as I had hoped for.
When driving the image through the neural network it is really important to know that the image consists of only the glyph but not a big black "blob" because then the neural net gets confused. So I went with the tracing version of this function. That function works perfectly.
In the figures below you can see the difference when faced with color images in the 2 different procedures I implemented.

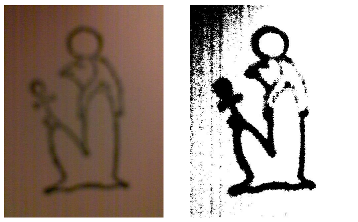


(a) Before Black-White Conversion (b) After Black-White Conversion

Figure 4.2: Black-White Conversion Traced Images

White Space Cutters

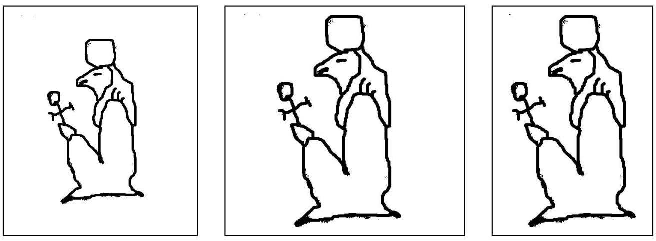
My white space cutters were in the final version exactly as the ones I had designed in the beginning. They start off by searching through the entire file looking for white lines(vertical) and then the image is sent off to the next function which is the downCutterHorizontal which



(a) Before Black-White Conversion (b) After Black-White Conversion

Figure 4.3: Black-White Conversion Untraced Images

takes out all of the lines that are all white. Originally I was working with images that had a lot of images aligned vertically so the function was a bit different then than it is now because I decided that I was going to work with only one glyph at a time in this stage of the system.
Then I went off to cut all the white columns out. That procedure was similar than to the first one. That is first I look at all the columns in the image and check if there are some that are all white, if there are I mark them. Then I send the image and the marked columns into the downCutterVertical function which prints all of the color codes(that aren't marked) back into a file.



(a) Before White Space Cutting (b) After Vertical White Space Cutting (c) After All White Space Cutting

Figure 4.4: Applying the White Space Cutters

Manual Operations #2

Now the user has to resize the image manually. He can for example open up the image in Photoshop and resize the image there or use some other image manipulation tool. But the user has to watch out for what algorithm he uses. The best one too use at this stage is the nearest neighbor algorithm.

4.3.2 Neural Network

As you can see in the figure 4.5 image the neural network is built up of various parts.

Functions	Main Purpose
main()	The main function is the driver of the whole system.
preprocessor()	The preprocessor function is the function that creates all the arrays needed.
initializer()	The initializer function is the function that gives all the arrays default values.
training()	The training function trains the network by calling the backpropagation and the answerFromNet functions.
test()	The test function is the recognizing part of the system
answerFromNet()	The answerFromNet function checks too see the output activations when presented with a pattern.
backpropagation()	The backpropagation function adjusts the weights of the network.
sigm(x)	The sigm function is the sigmoid function in code.
betaErrorOutput()	betaErrorOutput calculates the output error at the output layer.
betaErrorHidden()	betaErrorHidden calculates the output error at the hidden layer.
error()	The error function checks to see the overall error of the net.
errorMeasure()	The errorMeasure calculates error for each pattern of the network.

Table 4.1: Neural Network Architecture

4.3.3 Test & train file creator

The test & train file creator takes in as arguments an array both of train & test images to convert to test.dat & train.dat.

It has 4 vital objects:

- char *trainPattFilenames[47]
- char *testPattFilenames[47]
- void convertAndPrint(int nr);
- void convertTestAndPrint(int nr);

Those items are the biggest ones in the file creator system. The trainPattFilenames contains all of the images that the user wants to train with.

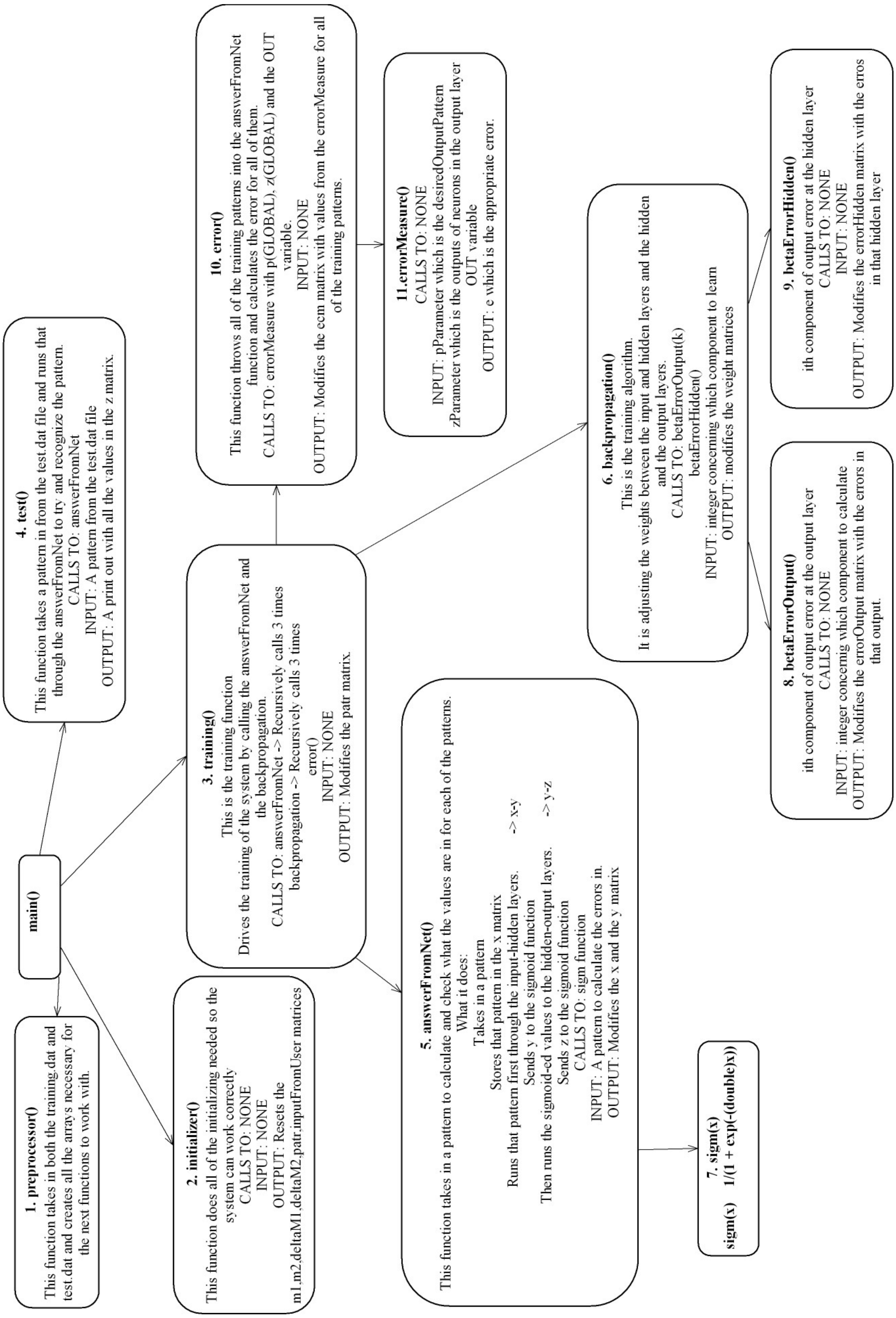


Figure 4.5: Data Flow Diagram - Neural Network

The testPattFilenames contain all of the images that the user wants to test with after the system has trained with the patterns from the previous array. Of course the user should train with "variations" of the patterns he is testing with.

The convertAndPrint takes in one image at a time from the trainPattFilenames array and converts that to the train.dat file. It takes all of the pixel codes in and converts them to either 0 or 1.

The convertTestAndPrint takes in one image at a time from the testPattFilenames array and converts that to the test.dat file. It takes all of the pixel codes in and converts them to either 0 or 1.

At the beginning of the file the number of training or test patterns is printed and there is also printed an additional -1 at the end of each glyph. Also before the color codings of a image the number(in the array) of the glyph is printed.

4.4 Summary

In this chapter I have given a thorough description of the implementation of the system like it is today. I started of by explaining briefly the major technologies I have been using throughout this project(C and Backpropagation). Then I talked about the major issues I have landed in in this project. Then I gave a description about the system and why I changed from the original design plans.

Chapter 5

Evaluation

I am going to test the neural network to see how it can handle incomplete or “bad” data. It is vital to see how the neural network recognizes various patterns because if it does badly the net might need some retuning(i.e. change in parameters) and etc. . .

Since there is a limit of how big an array can be in C then the evaluation can’t be done properly until that aspect in the system has been changed. So the neural network has to learn with maximum of 7 patterns.

To be able to let the neural network learn enough of each pattern I decided to limit the number of different patterns by 5 because the maximum amount of patterns there can be in the training file is 30 so I may at maximum have 6 patterns of each glyph.

The glyphs I am using in this experiment are the following:

- God
- Lion
- Scarab
- Vulture
- Red Flower

I have got 6 instances of each pattern and then I feed another instance of each of those patterns into the test.dat file for the neural network to recognize.

5.1 Testing

5.1.1 Experiment #1

Parameter Settings:

Epsilon(Maximum Mean Squared Error) = 0.001

beta(learning rate) = 0.05

alpha(momentum) = 0.1

Number of test patterns = 30

Number of train patterns = 5

I ran this through the neural network and got this result.

Number of patterns:5

Number of Glyph:0 Output activations : z[0] = 0.993970 z[1] = 0.001356 z[2] = 0.030588 z[3] = 0.004305 z[4] = 0.010111	Number of Glyph:1 Output activations : z[0] = 0.984573 z[1] = 0.001223 z[2] = 0.029213 z[3] = 0.007750 z[4] = 0.009906	Number of Glyph:2 Output activations : z[0] = 0.966921 z[1] = 0.002405 z[2] = 0.001555 z[3] = 0.008794 z[4] = 0.020329
Number of Glyph:3 Output activations : z[0] = 0.975136 z[1] = 0.000280 z[2] = 0.006607 z[3] = 0.028601 z[4] = 0.007874	Number of Glyph:4 Output activations : z[0] = 0.993970 z[1] = 0.001356 z[2] = 0.030588 z[3] = 0.004305 z[4] = 0.010111	

Table 5.1: Experiment # 1 - Output Activations

5.1.2 Experiment #2

Parameter Settings:

Epsilon(Maximum Mean Squared Error) = 0.001

beta(learning rate) = 0.01

alpha(momentum) = 0.01

Number of test patterns = 25

Number of train patterns = 10

Number of patterns:10

Number of Glyph:0 Output activations : z[0] = 0.967878 z[1] = 0.002802 z[2] = 0.020483 z[3] = 0.005207 z[4] = 0.021415	Number of Glyph:0 Output activations : z[0] = 0.976634 z[1] = 0.008815 z[2] = 0.021797 z[3] = 0.010731 z[4] = 0.026502	Number of Glyph:1 Output activations : z[0] = 0.961378 z[1] = 0.003475 z[2] = 0.001340 z[3] = 0.013620 z[4] = 0.017066	Number of Glyph:1 Output activations : z[0] = 0.963795 z[1] = 0.002044 z[2] = 0.010535 z[3] = 0.019611 z[4] = 0.012237
Number of Glyph:2 Output activations : z[0] = 0.967311 z[1] = 0.002071 z[2] = 0.009370 z[3] = 0.020297 z[4] = 0.019959	Number of Glyph:2 Output activations : z[0] = 0.013619 z[1] = 0.963674 z[2] = 0.006133 z[3] = 0.004257 z[4] = 0.018826	Number of Glyph:3 Output activations : z[0] = 0.026418 z[1] = 0.975261 z[2] = 0.000117 z[3] = 0.007425 z[4] = 0.023870	Number of Glyph:3 Output activations : z[0] = 0.002288 z[1] = 0.994421 z[2] = 0.001989 z[3] = 0.006747 z[4] = 0.002565
Number of Glyph:4 Output activations : z[0] = 0.013688 z[1] = 0.971023 z[2] = 0.007569 z[3] = 0.016969 z[4] = 0.024083	Number of Glyph:4 Output activations : z[0] = 0.003970 z[1] = 0.968554 z[2] = 0.015578 z[3] = 0.009412 z[4] = 0.024479		

Table 5.2: Experiment # 2 - Output Activations

5.2 Results

As you can see all the patterns classify as pattern 0 or 1 which is the God or Lion pattern. So there is a big error in the neural network somewhere but that seems to be an error that is locateable because the backpropagation algorithm is quitting the learning to soon.

5.3 Summary

This chapter described the evaluation I did on the system. It turned out not to produce good results. That is only because there is an error in the backpropagation algorithm.

Chapter 6

Conclusions

6.1 Objectives Reflection

The objectives in this project were the following from the beginning:

1. Take in an image with some number of glyphs in it.
2. Change it to black-white.
3. Find individual glyphs in the image.
4. Cut all the whitespace from the individual glyphs.
5. Resize the image.
6. Feed the image into a neural network which recognizes the glyph.
7. Output the name of each of the glyphs to the user again.

I have fulfilled some of those objectives fully but some only partially and even some that I haven't started working on yet.
I finished the first task partially. Partially because I went with the way of taking only one glyph in at a time but that is fairly easy to change again. I finished the making black-white fully and even in 2 different ways because the first one which was purely manual didn't work exactly as I

Functionality	What is left to do:
Black-White	I have to fix the black-white converter. So it can convert appropriately images that haven't been traced.
IdentifyGlyph	Implement the identifyGlyph part so it can identify multiple glyphs in one image.
Resizer	Implement the resizer function. Most likely with the nearest neighbour algorithm

Table 6.1: Current System - What is left to do

had hoped for. So I went with the other way but that one works perfectly.

I haven't got the find individual glyphs part in the current system. The problem was that I landed in the delimiter problem and I didn't find out how to come by that problem until it was too late to set that functionality back in. So that part isn't completed in the current system.

I have fully finished the white space cutting. That works as good as I had hoped for and I can say it works perfectly.

I didn't start the resizing functionality because of time constraints. I just made the assumption that the user manually resized the image.

I have partially finished the neural network part. The neural network accepts images but the network needs some tuning because it does a terrible job of recognizing the images. So the neural network need some tuning before I can say that that part has finished.

Like the system is today then it goes through the neural network and prints out which image in the array it is. It doesn't output the number it justs outputs an index in the array with the filenames in.

6.2 Further Work

There is a lot left to do in this project. The biggest things left to do are the following:

I have also thought of a lot of things to further this project too fulfill objectives that weren't in this project. Among those things are the following:

- When the outputs of the neural network fall below a certain threshold then the user is presented with a lot of ways to come by that problem. For example to lighten the picture with a function, erase “noise” or too dark spots with a function, draw a box around a glyph to identify exactly where the glyph is positioned in the image. The user should also be able to see all of the images(per each stage in the image processing part) in some sort of GUI so he can see where the system went wrong and correct that part.

- I have also thought of some methods so the system can reside on a server so a user can access the program through a server-client approach through a web browser. It is possible to access a C program through a clients web browser when using ASP scripts and some other technologies as well.

But the main priority right now is finishing the objectives stated in the beginning. That is that the user is able to input an image with a lot of glyphs aligned vertically and get as output the names of the glyphs.
To do that I have to finish the resizer, black-white, identifyGlyph and recognizeGlyph functions.

6.3 Importance and Contribution

What I did in this project isn't important to the HORUS project as it is. It has to be fully done so the project can be a factor in the HORUS project. The stage that they are in today needs that the glyphs can be recognized so the project can be published.

6.4 My Work

What I did in this project to make the system like it is today:

- Read a lot on neural networks.
- Found out what the main parts of the network do.
- Found out how to tune the network so it functions better.
- Learnt how to program in C.
- Wrote up the whole system like it is today.

- Wrote up a lot of functionality that didn't end up in the final system because of errors I had problems in coming through(like the delimiter problem).
- Learnt how to write reports in Latex.
- Wrote the interrim report.
- Wrote the final dissertation.
- Held a presentation.
- Had meetings with the supervisor.
- Came by endless amounts of problems.
- Made a user manual for the system. Can see it in Appendix C

6.5 Personal Reflections

Working on this project has made me realize how much work implementing and designing a system from scratch can be. It has been a good and satisfying experience.
Working on this project has made me more capable of doing the following:

- Writing reports.
- Organizing myself.
- Organizing my work.
- Designing a system.
- Implementing a system based on designs.
- Coming by problems.
- Avoiding problems.
- Presenting my work.

- Working with other people to fulfill the project's objectives(Nicola Whitehead)
- Finding resources.
- Implementing big systems in C.
- Working with neural networks.
- Expressing myself in English.
- Writing reports in Latex.
- Working long hours with high concentration.

When I first decided to take this project I wanted to build a big system preferrably with some kind of AI technique. So I decided on taking this project. I also had Nik as a supervisor in my Group Project(2nd year) and I liked the way she worked. After working on this system and endless amounts of printouts read I found out that Neural Networks are an interesting sector to maybe work in one day. I had hoped to find this project fun to work on but it exceeded those expectations.

When I was working on this project I always found it more fun when I got better at the skills I was working on. Especially when implementing in C I found it so much fun when I found out new ways to do things and better ways to work with pointers and memory locations. Also when I was working with the neural network I always found it fun when I found out what each part of the network did and how it was calculated.

The overall conclusion of this project is that I found this a fun and interesting project and the main technologies(C and backpropagation) interesting subjects to maybe work on in the future.

Bibliography

Araokar, S. (n.d.), 'Visual character recognition using artificial neural networks'. [Online; accessed 3-April-2007].
URL: *http://arxiv.org/ftp/cs/papers/0505/0505016.pdf*

Chhabra, T. (2006), 'Back-propagation neural net'. [Online; accessed 3-April-2007].
URL: *http://www.codeproject.com/cpp/BP.asp*

DigiCamSoft (2007), 'Bmp file format'. [Online; accessed 3-April-2007].
URL: *http://www.digicamsoft.com/bmp/bmp.html*

Generation5 (2007*a*), 'Back-propagation for the uninitiated'. [Online; accessed 3-April-2007].
URL: *http://library.thinkquest.org/18242/bp.shtml*

Generation5 (2007*b*), 'Introduction to neural networks'. [Online; accessed 3-April-2007].
URL: *http://library.thinkquest.org/18242/nnintro.shtml*

Generation5 (2007*c*), 'Multilayer feedforward network and the backpropagation algorithm'. [Online; accessed 3-April-2007].
URL: *http://library.thinkquest.org/18242/nn_bp.shtml*

Generation5 (2007*d*), 'Perceptrons'. [Online; accessed 3-April-2007].
URL: *http://library.thinkquest.org/18242/perceptron.shtml*

Heaton, J. (n.d.), 'Introduction to neural networks with java'. [Online; accessed 3-April-2007].
URL: *http://www.heatonresearch.com/articles/series/1/*

Hetzl, S. (1998), 'The .bmp file format'. [Online; accessed 3-April-2007].
URL: *http://www.fortunecity.com/skyscraper/windows/364/bmpffrmt.html*

Jones, B. L. & Aitken, P. (2002), *Sams Teach Yourself C in 21 Days*, sixth edn, Sams.

Kröse, B. & van der Smagt, P. (1996*a*), 'An introduction to neural networks'. [Online; accessed 3-April-2007].
URL: *http://www.avaye.com/files/articles/nnintro/nn_intro.pdf*

Kröse, B. & van der Smagt, P. (1996*b*), 'An introduction to neural networks'. [Online; accessed 3-April-2007].
URL: *http://www.avaye.com/files/articles/nnintro/nn_intro.pdf*

McCollum, P. (2004), 'An introduction to back-propagation neural networks'. [Online; accessed 3-April-2007].
URL: *http://www.seattlerobotics.org/encoder/nov98/neural.html*

McCollum, P. (n.d.), 'An introduction to back-propagation neural networks'. [Online; accessed 3-April-2007].
URL: *http://www.seattlerobotics.org/encoder/nov98/neural.html*

Perez-Uribe, A. (2002), 'Neural networks tutorial'. [Online; accessed 3-April-2007].
URL: *http://lslwww.epfl.ch/~anperez/NN_tutorial/NNdemo_example.html*

Rao, V. B. (1995), *C++ Neural Networks and Fuzzy Logic*, M&T Books, IDG Books Worldwide, Inc.

Silverman, D. C. (n.d.), 'Tutorial on artificial neural networks'. [Online; accessed 3-April-2007].
URL: *http://argentumsolutions.com/tutorials/neural_tutorialpg1.html*

Whitehead, N. (2006), 'Niksvik - horus project'. [Online; accessed 3-April-2007].
URL: *http://notendur.unak.is/not/nicolaw/horus.html*

Whitehead, N., Chan, T., Gagunashvili, N. & Yuan, T. (2007), 'Final year project description 2006 2007'. Description of the project made by Dr. Nicola Whitehead.
URL: *As a word document*

Wikipedia (2007), 'Windows and os/2 bitmap — wikipedia, the free encyclopedia'. [Online; accessed 3-April-2007].

URL: *http://en.wikipedia.org/w/index.php?title=Windows_and_OS/2_bitmap&oldid=119364180*

Appendix A

Code Listing

A.1 trainingPatternCreator.c

```
#include <stdio.h>
#include <stdlib.h>

#define NUMTRAIN 25 // Number of training samples
#define NUMTEST 10 // Number of test samples
#define instancesPerPattern 5 // instances per each
pattern in the training samples
#define instancesPerPatterTest 2 // instances per each
pattern in the test samples
#define filenameTrain "training.dat" // filename for
the training samples
#define filenameTest "test.dat" // filename for the
test samples
#define headerSize 54 // size of the header

void convertAndPrint(int nr);
void convertTestAndPrint(int nr);
```

```

char *trainPattFilenames[47] = {
                                    "BitmapsToFeedToNeura
                                    lNet40x40/God1.bmp",
                                    "BitmapsToFeedToNeura
                                    lNet40x40/God2.bmp",
                                    "BitmapsToFeedToNeura
                                    lNet40x40/God3.bmp",
                                    "BitmapsToFeedToNeura
                                    lNet40x40/God4.bmp",
                                    "BitmapsToFeedToNeura
                                    lNet40x40/God.bmp",
                                    "BitmapsToFeedToNeura
                                    lNet40x40/Lion1.bmp",
                                    "BitmapsToFeedToNeura
                                    lNet40x40/Lion2.bmp",
                                    "BitmapsToFeedToNeura
                                    lNet40x40/Lion3.bmp",
                                    "BitmapsToFeedToNeura
                                    lNet40x40/Lion4.bmp",
                                    "BitmapsToFeedToNeura
                                    lNet40x40/Lion.bmp",
                                    "BitmapsToFeedToNeura
                                    lNet40x40/Red
                                    Flower1.bmp",
                                    "BitmapsToFeedToNeura
                                    lNet40x40/Red
                                    Flower2.bmp",
                                    "BitmapsToFeedToNeura
                                    lNet40x40/Red
```

```
Flower3.bmp",
"BitmapsToFeedToNeura
lNet40x40/Red
Flower4.bmp",
"BitmapsToFeedToNeura
lNet40x40/Red Flower.
bmp",
"BitmapsToFeedToNeura
lNet40x40/Scarab1.
bmp",
"BitmapsToFeedToNeura
lNet40x40/Scarab2.
bmp",
"BitmapsToFeedToNeura
lNet40x40/Scarab3.
bmp",
"BitmapsToFeedToNeura
lNet40x40/Scarab4.
bmp",
"BitmapsToFeedToNeura
lNet40x40/Scarab.
bmp",
"BitmapsToFeedToNeura
lNet40x40/Vulture1.
bmp",
"BitmapsToFeedToNeura
lNet40x40/Vulture2.
bmp",
"BitmapsToFeedToNeura
lNet40x40/Vulture3.
```

```
                                    bmp",
                                    "BitmapsToFeedToNeura
                                    lNet40x40/Vulture4.
                                    bmp",
                                    "BitmapsToFeedToNeura
                                    lNet40x40/Vulture.
                                    bmp",
                                };

char *testPattFilenames[47] = {
                                   "BitmapsToFeedToNeural
                                   Net40x40/God6.bmp",
                                   "BitmapsToFeedToNeural
                                   Net40x40/God5.bmp",
                                   "BitmapsToFeedToNeural
                                   Net40x40/Lion6.bmp",
                                   "BitmapsToFeedToNeural
                                   Net40x40/Lion5.bmp",
                                   "BitmapsToFeedToNeural
                                   Net40x40/Red Flower6.
                                   bmp",
                                   "BitmapsToFeedToNeural
                                   Net40x40/Red Flower5.
                                   bmp",
                                   "BitmapsToFeedToNeural
                                   Net40x40/Scarab6.bmp",
                                   "BitmapsToFeedToNeural
                                   Net40x40/Scarab5.bmp",
                                   "BitmapsToFeedToNeural
                                   Net40x40/Vulture6.
```

```
                                    bmp",
                                    "BitmapsToFeedToNeural
                                    Net40x40/Vulture5.
                                    bmp"
                                };

FILE *fileTrainPatterns;
FILE *fileImages;
FILE *fileTestPatterns;

/*
* This function creates a file called training.dat
which holds all of the pixel values for each of the
* training patterns held in the
BitmapsToFeedToNeuralNet40x40 directory.
* It also creates a file called test.data which holds
all of the pixel values for the glyphs being tested
*/
void main()
{
    fileTrainPatterns = fopen(filenameTrain,"wb");
    /* Print to the file the number of training glyphs
    there are in the file */
    putc(NUMTRAIN,fileTrainPatterns);
    for (int var = 0; var < NUMTRAIN; ++var)
    {
        convertAndPrint(var);
    }
    fclose(fileTrainPatterns);
    printf("\n\n\n");
```

```
    fileTestPatterns = fopen(filenameTest,"wb");
    putc(NUMTEST,fileTestPatterns);
    for (int var = 0; var < NUMTEST; ++var)
    {
        convertTestAndPrint(var);
    }
    fclose(fileTestPatterns);
}

void convertTestAndPrint(int nr)
{
    int var, var2, byte1, byte2, byte3;
    printf("CONVERTING:%s --> %s\n", testPattFilenames[
           nr], filenameTrain);
    fileImages = fopen(trainPattFilenames[nr],"rb");

    /* Throw the header away */
    for (var = 0; var < headerSize; ++var)
    {
        getc(fileImages);
    }

    /* Print to the file what glyph the training
    pattern is */
    int nrOf = nr/2+1;
    putc(nrOf,fileTestPatterns);

    /* For each pixel in the image */
    for (var = 0; var < 1600; ++var)
    {
```

```
            byte1 = getc(fileImages);
            byte2 = getc(fileImages);
            byte3 = getc(fileImages);
            if (byte1 == 255 && byte2 == 255 && byte3 ==
                255)
            {
                putc(1,fileTestPatterns);
            }
            else if (byte1 == 0 && byte2 == 0 && byte3 == 0)
            {
                putc(0,fileTestPatterns);
            }
            else
            {
                // MAJOR ERROR
                printf("TERMINAL ERROR #4 SYSTEM WILL EXIT
                       NOW");
                exit(0);
            }
        }

    /* Print delimiter between glyphs */
    putc(-1,fileTestPatterns);
    fclose(fileImages);
}

void convertAndPrint(int nr)
{
    int var, var2, byte1, byte2, byte3;
    printf("CONVERTING:%s --> %s\n", trainPattFilenames[
```

```
            nr], filenameTrain);
    fileImages = fopen(trainPattFilenames[nr],"rb");

    /* Throw the header away */
    for (var = 0; var < headerSize; ++var)
    {
        getc(fileImages);
    }

    /* Print to the file what glyph the training
    pattern is */
    if (nr < instancesPerPattern)
    { // GOD
        putc(0,fileTrainPatterns);
    }
    else if (nr < instancesPerPattern*2)
    { // LION
        putc(1,fileTrainPatterns);
    }
    else if (nr < instancesPerPattern*3)
    { // RED FLOWER
        putc(2,fileTrainPatterns);
    }
    else if (nr < instancesPerPattern*4)
    { // SCARAB
        putc(3,fileTrainPatterns);
    }
    else if (nr < instancesPerPattern*5)
    { // VULTURE
        putc(4,fileTrainPatterns);
```

```
    }

    /* For each pixel in the image */
    for (var = 0; var < 1600; ++var)
    {
        byte1 = getc(fileImages);
        byte2 = getc(fileImages);
        byte3 = getc(fileImages);
        if (byte1 == 255 && byte2 == 255 && byte3 ==
            255)
        {
            putc(1,fileTrainPatterns);
        }
        else if (byte1 == 0 && byte2 == 0 && byte3 == 0)
        {
            putc(0,fileTrainPatterns);
        }
        else
        {
            // MAJOR ERROR
            printf("TERMINAL ERROR #4 SYSTEM WILL EXIT
                   NOW");
            exit(0);
        }
    }

    /* Print delimiter between glyphs */
    putc(-1,fileTrainPatterns);
    fclose(fileImages);
}
```

A.2 imageProcessing.c

```
/*
* This file is made to be able to make up a lot of
dummy data
* to throw into the neural network.
* This system makes it black-white and cuts the white
space out.
* Then I will throw this into photoshop to resize the
image.
* I did this to have something to fall back to if
planA falls to pieces.
*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "prototypes.h"

char filenameArray[15][31] = {
                                "Bitmaps/God.bmp",
                                "Bitmaps/GodTra.bmp",
                                "Bitmaps/God2.bmp",
                                "Bitmaps/Lion.bmp",
                                "Bitmaps/Lion1.bmp",
                                "Bitmaps/Lion2.bmp",
                                "Bitmaps/Red Flower.
                                bmp","Bitmaps/Red
```

```
                                    Flower1.bmp",
                                    "Bitmaps/Red Flower2.
                                    bmp",
                                    "Bitmaps/Scarab.bmp",
                                    "Bitmaps/Scarab1.bmp",
                                    "Bitmaps/Scarab2.bmp",
                                    "Bitmaps/Vulture.bmp",
                                    "Bitmaps/Vulture1.bmp",
                                    "Bitmaps/Vulture2.
                                    bmp"
                                };

char filenameBlackWhiteArray[15][41] = {
                                              "BitmapsBlack
                                              White/God.
                                              bmp",
                                              "BitmapsBlack
                                              White/GodTra.
                                              bmp",
                                              "BitmapsBlack
                                              White/God2.
                                              bmp",
                                              "BitmapsBlack
                                              White/Lion.
                                              bmp",
                                              "BitmapsBlack
                                              White/Lion1.
                                              bmp",
                                              "BitmapsBlack
                                              White/Lion2.
```

```
                bmp",
                "BitmapsBlack
                White/Red
                Flower.bmp",
                "BitmapsBlack
                White/Red
                Flower1.bmp",
                "BitmapsBlack
                White/Red
                Flower2.bmp",
                "BitmapsBlack
                White/Scarab.
                bmp",
                "BitmapsBlack
                White/Scarab1
                .bmp",
                "BitmapsBlack
                White/Scarab2
                .bmp",
                "BitmapsBlack
                White/Vulture
                .bmp",
                "BitmapsBlack
                White/Vulture
                1.bmp",
                "BitmapsBlack
                White/Vulture
                2.bmp"
        };

```

```
char filenameWithoutWhite1Array[15][44] = {
                "BitmapsWithoutWhite1/God.bmp",
                "BitmapsWithoutWhite1/GodTra.bmp",
                "BitmapsWithoutWhite1/God2.bmp",
                "BitmapsWithoutWhite1/Lion.bmp",
                "BitmapsWithoutWhite1/Lion1.bmp",
                "BitmapsWithoutWhite1/Lion2.bmp",
                "BitmapsWithoutWhite1/Red Flower.bmp",
                "BitmapsWithoutWhite1/Red Flower1.bmp",
                "BitmapsWithoutWhite1/Red Flower2.bmp",
                "BitmapsWithoutWhite1/Scarab.bmp",
                "BitmapsWithoutWhite1/Scarab1.bmp",
                "BitmapsWithoutWhite1/Scarab2.bmp",
                "BitmapsWithoutWhite1/Vulture.bmp",
                "BitmapsWithoutWhite1/Vulture1.bmp",
                "BitmapsWithoutWhite1/Vulture2.bmp"
        };

char filenameWithoutWhite2Array[15][44] = {
                "BitmapsWithoutWhite2/God.bmp",
                "BitmapsWithoutWhite2/GodTra.bmp",
                "BitmapsWithoutWhite2/God2.bmp",
                "BitmapsWithoutWhite2/Lion.bmp",
                "BitmapsWithoutWhite2/Lion1.bmp",
                "BitmapsWithoutWhite2/Lion2.bmp",
                "BitmapsWithoutWhite2/Red Flower.bmp",
                "BitmapsWithoutWhite2/Red Flower1.bmp",
                "BitmapsWithoutWhite2/Red Flower2.bmp",
                "BitmapsWithoutWhite2/Scarab.bmp",
                "BitmapsWithoutWhite2/Scarab1.bmp",
```

```
                "BitmapsWithoutWhite2/Scarab2.bmp",
                "BitmapsWithoutWhite2/Vulture.bmp",
                "BitmapsWithoutWhite2/Vulture1.bmp",
                "BitmapsWithoutWhite2/Vulture2.bmp"
        };

int rgbValue[3], widthValues[3], heightValues[3],
nrOfColorCodes;
int filesToCutDownVertically[120], nrOfGlyphs = 0;

FILE *fileToThreshold, *fileColor, *fileBlackWhite,
*fileToProcessWhite, *fileWithoutWhite;
int *dividersAtLines, *dividersAtColumns;
int byteSizeNeeded;

struct image
{
    int header[54];
    int headerBackup[54];
    int width;
    int height;
    double threshold;
    double threshold2;
};

struct image inputImage;

void dividerFinderHorizontal(int nr)
{
    // PRE PROCESSING OPERATIONS BEGIN
```

```
if (dividersAtColumns != NULL)
{
    /* If pointer referenced to a memory location
    free the pointer up */
    free((void *) dividersAtColumns);
}
fileToProcessWhite = fopen(
                     filenameWithoutWhite1Array[nr],
                     "rb");
byteSizeNeeded = inputImage.width*sizeof(int);
dividersAtColumns = (int *)malloc(byteSizeNeeded);
if (dividersAtColumns == NULL)
{
    /* If allocation unsuccessful, print message
    and exit.  */
    printf("TERMINAL ERROR #1 SYSTEM WILL EXIT
           NOW!\n");
    exit(0);
}
memset(dividersAtColumns,-1,byteSizeNeeded);
int wholeColumnWhite = inputImage.height*765;
int var, var2;
// PRE PROCESSING OPERATIONS END

/* Throw the header away */
for (var = 0; var < headerSize; ++var)
{
    getc(fileToProcessWhite);
}

```

```
/* calculate the number of junk bytes */
int nrOfJunkBytes = 4-((inputImage.width*3)%4);
if (nrOfJunkBytes == 4)
{
    nrOfJunkBytes = 0;
}

/* for each line in the image do */
for (var = 0; var < inputImage.height; ++var)
{
    /* for each column in the image look for black
    spots */
    for (var2 = 0; var2 < inputImage.width; ++var2)
    {
        int byte1 = getc(fileToProcessWhite);
        int byte2 = getc(fileToProcessWhite);
        int byte3 = getc(fileToProcessWhite);
        if (byte1 == 0)
        { // PixelBlack
            dividersAtColumns[var2] = 1;
        }
    }
    /* Take the junk bytes out */
    for (var2 = 0; var2 < nrOfJunkBytes; ++var2)
    {
        getc(fileToProcessWhite);
    }
}
```

```
    // POST PROCESSING OPERATIONS BEGIN
    fclose(fileToProcessWhite);
    // POST PROCESSING OPERATIONS END

    /* Uncomment for loop to see where the horizontal
    white space in the image is */
    /*  for (var = 0; var < inputImage.width; ++var) {
            printf("dividersAtColumns[%i]:%i\n", var,
                   dividersAtColumns[var]);
        }*/
}

void dividerFinderVertical(int nr)
{
    // PRE PROCESSING OPERATIONS BEGIN
    if (dividersAtLines != NULL)
    {
        /* If freeing of memory unsuccessfull, print
        message and exit.  */
        printf("TERMINAL ERROR #2 SYSTEM WILL EXIT
               NOW!\n");
        exit(0);
    }
    fileToProcessWhite = fopen(filenameBlackWhiteArray[
                              nr],"rb");
    byteSizeNeeded = inputImage.height*sizeof(int);
    dividersAtLines = (int *)malloc(byteSizeNeeded);
    if (dividersAtLines == NULL)
    {
        /* If allocation unsuccessful, print message
```

```
        and exit.  */
        printf("TERMINAL ERROR #1 SYSTEM WILL EXIT
               NOW!\n");
        exit(0);
    }
    int wholeLineWhite = inputImage.width*765;
    // PRE PROCESSING OPERATIONS END

    /*
     * Throw the header away
     */
    for (int var = 0; var < headerSize; ++var)
    {
        getc(fileToProcessWhite);
    }

    /* calculate the number of junk bytes */
    int nrOfJunkBytes = 4-((inputImage.width*3)%4);
    if (nrOfJunkBytes == 4)
    {
        nrOfJunkBytes = 0;
    }

    /*
     * For each line in the image do:
     */
    for (int var = 0; var < inputImage.height; ++var)
    {
        int sum = 0;
        /*
```

```
         * For each column in the image do:
         */
        for (int var2 = 0; var2 < inputImage.width; ++
             var2)
        {
            int byte1 = getc(fileToProcessWhite);
            int byte2 = getc(fileToProcessWhite);
            int byte3 = getc(fileToProcessWhite);
            sum += byte1;
            sum += byte2;
            sum += byte3;
        }
        if (sum == wholeLineWhite)
        {
            dividersAtLines[var] = -1;
        }
        else
        {
            dividersAtLines[var] = 1;
        }
        for (int var2 = 0; var2 < nrOfJunkBytes; ++var2)
        {
            getc(fileToProcessWhite);
        }
    }

    // POST PROCESSING OPERATIONS BEGIN
    fclose(fileToProcessWhite);
    // POST PROCESSING OPERATIONS END

```

```
    /* Uncomment for loop to see where the vertical
    white space in the image is */
    /*  for (int var = 0; var < inputImage.height; ++
             var) {
            printf("dividersAtLines[%i]:%i\n", var,
                   dividersAtLines[var]);
        }*/
}

void downCutterHorizontal(int nr)
{
    // PRE PROCESSING OPERATIONS BEGIN
    fileToProcessWhite = fopen(
                         filenameWithoutWhite1Array[nr],
                         "rb");
    fileWithoutWhite = fopen(filenameWithoutWhite2Array[
                       nr],"wb");
    int var, var2, var3;
    // PRE PROCESSING OPERATIONS END

    printf("   ---> %s\n", filenameWithoutWhite2Array[
           nr]);

    /* Save the header of the file */
    for (var = 0; var < headerSize; ++var)
    {
        inputImage.header[var] = getc(
                                 fileToProcessWhite);
    }

```

```
    /* Count nr of white columns */
    int nrOfWhiteColumns = 0;
    for (var = 0; var < inputImage.width; ++var)
    {
        if (dividersAtColumns[var] == -1)
        {
            nrOfWhiteColumns++;
        }
    }
    /* Change the header */
    changeWidthHeight(nrOfWhiteColumns,-1,-1);

    /* Print the new header to the file */
    for (var = 0; var < headerSize; ++var)
    {
        putc(inputImage.header[var],fileWithoutWhite);
    }

    int newWidth = inputImage.width - nrOfWhiteColumns;
    /* Calculate the number of "junk bytes"
     * 4-((pixelsPerColumn*3)%4) = nr of junk bytes
     */
    int nrOfJunkBytes = 4-((newWidth*3)%4);
    if (nrOfJunkBytes == 4)
    {
        nrOfJunkBytes = 0;
    }
    int nrOfJunkBytesToTake = 4-((inputImage.width*3)%4)
                                          ;
    if (nrOfJunkBytesToTake == 4)
```

```
{
    nrOfJunkBytesToTake = 0;
}

/* Print to the file the columns that aren't white
space.*/
/* for each line */
for (var = 0; var < inputImage.height; ++var)
{
    /* for each column */
    for (var2 = 0; var2 < inputImage.width; ++var2)
    {
        /* If pixel not in a white space column
        then print that pixel to a file */
        if (dividersAtColumns[var2] == 1)
        {
            putc(getc(fileToProcessWhite),
                 fileWithoutWhite);
            putc(getc(fileToProcessWhite),
                 fileWithoutWhite);
            putc(getc(fileToProcessWhite),
                 fileWithoutWhite);
        }
        /* If part of a white space column throw
        the pixel away */
        else
        {
            getc(fileToProcessWhite);
            getc(fileToProcessWhite);
            getc(fileToProcessWhite);
```

```
            }
        }
        for (var2 = 0; var2 < nrOfJunkBytesToTake; ++
             var2)
        {
            getc(fileToProcessWhite);
        }
        for (var2 = 0; var2 < nrOfJunkBytes; ++var2)
        {
            putc(0,fileWithoutWhite);
        }
    }

    inputImage.width = newWidth;

    // POST PROCESSING OPERATIONS BEGIN
    fclose(fileToProcessWhite);
    fclose(fileWithoutWhite);
    free((void *)dividersAtColumns);
    // POST PROCESSING OPERATIONS END
}

void downCutterVertical(int nr)
{
    // PRE PROCESSING OPERATIONS BEGIN
    fileWithoutWhite = fopen(filenameWithoutWhite1Array[
                       nr], "wb");
    fileToProcessWhite = fopen(filenameBlackWhiteArray[
                         nr], "rb");
    int var, var2;
```

```
// PRE PROCESSING OPERATIONS END

printf("  ---> %s\n", filenameWithoutWhite1Array[nr]
        );

/*
 * Now I have to take out the header.
 * change the header
 * putc header
 * getc putc all of the columns that are 1
 */

/* Save the header of the file */
for (var = 0; var < headerSize; ++var)
{
    inputImage.header[var] = getc(
                                  fileToProcessWhite);
}
/* Count nr of white lines */
int nrOfWhiteLines = 0;
for (var = 0; var < inputImage.height; ++var)
{
    if (dividersAtLines[var] == -1)
    {
        nrOfWhiteLines++;
    }
}
/* change the header */
changeWidthHeight(nrOfWhiteLines,-1,1);

```

```c
/* Calculate the number of "junk bytes"
 * 4-((pixelsPerColumn*3)%4) = nr of junk bytes
 */
int nrOfJunkBytes = 4-((inputImage.width*3)%4);
if (nrOfJunkBytes == 4)
{
    nrOfJunkBytes = 0;
}

/* Put the new header in to the new file */
for (var = 0; var < headerSize; ++var)
{
    putc(inputImage.header[var],fileWithoutWhite);
}

/* Put the "filtered" columns in */
/* For each of the lines in the image */
for (var = 0; var < inputImage.height; ++var)
{
    /* For all of the columns in the image either
    */
    for (var2 = 0; var2 < inputImage.width; ++var2)
    {
        /* putc them if they are not whitespace */
        if (dividersAtLines[var] == 1)
        {
            putc(getc(fileToProcessWhite),
                 fileWithoutWhite);
            putc(getc(fileToProcessWhite),
                 fileWithoutWhite);
```

```
            putc(getc(fileToProcessWhite),
                 fileWithoutWhite);
        }
        /* or throw them away if they are
        whitespace */
        else
        {
            getc(fileToProcessWhite);
            getc(fileToProcessWhite);
            getc(fileToProcessWhite);
        }
    }
    for (var2 = 0; var2 < nrOfJunkBytes; ++var2)
    {
        getc(fileToProcessWhite);
    }
    if (dividersAtLines[var] == 1)
    {
        for (var2 = 0; var2 < nrOfJunkBytes; ++var2)
        {
            putc(0,fileWithoutWhite);
        }
    }
}
inputImage.height -= nrOfWhiteLines;

// POST PROCESSING OPERATIONS BEGIN
free((void *)dividersAtLines);
fclose(fileToProcessWhite);
fclose(fileWithoutWhite);
```

```
    // POST PROCESSING OPERATIONS END
}

void identifyGlyph(int nr)
{
    dividerFinderVertical(nr);
    downCutterVertical(nr);
    dividerFinderHorizontal(nr);
    downCutterHorizontal(nr);
}

int calculatingHeightAndWidth(int arrayToConvert[])
{
    int total = 0;
    int individualValues[6];

    individualValues[5] = arrayToConvert[0] % 16;
    individualValues[4] = (arrayToConvert[0] - (
                          arrayToConvert[0] % 16)) / 16;
    individualValues[2] = arrayToConvert[1] % 16;
    individualValues[3] = (arrayToConvert[1] - (
                          arrayToConvert[1] % 16)) / 16;
    individualValues[0] = arrayToConvert[2] % 16;
    individualValues[1] = (arrayToConvert[2] - (
                          arrayToConvert[2] % 16)) / 16;

    for (int k = 0; k < 6; ++k)
    {
        int powerFunc = pow(16, k);
        total += individualValues[k] * powerFunc;
```

```
    }
    return total;
}

void resetHeader()
{
    for (int var = 0; var < headerSize; ++var)
    {
        inputImage.header[var] = inputImage.
                                        headerBackup[var];
    }
}

void makeBlackWhite(int nr)
{
    // PRE PROCESSING OPERATIONS BEGIN
    fileColor = fopen(filenameArray[nr],"rb");
    if (fileColor == NULL)
    {
        // If file open wasn't successfull print error
        & exit
        printf("TERMINAL ERROR #3 SYSTEM WILL EXIT
               NOW!\n");
        exit(0);
    }
    fileBlackWhite = fopen(filenameBlackWhiteArray[nr],
                     "wb");
    fileToThreshold = fopen(filenameArray[nr], "rb");
    double averageWholePicture = 0;
    // PRE PROCESSING OPERATIONS END
```

```

printf(" ---> %s\n", filenameBlackWhiteArray[nr]);

/*
* Start by reading the header into a header array
* and then print that out straight to the output
file.
* - This is done to read all the necessary values
from the header of the file.(i.e.  width&height)
*/
for (int stillHeader = 0; stillHeader < headerSize;
     ++stillHeader)
{
    int dataFromFile = getc(fileColor); // This
                            just gets the next integter
                            and saves it in a integer
                            variable.
    inputImage.header[stillHeader] = dataFromFile;
    inputImage.headerBackup[stillHeader] = dataFromF
                                               ile;
    putc(dataFromFile,fileBlackWhite);

    switch (stillHeader)
    {
    case 18:
        widthValues[2] = dataFromFile;
        break;
    case 19:
        widthValues[1] = dataFromFile;
        break;
```

```
        case 20:
            widthValues[0] = dataFromFile;
            break;
        case 22:
            heightValues[2] = dataFromFile;
            break;
        case 23:
            heightValues[1] = dataFromFile;
            break;
        case 24:
            heightValues[0] = dataFromFile;
            break;
        }
    }

    inputImage.width = calculatingHeightAndWidth(
                        widthValues);
    inputImage.height = calculatingHeightAndWidth(
                        heightValues);
    /* calculate the number of junk bytes */
    int nrOfJunkBytes = 4-((inputImage.width*3)%4);
    if (nrOfJunkBytes == 4)
    {
        nrOfJunkBytes = 0;
    }
    nrOfColorCodes = inputImage.width*inputImage.height;

    for (int var = 0; var < nrOfColorCodes; var++)
    {
        int value1 = getc(fileToThreshold);
```

```
        int value2 = getc(fileToThreshold);
        int value3 = getc(fileToThreshold);
        if (var % inputImage.width == 0 &&
            nrOfJunkBytes != 0 && var != 0)
        {
            for (int var2 = 0; var2 < nrOfJunkBytes; ++
                 var2)
            {
                getc(fileToThreshold);
            }
        }
        double value = (value1+value2+value3)/3;
        averageWholePicture += value;
        if (value > inputImage.threshold)
        {
            inputImage.threshold = value;
        }
    }

    inputImage.threshold2 = averageWholePicture/(
                            inputImage.
                            height*inputImage.width);

    /* Take in all the color codes and convert them to
    actual black white values */
    for (int variable = 0; variable < nrOfColorCodes;
         variable++)
    {
        if (variable % inputImage.width == 0 &&
            nrOfJunkBytes != 0 && variable != 0)
```

```
{
    for (int var2 = 0; var2 < nrOfJunkBytes; ++
         var2)
    {
        int byte1 = getc(fileColor);
        putc(0,fileBlackWhite);
    }
}
rgbValue[0] = getc(fileColor);
rgbValue[1] = getc(fileColor);
rgbValue[2] = getc(fileColor);

if (variable % inputImage.width == 0 &&
    nrOfJunkBytes != 0)
{
    for (int var2 = 0; var2 < nrOfJunkBytes; ++
         var2)
    {
        getc(fileToThreshold);
    }
}

//int LightOrDark = pixelLightOrDark(rgbValue,
                    inputImage.threshold/2.3);
int LightOrDark = pixelLightOrDark(rgbValue,
                  inputImage.threshold2/1.2);
                  // TOO MUCH LIGHT !
if (LightOrDark == 0)
{
    rgbValue[0] = 0;
```

```
                rgbValue[1] = 0;
                rgbValue[2] = 0;
            }
            else if (LightOrDark == 1)
            {
                rgbValue[0] = 255;
                rgbValue[1] = 255;
                rgbValue[2] = 255;
            }
            putc(rgbValue[0],fileBlackWhite);
            putc(rgbValue[1],fileBlackWhite);
            putc(rgbValue[2],fileBlackWhite);
        }
        fclose(fileColor);
        fclose(fileBlackWhite);
        fclose(fileToThreshold);
}

int pixelLightOrDark(int pixelData[], double threshold)
{
        /*
         * This change in this function is only to "whiten"
         out all the pixels that aren't black from the
         tracing
         */
        if (pixelData[0] == 0 && pixelData[1] == 0 &&
            pixelData[2] == 0)
        {
            return 0;
        }
```

```
    return 1;
    /* UNCOMMENT THIS ONE IF YOU ARE NOT USING THE
    TRACING METHOD */
    /*  int meanValue = (pixelData[0] + pixelData[1] +
                        pixelData[2])/3;
        if(meanValue < threshold){
            return 0;
        }
        return 1;*/
}

void changeWidthHeight(int nrOfColumns, int
                       modifyOrChange, int
                       widthOrHeight)
{
    int param;
    if (modifyOrChange == -1)
    {
        if (widthOrHeight == -1)
        { // WIDTH
            param = inputImage.width - nrOfColumns;
        }
        else if (widthOrHeight == 1)
        { // HEIGHT
            param = inputImage.height - nrOfColumns;
        }
    }
    else
    {
        if (widthOrHeight == -1)
```

```
        { // WIDTH
            param = inputImage.width - nrOfColumns;
        }
        else if (widthOrHeight == 1)
        { // HEIGHT
            param = inputImage.height - nrOfColumns;
        }
    }
    int individualValues[6];

    int powTwo = pow(16,2);
    int powThree = pow(16,3);
    int powFour = pow(16,4);
    int powFive = pow(16,5);

    individualValues[0] = param / powFive;
    individualValues[1] = (param-individualValues[0]
                            *powFive) / powFour;
    individualValues[2] = (param-individualValues[1]
                            *powFour) / powThree;
    individualValues[3] = (param-individualValues[2]
                            *powThree) / powTwo;
    individualValues[4] = (param-individualValues[3]
                            *powTwo) / 16;
    individualValues[5] = param % 16;

    int firstValue = individualValues[0]*16 +
                     individualValues[1];
    int secondValue = individualValues[2]*16 +
                      individualValues[3];
```

```
    int thirdValue = individualValues[4]*16 +
                     individualValues[5];

    if (widthOrHeight == -1)
    { // WIDTH
        inputImage.header[18] = thirdValue;
        inputImage.header[19] = secondValue;
        inputImage.header[20] = firstValue;
        inputImage.header[21] = 0;
    }
    else if (widthOrHeight == 1)
    { // HEIGHT
        inputImage.header[22] = thirdValue;
        inputImage.header[23] = secondValue;
        inputImage.header[24] = firstValue;
        inputImage.header[25] = 0;
    }
}

void main()
{
    int var = 0;
    printf("%s\n", filenameArray[var]);
    makeBlackWhite(var);
    identifyGlyph(var);

} // END OF main()
```

A.3 neuralNetwork.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "neuralNetworkDefines.h"

// Weights between input neurons & hidden neurons
float m1[IN][HIDDEN];
// Weights between hidden neurons & output neurons
float m2[HIDDEN][OUT];
// Delta between input neurons & output neurons
float deltaM1[IN][HIDDEN];
// Delta between hidden neurons & output neurons
float deltaM2[HIDDEN][OUT];
// Stores the input from the user
float x[IN];
// Stores the hidden activations
float y[HIDDEN];
// Stores the output activations to print out in the
end
float z[OUT];

float errorHidden[HIDDEN];
float errorOutput[OUT];
int patr[NUMTRAIN];
float ecm[NUMTRAIN];
float betaH=0.01;      /* learning rate */
float alpha=0.01;      /* momentum */
long int itr;
```

```
int inputFromUser[IN];

/*
* CALLS TO: NONE
* INPUT: NONE
* OUTPUT: Resets the m1,m2,deltaM1,deltaM2,patr,
inputFromUser matrices
*/
void initializer()
{
    int i,j;
    int ch;
    int num;

    /*
     * Assign Random Weights to the m1 and m2 array
     * "It is possible to start with randomly chosen
     values for the weights
     * and to let the weights be adjusted appropriately
     as the network is run through successive
     iterations.
     * This would make it easier also.
     * For example, under supervised training, if the
     error between the desired and computed output is
     used
     * as a criterion in adjusting weights, then one
     may as well set the initial weights to zero and
     let
     * the training process take care of the rest."
     * (file:///E:/Artificial%20Intelligence/C++
```

```
        _Neural_Networks_and_Fuzzy_Logic/ch05/093-096.
        html#Heading16)
     * Reset the deltaM1 and deltaM2 array
     */
    for (i=0;i<IN;i++)
        for (j=0;j<HIDDEN;j++)
        {
            m1[i][j] = -0.5 + (float) rand()/(double)
                        RAND_MAX;
            deltaM1[i][j] = 0;
        }
    for (i=0;i<HIDDEN;i++)
        for (j=0;j<OUT;j++)
        {
            m2[i][j] = -0.5 + (float) rand()/(double)
                        RAND_MAX;
            deltaM2[i][j] = 0;
        }

    /*
     * Reset the patr array
     */
    for (i=0;i<NUMTRAIN;i++)
        patr[i] = 0;

    /*
     * Reset the inputFromUser array
     */
    for (i=0;i<IN;i++)
        inputFromUser[i]=0;
```

```
}

/*
* This is the training session
* Drives the training of the system by calling the
answerFromNet and the backpropagation.
* CALLS TO: answerFromNet -> Recursively calls 3 times
*           backpropagation -> Recursively calls 3
times
*
* INPUT: NONE
* OUTPUT: Modifies the patr matrix.
*
* HAVE TO WORK ON THIS FUNCTION SOME MORE !
*/
void training()
{
    int i,l,num;
    long int j;
    int t;
    float p;
    int ch;
    i=0;
    j=0;
    num=0;
    do
    {
        do
        {
            /*
```

```
             * select a random training pattern:
             * i = (int)(NUMTRAIN*rnd), where 0<rnd<1
             */
            i = (int)(NUMTRAIN*(float) rand() /
                RAND_MAX);
        }
        while (patr[i]);

        /*
         * It is taking a pattern in 3 times in a row
         * And throwing that into the backpropagation
         algorithm
         * to modify the weights.
         */
        for (int rep=0;rep<3;rep++)
        {
            j++;
            answerFromNet(trainingPatterns[i]);
            backpropagation(i);
        }

        /*
         * Prints out every 102 argument with j
         */
        if (!(j%102))
            printf("\n%ld",j);
        error();
        l = 1;
        for (t=0;t<NUMTRAIN;t++)
        {
```

```
                patr[t] = ecm[t] < EPSILON;
                l = l && (patr[t]);
            }
        }
    while (!l);

    printf("\n\nEnd of training\n");
}

/*
* This function takes in a pattern to calculate and
check what the values
* are in for each of the patterns.
* What it does:
* Takes in a pattern
* Stores that pattern in the x matrix
* Runs that pattern first through the input-hidden
layers.          -> x-y
* Sends y to the sigmoid function
* Then runs the sigmoid-ed values to the hidden-output
layers.      -> y-z
* Sends z to the sigmoid function
*
* CALLS TO: sigm function
* INPUT: A pattern to calculate the errors in.
* OUTPUT: Modifies the x and the y matrix
*/
void answerFromNet(int patternToAnswer[])
{
    int i,j;
```

```
float totin;

for (i=0;i<IN;i++)
    x[i] = (float)patternToAnswer[i];

for (j=0;j<HIDDEN;j++)
{
    totin = 0;
    for (i=0;i<IN;i++)
    {
        /*
         * Sum all of the inputs multiplied by the
         weights
         * between the input and hidden layers.
         * Save that in the totin variable.
         */
        totin = totin + x[i]*m1[i][j];
    }
    /*
     * And then put that into the sigmoid function.
     * And store that in the y array.
     */
    y[j] = sigm(totin);
}

for (j=0;j<OUT;j++)
{
    totin = 0;
    for (i=0;i<HIDDEN;i++)
        /*
```

```
                 * Sum all of the inputs multiplied by the
                 weights
                 * between the hidden and the output layers.
                 * Save that in the totin variable.
                 */
                totin = totin + y[i]*m2[i][j];
            /*
             * And then put that into the sigmoid function.
             * And store that in the z array.
             */
            z[j] = sigm(totin);
        }
}

/*
* ith component of output error at the output layer
* CALLS TO: NONE
* INPUT: integer concernig which component to
calculate
* OUTPUT: Modifies the errorOutput matrix with the
errors in that output.
*/
void betaErrorOutput(int i)
{
    int j;
    /*
     * Reset the errorOutput array
     */
    for (j=0;j<OUT;j++)
        errorOutput[j] = 0;
```

```

    /*
     * Calculates the error for the Output layer by
     calculating
     * realOutput - desiredOutput
     */
    for (j=0;j<OUT;j++)
        errorOutput[j] = z[j] - (float)p[i][j];
}

/*
* ith component of output error at the hidden layer
* CALLS TO: NONE
* INPUT: NONE
* OUTPUT: Modifies the errorHidden matrix with the
erros in that hidden
*/
void betaErrorHidden()
{
    int i,j;
    /*
     * Reset the errorHidden array
     */
    for (i=0;i<HIDDEN;i++)
        errorHidden[i] = 0;

    /*
     * Calculates the error for the Hidden layer by
     calculating
     */
```

```
    for (i=0;i<HIDDEN;i++)
        for (j=0;j<OUT;j++)
            errorHidden[i] = errorHidden[i] + m2[i][j]
                                *z[j]*(1-z[j])*errorOutput[
                                j];
}

/*
* This is the training algorithm.
* It is adjusting the weights between the input and
hidden layers
* and the hidden and the output layers.
* CALLS TO: betaErrorOutput(k)
*           betaErrorHidden()
* INPUT: integer concerning which component to learn
* OUTPUT: modifies the weight matrices
*/
void backpropagation(int k)
{
    int i,j;
    float temp;

    betaErrorOutput(k);
    betaErrorHidden();

    /*
     * Adjustment for weight between ith neuron in
     hidden layer and jth output neuron
     * With momentum
     */
```

```
    for (i=0;i<HIDDEN;i++)
        for (j=0;j<OUT;j++)
        {
            temp = -betaH*y[i]*z[j]*(1-z[j])
                    *errorOutput[j];
            m2[i][j] = m2[i][j] + temp + alpha*deltaM2[
                        i][j];
            deltaM2[i][j] = temp;
        }

    /*
     * Adjustment for weight between ith input neuron
     and jth neuron in hidden layer
     * With momentum
     */
    for (i=0;i<IN;i++)
        for (j=0;j<HIDDEN;j++)
        {
            temp = -betaH*x[i]*y[j]*(1-y[j])
                    *errorHidden[j];
            m1[i][j] = m1[i][j] + temp + alpha*deltaM1[
                        i][j];
            deltaM1[i][j] = temp;
        }
}

/*
* This function throws all of the training patterns
into the answerFromNet function
* and calculates the error for all of them.
```

```c
* CALLS TO: errorMeasure with p(GLOBAL), z(GLOBAL) and
                                 the OUT variable.
* INPUT: NONE
* OUTPUT: Modifes the ecm matrix with values from the
errorMeasure for all of the training patterns.
*/
void error()
{
    for (int i=0;i<NUMTRAIN;i++)
    {
        answerFromNet(trainingPatterns[i]);
        ecm[i]=errorMeasure(p[i],z,OUT);
    }
}

/*
* CALLS TO: NONE
* INPUT: pParameter which is the desiredOutputPattern
*        zParameter which is the outputs of neurons in
the output layer
*        OUT variable
* OUTPUT: e which is the appropriate error.
*/
float errorMeasure(int pParameter[],float zParameter[],
                   int nrOfOutputsParameter)
{
    int i;
    float e=0;

    for (i=0;i<nrOfOutputsParameter;i++)
```

```
        e = e + ((float)pParameter[i] - zParameter[i])*(
            pParameter[i] - zParameter[i]);
    e = 0.5 * e;
    return e;
}

/*
* This function takes a pattern in from the test.dat
file and runs that through
* the answerFromNet to try and recognize the pattern.
* CALLS TO: answerFromNet
* INPUT: A pattern from the test.dat file
* OUTPUT: A print out with all the values in the z
matrix.
*/
void test()
{
    int i,j, numberOfPatts, nrOfGlyph;
    FILE *test = fopen("test.dat","rb");

    printf("Number of patterns:%i\n", numberOfPatts =
           getc(test));

    for (int var = 0; var < numberOfPatts; ++var)
    {
        printf("Number of Glyph:%i\n", nrOfGlyph = getc(
               test));
        for (i=0;i<IN;i++)
        {
            inputFromUser[i] = getc(test);
```

```
        }
        printf("\n\n Output activations :\n");
        answerFromNet(inputFromUser);
        for (i=0;i<OUT;i++)
            printf("z[%d] = %f\n",i,z[i]);

        /* TAKE OUT THE DELIMITER */
        int del = getc(test);
        if (del != 255)
        {
            printf("MAJOR ERROR:%i", del);
        }
    }
}

/*
* This function handles all the train pattern
creations.
*/
void preprocessor(void)
{
    FILE *training = fopen("training.dat","rb");
    if (training == NULL)
    { // trainingPatternCreator hasn't been run
        printf("TERMINAL ERROR #4 SYSTEM WILL EXIT NOW
               !");
        exit(0);
    }
    int numtrains = getc(training);
    if (numtrains != NUMTRAIN)
```

```
{ // The code hasn't been modified correctly
    printf("TERMINAL ERROR #5 SYSTEM WILL EXIT NOW
           !");
    exit(0);
}
int nrOfGlyph, var, var2, var3;

for (var = 0; var < NUMTRAIN; ++var)
{
    nrOfGlyph = getc(training);
    p[var][nrOfGlyph] = 1;
    for (var2 = 0; var2 < IN; ++var2)
    {
        trainingPatterns[var][var2] = getc(training)
                                                  ;
        if (trainingPatterns[var][var2] != 0 &&
            trainingPatterns[var][var2] != 1)
        {
            printf("%i\n", trainingPatterns[var][
                   var2]);
            printf("TERMINAL ERROR #6 SYSTEM WILL
                   EXIT NOW !\n");
            exit(0);
        }
    }
    /* Delimiter */
    if (getc(training) != 255)
    {
        // MAJOR ERROR EXIT
        printf("TERMINAL ERROR #6 SYSTEM WILL EXIT
```

```
                     NOW !");
              exit(0);
          }
      }
}

/*
* main method
* CALLS TO: preprocessor()
*           initializer()
*           training()
*           test()
* INPUT: Arguments
* OUTPUT:Runs the whole system
*/
void main(int argc,char *argv[])
{
    preprocessor();
    initializer();
    training();
    test();
} // END OF MAIN()
```

A.4 prototypes.h

```
void identifyGlyph(int nr);
void makeBlackWhite(int nr);
void resetHeader(void);
void changeWidthHeight(int nrOfColumns, int
```

```
                             modifyOrChange, int
                             widthOrHeight);
void dividerFinderVertical(int nr);
void downCutterVertical(int nr);
void dividerFinderHorizontal(int nr);
void downCutterHorizontal(int nr);

#define headerSize 54     /* number of bytes in the
header */

```

A.5 neuralNetworkDefines.h

```
#define sigm(x)      1/(1 + exp(-(double)x))
#define dxsigm(y)    (float)(y)*(1.0-y))
#define IN           1600  /* number if inputs */
#define HIDDEN       800   /* number of hidden units */
#define OUT          5     /* number of outputs */
#define EPSILON      0.001 /* maximum Mean Square Error
to stop training */
#define NUMTRAIN     25    /* number of training patterns
*/
#define LINES        40    /* number of lines in the
input from the user */
#define COLUMNS      40    /* number of columns in the
input from the user */

void initializer();
void training();
```

```
void answerFromNet(int afer[]);
float errorMeasure(int x[],float y[],int SIZE);
void backpropagation(int k);
void error();
void preprocessor(void);

/* training patterns */
int     trainingPatterns[NUMTRAIN][IN];

/* desired outputs */
int     p[NUMTRAIN][OUT];
```

Appendix B

Project Plan

B.1 View of Project

These are the major tasks that have to be taken.

B.2 Reading

Since I am a student in computer science that hasnŠt taken any course on either C-Programming, Machine Learning nor Artificial Intelligence I had to read up on all of those things.
I had to read something about all of those tasks:

- C-Programming
- Neural Networks
- Harvard Reference System
- Character Recognition System
- Backpropagation Algorithm

B.3 Design

I had to try and find some good ways to do the programming phase of this system easier. Also I tried to divide the task into 5 smaller tasks and tried to find ways to implement those steps in code.

B.4 Programming

I had to program the whole thing that was designed. Come by unforseen problems and work this system out so it functioned correctly.

B.5 Deliverables

These are things that I had to deliver.

- Interim Report
- Final Year Dissertation
- Presentation
- Demonstration

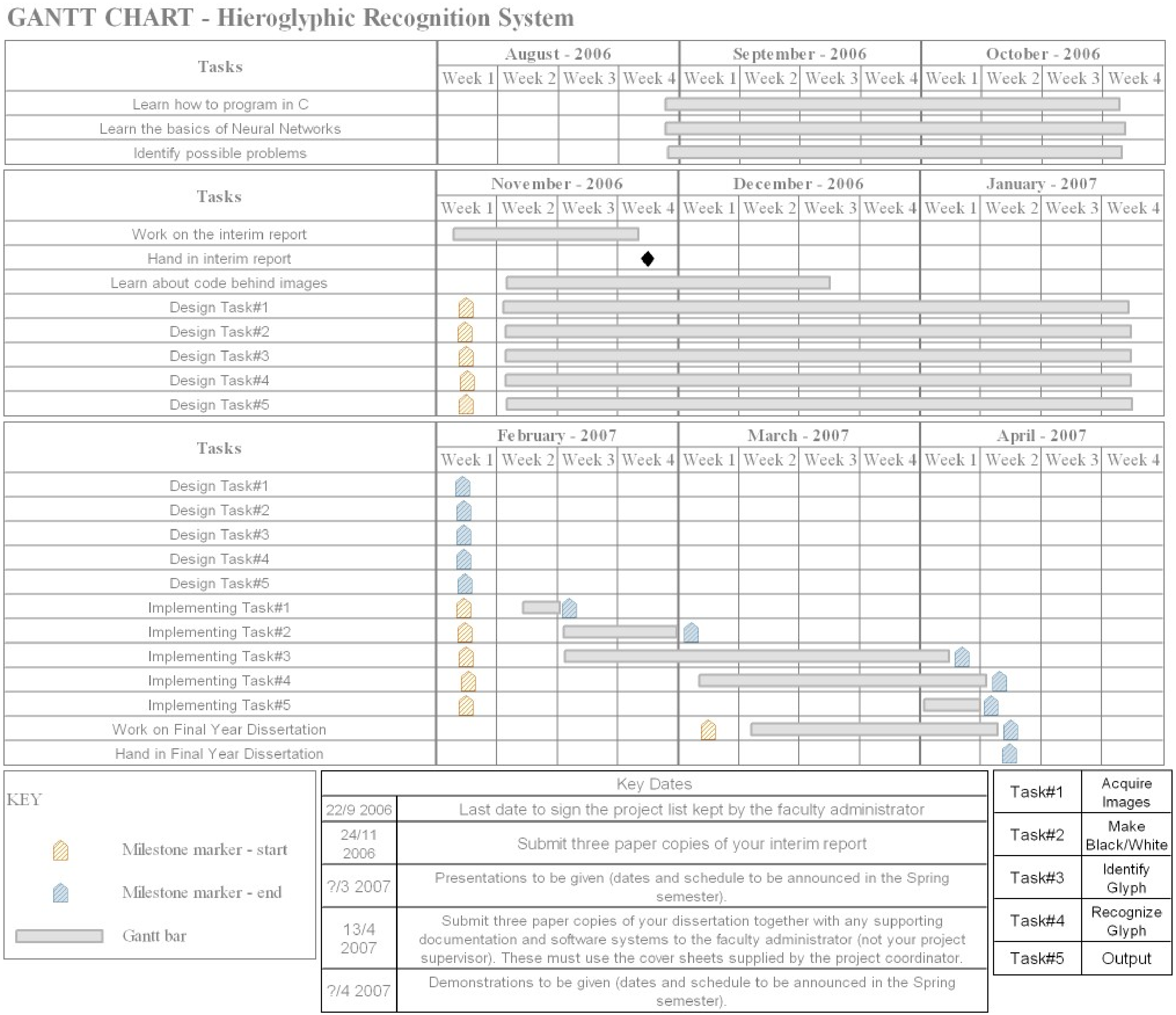


Figure B.1: Timeline of the project - Gantt Chart

Appendix C

User Manual

This is a user manual for the glyph recognition system.

C.1 How do I recognize ?

To recognize a glyph from an image you have to follow these steps:

Have the directories as it is on the CD somewhere on your personal computer.

Trace your image with black color.

Have your image saved as follows “Bitmaps/filename.bmp”.

The image has to be in 24-bit bitmap file format.

Then you have to change the imageProcessing.c file.

Change the filenameArray so the first index of the array contains your filename. In this format “Bitmaps/filename.bmp".

Change the filenameBlackWhiteArray so the first index of the array contains your filename. In this format “BitmapsBlackWhite/filename.bmp"

Change the filenameWithoutWhiteArray1 so the first index of the array contains your filename. In this format “BitmapsWithoutWhite1/filename.bmp"

Change the filenameWithoutWhiteArray2 so the first index of the array contains your filename. In this format BitmapsWithoutWhite2/filename.bmp

Run the imageProcessing system.

Then you should see in the filenameWithoutWhiteArray2 folder an image without whitespace and black-white.

You have to resize the image preferrably with the nearest neighbor algorithm. And save the resized image in the format “BitmapsToFeedToNeuralNet40x40/filename.bmp".

Then you have to change the trainingPatternCreator.c file.

Change the NUMTEST to 1 if you have 1 test sample.(I am assuming you are only working with the God, Lion, Scarab, Red Flower or Vulture glyphs because of simplicity for the user).

Then you have to change the testPattFilenames array to consist only of your image. In this format “BitmapsToFeedToNeuralNet40x40/filename.bmp”

Then you have to run the trainingPatternCreator.

Then you have to run the neuralNetwork.

C.2 Errors

You may sometimes experience errors. Here is a complete list of errors you might experience.

C.2.1 TERMINAL ERROR #1

If you experience this error then you haven’t got enough memory to run this program. I suggest you free up some memory, by for example closing some applications.

C.2.2 TERMINAL ERROR #2

If you experience this problem then there is some problem with your memory. I suggest you take a look at your memory chips or your operating system is playing some tricks on you.

C.2.3 TERMINAL ERROR #3

The filename you input into the array is unproperly formatted. I suggest you take a better look at the instructions.

C.2.4 TERMINAL ERROR #4

If you experience this error then you haven't got the training.dat file on the correct spot. I suggest you read the instructions better.

C.2.5 TERMINAL ERROR #5

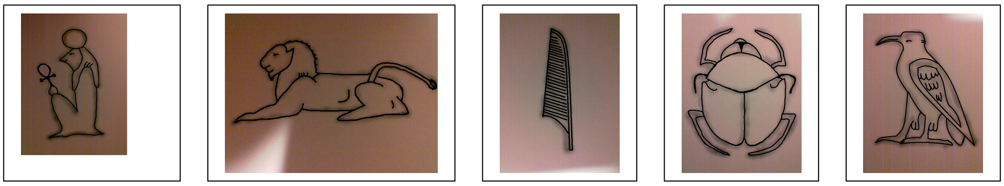
If you experience this error you haven't changed the code correctly.
I suggest you read the instructions better.

C.2.6 TERMINAL ERROR #6

If you experience this error then the code is bugged. I suggest you send an e-mail to jonorri333@gmail.com and complain about this. But there is absolutely nothing you can do about this error.

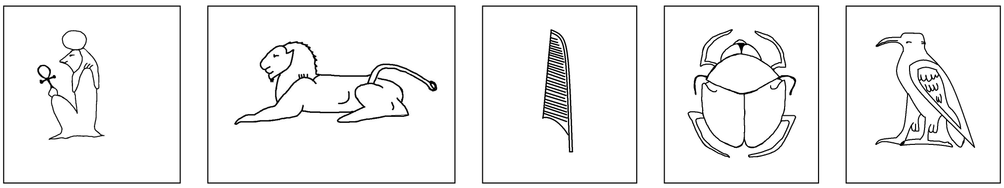
Appendix D

Input-Output



(a) God (b) Lion (c) Red Flower (d) Scarab (e) Vulture

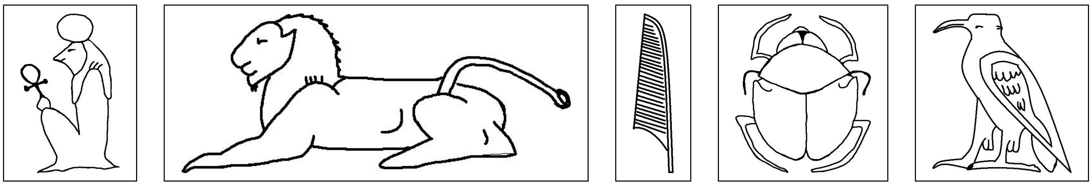
Figure D.1: Color Images



(a) God (b) Lion (c) Red Flower (d) Scarab (e) Vulture

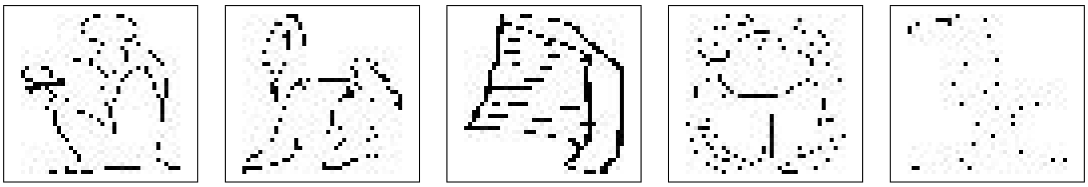
Figure D.2: Black-White Images

You can see how the neural network recognizes the images above in the next chapter(Sample Run).



(a) God (b) Lion (c) Red Flower (d) Scarab (e) Vulture

Figure D.3: No White-Space Images



(a) God (b) Lion (c) Red Flower (d) Scarab (e) Vulture

Figure D.4: Images After Resizing

Appendix E

Sample Run

102

204

306

408

510

612

714

816

918

1020

1122

1224

End of training

Number of patterns:5

Number of Glyph:0

Output activations :

z[0] = 0.972659

z[1] = 0.010870

z[2] = 0.012608

z[3] = 0.022152

z[4] = 0.020225

Number of Glyph:1

Output activations :

z[0] = 0.978165

z[1] = 0.004634

z[2] = 0.024988

z[3] = 0.001209

z[4] = 0.026680

Number of Glyph:2

Output activations :

z[0] = 0.963924

z[1] = 0.005914

z[2] = 0.010420

z[3] = 0.008939

z[4] = 0.010761

Number of Glyph:3

Output activations :

z[0] = 0.969534

z[1] = 0.007210

z[2] = 0.000637

z[3] = 0.003261

z[4] = 0.029097

Number of Glyph:4

Output activations :

z[0] = 0.000956

z[1] = 0.971745

z[2] = 0.010013

z[3] = 0.016399

z[4] = 0.024716

Appendix F

The System I Modified - bkProp.c

```
/*******************************************************
*****
;   Backpropagation with momentum
*
;   by Andres Perez-Uribe
*
;   Universidad del Valle, Cali, Colombia
*
;   sep/93
*
;
*
;   Email :  aperez@lslsun.epfl.ch
*
;           Logic Systems Laboratory
*
;           Swiss Federal Institute of Technology-
Lausanne  *
;           http://lslwww.epfl.ch/~aperez/
```

```
*
;*******************************************************
*****

   References :
   -  G. Hinton, "How neural networks learn from
   experience",
      Scientific American, sep 1992.
   -  P. Werbos,  "The Roots of Backpropagation:  From
   ordered derivatives
      to Neural Neworks and Political Forecasting",
      John Wiley and Sons,
      New York, 1994

   Compile :  gcc -o Bkprop Bkprop.c -lm
   Run     :  see example at the end of the C code.

    There is no guarantee that the code will do what
    you
    expect or that it is error free.  It is simply meant
    to provide a useful way to experiment with the
    Backpropagation learning algorithm.

    Last Update Oct 7/99...thanks to Stephane Pouyet <
    pouyet@nist.gov>
*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
```

```

#define sigm(x)      1/(1 + exp(-(double)x))
#define dxsigm(y)    (float)(y)*(1.0-y))
#define IN            35     /* number if inputs */
#define HIDDEN        5      /* number of hidden units */
#define OUT           10     /* number of outputs */
#define EPSILON       0.005  /* maximum Mean Square Error
to stop training */
#define NUMTRAIN      18     /* number of training patterns
*/

float inhiddw[IN][HIDDEN];
float hidoutw[HIDDEN][OUT];
float deltaihw[IN][HIDDEN];
float deltahow[HIDDEN][OUT];
float x[IN];
float y[HIDDEN];
float z[OUT];

/* training patterns */
int   actafer[NUMTRAIN][IN] = { { 0,1,1,1,1,1,0,
                                  1,0,0,0,0,0,1,
                                  1,0,0,0,0,0,1,
                                  1,0,0,0,0,0,1,
                                  0,1,1,1,1,1,0 },

                                { 0,0,0,0,0,0,0,
                                  0,1,0,0,0,0,1,
                                  1,1,1,1,1,1,1,
                                  0,0,0,0,0,0,1,
```

```
                                      0,0,0,0,0,0,0 },

                                    { 0,1,0,0,0,0,1,
                                      1,0,0,0,0,1,1,
                                      1,0,0,0,1,0,1,
                                      1,0,0,1,0,0,1,
                                      0,1,1,0,0,0,1 },

                                    { 1,0,0,0,0,1,0,
                                      1,0,0,0,0,0,1,
                                      1,0,0,1,0,0,1,
                                      1,1,1,0,1,0,1,
                                      1,0,0,0,1,1,0 },

                                    { 0,0,0,1,1,0,0,
                                      0,0,1,0,1,0,0,
                                      0,1,0,0,1,0,0,
                                      1,1,1,1,1,1,1,
                                      0,0,0,0,1,0,0 },

                                    { 1,1,1,0,0,1,0,
                                      1,0,1,0,0,0,1,
                                      1,0,1,0,0,0,1,
                                      1,0,1,0,0,0,1,
                                      1,0,0,1,1,1,0 },

                                    { 0,0,1,1,1,1,0,
                                      0,1,0,1,0,0,1,
                                      1,0,0,1,0,0,1,
                                      1,0,0,1,0,0,1,
```

```
                                0,0,0,0,1,1,0 },

                              { 1,0,0,0,0,0,0,
                                1,0,0,0,0,0,0,
                                1,0,0,1,1,1,1,
                                1,0,1,0,0,0,0,
                                1,1,0,0,0,0,0 },

                              { 0,1,1,0,1,1,0,
                                1,0,0,1,0,0,1,
                                1,0,0,1,0,0,1,
                                1,0,0,1,0,0,1,
                                0,1,1,0,1,1,0 },

                              { 0,1,1,0,0,0,0,
                                1,0,0,1,0,0,1,
                                1,0,0,1,0,0,1,
                                1,0,0,1,0,1,0,
                                0,1,1,1,1,0,0 },

                              { 1,1,1,1,0,0,0,    /* 4
                                */
                                0,0,0,1,0,0,0,
                                0,0,0,1,0,0,0,
                                0,0,0,1,0,0,0,
                                1,1,1,1,1,1,1 },

                              { 1,1,1,1,0,1,0,    /* 5
                                */
                                1,0,0,1,0,0,1,
```

```
                    1,0,0,1,0,0,1,
                    1,0,0,1,0,0,1,
                    1,0,0,0,1,1,0 },

                  { 1,0,0,0,0,0,0,     /*
                    7 */
                    1,0,0,0,0,0,0,
                    1,0,0,1,0,0,0,
                    1,1,1,1,1,1,1,
                    0,0,0,1,0,0,0 },

                  { 0,1,0,0,0,1,0,     /*
                    3 */
                    1,0,0,0,0,0,1,
                    1,0,0,1,0,0,1,
                    1,0,1,0,1,0,1,
                    0,1,1,0,1,1,0 },

                  { 1,0,0,0,0,1,1,    /* 2
                    */
                    1,0,0,0,1,0,1,
                    1,0,0,1,0,0,1,
                    1,0,1,0,0,0,1,
                    1,1,0,0,0,0,1 },

                  { 1,1,1,1,0,0,0,    /* 4
                    abierto */
                    0,0,0,1,0,0,0,
                    0,0,0,1,0,0,0,
                    1,1,1,1,1,1,1,
```

```
                                  0,0,0,1,0,0,0 },

                                { 0,0,0,1,1,1,0,   /* 0
                                  */
                                  0,1,1,0,0,0,1,
                                  1,0,0,0,0,0,1,
                                  1,0,0,0,0,1,0,
                                  1,1,1,1,1,0,0 },

                                { 0,1,1,0,0,0,1,       /*
                                  9 */
                                  1,0,0,1,0,0,1,
                                  1,0,0,1,0,0,1,
                                  1,0,0,1,0,0,1,
                                  0,1,1,1,1,1,1 } };


/* desired outputs */
int   desout[NUMTRAIN][OUT] = { { 1,0,0,0,0,0,0,0,0,0 },
                                { 0,1,0,0,0,0,0,0,0,0 },
                                { 0,0,1,0,0,0,0,0,0,0 },
                                { 0,0,0,1,0,0,0,0,0,0 },
                                { 0,0,0,0,1,0,0,0,0,0 },
                                { 0,0,0,0,0,1,0,0,0,0 },
                                { 0,0,0,0,0,0,1,0,0,0 },
                                { 0,0,0,0,0,0,0,1,0,0 },
                                { 0,0,0,0,0,0,0,0,1,0 },
                                { 0,0,0,0,0,0,0,0,0,1 },
                                { 0,0,0,0,1,0,0,0,0,0 },
                                { 0,0,0,0,0,1,0,0,0,0 },
```

```
                                { 0,0,0,0,0,0,0,1,0,0 },
                                { 0,0,0,1,0,0,0,0,0,0 },
                                { 0,0,1,0,0,0,0,0,0,0 },
                                { 0,0,0,0,1,0,0,0,0,0 },
                                { 1,0,0,0,0,0,0,0,0,0 },
                                { 0,0,0,0,0,0,0,0,0,1 }
                                  };


float ehid[HIDDEN];
float eout[OUT];
int patr[NUMTRAIN];
float ecm[NUMTRAIN];
float delta=0.5;      /* learning rate */
float alfa=0.1;       /* momentum */
long int itr;
int matrizin[35];

int init();
void training();
void netanswer(int afer[]);
float ec(int x[],float y[],int SIZE);
void backprop(int k);
void error();

int init()
{
      int i,j;
      int ch;
      int num;
```

```
    srand48(time(0));
    for (i=0;i<IN;i++)
        for (j=0;j<HIDDEN;j++)
        {
            inhiddw[i][j] = -0.5 + (float) drand48();
            deltaihw[i][j] = 0;
        }

    for (i=0;i<HIDDEN;i++)
        for (j=0;j<OUT;j++)
        {
            hidoutw[i][j] = -0.5 + (float) drand48();
            deltahow[i][j] = 0;
        }

    for (i=0;i<NUMTRAIN;i++)
        patr[i] = 0;

    for (i=0;i<35;i++)
        matrizin[i]=0;
    return 1;
}

void training()
{
    int i,l,num;
    long int j;
    int t,rep;
    float p;
```

```
int ch;

i=0;
j=0;
num=0;
do
{
    do
    {

        /* select a random training pattern:  i = (in
                                                 t)(
                                                 NUM
                                                 TRA
                                                 IN*
                                                 rnd
                                                 ),
                                                 whe
                                                 re
                                                 0<
                                                 rnd
                                                 <1
                                                 */
        i = (int)(NUMTRAIN*(float) rand() /
            RAND_MAX);
    }
    while (patr[i]);
    for (rep=0;rep<3;rep++)
    {
        j++;
```

```
            netanswer(actafer[i]);
            backprop(i);
        }
        if (!(j%102)) /*showerr();*/
            printf("\n%ld",j);
        error();
        l = 1;
        for (t=0;t<NUMTRAIN;t++)
        {
            patr[t] = ecm[t] < EPSILON;
            l = l && (patr[t]);
        }
    }
    while (!l /* && !kbhit() */);

    printf("\n\n End of training\n");

}

void netanswer(int afer[])
{
    int i,j;
    float totin;

    for (i=0;i<IN;i++)
        x[i] = (float)afer[i];

    for (j=0;j<HIDDEN;j++)
    {
        totin = 0;
```

```
        for (i=0;i<IN;i++)
            totin = totin + x[i]*inhiddw[i][j];
        y[j] = sigm(totin);
    }

    for (j=0;j<OUT;j++)
    {
        totin = 0;
        for (i=0;i<HIDDEN;i++)
            totin = totin + y[i]*hidoutw[i][j];
        z[j] = sigm(totin);
    }
}

float ec(int a[],float b[],int SIZE)    /* Error measure
            */
{
    int i;
    float e=0;

    for (i=0;i<SIZE;i++)
        e = e + ((float)a[i] - b[i])*(a[i] - b[i]);
    e = 0.5 * e;
    return e;
}

void betaout(int i)    /* error out */
{
    int j;
    for (j=0;j<OUT;j++)
```

```
            eout[j] = 0;

        for (j=0;j<OUT;j++)
            eout[j] = z[j] - (float)desout[i][j];
}

void betahid()    /* error hidden */
{
        int i,j;
        for (i=0;i<HIDDEN;i++)
            ehid[i] = 0;

        for (i=0;i<HIDDEN;i++)
            for (j=0;j<OUT;j++)
                ehid[i] = ehid[i] + hidoutw[i][j]*z[j]*(1-z[
                            j])*eout[j];
}

void backprop(int k)
{
        int i,j;
        float temp;

        betaout(k);
        betahid();

        for (i=0;i<HIDDEN;i++)
            for (j=0;j<OUT;j++)
            {
                temp = -delta*y[i]*z[j]*(1-z[j])*eout[j];
```

```
                hidoutw[i][j] = hidoutw[i][j] + temp +
                                alfa*deltahow[i][j];
                deltahow[i][j] = temp;
            }

    for (i=0;i<IN;i++)
        for (j=0;j<HIDDEN;j++)
        {
            temp = -delta*x[i]*y[j]*(1-y[j])*ehid[j];
            inhiddw[i][j] = inhiddw[i][j] + temp +
                            alfa*deltaihw[i][j];
            deltaihw[i][j] = temp;
        }
}
void error()
{
    int i;

    for (i=0;i<NUMTRAIN;i++)
    {
        netanswer(actafer[i]);
        ecm[i]=ec(desout[i],z,OUT);
    }
}

void test()
{
    int i,j;

    for (;;)
```

```
    {
        printf("- Test -\n\n[");
        for (i=0;i<7;i++)
        {
            for (j=0;j<5;j++)
                scanf("%d",&matrizin[j*7+i]);
            printf("\n");
        }
        printf("]\n\n Output activations :\n");
        netanswer(matrizin);
        for (i=0;i<OUT;i++)
            printf("\nz[%d] = %f",i,z[i]);
    }
}

void main(int argc,char *argv[])
{
    int read;

    init();
    training();
    test();
}

/*
Example :

gcc -o Bkprop Bkprop.c -lm

% ./Bkprop
```

```

102
204
306
408
510
612
714
816
918
1020
1122
1224
1326
1428
1530
1632
1734
1836
1938
2040
2142
2244
2346
2448
2550
2652
2754
2856
2958
```

```
3060
3162
3264
3366
3468
3570
3672
3774
3876
3978
4080

End of training
- Test -

[0 0 1 0 0

0 0 1 0 0

0 0 1 0 0

0 0 0 0 0                          <-------- a '1'
with some noise

0 0 1 0 0

0 0 1 0 0

0 1 1 1 0

```

```
]

Output activations :

z[0] = 0.073368
z[1] = 0.606160                         <------- the
       highest activation
z[2] = 0.101022
z[3] = 0.017971
z[4] = 0.101509
z[5] = 0.000393
z[6] = 0.014482
z[7] = 0.212412
z[8] = 0.003177
z[9] = 0.006917- Test -

*/
```