Development of a Neural Network Based Software Package for the Automatic Recognition of License Plate Characters

A Thesis Presented to the Faculty of the College of Engineering and Technology Ohio University

In Partial Fulfillment of the Requirements for the Degree Master of Science

By
Songqing Chen
June 1992
This Thesis Has Been Approved
by the Department of
Industrial and Systems Engineering
and the College of
Engineering and Technology

Russ Professor of Industrial and Systems Engineering

Dean of the College of Engineering and Technology
ACKNOWLEDGEMENT

The author would like to express his sincere appreciation to Dr. Helmut T. Zwahlen for his clear guidance, great help and encouragement. The author is grateful to Dr. Robert Williams, Dr. Luis Rabelo and Dr. Jeffrey C. Dill for their valuable suggestions. Without their support and understanding, it would have been very difficult, if not impossible, for me to finish this thesis.
Table of Content

Abstract

Chapter One Introduction

1.1 Introduction 1
1.2 Statement of Problem 3
1.3 Background 8
1.4 Introduction to Neural Networks 16
1.5 Objective 19

Chapter Two Requirements

2.1 Implementation Requirements 21
2.2 Other Requirements 22
2.3 System Requirements 23
2.4 Implementation Consideration 24

Chapter Three System Development

3.1 Structure of The System 26
3.2 Data Acquisition 26
3.3 Preprocessing 29
  3.3.1 Objective 30
  3.3.2 Strategy 30
  3.3.3 Horizontal Scan 32
    3.3.3.1 Edge Enhancement 32
    3.3.3.2 Tresholding Images 39
    3.3.3.3 Segmentation and Horizontal Block Separation 40
  3.3.4 Vertical Scan 47
    3.3.4.1 Image Identification 47
    3.3.4.2 Noise Filtering of The Row Block 50
    3.3.4.3 Vertical Block Separation 52
3.3.4.4 Scaling The Character Blocks

3.4 Neural Networks

3.4.1 Neural Network Technology

3.4.1.1 Neural Network Architecture

3.4.1.2 Multilayer Feedforward Networks

3.4.2 Design of The Neural Network

3.4.3 Training The Neural Network

3.4.3.1 Initializing

3.4.3.2 Generating Output With Forward Pass

3.4.3.3 Error Correction With Backward Pass

3.4.3.4 Improving The Training

Chapter Four System Test

4.1 Training The Network

4.2 Testing The System

Chapter Five Conclusions and Recommendations

5.1 Conclusions

5.2 Contributions

5.3 Suggestions and Recommendations

Bibliography and References

Appendix A User Guide

Appendix B.1 License Plates for Training

Appendix B.2 License Plates for Testing

Appendix C.1 Program Guide

Appendix C.2 Program Listing
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Figure 3.1 Structure of the System</td>
<td>27</td>
</tr>
<tr>
<td>2.</td>
<td>Figure 3.2 Flowchart of the System</td>
<td>31</td>
</tr>
<tr>
<td>3.</td>
<td>Figure 3.3(a) A Black-on-White Image</td>
<td>33</td>
</tr>
<tr>
<td>4.</td>
<td>Figure 3.3(b) A White-on-Black Image</td>
<td>34</td>
</tr>
<tr>
<td>5.</td>
<td>Figure 3.4 Shift and Difference Technique</td>
<td>36</td>
</tr>
<tr>
<td>6.</td>
<td>Figure 3.5 Convolution Calculation</td>
<td>37</td>
</tr>
<tr>
<td>7.</td>
<td>Figure 3.6 Convolution Kernel For Vertical Edge Enhancement</td>
<td>38</td>
</tr>
<tr>
<td>8.</td>
<td>Figure 3.7(a) A White-on-Black Image After Edge Enhancement</td>
<td>41</td>
</tr>
<tr>
<td>9.</td>
<td>Figure 3.7(b) A Black-on-White Image After Edge Enhancement</td>
<td>42</td>
</tr>
<tr>
<td>10.</td>
<td>Figure 3.8(a) The Row Histogram of a White-on-Black Image</td>
<td>45</td>
</tr>
<tr>
<td>11.</td>
<td>Figure 3.8(b) The Row Histogram of a Black-on-White Image</td>
<td>45</td>
</tr>
<tr>
<td>12.</td>
<td>Figure 3.9 Row Block Grouping and Separating</td>
<td>46</td>
</tr>
<tr>
<td>13.</td>
<td>Figure 3.10 Convolution Kernel For Noise Filtering</td>
<td>51</td>
</tr>
<tr>
<td>14.</td>
<td>Figure 3.11(a) Character Row Block of a White-on-Black Image</td>
<td>54</td>
</tr>
<tr>
<td>15.</td>
<td>Figure 3.11(b) Character Row Block of a Black-on-White Image</td>
<td>54</td>
</tr>
<tr>
<td>16.</td>
<td>Figure 3.12(a) Character Blocks of a White-on-Black Image</td>
<td>56</td>
</tr>
<tr>
<td>17.</td>
<td>Figure 3.12(b) Character Blocks of a Black-on-White Image</td>
<td>56</td>
</tr>
<tr>
<td>18.</td>
<td>Figure 3.13 Character Scaling</td>
<td>58</td>
</tr>
<tr>
<td>19.</td>
<td>Figure 3.14 Structure of Neural Network</td>
<td>66</td>
</tr>
<tr>
<td>20.</td>
<td>Figure 3.15 The Sigmoid Transfer Function</td>
<td>70</td>
</tr>
<tr>
<td>21.</td>
<td>Figure 3.16 The Derivative of the Sigmoid Transfer Function</td>
<td>74</td>
</tr>
<tr>
<td>22.</td>
<td>Figure 3.17 Error Back-propagation</td>
<td>77</td>
</tr>
<tr>
<td>23.</td>
<td>Figure 4.1 License Plates for Training</td>
<td>83</td>
</tr>
<tr>
<td>24.</td>
<td>Figure 4.2 Character Table of the Training Set</td>
<td>85</td>
</tr>
<tr>
<td>25.</td>
<td>Figure 4.3 License Plates for Testing</td>
<td>87</td>
</tr>
<tr>
<td>26.</td>
<td>Figure 4.4 License Plate Characters and the Outputs</td>
<td>88</td>
</tr>
<tr>
<td>27.</td>
<td>Figure A.1 Main Window of the System</td>
<td>101</td>
</tr>
<tr>
<td>28.</td>
<td>Figure A.2 Commands For Processing Pictures</td>
<td>103</td>
</tr>
<tr>
<td>29.</td>
<td>Figure A.3 Window For Entering a File Name</td>
<td>103</td>
</tr>
<tr>
<td>30.</td>
<td>Figure A.4 Window For Saving a File</td>
<td>104</td>
</tr>
<tr>
<td>31.</td>
<td>Figure A.5 Window For Displaying a Picture</td>
<td>105</td>
</tr>
<tr>
<td>32.</td>
<td>Figure A.6 Message From the Preprocessing</td>
<td>106</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>-----------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>A.7</td>
<td>Setting Options of The system</td>
<td>107</td>
</tr>
<tr>
<td>A.8</td>
<td>Commands For Training the Network</td>
<td>108</td>
</tr>
<tr>
<td>A.9</td>
<td>Displaying the Total Error in Training</td>
<td>109</td>
</tr>
<tr>
<td>A.10</td>
<td>Window Displaying the Training Time</td>
<td>110</td>
</tr>
<tr>
<td>A.11</td>
<td>Commands For Saving/Loading Weights</td>
<td>111</td>
</tr>
<tr>
<td>A.12</td>
<td>Output From the Neural Network</td>
<td>112</td>
</tr>
<tr>
<td>A.13</td>
<td>Changing Network Parameters</td>
<td>114</td>
</tr>
<tr>
<td>A.14</td>
<td>Basic Information About the System</td>
<td>114</td>
</tr>
<tr>
<td>B.1</td>
<td>Structure of the Program</td>
<td>188</td>
</tr>
<tr>
<td>B.2</td>
<td>The Preprocessing Sequence</td>
<td>190</td>
</tr>
<tr>
<td>B.3</td>
<td>The Operation of the Neural Network</td>
<td>193</td>
</tr>
</tbody>
</table>
ABSTRACT

This research studies the techniques being used in character recognition. Software has been developed to automatically recognize license plate characters. The program is written in Borland C++ 2.0, and runs on Microsoft Windows 3.0. The system consists of two main parts: the preprocessor and the neural network. The preprocessor separates all characters from a picture. The neural network has a three-layer architecture using a back-propagation training method. Instead of using the derivative of the sigmoid transfer function, a differential step-size function is applied to the output neurons to solve local minima problems in training. The neural network recognizes the characters from the preprocessor.

To run the system, an IBM 286 PC or compatible with at least 1M memory is required. An image digitizer system CapCalc was used to generate an image file in 512x480 format with 0-255 gray levels. It recognizes both white-on-black and black-on-white pictures. The pictures of the license plates should be fairly clear and complete.

The neural network was trained with 145 characters from 24 license plate pictures of different characters and fonts. The pictures included 15 black-on-white and 9 white-on-black. The system was tested with additional 35 license plate pictures. It was able to recognize about half of the characters on the 5 pictures which met the requirements. On a 386 PS/2 model 70, it took about 16 seconds to recognize one picture. The system demonstrated the feasibility of automatically recognizing license plate characters using image-processing techniques and a neural network. This was an exploratory lab study, and was not intended to be a realistic model of a biological vision system.
1.1 Introduction

Pattern recognition is an important field of computer applications. This research studies the methods and techniques being used in alphabet and number recognition. A software program is developed to automatically recognize license plate pictures. It is intended as a tool for recognizing fairly clear pictures taken in the lab, and is not intended for use in a realistic situation. This program is written in Borland C++ (version 2.0), and runs on Microsoft Windows 3.0 on IBM PC or compatible computers.

Interactions involving humans are effectively carried out through the medium of images. Images permit the expression of shape, size, position, and other information. Without them, cognitions and communications as we know them would be impossible.

Interest in digital image processing techniques dates back to the early 1920s. Applications of digital image processing became widespread in the middle 1960s, when third-generation digital computers began to offer the speed and storage capabilities required for practical implementation of image processing.

Image processing is a widely used technique for many tasks. It has been used on pictures returned from satellites, as a tool for investigation of the Earth's resources, and in robotics. In the past, the limitations of computer hardware, both in cost and size, restricted us to trying very simple techniques. Optical character recognition (OCR), used for a broad range of tasks, demands real, intensive computing power. Not surprisingly, the first OCR systems were
predominantly based on specialized machines with integrated, high accuracy scanners and dedicated recognition hardware.

Now image processing at the desktop with a personal computer is becoming more and more popular. This technology is possible because of increasingly higher performance levels of the personal computer, high-resolution graphics adapters, and the low cost of digitizer hardware. Optical character recognition software becomes more attractive as the need for volumes of accurate data increases and technology improves. Giving a computer the ability to see, even in a very limited sense, is an important and exciting endeavor.

Pattern recognition could be considered as mapping a pattern correctly from pattern space into one of many classes [Yoh-Han Pao, 1989]. The procedure by which a pattern is mapped into the correct class is an opaque one. The details of the process are not only inaccessible to other observers, but also the process is generally not even understood by the recognizer. The task of implementing computer-based pattern recognition is to replace the opaque mapping with a transparent mapping that can be described precisely to a computer.

As industry identifies and exploits different ways to utilize information through computers, the pressure grows to find cost and time-efficient methods to extract, enter, download or transfer information from various sources to formats which are readable by standard computer software. For years, one of the biggest challenges in information gathering has been to find an affordable and accurate way to capture information from the images which include letters, numbers, and other symbols. Recent developments in optical character recognition software have lowered the price of accuracy, of relative speed and of storing recognized characters in a variety of standard formats.
In the field of Transportation, much effort is spent in keeping track of the information pertaining to drivers and their vehicles. Very often the license plate number is the key to identify the driver and retrieve information from a database. To automate and improve automotive administration, it is essential to electronically recognize license plates.

Although computers are widely used in many aspects of our life, the automation of vision is still a very difficult problem. However, the use of neural networks is emerging as a good alternative to character recognition. Their noise tolerance, flexibility, and performance promise great hope for this area. This project has attempted to combine some digital image processing techniques with neural networks to develop a program to automatically detect characters on license plates.

1.2 Statement of Problem

Something that humans do well but computers still do poorly is pattern recognition. Pattern recognition is essential for seeing and understanding the content of images. Consider that 98% of the data we receive each second comes from our eyes. We have an inherently wider pipeline to the world. Pattern recognition is a difficult problem of computer applications because computers cannot see as human eyes do.

Pattern-recognition systems take a signal, remove the noise from it and attempt to fit it to a known pattern. Examples include optical character recognition (OCR) machines and software, digital image processing, digital sound processing, and natural language understanding and recognition. The pattern-recognition problem is three-fold:

1) Pattern matching.
Given a set of rules and a population of objects, determine which ones fit in the set,

2) Induction.

Given a set of objects that are said to be similar or dissimilar, find a minimal set of rules to describe the pattern of the objects,

3) Classification.

Given a taxonomical system and a set of objects, classify each object.

The features are difficult to implement in a computer. Achieving true machine intelligence will be difficult until this problem is overcome.

In the Department of Industrial and Systems Engineering at Ohio University, many projects with outside companies involve the detection and recognition of images. Some projects with the 3M Corporation involve the detection and recognition of license plates and traffic signs. These projects are very important to the automotive industry and the field of Transportation.

Managing automotive and traffic systems is a big problem for the police and the Transportation agencies. A large workforce is required to monitor and maintain this information. One of the tasks involved is recognizing and recording the license plate numbers and driver information. When police looks for a car, they usually look for the license plate number. Without an automated recognition system, keeping track of the license plate numbers is very time-consuming.

One issue that is being discussed in the area of automatic recognition is that of using the automatic recognition technology to connect a database to license plates, or to include the data on the license plate numbers. The license plate becomes significant when it is linked to a database in the Department of Motor Vehicles, revealing a great deal about the owner of that particular plate. It is impossible to coordinate among different states without electronically storing,
connecting and processing the license plate and driver information. License plate image processing and databases can allow users to access information very efficiently.

In many facilities, only the authorized vehicles are allowed to enter. All cars need to be checked in the entrances. Automatic recognition systems can help to simplify this procedure. All license plate numbers of authorized vehicles and related information are stored in a database and linked to the system. When a car stops at an entrance, the system records the license plate number, and checks against the database to determine whether the vehicle be allowed to enter or not.

An automatic recognition system may also be used in highway toll stations. Whenever cars pass through these stations, their license plates are scanned and recognized immediately. This could then be matched against a database of drivers who have not paid their fines or tickets, or if the cars have been involved in accidents or crimes. The police can be informed when a positive match is found.

In the imaging area, enhanced image resolution, manipulation algorithms and techniques that are currently available are not necessarily easing the information management problems that users face, such as automatically processing and manipulating their data. In fact, in some ways, these new methods may even complicate the operational side. What is needed are advances in character recognition technology -- advances that enable text information within the scanned image output to be effectively processed.

The crucial factor in the processing of license plates is time—the time to acquire and use large amounts of information as more and more license plates are issued. Advances in memory technology have helped in the handling of
image data. Cost-effective semiconductors, optical and magnetic memory have made the matter of storage easy and effective. Further progress on that front is expected as optical and magnetic media capacities expand and as multi-megabit devices come into the market place.

As memory capacities have mushroomed, so too has the volume of applications that require image acquisition and manipulation of the digital files derived from them. There has also been an increased demand for applications that merge text information with related image files. Higher speed CPUs and memory devices can help speed up the processing pace.

The need to combine image and text files poses other OCR challenges. For one thing, there are incongruities between the way the data are organized for a picture image and how they are structured and handled for a text file. Text information is created and stored through an alphanumeric code such as ASCII or EBCDIC, which uniquely defines each character. While, image information is handled on a bit-by-bit basis.

Humans recognize characters by applying an enormous body of knowledge that ranges from low-level image features to high-level facts about the world and the picture's content. These features and facts are highly interrelated. No information can be considered by itself. Hence, to apply the limited ability of current computer models to absorb a large body of facts is a major task in achieving automatic recognition of license plates. The effective models must determine, maintain, and program all necessary facts of license plates into the system. They must integrate the interrelations among pixels to rapidly interpret the characters.

Writing practical OCR software has turned out to be a major hurdle for developers. What the human eye and brain can understand easily has proven
quite difficult for computers. There are two main techniques used in character recognition: matrix matching and feature extraction. Feature extraction is a more sophisticated OCR technique than matrix matching because it is based on the principle that each character has distinct features. Some of the advanced feature recognition systems use artificial intelligence algorithms to 'learn' the characteristics of new fonts. Neural networks technology promises breakthroughs in OCR systems.

Many industries are working on the automatic character recognition system. For example, the U.S. Postal Service plans to establish a completely automated mail processing operation by 1995 [Vanessa Jo Grimm, 1990]. The Postal Board of Governors has budgeted $380 million for the letter automation project. The total cost of the automated mail processing system is estimated at $18 billion.

Some problems are still not possible to solve with today's technology. Many people would like to have a software package to scan the contents of forms for data-entry use, for instance. But recognition algorithms can not differentiate in any useful way between the text and the box surrounding it. And there is still no accurate way to recognize handwritten form entries. Parts of the problem are already being tackled. Today, industry researchers are addressing the issues of font learning, selection of portions of images to be read, and automatic graphics/text differentiation.

The power of the human vision system has attracted many to investigate the use of neural networks for machine vision. Neural networks technology promises further breakthroughs in the realm of reading. However, considerable work must be done before this technology can actually be used for real-world applications.
License plate recognition has many characteristics that are suitable for neural networks. First, there are enough license plates available to train and test neural networks. Second, characters on license plates may have marks and dust, which are usually not found in printed characters. Third, these characters may not be complete. Also, there is no algorithm or defined function to process images of license plates. These obstacles pose a great opportunity to apply neural networks for character recognition. Neural networks may offer improved performance over conventional algorithms in this area.

1.3 Background

The main progress made in the area of character recognition is that the optical character recognition (OCR) technology converts scanned documents into text files. Many businesses now want to convert their filing cabinets filled with letters, documents, and random scraps of paper into searchable word-processing files and databases. Hence, this leads to the great interest in optical character recognition (OCR), the process by which a computer turns a scanned image of a page into modifiable text. OCR technology can save hours of manual text transcription and make projects possible that would otherwise be too labor-intensive and costly.

The IBM AT, with its 16-bit expansion bus and higher clock speeds, made relatively accurate and reasonably fast OCR possible in a PC environment. But until recently, the best PC-based OCR systems (notably Caere's original OmniPage and Calera's TrueScan) required appropriate coprocessor cards to handle image buffering and the recognition tasks. The board added a cost premium that discouraged a wide use of these packages. Meanwhile, less-expensive, software-only products have offered competent recognition, but
they're less accurate than the board-based packages and less capable of handling the complex images with embedded graphics.

The increasing popularity of 386-based PCs changes things entirely. The extra processing power and memory capacity of the 32-bit 80386 processor makes it possible for OCR software developers to offer the same recognition capabilities without the coprocessor board. The task is so computation-intensive that no general-application OCR software exists for machines slower than the 80386 computers.

The number of reasonably priced OCR boards and software in the PC and Mac worlds is growing. OCR technology helps to bridge the gap between scanned image pixel bits and the multibit bytes used to define word characters. Several approaches can be used for this translation process. Matrix matching and feature extraction are two popular techniques.

The matrix matching font recognition technique isolates characters in a fixed matrix and compares them with tables of known images [Morton Balban, 1990]. When a match is made, an ASCII character is produced. Unrecognized characters are flagged or a best guess is made. Matrix matching is a relatively slow process that works best with monospaced characters like the ones produced by printers or typewriters.

Feature extraction is a more sophisticated way of doing the same thing. It's based on the principle that each character has distinct features, regardless of font or spacing considerations [Morton Balban, 1990]. The associated software analyzes the scanned character, builds a features list and then determines which character has most or all of these features.

One of the techniques based on feature extraction uses a decision-tree. It arranges all the features in a decision tree and detects characters based on the
features presented or not presented in the characters [Mike James, 1988]. Each node in the tree corresponds to a measurement and there is a branch for each value of the measurement. The recognition can be considered as a multi-step process. Each step uses a classification rule to divide the characters into increasingly smaller groups until classification is achieved. This approach makes only these measurements which are essential to classify the character in question.

The problem with a decision tree is that even a small defect in the character may affect some of its features and may change the values of the measurements. This may force the character into a different branch and eventually a wrong conclusion will result. Also, it requires people to analyse the characters manually, and sometimes it's not easy to build a decision tree to cover all the characters and fonts.

There are two kinds of OCR packages: those that recognize text automatically and those that need to be trained to recognize each font. It might seem that automatic recognition would have tremendous advantages over the trainable packages. But if an automatic package doesn't perform well on a given document, there isn't any way to improve the results, whereas trainable packages can learn to deal with difficult or unusual typefaces.

Trainable software usually is the older, more-familiar type. These packages typically use a pattern-matching recognition algorithm (also known as matrix or template matching) in which the image of a character is compared to the known images of the letters, numbers, and symbols in a given typeface [Lori Grunin, 1990]. During a typical training session, the software displays the image of each character and you type the letter to which it corresponds. The result is a font image file that can be saved for use with other pages with the same kind of
type. They require users to store the image of every character in every font for future comparison, a tedious process.

Though simple in concept, trainable packages have serious drawbacks in practice. The initial training process is tedious and time-consuming. In most cases, you must train each size of each typeface separately, and even when the package comes with pre-trained fonts, recognition can be stymied by slight variations in type style. Though some trainable packages use various methods of shortening the training cycle (such as interpolating a font table from a different size of the same typeface, or from a similar typeface), but these innovations don't always work very well.

The better packages rely on omnifont technology, which uses feature extraction (also known as topological recognition or intelligent character recognition) to discern letters independently of their specific shapes [Lori Grunin, 1990]. In this scheme, the system breaks down each character into its constituent parts (lines, curves, angles, ascenders, and descendents) and compares the result with its database of character information. This way of looking at characters results in a nearly unlimited font vocabulary for the package.

Omnifont technology produces far more accurate results with less work than trainable packages, and omnifont packages are better at reading multicolumn documents with graphics as well [Lori Grunin, 1990]. The accuracy of omnifont software on unusual fonts can be increased by incorporating trainability as a supplementary option, as in Ocron's Perceive [Bruce Brown, 1990].

OCR systems generally work best on typewritten text. OCR software reads equal-spaced and PostScript fonts with high accuracy. For smaller type sizes the result is an excessive number of joined characters, since the scanner
can't always find that single line of white pixels separating two characters. There are errors and clean up work to be done on OCR output. No current OCR system is perfect, even on simple text.

Optical character recognition (OCR) technology for the Macintosh has advanced greatly. Products range from low-end software packages to be used with inexpensive hand-held scanners to expensive dedicated software-hardware combinations. Automatic packages include: Caere Corp's OmniPage 2.1 ($795); Xerox Imaging Systems Inc's Datacopy AccuText 2.0 ($799); and CTA Inc's ScanReader ($495). Olduvai Corp's Read-It! O.C.R. 2.1 ($495) is a trainable package with type tables for many common fonts. Caere Corp's Parallel Reader ($10,995) is a high-end, dedicated hardware system.

Fully automatic programs like OmniPage can sort out text columns, type sizes and even images within text [Bill O'Brien, 1990]. After examining the white space on the scanned image of the document and determining its boundaries, OmniPage goes on to define the areas that might contain text.

Perceive 1.0 from Ocron Inc. combines fairly accurate omnifont OCR methodology with a cross-mapping learn mode. Problem fonts, odd characters and symbols can all be learned and stored in one or more trained font files [Bruce Brown, 1990; Mitt Jones, 1990]. Font files can then be used singly or in combination with the omnifont mode to scan and accurately recognize most conceivable fonts. Only one trained font can be used at a time, which makes it difficult to recognize documents with multiple fonts. Full-page scanning reads, or attempts to read, everything on a page in the left-to-right order in which it's found. To maintain multiple text columns or to cut around graphics, however, the page can be scanned in partial page mode, and then the operator can manually
create boxes around the text to be recognized. Drawing boxes around the text elements on the scanned page is time-consuming and sometimes awkward.

Hewlett-Packard Co.'s (HP's) AccuScan performs page decomposition automatically, mainly for scanning incoming faxes and typewritten data sheets [Bruce Brown, 1990]. AccuScan identifies regions as text or graphics to facilitate this process. When using AccuScan in an interactive mode, the program stops after identifying these regions on the scanned page and waits for the operator to adjust or change the marked regions. However, in manual mode, users must mark the regions themselves.

Flagstaff Engineering Inc's SPOT OCR software is a trainable program that comes with only seven pre-defined font tables, each restricted to a specific type size [Alfred Poor, 1990]. It cannot separate letters that touch each other. It can access the table for the font most similar to the one being trained and use this as a jumping off point for creating a new font table.

Recognize! from Dest Corp. is capable of automatically decomposing complex pages, processing multicolumn layouts into single columns, or producing multicolumn output for word processors that can handle such formatting [David Dean, 1990]. It has masking features that allow the elimination of individual areas on the page. But it offers no templates and no way to reorder the text regions it finds or to save graphics separately.

Most PC and Macintosh systems require constant operator attention and take about two minutes per page to scan a relatively simple document like a letter. Simple documents typed on good paper with fixed margins, even spacing and no special characters approach perfect OCR recognition. But most documents are not like that.

OCR algorithms have made great strides in accuracy, but no system is perfect even on simple text. Most packages tend to make more of some types of
errors than others. When scanned, text files require some editing. As is the case with most documents, it actually may take longer to scan a series of short documents than for a competent typist to rekey them. The average magazine page after scanning displays many errors, some of which probably can be tolerated. Others may be unacceptable depending on the application in question.

Some of the elements of type design that make text easier on the eye and more attractive to read can cause difficulties to omnifont and trainable packages. Fractures such as those in degraded photocopies can cause single characters to be interpreted as two or three. Dot-matrix print poses a related problem, since the dots that make up the letters often don't touch. However, since this type of print forms regular patterns, OCR software such as Calera's WordScan Plus and Caere's OmniDraft add-in for OmniPage incorporate a connect-the-dots capability to compensate.

Some of the more advanced feature extraction systems use artificial intelligence techniques that provide a degree of judgmental capability [Morton Balban, 1990]. These systems have the ability to "learn" the characteristics and idiosyncrasies of a new font. Artificial intelligence capabilities have given OCR technology more flexibility in recognizing different font types. It has given those systems the ability to recognize unknown fonts and translate them with a high degree of accuracy into computer files. That would permit image files to be more completely and accurately translated. Some newer systems are even attempting to read handwriting. Artificial intelligence features in OCR to detect unlikely letter combinations or out-of-place numbers now are appearing, but these are far from perfect. The current computer with arithmetical operations can't solve the complexity of automatic character recognition. Neural networks, however, might
hold the key to achieving rapid and more-reliable machine-based character recognition.

One research [Todd King, 1989] uses a three-layer network to associate each character with a number. The network has a linear or linear-threshold function. The neurons are binary, either on or off. But to uniquely classify 26 characters in the alphabet, the hidden layer must have 26 neurons. Each neuron memorizes one pattern, instead of extracting features from patterns.

Many more powerful networks using back-propagation have been developed, such as the one discussed in AI EXPERT (Oct., 1990). It is a 3-layer, maximally-connected neural network that uses the sigmoid function, back-propagation with momentum. It performs well in the 10-character example, but may get into local minima when trained to recognize 26 alphabetic characters.

Furthermore, some research is attempting to recognize hand-writing. An application of handwritten digit recognition is developed by Y Le Cun, B. Boser, J. S. Denker, D. Henderson, etc. in 1990. The system uses a four hidden-layer back-propagation networks. The preprocessing is performed by someone else, and the input to the network consists of normalized images of isolated digits. A complete training session takes about 3 days on a SUN SPARCstation 1 using the SN2 connectionist simulator. This method has 1% error rate and about a 9% reject rate on zip-code digits provided by the U.S. Postal Services.

Another hand-printed character recognition system is developed by Gale L. Martin and James A. Pittman in 1990. The training data are scanned from bank checks. The hand-printed digits are pre-segmented and size-normalized to a 15x24 grayscale array. The back-propagation network achieved an error rate of 1-2% at a 10% reject rate.

Some neural networks development environments can be found in the market, including NeuralWorks from NeuralWare Inc. They integrate training...
algorithms with useful utilities for fast application development, including character recognition. But it is time-consuming and the performance and integration with other functions should be considerations for such tasks.

At the same time, some research has been carried out to identify and recognize license plates. A study group of the Ontario Ministry of Transportation considered a computerized road and bridge toll system based on embedded microprocessor chips and scanning technology [Monica Frangini, 1990]. The chip is embedded in a car's license plate or windshield, which is scanned as the car passes through certain check points, leading to the identification of the license plate number and its owner. Another method under consideration is to use bar-code labels to uniquely identify each license plate. Both methods require some additional devices to be installed in the vehicles.

This research is the first attempt to use neural networks, combined with image processing techniques, to automatically recognize license plate characters. The system uses a single network to recognize characters of various fonts and sizes. It recognizes automatically without additional intervention by user.

1.4 Introduction to Neural Networks

Neural networks are artificial intelligence technologies that build upon process element nodes patterned after neural interconnections of the human brain. It is people's attempt to mimic the way the brain works in order to infer from incomplete information. A neural network is composed of many interconnected processing elements that operate in parallel. Neural networks offer improved performance over conventional technologies in areas including pattern recognition.
The most significant difference between conventional computers and neural networks is that the latter can learn by training. Instead of programming a neural network, you teach it to give acceptable answers. It learns by example and does not require conventional programming. It is trained to perform a task by showing it examples of the input it will receive paired with the output it is to deliver. The network then learns the correlations between these input examples and the expected outcomes. It can repeat these examples and generalize the relationships. After training, it generates appropriate outputs in response to new inputs. Neural networks are superior for learning complex relationships among many components because the system teaches itself the multi-dimensional mapping of the problem.

Neural networks have evolved based on the theory by Williams James in 1890 [Alianna J. Maren, Dan Jones, and Stanley Franklin, 1990]. He defines a neuronal process of learning and re-integration as a process which reconstructs the missing information and provides total recall. Using formal logic, McCulloch and Pitts in 1943 developed models of neural networks based on simple neurons which are considered to be binary devices with fixed thresholds. Early researchers also include Hebb and Lashley [Alianna J. Maren, Dan Jones, and Stanley Franklin, 1990]. The first computer simulations were not done until the mid-Fifties.

In the past few years, neural networks have received a great deal of attention and are being considered as one of the greatest computational tools ever developed. Much of the excitement is due to the apparent ability to imitate the brain's decision-making and conclusion-drawing capabilities when presented with complex, noisy, irrelevant, and partial information.

Neural networks offer specific processing advantages, including adaptive learning, self-organization, fault-tolerance and ease of insertion into existing
technology. Their parallelism, speed, and trainability make them fault-tolerant, as well as fast and efficient for handling large amounts of data. Neural networks perform well in pattern discrimination [Alianna J. Maren, Dan Jones, and Stanley Franklin, 1990]. Other available pattern recognition techniques are often not able to deal with the complexities in many applications. The kinds of problems they best solve are also those that humans do well: association, evaluation, and pattern recognition. They have been used to classify speech, and handwriting, predict financial trends, evaluate personnel data, etc.

Back-propagation networks are probably the most well-known and widely applied of the neural networks today. They are an outgrowth of earlier work on Perceptrons, with the addition of a hidden layer and use of the Generalized Delta Rule (by Dr. Paul J. Werbos in 1974) for learning [Alianna J. Maren, Dan Jones, and Stanley Franklin, 1990]. The earliest research was proposed by Dr. Paul J. Werbos in 1974, and later published independently by Parker in 1985 and Le Cun in 1986 [Alianna J. Maren, Dan Jones, and Stanley Franklin, 1990]. Dr. Werbos explains how the back-propagation method can be applied to dynamic processes and applies his model to socio-economic forecasting in his Ph.D. thesis. This method is popularized by David Rumelhart and James McClelland in 1986 as an important part of the re-emergence of the neural field [Alianna J. Maren, Dan Jones, and Stanley Franklin, 1990].

The key distinguishing characteristic of the back-propagation network is that it forms a mapping from a set of inputs to a set of outputs using features extracted from the input pattern. This is because the nodes in the hidden layer of the network learn to respond to features found in the input. Back-propagation neural networks excel at pattern classification.
Most back-propagation neural networks used in real applications have three layers or more. The input layer distributes the pattern throughout the network, the output layer generates an appropriate response, and the middle layer acts as a collection of feature detectors. Each neuron in the middle hidden layer looks for a key feature or features in the input pattern and reacts strongly when one is found. The output layer can then construct an appropriate output pattern based on the particular combination of features the hidden layer has detected.

The neurons are the processing elements to the networks' operation. The connections between neurons are also very important and especially their knowledge is stored on the weights on those connections. All the actual information processing is performed in the way the pattern is modified as it passes along the connections from the input layer to the output layer.

One of the major issues of neural networks is integration of neural network systems into the current systems and environment. The systems prepare data, and validate results, while the neural networks perform the classification, and pattern recognition. The system developed tries to use image processing techniques to extract the characters and the neural networks will make the final recognition.

1.5 Objective

The objective of this project is to develop a neural network computer software package which is able to recognize alphanumerics on license plates. The pictures have 512*480 pixels with 256 gray levels each. The program will be able to locate all the characters, as well as learn and recognize difficult fonts in the same picture. Position, size and different fonts won't be a problem to the program, however all characters must be placed on a fairly level line (+/- 5°
maximum deviation from horizontal).
Chapter Two  Requirements

No matter what the final system might be, the first thing that needs to do is to prepare the system's requirements and specifications. They should describe the system's behavior, what it does and how it does it. They serve as a model of the proposed system, and help us visualize the system before it is actually built. The requirements have to be analyzed for the given problem, and then the most appropriate and cost-effective system has to be designed to solve that problem.

2.1 Implementation Requirements

2.1.1 Hardware Requirements
a. An IBM 286 PC or compatible computer with at least one megabyte of memory, EGA/VGA adapter and monitor. A hard disk of at least 20 megabytes and a mouse are also required. A fast (25-33 MHz) 386 or 486 machine with math coprocessor is preferred.
b. The computer should be able to run Microsoft Windows 3.0 in Standard or Enhanced mode.
c. An image digitizer, such as CapCalc, that is capable of generating an image file in 512x480 pixel format with 0-255 gray levels for each pixel.

2.1.2 Software Requirements
a. Borland C++ compiler version 2.0 or later.
b. DOS operating system version 3.1 or later.
c. Microsoft Windows version 3.0 or later.
2.2 Other Requirements

To make the recognition feasible, some limitations must be imposed.

1) The license plate numbers should be the most significant objects in the pictures.

2) The image of the license plate should be fairly clear, and complete. There should be a good contrast between the target and the background. Based on experience, the minimum difference between the gray levels of characters and background is 20 for black-on-white images, and 40 for white-on-black images. The target and background luminances should fall within fairly narrow ranges, so that the above minimum difference may be satisfied.

3) The characters on license plates should not touch each other or other symbols.

4) The license plate should be placed in a horizontal position, at a distance from the camera so that a clear picture can be taken. The character string on the license plate can have a maximum deviation of 5° from the horizontal line. If the angle of deviation is greater than 5°, the characters in the license plate may appear to be joined with each other and other symbols on it. This makes recognition very difficult and even impossible. If the characters do not touch each other, then with proper training, the program should be able to recognize license plates having deviations of greater than 5°.

5) Regular room lighting conditions must be maintained.

6) The surface of the license plates and background should be flat. It should be parallel to the lens surface of the camera.

7) The license plate and its background must remain stationary while they are being photographed.
8) The license plate must not extend beyond the boundaries of the viewing area. It should be surrounded by background in the picture.

9) The license plate must not have any significant broken area around its characters. The program does not recognize most license plates with broken characters.

10) The program recognizes a minimum of five and up to seven letters and/or numbers, either white on black or black on white.

11) The height of the smallest character should not be less than 80% of that of the highest character in the same license plate. There is usually an additional mark among the characters on license plates which is less than or close to 70% of character height.

12) The pictures should be captured and digitized by the CapCalc or other similar systems. The image should be in 512 x 480 (horizontal by vertical) format. The gray level for each pixel should be in floating-point numbers or byte format ranging from 0 to 255.

13) This research is an exploratory lab study and not a field study.

2.3 System Requirements

The program being developed as a result of this project must:

1) Recognize the characters on a license plate in several seconds on a fast 386/486 machine with math coprocessor (see discussion in Chapter Five).

2) Under all the conditions mentioned above and with sufficient training, the program should be able to recognize 80% or more of all the characters on the license plates. A sample of 7 license plates with different sizes of characters (totally 35 pictures) is used to test the program.
3) Should be user friendly and not require any programming experience to use this software. It should support a mouse and all the commands may be selected from menus.

4) Be flexible to allow the user to interactively adjust the established parameters, including learning coefficient, and the momentum.

5) Be able to display the result of each preprocessing step. This gives the user an idea of what is going on, and what caused the problem when the recognition fails.

6) Be able to save all knowledge learned and use it later without retraining the network. The system is only useful when it does not have to be retrained every time it is used.

7) Be trainable to recognize new fonts without losing all knowledge accumulated up to this point.

8) Be able to display the image it is recognizing, so that the user knows what is being processed.

9) Be modular to allow reusability of some of its routines. It should be easily expandable by a person working on a similar topic.

2.4 Implementation Consideration

The C language is the choice to write this system. C is an excellent language for serious programming and is an extremely popular language. It is the language used by professional programmers for just about every type of application and on all types of computers: from PCs to mainframes. It allows programmers to write portable programs while taking advantage of the characteristics of the underlying processor. The actual implementation is written in Borland C++ 2.0. It was selected of because of its popularity, Windows support and high productivity platform.
The environment for running the system is Microsoft Windows. The memory management, multitasking abilities and ease of use of the Microsoft Windows 3.0 graphical user interface provide great values to the program.

The following chapters investigate the techniques and alternatives to meet the above requirements; an overall design and detail specifications of the system are presented; and a final test is conducted to guarantee that the system meets the requirements.
Chapter Three   System Development

3.1 Structure of The System

This chapter contains a detailed description of the functions for each component of the system. It also describes how the segments fit together and work with each other. It is important that all the data sets must satisfy the requirements listed in the previous chapter. The overall structure of the system is outlined in Fig 3.1.

The system consists of two main parts: the preprocessor and the neural network. Each image file is first input into the preprocessor. The preprocessor makes a horizontal and a vertical scan to separate all the characters from the whole picture. The horizontal scan identifies the row block containing the characters.

Based on their gray levels, the system divides all the pixels into foreground and background. In the vertical scan, all the potential blocks are separated by searching the foreground and background signals contained in each column. The system checks those blocks and drops all the background blocks. The character blocks are scaled into 7x9 array, a format that the neural network has already learned.

The knowledge learned by the neural network is saved in a file. Based on that, the neural network tries to recognize those 7x9 characters. The output is shown on a popup window.

3.2 Data Acquisition

It is important that there is enough data to yield sufficient training and test sets to train and evaluate the performance of the system.
Figure 3.1  Structure of the System
The recognition of images requires some hardware to capture images. The processing of an image into an array of numbers is referred to as digitization. A computerized luminance measurement and analysis system, CapCalc, was used to obtain the license plate pictures. It captures and digitizes the image, representing the analog intensities as digital gray levels on an IBM-compatible machine. The pictures were taken in the lab, with a white wooden board as the background. The wooden board was large enough so that only a license plate and the wooden board were shown in the picture. Attempts were made to place all license plates horizontally. The camera was placed at a distance of about 3 meters from the license plates. The license plates were centered in the picture, surrounded by the white background. Uniform illumination conditions for both the background and foreground were set up and used.

Each picture (a luminance value matrix) taken by the camera is represented as an array of numbers and consists of 245,760 pixel values (512 horizontal x 480 vertical). A total of 24 license plate pictures were taken with the CapCalc system and saved on the disk. The brightness values for all pixels were restricted to a fixed range of 0 to 255.

All the images processed by the system were taken under similar condition. If in future experiment this proves to be too difficult, then the camera may be adjusted or some additional hardware may be used to achieve the same effect. Consistency greatly improves the accuracy of the recognition.

The original image is saved in a format that the gray level of each pixel is represented by a floating-point number. The numbers are converted to integer values by rounding off the decimal point. On an IBM PC, four bytes are required to represent a floating-point number, while the converted image with integer gray
levels in the range 0 to 255 needs only 8 bits (1 byte) per pixel. It only requires one-fourth of the original disk space used. By making this change, the total of 24 image files used 5.76 Megabytes of disk space instead of 23.04 Megabytes. This can save a lot of storage for the computer, which may represent one-fourth or one half of the harddisk space available on many computers. Another advantage of integer values is that integer arithmetic and integer operations are simpler and much faster than the equivalent floating point operations. Indeed, as storage requirements and processing times for each point are multiplied by the number of pixels in an image, it is by the use of integer gray levels that images can be processed in a reasonable amount of time.

The input image to the system is a gray level image in which each point is assigned a single number (0-255) that indicates how bright it is. The brighter a pixel is, the higher gray level it has. The gray level of each point can be represented by a function level=(x,y) of two variables: row number x and column number y.

3.3 Preprocessing

Preprocessing was the most difficult area dealt with in designing the system. It applies many useful digital image processing techniques, including thresholding, edge detection, smoothing, segmentation [Mike James, 1988, Craig A. Lindley, 1991]. The key here is performing the proper processing in the proper sequences.

These processing functions can be classified into two categories: point processing and area processing [Craig A. Lindley, 1991]. Point processing changes a pixel's gray level based only on the original gray level of the pixel itself. The gray level of the pixel is replaced by a new one related to its original gray level. Point processing includes contrast stretching and thresholding.
Area processing alters a pixel's gray level based on its original gray level and that of the pixels that surround it. Those surrounding pixels are called neighbors. They are usually a two-dimensional square or rectangle of pixels with each dimension having an odd number of elements. They suggest how the gray levels are changing around the pixel. A 3 by 3 array is used for the neighbors throughout this system. Edge detection and smoothing are two types of area processing.

3.3.1 Objective

The objective of preprocessing is to separate each character from a picture. The system then feeds the characters one by one into the neural network for recognition.

3.3.2 Strategy

Preprocessing is based on the gray level differences between signal and background. For a black-on-white picture, the characters have lower gray levels than the background, and higher gray levels for a white-on-black picture. In this chapter, the background is referred to as the immediate background around the characters unless specified explicitly. The preprocessor locates the gray level differences, and separates the image into different groups according to the gray levels of pixels. The flowchart in Fig. 3.2 shows the steps involved in the preprocessing.

Separating each character means that the system has to find out where each character is located. The location of a character may be represented by both horizontal coordinates of the left and right sides of the characters and vertical coordinates of the top and bottom of the characters. The problem becomes one of finding the four coordinates. Based on the requirements, the
Figure 3.2  Flowchart of the System
license plates should be placed horizontally in the pictures. So the characters are located in approximately the same pixel rows. The rows containing these characters must be somewhat different from the other rows. It makes sense that the system first locates these rows. Once they are located, the system eliminates a lot of noise and has more relevant information about the image. This procedure is actually performed by a horizontal scan with edge detection.

After the rows are located, the characters need to be further separated from them. Based on the information obtained in the horizontal scan, the vertical scan separates each character block from the rows. All character blocks are scaled into a 7x9 format (the format used by the neural network).

In the context of pattern recognition, a description of how something is recognized is only useful if it is complete and exact that it can be successfully programmed. The following sections provide all the details for developing and programming the preprocessor. Two license plate images, one black-on-white and one white-on-black, are used to demonstrate the processing functions (Figure 3.3). Because of the limitations of the printer, these pictures are printed as binary pictures. The pixel values of the actual images used by the system range from 0 to 255.

3.3.3 Horizontal Scan

3.3.3.1 Edge Enhancement

An edge is a sudden change in the brightness of an image. It is the boundary between two regions with different gray levels. Its position is crucial to the location of the boundary between character and background. Edge enhancement is used as a preliminary step in image feature extraction. Edge enhancement enhances and detects the edges within an image [Mike James, 1988, Craig A. Lindley, 1991].
Figure 3.3 (a)  A Black-on-White Image
Figure 3.3 (b)  A White-on-Black Image
The Shift and Difference edge enhancement technique [Mike James, 1988, Craig A. Lindley, 1991] is used because only the vertical edges need to be enhanced. This method enhances image edges by shifting an image left by one pixel position and then subtracting the shifted image from the original. Figure 3.4 shows how this technique works by shifting and subtracting a simple image. All numbers represent the gray levels of the images. A absolute function is applied to the subtraction so the gray levels in the resulting image represent the amount of change, and not the direction. The result of the subtraction is a measure of the slope of the brightness trend. In an area of small changes in gray levels, the output would have low values, and would have relatively higher values in the area of large changes in gray levels.

Instead of actually shifting the images left, a convolution is used to obtain the same effect. The convolution algorithm is a very general-purpose and powerful algorithm that can be used in performing a variety of process transformations [Mike James, 1988, Craig A. Lindley, 1991]. A convolution can be thought of as a weighted summation of pixels next to each other. Each pixel in the small area (usually a 3x3 array) is multiplied by a similar dimensioned convolution kernel. The result of the calculation replaces the value of the center pixel of interest, as illustrated in Fig. 3-5.

Each element of the convolution kernel is a weighting factor. The size and the arrangement of the weighting factors contained in a convolution kernel determine the type of transform that is applied to the image data. Changing a weighting factor within a convolution kernel influences the magnitude and possibly the sign of the overall sum and therefore affects the value given to the pixel being computed. Most kernels are three by three, and all have odd numbers of rows and columns [Craig A. Lindley, 1991].
Figure 3.4 Shift and Difference Technique
Figure 3.5 Convolution Calculation

3x3 pixel neighborhood
P5 is the pixel being computed

Weighted Sum
p1 x w1 + 
p2 x w2 + 
p3 x w3 + 
p4 x w4 + 
p5 x w5 + 
p6 x w6 + 
p7 x w7 + 
p8 x w8 + 
p9 x w9

New Value for P5
Given a convolution kernel:

\[
\begin{array}{ccc}
    w_1 & w_2 & w_3 \\
    w_4 & w_5 & w_6 \\
    w_7 & w_8 & w_9
\end{array}
\]

In this kernel, \( w_1, w_2, \ldots, w_9 \) represent the weighting factors. The \( p_1, p_2, \ldots, p_9 \) are the gray levels of the pixel under consideration and the pixels around it. \( P_5 \) is the new gray level for the center pixel under consideration. This can be described by Equation 3-1 [Mike James, 1988, Craig A. Lindley, 1991].

\[
P_5 = w_1*p_1 + w_2*p_2 + \ldots + w_5*p_5 + \ldots + w_9*p_9
\]  

(3-1)

An absolute value function, \( f(x) = |x| \), is applied to the sum \( P_5 \), so that the system can detect both black-to-white and white-to-black pixel transition edges. If the sum is greater than 255, the new center pixel is assigned a gray level of 255. This processing applies to every pixel in the image, except the pixels at the top, left, right, and bottom borders of an image. The gray levels of an edge after detection are proportional to the changes of the gray levels surrounding the edge in the original image. The new image is used for further processing.

The kernel used for vertical edge enhancement is described in Fig 3.6 [Mike James, 1988, Craig A. Lindley, 1991].

\[
\begin{array}{ccc}
0 & 0 & 0 \\
-1 & 1 & 0 \\
0 & 0 & 0
\end{array}
\]

Figure 3.6 Convolution Kernel For Vertical Edge Enhancement
With this convolution kernel, the sum will be [Mike James, 1988, Craig A. Lindley, 1991]

\[ P_5 = |p_5 - p_4| \]  

This process actually is equivalent to the Shift and Difference technique as shown in Fig. 3.4. It achieves the same effect using a different implementation. Figure 3.7 shows the edges detected in the license plate pictures after the thresholding process.

3.3.3.2 Thresholding Images

Image thresholding is used for converting a gray-level (0 to 255) image into a binary (gray level either 0 or 1) image [Mike James, 1988, Craig A. Lindley, 1991]. To separate the characters (actually the edges) from the background, a threshold value is selected. All gray values greater than the threshold are considered to be signals, and all gray values equal to or less than the threshold to be background (for a white-on-black image). A threshold value results in a binary image that is an adequate representation of the original if the threshold has a value between the gray level of the signal and the gray level of the background.

In an automated pattern recognition system, there is usually no opportunity for human intervention, and this means that selecting a threshold has to be automated. One of the techniques used for threshold selection is the histogram method. If an image is composed of reasonably evenly illuminated objects against an evenly toned background then a histogram of the brightness values will show two peaks - one for the foreground and the other for the
background. If a threshold value is chosen to separate these two peaks, a good binary representation of the original will result.

Essentially, the threshold is chosen to lie in the valley of the image brightness histogram. The edges constitute only a small portion of the digitized image. The preprocessor discards most of the points below (or above) the threshold set by the system.

The thresholding function can be represented as [Mike James, 1988, Craig A. Lindley, 1991]:

\[
g(x,y) = \begin{cases} 
1, & \text{if } f(x,y) \geq T \\
0, & \text{otherwise}
\end{cases}
\]

\(f(x,y)\) is the old gray level of pixel \((x,y)\), and \(g(x,y)\) is the new one. \(T\) is the threshold. Based on the requirement in Chapter 2, the minimum difference between the gray levels of characters and background is 20 for all images. Threshold is assigned a value of 20 to detect the edges in the picture. The thresholding function considers all pixels, whose gray levels are 20 or more greater than that of their neighbor pixel \((P_5 \geq 20)\), as part of an edge (see Eq. 3-2). The image that results is a binary image. The output pictures are illustrated in Figure 3.7.

3.3.3.3 Segmentation and Horizontal Block Separation

One approach to image classification is to divide the image up into regions based on one property or another. Given an image, the task of dividing it into regions can be thought of as classifying each pixel in the image into one of
Figure 3.7 (a) A White-on-Black Image After Edge Enhancement
Figure 3.7 (b)
A Black-on-White Image After Edge Enhancement
a number of categories. Applying a thresholding operation may result in the
division of the pixels of an image into a foreground object and the background.

Segmentation is the process that subdivides an image into its constituent
parts or objects. A large region is divided into smaller and smaller regions
depending on finer and finer distinctions between pixels [Mike James, 1988,
Craig A. Lindley, 1991]. Segmentation is the most important element in
automated image analysis because it is at this step that objects or other entities
of interest are extracted from an image for subsequent processing. Segmentation algorithms generally are based on one of two basic properties of
gray-level values: discontinuity and similarity. Edges are enhanced based on
abrupt changes in gray levels.

After the edge detection and thresholding processes are completed, the
image buffer contains the pixel values of vertical edges and of the background.
The rows where the characters are located need to be found out. A summation
of gray levels of pixels is calculated for each row of the image, as shown in
Equation 3-3. Remember that all pixels have been through thresholding process
and have a value of 0 or 1.

\[
\text{Sum}_i = p_1 + p_2 + \ldots + p_j + \ldots + p_{512} \quad (3-3)
\]

Subscript \(i\) is the row number and can have a value between 1 and 480. \(p_j\) is the
gray level of one of the pixels in the row numbered \(i\).

The average of these sums is also calculated:

\[
\text{Avg}_{\text{row}} = \frac{\text{Sum}_1 + \text{Sum}_2 + \ldots + \text{Sum}_{480}}{480} \quad (3-4)
\]
The rows containing edges have higher values than the ones not containing any edges or only a few, as illustrated in Figure 3.8. Note that most edges are located around the characters (the license plate numbers).

All sums are compared to the average $\text{Avg}_{\text{row}}$ of those sums. If the sum $\text{Sum}_i$ of a particular row is greater than the average $\text{Avg}_{\text{row}}$, it is probably one of the rows containing the characters being searched. Otherwise it is considered as containing nothing but background pixels or some noise.

$$\text{Signal Row, if } \text{Sum}_i > \text{Avg}_{\text{row}}$$

$$\text{Row}_i = \{$$

$$\text{Background Row, otherwise}$$

In horizontal block separation, an image is partitioned to extract the most significant row blocks from the others based on the similarity of gray levels of the pixels of each row. All neighboring signal rows and background rows are combined into separately row blocks (see Figure 3.9).

The system searches for the signal row block with maximum number of rows through the procedure described below. These rows contain the characters being searched for.

1. Calculate the width (number of rows) in each signal row block by subtract the row number of the first row from the row number of the last row plus one.

$$\text{Width} = \text{No}_j - \text{No}_i + 1 \quad (3-5)$$

$\text{No}_j$ is the row number of the last row in this block, $\text{No}_i$ is the row number of the first row in this block. For example, if one block starts from row 220 and ends at row 312, the width of this row block is $312-220+1 = 93$.

2. Compare the widths for all signal blocks.
Figure 3.8 (a) The Row Histogram of a White-on-Black Image

Figure 3.8 (b) The Row Histogram of a Black-on-White Image
Figure 3.9 Row Block Grouping and Separating
Among these blocks, the one of maximum width is considered to be the one where the license plate characters are located and is called character row block. This row block is passed to the vertical scan for further separation. The preprocessing continues and performs a vertical scan (described below) on the row block selected through the process above.

3.3.4 Vertical Scan

3.3.4.1 Image Identification

Note that the gray levels of character pixels are higher than those of background for white-on-black images and lower for black-on-white image. The character row block separated contains the characters (the signal pixels). The average of its gray levels are compared to the average of the gray levels of the background. It can be found out whether the signals have higher or lower gray levels than the background. Based on that, the system knows what kind of image it is.

The average pixel value of the character row block of the original image is calculated by dividing the summation of the gray levels of all pixels in the row block by the number of pixels:

\[
\text{Sum}(n,m) = \sum_{h=n}^{m} \sum_{k=1}^{512} P_{h,k}
\]

\[
= (P_{n,1} + P_{n,2} + \ldots + P_{n,512}) +
(P_{n+1,1} + P_{n+1,2} + \ldots + P_{n+1,512}) +
\ldots +
(P_{m,1} + P_{m,2} + \ldots + P_{m,512})
\]

(3-6)

\[
\text{Avg}_s = \frac{\text{Sum}(i,j)}{(512^i j-i+1)}
\]

(3-7)
P is the pixel value, n is the lowest number and m is the highest number of a row block. Subscript i is the lowest number and j is the highest number of the character row block. \( \text{Sum}(i,j) \) and \( \text{Avg}_s \) are the summation and the average of all pixels values in the block.

The average pixel value of the background of the original image is also calculated in the same way. It is the summation of the gray levels of all pixels in the background blocks next to the row block divided by the number of pixels.

\[
\text{Avg}_b = \frac{\text{Sum}(a1,b1)+\text{Sum}(a2,b2)}{(512^*(b1-a1+b2-a2+2))}
\]  

Subscript \( a1 \) is the lowest number and \( bl \) is the highest number of a background row block on one side of the character row block. Subscript \( a2 \) is the lowest number and \( b2 \) is the highest number of the background row block on the other side. \( \text{Sum}_b \) ( = \( \text{Sum}(a1,b1) + \text{Sum}(a2,b2) \)) and \( \text{Avg}_b \) are the summation and the average of all pixels values in the background blocks.

The average \( \text{Avg}_s \) of the character row block is compared to the average \( \text{Avg}_b \) of the background blocks. If the row block has a higher average gray level than the background (\( \text{Avg}_s > \text{Avg}_b \)), the system determines that the original image is white-on-black, otherwise black-on-white.

To further separate the characters from the character row block, the system needs to locate where the signal pixels are. The characters have different gray levels from the gray levels of the background. The range of the gray levels of signal pixels are calculated to identify the signal pixels. The system sums all pixel values for each column in the row block:

\[
\text{Sum}_k = p_{k,i} + p_{k,i+1} + \ldots + p_{k,j}
\]
Subscript $k$ is the column number and is between 1 and 512. $p$ is the gray level, subscript $i$ is the lowest row number and $j$ is the highest row number of the character row block.

The average of these sums is also calculated.

$$\text{Avg}_k = \frac{\text{Sum}_k}{(j - i + 1)} \quad (3-10)$$

For black-on-white image, the gray levels of signal are lower than the gray levels of background. The more signal pixels a column has, the smaller its sum $\text{Sum}_k$ and average $\text{Avg}_k$ are. All the averages are compared to each other. The column with the minimum average $\text{Avg}_{\text{min}}$ should have the most signal pixels. The average $\text{Avg}_{\text{min}}$ is considered to be the average gray level $\text{Avg}$ of the character pixels. For white-on-black images, the average $\text{Avg}_{\text{max}}$ is considered to be the average gray level $\text{Avg}$ of the character pixels.

A copy of the original image is saved before the preprocessing and is used here. Additional thresholding process is used to convert the original image to a binary one. For white-on-black images, all pixels whose gray levels are close to or greater than the average gray level $\text{Avg}$ of the signal pixels are treated as signals. All other pixels are treated as background. For black-on-white images, all pixels whose gray levels are close to or less than the average are treated as signals. All the other pixels are considered as background pixels. A tolerance value is selected to make the conversion.

For black-on-white image:

$$P_i = \begin{cases} 
1, & \text{if } P_i < \text{Avg} + T \\
0, & \text{otherwise}
\end{cases}$$
For white-on-black image:

\[ P_i = \begin{cases} 
1, & \text{if } P_i > \text{Avg} - T \\
0, & \text{otherwise}
\end{cases} \]

\( P_i \) is the gray level of one pixel. \( T \) is the tolerance.

Because white-on-black images have higher contrast values than black-on-white (see requirement in Chapter 2), a lower tolerance value should be used for them. Based on the requirement in section 2.2 (1), a tolerance of 40 is used for white-on-black images, and 20 for black-on-white.

Also, the back-propagation neural network requires normalized input (such as 0 and 1). The values of each input should be within the range between 0 and 1. Converting the image into a binary one helps both the preprocessing and the neural network.

All pixels in the row block are separated into signal and background. After this processing, the system attempts to locate all objects in the image. Essentially, an object is anything other than background. The system looks for the foreground pattern by checking for whatever exceeds the pre-set background threshold. The system will perform a vertical scan to separate the characters from this row block.

3.3.4.2 Noise Filtering of The Row Block

Usually the filtered image contains some noise which can be reduced by noise filtering. The separation of the characters can generally be improved by smoothing the image to remove noise and fine detail that may produce ragged or broken regions.
This smoothing method is to replace each pixel value by the sum of the pixel brightness within a small area surrounding it. Given an N×M image f(x,y), the procedure is to generate a smoothed image g(x,y) whose gray level at every point (x,y) is obtained by averaging the gray-level values of the pixels of f contained in a predefined neighborhood of (x,y). Some improvement can be gained by using a weighted sum with the weights decreasing toward the edges of the kernel.

The convolution method is actually used for noise filtering with a different kernel. The kernel for noise filtering is:

\[
\begin{bmatrix}
1 & 1 & 1 \\
1 & 9 & 1 \\
1 & 1 & 1 \\
\end{bmatrix}
\]

Figure 3.10 Convolution Kernel For Noise Filtering

A sum is calculated based on Equation 3-1. Note that a thresholding has already been applied to the images, and all gray levels have been converted to a binary number (either 0 or 1). Signal pixels have a value of 1. All other pixels are background pixels and have a value of 0. The use of 1s around the 9 make all neighboring pixels play an equal role in noise filtering. The 9 in the center is greater than the sum of all eight 1s surrounding it. It makes the value of the pixel in the center more significant than that of the others (see Eq. 3-1).

The system tries to remove random noise, that is, remove the isolated signal pixels. In characters, most signal pixel should have at least three signal pixels around it. A signal pixel with none or a couple of neighbor signal pixels is fairly isolated from other signal pixels, and will be considered as a noise. All the
original background pixels will still be background pixels. Any signal pixel surrounded by less than three signal pixels is converted into a background pixel. This not only reduces noise, but more importantly it prevents the characters from touching each other.

If all the pixels in the 3x3 array (p₁ to p₉) are background pixels and have a value of 0, the sum will be 0 (see Eq. 3-1). If all the pixel in the 3x3 array (p₁ to p₉) are signal pixels and have a value of 1, the sum will be 17. If some of the pixel in the 3x3 array are signal pixels, the sum will have a value between 0 and 17. The sum should be greater than 11 so that the new pixel may qualify as a signal. To be classified as a signal pixel, it must be a signal pixel (in the center of the 3x3 array, p₅ = 1) and have at least three neighboring pixels that originally are also signal pixels. All background pixels as well as isolated signal pixels are considered as background (sum < 12). This process removes some random noise.

3.3.4.3 Vertical Block Separation

The image being processed here is the smoothed character row block (see Figure 3.11). The gray level of each pixel in this image is either 0 or 1. The system identifies all the column blocks in a process similar to the one in the horizontal scan. The columns where the characters are located need to be determined. A summation of gray levels of pixels is calculated for each column of the block, as shown in Equation 3-11. Remember that all pixels have been through the thresholding process and have a value of 0 or 1.

\[
\text{Sum}_h = p_i + p_{i+1} + \ldots + p_k + \ldots + p_j \quad (3-11)
\]
Subscript \( h \) is the column number and is between \( i \) and \( j \). Subscript \( i \) is the lowest number and \( j \) is the highest number of the character row block. \( p_k \) is the gray level of one of the pixels in the column numbered \( k \).

If the sum \( \text{Sum}_h \) of a particular column is greater than one, it is probably one of the columns containing the characters being searched for. Otherwise it is considered as containing nothing but background pixels or some noise.

\[
\text{Column}_h = \begin{cases} 
\text{Signal Column,} & \text{if } \text{Sum}_h > 1 \\
\text{Background Column, otherwise} & 
\end{cases}
\]

All neighboring signal columns and background ones are combined separately into several signal and background blocks. This process is similar to the row block grouping shown in Figure 3.9. The system checks each signal block to eliminate any block not containing characters. If a signal block meets any of the following conditions, it is considered as background and is discarded by the system:

1) The width of the block is less than 5 or greater than 100. A character of this size is too small or large to be recognized.

2) The width of a block is less than one-third or greater than three times the average width of all blocks. This block is too narrow or too wide to be a character block when compared to other blocks.

3) More than 90% of the pixels in the block are signal pixels. Because for white-on-black images, the gray levels of the background surrounding the license plate may be close to the gray levels of characters. These background pixels are classified as signals because they have similar gray levels to the signals. This block is not character block and should not be further recognized.
Figure 3.11 (a) Character Row Block of a White-on-Black Image

Figure 3.11 (b) Character Row Block of a Black-on-White Image
4) The height of the block is less than 80% of that of the row block. This condition is added because there is usually an additional mark in a license plate, which is about 80% or less of the characters' height (see the mark between the characters P and 4 of the license plate in Figure 3.11(a)).

5) The signals in the block stretch outside the row block (see the license plate edge in the Figures 3.11(a) and 3.3(b)). This is the edge of the license plate.

The system only recognizes the characters not having the above features. The character blocks identified are shown in Figure 3.12.

3.3.4.4 Scaling The Character Blocks

Since the sizes of the characters may be different, and the neural network is trained to recognize characters of one specific size, they must be scaled to a fixed size before they are fed into the neural network.

5x7 is the minimum size to represent any matrix font, and 7x9 can represent the fine details of characters better [Mark S. Sanders, Ernest J. McCormick, 1989]. The system is mainly designed to recognize fairly clean license plates, although the images of license plates may have some noise. Because the system is supposed to recognize license plates with different fonts, characters of 7x9 size are generated from the preprocessing. The 7x9 size is used to represent the details of the characters. The system should convert the character blocks into the 7x9 size.

Based on the discussions of the conditions of image acquisition in Chapter 2, the scaling is a matter of reducing the original image into a smaller one. Many pixels of the character blocks from the preprocessing are used to produce one pixel of the 7x9 character array. The system goes through the 7x9 output space a pixel at a time and transforms the source image to the output
Figure 3.12 (a) Character Blocks of a White-on-Black Image

Figure 3.12 (b) Character Blocks of a Black-on-White Image
pixels. The generation for each pixel of the output array from the character blocks guarantees that every pixel in the output arrays is given a value.

The scaling factor is first calculated to determine which pixels are involved for creating the output pixel. When generating the output pixel, the proportion of signal pixels to the total pixels in each of the horizontal and vertical directions is calculated. If half or more are signal pixels, an integer one is generated for this output pixel, representing it as foreground. Otherwise a 0 is generated. The character scaling is illustrated in Figure 3.13. A 7x9 array of ones and zeros is generated for each character. The preprocessing ends after one 7x9 array is generated for each character. These output arrays are fed into the neural network for final recognition.
Scaling Factor:

Height: $95 / 9 = 10.56$
Width: $33 / 7 = 4.71$

Each Area: (round off number)

Height: 10 or 11
Width: 4 or 5

Summation

$$\text{Sum} = P_{1,1} + P_{1,2} + \ldots + P_{10,4}$$

Total No = 4*10 = 40
Half No = 40/2 = 20

Output Array

\{ 
1, if Sum > 19 
0, if Sum < 20 
\}

Figure 3.13 Character Scaling
3.4 Neural Networks

3.4.1 Neural Network Technology

Neural networks are briefly discussed in Chapter One. Neural networks mimic the abilities of biological systems using large numbers of simple, interconnected artificial neurons. Artificial neurons are the processing elements of neural networks. They take in information from network input or other neurons, perform simple operations on the data, and pass the results to other artificial neurons.

3.4.1.1 Neural Network Architecture

An artificial neuron accepts an incoming signal or set of signals from other neurons or network input, sums the signals received, and passes the sum through a function called activation or transfer function. The output value is the activation of the neuron. The activation is bounded, usually between 0 and 1 or -1 and 1.

The most distinguishing feature of artificial neurons is their transfer function. This function specifies how the neuron scales its response to incoming signals. The transfer function is typically a threshold, or a continuous function.

The neurons of networks with threshold-logic function produce binary output either 0 or 1 by applying a simple thresholding function. If the summed input is greater than or equal to the neurons' threshold, the activation is 1, otherwise the activation is 0. These networks are easy to implement, but are very limited in their capabilities. Network learning could not be substantially improved without the continuous transfer function [Alianna J. Maren, Dan Jones, and Stanley Franklin, 1990].

Although a single neuron can perform certain simple functions, the power of neural computation comes from connecting neurons into networks. The
neurons of neural networks are connected to each other, and have a known strength on each connection. The values of these strengths are called connection weights. The newly activated neurons send their signal to the neurons to which they are connected. The signals sent out are multiplied by their connection weights. The purpose of these connection weights is to send different signals to neurons. This allows neural networks to produce different and useful patterns of output in response to input stimuli.

The arrangement and connection of neurons distinguish different types of network architectures. A typical neural network might have different layers of neurons. Some accept the input, others process it, and another layer generates and stores the output. Neural networks can be classified as single-layer, or multilayer by the number of layers in the networks. The single-layer network is a group of neurons arranged in a layer. The multilayer networks have been proven to have capabilities beyond the single-layer networks [Alianna J. Maren, Dan Jones, and Stanley Franklin, 1990].

The neurons of one layer in a multilayer network perform their processing in one step conceptually. This step is usually emulated in computer program and the operations are actually performed sequentially in most implementations. The neurons of different layers operate in a serial sequence. The neurons in the first layer access the input, process and pass the signal to the second layer. The signals are then processed and passed to the third layer. The resulting signals in the last layer (called the output layer) are the output of the neural network. The activation of one or more neurons in the output layer is interpreted as a pattern associated with the input.

3.4.1.2 Multilayer Feedforward Networks
In multilayer feedforward networks, all neural signals propagate in a forward direction through the network layers. The back-propagation network is one of the feedforward networks. Even though the results of forward signals may be used to correct weights of previous neurons, the operations of the network are strictly feedforward.

Feedforward networks are typically the networks of choice for pattern classification applications [Alianna J. Maren, Dan Jones, and Stanley Franklin, 1990]. They can learn to generalize the important distinguishing characteristics of their input patterns. Feedforward networks with more than two layers are the optimal choice when generalization or pattern recognition is desired. These networks all use supervised learning, and they are taught to classify input into several categories.

Training algorithms are categorized as supervised and unsupervised. Supervised training requires the pairing of each input vector with a target vector representing the desired output. These two vectors are called a training pair. Usually a network is trained over a number of such training pairs called a training set. An input vector is applied, the output of the network is calculated and compared to the corresponding target vector, and the difference (error) is fed back through the network and weights are changed according to an algorithm that tends to minimize the error. The vectors of the training set are applied sequentially, and errors are calculated and weights are adjusted for each vector, until the error for the entire training set is at an acceptably low level (0.1 for this system).

Unsupervised requires no target vector for the output, and there is no comparison to predetermined ideal response. The training set consists solely of input vectors. The training algorithm modifies network weights to produce output vectors that are consistent. That is, both application of one of the training
vectors or application of a vector that is sufficiently similar to it will produce the same pattern of outputs. The training process extracts the features of the training set and groups similar vectors into classes. Applying a vector from a given class to the input will produce a specific output vector.

Multilayer feedforward networks include the Perceptron, the ADALINE and MADALINE networks, the back-propagation network, the Boltzmann machine and the Cauchy machine [Alianna J. Maren, Dan Jones, and Stanley Franklin, 1990]. These networks differ primarily in the way in which they learn. The Perceptron, the ADALINE and MADALINE networks, and the back-propagation network all have similar learning methods. The Boltzmann machine, and its related derivative, the Cauchy machine have a different basis for their learning rules.

The Perceptron, developed by F. Rosenblatt, was the first neural network to emerge. It substantiated the concept of the artificial neuron which is still used today, where each neuron computes a weighted sum of its inputs, and passes this sum into a non-linear thresholding function [Alianna J. Maren, Dan Jones, and Stanley Franklin, 1990]. The original Perceptron model was strictly feedforward and could be extended up through multiple levels. The learning algorithm for the Perceptron allowed it to distinguish between classes of input for some problems.

The ADALINE network was developed by Bernie Widrow shortly after Rosenblattt developed the Perceptron [Alianna J. Maren, Dan Jones, and Stanley Franklin, 1990]. The back-propagation network is an outgrowth of earlier work on Perceptrons. The major difference between the two networks is that the learning law for the back-propagation network is substantially more powerful than the learning law for the ADALINE network [Alianna J. Maren, Dan Jones, and Stanley Franklin, 1990]. The key distinguishing characteristic of the back-
propagation network is that it forms a mapping from a set of input to a set of output using features extracted from the input pattern. The back-propagation network requires at least three layers of nodes, whereas the Perceptron and the ADALINE could each be instantiated with two layers. The layers for a back-propagation network are typically referred to as input, hidden, and output. The nodes in the hidden layer(s) of the network learn response to features found in the input and generalize [Alianna J. Maren, Dan Jones, and Stanley Franklin, 1990].

The learning method of the Boltzmann machine and its refinement, Cauchy machine, draws on an optimization approach which comes from the statistical modelling of thermodynamic process [Alianna J. Maren, Dan Jones, and Stanley Franklin, 1990]. They are structurally and dynamically similar to the back-propagation network. There are several reasons why back-propagation network is used more often then the Boltzmann machine. The performance of the two networks is very similar. But the back-propagation network is easier to learn and use. In contrast, the Boltzmann machine comes out of statistical mechanics and is hard to understand. Its learning rule takes much longer to write down, is more complex, and takes more time when training a network than the back-propagation learning rule [Alianna J. Maren, Dan Jones, and Stanley Franklin, 1990]. As a result, applications of the back-propagation method are exploding, while interest in the Boltzmann machine has dwindled [Alianna J. Maren, Dan Jones, and Stanley Franklin, 1990].

This selection of a network architecture is based on the comparison of application requirements to network architecture capabilities. Based on the above discussion and the problem at hand, supervised back-propagation network was selected and used for recognition.
3.4.2 Design of The Neural Network

Before a neural network is actually implemented, some characteristics have to be selected. They include the model, size, training method, number of nodes, etc. The network design comprises of several steps to determine the type of nodes, size and connectivity of the network layers, and the learning algorithm. Decisions to be made at this point include the number of layers, the size of each layer, the type of inputs, the type of outputs to expect, and how each layer should be connected.

A back-propagation network must have an input layer, output layer, and one or more hidden layers. Hidden layers act as layers of abstraction [Alianna Maren, 1990; David Bailey and Donna Thompson, 1990], pulling features from inputs. Increasing the number of sequential hidden layers increases the processing power of the neural network. But adding hidden layers increases both the time and the number of training examples to train the network properly [David Bailey and Donna Thompson, 1990]. For a classification problem, where the output node with the greatest activation will determine the category of the input pattern, one hidden layer will most likely be sufficient [Alianna J. Maren, Dan Jones, and Stanley Franklin, 1990]. One hidden layer is used in the system, and is capable of classifying the characters.

After selecting the number of layers for the network, the size (in number of nodes) of each layer can be determined. The input layer presents data to the network. The number of input nodes should be equal to the pattern size, that is, 7x9 or 63.

Determining the proper number of nodes for the hidden layer is difficult [Alianna J. Maren, Dan Jones, and Stanley Franklin, 1990, David Bailey and Donna Thompson, 1990], and is often achieved through experimentation. If there
are too few hidden neurons, the network probably won't train at all [Russell C. Eberhart and Roy W. Dobbins, 1990]. Too few nodes in the hidden layer impairs the network and prevents it from ever correctly mapping inputs to outputs. On the other hand, too many neurons tend to create a network that has memorized all the training patterns presented without extracting their features [David Bailey and Donna Thompson, 1990]. Thus, when presented with new patterns, the network is unable to process them properly because it has not discovered the underlying principles of the problem. In other words, it doesn't generalize well. A suitable size for one hidden layer is about 75% of the size of the input layer [David Bailey and Danna Thompson, 1990]. Since the input layer has 63 neurons, the hidden layer will have 47 (75% x 63) neurons.

The output layer responds to the presence or absence of features in the pattern. The number of outputs from the neural network is determined by the type of network output. Because the back-propagation network generates outputs of 1 and 0, the combination of the 1s and 0s should be able to represent all characters. The system is designed to recognize all numbers and letters, and there are 36 of them, 26 capital letters and 10 numbers. The binary combination of five output neurons can only represent a maximum $2^5$ (32) characters or conditions. The binary combination of six output neurons can represent up to $2^6$ (64) characters or conditions. The number of output neurons will be 6 (for the six-digit codes, see Chapter Four).

Generally, fully connecting adjacent layers within multi-layer networks is best [David Bailey and Donna Thompson, 1990]. Most feedforward networks are fully connected between layers [Alianna J. Maren, Dan Jones, and Stanley Franklin, 1990]. This approach provides the most flexibility when the training algorithm is searching for suitable weight settings. The training algorithm can
Each connection has a weight

Input either 1 or 0

Character Array

Total 63

Input Layer

Hidden Layer

Output Layer

Total 47

Total 6

Figure 3.14  Structure of Neural Network
nullify unnecessary links by setting their weights to zero. The final network used in the system is described in Figure 3.14.

Each neuron of the input layer is connected to every neuron of the hidden layer. Each neuron of the hidden layer is connected to every neuron of the output layer. A weight is associated with each connection. A weight actually is a floating-point number which is initialized to a small value and is adjusted as the network is being trained.

3.4.3 Training The Neural Network

3.4.3.1 Initializing

The objective of training the network is to adjust the weights so that application of a set of inputs produces the desired set of outputs. Before starting to train the network, a training set has to be prepared. The training set consists of all the inputs and their target outputs used by the neural network's learning algorithm to properly adjust the connection weights between nodes. The more complete the training set, the more accurately the network should perform. Gathering as much training data as possible within practical limits is a good idea. A good training set should be a representative of the data set [David Bailey and Donna Thompson, 1990]. One measurement of the data's representativeness is the breadth of the problem the training cases cover, including the different types and fonts of letters and numbers used in this study.

To prepare the data for the use by the neural network, preprocessing must be performed. Preprocessing includes filtering, transforming inputs into the integer type, the 7x9 arrays and converting the data into binary vectors with 63 elements. Before feeding the training data into the network, the data set should be normalized [Alianna J. Maren, Dan Jones, and Stanley Franklin, 1990; David
Bailey and Donna Thompson, 1990]. That is, the value of each input should be a binary number of 0 or 1. This is done in the preprocessing stage.

Before the training starts, the network must be initialized. The initialization is applied to the setup of the network, and the weights of all connections of the network. If all the weights are initialized to zero, all outputs of the neurons and the errors will be 0, as can be seen in the next sections. The network may be impossible to train [Russell C. Eberhart, and Roy W. Dobbins, 1990]. All the weights should be set using a small random number [Philip D. Wasserman, 1989; Russell C. Eberhart, and Roy W. Dobbins, 1990], between -0.5 and 0.5 in this system.

3.4.3.2 Generating Output With Forward Pass

Training a back-propagation network consists of two passes: forward pass and backward pass. The inputs to the network are a set of 7x9 matrices of binary numbers generated by the preprocessing (for detail, see Chapter Four). In the forward pass, the numbers of the matrices are input to the network and the network output are generated. The input neurons simply distribute the signal along multiple paths to the hidden layer neurons. The output of an input neuron is exactly equal to the input. The same output, multiplied by the connection weights between the input and hidden layers, will be passed to each neuron in the hidden layer.

The input to a hidden neuron is calculated as the sum of all values from all connections coming into this neuron. The signal presented to a hidden layer neuron due to one single connection is just the output value of the input neuron times the value of the connection weight. This is described in the equation 3-12 [Philip D. Wasserman, 1989; Russell C. Eberhart, and Roy W. Dobbins, 1990]:

...
The letter $I$ represents the net input to the hidden neuron, $W$ denotes a connected weight, and $O$ the output from input layer. The subscript $j$ refers to neuron $j$ in the hidden layer, the numbers 1 to $n$ represent the neurons in the input layer.

An activation function called the transfer function is applied to the input of a hidden neuron to produce its output. The transfer function is also used in calculating the outputs for the output neurons. The back-propagation learning method requires that the transfer function for each neuron be a continuous function and possess a derivative at all points [Philip D. Wasserman, 1989; Alianna J. Maren, Dan Jones, and Stanley Franklin, 1990]. This function should be asymptotic for both infinitely large positive and negative values of the independent variable (input), the weighted sum of the input into the neuron.

Because of its proven performance, the additive sigmoid neuron is used as the transfer function [Alianna J. Maren, Dan Jones, and Stanley Franklin, 1990; Russell C. Eberhart, and Roy W. Dobbins, 1990], as described in the equation 3-13 [Philip D. Wasserman, 1989; Russell C. Eberhart, and Roy W. Dobbins, 1990].

$$I_j = W_{1j}O_1 + W_{2j}O_2 + \ldots + W_{nj}O_n \quad (3-12)$$

$$O_j = f(I_j) = 1 / (1 + \exp(-I_j)) \quad (3-13)$$

The letter $O$ represents the output of the neuron, the letter $I$ is the input calculated by Eq 3-12. The subscript $j$ refers to neuron $j$ in the hidden layer.

The output, after being put through the sigmoid function, is limited to values between 0 and 1, as shown in Figure 3.15. The nonlinear nature of this
Figure 3.15  The Sigmoid Transfer Function

\[ f(I) = \frac{1}{1 + e^{\exp(-I)}} \]
sigmoid transfer function plays an important role in the performance of the neural network. Multi-layer networks have greater representational power than single-layer networks only if a nonlinearity for transfer function is introduced [Alianna J. Maren, Dan Jones, and Stanley Franklin, 1990; Russell C. Eberhart, and Roy W. Dobbins, 1990]. For the input of zero, the output is 0.5. For large negative input values, the neuron output approaches 0; for large positive values, it approaches 1. In this way, large signals can be accommodated by the network without saturation, while small signals are allowed to pass through without excessive attenuation.

After the outputs of all hidden layer neurons have been calculated, the net input to each output layer neuron and the outputs of the output neurons are calculated in an analogous manner. The outputs of the output neurons represent the output of the network.

The set of calculations that results in obtaining the output state of the network is carried out in exactly the same way during the training phase as they are carried out during the testing or running. The testing or running mode just involves presenting an input set to the input neurons and calculating the resulting output state in one forward pass.

3.4.3.2 Error Correction With Backward Pass

The system performs well only after it is taught. The learning capability is built into the system by training the network. In the backward pass, the difference between the actual and desired outputs generates an error signal. It is used to adjust the weights of the network to produce the desired output. During the training phase, the feed-forward calculation, combined with backward error propagation and the weight adjustment calculations, represents the network's learning.
The output error is the difference between the output value an output neuron is supposed to have and the value it actually has. The pattern error is the error of one character, and is calculated as the sum of the errors of all output neurons for that pattern. The network error is the sum of all the pattern errors, and its minimization is the goal of the training. Minimizing the network error with respect to the hidden neurons is the key to the back-propagation method. Equation 3-14 presents the formula to calculate the output error of an output neuron [Philip D. Wasserman, 1989; Russell C. Eberhart, and Roy W. Dobbins, 1990].

Output Error:

\[ E_j = T_j - O_j \]  

(3-14)

Pattern Error \[ = \text{Pat}E_i = |E_1| + |E_2| + \ldots + |E_n| \]

Network Error \[ = \text{Pat}E_1 + \text{Pat}E_2 + \ldots + \text{Pat}E_m \]

\( E \) is the neuron error, \( O \) is the output of the neuron, and \( T \) is the output value the neuron is supposed to have. The subscript \( j \) refers to neuron \( j \) in the output layer. The subscript \( i \) refers to pattern \( i \) in the training set. \( N \) is the total number of the output neurons, and \( m \) is the total number of training patterns. \( \text{Pat}E \) is the pattern error.

The neuron error is propagated back from the output layer to the hidden layer, and the weights are adjusted after each training pattern (a 7x9 matrix and the desired 6-digit binary output) is presented to the network. The changes in weights depend on the propagated signal error of the pattern. The propagated error for each output neuron is first calculated. The propagated errors of the
output neurons are defined by Equation 3-15 [Philip D. Wasserman, 1989; Russell C. Eberhart, and Roy W. Dobbins, 1990].

\[ PE_j = f'(I_j) (T_j - O_j) \]  \hspace{1cm} (3-15)

\( T_j, \ O_j, \) and \( j \) are the same as in equation 3-14. \( PE_j \) denotes the propagated error. \( f'(I_j) \) is the first derivative of the transfer function of the input \( I_j \) (for \( f(I_j) \) see Equation 3-13). Remember from Equation 3-13 that the transfer function is a sigmoid function, and the output of the output neuron is a function of its input \( (O_j = f(I_j)) \). Therefore the first derivative is

\[ f'(I_j) = \left( \frac{1}{1 + \exp(-I_j)} \right)' \]
\[ = \exp(-I_j) / (1 + \exp(-I_j))^2 \]

because

\[ O_j = 1 / (1 + \exp(-I_j)) \]
\[ \exp(-I_j) = (1 - O_j) / O_j \]

therefore

\[ f'(I_j) = \left( (1 - O_j) / O_j \right) \cdot O_j^2 \]
\[ = O_j \cdot (1 - O_j) \]

Note that the derivative is represented as a function of the output \( O_j \) rather than the input \( I_j \) [Philip D. Wasserman, 1989; Russell C. Eberhart, and Roy W. Dobbins, 1990]. The derivative may be calculated using the input \( I_j \), but at the cost of calculating the exponential function \( \exp \). The diagram of the derivative \( f'(I_j) \) is shown in Figure 3.16.
When $I=0$: $0=0.5$, so $f'(I) = 0.25$

Figure 3.16  The Derivative of the Sigmoid Transfer Function
The equation for calculating the propagated error of each output neuron should not be complicated and it is given in Equation 3-16 [Philip D. Wasserman, 1989; Russell C. Eberhart, and Roy W. Dobbins, 1990].

\[ PE_j = O_j \times (1 - O_j) \times (T_j - O_j) \]  \hspace{1cm} (3-16)

The changing of the weights depends on the propagated errors calculated in Eq. 3-16. The calculation of new weights for the connections from the output neurons to the hidden neurons may be described by Equations 3-17 and 3-18 [Philip D. Wasserman, 1989; Russell C. Eberhart, and Roy W. Dobbins, 1990]. All weights for all connections between the hidden layer and the output layer are adjusted using equations 3-17 and 3-18.

\[ WC_{ij}(\text{new}) = LR \times PE_j + M \times WC_{ij}(\text{old}) \]  \hspace{1cm} (3-17)

\[ W_{ij}(\text{new}) = W_{ij}(\text{old}) + WC_{ij}(\text{new}) \]  \hspace{1cm} (3-18)

"New" represents the current training pass, and "old" represents the previous one. \( W_{ij} \) is the weight of the connection between neurons i and j. \( WC_{ij} \) is the weight change, and the \( W_{ij}(\text{old}) \) (weight change of the previous pass) is saved in the previous pass. It is used to calculate the weight change of the current pass. The new weight is calculated based on the weight change of the current pass.

\( LR \) is the learning factor, \( M \) is the momentum. They are the main parameters affecting the training and performance of neural networks. The learning factor has a significant effect on training speed and efficiency. The magnitude of the weight change is controlled by the learning factor. The higher it is, the faster the network is able to train. However, a high learning factor makes
the network subject to instability, and increases the danger of saturation - a state in which the network won't show any improvement [Alianna J. Maren, Dan Jones, and Stanley Franklin, 1990; Russell C. Eberhart, and Roy W. Dobbins, 1990]. A small learning factor stabilizes the training, but results in slower learning and increases possibilities of failing to improve the training [Alianna J. Maren, Dan Jones, and Stanley Franklin, 1990]. A value of learning factor between 0.01 and 1.0 is typical [Philip D. Wasserman, 1989]. It has a value of 0.3 in this system.

Momentum is used to speed up learning and stabilize training. Use of momentum helps to prevent high-frequency variation and large changes in training, thus stabilizing the network [Alianna J. Maren, Dan Jones, and Stanley Franklin, 1990]. The momentum prevents large weight changes and reduces the risk of saturation while still permitting fast learning rates. The momentum factor is commonly set at around 0.9 [Philip D. Wasserman, 1989].

After the new weights on the connections from the hidden neurons to the output neurons have been calculated, the new weights connecting from the input neurons to the hidden neurons are calculated. It is not simple to figure out the error terms for the hidden neurons. The above equations (Eq. 3-14 and 3-15) can not be used here because there is no target value for each hidden neuron. Instead, the errors are produced by propagating the output errors back to the hidden layer. Back-propagation network trains the hidden layers by propagating the output error back through the network layer by layer, adjusting weights at each layer [Philip D. Wasserman, 1989]. The error propagation is illustrated in Figure 3.17. The error each hidden neuron receives is the sum of the propagated errors from all output neurons multiplied by their connected weights, as shown in Equation 3-19 [Philip D. Wasserman, 1989; Russell C. Eberhart, and Roy W. Dobbins, 1990].
Weight change for this connection after error is propagated to hidden neuron j

These weights are changed first

Figure 3.17  Error Back-propagation
The way to calculate propagated errors for the hidden neurons is the same as for the output neurons. The propagated error of each hidden neuron can be defined as in Equation 3-20 [Philip D. Wasserman, 1989; Russell C. Eberhart, and Roy W. Dobbins, 1990].

\[ E_i = W_{i1} \cdot PE_1 + W_{i2} \cdot PE_2 + \ldots + W_{in} \cdot PE_n \quad (3-19) \]

The weight changes for the connections from the input layer to the hidden layer are calculated in a similar manner as those from the hidden layer to the output layer [Philip D. Wasserman, 1989; Russell C. Eberhart, and Roy W. Dobbins, 1990].

\[ PE_i = f'(l_j) \cdot E_i = O_i \cdot (1 - O_i) \cdot E_i \quad (3-20) \]

The above process is repeated for each pattern in the training set. The network error is summed for each pass during training. The goal of the training process is to minimize the network error over all training patterns. It does this by performing a gradient descent procedure on error surface [Philip D. Wasserman,
When the network error is less than the acceptable total error (0.1), the training is terminated.

3.4.3.3 Improving The Training

Back-propagation employs a type of gradient descent; that is, it follows the slope of the error surface downward, constantly adjusting the weights toward minimum [Philip D. Wasserman, 1989, Russell C. Eberhart, and Roy W. Dobbins, 1990]. There is a chance that the back-propagation learning technique does not lead to a set of weights which are converging as training is increased [Alianna J. Maren, Dan Jones, and Stanley Franklin, 1990; Philip D. Wasserman, 1989, Russell C. Eberhart, and Roy W. Dobbins, 1990]. The network can get trapped in a local minimum and cannot get out of it when there is a much deeper minimum nearby. Non-convergence occurs when the system moves in a direction which produces a lower total error than it previously had, but which does not yield global error minimization - the bottom of the error surface. The use of the momentum helps to avoid this situation.

Note that as the activation (output) of a neuron approaches either zero or one, the derivative of the transfer function approaches zero. This is desirable and it makes the network more stable, because what is needed is to have the output neurons of the network produce values close to one of the two stable states (0 or 1).

The activation approaches zero for a large positive input, or one for a large negative input (see Eq. 3-13 and Figure 3.15). If the output of an output neuron approaches one when a value of zero is expected, or approaches zero when a value of one is expected, it produces both a derivative and a propagated error close to zero, based on Equation 3-16. Because the weight change is proportional to this derivative and its propagated error (see Eq. 3-17 & 3-18), the
weight change will be close to zero. The weight changes more slowly than is desired, and this may cause difficulties in training the network [Alianna J. Maren, Dan Jones, and Stanley Franklin, 1990].

This problem may be solved by using a differential step-size calculation instead of the derivative of the transfer function in the output layer only [R. Chen, P. Mars, 1990]. The output error is calculated by dropping the derivative term in calculating the propagated error for the output neuron (see Eq. 3-16). This increases the propagated error when the network makes a big mistake, and decreases the propagated error when the output error is small. The modified equation for calculating the propagated error for the output neurons is shown in Equation 3-23 [R. Chen, P. Mars, 1990; Don Tveter, 1991].

\[ PE_j = C \cdot (T_j - O_j) \]  \quad (3-23)

\( C \) is a constant, and has a value of 0.2 in the system.

The larger the output error, the larger the propagated error and the weight change are (see Eq. 3-23 and 3-17), and the more the network will learn from it. When the output error approaches zero, the propagated error and weight change are always close to zero and stabilize the network.

The errors become very small when they are propagated back to the hidden layer from the output layer. It is very difficult or even impossible to determine a range for the errors of hidden neurons. It is difficult to calculate the weight changes for the hidden neurons based only on the propagated errors. The calculation of propagated errors for the hidden neurons remains the same [R. Chen, P. Mars, 1990; Don Tveter, 1991].
The details of the system design have been given. The design of the system and its use can be justified only after it has been tested. A final test was conducted, and the results are discussed in the next chapter.
Chapter Four  System Test

A license plate recognition system has been developed. The techniques being used can also be applied for other pattern recognition systems. Before finishing this project, a final test was conducted to validate the system. It is described in the following sections.

4.1 Training The System

To test the preprocessor and train the neural network, 24 pictures of license plates with different characters and fonts were taken. All those pictures were taken under the conditions as described in the requirements in chapter 2. The 24 license plate pictures included 15 black-on-white and 9 white-on-black license plates. The 24 license plate codes are listed in Figure 4.1. The photocopies of those pictures are listed in the Appendix B.1.

The 24 license plate pictures are first compressed from floating-point numbers into gray levels between 0 and 255. These CapCalc pictures are then input into the system. The system performs the preprocessing and produces the 7x9 character arrays. The character arrays are used in the training set to train the neural network.

When training the network, a training set should be prepared and presented to the system. The character arrays generated by the preprocessor are combined in a data file as the training set. The training set must begin with the number of patterns in this training set, followed by all the patterns. Each pattern consists of a 7x9 array of 0 and 1 representing one letter or number, and its six-digit code. The data file includes a header, character arrays and their codes. The format of the header should present the following:
<table>
<thead>
<tr>
<th>No.</th>
<th>License Plate</th>
<th>Type of Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>1)</td>
<td>612 4TM</td>
<td>Black-on-White</td>
</tr>
<tr>
<td>2)</td>
<td>CQM 902</td>
<td>Black-on-White</td>
</tr>
<tr>
<td>3)</td>
<td>XOP 479</td>
<td>White-on-Black</td>
</tr>
<tr>
<td>4)</td>
<td>CNP 426</td>
<td>Black-on-White</td>
</tr>
<tr>
<td>5)</td>
<td>CLM 754</td>
<td>White-on-Black</td>
</tr>
<tr>
<td>6)</td>
<td>8708 TN</td>
<td>White-on-Black</td>
</tr>
<tr>
<td>7)</td>
<td>ABD 24</td>
<td>Black-on-White</td>
</tr>
<tr>
<td>8)</td>
<td>938 170</td>
<td>Black-on-White</td>
</tr>
<tr>
<td>9)</td>
<td>BCX 114</td>
<td>White-on-Black</td>
</tr>
<tr>
<td>10)</td>
<td>BKM 965</td>
<td>Black-on-White</td>
</tr>
<tr>
<td>11)</td>
<td>DR 4361</td>
<td>White-on-Black</td>
</tr>
<tr>
<td>12)</td>
<td>2FIK 144</td>
<td>Black-on-White</td>
</tr>
<tr>
<td>13)</td>
<td>3YZ 406</td>
<td>White-on-Black</td>
</tr>
<tr>
<td>14)</td>
<td>605 SRP</td>
<td>Black-on-White</td>
</tr>
<tr>
<td>15)</td>
<td>4148 W1</td>
<td>White-on-Black</td>
</tr>
<tr>
<td>16)</td>
<td>SGX 829</td>
<td>Black-on-White</td>
</tr>
<tr>
<td>17)</td>
<td>693 URA</td>
<td>Black-on-White</td>
</tr>
<tr>
<td>18)</td>
<td>754 IBX</td>
<td>Black-on-White</td>
</tr>
<tr>
<td>19)</td>
<td>PABLITO</td>
<td>Black-on-White</td>
</tr>
<tr>
<td>20)</td>
<td>CXE 497</td>
<td>Black-on-White</td>
</tr>
<tr>
<td>21)</td>
<td>RWD 594</td>
<td>Black-on-White</td>
</tr>
<tr>
<td>22)</td>
<td>FJZ 282</td>
<td>Black-on-White</td>
</tr>
<tr>
<td>23)</td>
<td>FWH 930</td>
<td>White-on-Black</td>
</tr>
<tr>
<td>24)</td>
<td>BTV 776</td>
<td>White-on-Black</td>
</tr>
</tbody>
</table>

Figure 4.1 License Plate for Training
number of 7x9 patterns: 145

This line gives the number of 7x9 patterns in this data file. Following the header are all character arrays used to train the network. The 7x9 character arrays are arranged in the order that the pictures are preprocessed and the character arrays are generated. Each character is a 7x9 array consisting 1 and 0. 1 stands for a signal and 0 stands for the background. Each array is followed by a code of six numbers, and the numbers are either 1 or 0. Each code identifies a character to be recognized, and could be any combination of 1's and 0's as long as it is unique for each character.

The six-digit codes that are assigned to the characters could be considered as six-digit binary numbers starting from 1 and increasing by 1 for the next character. For example, the first character 'A' has a code of 000001 (decimal number 1), and the second character 'B' has a code of 000010 (decimal number 2). The character '2' has a code of 011101 (decimal 29), and the character '3' has a code of 011110 (decimal 30). The following is a sample of a character array for the number 9 and its six-digit code. All the characters and their codes are listed in Figure 4.2.

```
0 1 1 1 1 1 0
1 1 0 0 0 1 1
1 1 0 0 0 1 1
1 1 0 0 0 1 1
0 1 1 1 1 1 1
0 0 0 0 1 1 1
0 0 0 0 1 1 0
0 0 1 1 1 0 0
0 1 1 0 0 0 0
1 0 0 1 0 0
```
<table>
<thead>
<tr>
<th>Character</th>
<th>Code</th>
<th>Occurrence</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>000001</td>
<td>3</td>
</tr>
<tr>
<td>B</td>
<td>000010</td>
<td>6</td>
</tr>
<tr>
<td>C</td>
<td>000011</td>
<td>5</td>
</tr>
<tr>
<td>D</td>
<td>000100</td>
<td>3</td>
</tr>
<tr>
<td>E</td>
<td>000101</td>
<td>1</td>
</tr>
<tr>
<td>F</td>
<td>000110</td>
<td>3</td>
</tr>
<tr>
<td>G</td>
<td>000111</td>
<td>1</td>
</tr>
<tr>
<td>H</td>
<td>001000</td>
<td>1</td>
</tr>
<tr>
<td>I</td>
<td>001001</td>
<td>3</td>
</tr>
<tr>
<td>J</td>
<td>001010</td>
<td>1</td>
</tr>
<tr>
<td>K</td>
<td>001011</td>
<td>2</td>
</tr>
<tr>
<td>L</td>
<td>001100</td>
<td>2</td>
</tr>
<tr>
<td>M</td>
<td>001101</td>
<td>4</td>
</tr>
<tr>
<td>N</td>
<td>001110</td>
<td>2</td>
</tr>
<tr>
<td>O</td>
<td>001111</td>
<td>1</td>
</tr>
<tr>
<td>P</td>
<td>010000</td>
<td>4</td>
</tr>
<tr>
<td>Q</td>
<td>010001</td>
<td>1</td>
</tr>
<tr>
<td>R</td>
<td>010010</td>
<td>4</td>
</tr>
<tr>
<td>S</td>
<td>010011</td>
<td>2</td>
</tr>
<tr>
<td>T</td>
<td>010100</td>
<td>3</td>
</tr>
<tr>
<td>U</td>
<td>010101</td>
<td>1</td>
</tr>
<tr>
<td>V</td>
<td>010110</td>
<td>1</td>
</tr>
<tr>
<td>W</td>
<td>010111</td>
<td>3</td>
</tr>
<tr>
<td>X</td>
<td>011000</td>
<td>8</td>
</tr>
<tr>
<td>Y</td>
<td>011001</td>
<td>1</td>
</tr>
<tr>
<td>Z</td>
<td>011010</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>011011</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>011100</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>011101</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>011110</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>011111</td>
<td>15</td>
</tr>
<tr>
<td>5</td>
<td>100000</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>100001</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>100010</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>100011</td>
<td>7</td>
</tr>
<tr>
<td>9</td>
<td>100100</td>
<td>9</td>
</tr>
</tbody>
</table>

*Figure 4.2  Character Table of the Training Set*
There are totally 145 character arrays in the training set. The occurrences of each character in the 145 characters are listed in Figure 4.2.

On an IBM PS/2 70 with a math coprocessor, it takes 21 minutes and 7 seconds to train the network with this training set of total 145 characters. On a 486 PC, it takes less than 20 minutes to train the network. After training, the network can recognize correctly all the sample characters in the training set.

Once trained, the network is able to recognize patterns, and the final weights should be saved so that the network can be reused without training the network each time. The set of final weights represents what the system has learned. These weights are used in testing the system's ability to recognize characters. Training of the network can be resumed by feeding in the previous set of weights, and making slight adjustments to them to enhance the performance of the network. This allows the network to adapt to a new font of characters.

4.2 Testing The System

To further test the performance of the system, especially the neural network, additional 35 CapCalc pictures were taken. These pictures include 7 license plates with different sizes in the pictures. The 7 license plates include 4 black-on-white and 3 white-on-black license plates. The 7 license plate codes are listed in Figure 4.3. The photocopies of those 35 pictures and the detail information of the pictures are listed in the Appendix B.2. Note that these pictures may look different actually, because the printer can print only binary pictures (black or white pixels).

The method for testing a neural network is to present a series of test cases and evaluate the network's recognition performance. The correctness of
the neural network's character recognition can be determined by comparing the input to the output. The main measurements are the percent correct of recognizing the license plate pictures that meet the requirements, and the time spent to recognize a picture.

<table>
<thead>
<tr>
<th>License Plate</th>
<th>Occurrence</th>
<th>Type of Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>3366 NT (a-i)</td>
<td>9</td>
<td>White-on-Black</td>
</tr>
<tr>
<td>AGT 616 (a-c)</td>
<td>3</td>
<td>White-on-Black</td>
</tr>
<tr>
<td>MA RIU3 (a-c)</td>
<td>3</td>
<td>White-on-Black</td>
</tr>
<tr>
<td>EKA 070 (a-d)</td>
<td>4</td>
<td>Black-on-White</td>
</tr>
<tr>
<td>SWO SAS (a-c)</td>
<td>3</td>
<td>Black-on-White</td>
</tr>
<tr>
<td>743 JHE (a-i)</td>
<td>9</td>
<td>Black-on-White</td>
</tr>
<tr>
<td>YGN 66 (a-d)</td>
<td>4</td>
<td>Black-on-White</td>
</tr>
</tbody>
</table>

Figure 4.3 License Plate for Testing

Note that (a-i) represents number a, b, ..., i, total 9 pictures of the license plate "3366 NT". It is the same as others.

Before the license plate pictures are recognized, the network weights saved are loaded into the network. The picture is first preprocessed, and the characters are separated from the picture and scaled into 7x9 arrays. The preprocessor can only process 5 pictures out of the 35 pictures. The preprocessor can not process the other 30 pictures because of the following reasons (for pictures, see Appendix B.2):

1. The gray levels of the background pixels are same as or close to the gray levels of the background. For some pictures, the differences between the gray
levels of the signals and the background are as small as 3. The preprocessor can not tell which pixels are background, which are signal. Also, a lot of the background pixels have the same gray levels as the signal pixels. These pictures include 3366NT (c)(d)(e)(f)(g)(h)(i), AGT616 (b)(c), MARIU3 (b)(c), EKA070 (a)(c)(d), SWOSAS (a)(b)(c), 743JHE (b)(c)(e)(f)(g)(h)(i), YGN66 (a)(b)(c)(d).

2. There are other objects in the pictures, such as table and computer. They are more significant than the license plate characters. These pictures include 3366NT (d)(e)(f)(g)(h)(i), AGT616 (b), MARIU3 (b)(c), EKA070 (a)(c)(d), SWOSAS (b)(c), 743JHE (d)(e)(f)(g)(h)(i), YGN66 (b)(c)(d).

3. The license plates are invisible in the pictures. They can not even be recognized with eyes. These pictures include 3366NT (i), AGT616 (c), EKA070 (a)(b), 743JHE (f)(g)(h)(i).

The preprocessor separates the characters from the 5 pictures and scales them into 7x9 character arrays. After the preprocessing, the system uses the neural network to recognize the character arrays. The license plate characters and the output of the network are shown in Figure 4.4.

<table>
<thead>
<tr>
<th>License Plate Number</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>3366NT (a)</td>
<td>3368NT</td>
</tr>
<tr>
<td>3366NT (b)</td>
<td>3368NT</td>
</tr>
<tr>
<td>AGT616 (a)</td>
<td>AEIM36</td>
</tr>
<tr>
<td>MARIU3 (a)</td>
<td>MANLW1</td>
</tr>
<tr>
<td>743JHE (a)</td>
<td>74J3FF</td>
</tr>
</tbody>
</table>

Figure 4.4 License Plate Characters and the Outputs
For the picture 3366NT (a)(b), the system gives fairly good response. The system outputs a character '8' when given a '6', and these two characters actually are pretty similar. For the other three pictures, the network recognizes correctly less than half of the characters. Some output characters are also similar to the given characters, even though they are different. These include generating a character 'I' given a 'T', and a 'F' given a 'E'. Overall, the system recognizes 16 out of the total 30 characters.

The recognition procedure for each picture takes less than one minute, depending on the computer used to run the program. On a 25 MH 386 PS/2 model 70 with math coprocessor, it usually takes about 16 seconds for each picture, including the preprocessing. More about the speed will be discussed in the next chapter.

Another two sets of license plates are used to make an additional test. Each set consists of three pictures of the same license plate with different sizes. The system is successful in recognizing those different-size license plate pictures. The recognition process was shown to the committee members on the presentation of the project. Unfortunately, these pictures are unavailable as this writing.

There are some requirements and limitations of this system, as specified in Chapter Two. The preprocessor could not process many of the pictures being tested. The neural network recognized some of the characters, while made mistakes on the others. The experiment suggests that the recognition method is feasible, even though it is still far from actual application. An overview of the system and the techniques will be given in the next chapter.
5.1 Conclusions

The structure of the system and how the system works have been discussed. An executable program has been generated and tested. Even though the system has a number of limitations as described in the system requirements and can not recognize many of the tested license plate pictures, the developed program is able to detect some of the characters. This project is supposed to be a research for recognizing pictures of license plates inside the laboratory. The software is not intended to be an exact model of a biological vision system. It demonstrates the feasibility of automatically detecting license plates using image processing techniques and neural networks. While all the images used were taken in the laboratory, the same methodology may apply to images taken in the field. The neural network could be used as a basis for recognizing other objects such as traffic signs.

The recognition of license plate characters includes locating and recognizing the characters. The overall approach is to first find out what the features of an image are and to understand the way they relate to each other. Once those attributes and the processing techniques have been selected, extracting information from an image is possible. The differences in the pixel values of the characters and background are essential in locating the characters, and they make the recognition of license plate characters possible. The edges of the characters are crucial in separating the characters from the background. Convolution is a very powerful technique to preprocess images and determine the positions of the characters. It detects the relative differences of the gray levels among pixels, instead of the absolute gray levels. Different
convolution kernels may have different impacts on the images. Selecting the right convolution kernel can effectively detect the edges in the pictures. This technique can also be applied to many other processes, such as noise filtering.

The order in which the different processing steps are applied to the pictures is very important. This requires a careful design of the preprocessor. The pictures should be handled in a way that each process will produce the desired input for the next step, and the pictures are divided into smaller and smaller regions depending on finer and finer distinctions between pixels. The preprocessor finally extracts the characters from the pictures.

Instead of using one network for characters of one particular size, a network is trained to recognize characters of different sizes. A scaling process is used to convert characters to one specific size. This saves the time of training multiple networks, and avoids the system from switching between multiple networks while trying to recognize characters of different sizes. It also allows a single network to adjust to characters of different sizes.

Back-propagation appears to be a good method for the pattern recognition. The network parameters learning factor and momentum have a great impact on the operation of the network. Selecting the right values would lead to fast and successful training. Other researches may suggest some choices, but the optimal values are usually obtained through experiment. While the original back-propagation is still widely used, some modifications are necessary to improve the performance of the network. The differential step-size calculation is used in the neural network to speed up the training and avoid the local minima problem.

Neural networks are good at some fields, and can be training be perform a single, well-defined task, such as character recognition. They perform poorly in other areas, including calculation, logic inference, and data transformation. The
combination of neural networks with other technology could promise great potential for some applications. This research provides a good example of applying neural networks with image processing techniques.

While people see hundreds of thousands of characters every day, the neural network was trained from only a total of 145 characters. We should not expect the system to recognize license plate characters as good as human beings can. As we may see in the character table in Figure 4.3, 20 out of the 36 characters have only 3 or less samples in the training set. Some of the characters have more than 7 samples, and the character '4' has 15 samples. The network was trained to recognized some characters better than the others. This causes unbalance in the recognition capability of the network. It could be avoided by using a large, and a more evenly-distributed training set. Before the system is applied to an actual field situation, hundreds or thousands of license plate pictures should be used to produce a more comprehensive training data set to train the neural network.

The recognition of each picture, including the preprocessing and recognition with the neural network, takes about 16 seconds on a 25Mhz 386 PC. It takes about 10 seconds on a 486 PC with the same CPU speed. The recognition time of 10 seconds may be further reduced by the following improvements:

(1) The program is designed with the idea that it just is a first step of a research effort in a laboratory setting, rather than for a real field situation. The structure, reusability, flexibility and clearness were considered more important than processing speed and time when the system was programmed. The preprocessor may be optimized for speed and shorten processing time to
achieve a higher speed of recognition. Some statements of the program could be combined to reduce calculations and assignments. A few statements could be replaced by other more efficient implementation, such as using summation instead of loop statement. Reducing the overhead of function-calling will also make the program run faster. It is estimated that the system may be made to run 5% to 20% faster.

(2) Although the program is written in Borland C++, it was attempted to use only the structures and functions of ANSI C, with the extensions to support Windows. The program may be recompiled under another C compiler which performs more optimizations than Borland C++, such as Watcom C or the Microsoft C compiler. This should speed up the program to some extent. The system is estimated to run 10% to 20% faster.

(3) The Microsoft Windows 3.0 and its applications are based on the 16-bit implementation. They don't fully use the potential capability of the 32-bit architecture of the 386 and 486 microprocessors. The system could be implemented as a 32-bit application on the IBM OS/2 2.0 operating system or with the Watcom/386 C compiler and DOS extender. The system could run 30% to 100% faster with the new implementation.

(4) With more powerful microprocessors, including 33 MHz, 40 MHz and 50 MHz 486 chips, the system can run much faster than it runs now. The forthcoming 100 MHz 486 and 586 chips will make the recognition even faster. The system may run 2 to 5 times faster with a new microprocessor.

With the above improvements, it is reasonable to accomplish the recognition in about 2 seconds. If the program is ported to other high-performance platforms, such as RISC workstations or SPARC stations, it can be imagined that the recognition time will be much shorter.
5.2 Contributions

Some new techniques have been created and used to achieve the objective of this research. The contributions to this thesis research efforts are:

1) It is a first attempt in the Dept of Industrial and Systems Engineering to use neural network, combined with image processing techniques, to automatically recognize license plate characters with high contrast values (Black-on-White and White-on-Black) as specified in Chapter Two.

2) Instead of using the first derivative of the transfer function in calculating the propagated errors for output neurons, the proposed differential step-size calculation is used to solve the local minima problem.

3) When the network is being trained and the network error is very small, the propagated errors are multiplied by a shifting factor. This technique greatly speeds up the training.

5.3 Suggestions and Recommendations

The directions recommended for further study of the pattern recognition techniques and enhancement of the program are as following:

1) The current system can recognize only the high-contrast license plate pictures. The first enhancement can be to extend the system to recognize license plates of less contrast. As discussed above, there should be a more comprehensive training set to train the network. More license plate pictures should be taken to test and train the neural network. If the preprocessor separates characters from the pictures, and the network can not recognize the characters, the output from the preprocessor could be added to the training set to train and enhance the neural network.

2) The current system cannot detect a picture in which the license plate is not placed horizontally. A line detector may be added to detect the edge of a
license plate. The convolution method used in the system could also be applied here. Selecting a convolution kernel for line detection can detect lines in different directions. Once the lines in the pictures are detected, the angle of the edge of the license plate is calculated. The program then rotates the image according to the angle of the edge.

3) The system is designed to recognize only the license plate characters. It may be extended to recognize objects other than characters, such as traffic signs.

4) Currently it takes a while to display a license plate picture on screen. The program can paint the image in a couple of seconds if the image is in Microsoft Windows Bitmap format. Some functions may be added to convert the image data into the Microsoft Windows Bitmap format. Also, pictures are being stored in many popular image formats, such as TIFF, PCX, GIF. It may be helpful in using the program with other applications, if the system could make these conversions. The conversions should be pretty straightforward since the detail specifications of these formats are already in the public domain.

5) Object-oriented techniques may be incorporated into the system to help the design of the system, and to improve the reusability of the program. Object-oriented design and programming promote better understanding of requirements, cleaner designs, and more maintainable systems. It reflects relations of the real world better, as well as reports and prevents most possible bugs in the program. A complete description of a object-oriented system requires three aspects: the object model, the dynamic model, and the functional model. For example, the object-oriented technique is applied to the design of a neural network. The object model is most fundamental and it describes all the entities (objects) in the network, such as the neurons in different layers, and the connections between layers. Similar objects are grouped into the same class,
and they share the same functions but different data. All neurons could belong to a neuron class, and have the same calculations but different data (input, output). The dynamic model describes the interactions among objects in the system, such as neurons pass output to other neurons, and the errors are fed back to the connections. The functional model describes the data transformations of the system, such as the output is calculated with a sigmoid transfer function within the neurons. The object-oriented approach focus first on identifying objects from the application domain, then the interactions among them, then fitting functions around them. It isolates the details of each object from the others. How the neurons calculate their output has nothing to do with how the connections change the weights. The data (such as input, error) and operations (such as transfer functions) of the objects (neurons, connections, etc) stay within each objects and are isolated from the others. The system works as the data are processed in the objects, while the objects interact with each other in a particular order (input neuron object to connection, to hidden neuron object, etc) to accomplish a task (recognition). This defines a basic view of the system, and limit any possible errors within narrow ranges. Modification of the system changes only some of the objects without affecting the data and functions in other objects.
Bibliography and References

5. Brown, Bruce. "Hewlett-Packard Co.: AccuScan A.05.03. (Software Review)", *PC Week*, Dec 3, 1990, v7 n48 p92(2)
11. Frangini, Monica, "Computerized road tolls not a priority yet, says ministry.", Computing Canada, March 29 1990, v16 n7 p12(1)
14. James, Mike, Pattern Recognition, John Wiley & Sons, 1988
16. King, Todd, "Using Neural Networks For Pattern Recognition", Dr. Dobb's Journal, January 1989
17. Lake, Matthew. "Strength of character recognition: 10 programs that take the drudgery out of digitizing the printed word.", Publish!, Jan 1991, v6 n1 p62(5).
22. Pao, Yoh-Han, Adaptive Pattern Recognition And Neural Networks, Addison-Wesley, 1989


Appendix A. User Guide

The software package is a Microsoft Windows program running on an IBM PC or a compatible PC with a DOS operating system. It can be used to train neural networks, process and recognize license plate pictures. The package consists of two main parts: the preprocessor and the neural network. The preprocessor is used to preprocess license plate pictures. It picks out the characters from the photographs of license plates and provides input for the neural network. The neural network recognizes the characters and gives a result.

The program is very user-friendly with graphical user interface support. This includes support for mouse, pull-down menus, dialog boxes and pop-up windows. It does not require any programming experience, but expect the user to have some work experience with the DOS and the Microsoft Windows environment. Before starting the program, make sure that DOS 3.1 or higher and Microsoft Windows 3.0 or a later version are properly installed. The computer to run this software should have a 286 or a higher CPU with at least one megabyte of memory (RAM). Also, the computer should have a harddisk with at least 20-megabyte capacity to store DOS, Microsoft Windows, the software and the license plate pictures.

The user can install the package in the computer harddisk by using MD dir-name and COPY commands. Dir-name is the name of the directory where the package is stored, such as C:\DETECT. The DOS command "COPY a:*.* dir-name" is used to copy the package to the directory dir-name (C:\DETECT in this case). The windows directory should be in the path. This can be done by
typing `PATH=windows-dir` at the DOS prompt. `Window-dir` is the directory in which Windows is installed (usually in `C:\WINDOWS`).

After turning on the PC, the user should change the current directory to the one where the executable DETECT.EXE is stored (`C:\DETECT`). To start the program, type `WIN DETECT.EXE` at the DOS prompt. This initializes the Windows environment and the main window of the program is displayed. It includes the menu bar with which the user can perform different operations which are described in the following sections. The current screen is shown in Figure A.1.

![Main Window of the System](image)

**Figure A.1** Main Window of the System
Note that each valid DOS file name has a main part of up to eight characters and a file extension of up to 3 characters. The file extensions are usually used to identify different kinds of files, such as all executable files have extension names of EXE, COM, and BAT. For example, the file `DETECT.EXE` has a main file name of `DETECT` and a extension name of `.EXE`. The extension represents that the file `DETECT.EXE` is an executable program.

All files are given different file extension names by the program so that the same main file name is used for different processing. They are identified by the different extension names. The extension of a file name shows what type of file it is. It helps the user to select the proper file for different operations, such as training and pre-processing. The system suggests a default extension for each type of files whenever the user is asked for a file name. It is recommended that different types of files should be given different extensions if the user chooses to give file extension names other than the ones recommended.

(1) Preprocessing

The picture of a license plate is generated by CapCalc or compatible devices and saved in the export format (each pixel is represented by a floating-point number between 0 and 256, arranging in 512x480 format). The name of the saved file has an extension name of EXP (for example, `6124TM.EXP` for the license plate "612 4TM"). The picture file is first compressed before the preprocessing by selecting the command Compress Image under the command Image shown in Figure A.2.

The system prompts for the file name of the license plate picture to be processed. The screen for entering the picture file is shown in Figure A.3.
Figure A.2 Commands For Processing Pictures

Figure A.3 Window For Entering a File Name
The user then is asked for the name of the file to save the compressed picture (shown in Figure A.4). The extension name of the file name is DAT (6124TM.DAT, given the export file 6124TM.EXP).

![Save File Name As:](d:\design\image)

Figure A.4  Window For Saving a File

The compressed picture can be loaded with the command **Load Image** (see Figure A.2). The program prompts for the name of the compressed picture file (contains 512x480 integer numbers representing 512x480 pixels) of the license plate. After the picture file is entered (6124TM.DAT in this case), the picture is loaded into the system. The user may display the picture of the license plate with the command **View Image**. The picture then is displayed in a pop-up
window (shown in Figure A.5). When the user clicks the mouse on the picture, the system shows the gray level of the pixel where the arrow of the mouse is along with the pixel's coordinates.

Figure A.5  Window For Displaying a Picture

To start the preprocessing of the picture, select the commands Image and Pre-Processing on the menu (see Figure A.2). If the system is able to process the picture, the character blocks are scaled into 7x9 character arrays. The user can choose the command Save Character under Image of the main menu
(shown in Figure A.2) and save the character arrays into a file with an extension name of OUT for training the network (i.e., $6124TM.OUT$). If the system is not able to process, a message is displayed to inform the user that the system cannot preprocess the picture (shown in Figure A.6).

In this case, the user may want to see the output from the horizontal scan and vertical scan in the preprocessing. It helps the user to figure out why the preprocessing fails. This can be done by selecting the option **Display Output** (see Figure A.7).
(2) Train a Neural Network

To train a neural network, a data file (training set) should be prepared in a proper format as described in Chapter Four. After the data file of all training patterns is created, the user may start to train the network. The extension name for the training data file is PTN (abbreviation of pattern).

The training can be started by selecting the command Network on the menu bar and then the Training command from a pull-down menu (see Figure A.8). The system prompts for the name of the data file (see Figure A.3). The
user simply enters the name of the data file created earlier for training, and the training process will start.

Figure A.8  Commands For Training the Network

The process may be stopped whenever the user clicks the mouse on menu bar during the training. After the training is stopped, it could be resumed with the command Continue (see Figure A.8). The user could also cancel the training, reset and unload the network with the command Reset (see Figure A.8).
The period in which a pattern goes through the forward and backward passes is considered as one pass. An iteration is considered as the process in which all patterns go through one pass. In each iteration, the network generates output for each pattern, and corrects pattern errors for each pattern. The total error is displayed after each iteration (shown in Figure A.9). The error for each pattern may also be displayed by selecting the option Display Error (see Figure A.7). This will greatly slow down the training and should be used only when monitoring the status of the training. After each iteration, the network continues with the next pass until the total error is within acceptable limits (less than 0.1).

![License Plate Recognition System Network Image Options About](image)

| Iteration: 16 | Total error: 41.589 |
| Iteration: 15 | Total error: 45.605 |
| Iteration: 14 | Total error: 54.780 |
| Iteration: 13 | Total error: 64.692 |
| Iteration: 12 | Total error: 70.716 |
| Iteration: 11 | Total error: 78.547 |
| Iteration: 10 | Total error: 86.715 |

Figure A.9 Displaying the Total Error in Training
Network iteration 43 Total error: 0.000000
Iteration: 42 Total error: 0.114

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Total error</th>
</tr>
</thead>
<tbody>
<tr>
<td>38</td>
<td>0.310</td>
</tr>
<tr>
<td>37</td>
<td>0.437</td>
</tr>
</tbody>
</table>

Iterations: 43, Elapsed time: 0[hr]:21[min]:7[sec]

Figure A.10  Window Displaying the Training Time

After the training is finished, a window will pop-up and display the time spent on the training (Figure A.10). The user may select the command System, and Save Weights commands (see Figure A.11) to save the network and the knowledge it just learned into a file. The user may enter the file name with extension name WGT (like NETWORK.WGT), and the network weights are saved.

The network can be trained for another training set based on the weights from the previous training by using the command Next Training Set (see Figure A.8). It is used when the user has more training data or makes the network adapt to new fonts without losing the knowledge it has learned earlier. The training
process is the same as described above, except that the network is trained based on previous weights instead of random weights.

Figure A.11  Commands For Saving/Loading Weights

(3) **Recognize a License plate**

The user should train the network and save the network weights before the recognition. The recognition of a license plate begins with the preprocessing as described in section (1). After the preprocessing, the system can make the final recognition based on the output from the pre-processing. The
user loads the network weights with the command **Load Weights** under **System** (see Figure A.11). The file extension name of the network weights is WGT (The file may be **NETWORK.WGT**). After the network weights are loaded, the network is ready to recognize the characters. When the user picks the commands **Network** and **Recognize** (see Figure A.8), the network tries to recognize those characters and the results are displayed on a pop-up window (Figure A.12).

![Figure A.12 Output From the Neural Network](image)

The system can also recognize the character arrays saved in a file (in this case **6124TM.OUT**) after the preprocessing. The commands **Network** and **Recognize File** are used (see Figure A.8) to test only the network, and not the preprocessing.
(4) Other Features of the System

The system uses the back-propagation method to train the neural network. The network has three layers: input layer, hidden layer, and output layer. The input layer has 63 neurons and distributes the character arrays to the network. The hidden layer has 47 neurons. The output layer has 6 neurons and generates output of the network. The sigmoid function is used as the transfer function for all the neurons.

The user may change the default parameters of the neural network with the commands Options and Parameters (see Figure A.7). Changing the parameters may greatly affect the training of the network. It could be used to experience how each parameter affects the operation of the network and determine the optimal values of the parameters. The screen for changing the parameters is shown in Figure A.13. Unless the user has a good understanding of the neural network and the back-propagation technique, these parameters should not be modified.

The user can display the logo and basic information about the system using the command About (shown in Figure A.14).
Figure A.13 Changing Network Parameters

Neural Network Based
Automatic Recognition of License Plate Characters
ISE Dept., Ohio University, 1992

Figure A.14 Basic Information About the System
Appendix B.1  License Plates for Training
Appendix B.2  License Plates for Testing
<table>
<thead>
<tr>
<th>Case</th>
<th>Pixel Height</th>
<th>ILL. in Number (cd/m²)</th>
<th>ILL. in Letter (cd/m²)</th>
<th>ILL. in Red Area (cd/m²)</th>
<th>ILL. at Background (cd/m²)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3366NT (a)</td>
<td>88</td>
<td>114.22</td>
<td>119.42</td>
<td>18.17</td>
<td>51.92</td>
</tr>
<tr>
<td></td>
<td></td>
<td>150.57</td>
<td>171.34</td>
<td>15.58</td>
<td>49.32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>160.95</td>
<td>145.38</td>
<td>12.98</td>
<td>54.52</td>
</tr>
<tr>
<td></td>
<td></td>
<td>147.97</td>
<td>122.01</td>
<td>20.77</td>
<td>57.11</td>
</tr>
<tr>
<td></td>
<td></td>
<td>129.80</td>
<td>111.63</td>
<td>7.79</td>
<td>62.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>106.44</td>
<td>106.44</td>
<td>25.96</td>
<td>46.73</td>
</tr>
<tr>
<td></td>
<td></td>
<td>116.82</td>
<td>101.24</td>
<td></td>
<td>44.13</td>
</tr>
<tr>
<td></td>
<td></td>
<td>119.42</td>
<td>116.82</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3366NT (b)</td>
<td>89</td>
<td>42.38</td>
<td>50.58</td>
<td>4.10</td>
<td>19.14</td>
</tr>
<tr>
<td></td>
<td></td>
<td>41.01</td>
<td>42.38</td>
<td>5.47</td>
<td>20.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>45.11</td>
<td>49.21</td>
<td>6.83</td>
<td>17.77</td>
</tr>
<tr>
<td></td>
<td></td>
<td>51.95</td>
<td>46.28</td>
<td>8.2</td>
<td>16.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>49.21</td>
<td>43.74</td>
<td></td>
<td>23.24</td>
</tr>
<tr>
<td></td>
<td></td>
<td>35.59</td>
<td>36.91</td>
<td></td>
<td>15.04</td>
</tr>
<tr>
<td></td>
<td></td>
<td>54.68</td>
<td>38.28</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>53.31</td>
<td>53.31</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>47.84</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3366NT (c)</td>
<td></td>
<td>3.55</td>
<td>4.27</td>
<td>0.71</td>
<td>1.42</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4.27</td>
<td>4.98</td>
<td>1.42</td>
<td>2.13</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2.84</td>
<td>2.13</td>
<td>0</td>
<td>0.71</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4.98</td>
<td>2.84</td>
<td></td>
<td>2.84</td>
</tr>
<tr>
<td>Case</td>
<td>Pixel Height</td>
<td>ILL. in Number (cd/m²)</td>
<td>ILL. in Letter (cd/m²)</td>
<td>ILL. in Red Area (cd/m²)</td>
<td>ILL. at Background (cd/m²)</td>
</tr>
<tr>
<td>-----------</td>
<td>--------------</td>
<td>------------------------</td>
<td>------------------------</td>
<td>--------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>3366NT (d)</td>
<td></td>
<td>112.18</td>
<td>109.51</td>
<td>24.04</td>
<td>56.09</td>
</tr>
<tr>
<td></td>
<td></td>
<td>117.52</td>
<td>46.91</td>
<td>16.03</td>
<td>58.76</td>
</tr>
<tr>
<td></td>
<td></td>
<td>146.91</td>
<td>136.22</td>
<td>21.37</td>
<td>61.43</td>
</tr>
<tr>
<td></td>
<td></td>
<td>120.20</td>
<td>157.59</td>
<td>18.7</td>
<td>53.42</td>
</tr>
<tr>
<td></td>
<td></td>
<td>122.87</td>
<td>117.52</td>
<td>13.36</td>
<td>48.08</td>
</tr>
<tr>
<td></td>
<td></td>
<td>106.84</td>
<td>138.89</td>
<td>29.38</td>
<td>42.79</td>
</tr>
<tr>
<td></td>
<td></td>
<td>133.55</td>
<td>184.3</td>
<td></td>
<td>50.75</td>
</tr>
<tr>
<td></td>
<td></td>
<td>141.56</td>
<td>114.85</td>
<td></td>
<td>45.41</td>
</tr>
<tr>
<td></td>
<td></td>
<td>154.92</td>
<td>101.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>149.58</td>
<td>162.93</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3366NT (e)</td>
<td></td>
<td>18.16</td>
<td>25.15</td>
<td>4.16</td>
<td>8.38</td>
</tr>
<tr>
<td></td>
<td></td>
<td>23.75</td>
<td>27.15</td>
<td>4.19</td>
<td>9.78</td>
</tr>
<tr>
<td></td>
<td></td>
<td>25.15</td>
<td>23.75</td>
<td>2.79</td>
<td>6.98</td>
</tr>
<tr>
<td></td>
<td></td>
<td>13.97</td>
<td>20.95</td>
<td>6.98</td>
<td>22.35</td>
</tr>
<tr>
<td></td>
<td></td>
<td>26.64</td>
<td>19.56</td>
<td>5.59</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>22.35</td>
<td>16.76</td>
<td>1.40</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>29.34</td>
<td>22.35</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3366NT (f)</td>
<td>35</td>
<td>2.9</td>
<td>4.35</td>
<td>0.73</td>
<td>1.45</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3.63</td>
<td>3.63</td>
<td>0</td>
<td>2.18</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2.18</td>
<td>5.8</td>
<td>1.45</td>
<td>0.73</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5.08</td>
<td>2.9</td>
<td></td>
<td>3.63</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4.35</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Case</td>
<td>Pixel Height</td>
<td>ILL. in Number (cd/m²)</td>
<td>ILL. in Letter (cd/m²)</td>
<td>ILL. in Red Area (cd/m²)</td>
<td>ILL. at Background (cd/m²)</td>
</tr>
<tr>
<td>------------</td>
<td>--------------</td>
<td>------------------------</td>
<td>------------------------</td>
<td>--------------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>3366NT (g)</td>
<td>14</td>
<td>67.5</td>
<td>129.6</td>
<td>54</td>
<td>45.9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>83.7</td>
<td>137.7</td>
<td>56.7</td>
<td>48.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>40.5</td>
<td>172.8</td>
<td>43.2</td>
<td>51.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>97.2</td>
<td>132.3</td>
<td>18.9</td>
<td>54</td>
</tr>
<tr>
<td></td>
<td></td>
<td>140.4</td>
<td>126.9</td>
<td>35.1</td>
<td>59.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>108</td>
<td>118.8</td>
<td>24.3</td>
<td>56.7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>89.1</td>
<td>124.2</td>
<td>21.6</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>151.2</td>
<td>102.6</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>116.1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3366NT (h)</td>
<td>11</td>
<td>25.34</td>
<td>40.83</td>
<td>9.86</td>
<td>11.26</td>
</tr>
<tr>
<td></td>
<td></td>
<td>30.98</td>
<td>42.24</td>
<td>4.22</td>
<td>12.67</td>
</tr>
<tr>
<td></td>
<td></td>
<td>29.57</td>
<td>39.62</td>
<td>7.04</td>
<td>14.08</td>
</tr>
<tr>
<td></td>
<td></td>
<td>28.16</td>
<td>28.16</td>
<td>1.41</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>26.75</td>
<td>32.35</td>
<td>5.63</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>33.79</td>
<td>25.34</td>
<td>2.82</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>23.94</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3366NT (i)</td>
<td></td>
<td>3.66</td>
<td>5.85</td>
<td>1.46</td>
<td>1.46</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4.39</td>
<td>5.12</td>
<td>0.73</td>
<td>2.19</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2.19</td>
<td>4.39</td>
<td>2.93</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3.66</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Case</td>
<td>Pixel Height</td>
<td>ILL. in Number (cd/m²)</td>
<td>ILL. in Letter (cd/m²)</td>
<td>ILL. in Red Area (cd/m²)</td>
<td>ILL. at Background (cd/m²)</td>
</tr>
<tr>
<td>---------------</td>
<td>--------------</td>
<td>------------------------</td>
<td>------------------------</td>
<td>--------------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>AGT616 (a)</td>
<td>88</td>
<td>12.98</td>
<td>15.58</td>
<td>7.79</td>
<td>7.79</td>
</tr>
<tr>
<td></td>
<td></td>
<td>15.58</td>
<td>12.98</td>
<td>5.19</td>
<td>10.38</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10.38</td>
<td>10.38</td>
<td>2.6</td>
<td>5.19</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7.79</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AGT616 (b)</td>
<td>36</td>
<td>10.88</td>
<td>16.69</td>
<td>5.08</td>
<td>5.08</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10.16</td>
<td>14.51</td>
<td>2.18</td>
<td>4.35</td>
</tr>
<tr>
<td></td>
<td></td>
<td>13.06</td>
<td>11.61</td>
<td>1.45</td>
<td>2.9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>14.51</td>
<td>15.96</td>
<td>8.71</td>
<td>3.63</td>
</tr>
<tr>
<td></td>
<td></td>
<td>16.69</td>
<td>10.88</td>
<td>2.9</td>
<td>0.73</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11.61</td>
<td>15.24</td>
<td></td>
<td>13.78</td>
</tr>
<tr>
<td></td>
<td></td>
<td>13.78</td>
<td>13.75</td>
<td></td>
<td>11.61</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>12.33</td>
<td></td>
<td>25.39</td>
</tr>
<tr>
<td>AGT616 (c)</td>
<td>11</td>
<td>3.66</td>
<td>4.39</td>
<td>2.19</td>
<td>3.66</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5.12</td>
<td>3.66</td>
<td>1.46</td>
<td>5.85</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5.85</td>
<td>2.93</td>
<td>0.73</td>
<td>2.19</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2.93</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6.58</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>7.32</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>8.05</td>
</tr>
<tr>
<td>AGT616 (d)</td>
<td>30</td>
<td>2.9</td>
<td>2.18</td>
<td>0.73</td>
<td>2.18</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2.18</td>
<td>4.35</td>
<td>1.45</td>
<td>1.45</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4.35</td>
<td>5.8</td>
<td>2.90</td>
<td>2.90</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5.08</td>
<td>4.35</td>
<td>5.8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>6.53</td>
<td>5.8</td>
<td>10.16</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2.9</td>
<td>10.88</td>
<td></td>
</tr>
<tr>
<td>Case</td>
<td>Pixel Height</td>
<td>ILL. in Letter (cd/m²)</td>
<td>ILL. in Number (cd/m²)</td>
<td>ILL. in Black Area (cd/m²)</td>
<td>ILL. at Background (cd/m²)</td>
</tr>
<tr>
<td>----------</td>
<td>--------------</td>
<td>------------------------</td>
<td>------------------------</td>
<td>--------------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>MARIU3 (a)</td>
<td>72</td>
<td>3.59</td>
<td>3.59</td>
<td>1.44</td>
<td>2.16</td>
</tr>
<tr>
<td></td>
<td>4.31</td>
<td>5.03</td>
<td>0.72</td>
<td>1.44</td>
<td>2.88</td>
</tr>
<tr>
<td></td>
<td>2.88</td>
<td>4.31</td>
<td>0.0</td>
<td>2.88</td>
<td>1.99</td>
</tr>
<tr>
<td></td>
<td>5.03</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MARIU3 (b)</td>
<td>41</td>
<td>16.76</td>
<td>13.97</td>
<td>1.40</td>
<td>8.38</td>
</tr>
<tr>
<td></td>
<td>12.57</td>
<td>18.16</td>
<td>5.59</td>
<td>9.78</td>
<td>15.37</td>
</tr>
<tr>
<td></td>
<td>18.16</td>
<td>15.37</td>
<td>2.79</td>
<td>6.98</td>
<td></td>
</tr>
<tr>
<td></td>
<td>15.37</td>
<td></td>
<td>4.19</td>
<td></td>
<td>12.57</td>
</tr>
<tr>
<td></td>
<td>13.97</td>
<td></td>
<td></td>
<td></td>
<td>22.35</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>20.95</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>60.07</td>
</tr>
<tr>
<td>MARIU3 (c)</td>
<td>18</td>
<td>116.10</td>
<td>97.20</td>
<td>0.0</td>
<td>56.70</td>
</tr>
<tr>
<td></td>
<td>113.40</td>
<td>99.90</td>
<td>2.70</td>
<td>54.00</td>
<td></td>
</tr>
<tr>
<td></td>
<td>118.50</td>
<td>113.40</td>
<td>13.50</td>
<td>40.50</td>
<td></td>
</tr>
<tr>
<td></td>
<td>94.50</td>
<td>121.50</td>
<td>10.80</td>
<td>43.20</td>
<td></td>
</tr>
<tr>
<td></td>
<td>97.20</td>
<td>110.70</td>
<td>18.90</td>
<td></td>
<td>97.20</td>
</tr>
<tr>
<td></td>
<td>91.80</td>
<td></td>
<td></td>
<td></td>
<td>18.90</td>
</tr>
<tr>
<td></td>
<td>83.7</td>
<td></td>
<td></td>
<td></td>
<td>137.70</td>
</tr>
<tr>
<td></td>
<td>121.50</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>124.20</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>126.90</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>129.60</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Case</td>
<td>Pixel Height</td>
<td>ILL. in Number (cd/m²)</td>
<td>ILL. in Letter (cd/m²)</td>
<td>ILL. in White Area (cd/m²)</td>
<td>ILL. at Background (cd/m²)</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------</td>
<td>------------------------</td>
<td>------------------------</td>
<td>-----------------------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>EKA070 (a)</td>
<td>67</td>
<td>0.72</td>
<td>0.72</td>
<td>2.88</td>
<td>1.44</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.0</td>
<td>1.62</td>
<td>3.59</td>
<td>2.16</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.0</td>
<td>2.16</td>
<td>0.72</td>
</tr>
<tr>
<td>EKA070 (b)</td>
<td>36</td>
<td>0.73</td>
<td>1.45</td>
<td>2.18</td>
<td>1.45</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.45</td>
<td></td>
<td>2.9</td>
<td>2.18</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.0</td>
<td>3.63</td>
<td>0.73</td>
</tr>
<tr>
<td>EKA070 (c)</td>
<td>36</td>
<td>1.4</td>
<td>2.79</td>
<td>9.78</td>
<td>8.38</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2.79</td>
<td>4.19</td>
<td>11.18</td>
<td>6.98</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.19</td>
<td>0.0</td>
<td>8.38</td>
<td>5.57</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.0</td>
<td>1.4</td>
<td>12.57</td>
<td>12.57</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>9.78</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>18.16</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>19.56</td>
</tr>
<tr>
<td>EKA070 (d)</td>
<td>36</td>
<td>13.36</td>
<td>10.68</td>
<td>93.49</td>
<td>42.74</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5.34</td>
<td>8.01</td>
<td>90.81</td>
<td>53.42</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8.01</td>
<td>2.67</td>
<td>96.16</td>
<td>50.75</td>
</tr>
<tr>
<td></td>
<td></td>
<td>18.70</td>
<td>21.37</td>
<td>88.14</td>
<td>56.09</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10.68</td>
<td>16.03</td>
<td>85.47</td>
<td>48.08</td>
</tr>
<tr>
<td></td>
<td></td>
<td>16.03</td>
<td>13.36</td>
<td>69.45</td>
<td>80.13</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>74.49</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>18.7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>133.55</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>381.95</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>432.7</td>
</tr>
<tr>
<td>Case</td>
<td>Pixel Height</td>
<td>ILL. in Letter (cd/m²)</td>
<td>ILL. in White Area (cd/m²)</td>
<td>ILL. at Background (cd/m²)</td>
<td>ILL. in Symbol (cd/m²)</td>
</tr>
<tr>
<td>-----------</td>
<td>--------------</td>
<td>------------------------</td>
<td>----------------------------</td>
<td>---------------------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>SW0SAS (a)</td>
<td>61</td>
<td>0.31</td>
<td>2.20</td>
<td>1.57</td>
<td>1.26</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.94</td>
<td>3.46</td>
<td>1.89</td>
<td>1.57</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.63</td>
<td>1.89</td>
<td>1.26</td>
<td>0.31</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.0</td>
<td>2.83</td>
<td>0.94</td>
<td>1.89</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2.51</td>
<td></td>
<td>0.63</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3.14</td>
<td></td>
<td>2.20</td>
</tr>
<tr>
<td>SW0SAS (b)</td>
<td>32</td>
<td>2.79</td>
<td>8.38</td>
<td>6.98</td>
<td>8.38</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4.19</td>
<td>9.78</td>
<td>8.38</td>
<td>5.59</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5.59</td>
<td>15.37</td>
<td>9.78</td>
<td>2.79</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.0</td>
<td>19.56</td>
<td></td>
<td>8.38</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.40</td>
<td>13.97</td>
<td></td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>16.76</td>
<td></td>
<td>1.40</td>
</tr>
<tr>
<td>SW0SAS (c)</td>
<td>13</td>
<td>16.20</td>
<td>72.90</td>
<td>40.50</td>
<td>13.50</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.0</td>
<td>86.40</td>
<td>43.20</td>
<td>10.80</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10.50</td>
<td>91.50</td>
<td>48.60</td>
<td>32.40</td>
</tr>
<tr>
<td></td>
<td></td>
<td>21.60</td>
<td>83.70</td>
<td>45.90</td>
<td>29.70</td>
</tr>
<tr>
<td></td>
<td></td>
<td>18.90</td>
<td>81.00</td>
<td>54.00</td>
<td>43.20</td>
</tr>
<tr>
<td></td>
<td></td>
<td>13.50</td>
<td></td>
<td>51.30</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>5.40</td>
<td></td>
<td>55.90</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2.70</td>
<td></td>
<td>48.60</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>8.10</td>
<td></td>
<td>126.90</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>29.70</td>
<td></td>
<td>219.60</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>116.10</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>118.80</td>
<td></td>
</tr>
<tr>
<td>Case</td>
<td>Pixel Height</td>
<td>ILL. in Number (cd/m²)</td>
<td>ILL. in Letter (cd/m²)</td>
<td>ILL. in White Area (cd/m²)</td>
<td>ILL. at Background (cd/m²)</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------</td>
<td>------------------------</td>
<td>------------------------</td>
<td>----------------------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>743JHE (a)</td>
<td>91</td>
<td>38.94</td>
<td>31.15</td>
<td>90.86</td>
<td>46.73</td>
</tr>
<tr>
<td></td>
<td></td>
<td>46.73</td>
<td></td>
<td>88.26</td>
<td>49.32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>36.34</td>
<td></td>
<td>98.65</td>
<td>51.92</td>
</tr>
<tr>
<td></td>
<td></td>
<td>44.13</td>
<td></td>
<td>103.84</td>
<td>57.11</td>
</tr>
<tr>
<td></td>
<td></td>
<td>41.54</td>
<td></td>
<td>106.44</td>
<td>44.13</td>
</tr>
<tr>
<td></td>
<td></td>
<td>44.13</td>
<td></td>
<td>109.03</td>
<td>54.52</td>
</tr>
<tr>
<td></td>
<td></td>
<td>28.56</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>743JHE (b)</td>
<td>84</td>
<td>8.2</td>
<td>8.2</td>
<td>19.47</td>
<td>9.57</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10.94</td>
<td></td>
<td>20.5</td>
<td>10.94</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9.57</td>
<td></td>
<td>25.97</td>
<td>8.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6.83</td>
<td></td>
<td>23.24</td>
<td>5.47</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8.2</td>
<td></td>
<td>21.87</td>
<td>6.83</td>
</tr>
<tr>
<td></td>
<td></td>
<td>12.3</td>
<td></td>
<td>17.77</td>
<td>12.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>743JHE (c)</td>
<td>88</td>
<td>1.42</td>
<td>1.42</td>
<td>2.84</td>
<td>1.42</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.71</td>
<td></td>
<td>2.13</td>
<td>2.13</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2.31</td>
<td></td>
<td>3.55</td>
<td>0.71</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Case</td>
<td>Pixel Height</td>
<td>ILL. in Number (cd/m²)</td>
<td>ILL. in Letter (cd/m²)</td>
<td>ILL. in White Area (cd/m²)</td>
<td>ILL. at Background (cd/m²)</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------</td>
<td>------------------------</td>
<td>------------------------</td>
<td>---------------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>743JHE (d)</td>
<td>35</td>
<td>26.71</td>
<td>40.07</td>
<td>98.83</td>
<td>53.42</td>
</tr>
<tr>
<td></td>
<td></td>
<td>45.41</td>
<td>42.74</td>
<td>90.81</td>
<td>42.74</td>
</tr>
<tr>
<td></td>
<td></td>
<td>32.05</td>
<td>32.05</td>
<td>82.8</td>
<td>45.41</td>
</tr>
<tr>
<td></td>
<td></td>
<td>48.08</td>
<td>29.38</td>
<td>114.85</td>
<td>56.09</td>
</tr>
<tr>
<td></td>
<td></td>
<td>34.72</td>
<td>48.08</td>
<td>101.5</td>
<td>58.76</td>
</tr>
<tr>
<td></td>
<td></td>
<td>29.38</td>
<td>26.71</td>
<td>93.49</td>
<td>48.08</td>
</tr>
<tr>
<td></td>
<td></td>
<td>53.42</td>
<td>45.41</td>
<td>104.17</td>
<td>32.05</td>
</tr>
<tr>
<td></td>
<td></td>
<td>42.74</td>
<td>24.04</td>
<td>88.14</td>
<td>50.75</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>34.72</td>
<td>109.51</td>
<td>40.07</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>743JHE (e)</td>
<td></td>
<td>5.59</td>
<td>6.98</td>
<td>15.37</td>
<td>6.78</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6.98</td>
<td>4.19</td>
<td>12.57</td>
<td>8.38</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8.38</td>
<td>5.59</td>
<td>13.97</td>
<td>9.78</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9.78</td>
<td>8.38</td>
<td>16.76</td>
<td>5.59</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>743JHE (f)</td>
<td>33</td>
<td>2.9</td>
<td>1.45</td>
<td>2.9</td>
<td>1.45</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.45</td>
<td>0.73</td>
<td>3.63</td>
<td>2.18</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2.18</td>
<td>2.18</td>
<td>5.8</td>
<td>2.18</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2.9</td>
<td>4.35</td>
<td></td>
</tr>
<tr>
<td>Case</td>
<td>Pixel Height</td>
<td>ILL. in Number</td>
<td>ILL. in Letter</td>
<td>ILL. in White Area</td>
<td>ILL. at Background</td>
</tr>
<tr>
<td>------------</td>
<td>--------------</td>
<td>----------------</td>
<td>---------------</td>
<td>--------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>743JHE (g)</td>
<td>14</td>
<td>67.5</td>
<td>129.6</td>
<td>54</td>
<td>45.9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>83.7</td>
<td>137.7</td>
<td>56.7</td>
<td>48.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>40.5</td>
<td>172.8</td>
<td>43.2</td>
<td>51.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>97.2</td>
<td>132.3</td>
<td>18.9</td>
<td>54</td>
</tr>
<tr>
<td></td>
<td></td>
<td>140.4</td>
<td>126.9</td>
<td>35.1</td>
<td>59.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>108</td>
<td>118.8</td>
<td>24.3</td>
<td>56.7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>89.1</td>
<td>124.2</td>
<td>21.6</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>151.2</td>
<td>102.6</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>116.1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>743JHE (h)</td>
<td>11</td>
<td>25.34</td>
<td>40.53</td>
<td>9.86</td>
<td>11.26</td>
</tr>
<tr>
<td></td>
<td></td>
<td>30.98</td>
<td>42.24</td>
<td>4.22</td>
<td>12.67</td>
</tr>
<tr>
<td></td>
<td></td>
<td>29.57</td>
<td>39.62</td>
<td>7.04</td>
<td>14.08</td>
</tr>
<tr>
<td></td>
<td></td>
<td>28.16</td>
<td>28.16</td>
<td>1.41</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>26.75</td>
<td>32.38</td>
<td>5.63</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>33.79</td>
<td>25.34</td>
<td>2.82</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>23.94</td>
<td></td>
<td></td>
</tr>
<tr>
<td>743JHE (i)</td>
<td></td>
<td>3.66</td>
<td>5.85</td>
<td>1.46</td>
<td>1.46</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4.39</td>
<td>5.12</td>
<td>0.73</td>
<td>2.19</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2.19</td>
<td>4.39</td>
<td></td>
<td>2.93</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3.66</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Case</td>
<td>Pixel Height</td>
<td>ILL. in Number (cd/m²)</td>
<td>ILL. in Letter (cd/m²)</td>
<td>ILL. in White Area (cd/m²)</td>
<td>ILL. at Background (cd/m²)</td>
</tr>
<tr>
<td>------------</td>
<td>--------------</td>
<td>------------------------</td>
<td>------------------------</td>
<td>----------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>YGN66 (a)</td>
<td>88</td>
<td>1.06</td>
<td>0.53</td>
<td>2.13</td>
<td>2.13</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.53</td>
<td>0.0</td>
<td>4.25</td>
<td>1.59</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.0</td>
<td>0.0</td>
<td>3.19</td>
<td>2.66</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.59</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>YGN66 (b)</td>
<td>35</td>
<td>0.73</td>
<td>0.0</td>
<td>15.96</td>
<td>8.71</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2.90</td>
<td>2.18</td>
<td>12.33</td>
<td>9.43</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4.35</td>
<td>4.35</td>
<td>13.78</td>
<td>7.98</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2.18</td>
<td>1.45</td>
<td>10.16</td>
<td>9.43</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2.90</td>
<td>15.24</td>
<td>10.16</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.73</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>YGN66 (c)</td>
<td>32</td>
<td>0.61</td>
<td>0.31</td>
<td>2.15</td>
<td>1.84</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.92</td>
<td>0.61</td>
<td>3.99</td>
<td>1.54</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.0</td>
<td>0.0</td>
<td>3.68</td>
<td>2.15</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.23</td>
<td>0.92</td>
<td>3.38</td>
<td>5.22</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1.54</td>
<td>3.07</td>
<td>4.30</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4.61</td>
</tr>
<tr>
<td>YGN66 (d)</td>
<td>13</td>
<td>3.66</td>
<td>5.85</td>
<td>19.02</td>
<td>10.26</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2.17</td>
<td>6.58</td>
<td>14.63</td>
<td>8.78</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.73</td>
<td>5.12</td>
<td>15.36</td>
<td>8.05</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4.39</td>
<td>8.78</td>
<td>11.70</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2.93</td>
<td>17.56</td>
<td>15.36</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.73</td>
<td>16.82</td>
<td>3.66</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1.46</td>
<td>16.09</td>
<td>1.46</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2.19</td>
<td></td>
<td>18.29</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3.66</td>
<td></td>
<td>51.94</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>7.32</td>
<td></td>
<td>72.42</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>84.85</td>
</tr>
</tbody>
</table>
Appendix C.1  Program Guide

Software has been developed to automatically recognize license plate characters. The program is written in Borland C++ 2.0, and runs on Microsoft Windows 3.0. It is attempted to modularize the program to increase the legibility and allow reusability of some of its routines. The variables and routines belonging to the same processes are gathered and arranged in one program file. All the names of the variables and routines are carefully selected so that they reflect the usages of the variables or the functions of the routines (see the following examples). Their naming also follows the popular naming convention (use underscore '_' and capital case letter to separate words in names). Comments are added to the program to make it easy to understand. However, knowledge of C language, DOS and Microsoft Windows programming experience are expected in order to understand the program.

The program consists of two main parts: the preprocessor and the neural network. The preprocessor separates all characters from a picture. The neural network recognizes the characters output from the preprocessor. The neural network has a three-layer architecture using a back-propagation training method. The overall structure of the program is illustrated in Figure C.1.

(1) The Main Subroutine

Like any C programs, this program has a main subroutine as the starting point of the program. When the program is loaded, the main subroutine is called (in the C program file DETECT.C). As a Microsoft Windows program, the main subroutine has a name of WinMain. The main subroutine creates a main window and the menu bar with its pull-down menus (for hardcopy, see User Guide). The menu structure is specified in the Windows resource file DETECT.RC.
Figure C.1 Structure of the Program
Resources are graphical objects used in Windows programs, such as icons, menus, cursors. The resource file is a text file with the file extension .RC, and contains text representations of resources used in the program.

After the program is loaded, the system waits for user's commands. The routine *WndProc* interprets the commands, and calls the corresponding preprocessing functions or the neural network functions. The constants for the menu selections are defined in the C header file *DETECT.H*. It closes the main window when the user exits the program.

(2) The Preprocessing Subprogram

The preprocessing subprogram is called when the system starts the preprocessing. This subprogram consists of several subroutines as shown in Figure C.2. The subroutines for loading image, compressing image are in the C program file *SCANNER.C*. The subroutines for edge enhancement, thresholding, horizontal separation, image identification, vertical scanning, scaling and saving output are in the C program file *PROCESS.C*. These routines are called in the same sequence as described in the preprocessing section in Chapter Three (shown in Figure 3.2).

The subroutine *LoadImage* is first called to load a CapCalc picture (512x480 format) in the system. It prompts for the file name of the license plate picture and the file name of the compressed output. The routines for entering the file name are in the C program file *FILEDLG.C*. Their constants are defined in the C header file *FILEDLG.H*. After the file name is entered, the program allocates storage and loads the image row by row (total 480 rows) into the 512x480 array *Image*. The gray levels of the pixels are converted from floating-point values to integers for every row. The pixel values (0-255) of each row are then written to the output file.
Figure C.2  The Preprocessing Sequence

Picture File

Compressing: floating-point number -> integer

Compressed data

Vertical Edge Enhancement
Thresholding: convert to binary

Picture with vertical edges

Horizontal Block Separation: find most significant row block

Lowest and highest row Nos.

Character row block

Image Identification: whether it's white-on-black, or black-on-white

Gray level=1, if signal pixel; =0, otherwise

Binary picture

Noise Filtering

Filtered data

Vertical Block Separation

Character blocks

Character Scaling: divide each block to 7x9 areas, convert each area to 1 or 0

Character Arrays
Programmers should be very careful when handling the memory allocation. The Microsoft Windows provides access to memory beyond the 640K limited by the DOS. To allocate these memories, a handle must first be obtained by the routine GlobalAlloc. A handle is simply a number that refers to an object (like memory). The routine GlobalLock is called to have exclusive access to the memory. The return values of both routines should be checked to make sure that the memory is actually allocated before it can be used.

The program then tries to separate the characters from the compressed image. The subroutine Convolution is called to enhance the vertical edges in the picture. The gray levels of the picture are passed to it in the array Image. The subroutine applies a convolution kernel to the picture row by row. To increase the speed, the summation statement is used in stead of the for loop to calculate the sums of the convolution (see Equation 3-1). The outputs of the convolution are generated and saved into another 512x480 array named Buffer. The gray values of the array Image remain the same, and are used in the subsequent image identification and vertical scan processings. The thresholding is performed to produce a binary image (showing the vertical edges) in the array Buffer.

The program calls the subroutine HorizontalScan to separate the character block from the picture, that is, to find out the row numbers of the starting row and the ending row of the block. The pixel values in the array Buffer are summed for each row, and the sums are saved in the array RowSum. All the 480 rows are grouped into several blocks based on the results of the summation. The width of each block is calculated as the starting row number of the block minus the ending row number plus one. All the widths are compared to each other, and the block with the maximum width is considered as the character row block. The starting row number is saved in the variable start_row, and the
ending row number is saved in another variable \texttt{end\_row}. The search for characters is performed only within this block.

The image is identified as either black-on-white or white-on-black based on the average gray levels of the character block and the other blocks in the subroutine \texttt{Imageldentification}. The image is then converted into a binary image with the signal value of 1 and the background value of 0. The subroutine \texttt{VerticalScan} is called to perform a vertical scan in a similar way as the subroutine \texttt{HorizontalScan}. The coordinates of the character blocks separated are saved in the array \texttt{all\_chars}. The subroutine \texttt{ScaleCharacter} is called to scale the character blocks into 7x9 arrays based on the information in the array \texttt{all\_chars}. The output 7x9 arrays are saved in the array \texttt{OutImage}.

The 7x9 character arrays may be saved to a file by calling the subroutine \texttt{OutputCharacter}. It prompts for the name of the output file. The arrays are then saved into the file one by one. For each array, the 7x9 numbers are arranged in rows, one at a time.

\begin{itemize}
\item \textbf{(3) The Neural Network Subprogram}
\item The neural network uses a back-propagation training method. The network training consists of a forward pass and a backward pass. In the forward pass, the training patterns (7x9 character arrays) are input to the network, and the outputs are calculated. In the backward pass, the outputs are compared to the desired outputs (the 6-digit codes). The pattern errors are then propagated back to the output and the hidden layers. Weights are adjusted based on the errors propagated back. However, the network recognition involves only the forward pass. The outputs of the network are considered as the result of the recognition. The processing of the neural network is illustrated in Figure C.3.
\end{itemize}
A pattern = A character array and its six-digit code

Figure C.3 The Operation of the Neural Network
When the training starts, a three-layer network is first set up, and all the patterns (character arrays and their 6-digit codes) are loaded into an input buffer \textit{Pattern} in the routine \texttt{read_pattern}. All the network variables, including \textit{Pattern}, are defined in the C program file \texttt{NEURAL.C}. The data types and function definitions are defined in the C header file \texttt{NEURAL.H}. All the routines for loading the patterns, setting up the network, saving the weights, and loading the weights are described in the C program file \texttt{FILEIO.C}. When the network is being set up, the array \texttt{neuron} is allocated to store all the neurons. The array \texttt{wgtHidIn} is allocated for the weights between the input and the hidden layers. The array \texttt{wgtOutHid} is allocated for the weights between the hidden and the output layers. All the weights are initialized to small numbers.

The training procedure consists of many iterations. Each iteration could be considered as a process in which all patterns go through a forward pass and a backward pass one by one. The routines \texttt{train_network} and \texttt{backpropagation} are used to control these iterations and record the training time. They are described in the C program file \texttt{NEURAL.C}. All the routines for the forward and the backward passes are described in the C program file \texttt{NETWORK.C}.

Each pattern first goes through the forward pass in the routine \texttt{forward_pass}. The patterns stored in the array \texttt{Pattern} are copied to the input layer. The outputs of the input, the hidden, and the output layers are calculated. The outputs of the output layer are compared to the desired outputs, and the deviations of the output layer are calculated in the routine \texttt{calc_output_error}. Any output deviation less than 0.01 is considered acceptable. Any deviation equal to or greater than 0.01 is considered as an error, and is used to train the network. The errors are propagated back to the hidden layer, and the errors of the hidden layer are calculated in the routine \texttt{calc_hidden_error}. The weights
between the output layer and the hidden layer are adjusted in the routine \texttt{adjust_output_weights}. The weights between the output layer and the hidden layer are adjusted in the routine \texttt{adjust_hidden_weights}.

The same procedure are performed on each subsequent pattern. The training finishes one iteration when all patterns go through this procedure. The total error of this iteration is displayed on the main window. Note that text can be printed on the screen with the C function \texttt{printf} in DOS applications. But Microsoft Windows programs are actually graphics applications. This means that the program must have the information (like size) of the font being used. So it knows where to print the next line without overwriting the previous line. The dimension of a font could be obtained by the function \texttt{GetTextMetrics}. The function \texttt{TextOut} is used to print text in a window with the exact position in pixels.

The training continues until no error is found for all patterns in the forward pass. The network weights are saved into a file in the routine \texttt{write_weights}. Before the network recognizes characters, the network weights are loaded into the arrays \texttt{wgtHidln} and \texttt{wgtOutHid} in the routine \texttt{read_weights}. The routine \texttt{recognition} then calls the forward-pass routine \texttt{forward_pass} to generate the output, and the output is displayed on a window.
Appendix C.2  Program Listing
## C Program File Listing

<table>
<thead>
<tr>
<th>No.</th>
<th>File</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>DETECT.H</td>
<td>198</td>
</tr>
<tr>
<td>2.</td>
<td>FILEDLG.H</td>
<td>199</td>
</tr>
<tr>
<td>3.</td>
<td>GLOBAL.H</td>
<td>200</td>
</tr>
<tr>
<td>4.</td>
<td>HEADER.H</td>
<td>202</td>
</tr>
<tr>
<td>5.</td>
<td>MAINWIN.H</td>
<td>203</td>
</tr>
<tr>
<td>6.</td>
<td>NEURAL.H</td>
<td>204</td>
</tr>
<tr>
<td>7.</td>
<td>DETECT.RC</td>
<td>206</td>
</tr>
<tr>
<td>8.</td>
<td>DETECT.DEF</td>
<td>210</td>
</tr>
<tr>
<td>9.</td>
<td>DETECT.C</td>
<td>211</td>
</tr>
<tr>
<td>10.</td>
<td>FILEDLG.C</td>
<td>224</td>
</tr>
<tr>
<td>11.</td>
<td>FILEIO.C</td>
<td>231</td>
</tr>
<tr>
<td>12.</td>
<td>NETWORK.C</td>
<td>242</td>
</tr>
<tr>
<td>13.</td>
<td>NEURAL.C</td>
<td>250</td>
</tr>
<tr>
<td>14.</td>
<td>PROCESS.C</td>
<td>255</td>
</tr>
<tr>
<td>15.</td>
<td>SCANNER.C</td>
<td>272</td>
</tr>
</tbody>
</table>
/*-------------------------
DETECT.H header file
Define the constants for the menu and dialog-box
-------------------------*/

#define IDM_LOAD 91
#define IDM_SAVE 92
#define IDM_EXIT 93

#define IDM_TRAIN 10
#define IDM_CONT 12
#define IDM_RESET 13
#define IDM_NSET 14
#define IDM_RECOGNIZ 15
#define IDM_RECOFILE 16

#define IDM_CLRWND 20
#define IDM_LOADIMG 21
#define IDM_PROCESS 22
#define IDM_VIEWIMG 24
#define IDM_COMPIMG 25

#define IDM_VERTEDGE 31
#define IDM_HORISCAN 32
#define IDM_IMGIDENT 33
#define IDM_VERTSCAN 34
#define IDM_SCALCHAR 35
#define IDM_SAVECHAR 36

#define IDM_ABOUT 41
#define IDM_DSOUT 50
#define IDM_PARAM 51
#define IDM_DSERR 52
#define IDM_MTASK 53

#define IDE_LCOE 101
#define IDE_MOME 102
#define IDE_AERR 103
FILEDLG.H header file

Define the constants for file saving/entering dialog-boxes

*sizeof(…)*/

#define IDD_FNAME 0x10
#define IDD_FPATH 0x11
#define IDD_FLIST 0x12
/* GLOBAL.H: header file for global variables and functions */

#define out_width 7  /* character width */
#define out_heigh 9  /* character height */

/* global variables */
extern HWND hMainWnd;
extern HANDLE hMainInstance;
extern char szAppName[];
extern char szBuffer[1000];
extern short cxChar, cxCaps, cyChar;
extern RECT rect;
extern HDC hdc;
extern PAINTSTRUCT ps;
extern BOOL isDisplayError;
extern BOOL MultiTasking;
extern int Reset;
extern int Trained;
extern WORD DataReady;
extern WORD NumOfChar;
extern WORD hasBuffer;
extern float CurrentError, Delta;
extern float Speed_Factor;
extern WORD DisplayOutput;

/* Global Variables */
extern const WORD max_row;  /* max row number in image, default 480 */
extern const WORD max_col;  /* max row number in image, default 512 */

extern HANDLE hHourGlass;  /* handle to hourglass cursor */
extern HANDLE hSaveCursor;  /* current cursor handle */

/* variables of file names */
extern char szWgtFileName[96];
extern char szDatFileName[96];
extern char szPtnFileName[96];
extern char szOutFileName[96];
extern char szExpFileName[96];

/* variables for output file names */
extern char szOutFileName[96];
extern char szOutFileSpec[];
/* dialog-box id */
#define IDD_OUTF 201
#define IDE_OUTF 202
#define IDD_WIDT 203
#define IDE_WIDT 204
#define IDD_HEIG 205
#define IDE_HEIG 206

/* Functions for file open and save */
int DoFileOpenDlg(HANDLE hInst, HWND hwnd, char *szFileSpecIn, char *szDefExtIn, WORD wFileAttrIn, char *szFileNameOut, POFSTRUCT pofIn);
int DoFileSaveDlg(HANDLE hInst, HWND hwnd, char *szFileSpecIn, char *szDefExtIn, WORD *pwStatusOut, char *szFileNameOut, POFSTRUCT pofIn);
LPSTR lstrrchr(LPSTR str, char ch);

/* Variables for opening and saving file */
extern WORD wStatus;
extern OFSTRUCT of;

/* Arrays to store pictures */
extern HANDLE hImage[480];
extern HANDLE hBuffer[480];
extern BYTE far *Image[480]; /* array to store the image */
extern BYTE far *Buffer[480]; /* array for processing use */
extern BYTE OutImage[7][9][7];

/* Function declarations */
void statistic(void);
void row_statistic(void);
extern long frequency[512];

/* Declarations of pre-processing functions */
void VerticalEdgeEnhance(void);
void ImagIdentification(void);
void HorizontalScan(void);
BOOL VerticalScan(void);
void ScaleCharacter(void);
void OutputCharacter(void);
/* HEADER.H: all include files */

#include <stdlib.h>
#include <stdio.h>
#include <malloc.h>
#include <string.h>
#include <math.h>
#include <time.h>
#include <windows.h>
#include "detect.h"
#include "neural.h"
#include "global.h"
extern HWND hMainWnd;
extern HANDLE hMainInstance;
extern char szAppName[
extern char szBuffer [1000];
extern short cxChar, cxCaps, cyChar;
extern RECT rect;
extern HDC hdc;
extern PAINTSTRUCT ps;
extern BOOL isDisplayError;
extern BOOL MultiTasking;
extern int Reset;
extern int Trained;

extern HANDLE hHourGlass;       /* handle to hourglass cursor */
extern HANDLE hSaveCursor;      /* current cursor handle */

extern char szWgtFileName[96];
extern char szDatFileName[96];
extern char szPtnFileName[96];
extern char szOutFileName[96];
extern char szExpFileName[96];

#define IDD_OUTF 201
#define IDE_OUTF 202
#define IDD_WIDT 203
#define IDE_WIDT 204
#define IDD_HEIG 205
#define IDE_HEIG 206
/* Neural.H
Define the structures and functions for neural network*/

#ifndef NEURAL_H
#define NEURAL_H

#define OUTPUT_LAYER (Layer_Num - 1)

#define MAX_LAYERS 3
#define MAX_HIDDEN_NEURONS 50
#define MAX_OUTPUT_NEURONS 10
#define MAX_PATTERNS 400

typedef struct Neuron{
    float input;
    float output;
    float error;
} NEURON;

typedef struct Weight{
    float weight;
    float delta;
} WEIGHT;

void setRandom(WEIGHT far*con);
void displaySelf(WEIGHT far*con);
void adjust_output_weights(void);
void adjust_hidden_weights(void);

typedef struct Pattern{
    float in[63];
    float out[6];
    float error;
} PATTERNS;

extern float Learning_Coefficient;
extern float Momentum;
extern int Noise_Type;
extern int Noise_Value;
extern int Noise_Range;

extern NEURON far* neuron[MAX_LAYERS];
extern WEIGHT far* wgtOutHid[MAX_OUTPUT_NEURONS];
extern WEIGHT far* wgtHidIn [MAX_HIDDEN_NEURONS];
extern HANDLE hNeuron[MAX_LAYERS];
extern HANDLE hWgtOutHid[MAX_OUTPUTNEURONS];
extern HANDLE hWgtHidIn [MAX_HIDDENNEURONS];
extern HANDLE hPattern;
extern PATTERNS FAR* Pattern;
extern int Layer_Num;
extern int Neuron_Num;
extern int Num_InLayer[MAX LAYERS];
extern int Pattern_Num;
extern int Cur_Pattern;
extern int Patn_Width, Patn_Depth, outWidth, outDepth;
extern int iteration;
extern float Acceptable_Error;
extern float Total_Error;

BOOL PatternAlloc(void);
void PatternFree(void);
void adjust_weights(void);
void calc_output_error(void);
void calc_hidden_error(void);
int trained(void);
int read_pattern(char*);
void DisplayError(void);
void display(void);
void displayTotalError(void);
void displayPerformance(unsigned long);
void forward_pass(void);
void backward_pass(void);
void recognition(void);
int all_patterns(void);

#endif
#include <windows.h>
#include "detect.h"
#include "filedlg.h"

Detectlcon ICON detect.ico

DetectMenu MENU {
    POPUP "&System" {
        MENUITEM "&Load Weights", IDM_LOAD
        MENUITEM "&Save Weights", IDM_SAVE, GRAYED
        MENUITEM SEPARATOR
        MENUITEM "&Clear Window", IDM_CLRWND
        MENUITEM "&Exit", IDM_EXIT
    }
    POPUP "&Network" {
        MENUITEM "&Training", IDM_TRAIN
        MENUITEM SEPARATOR
        MENUITEM "&Continue", IDM_CONT, GRAYED
        MENUITEM "&Reset", IDM_RESET, GRAYED
        MENUITEM "&Next Train Set", IDM_NSET, GRAYED
        MENUITEM SEPARATOR
        MENUITEM "&Recognize", IDM_RECOGNIZ, GRAYED
        MENUITEM "Recognize &File", IDM_RECOFILE, GRAYED
    }
    POPUP "&Image" {
        MENUITEM "&Compress Image", IDM_COMPIMG
        MENUITEM "&Load Image", IDM_LOADIMG
        MENUITEM "&View Image", IDM_VIEWIMG, GRAYED
        MENUITEM SEPARATOR
        MENUITEM "&Pre-Processing", IDM_PROCESS, GRAYED
        MENUITEM "&Save Character", IDM_SAVECHAR, GRAYED
    }
    POPUP "&Processing" {
        MENUITEM "&Vertical Edge Enhance", IDM_VERTEDGE, GRAYED
        MENUITEM "&Horizontal Scan", IDM_HORISCAN, GRAYED
        MENUITEM SEPARATOR
        MENUITEM "&Image Identification", IDM_IMGIDENT, GRAYED
        MENUITEM "&Vertical Scan", IDM_VERTSCAN, GRAYED
        MENUITEM SEPARATOR
        MENUITEM "&Scale Character", IDM_SCALCHAR, GRAYED
    }
}"
ABOUTBOX DIALOG DISCARDABLE LOADONCALL PURE MOVEABLE 40, 40, 165, 107
STYLE WS_POPUP | WS_CLIPSIBLINGS | WS_DLFRAME
BEGIN
  CONTROL "Neural Network Based" -1, "STATIC", WS_CHILD | WS_VISIBLE | WS_GROUP | 0x1L, 41, 20, 89, 12
  ICON "DetectIcon" -1, 8, 8, 0, 0
  CONTROL "Automatic Recognition of License Plate Characters" -1, "STATIC", WS_CHILD | WS_VISIBLE | WS_GROUP | 0x1L, 21, 33, 124, 17
  CONTROL "OK" 1, "BUTTON", WS_CHILD | WS_VISIBLE | WS_GROUP | WS_TABSTOP | 0x1L, 60, 79, 40, 14
  CONTROL "ISE Dept., Ohio University, 1992" 104, "STATIC", WS_CHILD | WS_VISIBLE, 30, 57, 110, 11
END

SETTING DIALOG DISCARDABLE LOADONCALL PURE MOVEABLE 26, 30, 218, 127
STYLE WS_POPUP | WS_VISIBLE | WS_CAPTION | 0x40L
CAPTION "Setting Network Parameters"
FONT 10, "Helv"
BEGIN
  CONTROL "Learning Coefficient:" -1, "STATIC", WS_CHILD | WS_VISIBLE, 34, 18, 78, 9
  CONTROL "$ IDE_LCOE, "EDIT", WS_CHILD | WS_VISIBLE | WS_BORDER | WS_TABSTOP | 0x82L, 114, 16, 57, 11
  CONTROL "Momentum:" -1, "STATIC", WS_CHILD | WS_VISIBLE, 66, 41, 46, 8
  CONTROL "$ IDE_MOME, "EDIT", WS_CHILD | WS_VISIBLE | WS_BORDER | WS_TABSTOP | 0x82L, 114, 40, 57, 11
  CONTROL "Acceptable Error:" -1, "STATIC", WS_CHILD | WS_VISIBLE, 45, 65, 66, 9
  CONTROL "$ IDE_AERR, "EDIT", WS_CHILD | WS_VISIBLE | WS_BORDER | WS_TABSTOP | 0x82L, 114, 64, 57, 11
CONTROL "OK" IDOK, "BUTTON", WS_CHILD | WS_VISIBLE |
WS_TABSTOP, 57, 94, 34, 12
CONTROL "Cancel" IDCANCEL, "BUTTON", WS_CHILD | WS_VISIBLE |
WS_TABSTOP, 129, 94, 34, 12
CONTROL "OK to accept change, Cancel to discard change" -1, "STATIC", 
WS_CHILD | WS_VISIBLE, 29, 115, 172, 9
END

/*-----------------------------*/
FILEDLG.DLG dialog definition
-----------------------------*/

FileOpen DIALOG 10, 10, 148, 116
STYLE WS_POPUP | WS_DLGFRA ME {
   LTEXT l "File &Name:" , -1, 2, 4, 76, 10
   EDITTEXT l IDD_FNAME, 2, 18, 100, 12,
   ES_AUTOHSCROLL
   LTEXT "&Files in", -1, 2, 40, 38, 10
   LTEXT "", IDD_FPATH, 44, 40, 98, 12
   LISTBOX IDD_FLIST, 2, 54, 70, 58,
   WS_TABSTOP | WS_VSCROLL
   DEFPUSHBUTTON "&Select", IDOK, 88, 62, 50, 14,
   WS_GROUP
   PUSHBUTTON "Cancel", IDCANCEL, 88, 86, 50, 14,
   WS_GROUP
}

FileSave DIALOG 10, 10, 180, 54
STYLE WS_POPUP | WS_DLGFRA ME {
   LTEXT l "Save File &Name As:" , -1, 6, 4, 84, 12
   LTEXT "", IDD_FPATH, 96, 4, 78, 12
   EDITTEXT l IDD_FNAME, 6, 20, 104, 12,
   ES_AUTOHSCROLL
   DEFPUSHBUTTON "OK", IDOK, 124, 20, 50, 14, WS_GROUP
   PUSHBUTTON "Cancel", IDCANCEL, 124, 36, 50, 14,
   WS_GROUP
}

VIEWIMAG DIALOG DISCARDABLE LOADONCALL PURE MOVEABLE 0, -2,
320, 222
CAPTION "Viewing Picture"
STYLE DS_SYSMODAL | DS_MODALFRAME | WS_POPUP | WS_CAPTION |
WS_SYSMENU | WS_MAXIMIZEBOX
BEGIN
```
;-------------------------------------
; DETECT.DEF module definition file
;-------------------------------------

<table>
<thead>
<tr>
<th>NAME</th>
<th>DETECT</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESCRIPTION</td>
<td>'License Plate Recognition (c) Songqing Chen, 1992'</td>
</tr>
<tr>
<td>EXETYPE</td>
<td>WINDOWS</td>
</tr>
<tr>
<td>STUB</td>
<td>'WINSTUB.EXE'</td>
</tr>
<tr>
<td>CODE</td>
<td>PRELOAD MOVEABLE DISCARDABLE</td>
</tr>
<tr>
<td>DATA</td>
<td>PRELOAD MOVEABLE MULTIPLE</td>
</tr>
<tr>
<td>HEAPSIZE</td>
<td>8192</td>
</tr>
<tr>
<td>STACKSIZE</td>
<td>10240</td>
</tr>
</tbody>
</table>
```
#include "header.h"  /* include definitions */

/* Function declarations for menu commands */
extern void network_init(void);
extern int train_network(int);
extern void recognition(void);
extern void finishup(void);
extern BOOL LoadData(void);
extern void ViewData(HWND);
extern void preProcessing(void);
extern BOOL read_weights(char *);
extern BOOL write_weights(char *);
extern BOOL read_characters(char *);
extern void EnableSystem(void);
extern void DisableSystem(void);
extern void SaveImage(void);

/* Function declarations for dialog-boxes */
long FAR PASCAL WndProc(HWND, WORD, WORD, LONG);
BOOL FAR PASCAL SettingDlgProc(HWND hDlg, WORD message, WORD wParam, LONG lParam);
BOOL FAR PASCAL ViewDataDlgProc(HWND hDlg, WORD message, WORD wParam, LONG lParam);
BOOL FAR PASCAL RowHistogramDlgProc(HWND hDlg, WORD message, WORD wParam, LONG lParam);

/* Global variables for preprocessing */
extern WORD DataReady;     /* preprocessing status */
extern long max_freq, threshold;
extern long frequency[512];

HANDLE hHourGlass;         /* handle to hourglass cursor */
HANDLE hSaveCursor;         /* current cursor handle */

/* Variables to store network status */
int Reset = 1;            /* is network reset */
int Trained = 0;           /* has network been trained */
int Loaded = 0; /* has picture been loaded */
BOOL isDisplayError = FALSE; /* display pattern errors */
BOOL MultiTasking = FALSE; /* multitasking in training */

/* Network parameters */
float Learning_Coefficient = 0.3;
float Momentum = 0.8;
float Acceptable_Error = 0.1;
float Total_Error = 0.1;
float Speed_Factor = 0.2;

/* Filename and their default extensions */
char szWgtFileName[96] = "*.WGT";
char szDatFileName[96] = "*.Dat";
char szPtnFileName [96] = "*.PTN";
char szExpFileName [96] = "*.EXP";
char szOutFileName [96] = "*.OUT";
char szOutFileSpec [] = "*.OUT";
char szExpFileSpec [] = "*.EXP";
char szWgtFileSpec [] = "*.WGT";
char szDatFileSpec [] = "*.DAT";
char szPtnFileSpec [] = "*.PTN";
WORD wStatus;
OFSTRUCT of;

char szAppName[] = "Detect"; /* name of the system */
char szBuffer [1000]; /* generic buffer */
HWND hMainWnd; /* handle of the main window */
HANDLE hMainInstance; /* handle of the program */
short cxChar, cxCaps, cyChar; /* information of current font */
RECT rect; /* display rectangle */
HDC hdc; /* handle of display context */
PAINTSTRUCT ps; /* pen information */

/* Program entry, initialize the system */
int PASCAL WinMain(HANDLE hInstance, HANDLE hPrevInstance,
LPSTR lpszCmdLine, int nCmdShow)
{
    MSG msg;
    WNDCLASS wndclass;

    if (hPrevInstance) {
        /* Specify parameters of this program */
        wndclass.style = CS_HREDRAW | CS_VREDRAW;
wndclass.lpfnWndProc = WndProc;
wndclass.cbClsExtra = 0;
wndclass.cbWndExtra = 0;
wndclass.hInstance = hInstance;
wndclass.hlcon = LoadIcon(hInstance, "DetectIcon");
wndclass.hCursor = LoadCursor(NULL, IDC_ARROW);
wndclass.hbrBackground = GetStockObject(WHITE_BRUSH);
wndclass.lpszMenuName = "DetectMenu";
wndclass.lpszClassName = szAppName;
RegisterClass(&wndclass);
} else
    exit (-1);

hMainInstance = hInstance;

/* Create main window */
hMainWnd = CreateWindow(szAppName, "License Plate Recognition",
WS_OVERLAPPEDWINDOW,
CW_USEDEFAULT, CW_USEDEFAULT,
CW_USEDEFAULT, CW_USEDEFAULT,
NULL, NULL, hInstance, NULL);

/* Get an hourglass cursor to use */
hHourGlass = LoadCursor(NULL, IDC_WAIT);
ShowWindow(hMainWnd, nCmdShow);
UpdateWindow(hMainWnd);

/* Retrieve messages from Windows */
while (GetMessage(&msg, NULL, 0, 0)) {
    TranslateMessage(&msg);
    DispatchMessage(&msg);
}
return msg.wParam;
}

/* Process messages for about dialog-box */
BOOL FAR PASCAL AboutDlgProc(HWND hDlg, WORD message, WORD wParam, LONG lParam)
{
    switch (message) {
    case WM_INITDIALOG:
        return TRUE;
    }
case WM_COMMAND:
    switch (wParam) {
    case IDOK:       /* End the dialog-box */
        EndDialog(hDlg, 0);
        return TRUE;
        break;
    case IDCANCEL:
        EndDialog(hDlg, 0);
        return TRUE;
    }

return FALSE;
}

/* Process all the commands user selects */
long FAR PASCAL WndProc(HWND hwnd, WORD message, WORD wParam, LONG lParam)
{
    static FARPROC lpfnAboutDlgProc, lpfnSettingDlgProc;
    static HANDLE hInstance;
    static HMENU hMenu;

    /* Variables used to display text */
    HDC hdc;
    short i, j, x, y;
    PAINTSTRUCT ps;
    TEXTMETRIC tm;

    switch (message) {
    case WM_CREATE:                /* Main window is being created */
        network_init();            /* initialize the network */

        hInstance = ((LPCREATESTRUCT) lParam)->h Instance;
        lpfnAboutDlgProc    = MakeProcInstance(AboutDlgProc,
        hInstance);

        hMenu = GetMenu(hwnd);
        CheckMenuItems(hMenu, IDM_DSOUT, MF_CHECKED);

        /* Get the font size information */
        hdc = GetDC(hwnd);

        GetTextMetrics(hdc, &tm);
        cxChar = tm.tmAveCharWidth;
cxCaps = (tm.tmPitchAndFamily & 1 ? 3 : 2) * cxChar / 2;
cyChar = tm.tmHeight + tm.tmExternalLeading;

ReleaseDC(hwnd, hdc);

return 0;

case WM_ACTIVATEAPP:
    /* Process multitasking
    if (wParam)
        EnableSystem();
    else
        DisableSystem();
    return 0;
    */

case WM_COMMAND:
    switch (wParam) {
        case IDM_LOAD:
            /* Load network weights
            if (DoFileDialog(hwnd, hMainWnd,
                szWgtFileSpec, szWgtFileName, &of))
                if (read_weights(szWgtFileName)) {
                    /* Enable network commands
                    EnableMenuItem(hMenu, IDM_SAVE, MF_ENABLED);
                    EnableMenuItem(hMenu, IDM_RECOGNIZ, MF_ENABLED);
                    EnableMenuItem(hMenu, IDM_RECOFILE, MF_ENABLED);
                    EnableMenuItem(hMenu, IDM_NSET, MF_ENABLED);
                    EnableMenuItem(hMenu, IDM_RESET, MF_ENABLED);
                }
            return 0;
            */

        case IDM_SAVE:
            /* Save network weights
            */
if (DoFileSaveDlg(hInstance, hMainWnd, szWgtFileSpec, szWgtFileName, &of))
{
    write_weights(szWgtFileName);
    EnableMenultem(hMenu, IDM_SAVE, MF_ENABLED);
}
return 0;

case IDM_EXIT:       /* Exit the system */
    SendMessage(hwnd, WM_CLOSE, 0, 0L);
    return 0;

case IDM_TRAIN:      /* Train the network */
    case IDM_NSET:
        if (!DoFileOpenDlg(hInstance, hMainWnd, szPtnFileSpec, szPtnFileName, &of))
            return 0;
    case IDM_CONT:      /* Continue training */
        if (wParam == IDM_CONT) {
            if (!Reset) {
                Trained = train_network(FALSE);
            } else {
                MessageBox(hwnd, "Please start training first", szAppName, MB_ICONINFORMATION | MB_OK);
            }
        } else {
            if (wParam == IDM_TRAIN) {
                finishup(); Reset = 0;
                Trained = train_network(TRUE);
            } else {
                Trained = train_network(2);
            }
        }

        /* Enable or disable network commands */
        if (Trained) {
            EnableMenultem(hMenu, IDM_NSET, MF_ENABLED);
        }
    }
EnableMenuItem(hMenu,
IDM_RECOGNIZ, MF_ENABLED);
EnableMenuItem(hMenu,
IDM_RECOFILE, MF_ENABLED);
} else {
    EnableMenuItem(hMenu,
IDM_RECOGNIZ, MF_GRAYED);
    EnableMenuItem(hMenu,
IDM_RECOFILE, MF_GRAYED);
    EnableMenuItem(hMenu,
IDM_NSET, MF_GRAYED);
}

if (!Reset) {
    EnableMenuItem(hMenu, IDM_CONT, MF_ENABLED);
    EnableMenuItem(hMenu, IDM_SAVE, MF_ENABLED);
    EnableMenuItem(hMenu, IDM_RESET, MF_ENABLED);
} else {
    EnableMenuItem(hMenu, IDM_SAVE, MF_GRAYED);
    EnableMenuItem(hMenu, IDM_CONT, MF_GRAYED);
    EnableMenuItem(hMenu, IDM_RESET, MF_GRAYED);
}

return 0;

case IDM_RECOGNIZ: /* Recognize patterns */
if (Trained)
    if (DataReady == 6)
        recognition();
else
    MessageBox(hwnd, "Please train network first",
MB_ICONINFORMATION | MB_OK);
return 0;

case IDM_RECOFILE: /* Recognize patterns in a file */
if (Trained)
if (DoFileOpenDlg(hInstance, szOutFileSpec + 1, 0x401, szOutFileName, &of))
if (read_characters(szOutFileName))
else
recognition();
else
MessageBox(hwnd, "Please train network first", szAppName, MB_ICONINFORMATION | MB_OK);
return 0;

case IDM_RESET: /* Reset the network */
finishup();
EnableMenuItem(hMenu, IDM_SAVE, MF_GRAYED);
EnableMenuItem(hMenu, IDM_CONT, MF_GRAYED);
EnableMenuItem(hMenu, IDM_NSET, MF_GRAYED);
EnableMenuItem(hMenu, IDM_RECOGNIZ, MF_GRAYED);
EnableMenuItem(hMenu, IDM_RECOFILE, MF_GRAYED);
EnableMenuItem(hMenu, IDM_RESET, MF_GRAYED);
return 0;

case IDM_PARAM: /* Change network settings */
lpfnSettingDlgProc = MakeProcInstance(SettingDlgProc, hInstance);
DialogBox(hInstance, "SETTING", hwnd, lpfnSettingDlgProc);
FreeProcInstance(lpfnSettingDlgProc);
return 0;

case IDM_DSERR: /* turn display error on/off */
if (isDisplayError) {
    isDisplayError = FALSE;
    CheckMenuItem(hMenu, wParam, MF_UNCHECKED);
}
else {
    isDisplayError = TRUE;
    CheckMenuITEM(hMenu, wParam, MF_CHECKED);
}
return 0;

case IDM_DSOUT: /* turn display output on/off */
    if (DisplayOutput) {
        DisplayOutput = FALSE;
        CheckMenuITEM(hMenu, wParam, MF_UNCHECKED);
    } else {
        DisplayOutput = TRUE;
        CheckMenuITEM(hMenu, wParam, MF_CHECKED);
    }
    return 0;

case IDM_MTASK: /* Turn multitasking on/off */
    if (MultiTasking) {
        MultiTasking = FALSE;
        CheckMenuITEM(hMenu, wParam, MF_UNCHECKED);
    } else {
        MultiTasking = TRUE;
        CheckMenuITEM(hMenu, wParam, MF_CHECKED);
    }
    return 0;

case IDM_CLRWND: /* Clear the main window */
    InvalidateRect(hwnd, NULL, TRUE);
    return 0;

case IDM_COMPIMG: /* Compress image */
    if (DoFileOpenDlg(hInstance, hMainWnd, szExpFileSpec, szWgtFileName, &of))
        szExpFileSpec = szExpFileSpec + 1, 0x4010,
    if (DoFileSaveDlg(hInstance, hMainWnd, szDatFileSpec, hMainWnd, szFileSpec, szWgtFileName, &of))
        szExpFileSpec = szExpFileSpec + 1, 0x4010,
szDatFileName, &of))
    {
        hSaveCursor = 0;
        SaveImage();
        SetCursor(hSaveCursor);
    }
    return 0;
}

case IDM_LOADIMG:     /* Load an image into system */
    if (DoFileOpenDlg(hInstance, hMainWnd,
        szDatFileSpec, szDatFileSpec + 1, 0x4010,
        szDatFileName, &of))
    {
        Loaded = LoadData();
        if (Loaded) {
            /* Enable preprocessing commands */
            EnableMenuItem(hMenu,
                IDM_VIEWIMG, MF_ENABLED);       EnableMenuItem(hMenu,
                IDM_PROCESS, MF_ENABLED);       EnableMenuItem(hMenu,
                IDM_SAVECHAR, MF_ENABLED);       EnableMenuItem(hMenu,
                IDM_VERTEDGE, MF_ENABLED);       EnableMenuItem(hMenu,
                IDM_HORISCAN, MF_ENABLED);        EnableMenuItem(hMenu,
                IDM_IMGIDENT, MF_ENABLED);        EnableMenuItem(hMenu,
                IDM_VERTSCAN, MF_ENABLED);        EnableMenuItem(hMenu,
                IDM_SCALCHAR, MF_ENABLED);    */
        }
    } return 0;
}

case IDM_VIEWIMG:     /* Display the image */
    /*
        lpfnSettingDlgProc =
        MakeProcInstance(ViewImageDlgProc, hInstance);
    */
DialogBox(hInstance, "VIEWIMAG", hMainWnd, lpfnSettingDlgProc);
FreeProcInstance(lpfnSettingDlgProc);
return 0;

case IDM_PROCESS: /* Preprocessing the image */
preProcessing();
return 0;

case IDM_VERTEDGE: /* Vertical edge enhancement */
VerticalEdgeEnhance();
return 0;

case IDM_IMGIDENT: /* Image identification */
ImageIdentification();
return 0;

case IDM_HORISCAN: /* Horizontal scan */
HorizontalScan();
return 0;

case IDM_VERTSCAN: /* Vertical scan */
VerticalScan();
return 0;

case IDM_SCALCHAR: /* Scale characters */
ScaleCharacter();
return 0;

case IDM_SAVECHAR: /* Save output characters */
OutputCharacter();
return 0;

case IDM_ABOUT: /* Display system information */
if (DialogBox(hInstance, "AboutBox", hwnd, lpfnAboutDlgProc))
    InvalidateRect(hwnd, NULL, TRUE);
return 0;
}
break;

case WM_DESTROY:    /* Close main window */
    ClearBuffer();
    finishup();
    PatternFree();
    PostQuitMessage(0);
    return 0;

} return DefWindowProc(hwnd, message, wParam, lParam);

/* Change network settings */
BOOL FAR PASCAL SettingDlgProc(HWND hDlg, WORD message, WORD wParam, LONG lParam)
{
    switch (message) {
        case WM_INITDIALOG:    /* Display the current settings */
            SetDlgItemText(hDlg, IDE_MOME, gcvt(Momentum, 6, szBuffer));
            SetDlgItemText(hDlg, IDE_LCOE, gcvt(Learning_Coefficient, 6, szBuffer));
            SetDlgItemText(hDlg, IDE_AERR, gcvt(Acceptable_Error, 6, szBuffer));
            return TRUE;

        case WM_COMMAND:    /* Get the changed settings */
            switch (wParam) {
                case IDOK:
                    GetDlgItemText(hDlg, IDE_MOME, szBuffer, 10);
                    Momentum = atof(szBuffer);
                    GetDlgItemText(hDlg, IDE_LCOE, szBuffer, 10);
                    Learning_Coefficient = atof(szBuffer);
                    GetDlgItemText(hDlg, IDE_AERR, szBuffer, 10);
                    Acceptable_Error = atof(szBuffer);*

                case IDCANCEL:    /* Don't change the settings */
                    EndDialog(hDlg, 0);
                    return TRUE;
            }
    }
    return TRUE;
}
break;

return FALSE;

}
/*---------------------------------------------*/
*     FILEDLG.C -- Open and Close File Dialog Boxes *
*---------------------------------------------*/

/* Include Files */
#include <windows.h>
#include "filedlg.h"

/* Function Declarations */
BOOL FAR PASCAL FileOpenDlgProc(HWND, WORD, WORD, LONG);
BOOL FAR PASCAL FileSaveDlgProc(HWND, WORD, WORD, LONG);
LPSTR lstrchr (LPSTR str, char ch);
LPSTR lstrrchr(LPSTR str, char ch);

/* Global Variables */
static char szDefExt[5];
static char szFileName[96];
static char szFileSpec[16];
static POFSTRUCT pof;
static WORD wFileAttr, wStatus;

/* function for open-file dialog box */
int DoFileOpenDlg(HANDLE hInst, HWND hwnd, char *szFileSpecIn,
char *szDefExtIn, WORD wFileAttrIn,
char *szFileNameOut, POFSTRUCT pofIn)
{
    FARPROC lpfnFileOpenDlgProc;
    int iReturn;

    /* Copy default file extension */
lstrcpy(szFileSpec, szFileSpecIn);
lstrcpy(szDefExt, szDefExtIn);
wFileAttr = wFileAttrIn;
pof = pofIn;

    /* Pop-up the dialog box */
lpfnFileOpenDlgProc = MakeProcInstance(FileOpenDlgProc, hInst);
iReturn = DialogBox(hInst, "FileOpen", hwnd, lpfnFileOpenDlgProc);
FreeProcInstance(lpfnFileOpenDlgProc);
lstrcpy(szFileNameOut, szFileName);
return iReturn;
}
/* function for save-file dialog box */
int DoFileSaveDlg(HANDLE hInst, HWND hwnd, char *szFileSpecIn,
                 char *szDefExtIn, WORD *pwStatusOut,
                 char *szFileNameOut, POFSTRUCT pofIn)
{
    FARPROC lpfnFileSaveDlgProc;
    int iReturn;

    /* Copy default file extension */
    Istrcpy(szFileSpec, szFileSpecIn);
    Istrcpy(szDefExt, szDefExtIn);
    pof = pofIn;

    /* Pop-up the dialog box */
    lpfnFileSaveDlgProc = MakeProcInstance(FileSaveDlgProc, hInst);
    iReturn = DialogBox(hInst, "FileSave", hwnd, lpfnFileSaveDlgProc);
    FreeProcInstance(lpfnFileSaveDlgProc);

    Istrcpy(szFileNameOut, szFileName);
    *pwStatusOut = wStatus;
    return iReturn;
}

/* Process messages for open-file dialog box */
BOOL FAR PASCAL FileOpenDlgProc(HWND hDlg, WORD message,
                                 WORD wParam, LONG lParam)
{
    char cLastChar;
    short nEditLen;

    switch (message) {
        case WM_INITDIALOG: /* Initialize the dialog box */
            SendDlgItemMessage(hDlg, IDD_FNAME, EM_LIMITTEXT, 80, OL);
            DlgDirList(hDlg, szFileSpec, IDD_FLIST, IDD_FPATH,
                       wFileAttr);
            SetDlgItemText(hDlg, IDD_FNAME, szFileSpec);
            return TRUE;

        case WM_COMMAND:
            switch (wParam) {
                case IDD_FLIST: /* User select the file list */
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FLIST:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
                case IDM_FNAME:
                case IDM_FPATH:
switch (HIWORD(lParam)) {
    case LBN_SELCHANGE: 
        /* select a different file */
        szFileName, IDD_FLIST);
        szFileSpec);
        IDD_FNAME, szFileName);

        open a file */
        szFileName, IDD_FLIST)) {
            szFileSpec);
            szFileName, IDD_FLIST,
            IDD_FPATH, wFileAttr);
            IDD_FNAME, szFileSpec);
            IDD_FNAME, szFileName);
            WM_COMMAND, IDOK, 0L);

            } break;

            case IDD_FNAME: 
                /* change file name */
                if (HIWORD(lParam) = = EN_CHANGE)
                    EnableWindow(GetDlgItem(hDlg, IDOK), (BOOL)
                    SendMessage(LOWORD(lParam), WM_GETTEXTLENGTH,
                    0, 0L));

                    return TRUE;

            case IDOK: 
                /* open the selected file */
                if (DlgDirSelect(hDlg, 
                    Lstrcat(szFileName, 
                    SetDlgItemText(hDlg, 
                    return TRUE;

            case LBN_DBLCLK: 
                /* select to open a file */
                szFileName, IDD_FLIST)) {
                    szFileSpec);
                    szFileName, IDD_FLIST,
                    IDD_FPATH, wFileAttr);
                    IDD_FNAME, szFileSpec);
                    IDD_FNAME, szFileName);
                    WM_COMMAND, IDOK, 0L);

                    } else {
                        SetDlgItemText(hDlg, 
                        SendMessage(hDlg, 
                        return TRUE;
szFileName, 80);
szFileName + nEditLen);

Istrchr(szFileName, '?') {
  if (cLastChar = = '\\' || cLastChar = = ':')
    Istrcat(szFileName, szFileSpec);
  if (Istrchr(szFileName, '*') ||
      Istrchr(szFileName, '?')) {
    if (DlgDirList(hDlg, szFileName,
                   IDD_FPATH,
                   wFileAttr)) {
      lstrcpy(szFileSpec, szFileName);
      SetDlgItemText(hDlg,
                     IDD_FNAME, szFileSpec);
    }
    else
      MessageBeep(0);
      return TRUE;
  }
  Istrcat(Istrcat(szFileName, "\""), szFileSpec);
  if (DlgDirList(hDlg, szFileName,
                 IDD_FPATH, wFileAttr)) {
    lstrcpy(szFileSpec, szFileName);
    SetDlgItemText(hDlg,
                   IDD_FNAME, szFileSpec);
    return TRUE;
  }
}
szFileName[nEditLen] = '\0';
if (-1 = = OpenFile(szFileName, pof,
                    OF_READ | OF_EXIST)) {
  Istrcat(szFileName, szDefExt);
  if (-1 = = OpenFile(szFileName, pof,
                      OF_READ | OF_EXIST)) {
    MessageBeep(0);
    return TRUE;
  }
}
lstrcpy(szFileName,
        AnsiNext(lstrrchr(pof-
                   > szPathName, "\"")));
EndDialog(hDlg, TRUE);
return TRUE;

case IDCANCEL: /* cancel opening a
file */
EndDialog(hDlg, FALSE);
return FALSE;

return FALSE;

/\* Process messages for save-file dialog box */
BOOL FAR PASCAL FileSaveDlgProc(HWND hDlg, WORD message, 
WORD wParam, LONG lParam)
{
    switch (message) {
        case WM_INITDIALOG: /* Initialize the dialog box */
            SendMessage(hDlg, WM_INITDIALOG, EM_LIMITTEXT, 80, 0);
            DlgDirList(hDlg, szFileSpec, 0, IDD_FPATH, 0);
            SetDlgItemText(hDlg, IDD_FNAME, szFileSpec);
            return TRUE;

        case WM_COMMAND:
            switch (wParam) {
                case IDD_FNAME:
                    /* change the file name */
                    if (HIWORD(lParam) = = EN_CHANGE)
                        GetDlgItem(hDlg, IDD_FNAME, wParam, (BOOL)
GetDlgItemText(hDlg, IDD_FNAME, szFileSpec, 80);
                    if (-1 = = OpenFile(szFileName, pof, OF_PARSE)) {
                        MessageBeep(0);
                        return TRUE;
                    }
            }
    }
    return FALSE;
}
if (-1 != OpenFile(szFileName,  
    pof,  
    OF_WRITE | OF_EXIST))
    wStatus = 1;
else if (-1 != OpenFile(szFileName, pof,  
    OF_CREATE | OF_EXIST))
    wStatus = 0;
else {
    MessageBeep(0);
    return TRUE;
}

lstrcpy(szFileName,  
    AnsiNext(lstrrchr(pof-  
    >szPathName, '\'')));

OemToAnsi(szFileName, szFileName);
EndDialog(hDlg, TRUE);
return TRUE;

/* Cancel saving a file */
EndDialog(hDlg, FALSE);
return TRUE;

/* Copy a string */
LPSTR lstrchr(LPSTR str, char ch)
{
    while (*str) {
        if (ch == *str)
            return str;
        str = AnsiNext(str);
    }
    return NULL;
}
/* Search a character in a string */
LPSTR lstrchr(LPSTR str, char ch)
{
    LPSTR str1 = str + lstrlen(str);
    do {
        if (ch == *str1)
            return str1;
        str1 = AnsiPrev(str, str1);
    } while (str1 > str);
    return NULL;
}
FILEO.c
-- Read in patterns for training 
and set up the neural network, also 
read in network weights or save network 
weights to a file
---------------------------------------------

I*
#include "header.h"

I*
Variables of Error Messages */
char     OPEN.FILE[] = "Error opening file";
char     BAD_FORMAT[] = "Bad format in data file";
char     WRITE_ERROR[] = "Cannot write to file";
char     PATTERN_ERR[] = "Error during memory allocation for
patterns";

I*
Global Variables */
float     Delta = 0.1;   /* bias added to pattern input */
static FILE *fptrl;     /* bias added to pattern input */
#define LINE 80

I*
Function to Setup Network */
int setup_network(int);

I*
Read Pattern File for training */
BOOL read_pattern(char* fileName)
{
    FILE *file_ptr;
    int lay, neu, patn;
    int temp, i;
    int from_neu, to_neu;

    /* Open pattern file to read */
    file_ptr = fopen(fileName, "rb");
    if(!file_ptr)
    {
        MessageBox(hMainWnd, OPEN_FILE,
                   szAppName, MB_ICONINFORMATION | MB_OK);
        return(FALSE);
    }

    /* Read number of layers */
    fgets(szBuffer, LINE, file_ptr);
if (sscanf(szBuffer, "number of layers: %d %d %d %d", &Layer_Num, &Num_inLayer[0], &Num_inLayer[1], &Num_inLayer[2]) == 4) {
    fgets(szBuffer, LINE, file_ptr);
    Layer_Num = 3;
    Num_inLayer[0] = 63;

    if (Num_inLayer[1] > 50)
        Num_inLayer[1] = 50;
    if (Num_inLayer[1] < 10)
        Num_inLayer[1] = 10;
}

if (sscanf(szBuffer,"number of 7x9 patterns: %d",&Pattern_Num) != 1) {
    MessageBox(hMainWnd, BAD_FORMAT,
    szAppName, MB_ICONINFORMATION | MB_OK);
    return(FALSE);
}

    /* fgets(szBuffer, LINE, file_ptr);
    if (sscanf(szBuffer,"pattern width: %d",&Patn_Width) != 1 ||
        Patn_Width != 7) {
        MessageBox(hMainWnd, BAD_FORMAT,
        szAppName, MB_ICONINFORMATION | MB_OK);
        return(FALSE);
    }
    fgets(szBuffer, LINE, file_ptr);
    if (sscanf(szBuffer,"pattern depth: %d",&Patn_Depth) != 1 ||
        Patn_Depth != 9) {
        MessageBox(hMainWnd, BAD_FORMAT,
        szAppName, MB_ICONINFORMATION | MB_OK);
        return(FALSE);
    }
    */
    Patn_Width = 7; Patn_Depth = 9;
    outWidth = 6; outDepth = 1;

    /* Allocate memory for pattern */
    PatternFreeO;
    if (PatternAlloc() == FALSE)
        return FALSE;

    /* Read all patterns */
    for(patn = 0; patn < Pattern_Num; patn++)
    {
for(i = 0; i < Patn_Width * Patn_Depth; i++)
{
    if (fscanf(file_ptr, "%d", &temp) != 1) {
        MessageBox(hMainWnd, BAD_FORMAT, szAppName, MB_ICONINFORMATION | MB_OK);
        return FALSE;
    }
    /* Add bias to pattern */
    if(temp == 1)
        Pattern[patn].in[i] = 1 - Delta;
    else
        Pattern[patn].in[i] = Delta;
}

/* Read the target output of the pattern */
for(i = 0; i < outWidth * outDepth; i++)
{
    if (fscanf(file_ptr, "%d", &temp) != 1) {
        MessageBox(hMainWnd, BAD_FORMAT, szAppName, MB_ICONINFORMATION | MB_OK);
        return FALSE;
    }
    if(temp != 0)
        Pattern[patn].out[i] = temp;
    else
        Pattern[patn].out[i] = 0;
}

fclose(file_ptr);
return TRUE;

/* Allocate memory for patterns */
BOOL PatternAlloc()
{
    /* allocate global memory */
    hPattern = GlobalAlloc(GHND | GMEM_ZEROINIT,
                           (LONG)Pattern_Num * sizeof(PATTERNS));
    if (!hPattern) {
        MessageBox(hMainWnd, PATTERN_ERR, szAppName, MB_ICONINFORMATION | MB_OK);
        return(FALSE);
Pattern = (PATTERNS FAR *) GlobalLock(hPattern);
return TRUE;

重要举措 memory used to store patterns */
void PatternFree()
{
if (Pattern)
    GlobalFree(hPattern);
Pattern = NULL;
Cur_Pattern = 0;
Total_Error = 0;

重要举措 Read in patterns for training without reset network */
BOOL read_characters(char* fileName)
{
    int temp, patn, i, j;

重要举措 Open pattern file */
    if ((fptr1 = fopen(fileName, "rb")) != NULL) {
        MessageBox(hMainWnd, OPEN_FILE, szAppName, MB_ICONINFORMATION | MB_OK);
return FALSE;
    }

重要举措 Read all patterns */
    NumOfChar = 0;
for(patn = 0; patn < 7; patn++)
    for(i = 0; i < Patn_Depth; i++)
        for(j = 0; j < Patn_Width; j++)
            if (fscanf(fptr1, "%d", &temp) != 1) {
                fclose(fptr1);
                if (patn < 5) { /* too few patterns */
                    MessageBox(hMainWnd, szAppName, MB_ICONINFORMATION | MB_OK);
return FALSE;
                }
            }
    NumOfChar = patn;
return TRUE;

OutImage[patn][i][j] = (BYTE) temp;
/* Save network weights to a file */
BOOL write_weights(char *outFileName) {
    int wret, in_neu, hid_neu, out_neu;
    float temp;

    /* Network hasn't been training */
    if (Reset) {
        MessageBox(hMainWnd, "Train first",
                    szAppName, MB_ICONINFORMATION | MB_OK);
        return FALSE;
    }

    /* Open file to save weights */
fptr1 = fopen(outFileName, "wb");
    if (fptr1 == NULL) {
        MessageBox(hMainWnd, OPEN_FILE,
                   szAppName, MB_ICONINFORMATION | MB_OK);
        return FALSE;
    }

    /* Save network architecture */
    wret = fprintf(fptr1, "number of layers: %d %d %d %d\n",
                   Layer_Num, Num_inLayer[0], Num_inLayer[1], Num_inLayer[2]);
    if (wret == EOF) {
        MessageBox(hMainWnd, WRITE_ERROR,
                   szAppName, MB_ICONINFORMATION | MB_OK);
        fclose(fptr1);
        return(FALSE);
    }

    printf(fptr1, "\n");
    hSaveCursor = SetCursor(hHourGlass);

    /* Save all the weights */
    for (in_neu = 0; in_neu < Num_inLayer[0]; in_neu++) {
        for (hid_neu = 0; hid_neu < Num_inLayer[1]; hid_neu++) {
            // Save weights here...
        }
    }
}
temp = wgtHidLn[hid_neu][in_neu].weight;
fprintf(fptr1, "%f ", temp);
}
fprintf(fptr1, "\n\n");
}
fprintf(fptr1, "\n\n");

for (hid_neu = 0; hid_neu < Num_inLayer[1]; hid_neu++) {
    for (out_neu = 0; out_neu < Num_inLayer[2]; out_neu++) {
        temp = wgtOutHid[out_neu][hid_neu].weight;
fprintf(fptr1, "%f ", temp);
    }
fprintf(fptr1, "\n\n");
}
fprintf(fptr1, "\n\n");

fclose(fptr1);
SetCursor(hSaveCursor); /* Remove the hourglass */

return TRUE;
}

/* Read in network weights */
BOOL read_weights(char *inFileName)
{
    int wret, in_neu, hid_neu, out_neu;
    int numLayer, numInLayer[3];
    float temp;

    if (!Reset)
        finishup();

    /* Open file to read */
fptr1 = fopen(inFileName, "rb");
    if (fptr1 == NULL) {
        MessageBox(hMainWnd, OPEN_FILE, szAppName, MB_ICONINFORMATION | MB_OK);
        return FALSE;
    }

    /* Read in network architecture */
wret = fscanf(fptr1, "number of layers: %d %d %d\n\n",    
                &Layer_Num, &Num_inLayer[0],    
                &Num_inLayer[1], &Num_inLayer[2]);
    if (wret != 4) {

MessageBox(hMainWnd, "error in weight file", 
    szAppName, MB_ICONINFORMATION | MB_OK);
fclose(fptr1);
return(FALSE);
}

hSaveCursor = SetCursor(hHourGlass);

/* Setup the network */
if (setup_network(FALSE) == FALSE) {
    finishup();
    SetCursor(hSaveCursor);
}

/* Read in all the weights */
for (in_neu = 0; in_neu < Num_inLayer[0]; in_neu++) {
    for (hid_neu = 0; hid_neu < Num_inLayer[1]; hid_neu++) {
        if (fscanf(fptr1, "%f", &temp) == 1)
            wgtHidIn[hid_neu][in_neu].weight = temp;
        else {
            MessageBox(hMainWnd, BAD_FORMAT, 
                szAppName, MB_ICONINFORMATION | MB_OK);
            fclose(fptr1);
            return FALSE;
        }
    }
}

for (hid_neu = 0; hid_neu < Num_inLayer[1]; hid_neu++) {
    for (out_neu = 0; out_neu < Num_inLayer[2]; out_neu++) {
        if (fscanf(fptr1, "%f", &temp) == 1)
            wgtOutHid[out_neu][hid_neu].weight = temp;
        else {
            MessageBox(hMainWnd, BAD_FORMAT, 
                szAppName, MB_ICONINFORMATION | MB_OK);
            fclose(fptr1);
            return FALSE;
        }
    }
}

Reset = FALSE;
fclose(fptr1);
SetCursor(hSaveCursor); /* Remove the hourglass */

return TRUE;

} /* Setup the neural network */
int setup_network(int init)
{
int neu, lay, from_neu, to_neu;

/* Allocate memory for all connections */
for(neu = 0; neu < Num_inLayer[1]; neu++)
{
    hWgtHidIn[neu] = GlobalAlloc(GHND | GMEM_ZEROINIT,
                              (LONG)Num_inLayer[0] * sizeof(WEIGHT));
    if(!hWgtHidIn[neu])
    {
        MessageBox(hMainWnd, "Error during allocation for
               weights", szAppName, MB_ICONINFORMATION | MB_OK);
        return(FALSE);
    }

    wgtHidIn[neu] = (WEIGHT FAR *) GlobalLock(hWgtHidIn[neu]);
}

for(neu = 0; neu < Num_inLayer[2]; neu++)
{
    hWgtOutHid[neu] = GlobalAlloc(GHND | GMEM_ZEROINIT,
                              (LONG)Num_inLayer[1] * sizeof(WEIGHT));
    if(!hWgtOutHid[neu])
    {
        MessageBox(hMainWnd, "Error during allocation for
               weights", szAppName, MB_ICONINFORMATION | MB_OK);
        return(FALSE);
    }

    wgtOutHid[neu] = (WEIGHT FAR *)
                   GlobalLock(hWgtOutHid[neu]);
}

/* Allocate memory for all neurons */
for(lay = 0; lay < Layer_Num; lay++)
{
    hNeuron[lay] = GlobalAlloc(GHND | GMEM_ZEROINIT,
                              (LONG)Num_inLayer[lay] * sizeof(NEURON));
    if(!hNeuron[lay])
    {

MessageBox(hMainWnd, "Error during memory allocation of neurons.", szAppName, MB_ICONINFORMATION | MB_OK);
return(FALSE);
neuron[lay] = (NEURON FAR *) GlobalLock(hNeuron[lay]);

/* Initialize the network weights */
if (init) {
   for(from_neu = 0; from_neu < Num_inLayer[0]; from_neu++)
      {
         for(to_neu = 0; to_neu < Num_inLayer[1]; to_neu++)
            {
              wgtHidIn[to_neu][from_neu].weight =
                  (((float)rand()/RAND_MAX) - 0.5) / 1;
              wgtHidIn[to_neu][from_neu].delta = 0;
            }
   }
   for(from_neu = 0; from_neu < Num_inLayer[1]; from_neu++)
      {
         for(to_neu = 0; to_neu < Num_inLayer[2]; to_neu++)
            {
              wgtOutHid[to_neu][from_neu].weight =
                  (((float)rand()/RAND_MAX) - 0.5) / 1;
              wgtOutHid[to_neu][from_neu].delta = 0;
            }
   }
Reset = FALSE; Trained = TRUE;
return TRUE;
}

/* Calculate default neuron number for each layer */
void calculate_layer_num()
{
   Num_inLayer[0] = Patn_Width * Patn_Depth;
   Num_inLayer[2] = outWidth * outDepth;
   Num_inLayer[1] = Num_inLayer[0] * 0.75;
}

/* Compress and save image */
void Savelmageo()
unsigned int i, j;
unsigned char  iData[512];
float  fData[512];
FILE  *fin, *fout;

/* Open original image file and output file */
fin = fopen(szWgtFileName, "rb");
if (fin == NULL) {
    MessageBox(hMainWnd, OPEN_FILE,
              szAppName, MB_ICONINFORMATION | MB_OK);
    return;
}

fout = fopen(szDatFileName, "wb");
if (fin == NULL) {
    MessageBox(hMainWnd, OPEN_FILE,
              szAppName, MB_ICONINFORMATION | MB_OK);
    fclose(fin);
    return;
}

/* compress and save image */
for (i = 0; i < 480; i++) {
    /* read 1 row each time */
    if (fread(fData, 4*512, 1, fin) != 1) {
        MessageBox(hMainWnd, "Cannot read from file",
                    szAppName, MB_ICONINFORMATION | MB_OK);
        fclose(fin);
        fclose(fout);
        return;
    }

    /* Round off the decimal point */
    for (j = 0; j < 512; j++)
        iData[j] = fData[j];

    /* write 1 row each time */
    if (fwrite(iData, 512, 1, fout) != 1) {
        MessageBox(hMainWnd, WRITE_ERROR,
                    szAppName, MB_ICONINFORMATION | MB_OK);
        fclose(fin);
        fclose(fout);
        return;
    }
}
fclose(fin);
fclose(fout);
}
I*------------------------------------------*
* Network.c -- perform all the operations *
* to update the status of the network, *
* including the weights and errors       *
*------------------------------------------*

/* Include Files */
#include "header.h"

/* message for recognition result */
char result[] = "The Pattern is: ";

/* Adjust weights from output to hidden layer */
void adjust_output_weights(void)
{
    int hid_neu, out_neu;
    WEIGHT FAR * wgtPtr;

    for(out_neu = 0; out_neu < Num_inLayer[OUTPUT_LAYER]; out_neu++)
    {
        for(hid_neu = 0; hid_neu < Num_inLayer[1]; hid_neu++)
        {
            wgtPtr = &wgtOutHid[out_neu][hid_neu];
            /* Calculate weight change */
            wgtPtr->delta = Learning_Coefficient *
                            neuron[2][out_neu].error *
                            neuron[1][hid_neu].output +
                            wgtPtr->delta * Momentum;
            /* Update this weight */
            wgtPtr->weight += wgtPtr->delta;
        }
    }
}

/* Adjust weights from hidden to input layer */
void adjust_hidden_weights(void)
{
    int hid_neu, in_neu;
    WEIGHT FAR *wgtPtr;

    for(hid_neu = 0; hid_neu < Num_inLayer[1]; hid_neu++)
    {
        for(in_neu = 0; in_neu < Num_inLayer[0]; in_neu++)
        { /*...*/
        }
    }
}
{ 
    wgtPtr = &wgtHidIn[hid_neu][in_neu];
    /* Calculate weight change */
    wgtPtr->delta = Learning_Coefficient * 
        neuron[1][hid_neu].error * 
        neuron[0][in_neu].output + wgtPtr->delta * Momentum;
    /* Update this weight */
    wgtPtr->weight += wgtPtr->delta;
}

/* Recognize patterns */
void recognition(void) {
    int patn, i, j;
    
    if (Reset) {
        MessageBox(hMainWnd, "Please load all the weights first",
            szAppName, MB_ICONINFORMATION | MB_OK);
        return;
    }
    isError = TRUE;
    
    hSaveCursor = SetCursor(hHourGlass);
    
    /* Allocate memory */
    PatternFree();
    Pattern_Num = NumOfChar;
    Patn_Width = out_width; Patn_Depth = out_height;
    outWidth = 6; outDepth = 1;
    if (PatternAlloc() == FALSE)
        return;
    
    /* Copy the patterns */
    for(patn = 0; patn < Pattern_Num; patn++)
        for(i = 0; i < Patn_Depth; i++)
            for(j = 0; j < Patn_Width; j++)
            if (OutImage[patn][i][j] == 1)
                Pattern[patn].in[i*Patn_Width+j] = 1 - Delta;
            else
                Pattern[patn].in[i*Patn_Width+j] = Delta;
/ * recognize those patterns */
for(Cur_Pattern = 0; Cur_Pattern < NumOfChar; Cur_Pattern++)
{
    float temp;

    forward_pass(); i = 0;  /* do a forward-pass */
    for (patn = 0; patn < 6; patn++)
    {
        temp = neuron[2][patn].output;  /* get network output */
        i = 2*i + temp + 0.5;  /* do a binary-combination */
    }

    if (i <= 26)  /* Convert to character */
        i += 64;
    else if (i <= 36)
        i += 21;
    else
        i = 63;
    sprintf(result+17+Cur_Pattern, "%c", i);

    /* */
    DisplayError(); */
    SetCursor(hSaveCursor);

    MessageBox(hMainWnd, result,
        "Output by Network", MB_ICONINFORMATION | MB_OK);
}

/* Perform the backward-pass */
void backward_pass(void)
{
    calc_output_error();
    calc_hidden_error();
    adjust_output_weights();
    adjust_hidden_weights();
}

/* Calculate errors of hidden neurons */
void calc_hidden_error(void)
{
    int hid_neu, out_neu;
    float err_sum, n1, n2;
    NEURON FAR *neuPtr;

    n1 = Num_inLayer[1];
for(hid_neu = 0; hid_neu < n1; hid_neu++)
{
    err_sum = 0; neuPtr = &neuron[1][hid_neu];
    n2 = num_inLayer[OUTPUT_LAYER];
    for(out_neu = 0; out_neu < n2; out_neu++)
        err_sum += neuron[2][out_neu].error *
                    wgtOutHid[out_neu][hid_neu].weight;

    /* Use derivative of transfer function */
    neuPtr->error = neuPtr->output *
                    (1 - neuPtr->output) * err_sum;
}

/* Calculate errors of output neurons */
void calc_output_error(void)
{
    int out_neu;
    float diff, this_error;
    NEURON FAR *neuPtr;

    this_error = 0;
    for(out_neu = 0; out_neu < num_inLayer[OUTPUT_LAYER];
        out_neu++)
    {
        /* calculate output error */
        diff = Pattern[Cur_Pattern].out[out_neu] -
               neuron[OUTPUT_LAYER][out_neu].output;
        if(diff < 0)
            { if(diff < -Acceptable_Error) this_error -= diff;
            }
        else { if (diff > Acceptable_Error) this_error += diff;
             }

        diff = diff * Speed_Factor; /* modified calculation */

        /* Increase output error to speed up training */
        if (CurrentError < 0.3) { diff *= 1.5;
            if (CurrentError < 0.3)
                diff *= 1.5;
        }
neuPtr = &neuron[OUTPUT_LAYER][out_neu];
neuPtr->error = diff;

} /* Update pattern error and network error */
Pattern[Cur_Pattern].error = this_error;
if(this_error > Acceptable_Error)
    Total_Error += this_error;

} /* Check if this pass is finished */
BOOL all_patterns()
{
    if(Cur_Pattern == Pattern_Num - 1){
        Cur_Pattern = 0;
        ++iteration;
        return TRUE;
    } else {
        Cur_Pattern++;
        return FALSE;
    }
}

/* Display information after the training */
void display()
{
    ScrollWindow(hMainWnd, 0, 3*cyChar, NULL, NULL);
    hdc = GetDC(hMainWnd); /* need handle to display context */
    TextOut(hdc, cxChar, cyChar, szBuffer,
            sprintf(szBuffer, "Network iteration %d Total error: %f",
                    iteration, Total_Error));
    ReleaseDC(hMainWnd, hdc);
    ValidateRect(hMainWnd, NULL);
}

/* Display error for the current pattern */
void DisplayError()
{
    int j, x, y;
    char *bptr;

    ScrollWindow(hMainWnd, 0, 3*cyChar, NULL, NULL);
    hdc = GetDC(hMainWnd);
    /* SelectObject(hdc, GetStockObject(SYSTEM_FIXED_FONT)); */
x = cxChar; y = cyChar;

TextOut(hdc, x, y, szBuffer,
    sprintf(szBuffer, "Pattern[%d] Error: %f\0", Cur_Pattern, Pattern[Cur_Pattern].error));
Istrcpy(szBuffer, "Desired: "); bptr = szBuffer + 10;

/* display all the actual outputs */
for(j = 0; j < outDepth * outWidth; j++)
    sprintf(bptr + j*11, " %6.3f ", Pattern[Cur_Pattern].out[j]);
TextOut(hdc, x, y + cyChar,
    szBuffer, strlen(szBuffer));
TextOut(hdc, x, y + 2*cyChar, "Actual: ", 9);

/* display all the desired outputs */
for(j = 0; j < outDepth * outWidth; j++) {
    float err = Pattern[Cur_Pattern].out[j] -
        neuron[OUTPUT_LAYER][j].output;
    if(fabs(err) > Acceptable_Error)
        SetTextColor(hdc, RGB(255,0,0));
    else
        SetTextColor(hdc, RGB(0,255,0));
    TextOut(hdc, x + 9*cxChar + 9*j*cxChar, y + 2*cyChar,
        szBuffer,
            sprintf(szBuffer, " %6.3f ",
                neuron[OUTPUT_LAYER][j].output));
}

SetTextColor(hdc, RGB(0,0,0));

ReleaseDC(hMainWnd, hdc);
ValidateRect(hMainWnd, NULL);
}

/* Display information for this iteration (pass) */
void displayTotalError(void)
{
    ScrollWindow(hMainWnd, 0, 2*cyChar, NULL, NULL);
    hdc = GetDC(hMainWnd);
    SelectObject(hdc, GetStockObject(SYSTEM_FIXED_FONT));

    TextOut(hdc, 4*cxChar, 2*cyChar,
        szBuffer, sprintf(szBuffer,
            "Iteration: %d Total error: %.3f", iteration, Total_Error));
ReleaseDC(hMainWnd, hdc);
ValidateRect(hMainWnd, NULL);

/* Display information for this training */
void displayPerformance(unsigned long used_time)
{
    float cps;

    sprintf(szBuffer, "Iterations: %d, Elapsed time: \
%d(hr):%d(min):%d(sec)\n", 
        iteration, (int)(used_time/3600), (int)(used_time/60),
        (int)(used_time%60));
    MessageBox(hMainWnd, szBuffer, 
        "Training", MB_ICONINFORMATION | MB_OK);
}

/* Perform a forward-pass */
void forward_pass(void)
{
    int lay, neu1, neu2, neu;
    float temp;

    /* set all inputs to zero */
    for(lay = 1; lay < Layer_Num; lay++)
    {
        for(neu = 0; neu < Num_inLayer[lay]; neu++)
            neuron[lay][neu].input = 0.0;
    }
    /* First, put the input pattern directly into the input neuron */
    for(neu = 0; neu < Num_inLayer[0]; neu++)
    {
        neuron[0][neu].input = Pattern[Cur_Pattern].in[neu];
        neuron[0][neu].output = neuron[0][neu].input;
    }
    /* Then, for input-to-hidden layer, feedforward */
    for(neu2 = 0; neu2 < Num_inLayer[1]; neu2++)
    {
        float temp;
        for(neu1 = 0; neu1 < Num_inLayer[0]; neu1++)
            neuron[1][neu2].input += neuron[0][neu1].output *
            wgtHidIn[neu2][neu1].weight;
        /* This is the "classic" transfer function. */
        temp = neuron[1][neu2].input;
        if (fabs(temp) > 30)
            temp = (temp > 0) ? 30 : -30;
neuron[1][neu2].output = 1 / (1 + exp(-1 * temp));
}

/*Then, for output layer, feedforward */
for (neu2 = 0; neu2 < Num_inLayer[2]; neu2++) {
    for (neu1 = 0; neu1 < Num_inLayer[1]; neu1++)
        neuron[2][neu2].input += neuron[1][neu1].output *
            wgtOutHid[neu2][neu1].weight;
/* This is the "classic" transfer function. */
    neuron[2][neu2].output = 1 / (1 + exp(-1 *
        neuron[2][neu2].input));
}

/* Is the training done? */
BOOL trained(void)
{
    /* Check if the network error is less than desired */
    if (Total_Error < Acceptable_Error)
        return TRUE;
    else
        return FALSE;
}
/*---------------------------------------------* 
* Neural.c -- initialize the neural network, * 
* and control the network for training * 
*---------------------------------------------*/

#include "header.h"

HANDLE hNeuron[MAX_LAYERS];
HANDLE hWgtOutHid[MAX_OUTPUTNEURONS];
HANDLE hWgtHidln [MAX_HIDDENNEURONS];
NEURON FAR* neuron[MAX_LAYERS];
WEIGHT FAR* wgtOutHid[MAX_OUTPUTNEURONS] = {NULL};
WEIGHT FAR* wgtHidln [MAX_HIDDENNEURONS] = {NULL};

HANDLE hPattern;
PATTERNS FAR* Pattern = NULL;

const int Layer_Num = 3;
int Neuron_Num;
int Num_inLayer[MAX_LAYERS] = {0};
int Pattern_Num;
int Cur_Pattern;
int Patn_Width, Patn_Depth, outWidth, outDepth;
int iteration;
float CurrentError = 1e8;

void network_init(void);
void calculate_layer_num(void);
int setup_network(int);
void finishup(void);
int backpropagation;

int train_network(int init)
{
    time_t start_time, end_time;
    unsigned long used_time, ret;

    /* Initialize the network */
    if (init) 
    
    /* Control training the network */

if (init == 1) {
    network_init();
    calculate_layer_num();
}

if(read_pattern(szPtnFileName) == FALSE) /* Read patterns */
{
    MessageBox(hMainWnd, "Error in initializing net",
               szAppName, MB_ICONINFORMATION | MB_OK);
    finishup();
    return(0);
}

if (init == 1) {
    if (setup_network(TRUE) == FALSE) {
        /* Setup the network */
        finishup();
        return(0);
    }
}
srand(37);

time(&start_time); /* Record the start time */

ret = backpropagation(); /* backpropagation training */
if (ret) {
    time(&end_time);
    display();
    used_time = (unsigned) end_time - (unsigned) start_time;
    displayPerformance(used_time); /* Display training time */
}

return ret;

/* Perform the backpropagation training */
int backpropagation(void)
{
    MSG msg;

    for(;;){
        /* check if user want to stop training */
        while (PeekMessage(&msg, hMainWnd, 0, 0,
                            PM_REMOVE | PM_NOYIELD)) {
            if (msg.message == WM_NCLBUTTONDOWN)
                return FALSE;
else if (msg.message == WM_PAINT)
    ValidateRect(hMainWnd, NULL);

if (MultiTasking)          /* handle multitasking */
    PeekMessage(&msg, NULL, 0, 0, PM_NOREMOVE);

forward_pass();           /* forward pass */
backward_pass();          /* backward pass */

if (isDisplayError)
    DisplayError();
if(all_patterns())         /* next pass */
{
    if(trained())
        return TRUE;
    displayTotalError();
    CurrentError = Total_Error;
    Total_Error = 0;
}

/* Initialize the neural network */
void network_init()
{
    Neuron_Num = 0;
    Pattern_Num = 0;
    Cur_Pattern = 0;
    Patn_Width = 7;
    Patn_Depth = 9;
    outWidth = 6;
    outDepth = 1;
    iteration = 0;
    Total_Error = 0;
}

/* Reset the neural network */
void finishup()
{
    int lay, neu;

    if (!Reset) {
        for(lay = 0; lay < Layer_Num; lay++)
            GlobalFree (hNeuron[lay]);
for (neu = 0; neu < Num_inLayer[1]; neu++)
    GlobalFree(hWgtHidIn[neu]);
for (neu = 0; neu < Num_inLayer[2]; neu++)
    GlobalFree(hWgtOutHid[neu]);

PatternFree();
}
Trained = FALSE; Reset = TRUE;
}

/* Activate the program in multitasking */
void EnableSystem()
{
    WORD i;

    /* request all the required memory */
    if (!Reset) {
        for (i = 0; i < Layer_Num; i++)
            neuron[i] = (NEURON far*)GlobalLock(hNeuron[i]);
        for (i = 0; i < Num_inLayer[1]; i++)
            wgtHidIn[i] = (WEIGHT far*)GlobalLock(hWgtHidIn[i]);
        for (i = 0; i < Num_inLayer[2]; i++)
            wgtOutHid[i] = (WEIGHT far*)GlobalLock(hWgtOutHid[i]);
    }

    if (Pattern)
        Pattern = (PATTERNS FAR *) GlobalLock(hPattern);

    if (hasBuffer) {
        for (i = 0; i < max_row; i++)
            Image[i] = GlobalLock(hImage[i]);
        for (i = 0; i < max_row; i++)
            Buffer[i] = GlobalLock(hBuffer[i]);
    }
}

/* deactivate the program in multitasking */
void DisableSystem()
{
    WORD i;

    /* release all the used memory */
    if (!Reset) {
        for (i = 0; i < Layer_Num; i++)
GlobalUnlock(hNeuron[i]);
for (i = 0; i < Num_inLayer[2]; i++)
    GlobalUnlock(hWgtOutHid[i]);
for (i = 0; i < Num_inLayer[1]; i++)
    GlobalUnlock(hWgtHidIn[i]);
}

if (Pattern)
    GlobalUnlock(hPattern);

if (hasBuffer) {
    for (i = 0; i < max_row; i++)
        GlobalUnlock(hImage[i]);
    for (i = 0; i < max_row; i++)
        GlobalUnlock(hBuffer[i]);
}

/*---------------------------------------*
* Pr0cess.c -- Image Pre-Processing Functions *
* This program includes all the functions *
* for image pre-processing. *
*---------------------------------------*/

/* Include file */
#include "header.h"

/* Convoluation kernels for vertical edge enhancement */
short verKernel[9] = {0, 0, 0, -1, 1, 0, 0, 0, 0};

/* Arrays for character arrays */
BYTE Outlmage[7][out_height][out_width];
long max_freq;
long frequency[512];

/* Structures to record block information */
typedef struct {
    WORD start, end, length;
} BLOCK;
BLOCK RowBlock[20];

/* Structures to record character information */
typedef struct {
    WORD start_col, end_col, start_row, end_row, width, height, flag;
} CHARACTER;
CHARACTER all_chars[8];

WORD WhiteOnBlack; /* is white-on-black */
WORD start_row, end_row; /* starting and ending row No.s */
WORD NumOfChar; /* number of characters */
WORD DataReady = 0; /* status of preprocessing */
WORD DisplayOutput = TRUE; /* display output in preprocessing */
/* Global Variables */
WORD index = 0, height, focus;
DWORD total, threshold;
extern WORD col, row, color, pix; /* pixel coordinates x, y */

/* Function declarations */
int Convolution(BYTE far *InImage[480], unsigned Col, unsigned Row,
               unsigned Width, unsigned Height,
short *Kernel, unsigned KernelCols,
unsigned KernelRows, unsigned Absolute,
BYTE far *OutImageBuffer[480]);

void ScaleImage(BYTE far *InImage[480], WORD SCol, WORD SRow,
WORD SWidth, WORD SHeight,
BYTE OutImage[out_height][out_width],
WORD DestWidth, WORD DestHeight, int flag);

/* Perform pre-processing in a batch */
void preProcessing(void)
{
    InvalidateRect(hMainWnd, NULL, TRUE);  /* Clear main window */
    UpdateWindow(hMainWnd);

    VerticalEdgeEnhance();
    HorizontalScan();
    Imageldentification();
    if (VerticalScan())
        ScaleCharacter();
    else
        MessageBox(hMainWnd, "The System cannot preprocess this picture",
                   "Preprocessing", MB_ICONINFORMATION | MB_OK);
}

/* Perform vertical edge enhancement */
void VerticalEdgeEnhance()
{
    if (DataReady != 1)
        return;
    hSaveCursor = SetCursor(hHourGlass);
    Convolution(Image, 1, 1, 510, 478,
                verKernel, 3, 3, 1, Buffer);

    threshold = 20;
    for(row = 0; row < max_row; row++) {  /* Thresholding */
        for (col = 0; col < max_col; col++) {
            pix = (Buffer[row][col] > =threshold) ? 1 : 0;
            Buffer[row][col] = pix;
        }
    }

    /* smoothing the data */
/**
   * for(row = 1; row < max_row - 1; row++) {
   *   Buffer[row][0] = Buffer[row][max_col - 1] = 0;
   *   for (col = 1; col < max_col - 1; col++) {
   *     Buffer[row][col] = (Buffer[row][col - 1] +
   *                       Buffer[row - 1][col - 1] + Buffer[row - 1][col + 1] +
   *                       Buffer[row + 1][col] + Buffer[row + 1][col + 1] +
   *                       Buffer[row - 1][col + 1] + Buffer[row + 1][col + 1] +
   *                       Buffer[row][col] + Buffer[row][col + 1]) > 9 ? 1
   *       : 0;
   *   }
   */
   * SetCursor(hSaveCursor);
   * DataReady = 2;
   * }
   *
   /* Perform horizontal scan */
   void HorizontalScan()
   {
     BYTE RowVal[480], RowStatus;
     DWORD RowSum[480];
     int i;

     if (DataReady != 2)
       return;
     hSaveCursor = SetCursor(hHourGlass);
     hdc = GetDC(hMainWnd);
     total = 0;

     /* Display image after edge enhancement */
     for (row = 0; row < max_row; row++) {
       RowSum[row] = 0;
       for (col = 0; col < max_col; col++) {
         pix = Buffer[row][col];
         RowSum[row] += pix;
         total += pix;
         if (DisplayOutput) {
           color = pix ? 50 : 200;
           SetPixel(hdc, col, row, RGB(color, color, color));
         }
       }
     }
   }
threshold = total / max_row;
/* Group all rows into several row-blocks */
for (row = 0; row < max_row; row++) {
    if (RowSum[row] > threshold)
        col = 200;
    else
        col = 100;
    RowVal[row] = col;
/*
SetPixel(hdc, col, row, 0x00FF0000); */
}

/* Identify all the signal row-blocks */
RowStatus = 100; index = 0;
for (row = 0; row < max_row; row++) {
    if (RowVal[row] != RowStatus)
        { if (RowStatus == 100)
            RowBlock[index].start = row;
        else {
            RowBlock[index].end = row;
            index ++;
        }
    RowStatus = RowVal[row];
}

if (index == 0)
    return;

height = 0; color = index;
/* Find the the maximum row-block */
for (row = 0; row < index; row++) {
    RowStatus = RowBlock[row].end - RowBlock[row].start;
    if (height < RowStatus) {
        height = RowStatus;
        i = row;
    }
}

/* Display the maximum row-bolck */
start_row = RowBlock[i].start;
end_row = RowBlock[i].end + 1;
/*
TextOut(hdc, 20, start_row, szBuffer,
    sprintf(szBuffer, "Start:%d", start_row));
TextOut(hdc, 20, end_row, szBuffer,
*/
sprintf(szBuffer, "End :%d", end_row));
for (row = RowBlock[i].start; row < RowBlock[i].end; row++)
    SetPixel(hdc, 300, row, 0x000000FF);
index = i;

DataReady = 3;
ReleaseDC(hMainWnd, hdc);
SetCursor(hSaveCursor);

/* Compute statistical information for each column */
void col_statistic()
{
    int row, col;
    BYTE level;

    for(col = 0; col < max_col; col++) {
        frequency[col] = 0;
        for (row = start_row; row < end_row; row++) {
            level = Image[row][col];
            frequency[col] += level;
        }
    }
}

/* Identify and convert image */
void ImageIdentification()
{
    int start, end, lines;

    if (DataReady != 3)
        return;
    hSaveCursor = SetCursor(hHourGlass);
    /* Calculate average gray levels of background */
    threshold = 0; end = start_row;
    if (index)
        start = RowBlock[index-1].end;
    else
        start = start_row - 5;
    if (index > 1 && (end-start) < 5) {
        start = RowBlock[index-2].end;
        end = RowBlock[index-1].start;
    }
    lines = end - start;
    for(row = start; row < end; row++)
for (col = 0; col < max_col; col++)
    threshold += Image[row][col];
start = end_row;
if (index != color)
    end = RowBlock[index + 1].start;
else
    end = end_row + 5;

for(row = start; row < RowBlock[index + 1].start; row++)
    for (col = 0; col < max_col; col++)
        threshold += Image[row][col];
threshold /= (lines + RowBlock[index + 1].start - end_row);

/* Calculate average gray levels of signal */
total = 0;
for(row = start_row; row < end_row; row++)
    for (col = 0; col < max_col; col++)
        total += Image[row][col];
total /= (end_row - start_row);

col_statistic();
max_freq = frequency[0];
if (total < threshold) {
    WhiteOnBlack = 0;

    /* Convert image to binary */
    for (col = 0; col < max_col; col++)
        if (max_freq > frequency[col])
            max_freq = frequency[col];
    focus = max_freq / (end_row - start_row);
    for(row = start_row - 3; row < end_row + 3; row++)
        for (col = 0; col < max_col; col++)
            { pix = Image[row][col];
              if (pix <= focus + 20)
                  pix = 1;
              else
                  pix = 0;
              Buffer[row][col] = (BYTE) pix;
            } }
} else {
    WhiteOnBlack = 1;

    /* Convert image to binary */
    for (col = 0; col < max_col; col++)
if (max_freq < frequency[col])
    max_freq = frequency[col];
focus = max_freq / (end_row - start_row);
if (focus < 60) focus = 60;
for(row=start_row-3; row<end_row+3; row++) {
    for (col=0; col<max_col; col++) {
        pix = Image[row][col];
        if (pix >= focus-40)
            pix = 1;
        else
            pix = 0;
        Buffer[row][col] = (BYTE) pix;
    }
}
}

SetCursor(hSaveCursor);
DataReady = 4;
}

/* perform vertical scan */
BOOL VerticalScan()
{
    BYTE cind;
    WORD ColVal[512], ColStatus, wide, i;
    BLOCK ColBlock[20];

    if (DataReady != 4)
        return FALSE;
    hSaveCursor = SetCursor(hHourGlass);
    hdc = GetDC(hMainWnd);
    /***************************************************************/
    /*** Scan Along the Coloum ***/
    /***************************************************************************/
    InvalidateRect(hMainWnd, NULL, TRUE);
    UpdateWindow(hMainWnd);

    /* smoothing the data */
    for (row=start_row; row<end_row; row++) {
        Buffer[row][0] = Buffer[row][max_col-1] = 0;
        for (col=1; col<max_col-1; col++) {
            Buffer[row][col] = ((WORD)Buffer[row][col-1] +
                Buffer[row-1][col-1] + Buffer[row-1][col+1] +
                Buffer[row-1][col] + Buffer[row+1][col] +
            ) / 5;
        }
    }
/* display the row-block */
if (DisplayOutput)
{
    for(row = start-row; row < end-row; row + +)
    {
        for(col = 0; col < max-col; col + +)
        {
            color = Buffer[row][col] ? 200 : 50;
            SetPixel(hdc, col, row, RGB(color, color, color));
        }
    }
}

/* Sum gray levels over each column */
for (col = 0; col < max_col; col + +)
{
    ColVal[col] = 0;
    for(row = start_row; row < end_row; row + +)
    {
        pix = Buffer[row][col];
        ColVal[col] += pix;
    }
}

/* Group all columns into background and signal blocks */
for (col = 0; col < max_col; col + +)
{
    if (ColVal[col] > 1)
        row = end_row;
    else
        row = start_row;
    ColVal[col] = row;
    if (DisplayOutput)
        SetPixel(hdc, col, row, RGB(200,0,0));
}

/* Identify all the column-blocks */
ColStatus = start_row;
index = 0;
for (col = 0; col < max_col; col + +)
{
    if (ColVal[col] != ColStatus) /* There's a change here */
    {
        if (ColStatus == start_row)
ColBlock[index].start = col;
else {
    ColBlock[index].end = col;  // Start a new block */
    ColBlock[index].length = col -
ColBlock[index].start;
    if (ColBlock[index].length < 100 &&
        ColBlock[index].length > 4)
        index++;}
    ColStatus = ColVal[col];
}

/* Image has less than 5 characters */
if (index < 5)
    return FALSE;

wide = 0; pix = 0;
for (i=0; i<index; i++) {
    if ((ColBlock[i].start < 10) ||
        (ColBlock[i].end > max_col-10)) { /* background */
        ColBlock[i].length = 0;
pix + + ;
    } else
        wide + = ColBlock[i].length;
}
wide /= (index - pix);
cind = 0;

/* eliminate not-letter objects */
for (i=0; i<index; i++) {
    if (ColBlock[i].length > = 3*wide)    /* too wide */
        ColBlock[i].length = 0;
    else if (3*ColBlock[i].length <= wide) /* Too narrow */
        ColBlock[i].length = 0;
    else if (5*ColBlock[i].length <= 4*wide) {
        threshold = 0;

        /* Check if this is the plate edge */
        for(row=start_row-3; row < end_row+3; row + + ){
            pix = 0;
            for (col=ColBlock[i].start; col<ColBlock[i].end;
                col + + )
                pix + = Buffer[row][col];
if (pix > 1) threshold ++;
if (row == start_row - 1)
    row = end_row - 1;
}
if (threshold > 3)                /* Extend outside */
    ColBlock[i].length = 0;
}

/* Check if there is the background around the plate */
if (ColBlock[i].length)
{
    total = 0; threshold = 0;
    for (row = start_row; row < end_row; row++) {
        pix = 0;
        for (col = ColBlock[i].start; col < ColBlock[i].end;
            col++)
            pix += Buffer[row][col];
        total += pix;
        if (pix > 1) threshold ++;
        ColVal[row] = pix;
    }
    /* 90% of the pixels are signal: wrong block */
    if ((total > 0.9 * (end_row - start_row) * 
        (ColBlock[i].end - ColBlock[i].start)) ||
        (threshold < 0.7 * (end_row - start_row)))
        ColBlock[i].length = 0;
}

/* Now we have all the characters */
if (ColBlock[i].length)
{
    row = start_row;
    while (ColVal[row] <= 1) row ++;    /* eliminate top blank */

    all_chars[cind].start_row = row;
    row = end_row - 1;
    while (ColVal[row] <= 1) row--;      /* eliminate bottom blank */

    all_chars[cind].end_row = row + 1;
    all_chars[cind].start_col = ColBlock[i].start;
    all_chars[cind].end_col = ColBlock[i].end;
    all_chars[cind].width = all_chars[cind].end_col - 
        all_chars[cind].start_col;
    all_chars[cind].height = all_chars[cind].end_row - 
        all_chars[cind].start_row;
all_chars[cind].start_row;
    cind += 1;
}
}
index = cind;
if (index < 5) /* less than 5 blocks left? */
    return FALSE;

/* Check which character's width less than of 2/3 average */
/* Is it a 1 with small width */
wide = 0;
for (col = 0; col < index; col++)
    if (wide < all_chars[col].width)
        wide = all_chars[col].width;
for (col = 0; col < index; col++)
    if (2 * wide >= 3 * all_chars[col].width)
        all_chars[col].flag = 1;
    else
        all_chars[col].flag = 0;

if (WhiteOnBlack && focus >= 80) {
    /* White-on-black converted to binary */
    for (i = 0; i < index; i++)
        for (row = all_chars[i].start_row;
            row < all_chars[i].end_row; row++)
            for (col = all_chars[i].start_col;
                col < all_chars[i].end_col; col++)
                pix = Image[row][col];
        if (pix >= focus - 60)
            pix = 1;
        else
            pix = 0;
        Buffer[row][col] = (BYTE) pix;
}
InvalidateRect(hMainWnd, NULL, TRUE);
UpdateWindow(hMainWnd);

/* Display all the blocks */
if (DisplayOutput)
    for (i = 0; i < index; i++)
for (row = all_chars[i].start_row; row < all_chars[i].end_row; row++)
    for (col = all_chars[i].start_col; col < all_chars[i].end_col; col++) {
        color = Buffer[row][col] ? 220 : 30;
        SetPixel(hdc, col + 20*i, row, RGB(color, color, color));
    }
TextOut(hdc, all_chars[i].start_col + 20*(i-1), /* display height */
        all_chars[i].start_row + 20,
        szBuffer, sprintf(szBuffer, "%d", all_chars[i].height));
TextOut(hdc, all_chars[i].start_col + 20*i, /* display width */
        all_chars[i].end_row + 20,
        szBuffer, sprintf(szBuffer, "%d", all_chars[i].width));
}

ReleaseDC(hMainWnd, hdc);
SetCursor(hSaveCursor);
DataReady = 5;
return TRUE;
}

/* Scale the characters into 7x9 */
void ScaleCharacter()
{
    int i;

    if (DataReady != 5)
        return;
    NumOfChar = index;
hSaveCursor = SetCursor(hHourGlass);

    for (i = 0; i < NumOfChar; i++)
        ScaleImage(Buffer, all_chars[i].start_col, all_chars[i].start_row,
            all_chars[i].width, all_chars[i].height,
            OutImage[i], out_width, out_height,
            all_chars[i].flag);
    SetCursor(hSaveCursor);
    DataReady = 6;
}
/* Save the characters */
void OutputCharacter()
{
    FILE *outFile;
    int point, i;

    if (DataReady != 6)
        return;

    if (!DoFileSaveDlg(hMainInstance, hMainWnd, szOutFileSpec,
                       szOutFileSpec + 1, &wStatus, szOutFileName, &of))
        return;

    /* Open the output file. Quit program if not found. */
    outFile = fopen(szOutFileName, "wb");
    if (outFile == NULL) {
        MessageBox(hMainWnd, "File open error",
                   szAppName, MB_ICONINFORMATION | MB_OK);
        return;
    }

    /* Save the characters */
    hSaveCursor = SetCursor(hHourGlass);
    for (i = 0; i < index; i++) {
        for (row = 0; row < out_height; row++) {
            for (col = 0; col < out_width; col++) {
                point = (int) OutImage[i][row][col];
                fprintf(outFile, "%d ", point);
            }
            fprintf(outFile, "\n\n");
        }
    }

    SetCursor(hSaveCursor);
    fclose(outFile);
    return;
}

/* Integer Convolution Function */
int Convolution(BYTE far *InImage[480], unsigned Col, unsigned Row,
                unsigned Width, unsigned Height,
short *Kernel, unsigned KernelCols,
unsigned KernelRows, unsigned Absolute,
BYTE far *OutImageBuffer[480])
{
    unsigned ColExtent, RowExtent;
    unsigned ImageCol, ImageRow, KernCol, KernRow;
    unsigned ColOffset, RowOffset, TempCol, TempRow;
    long Sum;
    short * KernelPtr;

    /* Image must be at least the same size as the kernel */
    if (Width >= KernelCols && Height >= KernelRows) {
        /* Clear the output buffer */
        for (ImageRow = 0; ImageRow < 480; ImageRow++)
            fmemset(OutImageBuffer[ImageRow], 0, 512);

        ColOffset = KernelCols / 2;
        RowOffset = KernelRows / 2;

        /* Compensate for edge effects */
        Col += ColOffset;
        Row += RowOffset;
        Width -= (KernelCols - 1);
        Height -= (KernelRows - 1);

        /* Calculate new range of pixels to act upon */
        ColExtent = Col + Width;
        RowExtent = Row + Height;

        for (ImageRow = Row; ImageRow < RowExtent; ImageRow++)
        {
            TempRow = ImageRow - RowOffset;
            for (ImageCol = Col; ImageCol < ColExtent; ImageCol++)
            {
                TempCol = ImageCol - ColOffset;
                Sum = 0L;
                KernelPtr = Kernel;

                /* Good format, but less efficient */
                for (KernCol = 0; KernCol < KernelCols;
                     KernCol++)
                    for (KernRow = 0; KernRow < KernelRows;
                         KernRow++)
                        Sum += InImage[(TempRow + KernRow)][TempCol + KernCol] * (*KernelPtr ++);
            }
        }
    }
}
/* Calculate the weighted sum, faster than above */

Sum = 
InImage[(TempRow + 0)][TempCol + 0]*(KernelPtr[0]) + 
InImage[(TempRow + 0)][TempCol + 1]*(KernelPtr[1]) + 
InImage[(TempRow + 0)][TempCol + 2]*(KernelPtr[2]) + 
InImage[(TempRow + 1)][TempCol + 0]*(KernelPtr[3]) + 
InImage[(TempRow + 1)][TempCol + 1]*(KernelPtr[4]) + 
InImage[(TempRow + 1)][TempCol + 2]*(KernelPtr[5]) + 
InImage[(TempRow + 2)][TempCol + 0]*(KernelPtr[6]) + 
InImage[(TempRow + 2)][TempCol + 1]*(KernelPtr[7]) + 
InImage[(TempRow + 2)][TempCol + 2]*(KernelPtr[8]);

/ * If absolute value is requested */
if (Absolute) Sum = labs(Sum);

/ * Summation performed. Scale and range Sum */
Sum >= = (long) Scale;

Sum = (Sum < 0) ? 0 : Sum;
Sum = (Sum > 255) ? 255 : Sum;

OutImageBuffer[ImageRow][ImageCol] = (BYTE) Sum;

} else
return 0;

} /* Scale the input image to fit the output image size */
void ScaleImage(BYTE far *InImage[480], WORD SCol, WORD SRow,
WORD SWidth, WORD SHeight,
BYTE OutImage[out_height][out_width],
WORD DestWidth, WORD DestHeight, int flag)
{ 
    WORD PtA, PtB, PtC, PtD, PixelValue, Il = 0;
    WORD SFromColNum, SFromRowNum, SToColNum, SToRowNum;
    WORD SrcCol, SrcRow, DestCol, DestRow;
    float ScaleH, ScaleV;
    float SFromColAddr, SFromRowAddr, SToColAddr, SToRowAddr;
    float FromColDelta, FromRowDelta;
    float ContribFromAandB, ContribFromCandD;

    /* 5x9 scaling for 1 with small width */
    /* with blank in both sides to get 7x9 */
    if (flag) {
        for (DestRow = 0; DestRow < DestHeight; DestRow++)
            for (DestCol = 0; DestCol < DestWidth; DestCol++)
                OutImage[DestRow][DestCol] = 0;
        DestWidth = 5; Il = 1;
    }

    /* Calculate scaling factors */
    ScaleH = (float) DestWidth / SWidth;
    ScaleV = (float) DestHeight / SHeight;

    /* Calculations from output (7x9) perspective */
    /* go through each row 1 to 9 */
    for (DestRow = 0; DestRow < DestHeight; DestRow++) {
        SFromRowAddr = DestRow / ScaleV + SRow;
        SFromRowNum = (WORD) SFromRowAddr;
        FromRowDelta = SFromRowAddr - SFromRowNum;
        if (FromRowDelta) SFromRowNum++;
        SToRowAddr = (DestRow + Il) / ScaleV + SRow;

        /* Go through each column 1 to 7 */
        for (DestCol = 0; DestCol < DestWidth; DestCol++) {
            SFromColAddr = DestCol / ScaleH + SCol;
            SFromColNum = (WORD) SFromColAddr;
            FromColDelta = SFromColAddr - SFromColNum;
            if (FromColDelta) SFromColNum++;
            SToColAddr = (DestCol + 1) / ScaleH + SCol;

            PixelValue = 0;
            /* Sum of gray levels of this area */
            for (SrcRow = SFromRowNum; SrcRow < SToRowAddr;
                SrcRow++)
                ContribFromAandB, ContribFromCandD;
        }
    }
}
for (SrcCol = SFromColNum; SrcCol < SToColAddr;

    SrcCol++)

    PixelValue += InImage[SrcRow][SrcCol];

    /* Does this area have at least half signal pixels? */
    if (2*PixelValue >= (SrcRow-SFromRowNum)*(SrcCol-

        SFromColNum))
        PixelValue = 1;
    else
        PixelValue = 0;

    /* Put the pixel into the destination buffer */
    OutImage[DestRow][DestCol+1] = PixelValue;
/* Scanning.c -- Load image into memory, and display the image  
*---------------------------------------------------------------------*/

/* Include File */
#include "header.h"

/* Function declarations */
BOOL MemAllocateO;
BOOL TempMemAllocateO;

/* global variables */
WORD col, row, color, pix, hasBuffer = FALSE;

const WORD max_row = 480;  /* max row number in image, default 480 */
const WORD max_col = 512;  /* max row number in image, default 512 */

/* Arrays to store image */
HANDLE hImage[480] = {0};
BYTE far *Image[480] = {NULL};
HANDLE hBuffer[480] = {0};
BYTE far *Buffer[480] = {NULL};

/* Load an image into memory */
BOOL LoadDataO()
{  
    FILE  *inFile;
    BYTE  scanline[512];
    static BOOL Readed = FALSE;
    int maxLevel = 0;

    /* Open the input file. Quit program if not found. */
inFile = fopen(szDatFileName, "rb");
    if (inFile == NULL) {
        MessageBox(hMainWnd, "File open error", szAppName, MB_ICONINFORMATION |
MB_OK);
        return (FALSE);
    }

    if (Readed == FALSE)
if (MemAllocate() == FALSE) /* Allocate memory for 1st time */
    return FALSE;

hSaveCursor = SetCursor(hHourGlass);

Readed = FALSE; DataReady = 0;
/* Loop to read all data */
for (row = 0; row < max_row; row++) {
    if (fread(scanline, 512, 1, inFile) != 1) { /* read 1 row each time */
        MessageBox(hMainWnd, "Can not read data",
                    szAppName, MB_ICONERROR | MB_OK);
        fclose(inFile);
        SetCursor(hSaveCursor);
        return FALSE;
    }
    _fmemcpy(Image[row], scanline, 512); /* save this row */
    _fmemcpy(Buffer[row], Image[row], 512);

    /* Loop to find the maximum gray level */
    for (col = 0; col < max_col; col++) {
        if (maxLevel < scanline[col])
            maxLevel = scanline[col];
    }
}

/* expand if the maximum gray level is too small */
if (maxLevel < 70) {
    for (row = 0; row < max_row; row++) {
        for (col = 0; col < max_col; col++)
            Image[row][col] = 70 / maxLevel;
    _fmemcpy(Buffer[row], Image[row], 512);
}
}

SetCursor(hSaveCursor); /* Remove the hourglass */
fclose(inFile);
Readed = TRUE; DataReady = 1;
return TRUE;

/* Display the image */
void ViewData(HWND hDlg)
{ int maxLevel = 0;

hdc = GetDC(hDlg);

hSaveCursor = SetCursor(hHourGlass);

/* Loop to find the maximum gray level */
for (row = 0; row < max_row; row++)
    for (col = 0; col < max_col; col++) {
        if (maxLevel < Buffer[row][col])
            maxLevel = Buffer[row][col];
    }

if (maxLevel == 0)
    return;

/* Loop to display data */
for (row = 0; row < max_row; row++)
    for (col = 0; col < max_col; col++) {
        color = (Buffer[row][col] * 205L) / maxLevel + 50;
        SetPixel(hdc, col, row, RGB(color, color, color));
    }

SetCursor(hSaveCursor); /* Remove the hourglass */
ReleaseDC(hDlg, hdc);

} /* Allocate memory to store image */
BOOL MemAllocate() {
    WORD i, j;

    for (i = 0; i < max_row; i++) {
        hImage[i] = GlobalAlloc(GHND, max_col); /* Allocate global memory */
        hBuffer[i] = GlobalAlloc(GHND, max_col);
        if (hImage[i] == NULL || hBuffer[i] == NULL) {
            MessageBox(hMainWnd, "Error during memory allocation for data",
                        szAppName, MB_ICONINFORMATION | MB_OK);
            for (j = 0; j < i; j++) {
                GlobalFree(hImage[j]);
                GlobalFree(hBuffer[j]);
            }
        }
    }

    for (i = 0; i < max_row; i++)
        GlobalFree(hImage[i]);
    for (i = 0; i < max_row; i++)
        GlobalFree(hBuffer[i]);
    return;
}
return (FALSE);

Image[i] = GlobalLock(hImage[i]);
Buffer[i] = GlobalLock(hBuffer[i]);

return (hasBuffer = TRUE);

/* Free memory used to store image */

void ClearBuffer()
{
    WORD i;

    /* clean up */
    if (hasBuffer) {
        for (i = 0; i < max_row; i++)
            GlobalFree(hImage[i]);
        for (i = 0; i < max_row; i++)
            GlobalFree(hBuffer[i]);
    }
}

/* Display the image */

BOOL FAR PASCAL ViewImageDlgProc(HWND hDlg, WORD message, WORD wParam, LONG lParam)
{
    int x, y, pixel;

    switch (message) {
        case WM_INITDIALOG:
            return TRUE;

        case WM_PAINT: /* Display the image */
            ViewData(hDlg);
            return FALSE;

        case WM_LBUTTONDOWN:
            x = LOWORD(lParam); /* Display the gray level and */
            y = HIWORD(lParam); /* its coordinates of the */
            pixel */
            if (x < 512 && y < 480) { /* which mouse points to */

SetDlgItemText(hDlg, 991, gcvt(y, 4, szBuffer));
SetDlgItemText(hDlg, 992, gcvt(x, 4, szBuffer));
pixel = Buffer[y][x];
SetDlgItemText(hDlg, 993, gcvt(pixel, 4, szBuffer));
}
return FALSE;

case WM_COMMAND:
    switch (wParam) {
        case IDOK: /* End the dialog box */
            break;
        case IDCANCEL:
            EndDialog(hDlg, 0);
            return TRUE;
    }
    break;

return FALSE;