(52)

# A Study of Improvements for the Winbank Courtesy Amount Recognition System

by Kelvin L. Cheung

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degrees of
Bachelor of Science in Computer Science
and Master of Engineering in Electrical Engineering and Computer Science
at the Massachusetts Institute of Technology

May 29, 1998 [June 1998]



Author_____
Department of Electrical Engineering and Computer Science
May 29, 1998

Certified by_____
Dr. Amar Gupta
Thesis Advisor

Accepted by_____
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

A Study of Improvements for the Winbank
Courtesy Amount Recognition System
by
Kelvin L. Cheung
Submitted to the
Department of Electrical Engineering and Computer Science
May 29, 1998
In Partial Fulfillment of the Requirements for the Degree of
Bachelor of Science in Computer Science
and Master of Engineering in Electrical Engineering and Computer Science

## ABSTRACT

Four separate improvements to the Winbank Courtesy Amount Recognition System are presented in this paper. The first alteration is a grammar engine to allow for improved recognition and handling of punctuation. An adjustment to the pre-processing steps is also presented which deals with the problem of non-uniformity in character size. A system for feedback between the segmentation and recognition algorithms is introduced which allows the system to handle splitting and merging of unrecognized characters. Finally, solutions to two specific problematic character recognitions are presented. Tests with experimental data from bank-check images show that these alterations improve the recognition rates of the system.

Thesis Supervisor: Amar Gupta

Title: Senior Research Scientist, MIT Sloan School of Management

# TABLE OF CONTENTS

# TABLE OF FIGURES

# Chapter 1

# Introduction

Optical character recognition (OCR) is the process of analyzing images of either handwritten or typed strings of characters and numerals in order to determine their contents. Character recognition comprises a significant subfield of pattern recognition, which has achieved impressive results on machine-printed text. Work on handwritten characters, however, has proven to be far more challenging due to non-uniformity of handwriting and connected characters.

OCR is performed in two environments on-line and off-line. On-line recognition involves handwritten writing done at the time of the recognition. For instance the OCR implemented by personal digital assistants is on-line. Off-line recognition is performed on pre-written material. Thus, on-line recognition is a simpler problem since dynamic information, such as stroke sequence, speed, pressure, and pen-up and pen-down position, is available to help deal with some of the problems arising from non-uniformity. On the other hand, off-line recognition deals only with the static information from the image of the writing itself, which requires completely different techniques than on-line recognition [1]. The form of character recognition studied in this paper is that of off-line numerical character recognition.

Specifically, this work is part of a larger, on-going software project here at MIT, whose purpose is to automate recognition of the monetary amount written in the courtesy

amount block on bank checks. This paper describes research undertaken to improve the character recognition algorithms in the package. The rest of this chapter contains a background of character recognition, a description of recent advances in the field, and relevant information necessary to understand the topics broached in this research.

Chapter Two describes the Winbank package and describes the level of performance it exhibited prior to any changes made during the research.

Chapter Three details the first area of the engine that was altered, the grammar engine.

Chapter Four explains the new need for a new pre-processing step in the algorithm, how it was implemented, and the results.

Chapter Five describes the new algorithm added to the engine allowing feedback between the segmentation and recognition portions of the recognition.

Chapter Six details the final change to Winbank, which involved adjustments made in the recognition algorithms for specific problem characters.

Chapter Seven describes the results of the changes made to the Winbank program.

## 1.1 Off-Line Character Recognition

While the work done for this paper focuses on a specific application in the field of character recognition, it is important to consider a broader background of the field in order to gain context on the problem.

### 1.1.1 Applications

While technology is tending to move all forms of communication in our society away

from paper to electronic media, the transition is far from complete, and there is still the matter of legacy information contained on paper media. This sets the context for the importance of effective character recognition applications. Such applications would be indispensable in enabling the transition from current forms of paper media to digital forms of communication. Computer processing could also be used to greatly increase the ability to interpret and catalog existing paper documents to allow their inclusion in efficient, electronic databases.

One way to categorize the applications of this technology is to differentiate between general-purpose page readers and task-specific readers. General-purpose page readers are designed to handle a broad range of documents such as business letters, technical writings, and newspapers. Most of these readers can read machine-written text, but only a few are able to recognize hand-printed alphanumerics.

Task-specific readers handle only specific document types. Some examples of this are bank checks, credit-card slips, envelopes or airline tickets. Often task-specific readers, more so than general-purpose readers, are used in high-volume applications requiring high system throughput. For this reason, it is helpful that task-specific reading is usually a more constrained problem, since such a reader will often only be required to extract data from certain pre-defined document regions, such as the account number on a credit-card slip. However, at the same time, task-specific applications present a different set of problems, mostly because such readers are often required to recognize handwritten strings. The task presented in this paper of reading bank-check images is a task-specific problem whose focus is off-line handwritten character recognition.

### 1.1.2 General Methods

Many different approaches to off-line handwritten character recognition have been proposed in implemented in the past. The most basic approach to the problem is template matching. Template matching involves comparing a characters digitized image to those of candidate character images and selecting the best match. Since such a brute force comparison is expensive, the standard approach is to narrow down the possible candidates by examining characteristics of the image and limiting the templates to only the ones which match these features. [12]

Other approaches use spatial, topological, or frequency-domain features to describe characters [13,14]. Such approaches employ methods from the decision-theoretic branch of pattern recognition. Decisions about the input character are based upon how the observed features map into the feature space of those characters stored in the database. [5] Another strategy based upon extraction of these features employs a different decision strategy which can be viewed as a decision tree as opposed to a feature map. [12]

The approach employed by the package with which this research deals involves using a neural network to recognize characters. Neural networks will be explained in further depth later this chapter, but the basic idea is to show a set of known characters to the net so that it may adjust its internal weights in an attempt to learn the character. After this training has been performed, the program should be able use this network to recognize unknown characters.

This survey covers the majority of the popular off-line handwritten recognition algorithms, though it is by no means comprehensive. However, regardless of which algorithm is used, in order for a program to recognize a character, it must know what that character looks like. In other words, it must have an internal representation for the

different characters to-be-recognized.

The manner of representation used in the program is defined by the programmer. In the case of a template-matching program, the most common representation of the candidate characters is a bitmap, which would then be compared to the bitmap of the input character. For other approaches, a unique feature descriptor list or a unique syntactic pattern grammar may be used to represent the character. These representations are not limited to one per character. Often it is necessary to account for several possible forms of a character that differ significantly. For instance, a system recognizing numerals can store information about the numeral 7 both with and without a horizontal crossbar [12].

### 1.1.3 The Recognition Process

The two essential components in the process of most character recognition algorithms are segmentation and recognition. Segmentation involves decomposing a string into its constituent components. This is an element of any algorithm at some level in most cases, algorithms will segment into individual characters though it is also possible to only segment to the level of whole words as units. The recognition process deals with determining what each segment contains. This process is where the majority of character recognition work has been aimed [2].

As Elliman and Lancaster wrote, these two components of segmentation and recognition are a chicken and egg situation [14]. This is because it is not fully possible to segment a string without recognizing what the characters are in the string, yet it is also not possible to recognize individual characters without first segmenting the string.

One proposed method of dealing with this problem is to implement a closed-loop

system in which the segmentation and recognition processes provide each other with feedback. Feedback can provide the segmenter with information about the current region being examined, such as whether or not the region contains a recognizable character [16].

Another possible approach is to perform segmentation completely independent of recognition. Here algorithms are required which have can analyze the shape of characters and the nature of joints between characters to make up for the lack of semantic information provided by the closed-loop system [12].

A number of applications dealing with numerical character recognition, such as OCR of tax forms and deposit slips, have simplified this dilemma because the forms which they are recognizing have boxed-off rectangles for each digit. By doing this, the strings are pre-segmented so that the system only has to deal with recognition [2].

## 1.2 Neural Networks

Neural networks are a creation of modern artificial intelligence originally meant to simulate true, biological networks of neurons. Neurons are specialized cells found in animals that carry signals across the nervous system. They receive input from other neurons through protuberances called dendrites. A neuron does nothing unless the collective influence of all its input reaches a threshold level. Whenever that threshold level is reached, the neuron produces a full-strength output to connected neurons by sending a signal down the axon, the neurons output channel. [17]

Artificial neural networks are a paradigm for computing. There are many varieties of neural-networks, including perceptrons, interpolation nets, and approximation nets, but this paper will only discuss the most popular type feed-forward neural networks. These constructions consist of simulated neurons that are connected to other nodes via links that correspond to the axon-dendrite connection. Each link is associated with a weight. In this

way, one nodes influence on another is the product of the influencing neurons output value times the links weight. In this way, a large weight corresponds to a strong excitation, and a link with a small, negative weight could be viewed as a weak inhibition [18].

Each node combines the separate influences from its input links to calculate an activation function. An example of a simple activation function would be a node that sums the input values to determine the nodes output. The output of each node is either 0 or 1 depending on whether the sum of the inputs is below or above the nodes threshold value. [18].

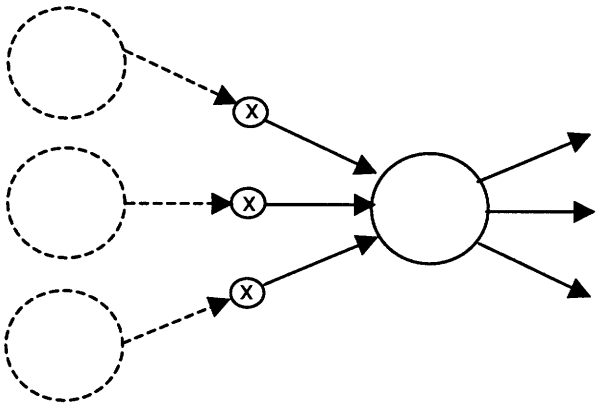


**Figure 1-1: A Simulated Neuron.**

In reality, much of the character of real-life neurons is not modeled by such simple constructions. Therefore, it is argued that such artificial networks do little to shed light on real neural activity and can not hope to perform anywhere near the complex level that the real things do. However, neural networks do have a clear strength in applications involving a great deal of sensory data. For this reason, neural networks have seen application in investment analysis, process control, marketing, signature analysis, and many other pattern-recognition fields, including handwriting recognition [19].

### 1.2.1 Feed-Forward Neural Networks

As mentioned above, the most common form of neural network is a feed-forward network. The basic structure of a feed-forward network is a set of unlinked input nodes, each of which has only a single input. These nodes lead to one or more layers of hidden, intermediate nodes. These intermediate layers are linked sequentially so that outputs from each layer only go forward, that is, to one or more nodes in the next intermediate layer. The final layer is the output layer, where each node has only a single output [18].

Determining the number of intermediate layers and the number of nodes in each layer is up to the discretion of the programmer. In general, a neural network with more weights can conform to the input data better. Of course, more weights results in more calculations which means more computation, however in general, the computation time of a neural network is low enough that this is not a primary concern. However, one does need to be concerned with over-fitting. A neural net with too many weights will have a tendency to fit the training data in too many ways, so that at a certain point, more weights actually decreases performance of the network [18].

### 1.2.2 Backpropagation Training

Perhaps the greatest strength of neural networks is the ability to train the network by showing it a large sample of test data, on which it can train its weights. The most popular method of doing this is through backpropagation. This procedure implements the idea of moving in the direction of most rapid performance improvement by varying all the weights simultaneously in proportion to how much good is done by individual changes. The key to the procedure is a relatively efficient way to compute how much performance improves with individual weight changes [18]. The details of the backpropagation

procedure are not vital to understanding of this research. For the equations and their explanation, one can refer to [18].

Using this procedure, the network is trained in epochs. A single epoch is one complete iteration through the sample data set. With each epoch, the network improves performance at recognizing the training data. The point that one must be concerned with while training is to avoid over-training. Just as with over-fitting, this results when the network becomes to closely conformed to the samples in the training set and fails to perform in a more general role in recognizing samples outside this training set. The way to avoid over-training is by testing the network on an independent data set at each epoch to determine at which point performance stops improving.

## 1.3 Recent Advances in Off-Line Character Recognition

Character recognition is still a blossoming field, meaning that new techniques are always being pursued, while others are being made obsolete. Here a few recent advances in the field shall be described.

### 1.3.1 The Chaincode Algorithm

A statistical-style contour-based feature extractor for hand-written numerical recognition has been developed by Kim and Govindaraju [21]. Chaincoding is a traditional method of representing image data by storing the boundaries of regions in the image in terms of directions. This allows for compact storage and the ability to easily detect sharp features and concavities [22].

The chaincode algorithm calculates the chaincode direction for each pixel by convolving a kernel over the image, which yields contour features for regions in the character. Contours are smoothed by local averaging of each pixels slope. The calculated curvature is then quantized into one of 5 values. These values serve as input to a neural network classifier trained to recognize numerical digits.

This algorithm has been employed successfully in a system developed at the Center for Excellence in Document Analysis and Recognition (CEDAR) at SUNY Buffalo for the purpose of reading zip codes on tax forms [23].

### 1.3.2 Bayesian Network Classifiers

While there has been general trend towards using backpropagation neural networks in character recognition, attempts have been made to use Bayesian network classifiers in both character segmentation and character recognition [24].

Bayesian networks, like backpropagation neural networks, are classifiers for generalizing a solution to an inexact problem without developing heuristic rules. Both are trained by showing the classifier a large set of data upon which it can customize a solution. Bayesian classifiers use this data to estimate the *a posteriori* probability of the occurrence of each class, given an input vector [24].

Smith, McNamara, and Bradburn performed experiments to use Bayesian classifiers for character segmentation in light of the success of Kahan, Pavlidis, and Baird in using Bayesian networks of binary valued features in character recognition [25]. The segmentation experiments on component merging and splitting problems did not demonstrate the same degree of success displaying success rates of 92.2% for component splitting and 69.8% on component merging.

### 1.3.3 Integrated Recognition and Segmentation

A highly robust recognition algorithm was developed by Rocha and Pavlidis for numerical character analysis based on recognition of subgraphs homeomorphic to previously defined prototypes of characters [26]. This system uses an algorithm which allows recognition and segmentation to be performed concurrently.

The crux of the algorithm is that gaps are identified as potential parts of characters by implementing a variant of the notion of relative neighborhood used in computational perception. This means that each subgraph of strokes that matches a character prototype is recognized anywhere in the word, even if it corresponds to a broken character or to a connected character. Each of these matches is assigned a value based on the quality of the match.

Then, each recognized subgraph is introduced as a node in a directed net that compiles different alternatives of interpretation of the features in the feature graph. A path in the net represents a consistent succession of characters in the word. This method allows the recognition of characters that overlap. A search for the optimal path in this network yields the best interpretation of the string.

So, this algorithm recognizes broken characters by looking for gaps between features that may be interpreted as part of a character and connected characters by the allowance of matching connected, but adjacent strokes.

### 1.3.4 Gray-Scale Segmentation and Recognition

The majority of recognition algorithms binarize the digital images prior to processing, that is, they threshold the image so that each pixel has either a 0 or 1 value. According to Lee, Lee, and Park [27], by doing this, a great deal of useful information may be lost, especially

regarding segmentation of touched or overlapped characters.

By analyzing gray-scale images, specific topographic features and the variation of intensities can be observed in the character boundaries. In [27], Lee et al. propose a technique for using gray-scale information in both character segmentation and recognition. In the proposed methodology, the character segmentation regions are determined by using projection profiles and topographic features extracted from the gray-scale images.

Then a nonlinear character segmentation path in each character segmentation region is found by using a multi-stage graph search algorithm. In order to confirm the nonlinear character segmentation paths and recognition results, a recognition-based segmentation method is adopted. This system demonstrates effective segmentation and recognition of touched and overlapped characters.

### 1.3.5 Genetic Algorithms Applied to Multilayer Cluster Neural Networks

Lee, S. proposed in [29] a scheme for off-line recognition of totally unconstrained handwritten numerals using a multilayer cluster neural network trained with the backpropagation algorithm. The innovation here is in the use of genetic algorithms to avoid the problem of finding local minima in training the multilayer cluster neural network with gradient descent technique to and improve the recognition rates.

A genetic algorithm is a model of a machine that derives its behavior from the processes of evolution in nature. This is usually done by the creation of a population of individuals with chromosomes describing routines to solve the problem at hand. By letting this population function as in nature, natural selection will cause the individuals

best suited to solve the problem to survive and reproduce [18].

In the proposed scheme by Lee, Kirsch masks, filters used for edge detection and enhancement, are adopted for extracting feature vectors. These vectors are used as input for a three-layer cluster neural network with five independent subnetworks. In order to verify the performance of the proposed multilayer cluster neural network, experiments with handwritten numeral database of Concordia University of Canada, that of Electro-Technical Laboratory of Japan, and that of Electronics and Telecommunications Research Institute of Korea were performed. For the case of determining the initial weights using a genetic algorithm, 97.10%, 99.12%, and 99.40% correct recognition rates were obtained, respectively.

### 1.3.6 Numerical Recognition with Deformable Templates

Jain and Zongker studied the application of the pattern-recognition technique, deformable templates, to the problem of handwritten digit recognition [30].

Deformable templates are a technique in pattern-recognition most commonly referred to in the context of face recognition. This method of template matching consists of directly comparing the appearance of a given image with a reference image by means of a suitable metric, an energy function. This function contains terms attracting the template to distinctive features in an image. The deformable templates interact with the sample image in a dynamic manner, and matches to templates are obtained by a minimization of the energy function. This method is relatively insensitive to variations in scale, tilt, and most forms of image noise [31].

In Jain and Zongkers proposal, two characters are matched by deforming the contour of one to fit the edge strengths of the other, and a dissimilarity measure is derived

from the amount of deformation needed, the goodness of fit of the edges, and the interior overlap between the deformed shapes. According to [30], classification using the minimum dissimilarity results in recognition rates up to 99.25 percent on a 2,000 character subset of NIST Special Database 1.

### 1.3.7 Combined Discrete and Continuous Recognition System

Ha and Bunke designed and implemented a numerical-string recognition system described in [32], which utilizes both a discrete and a continuous recognition system operating independently.

A discrete recognition system (like the majority of those described above) is one that employs a segmentation routine and a recognizer which acts on pre-segmented images. A continuous recognition system employs a sliding window which scans from one end of the image to the other so that the input image is converted to a sequence of feature vectors each of which represents the sub-image within the window at a given position.

Performing on the NIST Special Database 1, the system demonstrated 7.34%, 21.34%, and 6.54% error rates for the discrete system, the continuous system, and the combined system respectively. While neither the discrete nor the continuous system employed in this system was particularly innovative, this shows that by combining the outputs of the two independent systems, it was possible to achieve recognition rates higher than was possible using either system alone.

# Chapter 2

# The Winbank Project

The research done for this paper was done as part of the Winbank software project, whose purpose is to design OCR targeted at courtesy amounts on bank checks. The project was started before this research began and the code written here is meant for use with that codebase. The core of the Winbank character-recognition engine is a set of patented pre-processing steps leading into a neural-network based recognizer. The Winbank code is written for the 32-bit Windows 95/NT platform to be compiled with Borland C++ version 5.0.

## 2.1 Architecture of the Winbank Program

The architecture of the Winbank program before any modifications made in this research consists of the seven modules described here [1]:

i) *Input Image Handler*: This system uses a HP ScanJet scanner and the HP scanner control libraries to digitize pre-written strings of numerals into binary bitmaps. These bitmaps are dynamically thresholded so that each pixel possesses a value of either 0 or 1.

ii) *Pattern Representation Stage 1*: The bitmap is divided into connected regions and these regions are normalized to a constant height. This may involve zooming in or out on the image, however the aspect ratio remains unchanged.

iii) *Segmentation Module*: The string is then passed to the segmentation module, which breaks the string into distinct and meaningful pieces. Since the image has

already been broken up into its connected regions, this module deals strictly with connected characters, which are determined by looking at aspect ratios. The segmentation procedure is a hybrid of two algorithms a contour-splitting algorithm, originally described in [4] and a hit-and-deflect strategy (HDS) described in [5].

iv) *Segmentation critic*: The decomposition of the string performed in steps *ii* and *iii* is reviewed by the segmentation critic, which can veto any segmentation performed by the contour-splitting algorithm so that the HDS algorithm is used or simply judge it as a bad segmentation.

v) *Pattern Representation Stage 2*: In this stage, the segmented bitmaps of each numeral are resized to a form suitable for the neural-network recognizer. Because the input of the neural-network is a 16 x 16 layer, each bitmap is normalized to this size.

vi) *Pre-Processing*: This stage takes each bitmap in its current form and attempts to process it into a form as close to those on which the neural network was trained as possible. The first step is slant correction, during which the character is maneuvered into an upright, vertical position. Thinning and rethickening are the other pre-processing steps. These standardize the width of all numerals to a two-pixel thickness.

vii) *Neural-Network Based Recognizer*: The recognition stage consists of a three-layer neural network trained using the backpropagation algorithm described in [6]. The input layer consists of 256 linear units, arranged in a 16 x 16 matrix; the intermediate, hidden layer is made up of 40 nodes; and the output layer consists of ten units one for each of the ten digits.

## 2.2 Segmentation

Many of the algorithms dealing with segmentation of handwritten numerals have been proposed in the context of postal zip-code reading where the number of digits is known beforehand to be either five or nine [7,8,9]. This piece of information is not available for the specific application of courtesy-amount recognition, where the string in question is a dollar amount, which most probably has a cent value. The only grammar information available for this form of character recognition is punctuation, such as periods and commas.

The segmentation employed in the original version of Winbank has four portions as outlined above connected component extraction, contour splitting, the segmentation critic, and HDS. The connected component extraction algorithm described in [1] divides the string into blobs so that all the pixels in a blob are adjacent and no two separate blobs contain adjacent pixels.

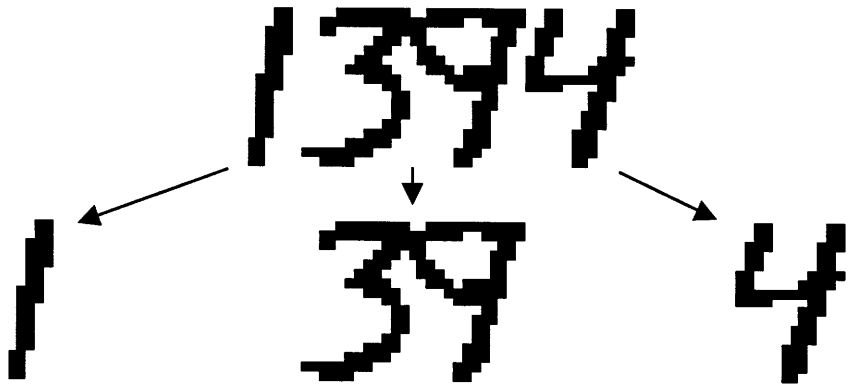

**Figure 2-1: Example of Connected Component Extraction.**

### 2.2.1 Segmentation Critic

Each blob is passed on to the segmentation critic. The segmentation critic checks the blob for its aspect ratio, which is the ratio of height to width. If the input does not exceed a predetermined minimum aspect ratio, then the blob is determined to be an unsegmented pair of characters, and it is passed on to the next module in the segmentation stage, which is the contour splitter. This is done because connected characters should generally be wider than single characters. This algorithm does not catch all connected characters and yield false positives on occasion. It also fails to recognize unsegmented strings of more than two characters.

### 2.2.2 Contour Splitter

The contour splitter uses ideas taken from [4] with some changes for this specific application. The process is described in detail in [1]. The basic idea is to view the top and bottom profiles of the connected component and locate the best coordinates between which to draw a line in order to split the characters. If the following conditions are not met, the blob is rejected and passed on to the next step: the bottom of the upper profile must remain above the top of the lower profile; the straight-line cut must be either horizontal or sloped up  vertical cuts are better handled using an HDS strategy; the straight line making the cut must not pass through more than two contour lines; and the resulting segments from the cut must be larger than set dimensional requirements for character fragments. If these requirements are not met, the segmentation critic sends the component on to the HDS segmenter.

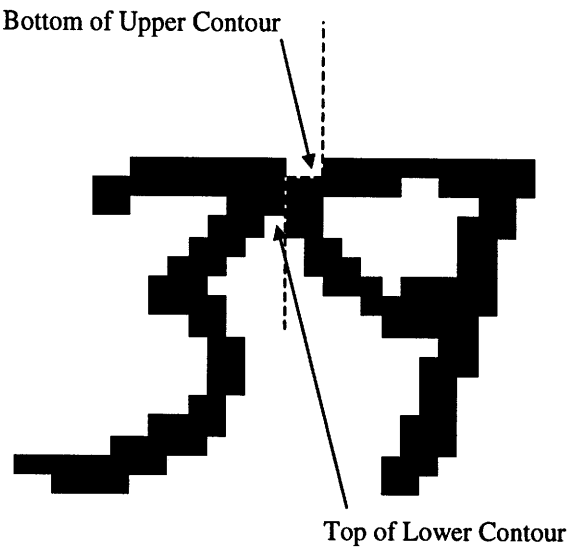


**Figure 2-2: Example of Contour-Splitting Segmentation.**

### 2.2.3 Hit-And-Deflect Strategy Segmenter

The Hit-And-Deflect strategy implemented in Winbank was derived from [5], but the specific details are described in [1]. The basic idea is to have a line veer its way through the space between the two characters, ricocheting off contours and deferring intersection with the lines as long as until finally the connection is split. The output of this phase is passed on to the segmentation critic, and if, as above in judging the initial segmentation, the height and weight tests fail, this segmentation is rejected. At this point, a rejected segmentation causes the segmentation critic to recombine the pieces and proceed to recognition.

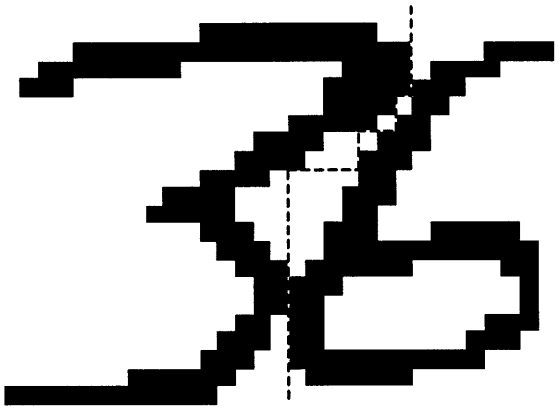

**Figure 2-3: Example of HDS Segmentation.**

### 2.2.4 Punctuation

Once the segments have been identified, the first step in recognition is punctuation. The system goes through all the segments and determines a vertical range for the string. It then checks each segment and looks for certain characteristics, which are used to describe the different forms of punctuation found in a courtesy amount  dash, comma, and period, where a dash is the horizontal line used to separate the numerator from the denominator in a fraction used to represent the cent amount in a string.

If the height of the segment is less than one-quarter the full height of the string and the aspect ratio is less than a threshold amount, the segment is recognized as a dash. If the height of the segment is less than one-quarter the full height, the top of the segment is located beneath the midline bisecting the height of the string, and the aspect ratio exceeds the threshold amount for a dash, the segment is recognized as a period. Finally, if the top of the segment is located beneath the midline and the aspect ratio is greater than another threshold, the segment is recognized as a comma.

## 2.3 Preprocessing

The steps leading up to the bitmap being passed to the recognizer are known as the preprocessing steps in the algorithm. As mentioned above, these steps are particularly important in this system where the recognizer is neural-network-based, because the inputs to the network must all be as uniform as possible, and must go through the same processes that were applied to the training data.

### 2.3.1 Normalization

Normalization zooms to either reduce or enlarge character images to a uniform size. The obvious reason for doing this is because the recognizer needs input of a uniform size. In the Winbank system, this is a 16 x 16 bitmap size per character. Normalization also functions to perform some noise reduction on larger characters. This happens because by zooming out on a character, some of the finer details will be lost, including undesired noise.

### 2.3.2 Slant Correction

The majority of handwritten text has some degree of slant. Some recognition methods require that this be reduced or the tilt may cause recognition to fail. Slant correction straightens a character with the goal that its main vertical components stand perpendicular to its baseline and its main horizontal components lie parallel to the baseline. This way, the same network can be used to recognize both left-handed and right-handed writing and the level of slant in the writing has a minimal effect on recognition ability.

In the Winbank system, slant correction is performed by rotating the bitmapped

character both clockwise and counter-clockwise around the center of the bitmap. The horizontal width of the character is calculated at intervals during this rotation to find the point at which the character is most narrow. This rotation is determined to be the slant-corrected representation.

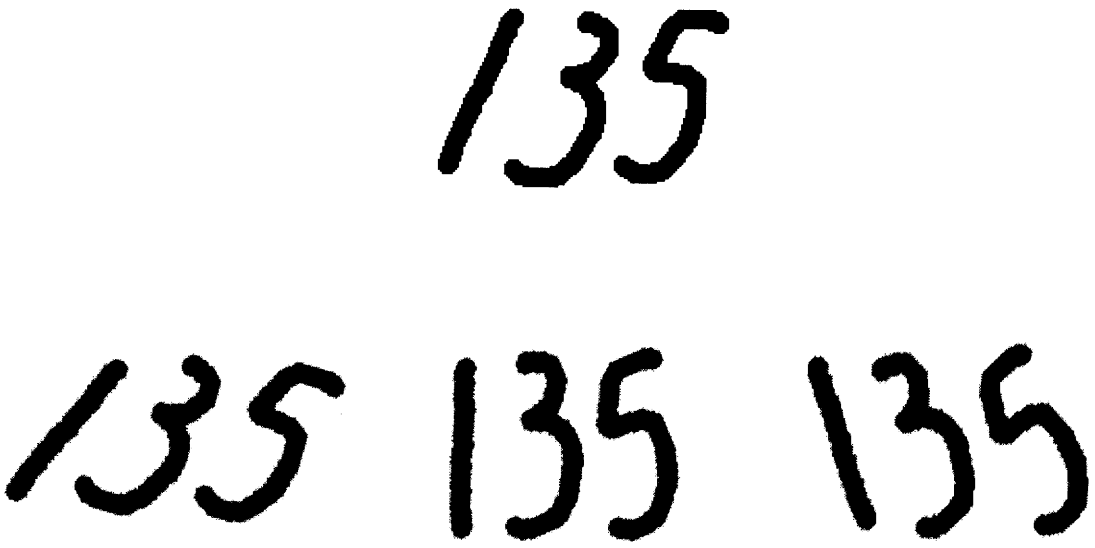

**Figure 2-4: Example of Slant Correction.**

### 2.3.3 Thinning

Thinning is the extraction of a characters skeleton by reduction of its overall thickness. This skeleton may be one to several pixels thick. The key is that the characters skeleton eliminates all features irrelevant to the shape of the character. This includes getting rid of varying stroke thickness and most surface irregularities. In the original Winbank system, thinning is performed immediately after normalization and the image is reduced to a one-pixel skeleton.

### 2.3.4 Rethickening

Thickening, like thinning, involves changing the thickness of strokes in a character while maintaining all of the essential shape of the character. Thickening plays an important role in the Winbank system, since the recognizer is neural-network based. Therefore, for successful recognition, it is necessary to have the pixels from the images match the pixels from the images in the training set. This is facilitated by thickening strokes in the characters to a two-pixel width. This way, there is a larger margin for error than would be available if the characters had only a one-pixel skeleton.

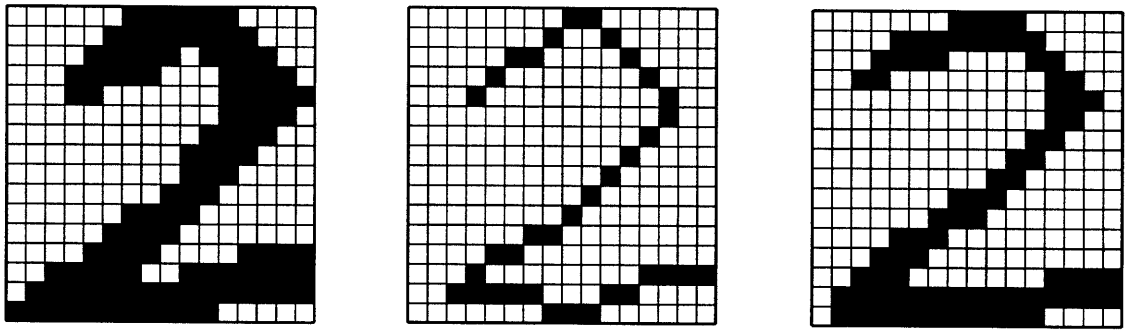

**Figure 2-5: Example of Thinning and Rethickening.**

## 2.4 The Neural Network

The first layer, the input layer, has 256 units in a 16 x 16 matrix, which represents the 16 x 16 bitmaps, which serve as inputs to the network. The second layer is the intermediate, hidden layer consisting of 40 nodes. The final layer has the ten output nodes, each representing a single digit.

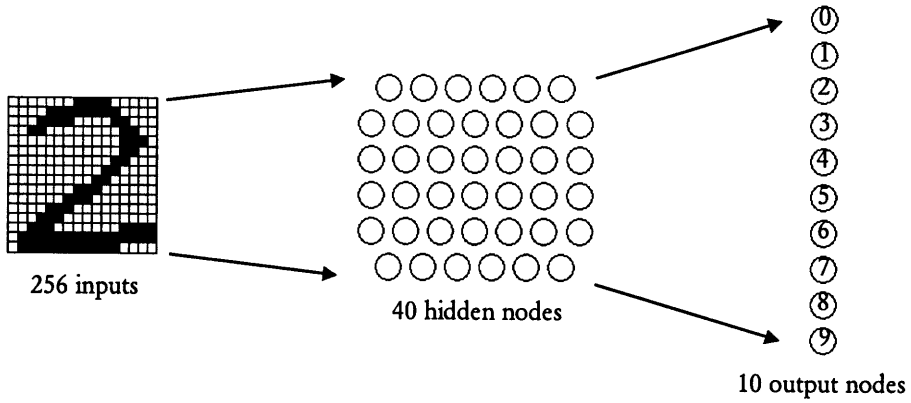


**Figure 2-6: Graphical Representation of Neural Network.**

The neural network was trained from data taken from the NIST database of numerical character fields taken from census data. A backpropagation training routine was employed to determine weights for the network. The training data set was 8000 characters in size. The evaluation data set was 2000 characters in size.

For every character, the neural network produces confidence ratings for all ten possible digits. If there is an output above a certain confidence threshold, without another digit with a sizable confidence, the high confidence level output is chosen as the recognized digit. If there are no outputs with considerable confidence ratings or if there are multiple outputs with high confidence ratings, no recognition is made and an unrecognizable output is produced.

## 2.5 Performance

The performance of the network was gauged on a 800-check set of checks obtained from a local bank [11]. The original program demonstrated the following performance on this set.

| Classification Class | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| Number of Patterns | 341 | 435 | 330 | 337 | 359 | 301 | 311 | 366 | 195 | 212 |
| Number Correct | 265 | 364 | 275 | 275 | 305 | 233 | 255 | 309 | 149 | 168 |
| Number Rejected | 37 | 25 | 19 | 23 | 29 | 41 | 34 | 16 | 13 | 22 |
| Number Substituted | 39 | 46 | 27 | 39 | 25 | 27 | 22 | 41 | 33 | 22 |
| Correct Percent | 77.7 | 83.7 | 83.3 | 81.6 | 85.0 | 77.4 | 82.0 | 84.4 | 76.4 | 79.2 |
| Substitution Percent | 11.4 | 10.6 | 8.2 | 11.6 | 7.0 | 11.1 | 7.1 | 11.2 | 16.9 | 10.4 |

**Table 2-1: Per-Digit Performance for the Original Winbank Program on [11].**

On per-check recognition, on the 800 checks, 416 were recognized correctly, 180 were rejected, and 204 were accepted with substitution errors.

# Chapter 3

# Grammar

In many applications of character recognition, it is possible to use information about surrounding characters and forms of punctuation to gain more information about the character in question. Often times, knowing this information it is possible to find constrain the set of possible characters and/or to find out which characters in the set are most likely. For instance, looking at a 3-character English word string, knowing the first two characters are a and n, it is possible to use a dictionary to say that the likely possibilities for a third character are d, t, and y. Another example of the use of grammar is in zip-code reading, where it is known that a zip-code string is either 5 characters or 9 characters with a hyphen separating the $5^{th}$ and $6^{th}$.

## 3.1 Grammar in the Courtesy Amount

While there are few restrictions on the order of characters or number of characters for courtesy amounts, there is some information that can be gleaned from the placement of punctuation in a dollar-value string. The following three points can be used to check for correctness of segmentation and recognition:

i) There should be exactly two numbers in a string located after a period. More or less than this could be the result of bad recognition of the period or bad segmentation of characters after the period.

ii) The occurrence of a dash or slash should be preceded by a value between 0 and 99 inclusive, which represents the cent value of the courtesy amount. Either of the

strings 00 or 100 may follow the dash or slash, however the cent value can also be represented without the 00 or 100 with these punctuators. If either of these conditions is not met, it could be due to bad recognition of the punctuator or bad segmentation or recognition of the other characters.

iii) There should be exactly one occurrence of the following three punctuators: period, dash, and slash.

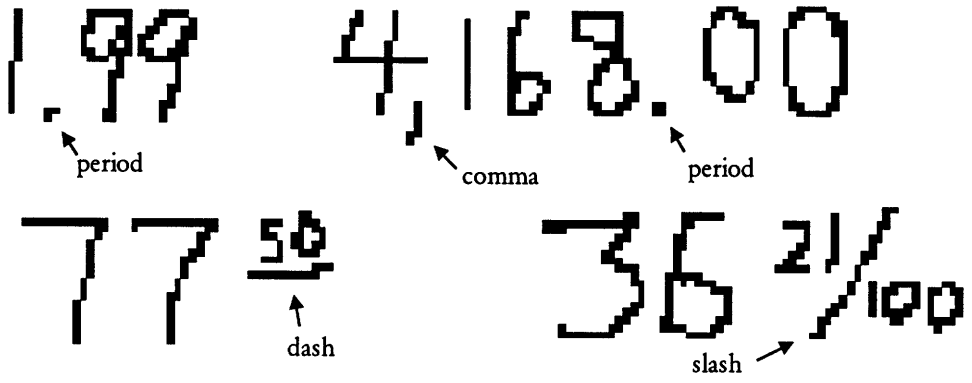


**Figure 3-1: Examples of Punctuators.**

## 3.2 Need for Grammar

The original Winbank system completely lacks any form of grammar processing. This means that it lacks a number of important features that could be added through the implementation of a grammar engine.

First of all, after recognizing each character, the software simply returns this string of digits and punctuators. If this system is intended to be used in any automated manner, the actual value of the courtesy amount should be calculated rather than having the string itself returned. For example, the string in the lower left of Figure 3-1 should be recognized

as 77.50 instead of 77-50 while the string in the lower right should be returned as 36.21, not 3621/100.

Secondly, if the courtesy amounts are being returned, information such as accidental punctuators should be omitted whenever possible. The best example of this would be a misplaced period. This is a common occurrence because random noise can easily be spotted as a period according to the identification criteria used by Winbank. Also, if a comma is mistaken for a period, this out-of-place period should not be reported as output. For example, in a string recognized as 7.540.99, the first period is clearly out-of-place, either as random noise or a mistaken comma, and should thus be ignored.

## 3.3 Detection of Slashes

Besides any sort of grammar processing, Winbank also lacks a means of detecting the slash form of punctuation. So, a means of detecting this punctuator is proposed.

In the original version of Winbank, slashes would fall through the punctuation checking and after slant correction, would usually be recognized as a one. Therefore, it becomes necessary to look at a segment recognized as a one in its context to determine whether it is a slash or a one. The following criteria were developed to differentiate ones from slashes.

A segment identified as a one is succeeded by a 00 or 100 terminating sub-string and no period is detected in the whole string. This segment in question should be recognized as a slash. However, as mentioned above, the use of a slash to represent the cent value in a string does not require the 00 or 100 sub-string. In this case, the grammar which can be used to identify the slash is if it is the ultimate segment in the string and is preceded by two digits of height less than two-thirds the maximum height of the characters in the string with no period preceding these two digits.

Analyses were performed as a part of this research on the same 800-check data set upon which the neural network was trained [11] using Winbank routines to test the validity of these criteria for differentiating ones from slashes. It was found that there were 68 cases where a segment identified as a one was succeeded by a 00 or 100 terminating string and no period was detected in the whole string. In every one of these cases, the character identified as a one was a slash punctuator.

While there were 14 cases of slashes in this context which were not identified by testing based upon this criterion, these cases involved the slash being connected to other characters, so that it would not have been possible to detect the slash without reengineering the segmentation algorithms. Therefore, the first criterion is considered valid and will be used to detect a slash punctuator in this context.

As for the situation where a 00 or 100 sub-string is not present, the second criterion was tested in a similar analysis. There were 39 cases where the ultimate segment in the string was recognized as a one and was preceded by two digits of height less than two-thirds the maximum height of the characters in the string with no period preceding these two digits. In every one of these cases, the character recognized as a one was in fact a slash punctuator.

As before, there were cases in which testing based on this criterion did not detect a slash, however again, in all but two of the 11 cases, this was the result of poor segmentation. Therefore, it is determined that this is a reasonable means of testing for a slash in this context and will be used in the grammar engine to detect slashes.

## 3.4 Implementation of the Grammar Engine

In the original Winbank program, all of the punctuation identification is implemented in

the step after segmentation and before size normalization. These are the routines for identifying periods, commas, and dashes. Since the routine for slash detection requires information from the recognizer, this means that obviously the entirety of the punctuation engine will not be able to reside in this location.

Instead, detection of slashes along with the grammar processing must be performed after recognition. So, the first step performed after the neural net recognizer is applied is to check to see whether or not there is a slash present. This is based on the criteria described in the previous section. The code that was added as a part of this research follows this rough outline (detailed code in Appendix A):

```
1 if(last three segments are "100" and there is no period preceding the
  last two digits)
2     if(fourth to last segment is a "1" and third to last segment has
      height less than 2/3 max string height)
3           fourth to last segment is a SLASH.
4     else
5           third to last segment is a SLASH.
6 else
7     if(last segment is "1", there is no period preceding the last two
      digits, and second and third to last segments have height less
      than 2/3 max string height)
8           last segment is a SLASH.
```

The purpose of the if statement on line 2 is to determine whether the 1 in the 100 is a slash preceding 00 or the 1 in a 100 sub-string. If a slash is detected, it is considered the correct cents punctuator and the two digits preceding it are recognized as the cents value of the courtesy amount.

At this point, the code added for this research iterates through all the segments in the string and determines whether or not there is a slash, period, or dash in the string. If

there is not, then the engine searches for a comma. If a comma is found with two terminating digit segments following it, the assumption is made that this comma was misrecognized and is actually a period, and the classification is changed. If there is no slash, period, or dash and no comma is found in the proper position, the string is identified without a cents portion.

While there is recognition for a slash punctuator in the original Winbank program, there is no handling for it or verification that it is placed correctly. A dash is considered to be placed correctly by the same criteria used with slashes from the previous section. The code added to process dashes follows this rough outline (detailed code in Appendix A):

```
1 if(last two segments are "00" and there is no period preceding last
  two digits)
2     if(third to last segment is a "1" and fourth to last segment is a
      DASH)
3           fourth to last segment is a correct DASH
4     else
5           if(third to last segment is a DASH)
6                 third to last segment is a correct DASH
7 else
8     if(last segment is a DASH, there is no period preceding the last
      two digits, and second and third to last segments have height less
      than 2/3 max string height)
9           last segment is a correct DASH
```

If a segment is considered a correct DASH, that means it is the correct cents punctuator, and the two digits preceding it are recognized as the cents portion of the courtesy amount. If a dash is encountered in the string and not classified as a correct DASH, it is considered a bad segmentation, which is explored in Chapter 5.

If a period is found preceding the last two digits of the string, it is considered the cents punctuator and the last two digits are recognized as the cents portion of the courtesy amount. Because of the criteria used to identify valid slashes and dashes, neither of these two forms of punctuation can coexist with each other or with a correctly placed period. Therefore, if either form is present in a string along with a period, it is assumed that the period is misplaced and is ignored.

# Chapter 4

# Adjustment of Pre-Processing Steps

Analyzing the pre-processing steps in the Winbank recognition engine, it was observed that the output after slant correction, thinning, and rethickening processes was not completely uniform.

## 4.1 Need for Change in the Pre-Processing Steps

The irregularities in question resulted from size-reduction from the slant correction and thinning of characters. Since the size normalization was performed prior to these two pre-processing steps, any changes in size made after normalization were not accounted for.

Slant correction reduces the size of normalized characters, because prior to the slant correction, characters are normalized to fit the 16 x 16 bitmap in both width and height dimensions. However, the goal of slant correction is to minimize the horizontal width of the character. So, its possible that after slant correction, the width of the character has decreased after this step. This causes a problem in the neural-network recognition, since the characters should be as uniform as possible in both training and recognition for optimal performance.

Thinning results in size normalization because the parallel thinning algorithm, as described in [3], causes a decrease in stroke thickness both vertically and horizontally. This is evident in the examples from Figures 4-1 and 4-2 that display bitmap outputs from the Winbank program. Because uniformity is the most important factor, this reduction would not matter as long as it was consistent. However, as one can see from the difference

between Figures 4-1 and 4-2, the degree of the size reduction is directly related to the stroke thickness of the character. This size reduction is in general a more significant factor than that resulting from slant correction.

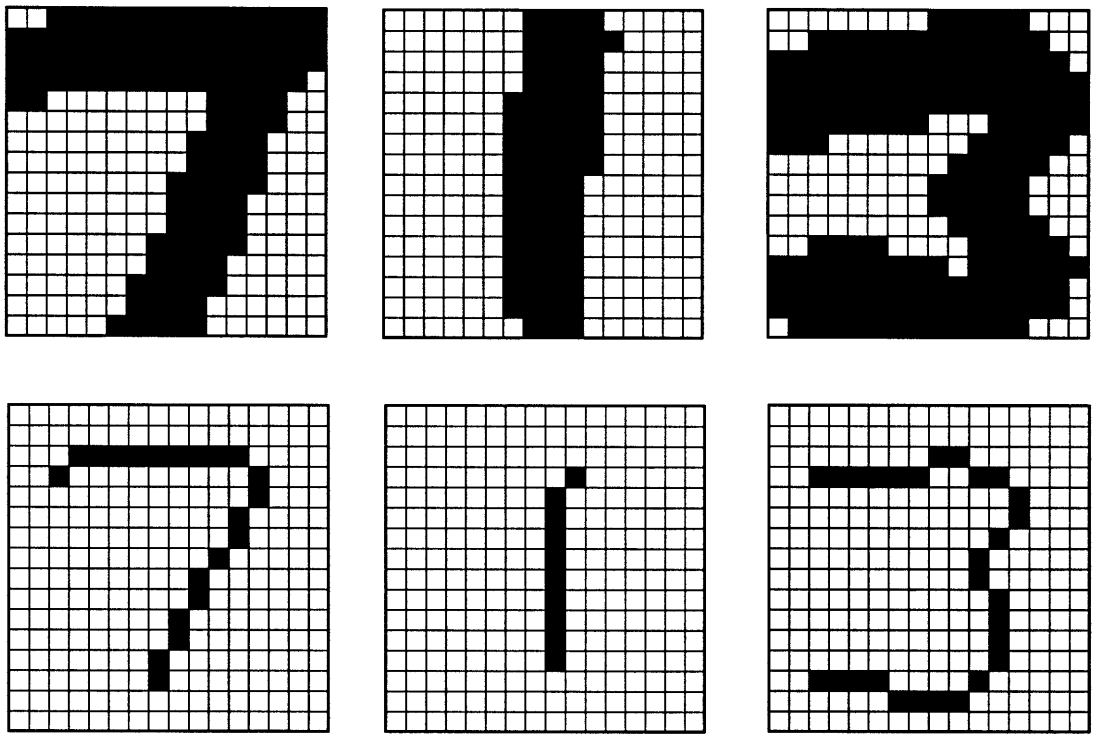

**Figure 4-1: Example of Size Irregularities from Pre-Processing (Thick Strokes).**

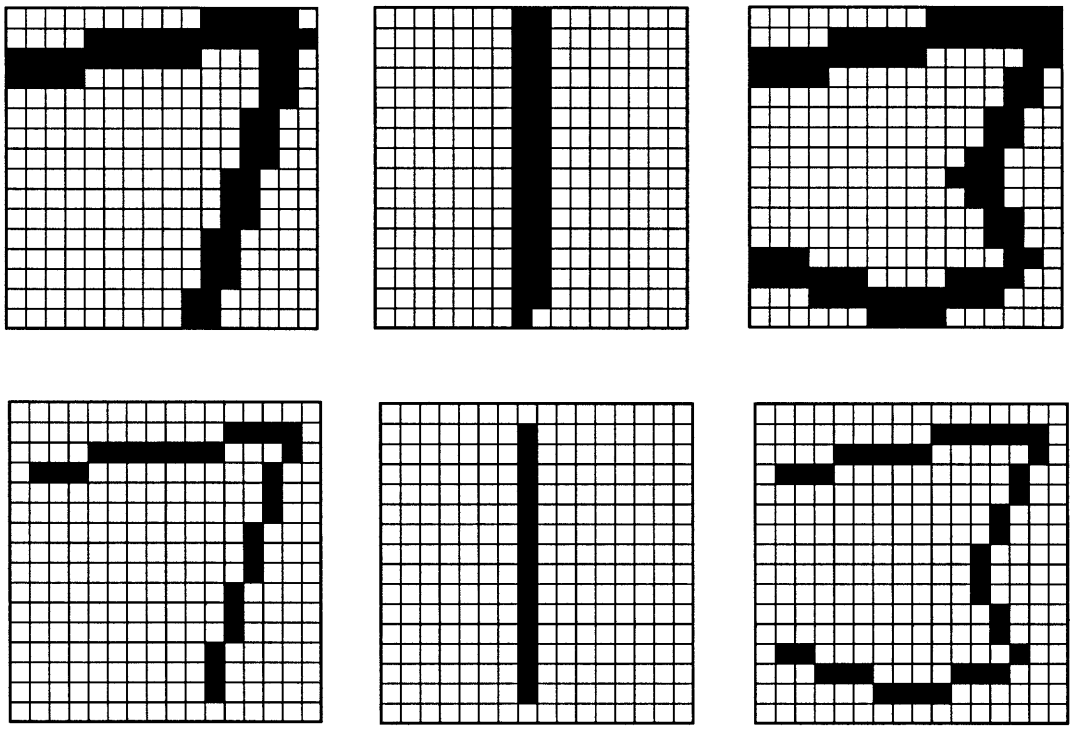

**Figure 4-2: Example of Size Irregularities from Pre-Processing (Thin Strokes).**

## 4.2 Implementation of Pre-Processing Modification

The solution to this problem was to perform the size normalization at two points in pre-processing. The original size normalization performed prior to slant correction was kept, simply because the thinning process has to be performed in a normalized-size space in order for the one-pixel thinning to mean anything. But Winbank was changed so that a second size normalization was performed after thinning and before rethickening. The same size-normalization procedure was used so little extra code was required.

The decision for placement of the new step was reached despite the fact that normalization after thinning but before rethickening could have the effect of diminishing the effects of the thinning process by potentially increasing the thickness of any contours. This happens as demonstrated in Figures 4-3 and 4-4, where instead of a one-pixel thick

skeleton, the output after size-normalization may be two pixels thick on some occasions. This is considered not to be a problem because of the nature of the neural net recognizer. The stroke thickening is a more desirable side-effect than the original size irregularities because the key to the neural network is for similarly shaped characters to be as close to uniform size as possible so that contours fall in the same location in each bitmap. While normalizing after thinning will have a tendency to thicken stroke sizes by a small margin, the sizes will remain uniform. On the other hand, with the old system of no size normalization, characters such as those in Figure 4-1 and 4-2 will dilute the network by having similarly shaped characters have non-uniform sizes.

The other possible problem that could result from size normalization at this point in the pre-processing steps is that rethickening causes size irregularities just as thinning does. So, the other possibility would be to place the extra size-normalization step at the end of pre-processing, after rethickening. However, the overall character size change due to rethickening was found to be negligible in comparison to the other two pre-processing steps, so no extra measures were taken to account for the size changes created by that process. The code for the new processing step and the location from where it is called are included in Appendix B and A respectively.

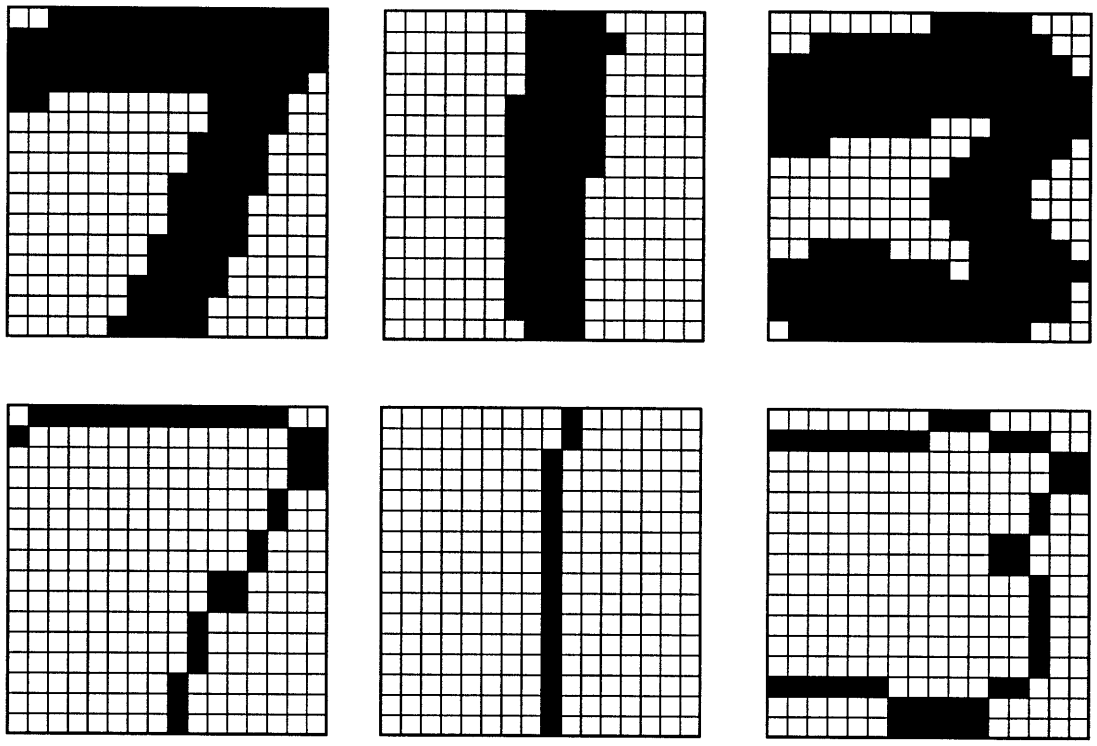

Figure 4-3: Sample Outputs of the New Pre-Processing Routine (Thick Strokes).

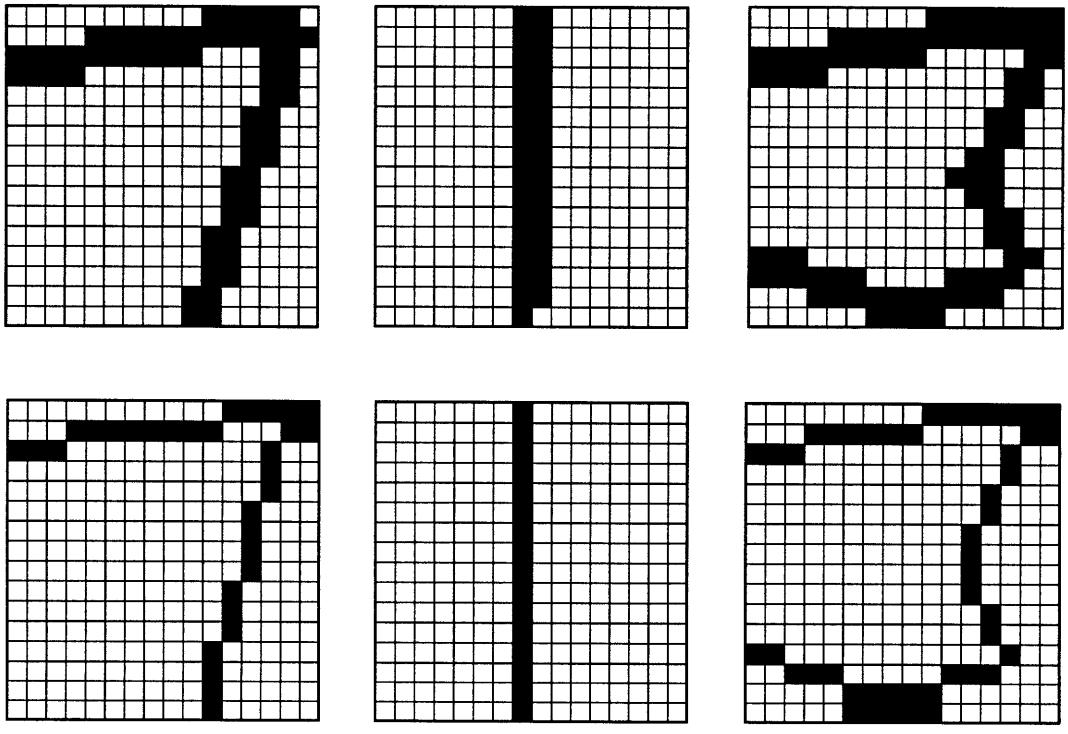

Figure 4-4: Sample Outputs of the New Pre-Processing Routine (Thin Strokes).

# Chapter 5

# Cross-Checking Segmentation and Recognition Stages

In the original version of the Winbank recognition scheme, there is no communication between the segmentation and recognition modules. This means that if an incorrect segmentation is made and the recognition module finds this, it is reported as unrecognizable and that if the recognition module fails to recognize a segment, which could be a connected pair of characters, no effort is made to segment the blob.

## 5.1 Need for Recognition Feedback

As mentioned above, there are two sorts of segmentation problems being looked at here a segmentation being made where it should not be and a segmentation not being made where it needs to be.

A common example of the first problem is an incorrect segmentation of a five. Four and five are the only two-stroke characters among numerals written in the traditional Arabic style. Of these two characters, it is more common to find five split into two unconnected segments. This happens when there is a gap between the two pieces of the five. If this gap separates the two blobs when the image is digitized, the original Winbank connected segment extractor will form two blobs out of the five. A number of different outcomes could result from this, however in all of them, the two pieces will persist

through the recognition process and undoubtedly yield an unrecognizable character or a bad recognition since there is no means of piecing the two segments together.

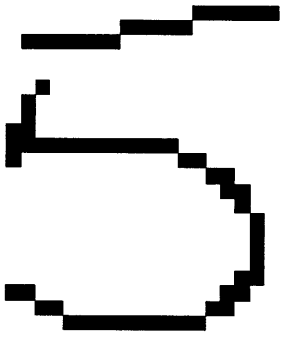

**Figure 5-1: Example of Segmented Five.**

The second problem is less common but must also be considered. The most common occurrence of an unsegmented connected pair of characters is when a one is connected to another number. This will not be segmented because the one does not add much to the width of the blob, so that the aspect ratio does not drop below the threshold required to attempt a segmentation. Almost invariably these blobs will result in an unrecognizable character.

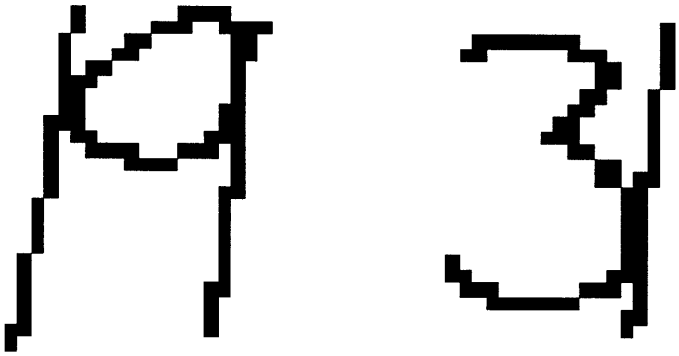

**Figure 5-2: Example of Unsegmented Connected Pairs.**

## 5.2 Implementation of Feedback Mechanism

As mentioned in Chapter 1, many algorithms have been proposed to allow for feedback between segmentation and recognition routines. The problem is that first of all, none of them have been shown to be dramatically superior to a quality sequential segmentation/recognition algorithm, such as the one in the Winbank program. Secondly, to implement any of these algorithms would require a complete reengineering of the routines that make up the Winbank recognition process. Therefore, the aim was to devise a feedback scheme that could fix the most common segmentation problems without having to rewrite the whole OCR system.

The proposed solution fixes both problems with one change to the recognition scheme that neatly fits in with the Winbank routines and data structures. The change occurs in the engine when there is an unrecognizable, non-punctuation segment in the recognition.

i) The unrecognizable character and its left neighbor segment are pieced together into one blob. This blob is then passed through the pre-processing and recognition steps.

ii) The same is done with the character and its right neighbor.

iii) The unrecognizable character is passed to the segmentation routine, overriding the aspect ratio requirement.

After these tests are complete, the outputs are analyzed. In most cases, the result of the test will be garbage, which will result in no recognition at all or a low confidence rating. If exactly one of them exceeds the confidence threshold required for a recognition, then that is accepted as the recognition. If more than one of them exceed the confidence threshold, the solution that yields the highest confidence rating is accepted. If none of the

tests produce a solution with a high enough confidence rating, then the unrecognizable state is kept.

The new code added for this research follows the following outline (detailed code in Appendix A):

```
1 if(blob[n] is unrecognizable)
2     Merge(blob[n] and blob[n-1]) into leftblob.
3     Pre-process and Recognize leftblob.  Output is leftresult with
      confidence leftconfidence.
4     Merge(blob[n] and blob[n+1]) into rightblob.
5     Pre-process and Recognize rightblob.  Output is rightresult with
      confidence rightconfidence.
6     Override aspect-ratio requirement and attempt to segment blob[n]
      first with contour-splitting and then HDS if that fails into blob1
      and blob2.
7     If segmentation succeeds, pre-process and recognize blob1 and
      blob2.  Output is blob1result and blob2result with confidence
      blob1confidence and blob2confidence.
8     bestconfidence = max(leftconfidence, rightconfidence,
      min(blob1confidence, blob2confidence).
9     if(bestconfidence > RECOGNITION_THRESHOLD)
10          switch(bestconfidence)
11                leftconfidence: bestresult = leftresult.
12                rightconfidence: bestresult = rightresult.
13                min(blob1confidence, blob2confidence): bestresult =
                  blob1result, blob2result.
14    else
15          bestresult = UNRECOGNIZABLE.
```

# Chapter 6

# Special Character Adjustments

Analyzing the performance of the neural network on a set of hand-written checks obtained from a bank [11], it was found that particular forms of some numbers had particularly poor recognition in the system. This chapter describes specific changes to the system aimed at improving recognition for these forms.

## 6.1 Need for Special Character Adjustments

The two character forms in question are 3-stroke ones, such as 1 as opposed to |, and sevens written in the international style, that is, with the horizontal crossbar in the middle of the character.

The justification for the poor performance while dealing with these characters is simple. When the neural network is trained on samples, which include two different forms of the digit, the weights for that particular output node must account for both forms. This means that in training the network, the two forms will, in effect, counteract each other to decrease the effectiveness of recognizing either form.

## 6.2 Implementation of Seven Solution

The standard way to solve this problem is as follows. Rearchitect the neural network so that there is an output node for each form of the digit. Actually, the two output nodes are not completely separate, because after determining the two confidence levels, the output

for the digit in question is considered to be the maximum of the two values. This way, the two output nodes cannot compete with each other. This is necessary because the original Winbank code not only looks at the highest confidence level output of the neural network but also the second highest. If the second highest output is above a certain threshold, the result is considered ambiguous and the character is deemed unrecognizable. So, in actuality, this rearchitecture involves adding an extra intermediate layer that deals exclusively with the digit in question.

If this change is made, the network must also be retrained. For this retraining, samples must be provided for both forms of the digit and differentiate between the two types. In this way, recognition of the two forms is isolated so that each is treated as a separate digit instead of having counteracting forces as before.

This solution is ideal for the case of distinguishing the two forms of the digit seven. In the case of seven, with the original structure of the neural network, the international-style seven was recognized with particularly poor accuracy. This is because the majority of the sevens on which the network was trained were with the non-international form. So, the weights associated with the inputs corresponding to the horizontal crossbar, which differentiates the two forms, did not have a significant effect on the recognition of a seven. Instead, international-style sevens were often being identified as fours or nines.

An output node was thus added to represent the international-style of the digit, and the network was retrained differentiating the two forms and including a significant number of both forms. To keep the ratio of intermediate, hidden nodes to output nodes constant, the number of nodes in the middle layer of the neural network was adjusted from 40 to 44 to account for the change in the number of output nodes from 10 to 11.

The code to handle the rearchitected neural network is outlined here (detailed code in Appendix A):

```
1 Pre-process and Recognize blob.  Output is bestresult,
secondbestresult, bestconfidence, and secondbestconfidence.
2 if(bestresult and secondbest are the two forms of 7)
3     secondbestconfidence = third best confidence level.
4 if(bestconfidence > RECOGNITION_THRESHOLD and secondbestconfidence <
      AMBIGUITY_THRESHOLD)
5     result = bestresult.
6 else
7     result = UNRECOGNIZABLE.
```

## 6.3 Implementation of One Solution

In the case of the digit one, the two different forms appear drastically different when passing through the Winbank recognition process. This is due to size normalization. In the case of the one-stroke form, when slant corrected and thinned, this form will normally be represented by a line with a width profile of at most two or three pixels. When this representation is expanded to a 16 x 16 bitmap, it will typically appear as a large block encompassing a large majority of the space in the map. Because of the high variability associated with size-normalizing a digit of this shape, a separate solution was employed in order to deal with this difficulty.

The problem to be solved was to find a separate way of differentiating ones from other characters, either with or without using the neural network. The solution that was come upon was to calculate the second moment of the horizontal coordinate of pixels comprising the digit [10]. This is to be calculated at the point in pre-processing after thinning the digit and before the second size-normalization. The equation for this calculation follows:

$$\sigma_x^2 = \sum_{i=1}^{n} \frac{(x_i - E(x))^2}{n}$$

Where E(x) is the first moment, or expected value, calculated by:

$$E(x) = \sum_{i=1}^{n} \frac{x_i}{n}$$

The justification behind using the second moment, or variance, of the horizontal coordinate is that after slant-correction and thinning, a one will have a much smaller horizontal variance than any other digit. This is less obvious when dealing with the three-stroke form of the one than with a one-stroke form. However the following data, which was calculated for this research gives the average, minimum, and maximum for horizontal variance for a 1000-digit subset of the NIST database [20], backs up the assumption.

| Digit | Average $\sigma_x^2$ | Minimum $\sigma_x^2$ | Maximum $\sigma_x^2$ |
|---|---|---|---|
| 0 | 17.61 | 9.43 | 23.23 |
| 1 | 0.33 | 0.00 | 1.80 |
| 2 | 11.55 | 6.41 | 17.56 |
| 3 | 10.92 | 5.86 | 15.72 |
| 4 | 13.44 | 5.50 | 22.41 |
| 5 | 8.79 | 3.17 | 16.72 |
| 6 | 13.21 | 5.45 | 17.89 |

| | | | |
|---|---|---|---|
| 7 | 11.65 | 6.14 | 22.69 |
| 8 | 8.55 | 4.03 | 15.10 |
| 9 | 11.55 | 4.72 | 21.11 |

**Table 6-1: Horizontal Variance Data for 1000 digits in the NIST Database.**

This data demonstrate that in every case for an 1000-digit subset of the NIST database [20], setting a cutoff of 2.0 for horizontal variance distinguished a one from another digit. After making this distinction and determining that the charasccter in question is a one, the course of action that was chosen was to not normalize the character in the horizontal direction. Ones are however still normalized in the vertical direction so that they cover the entire 16 pixels vertically. In this way, ones, which are by default positioned in the middle of the 16 x 16 bitmap after thinning, will be easily recognized by the neural network. This is because almost every instance of the digit will contain a straight vertical line in the middle of the bitmap, which is not present in any other digit representation.

# Chapter 7

# Results and Conclusions

## 7.1 Results and Analysis

After the above changes were made to the Winbank codebase, the network was retrained on the NIST database as before [20], the following performance was observed on the 800-check data set [11].

| Classification Class | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| Number of Patterns | 341 | 435 | 330 | 337 | 359 | 301 | 311 | 366 | 195 | 212 |
| Number Correct | 275 | 414 | 282 | 285 | 312 | 250 | 266 | 323 | 150 | 171 |
| Number Rejected | 17 | 5 | 12 | 11 | 21 | 20 | 19 | 12 | 11 | 14 |
| Number Substituted | 49 | 16 | 36 | 41 | 26 | 31 | 26 | 31 | 34 | 27 |
| Correct Percent | 80.6 | 95.2 | 85.5 | 84.6 | 86.9 | 83.1 | 85.5 | 88.3 | 76.9 | 80.7 |
| Substitution Percent | 14.4 | 3.7 | 10.1 | 12.2 | 7.2 | 10.3 | 8.4 | 8.5 | 17.4 | 12.7 |

**Table 7-1: Per-Digit Performance for the Modified Winbank Program on [11].**

As for per-check recognition, on the 800 checks, 451 were recognized correctly, 151 were rejected, and 198 were accepted with substitution errors.

For per-digit recognition, the general trend was for the number of rejected digits to decrease while the number correct and number substituted both rose. This would be a result of the segmentation-recognition feedback. A number of the rejections from the old results were because of digits being segmented incorrectly. Not all of these errors were

corrected, but a good portion were. The change in pre-processing steps may have also had something to do with the general upward trend on number of correct recognitions.

The special-character adjustments on ones and sevens was successful. For ones, the percent correctly recognized rose from 83.7% to 95.2%, while the number substituted dropped from 10.6% to 3.7%. The major error expected to be corrected with sevens was the substitution of international sevens with other digits. This enhancement was observed as the percent substituted for the digit seven dropped from 11.7% to 8.5%.

Overall per-digit recognition increased from 82.7% to 85.6% while per-check recognition increased from 52.0% to 56.4%. Since the average number of digits per check in the database was 3.98, this indicates that factors besides strictly character recognition were responsible for improved courtesy-amount recognition. This means that the modified punctuation and grammar engine had a successful effect also.

## 7.2 Further Study

Based on the results of this research, it is possible to direct efforts for improvement of the system. The work that showed the most promise was the special-character adjustments and the segmentation-recognition feedback.

Looking especially at the improvement seen with recognition of ones, it seems that a worthwhile effort would be to find methods for differentiating particular digits. The improvement made for ones was possible because it made the ones appear completely different than other digits to the neural network. However, one is partially unique in this respect because of its shape, so this would not be as easy to achieve dealing with other characters.

Therefore, it may not be possible to develop such specialized-character methods to use in conjunction with the neural network. Instead, efforts could be made to perform

such checks independent of the neural network. In this way, other forms of recognition could be used in parallel to the neural-network recognizer. This could have the two-fold benefit of increased ability to recognize some characters and more accurate confidence-level determination. The latter becomes almost as important as the former since the most important factor with this form of character recognition is to minimize the number of substitution errors.

As for the segmentation-recognition feedback, the results show that it did have the desired effect since the per-digit rejection rates dropped, however this resulted in both higher correct recognition rates and substitution rates, so it was not a complete success. Unfortunately, results were not determined for exclusively the segmentation-feedback communication change, however, with all changes taken into account the per-digit recognition improved from 82.7% to 85.6% while substitution rates dropped from 10.1% to 9.5%. So, it seems that instituting segmentation-recognition feedback to decrease rejection rates is a definite plus to the Winbank system, however a more complicated system, such as the closed-loop systems described in [16] might be the next step.

# Appendix A

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include <stdlib.h>
#include <conio.h>
#include <windows.h>
#include <alloc.h>
#include "scan.h"
#include "str.h"
#include "digittyp.h"
#include "protos.h"
#include "head2.h"
#include "datatype.h"
#include "cents.h"
#include "slant.h"
#include "back.h"
#include "postproc.h"
#include "scanner.h"

extern void DoNormalize2(int [NORM_X][NORM_Y]);

/*=======================================*/
/*         defines and macros            */
/*=======================================*/
#define MAX(a,b) (a>b ? a : b)
#define MIN(a,b) (a<b ? a : b)
#define BLOB_THRESH 25

int DigitHeight(digitType digitAttr)
{
      return digitAttr.lowerRightRow-digitAttr.upperLeftRow;
}

void Insert(char *str,int n, char c)
{     for(int i=strlen(str);i>=n;i--)
            str[i+1]=str[i];
      str[n]=c;
}

void Delete(char *str,int n)
{     for(int i=n;i<strlen(str);i++)
            str[i]=str[i+1];
}

int Process(int normMap[NORM_Y][NORM_X], int newMap[NORM_X][NORM_Y])
{
  if ((DoSlant(normMap)) == 0)
    return 0;
  DoThining(normMap);
  DoNormalize2(normMap);
  DoThicken(normMap);
  for(int k=0;k<NORM_Y;k++)
```

```
    for(int m=0;m<NORM_X;m++)
      newMap[m][k] = normMap[k][m];
}

void Recognize(int newMap[NORM_X][NORM_Y], char result[2],int &maxact)
{
  RecognType result;
  float *percent;
  int ind_maxact1 = 0, ind_maxact2 = 0  ;
  float maxact1 = 0.0, maxact2 = 0.0 ;
  int j,l;

  Recognize_Hand(newMap, &result, 0);

  j=result.maxActNode;
  percent=result.actsPtr;

  for(l = 0; l<nOutputNodes; l++)
    if(maxact1 <  percent[l])
    {
      maxact1 = percent[l] ;
      ind_maxact1 = l ;
    }
  for(l=0; l<nOutputNodes; l++)
    if(l != ind_maxact1)
      if (maxact2 < percent[l])
      {
        maxact2 =  percent[l] ;
        ind_maxact2 = l ;
      }
  if(((ind_maxact1 == 7)&&(ind_maxact2 == 11)) ||
     ((ind_maxact1 == 11)&&(ind_maxact2 == 7)))
  {
    ind_maxact1 = 7;
    maxact2 = 0;
    for(l=0; l<10; l++)
      if(l != ind_maxact1)
        if (maxact2 < percent[l])
        {
          maxact2 =  percent[l] ;
          ind_maxact2 = l ;
        }
  }
  else
    if(ind_maxact1 == 11)
      ind_maxact1 = 7;

  if(maxact1 < 0.7 || (maxact1 > 0.7 && maxact2 > 0.25))
  {
    char postchar[30];
    int q=j;
    j=start_postproc(normMap,percent);
    if(q!=j)
      badchar = 1;
    else
      if((maxact1 + .3)<.7)
```

```
        badchar = 1;
  }

  if(!badchar)
  {   itoa(j,result,10);
      maxact=maxact1;
  }
  else
    result[0]='\0';
  badchar = 0;
}

void DoRecognition(HWND hWnd, MATRIX *image, char *recog_numbers)
{
  tpSEG  curSegment = NULL;
  extern digitType *GetDigit();
  tpSTR   inputString;
  tSTRMAP inputMap;

  char tmpstr[60];
  int    digitCnt = 0;
  char rtt[32];
  digitType *digitAttr;
  int i=1;
  float x1,y1;
  int status;
  int k,m,row,col;
  int badchar = 0;
  int iThreshold;
  static int normMap[NORM_Y][NORM_X];
  static int newMap[NORM_X][NORM_Y];
  int numsegs;
  int digits;
  int decimal_yes_no = 0;
  int iMatch[32];
  for(k=0;k<32;k++)
    rtt[k]=' ';


  //First, remove temporary storage files
  rem_files();

  inputString = Str_Create();
  iThreshold = abs(255 - GetThresh(image));

  Str_Write(hWnd,inputString,image,iThreshold);
  numsegs = Str_Segment(hWnd,inputString);

  DoCentsData(hWnd,inputString,numsegs);
  digits = GetNumDigits();
  int cRealDigits=0;
  int maxBottom=0, maxTop = 0, nTop, nBottom;
  for (digitCnt = 0; digitCnt < digits; ++digitCnt)
    {
      int cBottom=0, cTop = 0;
      digitAttr = getDigit (digitCnt);
```

```
    if((digitAttr->isDigit ==DIGIT)&&(DigitHeight(*digitAttr)>15))
    {     for (int digitCnt2 = 0; digitCnt2 < digits; ++digitCnt2)
      {   digitType *digitAttr2 = getDigit (digitCnt2);
      if((digitAttr->isDigit ==DIGIT)&&(DigitHeight(*digitAttr)>15))
        { if(abs(digitAttr2->upperLeftRow-
                 digitAttr->upperLeftRow)<10)
          cTop++;
        if(abs(digitAttr2->lowerRightRow-
               digitAttr->lowerRightRow)<10)
          cBottom++;
          }
      }
    if(cTop>maxTop)
      {   maxTop = cTop;
      nTop = digitAttr->upperLeftRow;
      }
    if(cBottom>maxBottom)
      {   maxBottom = cBottom;
      nBottom = digitAttr->lowerRightRow;
      }
    }
  }
DoTestCents(nTop,nBottom);

for (digitCnt = 0; digitCnt < digits;
     ++digitCnt)
  {
    curSegment = (getDigit(digitCnt))->map;
    /* check if the segment is a puntuator */
    digitAttr = getDigit(digitCnt);

    //Print presorted segments to file
    DoNormalize(curSegment,normMap,
            &decimal_yes_no);
    sv_segs(normMap, "presort", digitCnt);

    /* send to preprocess and to recognize */
    if ((digitAttr->isDigit != DIGIT)||(DigitHeight(*digitAttr)<15)||
      ((abs(digitAttr->upperLeftRow-nTop)>10)&&
       (abs(digitAttr->lowerRightRow-nBottom)>10)))
    {
      switch(digitAttr->isDigit)
        {
        case PERIOD:
          rtt[i-1] = '.';
          iMatch[i-1] = digitCnt;
          i++;
          break;
        case COMMA :
          rtt[i-1] = ',';
          iMatch[i-1] = digitCnt;
          i++;
          break;
        case SLASH :
          rtt[i-1] = '/';
          iMatch[i-1] = digitCnt;
```

```
      i++;
      break;
    case DASH  :
      rtt[i-1] = '-';
      iMatch[i-1] = digitCnt;
      i++;
      break;
    case DBADSEG :
      rtt[i-1] = '&';
      iMatch[i-1] = digitCnt;
      i++;
      break;
    case GARBAGE :
      rtt[i-1] = 'G';
      iMatch[i-1] = digitCnt;
      i++;
      break;
    default
      rtt[i-1] = 'E';
    iMatch[i-1] = digitCnt;
    i++;
    }
}
else
{
  if(!Process(normMap,newMap))
  {
      MessageBox(hWnd, "Slant Error", "ERROR!",
                 MB_OK | MB_ICONHAND);
      rtt[i-1] = '*';
      iMatch[i-1] = digitCnt;
      i++;
      badchar = 1;   /* set badchar flag.*/
  }

  int j=0,l=0;
  float *percent;
  int ind_maxact1 = 0, ind_maxact2 = 0  ;
  float maxact1 = 0.0, maxact2 = 0.0 ;
  RecognType result;
  RecognType result2;


  Recognize_Hand(newMap, &result, 0);
  j=result.maxActNode;
  percent=result.actsPtr;

  for(l = 0; l<nOutputNodes; l++)
      if(maxact1 <  percent[l])
      {
        maxact1 = percent[l] ;
        ind_maxact1 = l ;
      }
  for(l=0; l<nOutputNodes; l++)
      if(l != ind_maxact1)
      {
```

```
      if (maxact2 < percent[l])
        {
          maxact2 =  percent[l] ;
          ind_maxact2 = l ;
        }
    }
if(((ind_maxact1 == 7)&&(ind_maxact2 == 11)) ||
   ((ind_maxact1 == 11)&&(ind_maxact2 == 7)))
  {
    ind_maxact1 = 7;
    maxact2 = 0;
    for(l=0; l<10; l++)
    {
      if(l != ind_maxact1)
          if (maxact2 < percent[l])
          {
            maxact2 =  percent[l] ;
            ind_maxact2 = l ;
          }
    }
  }
else
  if(ind_maxact1 == 11)
    ind_maxact1 = 7;

if(maxact1 < 0.7 || (maxact1 > 0.7 && maxact2 > 0.25))
  {
    char postchar[30];
    int q=j;
    j=start_postproc(normMap,percent);
    if(q!=j)
    {
      badchar = 1;
      rtt[i-1] = '*';
          iMatch[i-1] = digitCnt;
      i++;
    }
    else
    {
      if((maxact1 + .3)<.7)
        {
          badchar = 1;
          rtt[i-1] = '*';
          iMatch[i-1] = digitCnt;
          i++;
        }
        }
  }

if(!badchar)
  {
    char tempchar[2];
    itoa(j,tempchar,10);
    rtt[i-1] = tempchar[0];
    iMatch[i-1] = digitCnt;
    i++;
```

```
        }

      badchar = 0;     /* Reset badchar flag */
    }
  }

rtt[i-1] = '\0';

tpSEG tempseg1,tempseg2;
tpSEG left,right;
tempseg1=SegCreate();
tempseg2=SegCreate();
char resultleft[2],resultright[2];
resultsplitleft[2]="\0",resultsplitright[2]="\0";
float maxactleft=0,maxactright=0;
float maxactsplitleft=0,maxactsplitright=0;

for(int ii=0;ii<i-1;ii++)
  { if((rtt[ii]=='&')||(rtt[ii]=='*')||(rtt[ii]=='E'))
    {
      if(ii!=0)
      {  left=getDigit(iMatch[ii-1])->map;
         right=getDigit(iMatch[ii])->map;
         MergeMaps(left->strmap,right->strmap,tempseg1->strmap);
         tempseg1->topLeft.row=MIN(left->topLeft.row,
                                   right->topLeft.row);
         tempseg1->topLeft.col=MIN(left->topLeft.col,
                                   right->topLeft.col);
         tempseg1->botRight.row=MAX(left->botRight.row,
                                    right->botRight.row);
         tempseg1->botRight.col=MAX(left->botRight.col,
                                    right->botRight.col);

         for(row=tempseg1->topLeft.row;
             row<=tempseg1->botRight.row;row++)
           for(col=tempseg1->topLeft.col;
               col<=tempseg1->botRight.col;col++)
           {
             if(tempseg1->strmap[row][col] == ONE)
               if((col-tempseg1->topLeft.col) < SEG_LEN)
                 tempseg1->segmap[row][col-tempseg1->topLeft.col] =
                      ONE;
           }
         DoNormalize(tempseg1,normMap,&decimal_yes_no);
         Process(normMap,newMap);
         Recognize(newMap,resultleft,&maxactleft);
      }
    if(ii+1!=i-1)
    {  left=getDigit(iMatch[ii])->map;
       right=getDigit(iMatch[ii+1])->map;
       MergeMaps(left->strmap,right->strmap,tempseg1->strmap);
       tempseg1->topLeft.row=MIN(left->topLeft.row,
                                 right->topLeft.row);
       tempseg1->topLeft.col=MIN(left->topLeft.col,
                                 right->topLeft.col);
       tempseg1->botRight.row=MAX(left->botRight.row,
```

```
                                right->botRight.row);
    tempseg1->botRight.col=MAX(left->botRight.col,
                                right->botRight.col);

    for(row=tempseg1->topLeft.row;
        row<=tempseg1->botRight.row;row++)
      for(col=tempseg1->topLeft.col;
          col<=tempseg1->botRight.col;col++)
      {
        if(tempseg1->strmap[row][col] == ONE)
        if((col-tempseg1->topLeft.col) < SEG_LEN)
          tempseg1->segmap[row][col-tempseg1->topLeft.col] = ONE;
      }
    DoNormalize(tempseg1,normMap,&decimal_yes_no);
    Process(normMap,newMap);
    Recognize(newMap,resultright,&maxactright);
}
static tCOORD min,max;
tpSTEP  ppath = NULL;
static tDIMEN blobdims;
int split=0;
static DIR dir;

blobdims.startc=left->topLeft.col;
blobdims.stopc=left->botRight.col;
blobdims.startr=left->topLeft.row;
blobdims.stopr=left->botRight.row;
Profile(hWnd,left->strmap,TOP,left->topLeft.col,
        left->botRight.col,&min);
Profile(hWnd,left->strmap,BOTTOM,left->topLeft.col,
        left->botRight.col,&max);
if((ppath = Bridge(left->strmap,min,max,blobdims)) == NULL)
{
  ((HEIGHT-max.row) >= min.row) ? (dir = UP) : (dir = DOWN);
  if((ppath =
      HitNDeflect(hWnd,left->strmap,dir,
                  min,max,blobdims)) != NULL)
    split = TRUE;
}
else
  split = TRUE;
if(split)
{
  Separate(left->strmap,tempseg1,blobdims.startc,ppath);
  StepDestroy(ppath);
  InitMap(tempseg2->strmap);
  if(GetBlob(left->strmap,tempseg2->strmap,RIGHT,
             &blobdims) == OK)
    if(blobdims.weight > BLOB_THRESH)
      {
        tempseg2->topLeft.row = blobdims.startr;
        tempseg2->topLeft.col = blobdims.startc;
        tempseg2->botRight.row = blobdims.stopr;
        tempseg2->botRight.col = blobdims.stopc;

        for(row = blobdims.startr;row <= blobdims.stopr; ++row){
```

```
                for(col = blobdims.startc;col <= blobdims.stopc; ++col)
                  if(tempseg2->strmap[row][col] == ONE)
                    if((col-blobdims.startc) < SEG_LEN)
                      tempseg2->segmap[row][col-blobdims.startc] = ONE;
              }
              DoNormalize(tempseg1,normMap,&decimal_yes_no);
              Process(normMap,newMap);
              Recognize(newMap,resultsplitleft,&maxactsplitleft);

              DoNormalize(tempseg2,normMap,&decimal_yes_no);
              Process(normMap,newMap);
              Recognize(newMap,resultsplitright,&maxactsplitright);
            } //getblob Ok
        } //split
        float maxactsplit = MIN(maxactsplitleft,maxactsplitright);

if((maxactsplit!=0)&&(maxactsplit>maxactleft)&&(maxactsplit>maxactright
))
    {
       Delete(rtt,ii);
       Insert(rtt,ii,resultsplitright[0]);
       Insert(rtt,ii,resultsplitleft[0]);
    }
    else
       if(maxactleft>maxactright)
       {
         Delete(rtt,ii-1);
         Delete(rtt,ii-1);
         Insert(rtt,ii-1,resultleft[0]);
       }
       else
         if(maxactright>0)
           { Delete(rtt,ii);
             Delete(rtt,ii);
             Insert(rtt,ii,resultright[0]);
           }
     }
   }

   int nStringHeight=abs(nTop-nBottom);
   int nSlash=0;
   for(i=0;(i<32)&&(rtt[i]!='\0');i++);

   if((i>=3)&&(rtt[i-1]=='0')&&(rtt[i-2]=='0')&&(rtt[i-3]=='1'))
   { digitAttr = getDigit(iMatch[i-3]);
     if((i>=4)&&
        (rtt[i-4]=='1')&&(DigitHeight(*digitAttr)<2.0/3*nStringHeight))
     { rtt[i-4]='/';
       nSlash=i-4;
     }
     else
     {
       rtt[i-3]='/';
       nSlash=i-3;
     }
   }
```

```
  else
    if((i>=3)&&(rtt[i-1]=='1'))
    { digitAttr = getDigit(iMatch[i-2]);
      if((rtt[i-2]>='0')&&(rtt[i-2]<='9')&&
       (DigitHeight(*digitAttr)<2.0/3*nStringHeight))
      { digitAttr = getDigit(iMatch[i-3]);
        if((rtt[i-3]>='0')&&(rtt[i-3]<='9')&&
          (DigitHeight(*digitAttr)<2.0/3*nStringHeight))
        { nSlash=i-1;
          rtt[i-1]='/';
        }
      }
    }
  if((i>=3)&&(rtt[i-1]=='0')&&(rtt[i-2]=='0'))
  {   if((i>=4)&&(rtt[i-3]=='1')&&(rtt[i-4]=='-'))
      { rtt[i-4]='/';
        nSlash=i-4;
      }
      else
        if(rtt[i-3]=='-')
        { rtt[i-3]='/';
          nSlash=i-3;
        }
  }
  else
    if((i>=3)&&(rtt[i-1]=='-'))
      { digitAttr = getDigit(iMatch[i-2]);
        if((rtt[i-2]>='0')&&(rtt[i-2]<='9')&&
           (DigitHeight(*digitAttr)<2.0/3*nStringHeight))
        { digitAttr = getDigit(iMatch[i-3]);
          if((rtt[i-3]>='0')&&(rtt[i-3]<='9')&&
             (DigitHeight(*digitAttr)<2.0/3*nStringHeight))
          { rtt[i-1]='/';
            nSlash=i-1;
          }
        }
      }

  if(nSlash)
    if(nSlash>=2)
    { Delete(rtt,nSlash);
      Insert(rtt,nSlash-2,'.');
      if(nSlash!=i-1)
      rtt[nSlash+1]='\0';
    }
  strcpy(recog_numbers,rtt);
}
```

# Appendix B

```
#include <stdio.h>
#include <math.h>
#include <windows.h>
#include "common.h"
#include "segtyp.h"

#define SIZEI XMAX /* number of rows and colomns in the input array */
#define SIZEJ SEG_LEN

#define SIZEP 16  /* number of rows and colomns in the output array */
#define SIZEQ 16

#define MAX_WIN 16  /*MAX_WIN is the maximum window size for reducing
                    to a single cell. Magnifying factor */
#define MOMENT_THRESHOLD 2.0;

/* Statics */
static int input[SIZEI][SIZEJ]; /* FIlled by DataReader */

/* Prototypes */
static void data_reader (char char_input[][SIZEJ]);
struct array_place starting_place (void);
static int normalizer(int norm_map[][NORM_X]);
static void data_reader (char char_input[][SIZEJ])
{
  register int i, j;

  for (i=0; i<YMAX; i++)
     for (j=0; j<SIZEJ; j++)
      input[i][j] = char_input[i][j] - 48;
}




float moment()
{
  int i,j,num=0;
  float sum=0,avg;

  for(i=0;i<NORM_Y;i++)
    for(j=0;j<NORM_X;j++)
      if(input[i][j]) {
      sum+=j;
      num++;
      }
  if(num)
    {
      avg=sum/num;
      sum=0;
      for(i=0;i<NORM_Y;i++)
```

```
      for(j=0;j<NORM_X;j++)
        if(input[i][j])
          sum+=(j-avg)*(j-avg);
      return sum/num;
    }
  return 0;
}

void DoNormalize2(int norm_map[NORM_Y][NORM_X])
{

  int i, j;
  int rows, cols;
  rows = SIZEI;
  cols = SIZEJ;
  s = 1;

  i_min=-1;
  j_min=NORM_Y+1;
  j_max=-1;
  for(i=0;i<NORM_X;i++)
    for(j=0;j<NORM_Y;j++)
      {
      input[i][j]=norm_map[i][j];
      if(norm_map[i][j])
        {
          i_max=i;
          if(i_min==-1)
            i_min=i;
          if(j<j_min)
            j_min=j;
          if(j>j_max)
            j_max=j;
        }
      }

  if(moment()<2.0)
    {
      j_min=0;
      j_max=15;
    }

  float x_step=(float)(j_max-j_min+1)/(float)NORM_X;
  float y_step=(float)(i_max-i_min+1)/(float)NORM_Y;
  for(i=0;i<NORM_Y;i++)
    for(j=0;j<NORM_X;j++)
      norm_map[i][j] =
input[(int)(i_min+y_step*i)][(int)(j_min+x_step*j)];
}
```

# Bibliography

[1] Sparks, P., Nagendraprasad M. V., and Gupta A., An Algorithm for Segmenting Handwritten Numeral Strings, Research Report, Sloan School of Management, 1990.

[2] Discussions with 50k group, including Gupta, A., Khan, S., Wu, T., Sae-Tan, W., Chien, T., August 1997-May 1998.

[3] Nagendraprasad, M.V., Wang, P., Gupta, A., Algorithms for Thinning and Rethickening Binary Digital Patterns, in *Digital Signal Processing*, pp. 97-102, 1993.

[4] Fenrich, R. And Krishnamoorthy, S., Segmenting Diverse Quality Handwritten Digit Strings in Near Real-Time, in the *Proceedings of the United States Postal Service Advanced Technology Conference Proceedings*, pp 523-537, 1990.

[5] Shridhar, M. And Badreldin, A., Recognition of Isolated and Simply Connected Handwritten Numerals, *Pattern Recognition*, 19(1), pp 1-12, 1986.

[6] McClelland, J., Rumelhart, D., *Explorations in Parallel Distributed Processing.*, MIT Press, Cambridge, MA, 1988.

[7] Pervez, A. And Suen, C. Y., Segmentation of Unconstrained Handwritten Numeric Postal Zip Codes, in the *Proceedings of the 6$^{th}$ International Conference on Pattern Recognition*, pp 545-547, 1982.

[8] Hull, J. J, Srihari, S. N., Cohen, E., Kuan, C. L., Cullen, P. And Palumbo, P., A Blackboard-based Approach to Handwritten ZIP Code Recognition, in the *Proceedings of the United States Postal Service Advanced Technology Conference*,

1988.

[9] Mitchell, B. T. And Gillies, A. M., Advanced Research in Recognizing Handwritten ZIP Codes, in the *Proceedings of the United States Postal Service Advanced Technology Conference*, 1988.

[10] Discussion with Mui, L., April 1998.

[11] 800-check bank-check database from Unisys Corporation, 1990.

[12] Sparks, P., A Hybrid Method for Segmenting Numeric Character Strings, Research Report, Sloan School of Management, 1990.

[13] Bongard, M. *Pattern Recognition.* Spartan Books, New York, 1970.

[14] Fu, K. S. *Syntactic Pattern Recognition and Applications.* Prentice-Hall, Englewood Cliffs, NJ, 1982.

[15] Elliman, D.G. and Lancaster, I.T. A review of segmentation and contextual analysis techniques for text recognition. *Pattern Recognition*, 23(3), 1990.

[16] Casey, R.G and G. Nagy. Recursive Segmentation and Classification of Composite Character Patterns. In *Proceedings of the 6th Internation Conference on Pattern Recognition*, 1982.

[17] Stryer, L. *Biochemistry.* Third edition, W.H. Freeman and Company, New York, 1988.

[18] Winston, P. *Artificial Intelligence.* Addison-Wesley Publishing Company, Reading, Massachusetts, 1977.

[19] Smith, Leslie, Centre for Cognitive and Computational Neuroscience, Department of Computing and Mathematics, University of Stirling. October, 1996.

[20] NIST Special Database 1. Binary Images of Printed Digits, Alphas, and Text. Produced by Image Recognition Group. May 1990.

[21] Kim, G., and V. Govindaraju. Efficient Chaincode Based Image Manipulation for Handwritten Word Recognition, *IS&T/SPIE.s Symposium on Electronic Imaging: Science & Technology*, San Jose, CA, 1996.

[22] Freeman H, Computer processing of line drawing images, *Computing Surveys*, Vol. 6, No. 1, pp. 60-97, 1974.

[23] Srihari, S. N., Shin Y., Ramanaprasad V., and Lee D. A system to Read Names and Addresses on Tax Forms, Center for Excellence in Document Analysis and Recognition, SUNY Buffalo, 1994.

[24] Smith, R., McNamara, J., and Bradburn D., Pattern Classification Techniques Applied to Character Segmentation, TRW Financial Systems, Berkeley, CA., 1998.

[25] Kahan, S., Pavlidis, T., and Baird, H., On the Recognition of Printed Characters of Any Font and Size, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 9, No. 2, 1987.

[26] Rocha, J. and Pavlidis, T. Word Recognition without Segmentation, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 17 pp. 903-906, 1995.

[27] Lee, S., Lee, D., and Park, H. A New Methodology for Gray-Scale Character Segmentation and Recognition, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 18, No. 10, 1996.

[28] Fujisawa, H., Nakano, Y., and Kurino, K. Segmentation Methods for Character Recognition, *Proceedings of the IEEE*, Vol. 80, No. 7, pp. 1079-1092, 1992.

[29] Lee, S. Off-Line Recognition of Totally Unconstrained Handwritten Numerals Using Multilayer Cluster Neural Network, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 18, No. 6, 1996.

[30] Kain, A. and Zongker, D. Representation and Recognition of Handwritten Digits Using Deformable Templates, *IEEE Transactions on Pattern Analysis and Machine Intelligence,* Vol. 19, No. 12, 1997.

[31] Yuille, A. L. Deformable Templates for Face Recognition, *Journal of Cognitive Neuroscience,* Vol. 3, No. 1, pp.59-70, 1991.

[32] Ha, T. and Bunke, H. Off-Line Handwritten Numeral String Recognition, Institute for Computer Science and Applied Mathematics, University of Berne, 1997.