A System for Offline Automated Recognition of

Unconstrained Handwritten Numerals

by

Fawaz A. Chaudhry

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 Engineering

and Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

September 11, 2001

Author_____

Department of Electrical Engineering and Computer Science
Sep 11, 2001

Certified by_____

Dr. Amar Gupta
Thesis Supervisor

Accepted by_____

Arthur C. Smith
Chairman, Department Committee on Graduate Theses

A System for Offline Automated Recognition of
Unconstrained Handwritten Numerals
by
Fawaz A. Chaudhry

Submitted to the
Department of Electrical Engineering and Computer Science

Sep 11, 2001

In Partial Fulfillment of the Requirements for the Degree of
Bachelor of Science in Computer Science and Engineering
and Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT:

This paper presents a system that performs offline automated reading of handwritten characters, particularly strings of handwritten numerals. This paper brings together all the research carried out at MIT Sloan Institute, Profit Initiative on handwritten numerals over the past decade on various modules. It puts these modules together in a huge, comprehensive and robust system. Along the way the paper made several improvements in the way these modules communicated with each other as well as discussed and compared the other strategies currently being employed in latest research in the field.

Thesis Supervisor: Dr. Amar Gupta
Title: Co-Director, Productivity From Information Technology (PROFIT) Initiative
Sloan School of Management

TABLE OF CONTENTS

1. Introduction

Throughout history man has strived to develop tools better and machines to do the tasks more efficiently and eventually more proficiently than he possibly could. Machines have been used to an ever increasing extent to reduce physical labor and to achieve what was previously impossible or impractical. The increasing sophistication of machines brought about the modern day computers and also increased our reliance on it. Computer prowess has grown so sophisticated that almost all human mental tasks can be mapped onto them or seems within their reach. One of the most complex problems currently being tackled worldwide includes visual and audio recognition and deduction. A much smaller portion of that research includes handwriting recognition.

The replication of human function has always has been the subject of much research in the artificial intelligence field. Handwriting recognition can be utilized in uncountable number of ways once it's perfected. All sorts of applications carrying from zip-code recognition, check amount encoding, document reading etc come to mind. Huge amounts of resources have been allocated to this task but the complex has revealed layers of complexity that only grow as the research intensifies.

The task of handwriting recognition was initially perceived to be simple due to the early success in recognition of machine printed information. The characters and numerals being printed by machines were uniform and it was easy to apply pattern-matching techniques with tremendous success. This led to a false belief and over anticipation in seeking success in handwriting recognition. Handwritten characters are non-uniform and

encompasses myriads of styles in writing, each of which the system must learn to master. Despite all the tremendous leaps in the technology available to us today, unconstrained handwriting recognition still has relatively poor and unacceptable success rates for most practical purposes. A sub-category of handwriting recognition would be to concentrate only on numerals. Since numerals limit the domain of characters as well as being not connected (except by accident, which is ofcourse dealt with), they have become the focus of the majority of the research and some remarkable results have been produced.

This paper focuses on the research done at MIT on unconstrained handwritten numerals. The paper will describe the architecture of the system devised at MIT in it's most mature form and compares the research with modern research being done on this subject across the globe.

1.1 History of Character Recognition

The history of character recognition is comparatively old in the field of pattern recognition. As mentioned previously, characters were easy to deal with and researchers expected the problem would be easily solved, as well as the fact that there were many potential applications given success in this area. In fact, character recognition has its roots before the dawn of the computer age. In 1929, the first patent for optical character recognition (OCR) was obtained by a German, Tauscheck. Tauscheck's patent used the principle of template matching and employed the optical and mechanical technologies available at the time. Light passes through mechanical masks and is captured by a photodetector. When an exact match with a character being tested occurs, no light passes

through the mask and no light reaches the photodetector. This basic idea of superposition is still used in character recognition applications today, almost 70 years later.

The history of research in OCR follows three main paths: template matching, structural analysis and neural networks. Template matching has been mostly used for the recognition of machine-printed characters, while structural analysis and neural networks have been employed for hand-printed data. Template matching and structural analysis have been employed and studied for many years, but use of neural networks is a relatively new technique that has been popularized in the last 15 years or so.

1.1.1 Template Matching

Template matching is accomplished by superimposing a test template and a known template, taking the total sum of the differences between each sampled value and the corresponding template value; if the degree of difference is within a pre-determined amount, then the templates match. All of the early work done in OCR was analog-based- at the time analog/hardware technology was far more advanced than digital/software technology. In 1962, RCA made a very sophisticated OCR using electron tube technology. The system used 91 channels, and the development team concluded that the system could reliably recognize all of the English and Russian characters. However, no commercial product was marketed and development in this field did not progress at RCA.

1.1.1.1 Basic Template Matching

Of course, the arrival of the computer age changed the direction of OCR with respect to both hardware and algorithms. Instead of the traditional template matching described

previously, research turned towards logical template matching; researchers were able to simplify templates through the process of binarization into two levels of information, i.e. black and white. The most basic method of template matching using computers is known as the peephole method which is basically the computer counterpart of superposition method discussed previously. A two-dimensional memory plane (or an array) is used to store a binarized input image where the position of the character is pre-determined (i.e. top right, center, etc.). Then for an ideal character of constant size and width, the black portions are always black and the white portions are always white. Thus, a logical matching scheme is easily selected. Solatron Electronics Group Ltd. built a product based on this method in 1957. Their product read numbers output by a cash register at a rate of 120 characters/second.

Template matching seems like a sensible and reliable method of character recognition, but it has some inherent weaknesses. First, as can be clearly inferred, template matching involves comparison with all possible candidates which is time-consuming; this factor is magnified given that much of the study here was going on in the 1960s. Secondly, shift variance, rotation variance and other issues of normalization degrade the recognition rate. Researchers began to develop methods which were shift invariant around this time. Two methods they focused on were autocorrelation and moment based methods; the second method is both shift and rotation invariant. Furthermore, transformation and series expansion techniques reduce the dimensionality required to extract a feature vector based on individual points.

Horowitz and Shelton of IBM did very extensive research on the autocorrelation method in 1961. They proposed a very large special purpose shift register machine for calculating autocorrelation exactly and with high speed. However, the results were disappointing. The differences between more than a few characters were less than 1%, degrading performance beyond usefulness.

Another prime researcher in the area of autocorrelation was Iijima of Japan. His work in the early 60s focused on the normalization of characters with regards to displacement and blurring. Through the experimental study of OCR, Iijima realized that blurring an input character could improve recognition rates by increasing the similarity between characters with the same value. Since then, blurring or thickening (as we called it previously) has become a common pre-processing step in recognition. Iijima derived a partial differential and integral equation for recovering displacement by introducing time where its behavior is controlled by a function f(x,t). The equation was implemented on a hardware system with success, however the system was deemed too expensive and other approaches were considered.

In the late 1960s, Iijima investigated the possibility of using similarity itself as a measure of positive matching. At the same time, series expansions methods were being used to extract features and achieved good recognition rates on machine-printed data. Iijima found that using similarity itself involved a comparable amount of computation time. He also found a very important property of similarity related to blurring. Intuitively, one would guess that $S(f_0,f) \rightarrow 1$ (where S is a measure of similarity) as blurring increases, but Iijima proved that when canonical transformation is done $S(h_0,h) \rightarrow \square < 1$. That is,

even with heavy blurring, there are still some essential differences between patterns that remain. Furthermore, if blurring is employed, intuitively, we can reduce the number of sampling points taken. In 1970, Iijima introduced the multiple similarity method based on the principles discussed above. The method was considered to overcome the weakness of similarity against displacement noise. He investigated the trace of patterns in Hilbert space, when displacement noise is expressed by the first and second terms of its Taylor expansion expressed by:

$H_0(r+d),\ d=id_1+jd_2.$

The pattern $h_0(r+d)$ moves on the same definite locus as d moves from 0. Therefore, it is definitely impossible to compensate the displacement noise by just one template; however, more than one template can be used and a new similarity in terms of the new templates can be defined. This method has been very successful with machine printed digits and is the main algorithm used in Toshiba's OCR systems.

1.1.1.2 Moment Based Methods

The moment method can be used to account for position, size and/or slant variation has been relatively successful. In 1962, a first systematic experiment was conducted by Alt of the National Bureau of Standards, in which one font for the alphabet was used (normalized in position, size and slant). Moments up to the second order were used as the normalization factors, and the third to the sixth (22 kinds) were used for classifications. No recognition numbers were given for the experiment, however Alt was optimistic. In spite of this, very little work in the moment based approach followed. Then, in 1987, 25

years later, Cash and Hatamian at Bell Laboratories conducted a reliable experiment on the moment-based method. They used 10 moments for classification. The data set tested consisted of six different machine printed fonts. Their study produced recognition rates over 95% or better for all six fonts. However, the print quality of the digits was good and no rejection class was taken, so those numbers may be inaccurate from a practical point of view.

1.1.1.3 Fourier Series Based Methods

Next, we cover the use of Fourier series expansion in character recognition. Fourier is the most popular series expansion and is thus a natural choice for application to OCR.

Zhan and Oskie wrote a study on the relevance of Fourier series to character recognition in 1972. To review, the set $\{A_k, a_k, |k=1,2,\ldots\}$ is called the Fourier descriptor (FD) for a curve, where A_k and a_k are the amplitude and phase of the kth harmonic term. FD's have invariant properties with respect to position and scale, but they do depend on the starting point of a boundary tracing. The representation of Fourier coefficients of a boundary can be divided in two ways. The first is based on the cumulative angular function (CAF) which is expanded into a Fourier series. In this case, a reconstruction experiment was done using digits made of a 24x24 binary matrix. The results showed that the reconstruction of a hand-written '4' is almost sufficient using up to 10 harmonics (10 amplitudes and 10 phases), but the image was not closed (which is a general feature of the CAF). The second method was developed by Persoon and Fu and published in 1977. They proposed that a point moving along the boundary generates the complex function $u(s) = x(s) + jy(s)$, which is periodic with period S. The FD(CF) becomes:

$a_n = S^{-1} \square \, {}_s u(t) exp(-j2\square \, S^{-1} xnt) dt$

Using CF for the reconstruction experiment, it was found that it out-performed the CAF. In the case of the CF, reconstructed figures are closed and 5 harmonics are almost sufficient to differentiate between closely similar characters such as '2' and 'Z'.

Systematic and reliable experiments were conducted for Fourier Descriptors by Lai and Suen in 1981. In their experiment Fourier descriptors were used both as global features and as local boundary features. They used a data set of 100,00 hand-printed numerals in 64x32 binary arrays. In the first case, recognition rates upwards of 95% were achieved, while in the latter case the rate was only about 80%. Overall, Fourier descriptors are exceptionally insensitive to rotation of the character or the location of the character within the bitmap. They have also proven to be insensitive to noise, however this trait has been a disadvantage in certain cases where useful material is filtered out, i.e. Os cannot be distinguished from Qs.

1.1.2 Structural Analysis

Structural analysis of characters is a technique primarily used for handwritten data. As seen in previous analysis, recognition of machine-printed data is nicely handled by the various methods of template matching. However, in the case of handwritten characters, it is very difficult to construct a template for a specific character because the variation of shape and style can be very large. For this reason, structural analysis of characters began being researched. Furthermore, simple structural methods have also been employed to

complement and verify recognition techniques for machine-printed data as well as constrained hand-printed characters since the earliest phases of OCR.

Unlike template matching, there is no mathematical principle behind structural analysis. Instead, intuition is most reliably used to develop heuristics for recognition. Most of these heuristics are based on the idea that a structure can be broken into parts, and consequently, these parts can be described by features and the relationship between the different parts. The problem then becomes how to choose features and relationships such that the description can correctly and uniquely identify each possible character in the recognition set.

1.1.2.1 Slit/Stroke Analysis

One of the simplest methods of structural analysis is the slit/stroke analysis method which is an extension of the peephole method discussed previously. The peephole is expanded into a slit or window whereby it is not necessary to fix the slit at a specific position on the plane. The logical relationship between the two pixels can be extended to a general relationship between them, i.e. the feature of the slit can be determined by the number of black regions in it.

In 1961, Weeks used this idea for his recognition module where the binary image is scanned in four directions: vertical, horizontal and two orthogonal diagonals. For each direction, six equally spaced and parallel lines are used to cover an input character; then a crossing count (the number of times the line hits a black region) is made for each line. Thus, the input character is coded into a pattern of crossing counts which is used for

recognition based on statistical decision making. This method is very important for pattern recognition in general.

Johnson and Dimond used the same basic principle in 1957, except they used sonde instead of a slit that is simply given as a straight line. In this method, the researchers assumed that there were two basic points around which most numerals are written; thus, it is effective to span some sonde/slits centering the two points. The number of crossing times can be counted for each slit and this vector can be used to distinguish numerals.

The sonde method does have weaknesses in that it misses certain relevant features of the input character. However, either implementation of the stroke/analysis method can be considered as fairly effective when the number of categories of characters is limited and especially if the character shapes are pre-defined (the result of machine fonts).

1.1.2.2 Thinning Line Analysis

In this case, thinning is used not only as a pre-processing step but also as a method of abstracting lines from the input image. As noted previously, thinning is an imperfect process and information of the image can be lost. Given that, lines are abstracted from the image and used for analysis and recognition. Two methods are commonly employed here: detecting nodes/singular points, describing edges/arc. In both cases, 3x3 matrix is generally used. For example, in the case of describing arcs and edges a technique known as the Freeman code or chain encoding is employed. Here, a 3x3 matrix moves across a binary image until it meets a black pixel and then it follows the line. Eight local directions have been determined.

This method describes the shape locally and from this the image needs to be defined globally. Thinning line analysis is used in conjunction with some of the other structural recognition methods that will be discussed later.

1.1.2.3 Stream Following Analysis

This idea was first developed by Rabinow in 1962. The idea is basically to trace the curve of the input character by emulating the current of a stream. We can assume a coordinate system based on top to bottom or right to left. Then we start from a given side and use the slit method explained previously to extract information from the input. For example, if we move from right to left, we track the black regions found in each slit. So, if we find a black region that is not connected to any other black region, we establish a principle or a point to track. Then if the stream splits, we note the split and if parts of the stream reform, we note the union. At the end of the scanning process, a simple structure or shape of the segment has been determined using a vocabulary of principles, splits and unions.

The description of this method is fairly simple and because of this simplicity different shapes such as '+' and 'T' or 'U' and '□ ' can be classified as the same. Thus the stream-following method is generally complemented with some analysis of the geometric attributes. A stream-following algorithm developed by Hoshino and Miyamoto is the main algorithm in the first NEC postal code number reader.

Basically, all structural analysis methods use the overall structure of the character's bitmap to determine features. There are many different methods that have been formulated such as contour following analysis which basically follow the same

methodology. These methods tolerate slight differences in style and some deformation in character structure. However, it has been shown to be quite difficult to develop a guaranteed algorithm for recognition, instead heuristics are used. While these heuristics are somewhat tolerant, structural feature extraction has proven to be difficult in handwritten character recognition because of the significant variations in handwritten data.

1.1.3 Neural Networks

Most of the methods discussed previously were studied very early on in the respective history of OCR. The 1960s were a time in which much interest in character recognition was generated; the work at this time produced extremely successful results in the recognition of machine-printed characters, but similar success for handwritten data had not been achieved. The 1970s and most of the 1980s produced a relative drought of new information and research in the field of OCR. In 1986, the development of the back-propagation algorithm for multi-layer feed-forward neural networks was reported by Rumelhart, Hinton and Williams. Back-propagation learning for neural networks has emerged as the most popular learning algorithm for the training of multi-layer perceptrons. The popularity of the algorithm struck new life into the research in OCR. Neural networks have been considered seriously since the 1960s, however no solutions in how to use them productively came about until the 1980s. Reasons for this include the absence of computer or software technology until the 1980s and the lack of belief in and funding for neural networks. The development of the back-propagation algorithm meant that neural networks could be used for pattern classification. The advantage of using a

neural network for pattern classification is that it can construct nonlinear decision boundaries between the different classes in a non-parametric fashion, and thereby offer a practical method for solving highly complex pattern classification problems. Neural networks provide a superior method for the recognition of handwritten material. With neural networks, the distributed representation of features and other characteristics provides an increased fault tolerance and classification even when noise is present. Next, some of the research done in the area of neural networks and character recognition is presented.

As mentioned previously, the study of neural networks in the realm of character recognition was popularized in the late 1980s. One of the early models was designed at AT&T to recognize postal zip-codes. They used a Neocognitron model in conjunction with traditional back propagation. In their model, three hidden layers are used with local averaging and subsampling between layers. However, it has since been shown that the number of hidden layers does not affect the performance of a neural network greatly. Because of the time period, one of the major obstacles they faced was the length of the training process. To minimize this, they used local connections instead of a fully-connected network and they added constraints for weights. Given, this the training process still took 3 days; they achieved a recognition rate of 90% with 1% false positive.

At Nynex, they also used a version of the Neocognitron model where a rule-based system is used to complement the network output. In their model, efferent connections are determined not only by the maximum output detector but also by expected outputs and probabilistic update rules. These rules are based on a priori statistics such as the amount

of a customer's previous bill. Using a system to complement a given neural network architecture has become a common practice; these measures are taken mainly to reduce false positive rates which can be a salient issue in applications that cannot absorb error well.

Lee and Kim of Korea experimented with different neural net structures for recognition; they proposed a recurrent neural network based on the standard 3 layer feed forward multi layer perceptron (MLP). In this case, each output is connected with itself and all the other output nodes in addition to the standard feed-forward connections. The presumption is that this will provide more information to the output units in order to improve discrimination and generalization power in recognizing characters.

They tested their proposed neural net structure on a totally unconstrained numeral database; their results showed that the use of recurrence greatly improves the convergence speed of the network during training. The added information given to the output proves itself as useful as the network does indeed learn faster. With a training set of 4000 digits and a testing set of 2000 digits, they achieved a 97.3% correct recognition rate and no rejection class was taken.

In1995, researchers at SUNY Buffalo developed a system which a system which was able to separate hand-printed data from machine-printed data and then recognized characters using a neural network architecture. They distinguished between hand/machine font using 6 symbolic features: standard deviation of connected component widths and heights, average component density, aspect ratio, and distinct different height and widths. The network used to identify the handwritten data was a standard feed-forward MLP (296

inputs, 100 hidden units, 10 outputs) with feature extraction. They based their feature extraction on chain encoding (discussed previously). The recognition rate they achieved for recognizing hand-written zip-codes (5 digits) was 57.5 % (or 89% per digit) while for machine-printed zip-codes they achieved a rate of 95.5%. These results illustrate two things: 1. The recognition of hand-printed numerals as discussed previously is indeed much more complex than that of machine-printed data, 2. Even when the number of digits being recognized is known, simplifying the task of the segmentation module, total field recognition rate is still difficult and limited by per digit recognition.

In general, neural network techniques have been successfully used for handwritten data recognition. They perform better then traditional image processing techniques alone. Template matching is a technique most useful for machine-printed data and has been successfully employed in this area. However, handwritten data varies so widely that template matching becomes too complex and unreliable. While Fourier transforms have been used for this task with some success, the associated computation time involved is too large. Structural recognition modules face a similar problem. The structure of handwriting cannot be algorithmically quantified and computation time is limited. Therefore, neural network techniques have proved the most successful and reliable for handwriting recognition. Good results have been found in many different cases. However, no one neural network model appears to be inherently better than others to a significant extent. Instead, requisite accuracy rates are achieved by tailoring network models to a particular problem environment.

2. System Architecture

Automated recognition can be performed in two environments, online and offline. In the online case recognition is performed as the characters are being written off and hence dynamic information like stroke sequence, speed, pressure, and pen-up and / or pen-down positions can be utilized to overcome some of the difficulties created by the non-uniformity and connectivity of unconstrained handwriting. In offline recognition, only static information contained in the image of the numerical strings is available which makes the task that much more difficult.

The architecture for this system for offline reading of handwritten numerals consists of six modules and is discussed below.

Segmentation Module: The purpose of this module is to take a string of unconstrained handwritten numerals and divide them up into separate numerals.

Segmentation Critic Module: The purpose of this module is to take the list of connected components from the Segmentation Module and to validate them as well as transform them into a classified string which permits best recognition.

Preprocessing Module: This unit brings uniformity to the components and reduces variability in slant, thickness and size.

Neural Network Recognition Module: This module applies the trained neural net to determine what numeral the character is.

Post Processor Module: This module uses classification schemes to verify the results of the Neural Network Recongnition Module

3 Segmentation Module

3.1 Segmentation Overview

The process of segmentation merely involves the classification of the numeral string into its constituents. Since we are researching on *unconstrained* handwritten numerals, inevitably there will be some overlap between the numerals, or a numeral could be disjointed, complicating the task several degrees. Also we do not have information available of pen-up and pen-down positions since this we deal with off-line character recognition. Hence the only data available to us is static information contained in the image of the numeral strings.

The majority of research done at Sloan regarding unconstrained handwritten numerals had a very practical goal in mind; to automate the bank check processing method. Hence a good segmentation module was critical for the system's success. Lets explore the basic processes common to any system dedicated to handwritten numeral segmentation and the assumptions they are based on.

3.2 Connected Components

The numerical image is first modified so each pixel value is either black or white. The threshold used should be one appropriate to the intensity range of pixels in the image. The first step in the segmentation process would be to use search algorithms or any standard tree traversal algorithm to find all pixels attached to any random black pixel and hence extract out connected components. As each connected component is classified and

removed, the rest can be examined and similarly classified. Repeating this process classifies every pixel to a part of a connected component. Any pixel adjacent to another is deemed 'attached' to it for classification purposes.

If the digits in the numeric string were well spaced out, and not disjointed, every connected component would be a separate numeral and we could jump straight to the Preprocessing Module. Unfortunately this is seldom the case, and even more so in bank check courtesy amounts and hence this module was further refined to identify and separate the connected components into individual numerals.

The connected components can hence be classified into 3 categories; either a single character, more than one character, or part of a character. Using the aspect ratio and position of the component, it can usually be classified into one of the three categories. Here aspect ratio is described as the height to width ratio. For instance when the width is greater than a certain multiple of the height then the component usually does not comprise of single characters. That information alone is not sufficient though. For example a 1 could be connected with another numeral and the resulting connected component would have a height to width ratio similar to that numeral alone.

Hence this output is forwarded to the Segmentation Critic Module that checks its input in a vast number of ways including aspect ratios. After parsing the components it is recycled back to the Segmentation Module with some feedback. Hence the connected components are classified and now the Module can apply advance techniques to separated components that incorporate multiple characters. To simply this discussion, we will focus on the case where there are 2 connected components involved and need to be separated, although

with minimal adjustments the same heuristics can be extended to incorporate multiple characters.

3.3 Min Max Algorithms

The fastest way to separate a connected component of a pair of characters would be employ a technique which takes advantage of structural feature based methods. The premise here is that most characters that touch or overlap are joined at a point where the distance between the upper and lower contours is minimized. Hence the algorithm strategy is very simple and extremely fast to check before employing more advanced techniques.

The algorithm determines the upper and lower contour of the connected component. Then it determines the minimum point of the upper profile and maximum point of the lower profile. The range over which the profiles are scanned to find the minimum or the maximum does not extend over the full length of the profile. The joints usually occur near the center of the pair of connected characters in the component. Hence we must only scan a fraction of the length of on either side of the center for both the upper and lower contours. This way misleading minimum and maximum values can be largely eliminated.

After the determination of the minimum and maximum values of the upper and lower contours respectively, a straight-line path between them is picked and used to disconnect the connected component. The information along with the maximum, L, and minimum, U, values used is sent along to the Segmentation Critic Module for feedback. The Critic module checks for various conditions:

(i) L, the maximum point of the lower contour, must be less than or equal to U, the minimum for the upper contour.

(ii) The line that joins L and U must be horizontal or sloping upwards when viewed from L.

(iii) The line that determines the cut must not pass more than 2 contours or it is likely penetrating the body of the numeral.

(iv) The pieces separated must satisfy proper aspect ratio and other tests to detect character fragments and disjointed pieces.

In the case the above conditions are not satisfied, the Segmentation Critic Module returns the connected components to the Segmentation Module and more advance techniques are employed.

3.4 Hit and Deflect Strategy

One method of describing Hit and Deflect Strategies would be to say they weave through the connected component. Almost all implementations of these algorithms involve a set of rules to use to guide the counter weave through the component and hence split it into separate characters. The rule-set is critical in the correct determination of the cutting path. Hit and Deflect Strategies are most useful where a vertical scan line of the image reveals more than 2 transitions. In this case the characters are probably slanted and the Min Max algorithm would have failed. The cutting edge would develop by trying to stay in the space between the two characters for as long as possible and avoiding the intersection till

the last minutes. Hence it just ricochets off the walls along its path that gives it the name 'Hit and Deflect'.

Before the algorithm can be described, two terms must be defined.

Contoured component bitmap: A contoured bitmap is obtained by setting all the pixels inside the component, which are not on the contour to white, or zero. Hence we obtain a pseudo-hollow component that has an edge only one pixel thick from all sides.

Solid component bitmap: The regular connected component in it's unaltered state. Hence this one is not hollowed out.

The algorithm may start from a peak in the lower contour or a valley in the upper contour depending on which direction it chooses to move in. The point closest to the opposite side in vertical terms is preferred amongst the maximum and the minimum. Once the point is chosen and determined, the algorithm will proceed according to the following rules:

(i) Observe the solid component bitmap and if the pixel is not bordered by any black pixel proceed forward. The reasoning behind this is that you will keep moving the cutting edge forward till you actually encounter the component (either at the maximum or the minimum as you have previously determined)

(ii) Otherwise if the pixel in front is white, then proceed forward regardless of where the black pixel is located. Reason being you want to penetrate as deep as you want.

(iii) Otherwise if the pixel on left is white (note the front one must be black), move

to the left. This is the condition where the black pixel is on the left and we have deflected.

(iv) Otherwise if the pixel to the right is white, move right. This is the case where the front and left sides are blacked with black pixels.

(v) Otherwise move forward. In this case all sides are blocked with a black pixel and the cutting edge starts cutting the component.

In this case as well the output is passed on to the Segmentation Critic Module that does all varieties of checks and balances with aspect ratios etc to determine whether this is a suitable separation. In the case where the Hit and Deflect Strategy fails, a more advance technique is applied.

3.5 Hybrid Drop Falling Technique

Drop falling based algorithms are built on the principle that the connected components can be optimally separated if one were to drop a ball near the joint and let it 'roll' off till it stops and make the cut there. A very intuitive approach and one that is surprisingly robust. Though this approach was not all encompassing and there are lots of cases which form an exception to this technique.

This basic drop falling algorithm has been very aggressively modified into a hybrid approach which successfully segments almost every scenario out there, though it is computationally more intense than other methods mentioned above and is hence used as a last case scenario.

3.5.1 Traditional approach

One of the first decisions to be made is to where to start the drop-falling from. There are several methods available to determine that. It is intuitively obvious that the most beneficial starting point would be one closest to the point at which the two characters meet. The easiest way to calculate this point is to scan the image row-by-row until a until two adjacent black boundary pixels are detected. This pixel is used as the point from which to start the drop fall then.

Once the pixel has been located the drop fall begins. Since the drop fall mimics an actual fall the movement at every step must either be downwards or sideways if downward movement is blocked. Hence the algorithm continues to move in this fashion till it reaches the bottom at which point the cut is made vertically.

3.5.2 Variations

The standard variation falls down and to the right. It is obvious that there must be three other variations with which one can traverse the component as well. Firstly, the top-right drop fall, that initiates from the top right of the connected component instead of the top left and falls to the down and left. One can note that this drop fall is equivalent to the regular drop fall if the image is flipped vertically. Hence we note that we can perform this variation of the drop fall technique simply by vertically inverting the image and applying the standard drop down technique again.

Similarly the bottom left drop fall algorithm variation, which starts in the bottom left corner and climbs upwards and to the right can also be done via the standard technique

via a transformation to the image. In this case the transformation needed would be a horizontal flip and then the standard drop fall technique which starts from top left and travels down and to the right can be re-used.

Going by the same scheme used for the other variations, the bottom right drop fall variation can also be transformed. The bottom right drop fall technique starts in the bottom right corner and travels upwards and to the left. In this case the image would be an exact opposite for it to be used by the standard technique. This can be achieved by applying both a horizontal and a vertical flip to the image and then the standard drop fall technique mirrors the process of a bottom right drop fall variation.

The advantage of all these different variations is that while the standard drop fall scheme might encounter various connected component images where it is unable to separate the characters effectively, it is very hard to come up with a connected component of two numerals, which cannot be separated effectively using against all of these techniques. By flipping the image vertically, horizontally or both, we can manipulate the situation to allow us to make the drop fall technique successful, all the time using the same rule-set we wrote for the original standard drop fall technique.

Using the fact that it is very hard to come up with a connected component string on which none of the 4 different drop fall techniques work well, we start building out next variant. The hybrid is so called because it tries to harness the power of all four variations.

3.5.3 Hybrid drop fall algorithm

Based on the evaluation of the four variants described above, it is clear that combined

together they form an extremely powerful solution for segmentation of these connected components. Considering that the standard top left based variation and the bottom right based variation are opposite to each other, it seems appropriate to harness the power of these two variations as they will together encompass all scenarios that the drop fall variations could possibly be successful at (pretty much all legible numeral scripts).

The algorithm starts off by attempting a top left drop fall on the image and segments it in some manner and sends the results to the Segmentation Critic Module. The Critic Module would again examine the results very thoroughly and give feedback the this Module on the success or failure of the segmentation. If the segmentation fails the process takes advantage of the knowledge of the cases where top right drop fall techniques generally fail.

The top left drop fall variation fails when the second character has a local minima in its lower stroke. This is especially true for all 3s and 5s. One way to establish this is to check the gradient at the local minima of the upper contour. The minima of the lower strokes of 3s and 5s tends to have a very gradual gradient associated with it while regular joints tend to have very sharp gradients.

Hence it follows that if the joint or the minima is very sharp and steep, then top left approach will almost always yield the desired results. On the other hand a gradual local minima on the lower stroke will result in failure for the top left drop fall technique. The Segmentation Critic Module will easily confirm the result. In such a case we can apply the bottom right drop fall off by inverting the image both vertically and horizontally and applying the standard drop fall technique.

To enhance the process and minimize computation one must not infer that just by examining the gradient of the local minima is sufficient to determine which of the two variations would be needed. The actually computation of the gradient requires traveling along the contours of the characters 5 pixels to the left or the right and noting down the how much the contour rises. After a certain threshold is crossed, the gradient is then declared steep and sharp. From lots of testing it has been deduced that a 0.5 threshold is sufficient to separate the steep gradient from the gradual.

4. Pre-Processing Module.

4.1 Thinning and Re-thickening

As has been previously noted, pattern recognition and image processing applications frequently deal with raw inputs that contain lines of different thickness. Though this variation might be beneficial in some cases, it is detrimental to our cause in most cases. It's been noted on numerous occasions that the degree of variability in the thickness of the lines in handwritten characters directly influences the probability of success and infact decreases it by a significant margin; even more so when the method of choice is neural network based recognition.

Hence algorithms have been introduced in this module that would transform the components to a uniform thickness for increased recognition. Thinning and thickening are complimentary processes, which make the lines of each segment a uniform width. Basically, in order to accomplish a final uniform width, the digits needs to be skeletonized to account for varying widths in the initial segment.

Hence thinning reduces the pattern to a thickness of one pixel while re-thickening increases it back to a constant thickness. It is shown that the basic structure and connectivity of the underlying complexity can be preserved through the stages of thinning and re-thickening.

4.1.1 Thinning Algorithm

Generally three types of thinning algorithms have been proposed and utilized which

include serial, parallel and maximum methods. The algorithm utilized here provides an excellent parallel based thinning algorithm. It utilizes the following functions listed below.

(i) Initial: This function computes the contour loop number and the first contour point at its previous point for each contour loop. It also sets the initial value 1 to loop decision variable pointing towards the contour loops points to be thinned.

(ii) Loop-end-test: A function to test whether the k^{th} contour loop has been terminated.

(iii) Contour: This function tests whether a point is lying on a contour or not.

(iv) Successor: This function determines the successor point from previous point according to a clock-wise order around the neighbours to meet the first "on" pixel.

(v) Deletion: A function that deletes a point based on certain conditions, based on a clockwise walk around the neighbour.

The actual algorithm is omitted here and readers are encouraged to look for [1]. The algorithm is incredibly fast and is very readily programmable.

4.1.2 Re-thickening

This algorithm is incredibly simple and once the input data has been skeletonized and

reduced to a single pixel thickness, this algorithm merely looks at a prespecified adjacent neighbour of each filled in pixel, and fills it in. Hence the skeleton is easily transformed to a uniform thickness. This method yields better results in the Recognition Module and is currently not in use anywhere else. Most researches merely skeletonize the image and do not re-thicken it to gain superior results.

4.2 Normalization

This is the process of resizing the bitmaps so they are all of a uniform size. Although this will seemingly affect the thickness as the image is blown up, in actuality the image are almost always reduced in size. In most cases from a typical 30x50 pixel size, the image is reduced to, for instance, 16x16 pixels.

4.3 Slant Correction

Slant correction is intended to straighten a character so that its main vertical components stand perpendicular to its baseline and its main horizontal components lie parallel to its baseline. This method is employed because the recognition methods used can be sensitive to the pitch of a character and may fail if the image is tilted despite having a clear form. Slant numerals can often cause problems for the Recognition Module since in neural net based algorithms, less variance will result in increased accuracies and vice versa.

Slant correction used to almost completely rely upon moment normalization techniques. But the techniques used in this system far surpass those. The notation used here states that a slant of y degrees from the vertical position in the anti-clockwise direction is considered a slant of $-y$ and vice versa. The function for the bitmap transformation through an angle

y is as follows:

For all (x,y) such that x,y is a "on" pixel (black pixel) in the input bit map, intrpduce a new pixel where

$x' = x - y * tan\ (y)$

$y' = y$

Using the above function, the slant algorithm is listed below.

(i) Initialize y to 45 degrees and "active" bit map to input bit map.

(ii) When angle $y > 1$, do steps (iii) and (iv)

(iii) Let the "active" bit map be B1 and transform it through –y and +y using the bitmap transformation function described above.

(iv) Let the minimum width bit map amongst the three bit maps, B1, B2, B3, be the new "active" bit map.

The gradient decent features of this algorithm are vastly different to those discussed elsewhere. For instance the step sizes on the error surface while performing the descent are much larger initially than anything used elsewhere and are then geometrically reduced. This, amongst numerous other improvements, increases running times significantly.

5 Neural Network Recognition Module

5.1 Overview

The recognition employed in this system forms an efficient and accurate module for segmented digits. Since the research at MIT focused on automating bank check processing and detecting the courtesy amounts in particular, they started the Recognition Module with a pre-processing unit. The purpose of the pre-processing step was to structurally recognize commas and minuses. It also segments components that can be interpreted as noise or not valuable to recognition and are henceforth discarded.

Following this, the segment is input to the neural network recognition module. This section is the foundation of the recognition module. To improve the systems' generalization power, a lot of features are extracted from it and process before this is sent along to various neural networks. Then the input, represented by the features extracted from it, is passed to an array of neural networks. Each network has been trained to match the result to a particular numeral. As the input arrives all neural networks produce an output and an associated confidence value for its output value. The arbiter takes this information from each network and produces an output for the segment. The arbiter can either decide that the confidences output are not adequate to decide on a positive value and output that the segment has not been recognized or it can positively assign a value for the segment. If the segment is not recognized, the feedback module assumes that the input was not a recognizable character and moves to re-segment the input and sends it back to the Segmentation Module.

If the segment is accepted, the output, which contains the recognized value, is sent to the Post Processor Module. This Module looks to verify the value that was recognized and in doing so aims to reduce or possibly eliminate the false positive rate. One can say that the module attempts to make sense of the structure given its recognized value.

5.2 The Neural Network

The main recognizers used in this application are all neural networks. Research in this area has indicated that neural networks produce reliable results for character recognition. Techniques such as template matching have been ruled out because of their inefficiency in handling handwritten data as well as their inability to handle a wide variety of inputs. Structural models are too simplistic and do not provide adequate results as a main recognition module. Neural networks provide the most viable option in terms of both reliability and computation time. The advantage of using a neural network for character recognition is that it can construct nonlinear decision boundaries between the different classes in a non-parametric fashion, and thereby offer a practical method for solving highly complex pattern classification problems. Furthermore, the distributed representation of the input's features in the network provides an increased fault tolerance in recognition; thus character classification can occur successfully when part of the input is broken off and not present in the image, as well as when extra input signals are present as a result of noise. This is a very important characteristic for a recognition module in this application.

Remember, the initial input into the application is an image file of the unconstrained numeric string that is parsed through a Segmentation Module, a Segmentation Critic

Module and a Pre-processing Module, before this module receives its input. The Recognition Modules is dependent on the earlier Modules to have a high quality output with clear well segmented numerals who have been thinned, re-thickened, normalized and have had their slant removed. Unfortunately any one of the other Modules might cause an exception and produce an error. This unconstrained environment for the Recognition Module can cause the precise cases noted above; namely, noisy images and inputs with broken structure or additional pieces added on. Because the Recognition Module needs to absorb such variations in the input, the logical choice for successful character recognition is a neural network-based architecture.

There is a vast variety of neural structures that can be employed. The research at MIT utilized considered and utilized multi-layer perceptron networks, neocognitron networks and recurrent neural networks. Other studies, later discussed, have successfully employed other forms of strategies as well. In all of these cases, the recognition rates were relatively similar and all required additional infrastructure to achieve the requisite recognition rates to achieve their goals. Research has indicated that no one neural network model is inherently better than others to a large extent, but recognition is improved by tailoring the network to its application environment.

5.2.1 The Multi Layer Perceptron

The model we will discuss here is a three-layer, fully connected, feed-forward multi-layer perceptron. The MLP structure is the simplest to understand and to implement. As the name suggests, an MLP is a network of perceptrons or neurons. Each neuron takes in inputs, processes it and produces an output. The neuron takes a set of inputs where each

input has a given weight attached to it. The combination of input*weight pairs are summed and added to the bias of the node. This total is normalized to a final node output between 0 and 1.

Layers organize this network of neurons. The nodes of the input layer represent the pixels of the input bitmap or features extracted thereof. In the hidden layer, 100 nodes are used. To refresh, the hidden layer is used to extract increasingly meaningful patterns from the input data. The number 100 can be experimented with; general trial and error with the number of nodes in this layer has shown that 100 is a number that can give accurate results while not consuming excessive amounts of computation time. Too few nodes do not allow the non-linear features of the input map to be interpreted properly by the network and too many increase computation time beyond applicable use.

Another important feature to note about the network is the use of only one hidden layer. Recent studies have indicated that the number of hidden layers is generally not a factor in the performance of the network. In fact, it has been rigorously proved that the performance achieved with any number of hidden layers can be matched with one layer of hidden nodes. Therefore, the more important factor is the number of nodes and the amount of computation that they can accomplish.

In the output layer, we take advantage of the fact that final values range from 0 to 1. Given this, we can interpret each output of a node in the output layer as a confidence level for each possible output. So, each node in the output layer represents a possible digit or symbol being recognized. The output layer is set to recognize 10 possible outputs: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, and NOT_RECOGNIZED. The NOT_RECOGNIZED output simply

represents segments that are not identifiable as any of the other possible outputs. This option exists for the scenario that the unit was somehow not properly segmented and slipped through or is just too illegible.

These components are then returned to the Segmentation Module for re-segmentation. Some feedback will also be provided if the Recognize Module is able to identify to with some certainity the presence of two characters or the presence of a disjointed character. This feedback will also be extremely useful for the Segmentation Module in its attempt to re-segment the component.

Lastly, the MLP implemented is fully-connected which means that every node in a given layer feed its output to every node in the following layer, i.e. the input layer is fully connected to the hidden layer and the hidden layer is fully connected to the output layer. Also, local connections allow the network to extract features from a given section of the network and recombine the traits in another layer. However, as mentioned earlier, it has since been uncovered that performance of any number of hidden layers is comparable to one layer with a certain number of nodes. Using a fully-connected network allows the network to extract meaningful information from the input map and allows the network to derive distributed features by using knowledge derived from the entire input map. Therefore, this implementation uses a fully-connected network.

5.2.2 Training the network

Given that inputs to the network can vary as widely as 2^n, the success of neural networks is largely dependent on the appropriateness of the weights that each nodes uses. The

networks can be trained with a set of inputs which will set the weights, the free variables, using the knowledge that when the actual value of a segment is known then the desired output for that value is 1 and all the other outputs should be 0. The standard back-propagation algorithm is perfect for this sort of task.

The back-propagation algorithm uses input patterns whose classifications are known. It passes these patterns through the network and measures the error at each output node based on the known value. The error is then propagated backward through the network and used to adjust the free parameters of the network, or the weights of the nodes. The weights of the network represent plausible microinferences which, when applied to the input, reveal patterns that satisfy the largest number of these constraints and contradict the least. Back-propagation is the most popular and the most successfully employed training algorithm. Thus, this algorithm was used to train the networks presented here.

The neural network is designed to have a large generalization power; however, if the input it sees is substantially different from the input it has been trained on, the network does not stand a fighting chance of interpreting the digits correctly. For this reason, standard databases such as the NIST database which is commonly used for network training and carries more than 10,000 digits (obtained in the US) have not been used.

Secondly, we have used digits outputted by the system's own Segmentation Module because of the nature of the outputs produced. The digits obtained from standard databases are pre-segmented and therefore are not susceptible to corruption by fragments etc from other digits. Using digits outputted from the application's own Segmentation Module enables the Recognition Module to acclimate to the kind of output it will receive

in general use. The Segmentation Module is a rule-based system so it's output is consistent and can be predicted well if the neural network is tuned to it. As suggested above, the Segmentation Module cannot always do a perfect job; often times, when digits that were connected in the original image are separated, pieces of the digits are in the incorrect portion of the partition.

The network will be absorb such errors if it can be trained to expect them, and thus a high recognition rate can be achieved which is essential to a useful system. The feedback module counts on the neural network to correctly identify and reject connected and separated digits so that it can correctly segment that block of the image. Therefore, training on data retrieved is nearly essential for identifying some of the patterns that are outputted by the segmentation module.

Research carried out at MIT utilized a Brazilian check database to train the network for eventually identifying the courtesy amounts on Brazilian checks. For these reasons, data retrieved from Brazilian data were included in both the training set and the testing test. One of the main problems with collecting segments for the application was balancing the number of segments used for each possible value. Because digits were gathered from checks, there were inevitably predilections to certain digits that appear more commonly on checks, i.e. 1 and 0, while some values appeared very infrequently. When the networks are trained with a greatly imbalanced number of segments for values, the performance is greatly degraded. This problem was unique to research being carried out at MIT. At other institutes the data used to train networks included all digits in a proportionate ratio.

The input patterns in the training set are presented to the networks in pseudorandom order, i.e. training files with the exact same set of randomly ordered data were set in a random order and this array of files were presented in succession to the networks. The randomness is another factor which allows the network to keep its generalization power;

5.2.3 The Array of Networks

The module designed for this application employs the neural network architecture described above in an array. As can be inferred, the use of an array increases the generalization power of the network. Given one training set and two identically structured networks with inputs set identically, the weights for the networks after training will not be the same. Thus, this system employs an array of four neural networks for the recognition of the input bitmaps. This improves recognition rate while also reducing false positive rate by employing redundancy.

To explain the reason for different outputs, prior to training, the network is set with randomly chosen weights. Therefore, when the network is presented with known inputs, the amount of change on the weights is not uniform. Furthermore, the networks are presented with inputs in pseudo-random order that also contributes to the varied results of each network.

5.2.4 Feature Extraction

MIT research indicated that running multiple classification systems in parallel permits for simultaneous design and production of several modules independently, facilitates expandability and most importantly, increases accuracy of classifications. Given this, the

recognition module has not only been organized with an array of networks, but some of the networks employ feature extraction. To be specific, two of the neural networks do not use feature extraction and the other two networks employ feature extraction based on directional distance distribution (DDD). The various classification modules are inherently biased to different types of inputs; combining the systems allows for an increase an accuracy.

The performance of a feature-based neural network depends heavily on what features are being used. In fact, the literature is abundant with various features used in recognition modules and many of these options were discussed in the history section of this paper. In selecting a feature, certain criteria must be employed. First, the feature should incorporate a generalization of the input map, so that different representations of the same feature can be properly identified. Secondly, it should incorporate a discriminating power so that it aids in differentiating between different characters. Finally, computation time must be taken into account, and the feature chosen for use should meet demands time-defined constraints.

One of the features considered, but not used in the final recognition module is a mesh-based feature. Basically, the input pattern is normalized into an 8x8 mesh. The sum of the pixel values in each 2x2 block of the input map (16x16) is calculated and used as input into the network. The strengths of this feature include its simplicity and its generalization power. Clearly, calculating the mesh is not computationally intensive. Plus, it reduces the number of required input nodes from 256 to 64- a significant reduction in computation time. It generalizes the input by associating local regions with one feature. After testing

this feature, it was concluded that while it did have its merits, the overall feature did not improve recognition rates beyond that of the non-feature based network. Mainly, the number of input nodes was reduced, which limited the computation and analysis power of the neural network; thus the networks differentiating capability was limited. Therefore, the mesh feature was not incorporated into the final recognition module.

The system developed here is based on the distance information computed for both black and white pixels in 8 directions. The image is regarded as an array that wraps around which can achieve a better discriminating power for the feature. This feature has been chosen for its generalization power and also for its simplicity. The computation time of the extraction algorithm is relatively short since its dominant operations are integer comparison and addition.

To define the features of an input, 2 sets of 8-integer arrays are assigned to each pixel. One array, W, represents the distance to the nearest black pixel and the other, B, represents the distance to the nearest white pixel. The eight slots represent each of the eight possible directions to search in. For a black pixel, the next white pixel in each of the eight possible directions is searched for, and vice versa in the case of a white pixel. The number of pixels searched to find the appropriate pixel is entered into the appropriate slot. The search regards the input as circular and can wraparound. If the goal pixel value is not found, the maximum dimension of the array is recorded.

Then, the (16x16) input map is divided into 4x4 grids. One W array and one B array is used to average the values calculated for each pixel. This averaged 16 element W and B array represent the 4x4 grid and thus 16 inputs into the neural network.

The DDD feature has, in fact, proved to increase recognition results in this system especially when used as an element of the neural network array. The DDD based networks and the standard MLP network can complement each other in different situations and work to correct each other's mistakes. The DDD feature incorporates generalization into its representation by averaging the W and B arrays computed for each pixel over 4x4 subsections in the input bitmap. At the same time, it increases discriminating power through the actual calculations of distance to black and white pixels. Intuitively, it can be seen that the search for pixels in each direction provides the networks with a much different input representation to work with than that of the simple input bitmap.

Also, since the algorithm employed regards the array as being circular, the input distributions to the networks are affected by pixels throughout the input bitmap. Further, the DDD feature combines information about the black/white distribution as well as the direction distance distribution of pixels in the input bitmap. Preliminary tests using the DDD feature indicated this finding as well. Finally, the computation of the DDD feature also meets the need of the application, since it is not very computationally intensive. Most of the work done is simple integer comparison and addition. Thus, the DDD meets the three criteria that have been set for appropriate feature selection and has been employed in our system.

5.3 Decision maker

The final module in the neural network architecture is the Decision Maker hereby referred to as the Arbiter. The arbiter takes the results of each network in the network array and

determines an output. Basically, it cannot simply take the highest average confidence and return that as the value. In order to minimize the false positive rate, which is an important concern in this check-reading application, the arbiter checks for a high confidence in a prospective value. It will not accept a value if it does not have a minimal confidence value. Secondly, it compares the highest confidence value with the other confidences. When the highest confidence is greater than a certain pre-set threshold, and the second highest confidence is within a pre-set difference of the highest confidence, the highest confidence value is sent to the special 'Two digit processor' and the digit is assigned as not recognized. When the confidences are comparable, this obviously indicates the networks are not highly confident in the selected possible value. The function of the arbiter is fairly simple; its main goal is to assign an output value with certainty as high as possible.

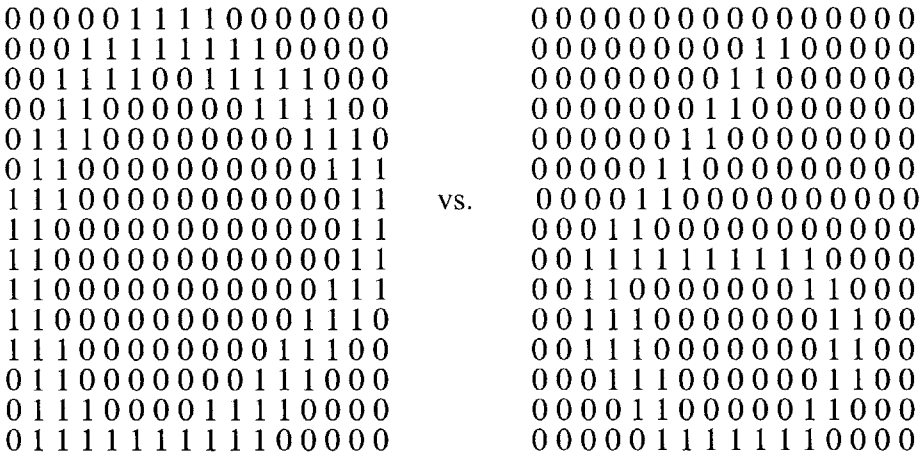
5.4 Two Digit processor

After the neural net has produced confidence levels for all possible digits, the confidence levels are examined to see if any reach the required tolerance level. In the special case where the confidence levels are high for two potential digits, this special post-processing algorithm examines the image component and determines which of the two digits is the actual digit. For the purposes of this explanation assume the segment components are 16 pixels tall and 1 pixels wide.

Type 1: Checks for Loops

The Type 1 algorithm checks for loops in the digit. For example, the difference between

a 0 and a 6 is that on the top half of the 6, there is no loop. So the algorithm scans the first 6 rows of the bitmap and checks how many times the bits are flipped. The bitmap for the fourth row of each digit would look like this: 0000000110000000 for the 6 and 0011000000111100 for the zero. In the case of the 6, we have a string of zeros, followed by 2 ones, and then back to a string of zeros, which results in two distinct bit flip cases (0->1 and 1->0), while for the 0, there are four bit flip cases. As soon as the bit flip counter for a row hits 3, then there is a loop in that row. If there are more than 2 loops counted, the algorithm returns the number that should have a loop in the area checked. Otherwise, it returns the number that should not have a loop.

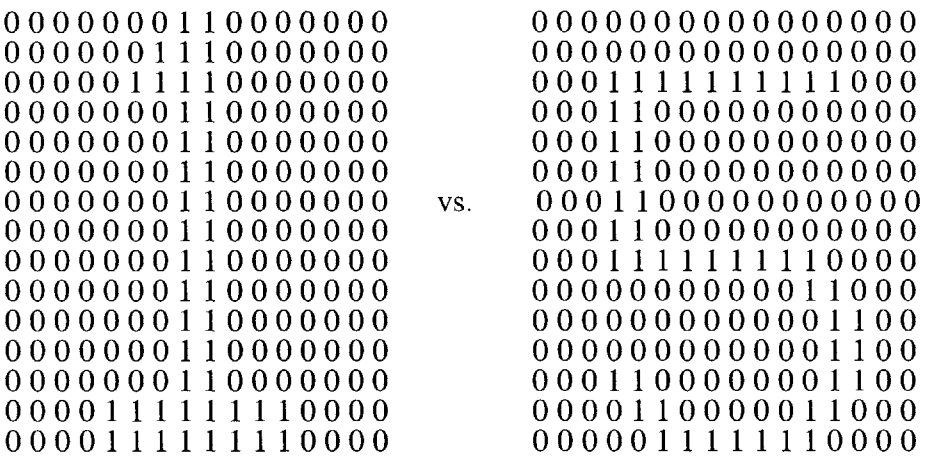


0 vs. 6 case

The Type 1 algorithm can be further divided into three different categories, A, B, and C, depending on where the algorithm is checking for loops. Type 1-A checks the bottom six rows, while 1-B checks the middle six rows, and finally 1-C checks the first six rows.

Type 2: Checks for 1's

The second algorithm is a relatively simple. It is generally used for checking between a 1 and another number. Since a 1 is usually relatively thin, there are usually few pixels that are on at the outside columns. The algorithm runs through the first and last five columns and keeps track of the number of on pixels. If that sum is more than six, then it most likely is not a 1 and is instead the other number.



1 vs. 5 case

In the example above, the 1 has four pixels on in the outside columns, while the 5 has 33 pixels on in the outside columns. Originally, the algorithm returned a 1 if there were less than two on pixels on the outside columns. However, this would not handle the case of the 1 shown above because of the horizontal line at the bottom. Testing showed that this was correct about half the time (6 out of 12) at high thresholds. An example of the Two Digit processor being correct is as follows. If the actual number is a 1, and the neural network's highest confidence was 5 and its level is above the threshold. Then if its second highest was 1 (and its level is within the difference), and the Two Digit processor checks this case and returns a 1 instead, then it was correct. Conversely, if the neural network's output was correct but then the Two Digit processor returns the second highest

confidence output, then it is said to be wrong.

Type 3: Checks 0 and 8

The third algorithm deals specifically with the case of checking between a 0 and an 8. Since a 0 is much like an 8 except that its middle is hollow, the algorithm checks a 3x3 square in the middle of the bitmap. If the number of on pixels in the square is greater than 2, the algorithm returns an 8. Otherwise, the square is relatively empty and it returns a 0.

Type 4: Checks 2 and 6

The third algorithm deals specifically with the case of checking between a 2 and a 6. This algorithm works on the premise that the on pixels at the midsection of a 2 are located towards the right, while for a 6, they are towards the left. It basically compares the number of pixels that are on from the left side to the ones from the right side. If there are more pixels that are on for the left, than it returns a 2. If there are more on pixels on the right, a 6 is returned. Otherwise, if the numbers of on pixels are the same for left and right, a NOT_IDENTIFIED is returned to indicate that the post processing could not determine whether it was a 2 or a 6.

Type 5: Checks 3 and 5

The third algorithm deals checks between a 3 and a 5 by going from the 4^{th} row to the 7^{th} row and summing the location (column), which a pixel appears on. It then divides by the number of on pixels encountered to determine the average of the location of those pixels.

If the average on pixel's location is below 8.5, then it is towards the left, and returns a 5. This is because of the tendency of the top middle segment of a 5 to be towards the left. Otherwise, if the average is above or equal to a 7.5, then it returns a 3 because the top middle segments of a 3 is usually on the right side.

Type 6: Checks 3 and 6

This algorithm checks between a 3 and a 6. It works on the premise that a 6 has a curve on the left side, so the first on pixel of each row would be relatively close to the left edge, while a 3's first on pixel for its middle rows would be farther to the right. The algorithm goes through the 9^{th}, 10^{th}, and 11^{th} row to find the first on pixels on those rows. Then, it averages the location of the two rightmost on pixels in those three rows and returns a 3 if it is greater than 5 and 6 otherwise.

Type 7: Checks 3 and 8

The Type 7 algorithm deals specifically with the case of checking between a 3 and an 8. It operates on the assumption that a 3 is like an 8 except that it is missing the left curves in the middle rows. Therefore, the first on pixels for those rows are likely to be farther away. The fourth, fifth, tenth, and eleventh rows that are checked, and the average of the first on pixels of those rows are calculated. If the average is below 5.5, then it returns an 8 because it means that it is relatively close to the left edge. Otherwise, it returns a 3 due to the lack of the left curve causing the first on pixel to be farther out.

Type 8: Checks 3 and 9

The eighth algorithm deals specifically with the case of checking between a 3 and a 9. It works much like the Type 7 algorithm except that it checks only the fourth and fifth rows, since a 9 only has the left curve on the top half. After determining the average of the first on pixels in those rows, it returns a 9 if it is below 5.5 and a 3 otherwise.

Type 9: Checks 4 and 9

This algorithm deals specifically with the case of checking between a 4 and a 9. It does so by exploiting the fact that the top part of a 9 is closer to edge than it would be for a 4. The 4 has a peak in the middle really close to the top edge but then it slopes downwards on the left side. In this algorithm, four middle columns (7^{th}, 8^{th}, 9^{th}, and 10^{th}) are examined to find the location of the first on pixel in each column. The average of the four locations is computed, and if it is greater than 2.5, then a 4 is returned since it is not very close to the top. Otherwise, a 9 is returned since those pixels are pretty close to the top.

Type 10: Checks 5 and 6

The tenth algorithm deals specifically with the case of checking between a 5 and a 6. It works on the premise that 5's are missing the left curve on the bottom half that 6's have. So, this algorithm calculates the average column of the first on pixels from the ninth to thirteenth rows, where the missing curve would most likely be. If the average is less than or equal to 4, then it returns a 6. If the average is greater than or equal to 5, then it returns a 5. If it is between 4 and 6, it checks the location of the latest first on pixel in those rows. If that is greater than 6, then a 5 is returned since it is far away from the left edge.

Otherwise, the function returns a 6.

Type 11: Checks 5 and 8

The Type 11 algorithm deals specifically with the case of checking between a 5 and an 8. It assumes that from the fourth row to the sixth row, the last on pixel will be close to the right border for an 8 and close to the left border for a 5. If the average in these rows is less than 7, then it returns a 5. If it is greater than 9, then it an 8 is returned. If it is between 7 and 10, then the next assumption is checked. The first on pixel from the ninth row to the twelfth row would be closer to the right border for a 5 and for an 8, the left border. The average of the first on pixels in these rows is calculated and if it is less than 7, then an 8 is returned. Otherwise, a 5 is returned.

Type 12: Checks 5 and 9

This algorithm works on checking between a 5 and a 9. First, it obtains the first and last on pixels from the third row to the seventh row. Then, it computes two averages, one for the last on pixels (last_avg), and another for the difference between the last and first on pixels (diff_avg). Then, if last_avg is less than 8, then it returns a 5 because 5's have a vertical line on the left side. Otherwise, if last_avg is greater than or equal to 11, then it returns a 9 since 9's have a right curve.

Further checking is done in the cases that last_avg is less than 11 and greater than or equal to 8. This time, diff_avg is checked. Since a 5 will only have a vertical line, the width should be relatively small, meaning diff_avg is relatively small. For a 9, however, it would be bigger since it would have a circle loop on its top half. So, if diff_avg is less

than or equal to 3, a 5 is returned and if it is greater than or equal to 6, a 9 is returned. In the case in between, even more checking must be done.

Finally, the last check done on the image to determine between a 5 and a 9. It first calculates the number of last on pixels that occurred in the left half (before column 9, called last_half) and also the number of last on pixels that occurred in the leftmost quarter (before column 5, called last_quarter) in the five rows checked earlier. If last_half is greater than or equal to three, then it is a 5 because a 9 usually has its last on pixels on the right side. If last_quarter is greater than or equal to two, then it is a 5 for the same reason as above. Otherwise, if last_quarter is less than two, then a 9 is returned.

Type 13: Checks 7 and 9

The thirteenth and final algorithm deals specifically with the case of checking between a 7 and a 9. It does so by going three diagonal lines. First, from the position (10, 1) to (1, 10) and subsequently from the (11, 1) to (1, 11) and from (12, 1) to (1, 12). Along each line the number of pixels until the first on pixel is added to the total. Since a 7 will not have anything until the end of the line because of its shape, the location of the on pixels will be high. For a 9, however, it will be different, as the lower end of the loop will be hit relatively early. So, if the total is greater than 16, which would be relatively late, a 7 is returned. Otherwise, a 9 is returned.

Similarly all pairs were matched up and using these techniques one can determine of the two digits with the highest confidence level, which one of them is actually the ball. The performance of the post processing step was calculated in the following way. First, the

performance of the system without post processing was obtained for various thresholds for confidence levels. Then, the performance was calculated for the system that will activate the post processing for various thresholds and difference thresholds. For example, if the threshold is 0.7 and the difference threshold is .2, then the post processor will be activated if the highest confidence returned by the neural net is greater than 0.7 and the second highest confidence is within 0.2 of the highest.

Results from the test showed that the addition of the Two Digit processing step increased the overall accuracy for all the different threshold and difference levels except for the 0.9 and 0.1 case. The trend from the results is that the added value (increase in accuracy rate) of post processing decreases as the threshold level increases and it increases as the difference level increases.

6 Post-Processing Module

The structural post-processing module handles segments that have been recognized as digits. The segments that have been identified as connected or have not been recognized are sent back to the feedback module for re-segmentation. The role of the structural post processor is then to ensure that the neural network architecture has done a proper job of recognition. A structural system is an ideal candidate for post-recognition. Structural analysis is too computationally intensive as the main recognition module, and the variations between written versions of the same alphanumeric character can vary too widely for a structural system to handle. However, as a post-processing module, it serves as a very good verifier because the needed computation time is reduced and the module has an idea of what it is looking at so it can use simple guidelines to look for a few characteristics instead of a great many. Therefore, the post-processor accomplishes this task by associating the recognized value to general heuristics about that value and compares this to the structure of the input.

The structural post processor is used when the confidence of the assigned value to the digit is less than 90%. When the neural network module is very sure of its output, the processor does not need to be employed. This practice simply saves computation since the network is generally correct when its confidence is very high. The structural processor implemented for this application does not take a recognized value for a digit and switch it to another value. Instead, if it finds that the segment's structure does not match the assigned value, it switches the value to not-recognized. Then this segment can be passed back to the feedback module for re-segmentation and re-recognition.

One must remember that if the structural post-processor could accomplish the task of recognizing numbers accurately and reliably, there would not be a need for the neural network architecture. The reason that structural classifiers are not used as the primary recognizer is because of the difficulty in setting a standard for handwritten digits that can then be used to identify these digits. Therefore, this application avoids giving the task of assigning a recognized value to the post-processor. Instead, the post-processor uses loose or general guidelines that determine the basic structure of a value and uses these to verify the assigned value. If the input signal does not meet this criterion, the segment is then switched to not-recognized. In this sense, the neural network architecture is given the benefit of the doubt in its recognition.

Furthermore, the option of the feedback module is available. In other applications, rejection of a value translates to rejection of an entire amount which can severely reduce recognition rate. However, this system uses re-segmentation and re-recognition to improve its correct recognition rate while minimizing the false positive rate. Thus, through feedback, the Recognition Module has the luxury of re-using the network architecture for recognition, instead of a heavy reliance on the post-processor.

As mentioned earlier, the post-processor uses various heuristics to verify values. For example, it employs loop detection to look for loops where there should be loops, i.e. 6,8,9,0. It can also identify open loops such as those found in '3' or '5'. Remember, the network architecture is designed to absorb errors in the input image such as broken strokes because of segment separation or other reasons; this can cause a number such as 5 to be recognized as a 6 or vice versa. Their structures are similar, however the 5 has an

open loop on the bottom where the 6 has a closed loop. Using loop detection on a number such as 5 can indicate whether the digit has a closed loop or not. If the '5' has a closed loop it can safely be rejected. This is very similar in nature to the Two Digit verification system.

Other heuristics are used for verification; the module will look for the width of a character in appropriate regions of the segment and it examine the position of lines within the segment. The system employs a unique approach for structural verification derived from the known and discussed methods. The following section will discuss the loop detection method employed and its use as well as show how the heuristics have been applied to different digits.

6.1. Loop Detection

The loop detection method derived is based on counting the number of strokes per line (recall the slit analysis method explained earlier). To refresh, a stroke is defined as a region where all connected pixels are black. The loop detection module is given a row from which to start looking for the loop. By counting strokes per line, this algorithm takes into account the common occurrence of open loops.

The loop detection algorithm also tolerates broken stroke- important for handling connected digits that have been segmented. Furthermore, it allows the user to decide how much of a gap in the loop to allow by setting a value for k; thus this amount can be adjusted from digit to digit. When methods such as stream-following analysis or other

contour detection algorithms are used, only fully closed loops are recognized which is an unrealistic constraint to put on the processor while still achieving high recognition.

The loop detector returns the row numbers denoting the start of the loop and the end of the loop. These bounds are used to determine the size and location of the loop, which are then used to ensure that the segment is indeed the recognized value. For example, if a '6' is assigned to an input and the loop detector finds a hole in the top half of the segment, the post-processor clearly should reject the assigned value to avoid error.

The loop detector can also be used with digits that are intended to have open loops such as '2','3','5', and '7'. Here the loop detector again finds the loop, but this time it also checks the loop to determine if it is an open loop. The algorithm used here is fairly intuitive. For a loop that should have the top opened, such as that given in the figure below, we begin with the loop detecting method described previously. From this, we obtain the top of the loop, where there is only one stroke.

By the loop detection algorithm, it is guaranteed that the next row has more than one stroke in it. Therefore, each stroke on the next row is noted and enumerated. If the stroke in the top row of the loop is connected to two or more of the strokes in the next row, and then the loop is considered closed, otherwise it is open.

Once these algorithms are in place, simple calculations can be done to determine the locations of lines and the widths of the inputs at different positions within the input image. These calculations are derived from the slit analysis discussed previously. For example, in the case of a '4', the line on the bottom can be detected, first by detecting the

loop in the top of the four and then number of strokes below can be tallied along with the location of the strokes. If the number of strokes is less than or equal to the number of lines searched and the strokes occur in the latter half of the segment, then the four has been verified. The methods to discern the specific digits will now be discussed in more detail.

6.2 Verifying Digits

In this section, a few of the actual verification modules for specific digits will be presented. To avoid redundancy, the only digits that will be covered are 0, 1, and 5.

In order to verify a zero, the main procedure is simply to perform a loop detection on the entire segment. The bounds of the loop are check to make sure the loop is long enough to be considered a zero. The interesting problem faced here was the handling of zeroes that have been separated from a larger connected region. Often times, this results in a broken 0 that can easily be rejected. However, since this case occurs so frequently, the tolerance of the broken loop has been adjusted to be slightly less than the height of the segment.

The next case to consider is '1'. This case seems simple at first since the one is simply a straight line. However, the one can be written with a hook at the top and a line at the bottom.

To correctly verify samples such as these, we negate the possible line at the bottom. Then, it is possible to measure the average width in the rest of the segment. Even with the hook, the width of the segment is still relatively small; when the digit is scaled to fit inside the input map, the long line on the bottom prevents the hook from being scaled too wide.

Thus, we can certainly use width as a guideline to verify ones. The unhandled issue then becomes the case where the '1' has a hook but no line on the bottom. During scaling, the '1' can be distorted beyond recognition. To prevent this from occurring, the scaling algorithm does not scale a segment to be excessively wider than it already is, i.e. it uses the aspect ratio of the segment to determine whether to scale or not. This method prevents over-scaling in most cases, but not all of them. This case is not handled perfectly, however, the error occurs infrequently and when it does the assigned value is rejected because is too wide and the post processor does not reassign values.

Finally, we cover the case of the '5'. The '5' is a particularly difficult case because it is structurally very similar to both '3' and '6'. In this case, the loop detection method derived can be employed. To differentiate the input signal from a 6, a loop is searched for from the middle of the segment. Since, an open loop is being searched for, a low broken loop tolerance is given. Then, if a loop is found, the open loop detector is employed. Obviously if the loop is closed, the assigned value is rejected. The process of differentiating between a '3' and '5' is trickier. They both contain an open loop on the bottom; the main difference is the orientation of the curve in the top half of the segment. Therefore, the position of the top edge of the segment is found; if this edge is in the last

quarter of the segment, the assigned '5' is rejected. The reasons for allowing such a lax standard are twofold. One reason is a particular style of writing five where the top of the five and the loop of the five are joined and the edge is not detectable.

The task of differentiating using structural methods is too inexact to always be reliable. Hence the neural net architecture is given the benefit of the doubt a lot of the times as

well. It has the benefit of a non-linear feature extraction, which can indicate features not clearly visible. Further, turning too many digits to not-recognized digits severely affects the performance of the recognition module. Thus, lax constraints are chosen to verify value assignments.

To finish, the structural post-processing module is an inexact methodology. As was found in the many studies covered here, structural recognition is a difficult task for handwritten digits since handwritten data can vary so widely. Nevertheless, structural methods do have use as a verifying module. By applying the most general guidelines about a digit's structure into use, the idiosynchrocies of neural network recognition can be identified and false recognition of characters can be avoided.

7. Other Research

Most research done outside MIT refuses to acknowledge the problems faced with the segmentation scheme alone. Almost all the researches, though they claim to devise solutions for *unconstrained* handwritten numerals, they almost always take separately segmented numerals as input for recognition, which simplifies the task tremendously. Instead of the Segmentation Module and the Feedback associated with it, we are now left with a simple straight to basics Recognition Module and it's sub categories.

Most research papers begin the pre-processing stage from various normalization techniques including Thinning, Resizing and Slant Correction. Re-thickening was notably absent from most research avenues who utilized thinning whereas re-thickening has been shown to improve the performance of the Recognition Module and has almost no overhead. Although some tangential segmentation strategies did come across despite all this.

7.1 Segmentation Scheme

In Concordia University, Quebec, Canada, Kim et al [2] proposed an innovative and fast algorithm for the recognition of unconstrained handwritten numeral strings. In their proposal 4 candidate segmentation points are determined from contour profiles not just 2. Two of these points are derived from vertical boundary difference from the difference of the upper and lower contour, while the other two are derived from valley and mountain ranges of the upper and lower contours respectively.

These four segmentation points go through various transitions governed by the algorithm's equations to determine whether the overlapping strings are numeral in nature. This research can be extremely useful at MIT as the majority of the research on the recognition of unconstrained handwritten numerals had a practical application in mind which was automated bank check reading. Currently the location of the CAB (courtesy amount block) is usually pre-specified or the location is detected with some priori knowledge. With this research the CAB location can be automatically and easily accessed.

7.2 Biorthogonal Spine Wavelets

In the paper "Recognition of unconstrained handwritten numerals using biorthogonal spline wavelets" by Corriea, S.E.N and De Carvalho, J.M., Biorthogonal Spine Wavelets are very innovatively used [3]. They use the Biorthogonal Spine Wavelets CDF 3/7 as a feature extractor and a multi-layer cluster neural network as a classifier. These wavelts can be obtained using the analysis filters for decomposition and the synthesis filters for reconstruction.

The advantage of using them is the multiresolution analysis which provides local information in both the space domain and frequency domain. The low frequency components reflect the basic shape while the high frequency components reflect the details of the character. A very novel idea which also garnered a 97% accuracy.

7.3 Improvements to Structural Techniques

Utilizing the Hidden Markov Model commonly found in automatic speech recognition and online handwriting recognition, Cai et al [4]. They extract chain code-based features along the outer contours since HMM requires sequential inputs. Using these chain code-based features contour shapes and curvature areas and their orientation are mapped. The novel feature is the mapping of this problem onto a finite state sequence in the Hidden Markov Model which has a known solution. They achieved remarkable success rates of 98% or higher etc but the algorithm comes with its costs. It is so computationally intensive that even on an extremely fast Pentium 3 machine it manages only a sluggish few digits per second.

7.4 Crossing Features

When speed is of essence. Chen et al [5] propose a recognition module based entirely a novel scheme of crossing features. What this does is, it uses horizontal and vertical cuts or crossings to divide the image into separate components extremely fast. The module is based entirely on a classification tree, which keeps classifying the image to one of the branches after the data from each cross is processed. As we are well aware, the classification tree has to be absolutely huge if 99% or so success rates are required. But then more and more time will be sacrificed for minimal improvements as you make the tree larger and larger. They chose a blazing fast algorithm with a respectable 91% success rate, much less than most other recognition modules out there, but the low processing time makes this algorithm very useful in certain scenarios.

7.5 Differing Neural Networks

In Queensland, Australia, Atukorale et. Al [6] proposed a modification to the rarely used NG – neural gas network. They made the NG network run faster by defining an explicit ranking scheme. This unique network's hierarchical overlapped architecture allowed them to obtain multiple classifications per sample data. Since multi-classification hinders when it comes to pattern recognition, they developed a multiple classifier system for it based on multiple extraction methods. The authors hope that this will spark interest in the use of NG networks in other areas too, but even as of now, despite their high success rate, molding this network to your needs seems to much bother and not enough rewards.

8. Conclusion

The overall application handles the processing of the numeral image and the location of the segments that are individual digits. The segments produced are normalized in order to simulate uniformity between characters that should be recognized as the same value. This is required because the major difficulty in handwriting recognition (without consideration to segmentation or any other outside factors) is the variation of handwriting style.

Results achieved were promising but there is certainly room for improvements in the realm of the actual recognition module as well as the overall operation environment. First, the recognition module is operating with 16x16 binarized segments. The 16x16 segments are obtained through normalization from the original binarized image which usually reduces the size to fit in the segment. When size is reduced, valuable information can be lost. Thus, the first suggested change would be to increase the size of the input, thus preserving detail and allowing the neural nets and the post-processing module to gather more information. This task has not yet been accomplished because 16x16 is the standard size used in most databases of segments and acquiring all the number of segments needed for training is a very laborious task. However, it may help to improve the performance of the network. Furthermore, this application operates in a totally unconstrained environment and poor quality images are an issue. Better quality images with reasonable contrast and brightness are necessary to improve performance.

In general, the recognition of handwritten information is a difficult process. Research in this area has continued for over 30 years and there is still no reliable or widely successful

handwriting recognition product from pre-written material. In order to improve the performance of any recognition system, constraints on the use or the environment should be used. Still, the system could be further improved through constraining the application environment further. While there is still room for improvement, the recognition module presented has successfully accomplished its task.

One thing has been remarkably clear that the research done at MIT has some of the most comprehensive and robust system designs anywhere. Tackling the very complicated Segmentation Module and improving it with the Segmentation Critic Module and then even adding Feedback back from the Recognition Module to the Segmentation Module has put MIT ahead in the robustness in its scheme. Perhaps since the research was geared towards a very practical application that the system was made more robust with each passing.

9 Bibliography

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

[2] Recognition of unconstrained handwritten numeral strings by composite segmentation method. Kye Kim, Jin kim, Ching Suen. IEEE 2000.

[3] Recognition of Unconstrained Handwritten Numerals using Biorthogonal Spline Wavelets. Corriea, S.E.N; De Carvalho, J.M.Computer Graphic and Image Processing Proceedings, Oct 2000, . Pg 338.

[4] Integration of structural and statistical information for unconstrained handwritten numeral recognition. Jinhal Cai, Zhi-Qiang Liu. Pattern Recognition 1998. Vol 1, Pages 378-380.

[5] Recognition of unconstrained handwritten numerals using crossover features, Chen, M.W; Ng, M.H. Signal Processing and it's Applications, 1999 August. Page 283-288.

[6] Combining Multiple HONG Networks for Recognizing Unconstrained Handwritten Numerals. Ajantha S. Atukorale and P.N. Suganthan. Neural Networks, 1999, IJCNN '99. International Joint Conference on Neural Networks, Voulme 4. July 99

[7] Sparks, P., Nagendraprasad, M.V., Gupta, A. (1992) An Algorithm for Segmenting Handwritten Numeral Strings. *Second International Conference on Automation, Robotics, and Computer Vision.* Singapore, Sept. 16-18, 1.1.1 - 1.1.4.

[8] Feliberti, V. and Gupta, A. (1991) A New Algorithm For Slant Correction of Handwritten Characters, Masters Thesis, Massachusetts Institute of Technology.

[9] Kelvin Cheung: A Study of Improvements for the WinBank CAR System, June 1998.

[10] Anshu Sinha: An Improved Recognition Module for the Recognition of Handwritten Digits, May 1999

[10] Sparks, P. (1992) A Hybrid Method for Segmenting Numeric Character Strings.

[11] Khan, S. (1998) Character Segmentation Heuristics for Check Amount Verification. Masters Thesis, Massachusetts Institute of Technology.

[12] Nagendraprasad, M., Sparks, P., Gupta, A. (1993) A Heuristic Multi-Stage Algorithm for Segmenting Simply Connected Handwritten Numerals. *The Journal of Knowledge Engineering & Technology* 6 (4) 16-26.

[13] Dey, S. Adding Feedback to Improve Segmentation and Recognition of Handwritten Numerals. MIT Thesis. 1999.

[14] Duggan, M.G. "Enhancing Accuracy of Automated Numeral Recognition." MIT thesis. 1992.

[15] Feliberti, V.C. and Gupta, A. "A New Algorithm for Slant Correction of Handwritten Characters." International Financial Services Research Center (IFSRC) Discussion Paper. No. 173-91.

[16] Roman, Gupta, Riordan." Integration of Traditional Imaging, Expert Systems, and Neural Network Techniques for Enhanced Recognition of Handwritten Information." IFSRC. 1990. No. 124-90.

[17] A New Algorithm for Slant Correction of Handwritten Characters, Vanessa C. Feliberti,Amar Gupta

[18] Tony Wu: Enhancement of Post Processing Step in WinBank's Offline Character Recognition System, June 1998.