

**An Improved Recognition Module for the
Identification of Handwritten Digits**

by

Anshu Sinha

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

May 21, 1999

Copyright 1999 M.I.T. All rights reserved.

Author _____

Department of Electrical Engineering and Computer Science

May 21, 1999

Certified by _____

Dr. Amar Gupta

Thesis Supervisor

Accepted by _____

Arthur C. Smith

Chairman, Department Committee on Graduate Theses

**An Improved Recognition Module for the
Identification of Handwritten Digits**

by

Anshu Sinha

Submitted to the

Department of Electrical Engineering and Computer Science

May 21, 1999

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:

Optical Character Recognition (OCR), more specifically handwriting recognition,

is a complex subject that has received much attention over the past 35 years. However, in spite of many years of research, optical readers that can recognize handwritten materials at a satisfactory rate are rare. Reasons for difficulty in reading hand-printed alphanumeric data include the variety of handwritten characters and the ambiguity in the distinction of characters. This paper proposes a new and improved recognition module designed to read the courtesy amount on Brazilian checks. This system focuses on a neural network classifier with a structural system to verify recognition which have both been designed and tailored in a unique way. The

specific techniques employed are discussed and the accuracy results are presented in this paper. While the application described has shown success, there is still room for improvement.

Thesis Supervisor: Dr. Amar Gupta

Title: Co-Director, Productivity From Information Technology (PROFIT) Initiative

Sloan School of Management

ABSTRACT [*](#)

LIST OF FIGURES [*](#)

1. Introduction [*](#)

2. A Check Reading System [*](#)

2.1. Scanning [*](#)

2.2. Thresholding [*](#)

2.3. Automatic CAB Location [*](#)

2.4. Segmentation [*](#)

2.5. Normalization [*](#)

2.5.1. Thinning & Rethickening [*](#)

2.5.2. Slant Correction [*](#)

2.6. Feedback [*](#)

2.7. Syntax Verification [*](#)

3. History of Character Recognition [*](#)

3.1. Template Matching [*](#)

3.1.1. Basic Template Matching [*](#)

3.1.2. Moment Based Methods [*](#)

3.1.3. Fourier Series Based Methods [*](#)

3.2. Structural Analysis [*](#)

3.2.1. Slit/Stroke Analysis [*](#)

3.2.2. Thinning Line Analysis [*](#)

3.2.3. Stream Following Analysis [*](#)

3.3. Neural Networks [*](#)

4. Recognition Module [*](#)

4.1. Pre-Processing [*](#)

4.2. The Neural Networks [*](#)

4.2.1. The Multi Layer Perceptron (MLP) [*](#)

4.2.2. Training the network [*](#)

4.2.3. The Array of Networks [*](#)

4.2.3. Feature Extraction [*](#)

4.3. Arbiter [*](#)

4.4. Structural Post Processor [*](#)

4.4.1. Loop Detection [*](#)

4.4.2. Verifying Digits [*](#)

5. Performance Analysis [*](#)

6. Conclusion [*](#)

7. References [*](#)

LIST OF FIGURES

Figure 1. A sample Brazilian check.. [*](#)

Figure 2. Flow Diagram of System. [*](#)

Figure 3. Method of selecting thresholding point [*](#)

Figure 4. Example of connected and separated digits [*](#)

Figure 4. An HDS Split [*](#)

Figure 5. Shows digits initial form, then after thinning, then after thickening. [*](#)

Figure 6. Segments before and after slant correction. [*](#)

Figure 7. The slit analysis method. [*](#)

Figure 8. Illustration of the sonde method.. [*](#)

Figure 9. The Freeman code; an example and the coding rule [*](#)

Figure 10. Recurrent Neural Network based on an MLP. [*](#)

Figure 11. The flow of the recognition module. [*](#)

Figure 12. Segments input to the neural network. [*](#)

Figure 13. Neuron model. [*](#)

Figure 15. Connected segments. [*](#)

Figure 16. Brazilian digits that vary from US digits. [*](#)

Figure 17. Segmentation where part of the 2 is broken off and added to the 0. [*](#)

Figure 18. A Typical Segment of Connected Zeroes [*](#)

Figure 19. Breakdown of digits used for training and testing. [*](#)

Figure 20. Two identical networks recognizing with different outputs. [*](#)

Figure 21. Standard MLP vs. DDD MLP. [*](#)

Figure 22. Calculating elements for the DDD feature. [*](#)

Figure 23. 4x4 grids used for computing DDD. [*](#)

Figure 24. Cases where the neural network architecture requires the aid of the structural verifier. [*](#)

Figure 25. Algorithm for Loop Detection. [*](#)

Figure 26. A '0' made with an open loop. [*](#)

Figure 27. A '5'. Note the open loop on the bottom. [*](#)

Figure 28. The broken loop tolerance for '0'. [*](#)

Figure 29. Samples of different styles of '1's. [*](#)

Figure 30. The ideal case showing the difference [*](#)

Figure 31. The results on the training/testing sets for each network. [*](#)

Figure 32. The outputs of the arbiter. [*](#)

Figure 33. The actions of the structural post processor and the final results [*](#)

Figure 34. An example of a poor quality Brazilian check. [*](#)

Figure 35. Results of testing on Brazilian checks. [*](#)

1. Introduction

The replication of human function has been the subject of much research in the artificial intelligence field over the past 35 years. Optical Character Recognition (OCR) is one facet of this area that has attracted much attention for a variety of reasons; there are many applications for a recognition product such as zip-code recognition, check amount encoding and document reading. Also, in the early days of pattern recognition research, characters were considered very handy to deal with and were regarded as a problem that could be solved easily. However, after some initial easy progress in the area, it became clear that the problem of character recognition was actually quite difficult [13].

The task of recognition can be broadly separated into two categories: the recognition of machine printed data and the recognition of handwritten data. Machine printed characters are uniform in size, position and pitch for any given font. In contrast, handwritten characters are non-uniform; they can be written in many different styles and sizes by different writers and by the same writer. Therefore, the reading of machine printed writing is a much simpler task than reading hand-printed writing and has been accomplished and marketed with considerable success. In fact, several companies have produced low-cost software that can, when used with scanners, read in documents of several different 10 and 12-pitch fonts. However, in spite of many years of research, optical readers that can recognize handwritten materials at a satisfactory rate are rare. Even if they exist, they typically can read only hand-printed digits that must be legible and consistently placed. For example, some Japanese machines can read alphanumeric hand-printed data in pre-printed boxes. Knowledge the number of characters to expect is a technique employed by the US Post Office for reading zip-codes. Right now, the only method of satisfactorily recognizing totally unconstrained digits and characters is through on-line products where real-time information (such as pen-stroke) and features are extracted to aid recognition [13].

Reasons for difficulty in reading hand-printed alphanumeric data include the variety of handwritten characters

and the ambiguity in the distinction of characters. In contrast, systems that recognize machine-printed characters exploit information such as the width and spacing of characters, as well as the uniformity of the printed character to greatly simplify the recognition process [24]. Furthermore, they can easily utilize techniques such as template matching. Template matching uses the principle of superposition to determine how much a test or scanned in character matches a template or a standard. This is not possible with handwritten characters, which vary widely because of differences in writing habits and styles between different writers. In addition, the same writer can print a character differently each time he/she writes it. As such, it is clear that a recognizer for handwritten data needs to incorporate a power for generalization of inputs in order to equalize the wide variations that occur.

One application in this area that has received a good amount of attention is the recognition of check amounts. The system built and discussed here is an application for the recognition of the courtesy amount block of Brazilian checks. The courtesy amount block is the location on a check where the amount is written in numbers. A sample of a Brazilian check is shown below:

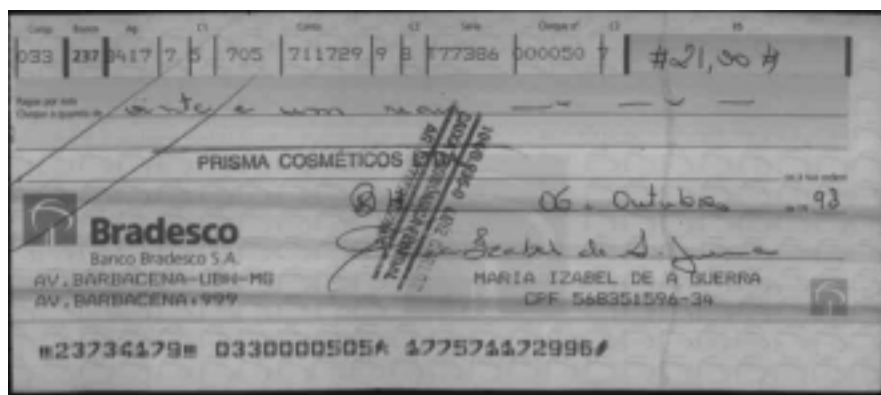


Figure 1. A sample Brazilian check. The courtesy amount block is located in the top right corner.

One reason for choosing this area is that the issue of reading courtesy amounts from the amount block on checks simplifies the task of recognition by reducing the domain of possible values from all alphanumeric characters to the set of digits plus a few delimiting characters (such as \$). As far as relevance of the task, implementing an OCR for check reading can greatly reduce time and money spent in the banking industry. Currently, in check processing, amount encoding is a labor-intensive step which can require as many as 2 people to encode each check (redundancy is used for accuracy). Automating this task can dramatically reduce the workload of menial labor and correspondingly, reduce the cost of check processing. Furthermore, a human operator generally takes 5-10 seconds to process a check while an automated system should be able to offer better speeds [3]. The main focus of the design of a good check reading system is accuracy and error rate. A high error rate may be unacceptable in the banking industry for obvious reasons such as over and under billing.

The area of focus in this paper is the recognition module. There are many methods used to recognize digits. This system focuses on a neural network classifier with a structural system to verify recognition. Both sections have been designed and tailored in a unique way for the Brazilian environment and will be discussed in detail in the course of the paper. The next section of the paper will cover the overall structure of the OCR system, followed by a brief history of character recognition. Then, the paper will discuss the details of the classifier that we have implemented. First, we will cover in depth the main portion of the recognition module. Then, we will explain the structural verifier used followed by performance results of our system and finally, concluding remarks.

2. A Check Reading System

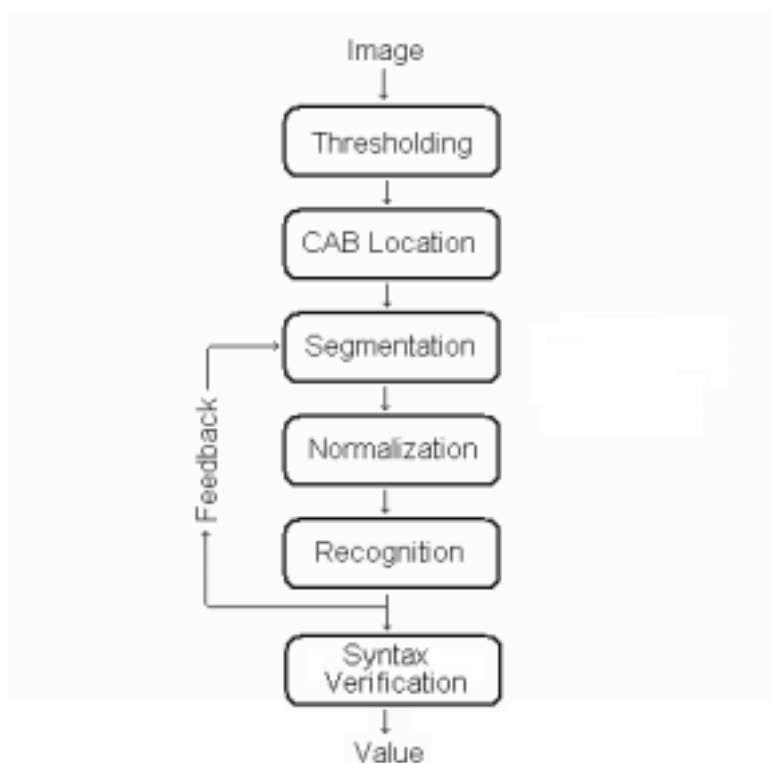


Figure 2. Flow Diagram of System.

2.1. Scanning

In this phase, a grayscale image is captured and digitized using a scanner or a digital camera. It is important to extract as good quality image as possible in order to obtain information from the image later. However, clean good quality documents are seldom encountered in real applications. Common problems with digital images obtained in testing include smudged characters, poor writing or print quality (i.e. extraneous writing in the background) and poor contrast between the writing (foreground) and the background. Generally, images are scanned in at 100 dpi and adequate results are obtained with this quality.

2.2. Thresholding

The next step is to extract a binary (0,1) image from the obtained digital image, which is then used for analysis in determining the value of the character. Image thresholding classifies the pixels of an image into the foreground (the writing) and the background. In the case of grayscale images, the pixels initially have a value from 0-255, and the general idea of thresholding is to convert pixels above a certain level of gray into foreground and to convert pixels below the level into background. In the simplest method, global thresholding, a pre-determined constant T is used to threshold the image. Clearly, this technique does not consider the difference between characters in a given image or between various images; contrast and brightness in an image can vary making this method too simplistic. Depending on image quality, the background may be very dark or light, the writing may be fuzzy and light or dark and clear. For this reason, a dynamic process must be used to threshold the image. For example, another method is to use the pixels in the corner of the image which are assumed to be background and create a threshold value based upon these pixels. However, while this method does pay attention to the shade of the background, it still ignores the foreground and can lead to poor and spotty results. A common method practiced is to use a histogram of the pixel values in the image; there should be a large peak indicating the general value of the background pixels and another, smaller peak indicating the value of the foreground pixels.

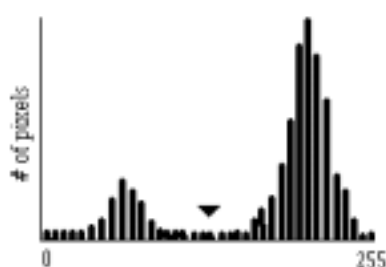


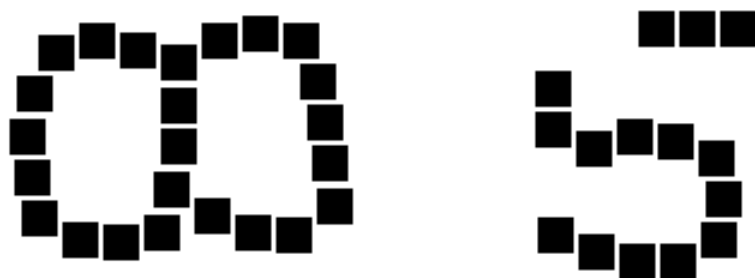
Figure 3. Method of selecting thresholding point

A threshold can then be chosen in between the two peaks; this is known as valley-seeking [22]. This is a successful method, however images do not always contain well-differentiated foreground and background intensities due to poor contrast and noise. Perfect thresholding is a difficult task, and this method provides adequate results so a variation of this method has been implemented for use in this application.

2.3. Automatic CAB Location

The courtesy amount block (CAB) contains the numerical amount and is necessarily obtained to read the check amount. The CAB on checks from different banks is usually in the top right hand corner of the check image; however, the exact size and position of the CAB can vary. This makes it necessary to automatically locate the CAB within the image. Currently, this is accomplished by searching for horizontal and vertical lines which delimit the CAB, and if these lines are not found, the CAB location is set to a default. While this method is clear and logical, some difficulties are encountered in locating these lines. CAB delimiting vertical lines are sometimes difficult to distinguish from vertical lines in the handwritten digits. In addition, varying CAB styles require us to account for many possibilities. For example, two vertical lines are often used to indicate the CAB area, but sometimes only a single line or two horizontal lines are used. Furthermore, if the contrast between the lines of the CAB and the color of the background is low, the lines are difficult to distinguish from the background during thresholding. Because of the need to search for the location of the CAB, locating the handwritten digits being recognized remains problematic and research in this area is beyond the scope of this paper.

2.4. Segmentation



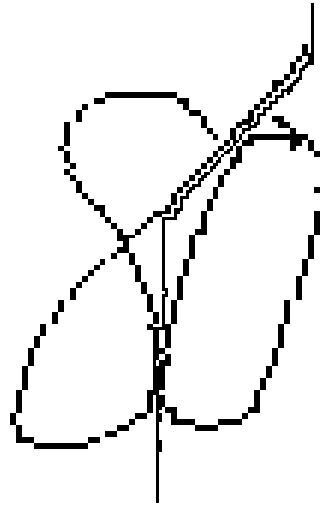
The next step in the process of recognition is to break up the binary image of the CAB into the images of each individual digit. Then each digit will be represented by a 2 dimensional bitmap array. This is one of the most difficult pieces of the OCR system. Basically, the segmentor is expected to divide the CAB into digits without any prior knowledge of the digits in the image or even what a digit is. It has no information about the width or the height of the character and this data varies greatly depending on the digit. Given this, it should also be clear that it can be very hard for the segmentor to determine whether an extracted segment represents 2 digits or 1, or even only a part of a digit. The algorithms that are used, then, must make use of heuristics that address the nature of joints between characters [21].

Figure 4. Example of connected and separated digits

Clearly, it is impossible to produce perfect segments 100% of the time; however, as long as the segmentor can output segments that contain all or most of the actual character, the recognizer should be able to absorb some errors in the segmentation.

Whitespace division is the simplest method used for segmentation. In this case, the segmentor simply searches for a vertical line of only background pixels and divides between segments at that juncture. Weaknesses of the method include its inability to handle connected digits, as well as slanted or adjacent characters which overlap on common vertical column(s) thus preventing segmentation. However, this algorithm requires minimal computation and is simple to implement [21].

More commonly, the connected regions of the CAB are found through contour detection, and computations are done on these regions [21]. Here, the segmentor sees what pieces are connected and uses heuristics to determine if the region is a digit, a part of a digit or multiple digits, and operations are performed accordingly. For example, regions can be merged if the segmentor decides that two regions are close enough and have a high degree of vertical overlap. The task of separating connected digits is a much more wieldy endeavor to accomplish accurately. Connectivity algorithms are more expensive than the whitespace algorithm but they are built to address the more complex issues of connected and overlapping characters. An example of such an algorithm is the Contour Splitting Algorithm developed and explained by Shridhar and Badreldin [19]. Basically, this algorithm begins with the contour lines of a connected region to be split. Contour lines refer to the set of points on the outermost or innermost surfaces of a connected region of black pixels, i.e. points touching whitespace. The algorithm drops a vertical line through the center of the segment. If it encounters none or 2 contour points it splits the segment. Otherwise, it continues to drop vertical lines, incrementing one and alternating between the left and right. If no split is found, a Hit and Deflect Strategy (HDS) is used [19]. This is similar to contour splitting except a vertical line is not used; in its place, a simple dividing line zig-zags its way through the space between characters. This strategy assumes that a good point to start the line has been chosen,



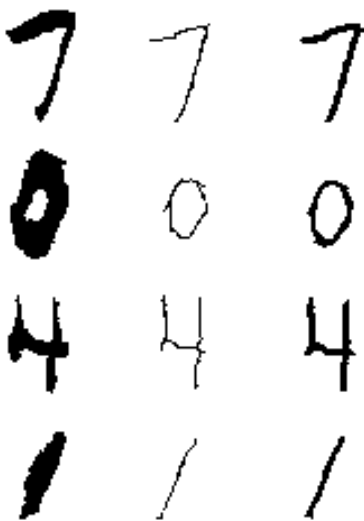
otherwise the segment can be cut in strange places. These are 2 basic strategies that are commonly used. There are other variations, but most of the algorithms are reliant upon the use of contours.

Figure 4. An HDS Split

2.5. Normalization

The next step in the process is normalization. Because of the use of neural networks in our recognition process, we must scale down the segments obtained to a pre-determined size. The network has a fixed amount of input nodes which is why this constraint is required. Furthermore, in this phase, the segment is standardized in order to minimize structural variations so that the recognizer does not have the impossible task of accounting for many different writing styles. Once, the segment has been scaled down to appropriate size, the normalization steps are: thinning and rethickening and slant correction.

2.5.1. Thinning & Rethickening



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. To do this we implemented a variation of the fast-parallel algorithm for thinning digital patterns proposed in [14].

Figure 5. Shows digits initial form, then after thinning, then after thickening.

Once we have the thinned segment, we can rethicken to a uniform width simply by ‘blotting’ or adding ‘1’s to each ‘1’ in the skeletonized digit.

2.5.2. 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. To accomplish this task, the heuristic that all digits are taller than they are wider is exploited. Therefore, the segment is rotated around its center of gravity until this point is found [5]. The figure below shows some examples.

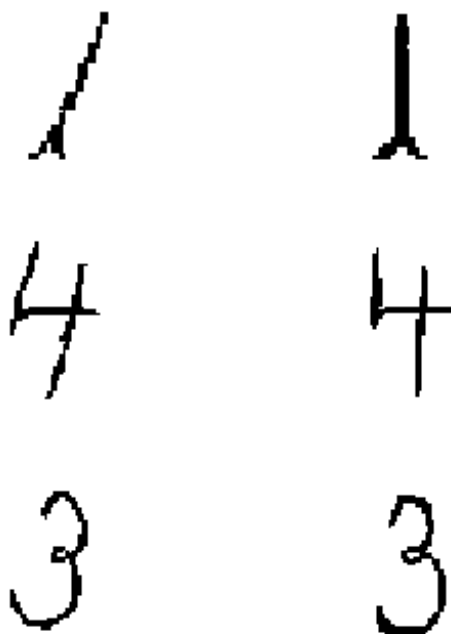


Figure 6. Segments before and after slant correction.

2.6. Feedback

Once the segments have made a pass through the recognizer, they either have a value assigned to them or are not recognized. The goal of the feedback architecture is to improve the handling of unrecognized segments. Rather than immediately rejecting them or having a specified action to take, the feedback system examines the rejected segments and makes more intelligent decisions. Merged fragments may be unmerged; separated characters may be separated differently or not separated at all. The characteristics of the rejected segments are used to identify them as candidates for merging or segmentation.

The segmentation algorithms generate a set of paths along which to divide a region; these paths are ranked and the best path is selected for use. Various features of the paths are used to assign a score to each, and the path with the best score is chosen for use in segmentation. Then, if a given path does not produce recognition, the other paths are used [2].

2.7. Syntax Verification

The final step in the recognition of a bank amount is to determine the amount of money given by the recognized segments. If the recognized amount does not make a monetary value then the recognition is rejected. For example, if the recognized amount is "1,2,3", the syntax verifier can reject this value as not forming a monetary value.

3. 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 [13]. 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.

3.1. 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 [8].

3.1.1. 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 [4].

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. [3]

Horowitz and Shelton of IBM did very extensive research on the autocorrelation method in 1961 [13]. 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 [10]. 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 [13] 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 $S(h_0, h) \rightarrow \delta < 1$. 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 [10]. 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 [13].

3.1.2. 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 [13] 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 [13].

3.1.3. 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 [23]. To review, the set $\{A_k, a_k, |k=1,2,\dots\}$ is called the Fourier descriptor (FD) for a curve, where A_k and a_k are the amplitude and phase of the k th 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 [13] 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:

$$a_n = S^{-1} \int_S u(t) \exp(-j2\pi S^{-1}xnt) dt$$

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'.

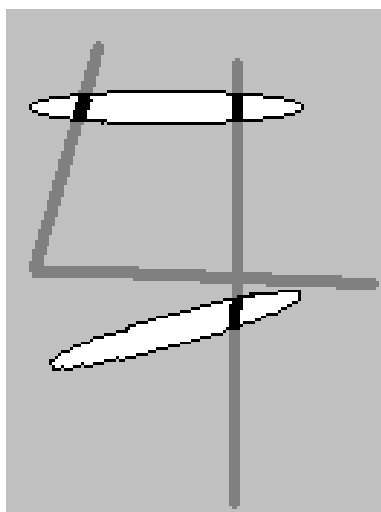
Systematic and reliable experiments were conducted for Fourier Descriptors by Lai and Suen in 1981 [13]. In their experiment Fourier descriptors were used both as global features and as local boundary features. They used a data set of 100,00 hand-printed numerals in 64x32 binary arrays. In the first 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.

3.2. 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.

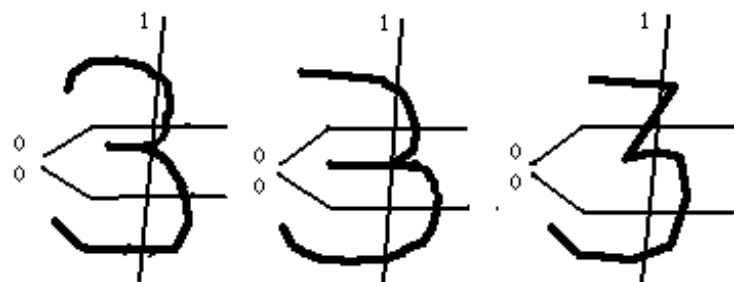
3.2.1. Slit/Stroke Analysis



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 [13,3,21]. 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 as shown in the figure below.

Figure 7. The slit analysis method.

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 and more information is provided in [15].



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.

Figure 8. Illustration of the sonde method. The last example shows the weakness of sonde.

As can be seen in the figure, 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) [13].

3.2.2. 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 [13,21]. 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. For example, the figure shows an

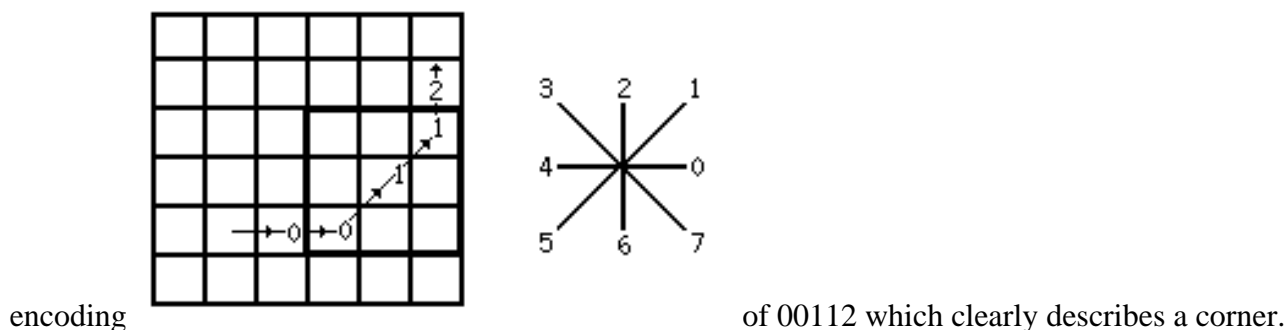


Figure 9. The Freeman code; an example and the coding rule

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.

3.2.3. Stream Following Analysis

This idea was first developed by Rabinow in 1962 [17]. 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 [13].

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. [3]

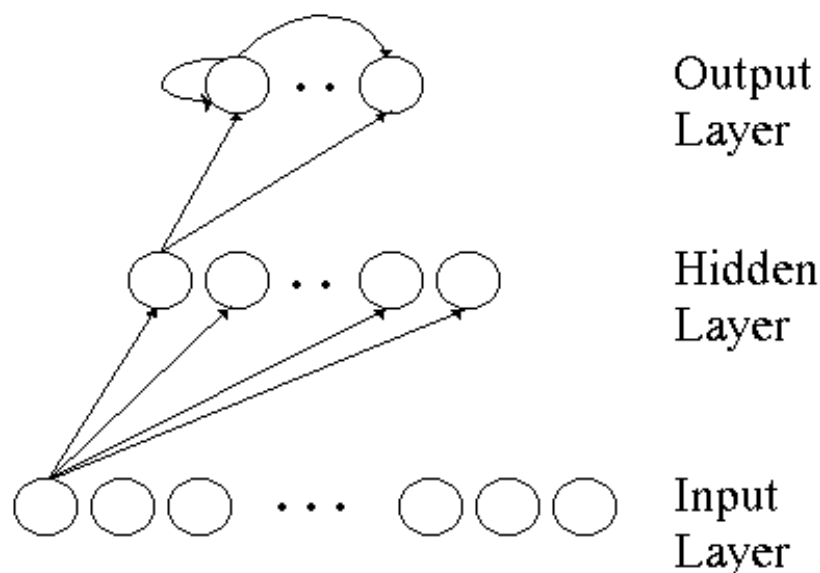
3.3. Neural Networks

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 [9]. 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 [18]. For a basic background of neural networks see [9]; a simple overview is presented in section 4. 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 [6] in conjunction with traditional back propagation. In their model, three hidden layers are used with local averaging and subsampling between layers [1]. However, it has since been shown that the number of hidden layers does not affect the performance of a neural network greatly [9]. 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 [18]. 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) [11]. 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. The structure is shown below:

Figure 10. Recurrent Neural Network based on an MLP.

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.

In 1995, researchers at SUNY Buffalo developed a system which was able to separate hand-printed data from machine-printed data and then recognized characters using a neural network architecture [20]. 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 than 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.

4. Recognition Module

The recognition employed in this system uses the information and experiences gathered from previous systems to form an efficient and accurate module for segmented digits. The basic structure is shown in the figure below:

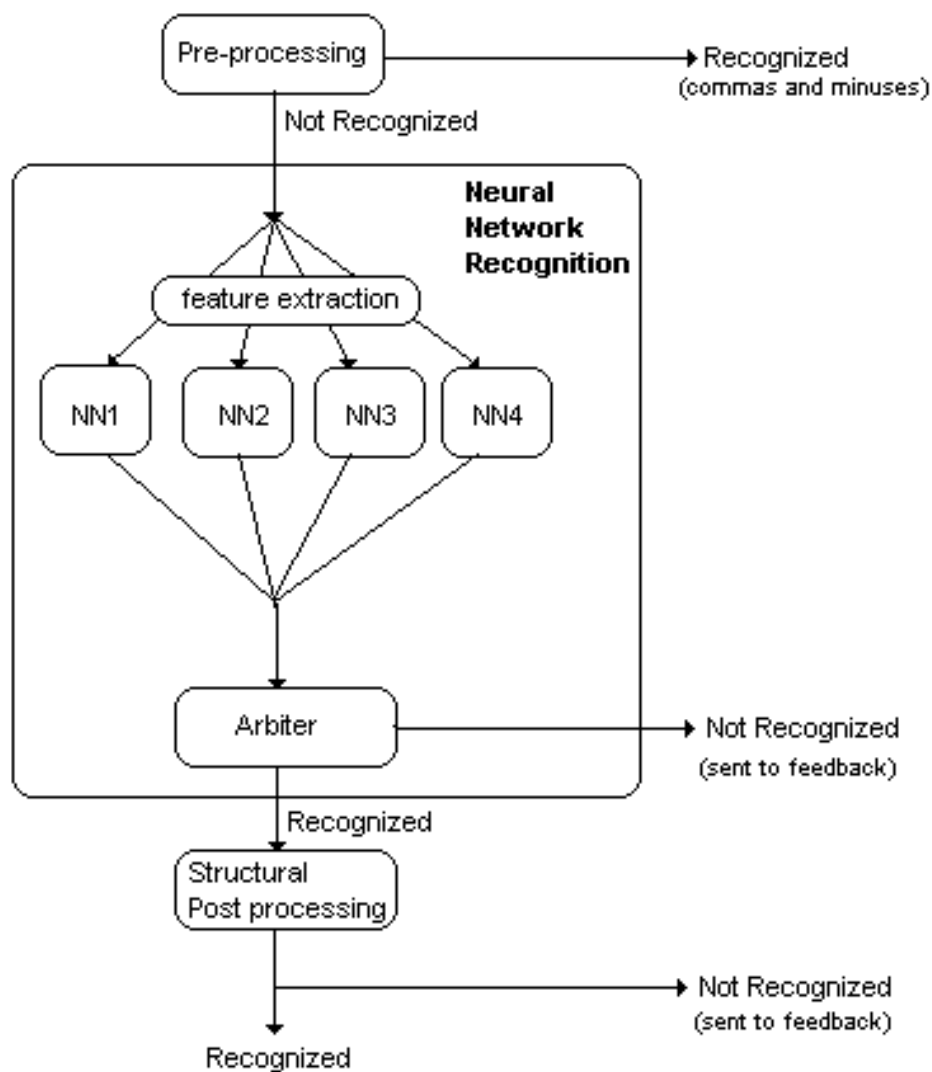


Figure 11. The flow of the recognition module.

As the figure shows, the first step in the recognition process is a special pre-processing step. Here, commas and minuses are structurally recognized; also, segments that can be interpreted as noise or not valuable to recognition are discarded.

In the next step, the segment is input to the neural network recognition module. This section is the meat of the recognition process. The input is taken and features are extracted in order to increase generalization power. Then the input, represented by the features extracted from it, is passed to an array of neural networks. Each network produces an output and an associated list of confidences for each possible 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. Thus, it attempts to connect the digit with a fragment to complete the digit or separate the segment into pieces that are digits. It uses various segmentation techniques in this process as discussed previously.

If the segment is accepted, the structural post-processor then takes control. It looks for structural information in the segment that can verify that the recognized value is correct. Basically, it attempts to make sense of the structure given its recognized value. Using structural post-processing not only reduces the false-positive rate, but also clearly saves time over a purely structural methodology. The next sections discuss each piece of the module more specifically.

4.1. Pre-Processing

As discussed previously, this module recognizes commas and minuses as well as discards information that cannot possibly be a digit. To do this, it uses heuristic guidelines. For example, if the segment has a short height and wide width, it is probably a minus. Furthermore, if the segment has a short height and there is more than 1 stroke per line, where a stroke is defined as the number of isolated black regions in a line, then it is probably digits connected together. The reason for the small height would then be that the connected digits are too wide and when the segment is scaled to fit into a segment for recognition, the height is reduced. Commas are recognized in a similar fashion. Lastly, segments where the black pixels occupy a substantially large portion of the segment are interpreted noise that has been blown up; these segments are thrown away. This process readies the segment for the neural network portion of the array. The neural network could actually handle all of these cases. In fact, connected digits are handled by the neural network. However, in the case of dashes and commas, not enough segments were acquired to train on. Training is the integral step in neural network accuracy and an unbalanced number of segments in a given output category can throw off the overall atunement and accuracy of the entire network. For this reason, some types of outputs were handled structurally.

4.2. The Neural Networks

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 a check image, and the input that the recognizer receives is the output of a segmentor that is handling input images with an unknown quality as well as with an undetermined number of characters. This unconstrained environment for recognition can cause the precise cases noted above; namely, noisy images and inputs with broken structure or additional pieces added on (see figure

below). Because the recognizer needs to absorb such variations in the input, the logical choice for successful character recognition is a neural network-based architecture.

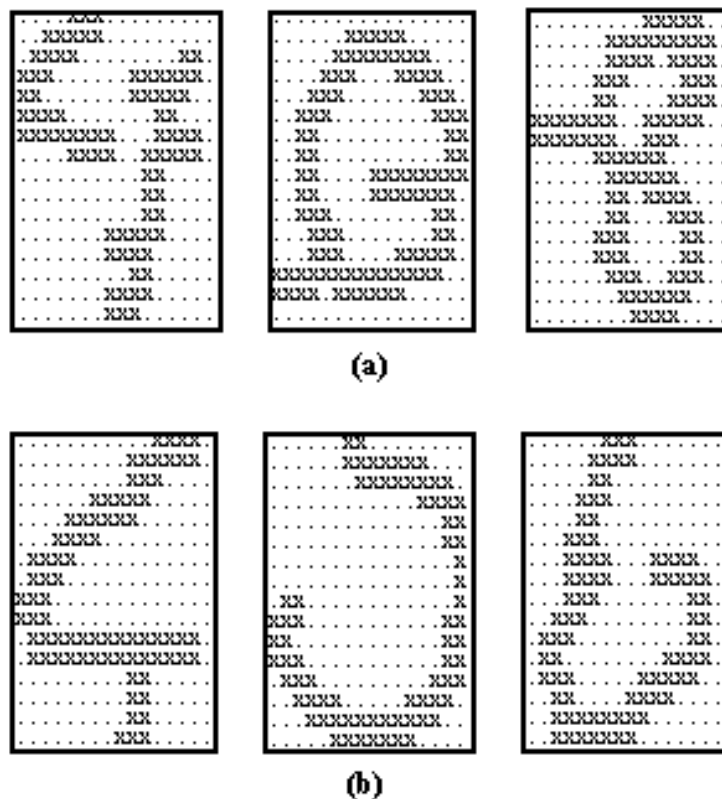


Figure 12. Segments input to the neural network. (a) show a ‘4’, ‘0’, and ‘8’ that are noisy or have pieces attached. (b) shows a ‘4’, ‘0’ and a ‘6’ with pieces broken off.

As discussed in section 3.3, there are many different neural structures that can be employed. In this paper, multi-layer perceptron networks, neocognitron networks and recurrent neural networks were all touched upon. Other studies, not discussed here have successfully employed radial basis function networks (RBF) and time delay neural networks (TDDN) for character recognition [12]. 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 others to a large extent, but recognition is improved by tailoring the network to its application environment.

4.2.1. The Multi Layer Perceptron (MLP)

For this reason, the model implemented in this case was a three-layer, fully connected, feed-forward MLP. The MLP structure is the simplest to understand and to implement. After much testing and experimentation with various types of networks found in the literature, it is the most widely used for character recognition [18].

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 model of a neuron is given in the figure below:

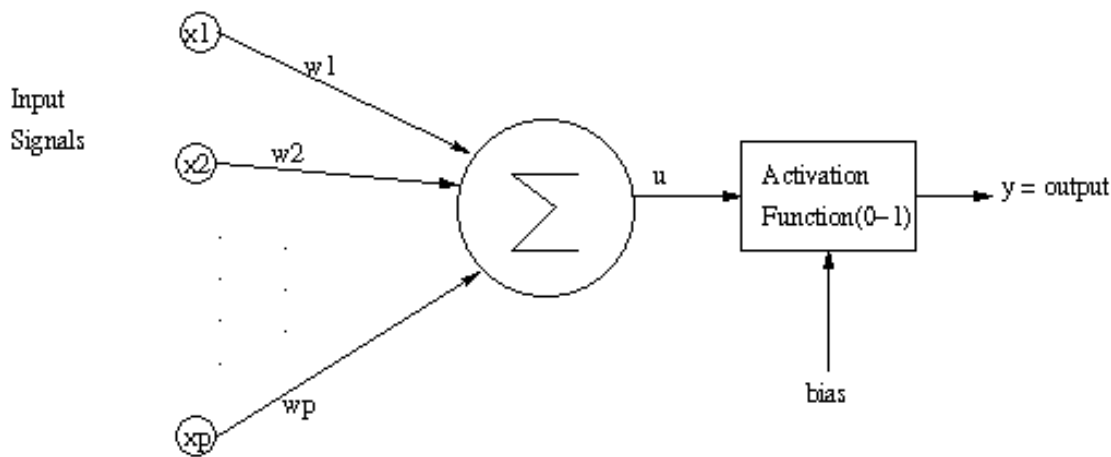
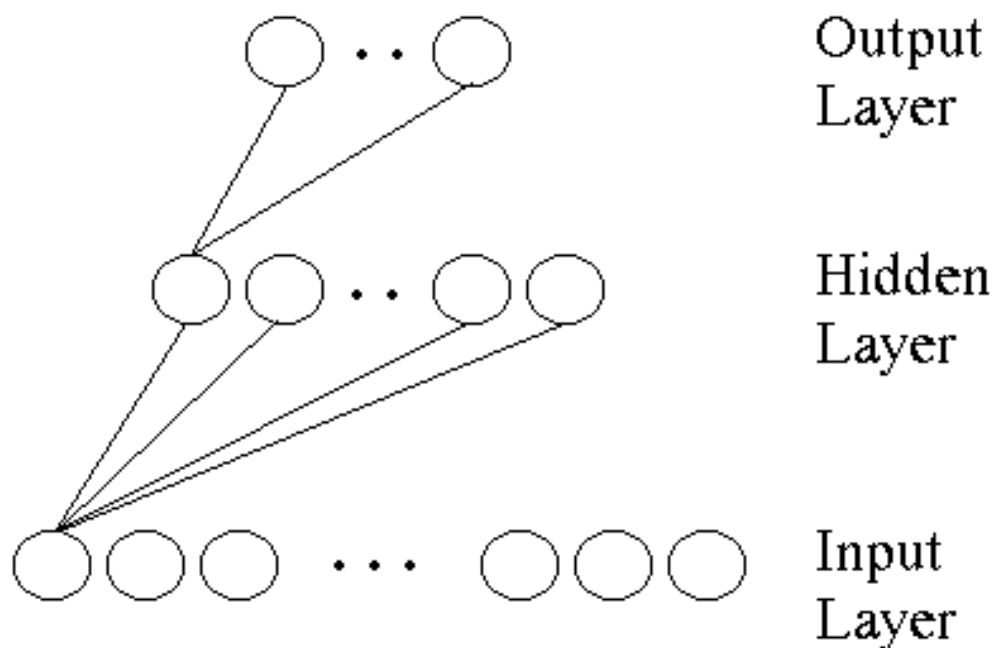


Figure 13. Neuron model.

The neuron takes a set of inputs. 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. The output of the neuron is simply stated in the following 2 formulas:

$$y = \text{Activate}(u - \text{bias}) \quad \&\& \quad u = \sum_{(i=0 \text{ to } p)} \{x_i * w_i\}$$

, where y is the node output.



The network of neurons is organized by layers. The MLP structure implemented has three layers: an input layer of 256 nodes, a hidden layer of 80 nodes and an output layer with 11 nodes. The diagram below illustrates the basic structure.

Figure 14. A fully connected, three layer, feed forward network.

The nodes of the input layer represent the pixels of the input bitmap or features extracted thereof. In the hidden, layer 80 nodes are used. To refresh, the hidden layer is used to extract increasingly meaningful patterns from the

input data. The number 80 can be experimented with; general trial and error with the number of nodes in this layer has shown that 80 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 [9]. 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 11 possible outputs: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, #, and NOT_RECOGNIZED. The rationale behind the numeric outputs are clear; the remaining outputs require additional explanation. The application being discussed has been tailored for Brazilian checks due to their sponsorship of this project. In Brazil, # is a common delimiter used in checks.

As for the NOT_RECOGNIZED output, it simply represents segments that are not identifiable as any of the other possible outputs. It should be remembered that the recognizer is operating in an unconstrained environment and is handling outputs straight from the segmentor which may or may not have been segmented properly. Initially, the plan had been to include a connected-digits output as well as a connected-zeroes output, because the segmentor outputs both frequently enough. Connected digits would represent what the network thinks are 2 or more digits that need to be separated, connected zeroes would represent connected zeroes (00) (see figure). Handling these

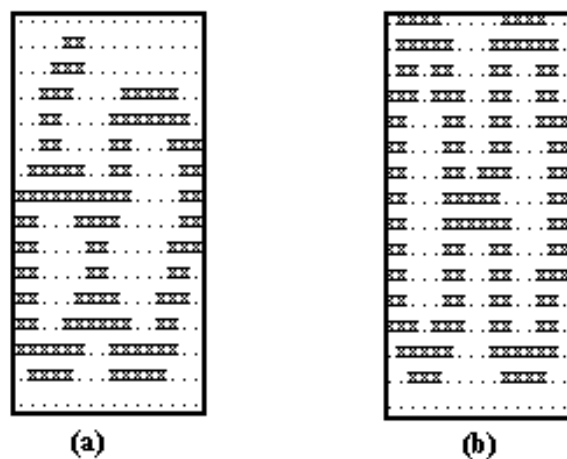


Figure 15. Connected segments. (a) shows a '6' connected to a '0' (b) shows connected '0's.

segments would also greatly aid the task of the segmentor and feedback module. For example, if connected zeroes are identified, the feedback module does not have to attempt to re-segment them, it can just count the segment as recognized. Likewise, a connected-digits output would also help the feedback module choose what action to take when re-evaluating the input. As it turns out, both outputs are excellent choices to include as network outputs, however the number of these segments found was limited. In order to properly train the network, there should be balanced amounts of each possible output. This prevents the network from overly tuning itself to one particular output, or in this case, not tuning itself enough which causes the outputs of the

particular nodes to be erratic and unpredictable. Thus, connected segments are not given their own output nodes, but they are still included in the training module under the umbrella of not-recognized segments. This allows the network to still learn about what segments to expect while maintaining the recognition capabilities.

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. Some networks found implemented in the research use local connections in order to reduce the amount of computation that the network needs to do. An example of this is the AT&T model discussed [18]; however, that model was implemented before the 1990s, and increased computation power has made this concern moot. 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.

4.2.2. 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. As discussed previously, 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 [18].

Back-propagation is the most popular and the most successfully employed training algorithm. Thus, this algorithm was used to train the networks presented here. Again, for specific information on the details of this process, see [9].

All of the neural networks in this module have been trained on Brazilian digits which have been output by the segmentor implemented for this application. The reasons for this training method are threefold. For one, the application being discussed has been tailored for Brazilian data, and the difference between handwriting styles and practices between Brazil and the US are significant enough to warrant a separate training set (see figure).

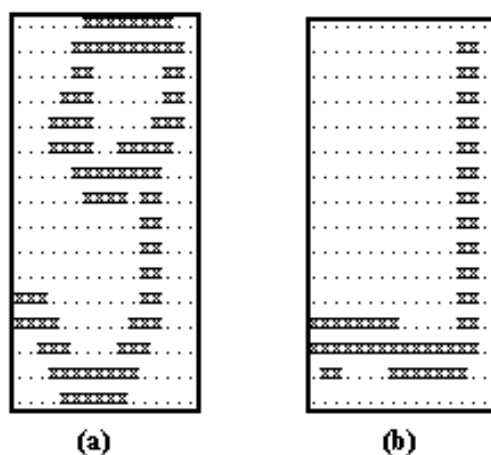


Figure 16. Brazilian digits that vary from US digits. (a) shows a ‘9’ with a hook on the bottom and (b) shows a ‘1’ with a line on the bottom; both cases are common in Brazilian data but rarely seen in US data.

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 [7].

Secondly, we have used digits outputted by the application’s segmentor 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 from other digits. Using digits outputted from the application’s own segmentor enables the recognizer to acclimate to the kind of output it will receive in general use. The segmentor is a rule-based system so its output is consistent and can be predicted well if the neural network is tuned to it. As suggested above, the segmentor 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 figure below gives an example of

such a case:

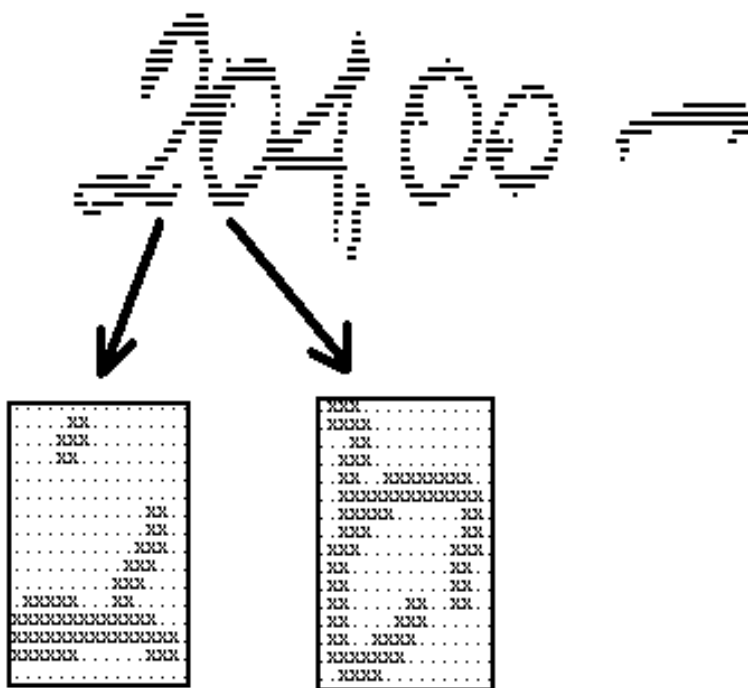


Figure 17. Segmentation where part of the 2 is broken off and added to the 0.

The network will 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.

Lastly, digits obtained from a database generally do not include some inputs needed to properly identify the entire check amount fields found in Brazilian data. First, some of the segments produced from Brazilian checks are not found at all in US checks. For example, the # sign is commonly used to delimit both sides of the written monetary amount. Secondly, because a segmentation based recognition model is employed in this system,

segments with connected information need to be handled. Connected zeroes and/or connected digits are commonly outputted by the segmentor module for input into the network, an example is shown below.

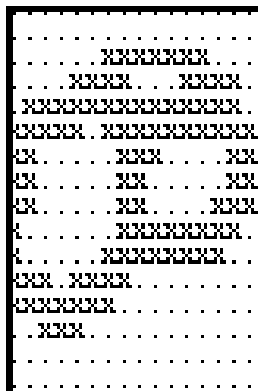


Figure 18. A Typical Segment of Connected Zeroes

In fact, 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 from the check data is nearly essential for identifying some of the patterns that are outputted by the segmentation module.

For these reasons, data retrieved from Brazilian data were included in both the training set and the testing test. Roughly 1000 Brazilian checks were used to obtain data for the database of segments. In order to obtain enough segments to train on and to balance the number of segments for each possible value, data from outside sources were also added to the training set. One of the main problems with collecting our own 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 which appear more commonly on checks, i.e. 1 and 0, while some values appeared very infrequently, such as #. When the networks are trained with a greatly imbalanced number of segments for values, the performance is greatly degraded. Generally, the false positive rate increases because digits found frequently in the training set are subsequently guessed more often since the network is tuned to expect them. The figure below gives the breakdown for segments used in the training and testing sets.

	# in Training File	# in Testing File
0	299	270
1	328	183
2	287	120
3	259	107
4	279	87
5	293	137
6	242	78
7	266	131
8	221	95
9	285	72
#	214	100
NOT RECOGNIZED	130	64
Total:	3103	1444

Figure 19. Breakdown of digits used for training and testing.

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; for information on this, see [9]. The networks were trained with sets until the accuracy rate on the training set was greater than 95%; the limit was set here in order to balance recognition rate while not tuning to finely to the training set of data.

4.2.3. 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. The illustration below shows the recognition of two separate networks with no feature extraction trained on the same data. The figure only shows recognition numbers for 0 through 5 simply to illustrate the point clearly. In this case, the first network guesses a three with very high confidence but also indicates that the input has the characteristics of a five. The second neural net does not have a high confidence for any value but still is more confident that it is a five than a three. Since the first network is very confident of the three, it acts to balance the overall module and produces a correct output.

	0	1	2	3	4	5
NN1	0.00	0.00	0.00	0.94	0.00	0.35
NN2	0.00	0.00	0.00	0.42	0.00	0.62

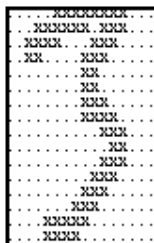


Figure 20. Two identical networks recognizing with different outputs.

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. As seen in the figure, one network is highly confident while the other is unsure. In this case, one network corrects the other; in another situation this situation may be reversed. In both cases, at the very least, an incorrect answer is avoided and in many cases, the right answer is found.

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, as discussed above, the networks are presented with inputs in pseudo-random order which also contributes to the varied results of each network.

4.2.3. Feature Extraction

Gupta, et. al. states 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 [18]. 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 allow for an increase an accuracy. The example below shows cases where the standard MLP networks correct the DDD networks and vice versa.

	0	1	2	3	4	5		0	1	2	3	4	5
DDD NN 1	0.00	0.00	0.06	0.98	0.00	0.01		0.00	0.00	0.18	0.00	0.00	0.06
FOS NN 1	0.00	0.00	0.63	0.94	0.00	0.01		0.00	0.00	0.93	0.00	0.01	0.00
DDD NN 2	0.00	0.00	0.03	0.99	0.00	0.04		0.00	0.00	0.12	0.00	0.00	0.17
FOS NN 2	0.00	0.00	0.61	0.84	0.00	0.06		0.00	0.00	0.80	0.00	0.01	0.00
FINAL	0.00	0.00	0.33	0.94	0.00	0.03		0.00	0.00	0.51	0.00	0.00	0.06

(a)

(b)

Figure 21. Standard MLP vs. DDD MLP. (a) shows the standard MLP correcting the DDD MLP and (b) shows

the opposite case.

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.

In the system developed here, a version of the directional distance distribution feature discussed by Oh and Suen [16] is used. Their feature is based on the distance information computed for both black and white pixels in 8 directions. They consider the distances from black pixel to white pixel and white pixel to black pixel. Furthermore, Oh and Suen regard the image 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. The figure below should clarify this part of the algorithm.

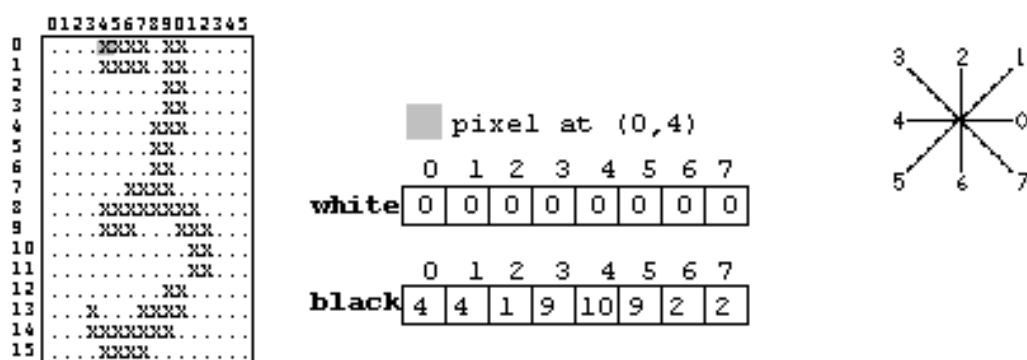


Figure 22. Calculating elements for the DDD feature.

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. For more information, see [16].

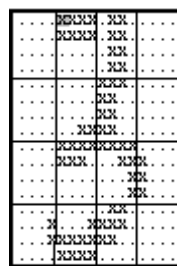


Figure 23. 4x4 grids used for computing DDD.

The DDD feature has, in fact, proved to increase recognition results in our system especially when used as an element of the neural network array. As seen previously, 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.

4.3. Arbiter

The final module in the neural network architecture is 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. If another confidence is close to or comparable to the highest confidence, the highest confidence value is rejected 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.

4.4. Structural Post Processor

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.

Many studies and implementations of structural post-processing module do indeed try to determine the correct value structurally if the assigned value is rejected [3]. This application is tailored for the banking industry which inherently has a low tolerance for errors, and thus does not attempt this task. Further, the difficulty in reassigning values to a particular input comes because many of the mistakes in recognition come in digits that can look very similar, such as 3s and 5s or 1s and 7s; the task of distinguishing the pairs of characters reliably is a wieldy task. In other words, if the structural post-processor could accomplish this 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 application can use re-segmentation and re-recognition to improve its correct recognition rate while minimizing the false positive rate [2]. Thus, through the feedback module, 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. The figure below shows examples of cases where the structural postprocessor can be used to correct the neural net architecture:

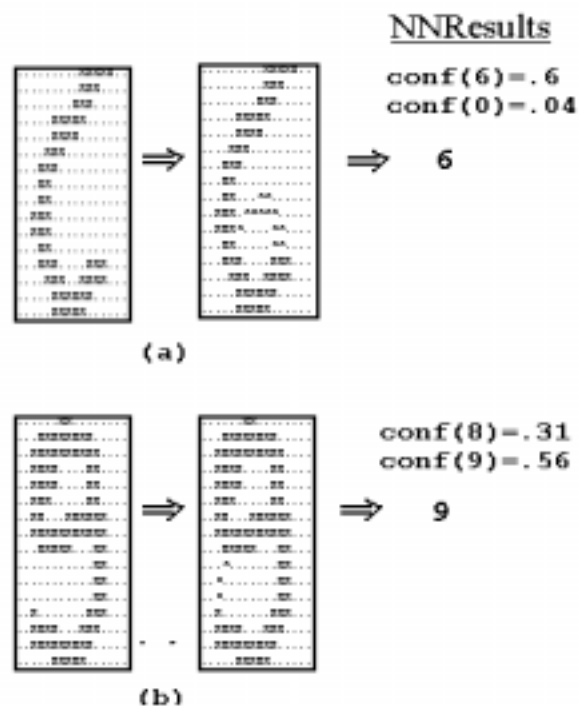


Figure 24. Cases where the neural network architecture requires the aid of the structural verifier.

(a) shows a half of a '0' that has been interpreted as a '6'. (b) shows how the '9' can resemble an '8'. The *s in the images denote the locations where the stroke could have been broken off.

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.

4.4.1. Loop Detection

First, the structure of the algorithm will be outlined and then the merits of it will be discussed.

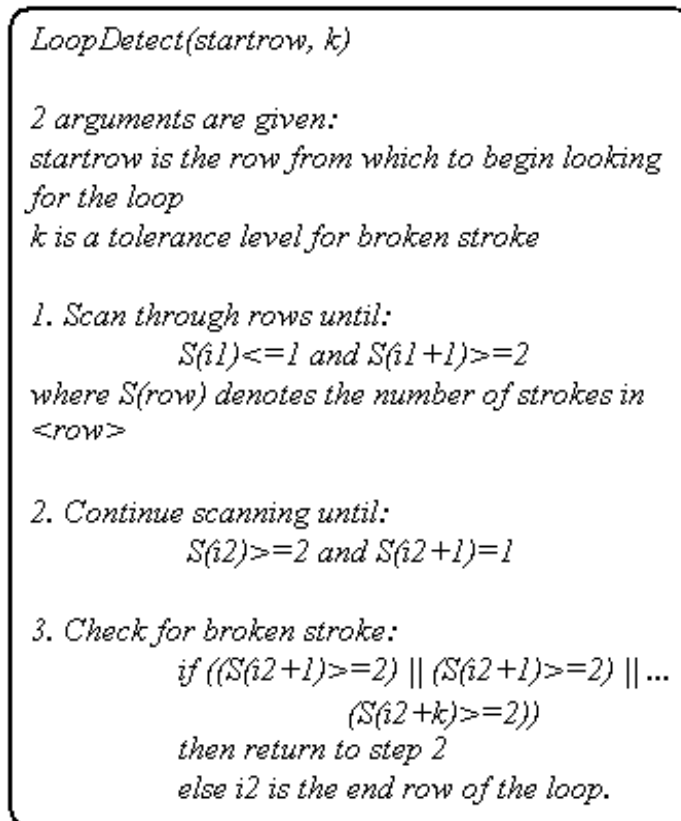


Figure 25. Algorithm for Loop Detection.

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 figure below shows a '0' that has been written where the loop is not closed, but the digit is still clearly a zero.

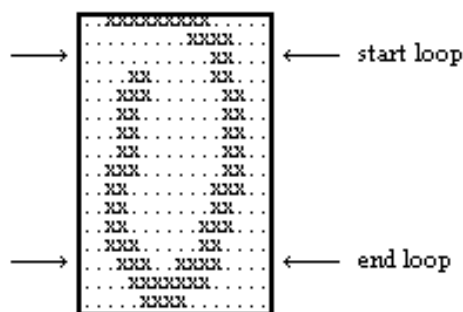


Figure 26. A '0' made with an open loop.

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.

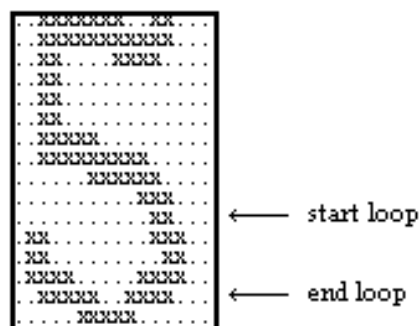


Figure 27. A '5'. Note the open loop on the bottom.

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. Connectivity between strokes is calculated by:

```
if ((BeginHoleStroke.start<=NextRowStroke.end+1) && (BeginHoleStroke.end>=NextRowStroke.start-1))
    then BeginHole Stroke connected to NextRowStroke.
```

where start and end denote the start and end of the associated stroke. Given this, open loops can be successfully detected and differentiated from closed loops.

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.

4.4.2. 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 figure below depicts this case:

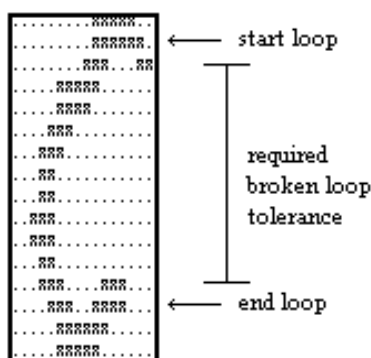


Figure 28. The broken loop tolerance for '0'.

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. The figure below illustrates various methods of writing '1's.

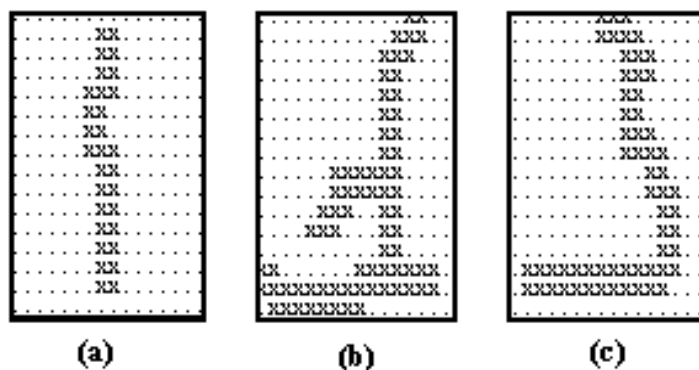


Figure 29. Samples of different styles of '1's.

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 figure below shows various cases of '5's that have to be handled.

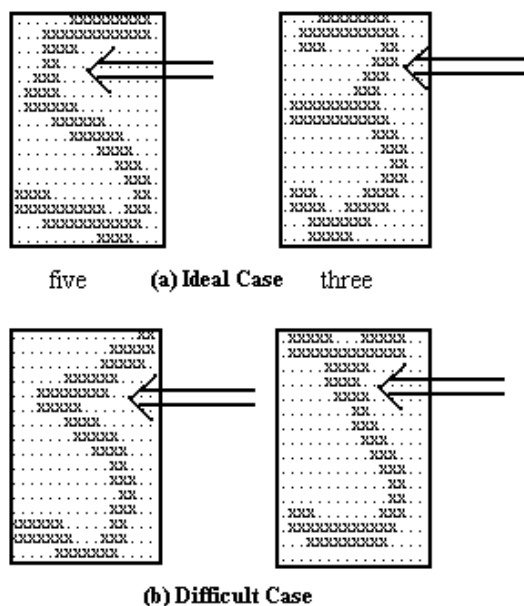


Figure 30. The ideal case showing the difference between '3's and '5's and the difficult case where even the human eye has trouble differentiating between '3' and '5'.

The patterns shown in the difficult case are hard to distinguish even with the human eye. Therefore, the task of differentiating using structural methods is too inexact to always be reliable. Secondly, the neural net architecture is given the benefit of the doubt. 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 effects 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 idiosyncrocies of neural network recognition can be identified and false recognition of characters can be avoided.

5. Performance Analysis

		Train Set #=3103	Test Set #=1444
MLP1	#correct	2950 (95%)	1158 (80%)
	#not. recog.	114 (3.7%)	139 (9.6%)
	#false pos.	39 (1.3%)	147 (10%)
MLP2	#correct	2960 (95.4%)	1149 (80%)
	#not. recog.	106 (3.4%)	146 (10%)
	#false pos.	37 (1.2%)	149 (10%)
DDD1	#correct	2962 (95.5%)	1222 (84.6%)
	#not. recog.	98 (3.2%)	139 (9.6%)
	#false pos.	43 (1.4%)	83 (5.7%)
DDD2	#correct	2944 (94.9%)	1240 (85.9%)
	#not. recog.	107 (3.4%)	113 (7.8%)
	#false pos.	52 (1.7%)	91 (6.3%)

In this section, the performance of the proposed recognition module is measured. To test the architecture, two trials were employed. First a set of testing samples were used; these samples have been taken from the Brazilian data and some outside sources, and the set is completely separate from the training data. The segments here were obtained from an unconstrained environment as noted and described previously, however, in fairness, the segments have been selected specifically to be tested on. 1444 segments were tested; see figure 19 for a per digit breakdown of the set. Using the testing set, the results at each stage of the recognition system are presented, illustrating the performance of each module employed. Secondly, recognition performance was taken directly from a set of Brazilian checks, and performance here was measured on a per field as well as a per digit basis. 144 checks were tested. First, the data for the testing set is presented.

Figure 31. The results on the training/testing sets for each network.

Number of Samples: 1444
 Number Correct: 1253 87%
 Number No Recognition: 96 6.6%
 Number of False Positives: 95 6.6%
 Average Confidence of a Correct Guess: 0.870235
 Average Confidence of an Incorrect Guess: 0.578269

ACTUAL VALUE

Digit	#tested	#cor.	#noRec.	#falsePos.
0	270	249	16	5
1	183	160	8	15
2	120	105	9	6
3	107	83	12	12
4	87	79	6	2
5	137	126	6	5
6	78	75	2	1
7	131	118	10	3
8	95	83	4	8
9	72	67	2	3
10	100	68	21	11
11	64	40	0	24

GUESSED VALUE

	0	1	2	3	4	5	6	7	8	9	10	11
0	249	0	1	1	0	1	1	0	1	0	0	3
1	0	160	8	4	1	0	1	1	0	0	0	1
2	2	1	105	0	0	1	0	2	0	0	0	2
3	0	1	2	83	0	5	0	3	0	0	1	3
4	0	0	0	0	79	0	0	0	0	1	1	1
5	0	0	0	1	1	126	3	0	0	0	0	1
6	0	0	0	0	0	0	75	0	1	0	0	0
7	0	0	0	1	1	0	0	118	0	1	0	3
8	0	0	1	0	2	0	2	0	83	2	1	2
9	0	0	0	1	2	0	0	0	0	67	0	1
10	0	0	0	1	3	0	2	4	1	0	68	6
11	3	3	1	1	6	5	6	10	4	4	7	40

As stated earlier, each of the networks in the recognition module was trained until a recognition rate over 95% was achieved when no rejection class is taken. The figure on the previous page shows the overall performance of each network on the training set when rejection class is taken and on the testing set. This figure shows that the DDD based networks have outperformed the standard MLPs by about 5% in the testing set. This verifies our assumptions about the feature extraction employed. The DDD feature has indeed increased the generalization and discrimination power of the network. From examining the table, the average recognition rate and false positive rate are 82.7% and 8% respectively. When the results produced by the arbiter are examined, they should at least match this standard. The results found from the arbiter output are produced below:

Figure 32. The outputs of the arbiter. (from left to right) The overall statistics of the arbiter outputs. Next, a chart of overall statistics for each digit. Finally, a detailed chart of the actual digits and all the resulting guesses.

	total	corr.	movedNR	incorr.	movedNR
0	69	50	16	5	1
1	38	18	0	12	7
2	41	27	6	5	0
3	40	18	4	10	0
4	28	20	2	2	0
5	39	30	0	3	3
6	17	14	0	1	0
7	37	24	9	3	2
8	31	21	1	6	1
9	31	26	2	3	1

Total Correct and Moved to Not Rec.: 40
 Total Incorrect and Moved to Not Rec.: 15

Final Results

Total Correct: 1213 (84%)
 Total Not Recognized: 151 (10%)
 Total Incorrect: 80 (5.5%)

As can be seen in the figure, the overall recognition rate is 87% and the false positive rate is 6.6%, an improvement in both cases over all of the individual networks. Next, the output of the structural processor is examined along with the final results.

Figure 33. The actions of the structural post processor and the final results . movedNR denotes values moved to not-recognized.

The structural processor took 40 correct inputs and switched them to not-recognized, while successfully identifying 15 incorrect inputs. Thus, while the recognition rate is reduced, so is the false positive rate. Considering the fact that this system is designed for the banking industry, error rates are extremely important to minimize. Therefore, recognition rate can and should be sacrificed to a certain extent. Nevertheless, 40 is a large number to have been misinterpreted, but it should be remembered that the recognizer is operating in an unconstrained environment where images can be of very poor quality. In such a case, many of the input segments may be considered barely recognizable upon sight, but the network module can still identify it. In general, if a segment is only barely identifiable, rejection is tolerated especially since it will aid in the case of initial false recognition. As long as the rules of the post processor are sound and the guideline that the heuristics used only loosely define a value is followed, the post processor is a useful module. The final results (84% correct, 5.5% error rate) do indeed show that the overall module has outperformed any and all of its individual pieces.

A recognition rate on a test set of 84% may be considered low in comparison to some of the other research provided. However, this is again a result of the unconstrained operation environment. The image below shows an example of a poor quality image that is common in this system and must be handled.

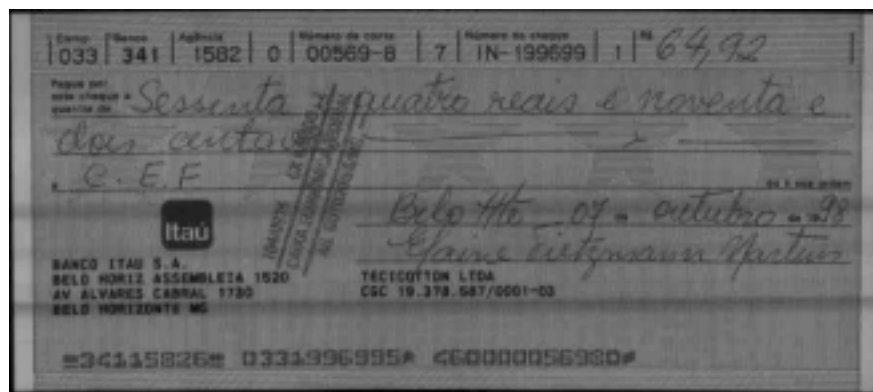


Figure 34. An example of a poor quality Brazilian check.

As is clear from the check, contrast between the foreground and background are low and the overall brightness of the character is dim. Starting from this initial poor quality, both thresholding and CAB location which handle the actual image will be degraded. Once this portion of the system has been effected, the performance of the rest of the system will be accordingly effected as well. Segments will be noisy or portions of the written data may get lost in thresholding causing broken segments; both cases produce cases where recognition becomes very difficult. Situations where image quality is poor, then, have been found to severely effect performance.

Next, the per check results will be discussed. As stated above, the system was tested on 144 Brazilian checks. The overall results are depicted in the graph below.

# checks	144		# digits	810	
# bad threshold	0		#extra characters	17	
# bad CAB location	3		#segment errors	47	5.8%
#error	7	4.86%	# false positive	26	3.2%
#reject	107	75%	# bad rejection	80	9.5%
#correct	30	20.8%	# rejection 140	17.3%	
			# correct	644	79.5%

Figure 35. Results of testing on Brazilian checks. The first graph shows per check data. The second graph shows the per digit data. Note. The segment errors are included in the recognition rate counted. Bad rejection denoted things rejected by the recognizer that should have been accepted, and rejection counts all segments that were rejected.

Per check recognition on these checks was 20.8% and the error rate was about 5%. With an 84% per digit recognition rate and a 5 % per digit false positive rate from the testing set and assuming about 5 characters per check on average, these results seem low. However, it must be noted that the recognition module itself is not the only factor that determines recognition rate. Segmentation errors, in addition to the already stated thresholding and CAB location errors, can reduce field recognition rate. For example, if extra segments (such as parts near or in the CAB which are not part of the amount or the segmentor fails to produce recognizable segments) are passed to the recognizer which it does not recognize, field recognition will not occur. Or worse, extra segments can look like actual digits and be falsely recognized through no fault of the recognition module. As far as field recognition goes, the entire application must be allowed to perform its job effectively. Given this, the performance results found are understandable.

6. Conclusion

This paper proposed a recognition module to fit inside a check reading application. The overall application handles the processing of the check image and the location of the segments that are the individual digits of the check amount. 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. Variations in width, height, slant and stylistic differences in handwriting all make recognition extremely difficult. Therefore, segments are normalized to a 16x16 binarized input for the recognition module.

The recognition module is split into a preprocessing module, a main module and a post-processing verifier. The main module is neural network architecture based on MLP networks. This structure was chosen over other

techniques (template matching, structural analysis) because of the comparatively greater success already achieved in this field using neural networks. The success of neural net architecture does not simply depend on the type of network used, but rather on the methods used to tailor the total architecture to the environment that it will operate in [18]. In this application, the recognition has accomplished this task through different methods. First, redundancy is utilized through an array of networks to both increase recognition rate and reduce false positive rates. Second, a version of the DDD feature extraction is used, tuning the networks to different information from the input segments and proving to increase the discrimination and generalization power of the respective neural nets and thus the overall architecture. Finally, the networks have been trained on segments gathered from the application's segmentor itself so that it can adjust to the specific nature of Brazilian checks as well as anticipate the output of the segmentor. Structural post processing is used to verify the recognized value through a version of loop detection and line detection. Values that are not verified are passed back to a feedback module for re-segmentation.

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 have clearly degraded the performance of the overall architecture. Better quality images with reasonable contrast and brightness are necessary to improve performance. Finally, the CAB location on Brazilian check is not standardized and the process of locating the CAB can result in extra segments input into the network which can be misinterpreted because of their similarity to actual digits. Thus, standardizing this location would be a great aid in increasing accuracy rate for the recognition module.

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. Thus, the recognition module presented here has been tailored to for the recognition of Brazilian checks. By constraining the possible inputs to digits plus special characters, the job of the recognition module is mitigated. Given this, the recognition module presented has utilized standard techniques in a unique way and has successfully taken advantage of the environment that it operates within. 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.

7. References

- [1] Denker, J. et. al. "Neural Network Recognizer for Handwritten Zip Code Digits." AT&T Bell Laboratories. p. 323-333.
- [2] Dey, S. Adding Feedback to Improve Segmentation and Recognition of Handwritten Numerals. MIT Thesis. 1999.
- [3] Duggan, M.G. "Enhancing Accuracy of Automated Numeral Recognition." MIT thesis. 1992.
- [4] ERA. "An Electronic Reading Automation." Electronic Engineering. 1957. P 189-90.

- [5] 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.
- [6] Fukushima, K. et. al. "Neocognitron: A Neural Network Model for a Mechanism of Visual Pattern Recognition." IEEE Trans. on Systems, Man and Cybernetics. Sept., 1983. p. 826-834.
- [7] Garris, M. D. et. al. NIST Form-Based Handwritten Recognition System: Release 2.0. US Dept. of Commerce, Technology Administration. National Institute of Standards and Technology. 1997.
- [8] Hannan, W.J. "RCA Multifont Reading Machine." Optical Character Recognition. Fisher, et. Al. ed. 1962. P. 3-14.
- [9] Haykin, Simon. Neural Networks: A Comprehensive Foundation. Prentice Hall. 1994.
- [10] Iijima, T. Theory of Pattern Recognition. Morishita Publishing. 1989.
- [11] Lee, S. and Kim, Y. "A New Type of Recurrent Neural Network for Handwritten Character Recognition." IEEE. 1995.
- [12] Lethelier, E., et. al. "An Automatic Reading System for Handwritten Numeral Amounts on French checks." IEEE. 1995.
- [13] Mori, Shunji , Suen, Ching Y. and Yamamoto, Kazuhiko. "Historical Review of OCR: Research and Development." Document Image Analysis. 1992. p. 244-269.
- [14] Nagendraprasad, M.V. et. al. "Algorithms for Thining and Rethickening Binary Digital patterns." International Financial Services Research Center (IFSRC) Discussion Paper.
- [15] Nagy, G. "Optical Character Recognition: Theory and Practice." Handbook of Statistics. Krishnaiah and Kanal, eds. 1982. v. 2. p. 621-649.
- [16] Oh, I. and Suen, C. Y. "A Feature for Character Recognition Based on Directional Distance Distributuion." Proceedings of 4th International Conference on Document Analysis and Recognition. ICDAR. 1997. v.1.
- [17] Rabinow, J. "Development in Character recognition Machines at Rabinow Engineering Company." Optical Character Recognition. Fischer, G., ed. 1962. p. 27-51.
- [18] Roman, Gupta, Riordan." Integration of Traditional Imaging, Expert Systems, and Neural Network Techniques for Enhanced Recognition of Handwritten Information." IFSRC. 1990. No. 124-90.
- [19] Shridar, M. and Badrelin, A. "Recognition of Isolated and Simply Connected Handwritten Numerals." Pattern Recognition. 19(1). 1986.
- [20] Srihari, S.N. et. al. "Name and Address Block Reader System for Tax Form Processing." IEEE. 1995.
- [21] Sparks, Peter L. A Hybrid Method for Segmenting Numeric Character Strings. MIT thesis. 1990.
- [22] Tsai, Wen-Hsiang. "Moment-Preserving Thresholding: A New Approach." Document Image Analysis. 1984. p. 44-59.
- [23] Zahn, C.T. and Roskies, R.Z. "Fourier Descriptors for Plane Closed Curves." IEEE Trans. Comput.

March, 1972. v. C-21. p. 269-281.

[24] Zhou, Jie, et. al. "A High performance Hand-Printed Numeral Recognition System with Verification Module." Proceedings of 4th International Conference on Document Analysis and Recognition. ICDAR. 1997. v.1.