170
95

A BACKPROPAGATION NEURAL NETWORK IN AN ADDRESS BLOCK CLASSIFICATION SYSTEM

by

Matthew Phillip Grzech

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

in

Electrical Engineering

APPROVED:

Fred C. Ricci, Chairman

Donald P. D'Amato

Hugh F. VanLandingham

May, 1991

Blacksburg, Virginia

A BACKPROPAGATION NEURAL NETWORK IN AN
ADDRESS BLOCK CLASSIFICATION SYSTEM

by

Matthew Phillip Grzech

Committee Chairman: Fred C. Ricci
Electrical Engineering

(ABSTRACT)

The U.S. Postal Service (USPS) is investing heavily in research and development of automated mail handling systems. A major component in these systems is the use of Optical Character Recognition (OCR) to read the destination address and ZIP Code, and then bar code the mail piece. High speed sorting equipment can then sort the mail using the bar code. Current USPS OCR/automated mail handling systems only process letter mail (no automated address-reading systems exist for nonletter mail). Moreover, these OCR systems only capture and read a restricted field-of-view image. Letters can be rejected by these OCR systems because of nonstandard address location (outside the field-of-view), skewed address lines, or handwritten addresses. Current research is working toward building OCR systems capable of processing all forms of mail which include letters, flats, and irregular parcel and pieces (IPPs). These systems must scan an entire mail image for the destination address block which can assume any orientation. For nonletter mail, such as magazines, this is an exceedingly difficult task, since the entire face of

up to 11 by 14 inches must be searched, and the address block must be chosen from all the other extraneous nonaddress information.

This paper details an experimental address block location system developed at MITRE. The system uses a backpropagation neural network trained to discriminate the frequency characteristics of address blocks from other candidates. The current system is trained on magazine flat mail.

ACKNOWLEDGEMENTS

I want to thank the MITRE management for having faith in my abilities, and giving me the funds to build this experimental system. I want to thank Russell Leighton for helping me with the *Aspirin for MIGRAINES* software, Jim Fisher for tolerating my exhaustive training and testing sessions, and Don D'Amato for his general guidance and support. Finally, I want to thank my mother and father, sisters and brothers, and friends for all their love and encouragement.

TABLE OF CONTENTS

LIST OF FIGURES

1.0 INTRODUCTION

The U.S. Postal Service (USPS) handled nearly 160 billion pieces of mail in 1989. The present and anticipated future scale of this operation offers strong incentives for the use of automation in the handling of mail.

The USPS plans to use automated address readers linked to automated sorting systems for its four categories of mail: Machine Readable Letters, Handprinted and other non-Machine Readable Letters, Flats, and Irregular Parcel and Pieces (IPPs). These four mail categories have a wide range of problems and degrees of difficulty in applying automated address finding algorithms. A major problem for automated address readers is the presence of extraneous features, such as advertising, on mail pieces. Such features present problems in the automated reading of addresses, because it is too time consuming to perform Optical Character Recognition (OCR) on all candidate features found in a large proportion of the mail stream. For this reason, prototype development efforts (sponsored by the USPS) are currently under way at the State University of New York (SUNY) [1-3], EKTRON Applied Imaging (a division of Kodak) [4-6], SRI International [7-9], and Elettronica S. Giorgio (ELSAG) [10,11]. These organizations are tasked to build real-time systems that can automatically locate the position, size, and orientation of the

destination address for mail of all degrees of complexity. However, none of the address block location techniques being used by these organizations employ neural networks.[1] (These organizations use traditional connected components [12] and component clustering techniques for address block location.)

An experimental address block location system has been developed at the MITRE Corporation. This work was funded as a MITRE Sponsored Research (MSR) project[2] and was conducted between July and October 1990. This system was developed to:

- Demonstrate the feasibility of using neural networks in address block location
- Determine what could be accomplished using in-house tools (ie., workstations, image scanners, programming languages, image processing tools, neural network development tools)

This system uses a backpropagation neural network trained to recognize candidate address blocks based on the magnitude of their two-dimensional Fourier transforms. The scope of this system during training was limited to magazine flat mail for several reasons:

- There are no fully automated systems for handling magazine flat mail

1 SRI International [9] used a backpropagation neural network to rank candidate address blocks based on block location and profile descriptors. However, the candidate address blocks were located using connected components analysis.

2 On a yearly basis, MITRE employees can compete for research funds that are set aside under the MITRE Sponsored Research program.

- Since this system is a feasibility test, the system should be limited to machine printed address blocks (addresses on magazines are rarely handwritten)
- Given magazine size and complexity compared to letter mail, the capabilities of the system could be tested in terms of processing time and accuracy

An examination of magazines reveals that the lines of text in the destination address block (whether the address is on a white label or the image background) are aligned with one of the four boundaries of the image. Using fifty magazine samples, the maximum observed deviation of the destination address block from this boundary alignment was less than 3.5 degrees. If overlapping sub-images roughly the size of an address block are extracted in both a horizontal and vertical orientation relative to the top/bottom and the side boundaries of a magazine image, a sub-image will be found such that the destination address block falls within its boundaries. The experimental system developed at MITRE uses such an approach to extract sub-images prior to their Fourier transformation and subsequent neural network classification.

This paper reports on the address block location system developed at MITRE. The organization of this paper is as follows. Section 2 is background in current methodologies (and the work of research organizations) in address block location.

Section 3 provides a brief overview of neural networks and backpropagation training. Section 4 gives a system overview, system technical considerations, tools used, and source code description. Section 5 discusses the collecting and preprocessing of training data. Section 6 discusses two network topologies along with the training results. Section 7 gives test results, Section 8 is the conclusion, and Section 9 remarks on issues for further research.

2.0 BACKGROUND

A typical mail piece has several regions or blocks that are meaningful to mail processing, for example, address blocks (destination and return), postage (meter mark or stamp), as well as extraneous blocks such as advertising text, logos, and graphics. The most difficult task in address processing systems is in locating the destination address block among all the other candidate regions and then extracting the ZIP Code.

The USPS employs approximately 400 Multi-Line Optical Character Reader (MLOCR) systems in its mail facilities around the United States. These machines process a restricted field-of-view image on letter mail only. Currently, about 45 percent of the letters are automatically sorted by these MLOCR devices.[3] A letter may be rejected by the MLOCR because of nonstandard address block location, poor character quality (including image background noise), address block skew (rotation), or handwritten address. The letters that are not automatically sorted, as well as flats, are presented by a transport mechanism to a human operator who manually keys in the destination ZIP Code. To improve the performance of present MLOCRs, as well as future Postal OCRs and handprinted address

3 The USPS has a goal to achieve nearly 100 percent automation by 1995, using a combination of systems including improved MLOCRs, various semi-automated off-line systems, and wide-area bar code reading systems.

recognition devices, the USPS is currently funding the four organizations previously listed to build prototype real-time address block location systems. These prototype systems must be capable of processing twelve letters per second, five flats per second, and one IPP per second.

2.1 Current methods in address block location

In current experimental address block location systems, after an image has been captured and binarized[4] , algorithms are performed to segment the image into regions of text and non-text. Candidate address blocks are chosen based on knowledge about the size and structure of text within an address block. The candidate address blocks are ranked using structure and location statistics and the destination address block is selected based on these rankings.

In general, mail images can be captured in the following ways: gray-scale, color (a composite image of red, green, and blue filters), infrared, or color under ultraviolet illumination [3]. Algorithms for binarizing gray-scale or infrared images (which are single spectral) are different than algorithms for binarizing multispectral color images. Gray-scale and infrared images are binarized using adaptive thresholding algorithms [13] that employ some form of local contrasting (i.e, sliding window

4 Each pixel in a binarized image has two levels or intensities.

functions). Local contrasting is used to reduce the effects of nonuniform illumination that frequently occurs during image capture. In some cases, edge enhancement algorithms [14] (i.e., gradient operators such as the *Laplacian* filter) may be performed prior to the adaptive thresholding in order to highlight strokes in machine printed and handprinted text. On the other hand, more sophisticated algorithms are required to binarize color images (or any other multispectral composite image) [15]. A multi-dimensional histogram is formed from the pixels in the multi-spectral image. Thresholding is performed by identifying clusters in a multi-dimensional space. The image can be segmented by assigning one intensity to pixels whose spectral components are closer to one cluster and another intensity to the other pixels in the image. The principal of cluster seeking becomes more complex as the number of spectral inputs are increased.

Algorithms used to locate candidate address block text in binarized mail images come from a broader class of techniques applied in text/image segmentation [16,17]. The segmentation is performed using either top-down or bottom-up methods [17]. Top-down methods use algorithms that recursively segment larger regions into smaller ones (e.g., the Run-Length Smoothing Algorithm (RLSA) [18,19], and projection profile cuts [20,21]).

Bottom-up methods initially group pixels together as connected components and progressively merge these components into larger regions (e.g., connected components analysis [3,16], and graph-theoretic minimal spanning trees [6,22,23]). While top-down approaches are generally faster, bottom-up approaches are more resistant to noise (salt and pepper noise) and text-line skew. Given the enormous variabilty in the quality of binarized mail images, bottom-up segmentation methods are traditionally used.

After components in an image are separated, unary tests determine maximum and minimum height and width of each component and the number of pixels in each component. Binary component tests check the distance between components and the difference in the number of pixels in each component. The data extracted on each component are stored in some form of linked-list (i.e., a data structure that contains component profiles, numbers of pixels, position coordinates, nearest-neighbors, etc.). This information is used by a knowledge-based system to locate areas in an image that possess characteristics similiar to an address block. The system correlates knowledge about typical address block character/component size, and inter-character, inter-word, and inter-line spacings (frequencies) with candidate areas in the image. Information that may be used to distinguish the destination address block from other candidate blocks are:

(1) block position; (2) line justification; and (3) strategic component position (e.g., relative position of the ZIP Code in an address block).

The Document Image Understanding Group at SUNY (Buffalo), which is highly regarded in this area of research, is building a prototype address block location subsystem (ABLS) which uses a blackboard model problem-solving approach [3]. The basic idea of this approach is to divide a complex problem into loosely coupled subtasks; each subtask is attacked by a specialized tool that is called a knowledge source (KS). A blackboard model is usually composed of three major components; KSs, the blackboard data structure, and the control mechanism. The KSs consist of a community of independent experts or tools that incrementally contribute information to the overall solution on the blackboard. Communication between KSs is exclusively through the blackboard. The blackboard data structure is a global database representing the current state of the problem. The blackboard data structure is derived from candidate address block location statistics. These statistics are extracted differently for the four categories of mail. The first step in extracting these statistics is to perform color-thresholding on flats or gray-scale thresholding otherwise. Then a bottom-up segmentation is performed to determine the connected components in the image.

Height and width unary tests are then performed on each component followed by binary tests to determine the relationship between components. Finally, the control mechanism controls the invocation of KSs (each KS uses a part of the blackboard information) and is responsible for integrating information on the blackboard and selecting the proper destination address block.

A key aspect of the address block location work being performed at EKTRON [6] is the use of a graph-theoretic point-clustering Minimal Spanning Tree (MST). This method enables the separation of features into clusters, from which a number of candidate address blocks are identified. Statistics drawn from each cluster are used to characterize candidate blocks and to rank them in order of likelihood that they correspond to the destination address.

2.2 Neural networks in text/non-text classification

Research recently performed at MITRE [24] indicates that machine printed text may be linearly separable from non-text in some U.S. Patent and Trademark Office (PTO) patent document images. This research was based on the use of backpropagation neural networks trained to distinguish the two-dimensional power-spectral frequency characteristics of eight and ten point oriented machine printed text from non-text in these document

images (the non-text images were composed of high and low density line drawings, salt and pepper noise, and white space).[5] In a comparison of the various neural networks trained, it was found that a single layer neural network linearly separated text image data from non-text image data in test documents. The MITRE Address Block Location system (which is the subject of this paper) uses a similiar classification approach. However, the inputs of these neural networks are sized to fit a typical address block, and the connection weights are trained to discriminate address blocks from other potential candidates.

5 This research was independent of MITRE's work in document image segmentation [19].

3.0 LAYERED NEURAL NETWORKS

A layered neural network consists of several distinct layers of processing nodes. An input layer i_i is used to gather external signals, and an output layer o_j is used to transmit signals to the external environment. These symbols are used to refer to either the node activation or the node output signal, or both [25]. In pattern recognition terms, external inputs correspond to *pattern features* and outputs correspond to *classes*.

A layered network may also contain one or more hidden layers. Hidden layers are used to represent domain knowledge and to solve problems [26]. A layered neural network with a single hidden layer is illustrated in Figure 1. Hidden nodes are typically represented by the symbol notation h_k. Generally, nodes in one layer communicate to all nodes in the adjacent layer through unidirectional synaptic links. In a three layered network, a synaptic link between an input node i_i and a hidden node h_k has a weight v_{ik} associated with this link. Moreover, a synaptic link between a hidden node h_k and an output node o_j has a weight w_{kj} associated with this link. Weights measure the degree of correlation between the activity levels of the nodes they connect [25].

A node receives a net signal input which is the weighted sum of all its inputs. Since the learning algorithm (which is

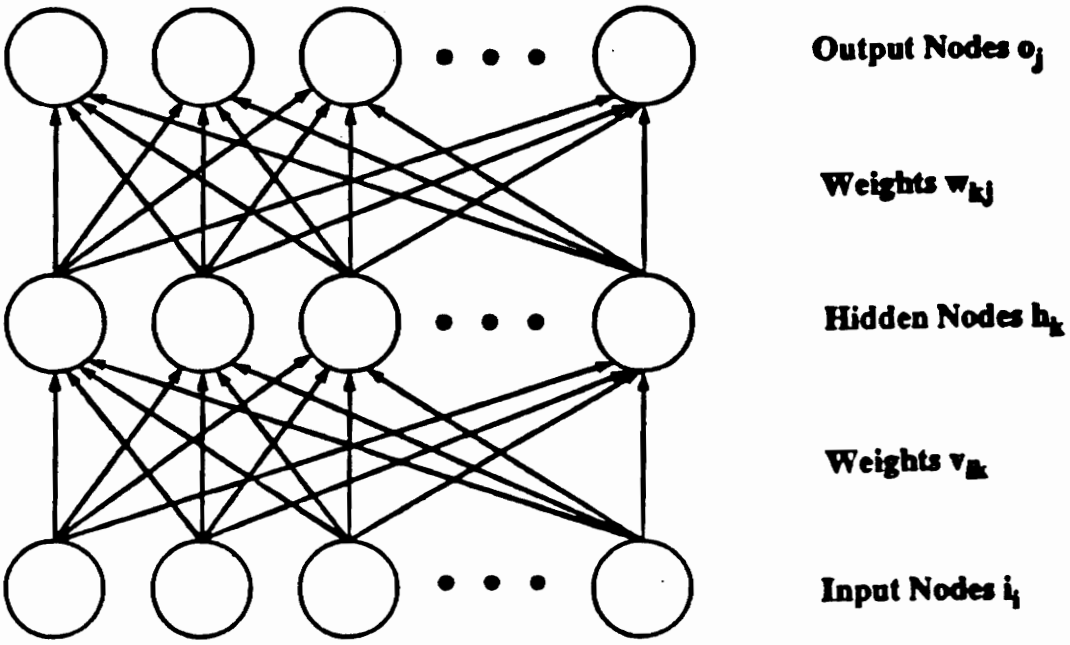


Figure 1: A Layered Neural Network

described later) requires that the output of a node be differentiable, the following logistic output function for the hidden nodes and the output nodes, respectively, are generally used:

Hidden Nodes $$h_k = \frac{1}{1 + \exp(\Sigma v_{ik} i_i)} \quad (1)$$

Output Nodes $$o_j = \frac{1}{1 + \exp(\Sigma w_{kj} h_k)} \quad (2)$$

3.1 Backpropagation training

During the testing of a layered neural network, information flows from input to output. Since the weights of a network are initially undetermined, a learning process is needed where error information propagates in reverse, from the output through the hidden layer(s) to the input layer. This error information during the learning process is used to modify the weights. The error signal at the output is defined by:

$$\varepsilon = 0.5\ \Sigma(t_j - o_j)^2 \quad (3)$$

where o_j represents the observed output, t_j represents the target output, and ε represents a scaled summation of the error squared over all outputs. The partial derivative of ε with respect to a network weight defines a *credit assignment rule* [25] which is used to modify the weight. Credit is assigned proportionately to those weights most responsible for the error, thus ensuring *gradient steepest descent* in the weight space.

The network is initially primed with random weights. A pattern is then presented and a value δ_j is computed for each pattern and is defined by:

$$\delta_j = o_j(1-o_j)(t_j-o_j) \tag{4}$$

Weights are changed according to:

$$\Delta w_{kj}(n) = \alpha\delta_j h_k + \beta\Delta w_{kj}(n-1) \tag{5}$$

where α is the *learning rate* and β is the *momentum* which reduces sharp changes in the weight space. A fraction β of the weight change from the previous (n-1) pattern is added to the current weight. This *momentum* term functions as a low pass filter which reduces dramatic changes in the weight space and also facilitates a *gradient steepest descent* in the weight space.

When δ_j values for the output nodes have been computed, the δ_k values for the hidden nodes can be computed according to:

$$\delta_k = h_k(1-h_k)\Sigma\delta_j w_{kj} \tag{6}$$

Similarly, the weight correction for v_{ik} is:

$$\Delta v_{ik}(n) = \alpha\delta_k i_i + \beta\Delta v_{ik}(n-1) \tag{7}$$

Weights can be adjusted in two ways. They can be adjusted after each presentation (*iteration*) of an input pattern. Or, they can be adjusted after the presentation of the entire set of patterns (*epoch*) where the accumulated change is kept for each weight. Also, patterns separate from the training set can be used to test the accuracy or generalizing ability of the network.

3.2 Discrimination capacity of layered networks

Layered neural networks classify input patterns by partitioning the input space with hyperplanes [26]. Consider Figure 2, which shows a four-point function represented geometrically. A *threshold function* corresponds to a line that separates the points that map to +1 (labeled X) from those that map to -1 (labeled O). Certain functions can be separated by a line (e.g., Figure 2(a)), while other functions cannot (e.g., Figure 2(b)). In higher dimensions, the separation is done using a hyperplane. The discrimination capacity of a node, which can be viewed as a simple threshold function, is measured by the maximum number of points that can be separated, in every possible way using a hyperplane [27]. If a threshold function is not found without any hidden layers, then it is possible that one or more hidden layers may define a hyperplane that finds a threshold function.

A single hidden layer allows *AND* operations of the half-spaces formed by the hyperplanes of the first layer of weights. This produces convex regions in the space. A second hidden layer can *OR* the convex regions to produce arbitrarily shaped regions with concavities and holes. Therefore, two hidden layers are sufficient to form regions or clusters of any shape [26].[6]

---

6 Contrary to this theory, however, two hidden layers are not necessary for arbitrary shapes, as Wieland and Leighton [28] have demonstrated. Their method forms arbitrary shapes by linear superposition in a single hidden layer.

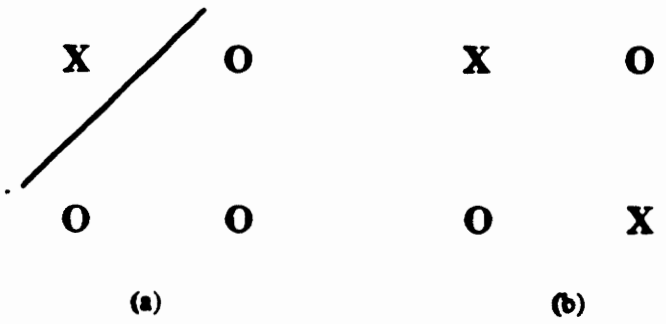


Figure 2: Separable and Unseparable Geometric Functions

It is the contention of this author that there is one overwhelming feature or several lesser distinguishing features that can be used to discriminate a destination address block from other candidate regions in a mail image. These features can be processed by nodes that have learned to discriminate and form hyperplanes between these regions. Since an address block of any font size/style or line spacing has a characteristic periodicity of characters, words, and lines, these distinguishing features can be found in the two-dimensional Fourier transform of the address block image. Also, other characteristics such as the placement of the ZIP Code and the justification of address lines may be important features in the Fourier transform that a neural network uses in the classification of an address block.

4.0 TECHNICAL APPROACH

If all the characteristics which describe a mail image were understood, then the development of a knowledge-based system for locating the destination address block would be sufficient (if the solution is not too time consuming). Since there is an incomplete understanding of the mail image features and the relationship among these features, a properly configured neural network system trained on a large sample of mail images would be better suited to this problem. The neural network would learn to identify and associate all the characteristics which describe a destination address block. Moreover, the neural network approach would fit a parallel implementation better than the knowledge-based approach.

A system of this kind would have been trained on the local characteristics of an address block through one or more input sources. These sources may include two-dimensional frequency measures, one or two-dimensional histograms, pixel densities, etc. A mail image would be separated into overlapping sub-images, and each sub-image would be classified by this neural network using these input source descriptors. This stage of the system would isolate the most likely candidates. These candidates may include the destination address, the return address, and possibly other sub-images with characteristics

similar to an address block. The destination address block could be singled out using a second stage neural network similar to the clustering network of Mulgaonkar and Bergman [9]. Conversely, information about block position, profile statistics, and inter-block object placement, such as the placement of the ZIP Code, may be used to identify the destination address block. Locating a ZIP Code would isolate the destination and return address blocks from other candidates. (Segmenting the ZIP Code would require a connected components procedure.) Finally, after the field of candidates has been diminished, the most likely destination address block would be the one nearest the center of the image.[7]

This system would ideally process all forms of mail (letters, flats, and IPPs) that contain both machine printed and handwritten addresses. Toward this end, MITRE's address block location system provides the initial groundwork for such a system. The current configuration of the system explores the use of neural networks trained on address block Fourier amplitudes in magazine flat mail exclusively.

4.1 Magazine flat mail characteristics

It has been found that the destination address block on flat mail, in general, is printed on a white background 84 percent of

7 Statistically, the destination address block will appear closer to the center of the image than the return address in more than half of all mail images where both destination and return addresses are present [7].

the time [3]. More importantly, the destination address on magazines (a subset of flat mail) almost always appears on a 3.5 by 1 inch white label (the address is rarely printed on the cover) and is always machine printed. Also, the font size in the address block is either 10 or 11 point.

A cursory examination of magazines reveals that lines of text in the destination address block (whether the address is on a white label or not) are usually aligned to the horizontal or vertical edges of the magazine. A study of 3000 flats[8] by the Georgia Institute of Technology Research Institute (GTRI) revealed that the destination address block was skewed (deviation from the horizontal or vertical edge alignment) by five degrees or more in 10.5 percent of the examples [29]. MITRE recently tested fifty magazines and found a maximum deviation from this alignment of 3.5 degrees. These factors influence both the size of the scanning sub-image window and the orientations of the magazine image during the scanning process.[9]

4.2 System overview

A diagram that illustrates the major steps in the MITRE address block location system is shown in Figure 3, and concluded in Figure 4. The image being processed is scanned vertically and

8 18.7 percent of the flats tested were magazines.

9 In general, the system that will soon be described does not discriminate between an address that has been printed on a label or directly on the magazine cover.

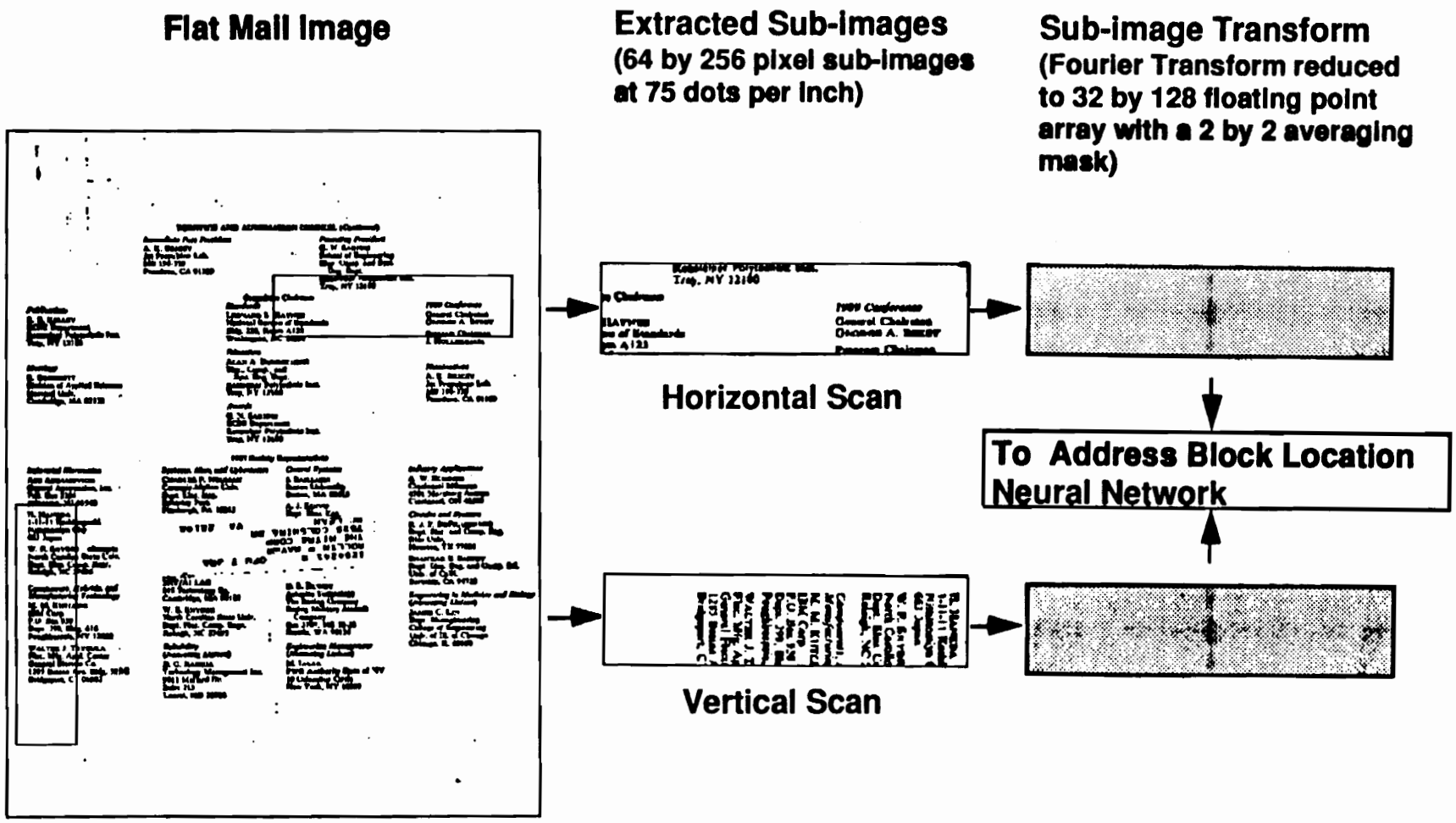


Figure 3: System Functional Overview

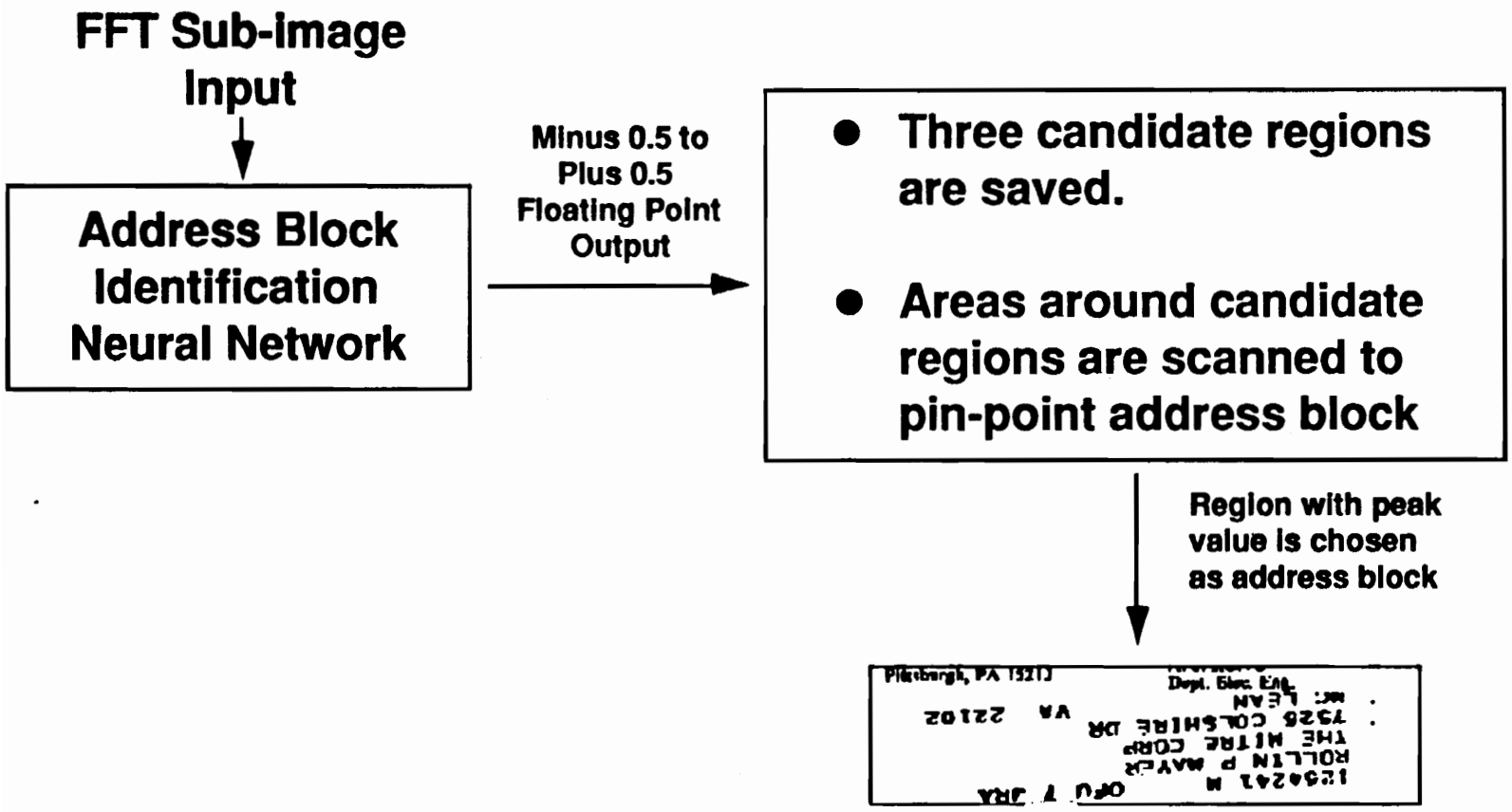


Figure 4: System Functional Overview (concluded)

horizontally in a rough fashion to extract 256 by 64 pixel sub-images for classification. (The vertical scan is accomplished by rotating the image ninety degrees and scanning it horizontally.) The sub-images are then preprocessed by computing the magnitude of the two-dimensional FFT. 2 by 2 averaging masks are then used on these two-dimensional Fourier magnitudes to lower their resolution to 128 by 32 floating point magnitude arrays. These arrays are then passed to the backpropagation neural network for classification. The neural network responds with a single number between -0.5 and 0.5 for each array. The confidence that the system detects an address block sub-image increases as the output of the neural network approaches 0.5. Three sub-images with the highest output values are saved as candidate locations. The destination address block is then located when a fine scan is performed around the three candidate regions.

4.3 System technical considerations

There were many factors that shaped the current implementation of MITRE's Address Block Location system. Important considerations that are reflected in the system and source code are sub-image capture size and image resolution, scanning sub-image overlap (coverage), image normalization, and floating point processing speed.

4.3.1 Sub-image capture size and image resolution

Since this system (in its current configuration) is trained to locate the destination address block on magazine flat mail, the sub-image capture size (for subsequent processing and classification) corresponds roughly to the size of a typical magazine address label (i.e., 3.5 by 1 inch, Section 4.1). However, the actual size (which is near this ideal) was dictated by the resolution of the scanned images. The relationship between the sub-image capture size and the image resolution is based on the input requirements of the two-dimensional FFT in this system. Briefly stated, this FFT uses the *successive doubling* method [30] which requires the input dimensions to be a power of 2.

Images were scanned at the de facto standard resolution of 300 dots per inch (dpi). This resolution was initially used in the training and testing of the system. Given this resolution, the sub-image capture size was 1024 by 256 pixels, which is 3.08 by .85 inches. This size satisfies the dimension requirements of the FFT used in this system. However, during testing, it was found that the performance of the system was hindered by the resolution of this sub-image. An FFT on a 1024 by 256 pixel sub-image required a little more than one minute on a *Sun 4/280* workstation. It required several hundred minutes to process a magazine image because of the nearly 200 sub-images that are

extracted, preprocessed, and classified by the system. This was not acceptable for research and testing purposes.

The organizations participating in the USPS-supported Address Block Location research have apparently discovered the same problem. All of these organizations are performing candidate address block segmentation at spatial sampling frequencies below 100 pixels per inch (ppi). In fact, ELSAG [10] has shown reasonable success at 30 ppi. While candidate address block post processing may require higher resolution images (e.g., OCR on candidate blocks to identify ZIP Codes, thereby narrowing the candidate field to the destination and return addresses), it is not necessary to read individual characters to identify text block areas. These areas can be located based on the physical nearness of characters, words, lines, and possibly stroke thickness. However, as the resolution becomes lower, there is a point where the distinction between characters, words, and lines no longer exists. In MITRE's research, this minimum resolution was not explored. However, the images were evenly down-sampled by a factor of four in both image dimensions. Thus, the sub-image input to MITRE's system was reduced to 256 by 64 pixels at 75 dpi. Example images of the same address block are shown at 300 dpi and 75 dpi in Figure 5. The characters in the 75 dpi image (Figure 5(b) have slightly thicker and less refined strokes compared to the 300 dpi image (Figure 5(a)). However, the

```
*****************5-DIGIT   22102
ED4535209 2110 121       3026
M.GRZECH R&D MGT         TEP
CIVIL SYS DIV Z5          04
                          10   D
7525 COLSHIRE DR
MCLEAN VA                 22102
```

Figure 5(a): 300 dpi address block

```
*****************5-DIGIT   22102
ED4535209 2110 121       3026
M.GRZECH R&D MGT         TEP
CIVIL SYS DIV Z5          04
                          10   D
7525 COLSHIRE DR
MCLEAN VA                 22102
```

Figure 5(b): 75 dpi address block

character, word, and line distinction still remains in the 75 dpi image.

The processing time was reduced significantly by lowering the input resolution from 300 to 75 dpi. This improvement came primarily from the reduced computational complexity of the FFT, which consumes more than 75 percent of the Address Block Location processing time. Expressed mathematically, the computational order of the two-dimensional FFT in terms of the number of complex multiplications and additions is [14]:

$$O(M,N) = MN(\log_2(M) + \log_2(N)) \tag{8}$$

where:
$O(M,N)$ = computational order
$M = 2^i$
$N = 2^j$
M and N are dimensions, i and j are positive integers

Using this equation, the factored reduction in the computational order of a 300 dpi image compared to a 75 dpi image is:

$$\frac{O(1024,\ 256)}{O(256,\ 64)} = 20.6 \tag{9}$$

While the FFT only improves logrithmically in terms of the number of floating point operations as the resolution is reduced, the complexity for the remainder of the routines in the system improves linearly.[10] Moreover, the system processes a 75 dpi magazine image in 4 to 8 minutes (processing nearly 200 sub-images), depending on the workstation used.

10 The improvement in speed was observed to be around a factor of 30 for a 75 dpi image.

4.3.2 Scanning sub-image overlap

Critical elements of MITRE's Address Block Location system are the horizontal and vertical increment parameters used in the continual re-positioning of the sub-image input window. The ideal scenario in terms of coverage is to move the sub-image input window in one pixel increments in both horizontal and vertical directions. Assuming that the sub-image window is the same size as the address block, the sub-image window would be guaranteed to fall directly over the address block at some point. However, the ideal scenario is too costly in terms of processing time. Therefore, a set of increment parameters should be chosen such that the performance tradeoffs are acceptable in terms of speed and accuracy.

An interesting feature of the Fourier transform is the translation independence of its magnitude:

$$f(x - x_0, y - y_0) \Leftrightarrow F(u,v) \exp[-2j\pi(ux_0 + vy_0)/N] \qquad (10)$$

$$| F(u,v)\exp[-2j\pi(ux_0 + vy_0)/N] | = | F(u,v) | \qquad (11)$$

The Fourier magnitude will be identical for any address block image $f(x,y)$ that falls anywhere within the bounding limits of the transform. Even if a small portion of the address block were outside these bounding limits, the magnitude of the two-dimensional Fourier transform would not change drastically. Moreover, the representative spectra of an address block $f(x,y)$ would change gradually as less of the address block appears in

the sub-image. Thus, the increment parameters can be larger than would otherwise be possible if this feature of the Fourier transform were not present.

The increment parameters are currently user adjustable (i.e., command-line). The rough scan (Section 4.2) uses these increment parameters. Rough scan parameters that were equal to half the sub-image dimensions (i.e, 128 and 32), were used in this research project. (The fine scan ratio is fixed and is based on one quarter the distance of the rough scan parameters. The area around the candidate region is scanned to a distance equal to the horizontal and vertical rough scan parameters.) Figure 6 shows a worst case scenario using these rough scan parameters. This scenario shows that the sub-image window would miss the address block by one quarter in both dimensions. This would result in only a 56 percent coverage during rough scan.[11] The *Region of possibility* area in Figure 6 spans the realm between the best and worst case scenarios. At these rough scan parameters, the upper left corner of the sub-image window could fall anywhere within this box. In the best case scenario, the sub-image window would fall directly over the address block during rough scan. In any case, the fine scan is used to correct

11 This scenario is based on the assumption that the sub-image window is the same size as the address block. In this system, however, the sub-image is slightly smaller than the address block, and therefore, coverage is a little better.

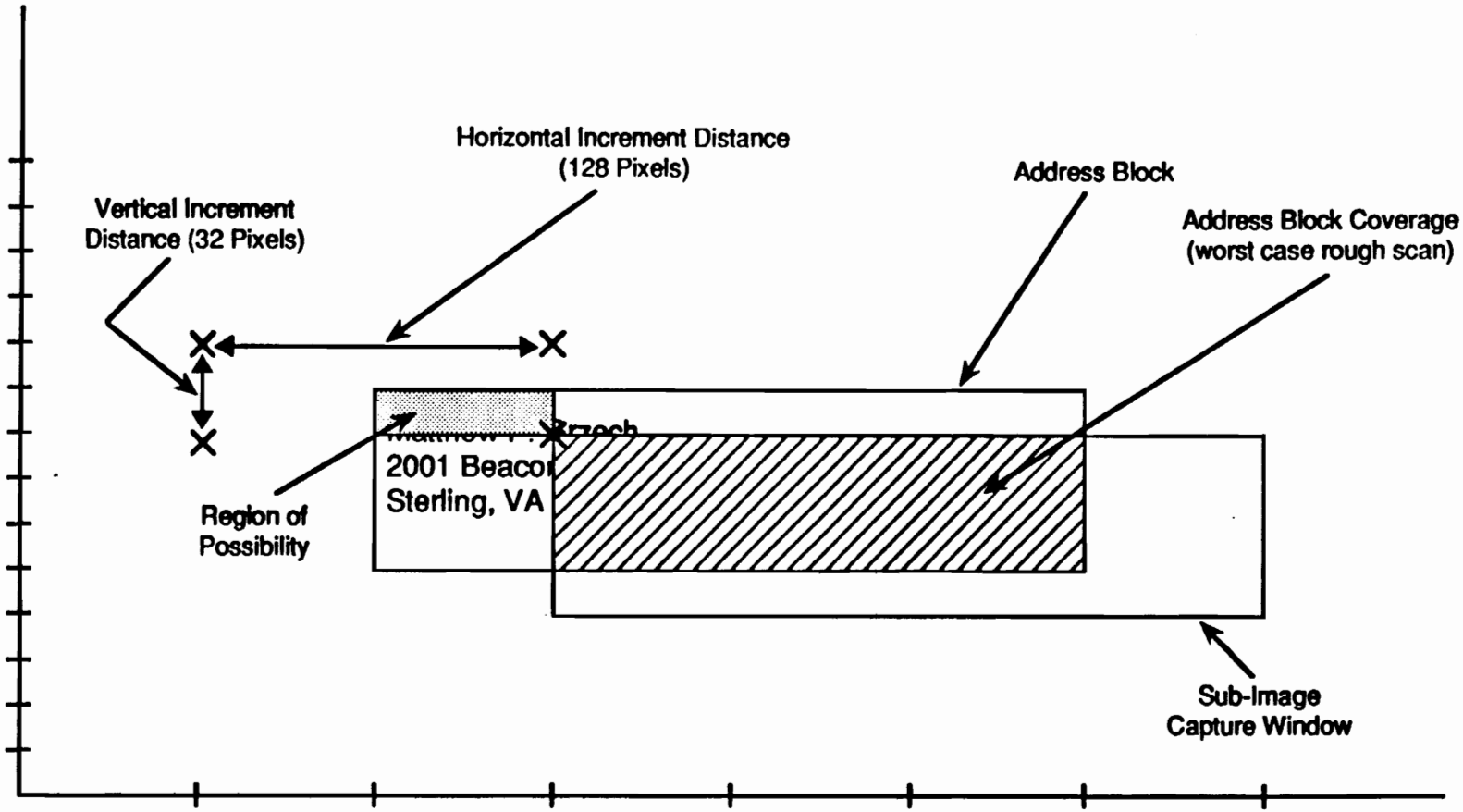


Figure 6: Worst case sub-image coverage of address block

these coverage problems at minimal cost (only three candidate regions are selected for fine scanning).

4.3.3 Image normalization

The image of a magazine address label can appear in normal or inverse form after the magazine has been captured and binarized. This contrast is shown in Figure 7(a) and 7(b) for an address label that was scanned on two different devices. This may occur due to the varying reflective properties of magazine covers and address labels under different circumstances (i.e., image capture systems that have different illuminations, contrast settings, adaptive thresholding algorithms, etc.).

The system should not be concerned with the background level (black or white) of the address labels. This system should be more concerned with the transitions between the black and white pixels. This can be accomplished by normalizing the image such that the Fourier magnitude of an image and its inverse are equal. The binary pixel values must be chosen such that the black pixel value is the negative of the white pixel value. (The Fourier magnitude of an image $f(x,y)$, and its inverse, $-f(x,y)$, are equal [14].) This implies that the values must be polar (e.g., $\{1,-1\}$, $\{0.5,-0.5\}$, $\{-5,5\}$, etc.). The values used in this system were $\{1,-1\}$.

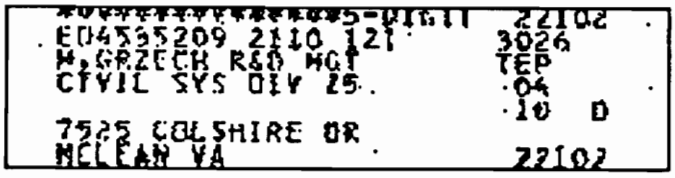


Figure 7(a): 75 dpi image binarized normally

Figure 7(b): 75 dpi image binarized inversely

4.3.4 Floating point processing speed

In terms of training and testing processing speed, a very important consideration in MITRE's system is the use of a floating point processor (since the majority of routines in this system are floating point intensive). Floating point arithmetic (single- or double-precision) is implemented on an integer processor (e.g., central processor) using shift and add or subtract sequence operations. Depending on the computer, this implementation can require up to several hundred machine cycles per floating point operation. In contrast, a floating point processor (in conjunction with a central processor) can typically perform the same operation in one machine cycle (which is roughly one to three orders of magnitude faster, depending on the computer). If a 10+ MIPS workstation with a floating point processor were used to implement MITRE's experimental system, it would require less than ten minutes to locate the address block on a magazine cover. This same workstation without a floating point processor would require several days to locate the address block.

4.4 Tools

The MITRE address block location software currently runs on *Sun Microsystems 4/XXX* workstations. However, the source code, which is written in the *C* programming language, can be compiled to run on any *UNIX* based system.

The current address block location software is called within an executable *Sun 4/XXX UNIX* script file. This script file performs three tasks:

- it creates a ninety degree rotated version of the original image;
- it provides the input arguments to the address block location software (these arguments include the original and rotated image names along with the scanning increment parameters discussed in Section 4.3.2); and,
- it uses the coordinates from the address block location software to extract and display the address block from either the original or rotated image.

Simple commands from the HIPS[12] [31] library of image processing routines are used to rotate the original image and display the extracted address block.[13]

The neural network tool used in this system is a public domain research and development software package created at MITRE called *Aspirin for MIGRAINES* [32]. This software package simulates general feed-forward neural networks. *Aspirin* is a

12 HIPS is a general purpose image processing package available through the University of New York.

13 The original image must have a HIPS image header before it can be processed by the script file (the HIPS header gives the gray-level format and size of the image). Currently, original images are captured in a *Sun* raster format (discussed later in this section) and are converted to HIPS format using the HIPS *suntohips* routine.

declarative language which is used to describe arbitrarily complex neural networks. An *Aspirin* description of a network is used to generate a computer program to train and simulate that network. Using *Aspirin* link libraries, the network code can be incorporated in user applications written in the *C* programming language. *MIGRAINES* (MITRE Interactive Graphic Research and Investigation Neural Network Emulation System) is a graphical interface for evaluating and interacting with a neural network. *MIGRAINES* has intentionally been kept separate from *Aspirin* so that the limitations of *MIGRAINES* do not restrict the performance of *Aspirin*. However, in practice, *Aspirin* and *MIGRAINES* are used as a single, cohesive unit. *Aspirin* has been ported to most *UNIX* based workstations. *MIGRAINES* will work only on workstations that use the *NeWS* windowing software (i.e., *Sun Microsystems 4/x* workstations and *Silicon Graphics* workstations).

Magazine flat mail images for training and testing were captured on a *MICROTEK 300GS* [33] gray-scale scanner. These images were captured in binary format using the default *brightness* and *contrast* settings of this scanner. The images were captured at the de facto standard resolution of 300 ppi and stored in the *Sun* raster format. The images were down-sampled (reduced in resolution) to 75 ppi for subsequent processing. The magazine images used in training were all near the standard page size of 8.5 by 11.0 inches. Therefore, the input area used

during image scan was set to a width and height of 2550 by 3300 pixels.[14]

4.5 Source code

A detailed description of MITRE's Address Block Location source code is in Appendix A, and a listing of this code is in Appendix B. The Address Block Location program is called within the *UNIX* executable script file in Appendix B.1. (The program is called *abl_nn_class75demo* in the script file.) The details of the script file are discussed in Section 4.4.

The main source code listing is in Appendix B.2. The program is written in the *C* programming language. Moreover, the program uses the HIPS [31] commercial software image header format, and therefore links to the *include* file *hipl_format.h*. The program also uses *Aspirin for MIGRAINES* [32] neural network routines to generate outputs from preprocessed sub-image inputs.[15] The *Makefile* used to build the executable program is listed in Appendix B.9. The path names for network description files and *Aspirin* resource libraries must be in this file.

---

14 This area is used for various workstation computational speed comparisons in a later section. The system, however, is capable of processing images of any size.

15 More specifically, these routines use the file called *Network.save* that was created by *Aspirin for MIGRAINES* during network learning (training data is discussed in Section 5 and network learning is discussed in Section 6).

5.0 COLLECTING AND PREPROCESSING THE TRAINING DATA

The success of any neural network performing image feature extraction rests heavily on the amount of information used to train the network. A system which uses neural networks to locate the destination address block on U.S. mail must be taught to discriminate those features which describe a destination address block from those features which describe other candidate address blocks. This ability to discriminate among candidate address blocks requires that the neural network be trained on a large number of mail image examples.

Thirty magazine images were used to train the neural network in MITRE's Address Block Location system (see Section 4.4 for image scanning details, and Section 4.1 for magazine address block characteristics). Each magazine image was segmented into an address block region and one or more non-address block regions. The magazine may have been separated into several non-address block regions to preclude 10 and 11 point font text from appearing in the non-address block images.

The extracted address blocks were 256 pixels by 80 pixels (at 75 dpi). The address block of each of the thirty examples appeared in the center of this sub-image area. The height of the addresses that were extracted from the magazine images were

larger than necessary (80 pixels as opposed to the required 64 pixels) so that each address block would yield several training samples. More specifically, four 256 by 64 pixel training samples came from each address block (i.e., one from the top of the 256 by 80 pixel address image, one from the bottom, and two more that are 180 degree rotated copies of the first two). In addition, fourteen of the 256 by 80 pixel address blocks were also used to create rotated or skewed examples. Seven examples were rotated at positive angles, one example for each of the integer angles between 1 and 7 degrees. Moreover, the other seven covered the angles between -1 and -7 degrees. Four training samples from each of the rotated address blocks were then extracted. All of the samples were then preprocessed using a modified version of the Address Block Location program, and joined in an address block (positive) training file. The preprocessing of each sub-image involved performing an FFT, computing the magnitude of the FFT, then reducing the FFT using a 2 by 2 averaging mask (see Appendix A for source code description). There were 176 preprocessed address blocks in the positive training file (*flat.pos*).

Sub-images from the non-address block regions were also extracted and preprocessed using a modified version of the program. However, the sub-images were extracted in a

non-overlapping manner and were 256 by 64 pixels each (the required dimensions). There were 623 preprocessed non-address blocks joined in a negative training file (*flat.neg*).

6.0 NEURAL NETWORK TOPOLOGIES AND PARAMETERS

Many backpropagation neural network topologies were tested before one was selected and incorporated in the Address Block Location program. All of the networks trained, however, had one output node, since the object was to separate, or classify, destination address block regions from non-address block regions. Two of the neural network topologies that were trained, the simplest network and the network chosen for the program, are compared.

6.1 Backpropagation training parameters and strategies

The inputs to the neural networks trained came from the address block training file, *flat.pos*, and the non-address block training file, *flat.neg* (see Section 5 for details on training data). The inputs were an array of 128 by 32 positive 32-bit floating point numbers. The floating point inputs typically had a range between 0.0 and not greater than 250.0. The output node had a floating point range of -0.5 (non-address block) to 0.5 (address block).[16]

16 A sigmoidal node (see Section 3) has a floating point range between 0.0 and 1.0. The *Aspirin* software, however, allows the user to adjust the output of any sigmoidal node by adding an arbitrary bias term. The bias output term, used only on the output nodes of the networks trained, was -0.5. This would allow the output to swing between -0.5 and 0.5 so that non-address block regions could be classified with numbers near -0.5, and address blocks classified with numbers near 0.5.

During training of each of the networks, the *learning rate* was 0.05 and the *momentum* was 0.95 (see Section 3 for parameter definitions). Network weights were adjusted after each sub-image pattern presentation (*iteration* method) instead of each *epoch* (an *epoch* is the presentation of all the patterns). The sub-image patterns were presented during learning such that an address block sub-image pattern, from the file flat.pos, and a non-address block sub-image pattern, from the file flat.neg, alternated between input training presentations. An *epoch* represents 1245 pattern presentations, of which 623 were non-address block example images each presented once, and 176 were address block example images each presented approximately 3.5 times during the *epoch*. The training on all the networks was limited to 1000 *epochs*, after it was discovered that the networks improved little beyond this point.[17] Furthermore, during training, the target output for an address block was 0.5, and the target output for a non-address block was -0.5.

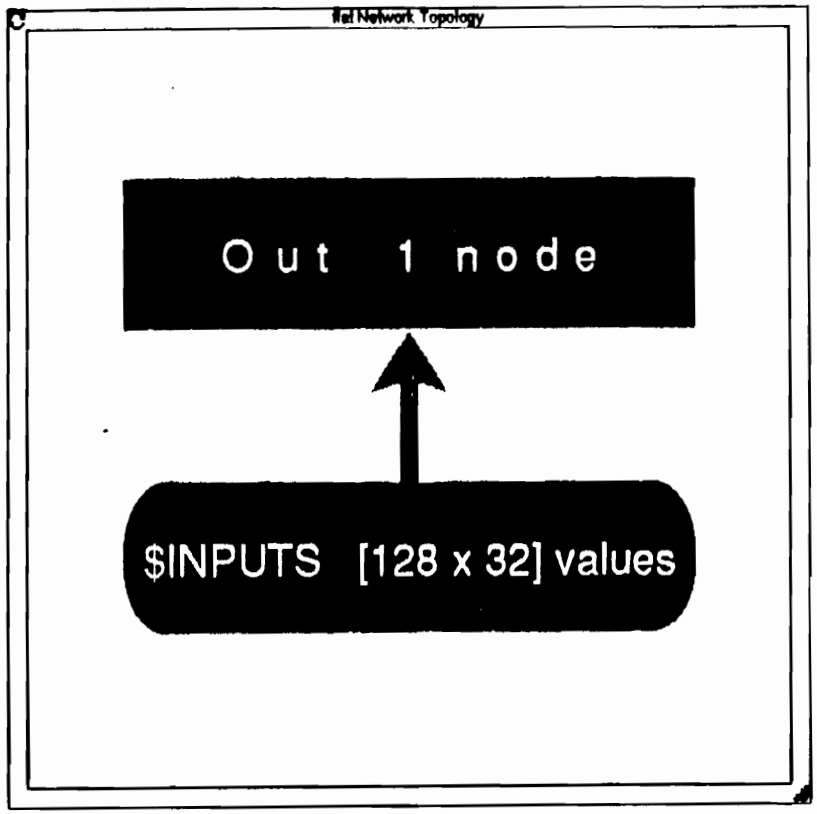
6.2 Topologies and training results

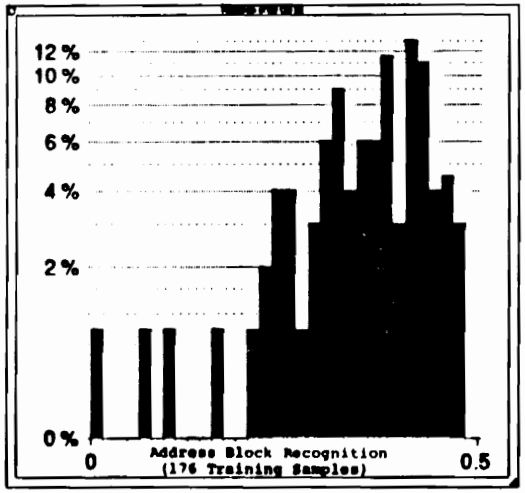
The first networks trained had fully-connected input architectures. This is where all the inputs are connected through weighted links to all nodes in a hidden layer or output

17 All network topologies considered, required less than 24 hours to train on a *Sun 4/280* computer.

layer. When these networks were trained, they were not able to converge in a reasonable amount of time (several days) to the desired output responses for many of the input examples. This can occur frequently in large input networks that are trained on many input examples. This problem was solved by using a technique where the inputs are separated into regions called tiles (tessellation), and each of the tiled areas or inputs is connected to its own hidden node (the tiled inputs can overlap). The outputs of the hidden nodes are then fully-connected to a node in a second hidden layer, or an output node. This technique reduces the amount of information that any node must learn or process. The network in the Address Block Location program uses this strategy.

Figure 8 shows the topology of a simple single layer network. This figure also shows histograms of the output responses to the training data after training was stopped. The upper histogram shows the responses for the 176 example address blocks, and the lower histogram shows the responses for the 623 non-address blocks. These histograms show an excessive overlap in the output responses (the responses for the address block examples are spread between 0.0 and 0.5, and the responses for the non-address blocks are spread between -0.5 and 0.5). Non-address blocks could be classified as address blocks, and vice versa.





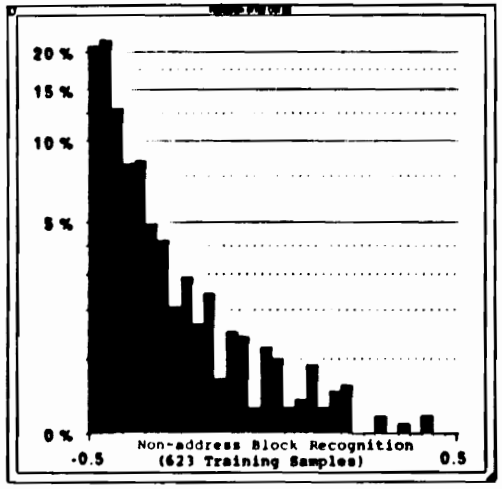
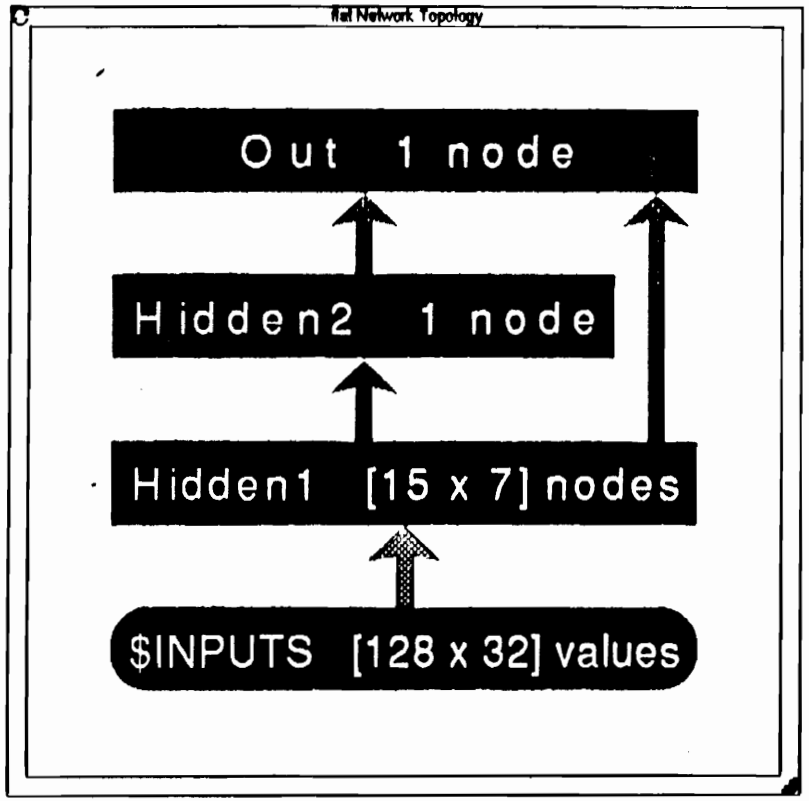
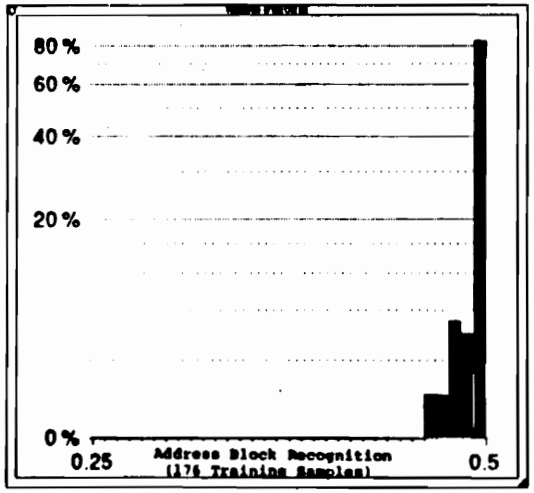


Figure 8: Simple one-layer address block location neural network, and output histogram training results

Figure 9 shows the topology chosen for this system. The input is separated into 15 by 7 overlapping tiles (one-half overlap in both dimensions). Each of the 15 by 7 nodes in the first hidden layer are fully-connected through input weights to a single tile in the input space. The first layer hidden nodes are fully-connected to a single node in a second hidden layer, and also to the output node (skip layer).[18] The output histograms on the training data are also shown in Figure 9. The separation between the two sets of data is evident. No address block training examples are classified less than 0.45. Also, the non-address block training examples are not classified above -0.25. This separation covers 70 percent of the span between the two target values (0.45 - [-0.25] = .70).

18 A tiled network topology without the second hidden layer node was trained. However, this network was not able to separate all of the training examples. The second hidden layer was incorporated to handle these more difficult examples, and it worked!.





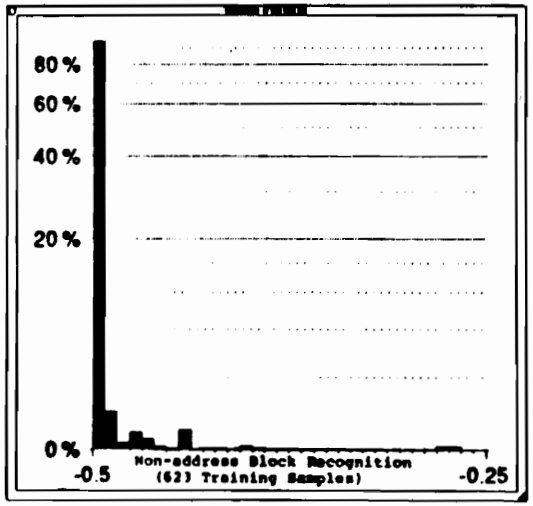


Figure 9: Tessellated address block location neural network with one-node second hidden layer and skip layer, and output histogram training results

7.0 SYSTEM TESTS

7.1 Test examples and processing time

Twelve images were tested. Examples are shown (not actual size) in Appendix C. The extracted sub-images next to each image are the areas that the system identified as the destination address block. The first six test examples are magazines of varying complexity, and the last three are the more interesting letter and non-magazine flat mail examples tested.

The system correctly identified the address block in five of the six magazine test examples. The magazine address block that was classified incorrectly has a font size similar to a typical address block, and a character density (number of characters per address block area) greater than a typical address block. Moreover, of the six non-magazine images tested, five were classified correctly (the incorrectly classified sub-image was a return address).

The system processes 210 sub-images on a 2550 by 3300 pixel magazine image. This number includes both the rough scan (32 by 128 rough scan input parameters) and fine scan sub-images. It required eight minutes to process such an image on a *Sun 4/280*, six minutes on a *Sun Sparcstation 2*, and four minutes on a *Sun 4/370*.

7.2 Address rotation limits

The system was tested on an address block image that was rotated between one and seven degrees, in one degree increments. Figure 10 shows an image rotated between one and seven degrees, and the corresponding floating point output value (six significant digits) of the neural network for each of these sub-images. The output remains relatively constant (≈0.49) until the rotation reaches seven degrees (output at seven degrees is approximately 0.43). This is not surprising since the network was trained on examples between seven and negative seven degrees.

7.3 Analysis of feature extraction

Figure 11 shows an address block and its magnitude Fourier transform. The dominant features in this magnitude Fourier transform are the DC spike in the middle and the recurring vertical ridge structures. An experiment was performed to determine those features in the magnitude FFT that the neural network was using to identify a possible address block.

A Run-Length Smoothing Algorithm (RLSA) [18,19] was performed on this address block to produce an address block that has filled characters. (This address block is shown in Figure 12(a).) Filling the characters would test the importance of character stroke frequencies to the Address Block Location system. RLSA was performed again to produce an address block

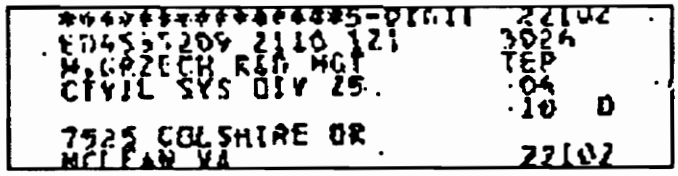
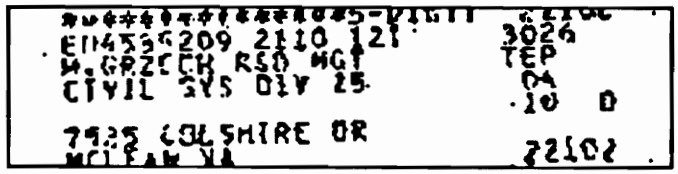
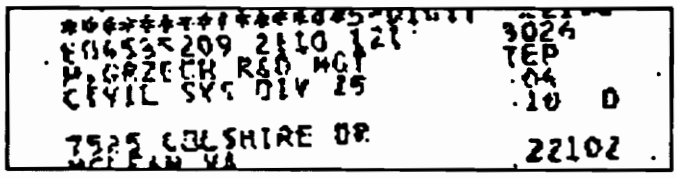
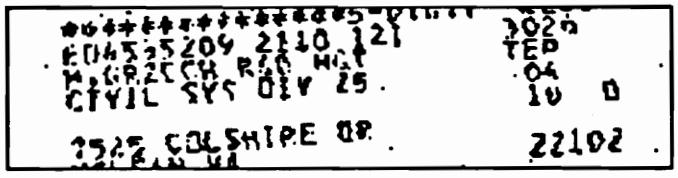
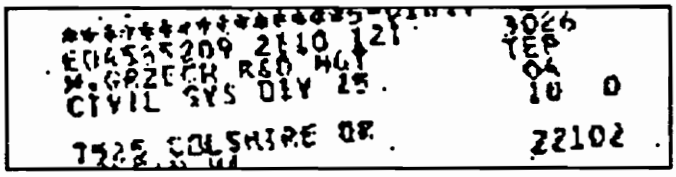
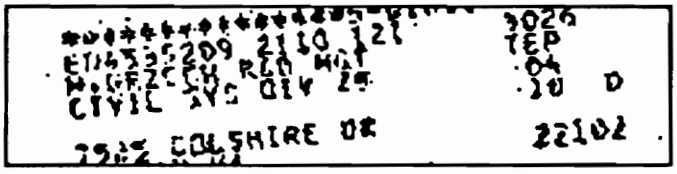
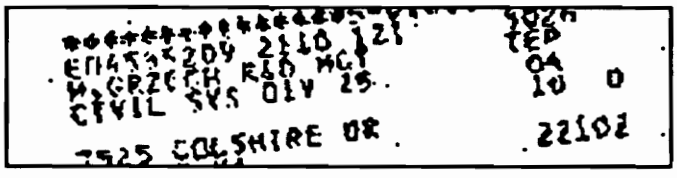
IMAGE	OUTPUT VALUE
	0.495282
	0.497264
	0.497731
	0.497731
	0.493661
	0.481595
	0.438576

Figure 10: Neural network output value for address block rotated between one and seven degrees in one degree increments

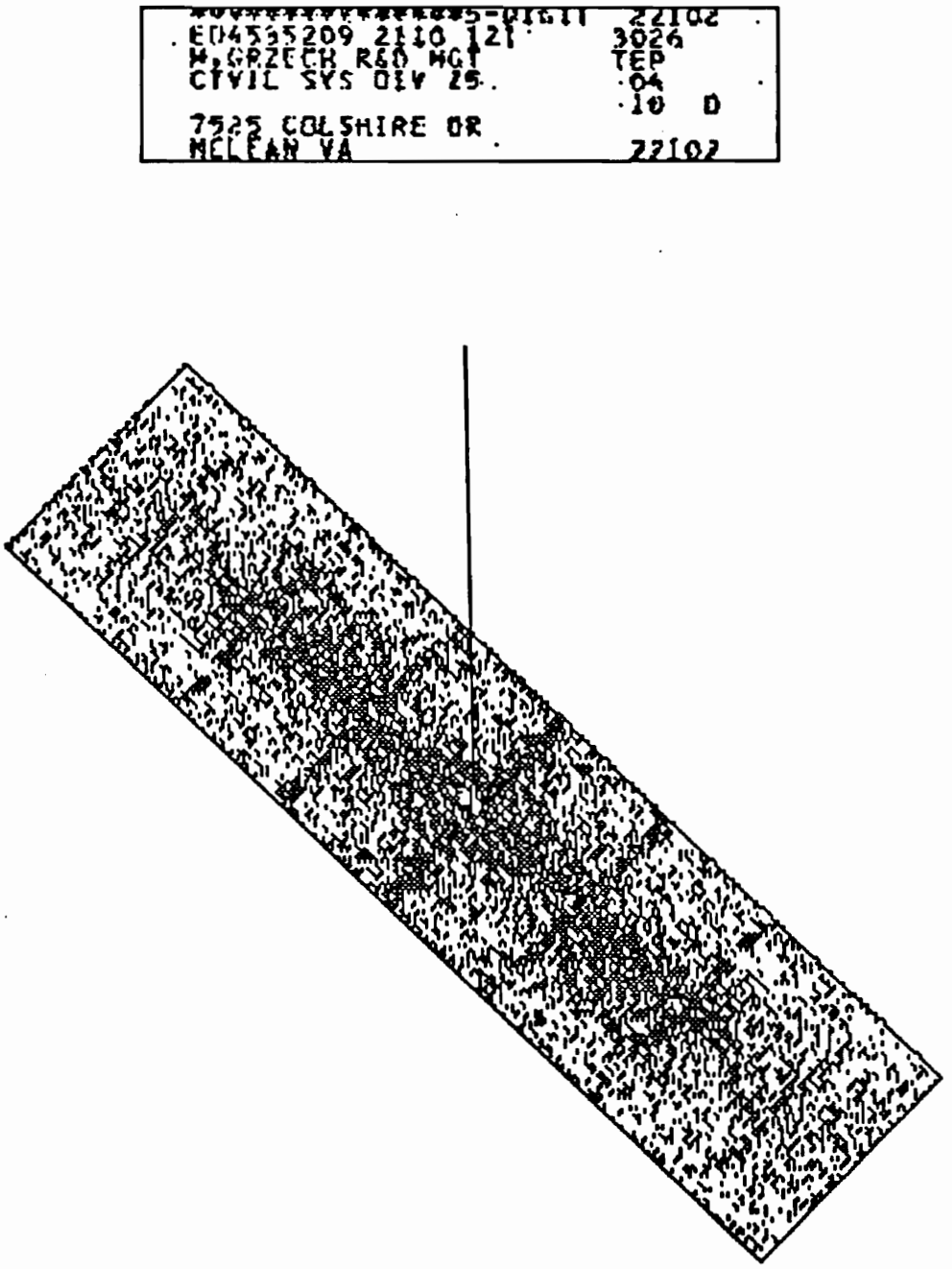


Figure 11: Address block and its magnitude FFT

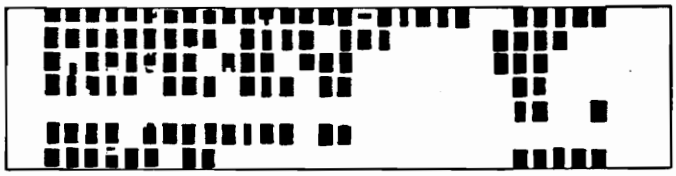


Figure 12(a): Character RLSA in 75 dpi address block

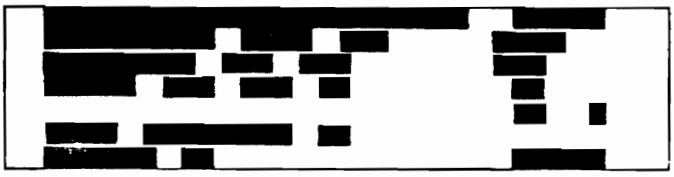


Figure 12(b): Word RLSA in 75 dpi address block

where the characters are joined and filled at the word level (Figure 12(b)). This would test whether inter-character frequencies are important.

The system responded to the sub-image in Figure 12(a) (filled characters) with an output value of 0.499250. This value is very close to the address block target value of 0.5. This indicates that character stroke frequencies are not used to classify an actual address block. The system responded to the sub-image in Figure 12(b) (filled and joined characters at the word level) with an output value of -0.499999. This very low output value (approximately equal to the non-address block target value of -0.5), and the output value from the previous RLSA test, indicates that the neural network uses character frequencies only to identify possible address blocks (the inter-word frequencies are not used). (Inter-line frequencies are also not used, since the lines are still identifiable in Figure 12(b), and the output value was so low.).[19]

19 Confirming this hypothesis, an analysis of the weights (using the *Aspirin for MIGRAINES* graphics display tools) in the first layer of the neural network, shows a cluster of large weight values in the area corresponding to the relative inter-character frequencies of the training examples.

8.0 CONCLUSIONS

The neural network used in this system was able to identify the strongest feature in the FFT of address block images, namely the inter-character frequencies, and incorporate these features as large positive connections in its weight space. Frequencies that were common and strong in non-address block regions (those non-address block regions that were used as examples), but were weak in the example address blocks, were given large negative values in the neural network weight space. Other strong frequencies common to both address blocks and non-address blocks, were correspondingly given near zero values in the weight space. The backpropagation gradient descent method of learning was the algorithm used to mold the weight space to minimize the error between the actual and expected outputs for all training examples.

The system performed well on the example images. However, the system failed when presented with an image that has text of a similar size and spacing to the address block text examples that the system was trained to recognize. This is where more input information is needed in identifying the address block. Histogram profiles of sub-images in the horizontal and vertical orientations could be useful information in reducing the number of candidate address block images. Also, characteristic pixel

densities of address blocks may be helpful. The histogram and pixel information could possibly be encoded in a single neural network. At a higher level, connected component analysis could be performed in conjunction with the present neural network approach to locate possible ZIP Codes in candidate address block images. This may be coupled with current Optical Character Recognition (OCR) algorithms to read the (potential) ZIP Codes and street addresses.

The system worked well at 75 dpi, and the software is portable enough to work across many computing platforms. The implementation is also scalable in several areas. An image can be broken into smaller regions and multiple versions of the code can process these regions on separate systems. The information could then be consolidated (pyramid image processing architectures [34]). The neural network and two-dimensional FFT algorithms are both scalable and can be performed on multiple processors. The major routines in the system (i.e., sub-image extraction, magnitude FFT computation, FFT reduction, and sub-image FFT classification) can be pipelined as well. All of these factors are important if an approach is to succeed outside of pure research in a highly demanding production environment.

9.0 ISSUES FOR FURTHER RESEARCH

The current design of MITRE's Address Block Location system is very simple, and the performance is good. The simplicity of the approach makes the task of converting it to a production environment, like the USPS, not difficult. MITRE's Address Block Location software would have to be implemented in a prototype system with parallel and pipeline processing capabilities. An address block re-orienting algorithm would be added (i.e., de-skew the address block and rotate it 180 degrees if it is inverted). The system would then have to be tested on a representative magazine flat mail set.[20] A cost/benefit analysis of the system would then be needed to compare this system with other alternatives.

Continuing research would improve the software by incorporating new address block input characteristics (see Section 8). Lower image resolutions would be explored as well. In the current implementation, a 2 by 2 averaging mask (Section 4) is applied to the discrete Fourier magnitudes of each of the 75 dpi sub-images extracted. A natural choice would be to test the approach at half the current resolution, or 37.5 dpi, since the 2 by 2 averaging of the discrete Fourier magnitude would not

20 The USPS currently has not defined a representative magazine flat mail set.

be necessary at this resolution.[21] Moreover, the accuracy of this approach on 37.5 dpi images would probably degrade little from the current 75 dpi implementation.[22] However, the processing time would improve remarkably.

Additional experimentation is also needed to determine if backpropagation neural networks learn when the image is handwritten as opposed to machine printed, and is gray-scale as opposed to binarized. Gray-scale images should not present any learning difficulties for the present network since a magnitude FFT is performed on the sub-images prior to classification. The network can be taught to filter local background contrast in the gray-scale image and, hence, detect the inter-character frequencies of the machine printed address block. This would reduce the processing time of a prototype system since gray-scale image thresholding would not be necessary.

---

21 To maintain the current sub-image capture size of 3.08 by .85 inches, the extracted sub-images would be 128 by 32 pixels at 37.5 dpi. The discrete Fourier magnitude of a sub-image with these dimensions is also 128 by 32. This corresponds to the input dimensions of the neural network used in this system (although the neural network would be trained on 37.5 dpi examples).

22 Although information is lost when an image is down-sampled from 75 to 37.5 dpi (which may make processing images at 37.5 dpi less attractive), information is also lost in the current implementation when a 2 by 2 averaging mask is applied to the discrete Fourier magnitude of each of the 75 dpi sub-images.

REFERENCES

[1] P.W. Palumbo, J. Soh, S.N. Srihari, V. Demjanenko, R. Sridhar, Real-Time Address Block Location using Pipelining and Multiprocessing, Proceedings of the USPS Advanced Technology Conference, Washington, D.C., November 5-7, 1990, pp. 73-87.

[2] S.N. Srihari, J. Soh, P.W. Palumbo, Towards Developing a Real-time System to Locate Address Blocks on Mail Pieces, Proceedings of the USPS Advanced Technology Conference, Washington, D.C., May 3-5, 1988, pp. 57-64.

[3] S.N. Srihari, C. Wang, P.W. Palumbo, J.J. Hull, Recognizing Address Blocks on Mail Pieces: Specialized Tools and Problem-Solving Architecture, AI Magazine, Winter 1987, pp. 25-40.

[4] A. DiBiase, H. Liff, An Integrated Address Block Location and Recognition Processor, Proceedings of the USPS Advanced Technology Conference, Washington, D.C., November 5-7, 1990, pp. 1217-1230.

[5] A.F. Lehar, The Automatic Detection and Ranking of Address Blocks in Unconstrained Mail, Proceedings of the USPS Advanced Technology Conference, Washington, D.C., October 21-32, 1986, pp. 132-142.

[6] L. Turnbaugh, J.C. Weaver, The Case for Graph-Theoretic Clustering for Address Finding, Proceedings of the USPS Advanced Technology Conference, Washington, D.C., October 21-32, 1986, pp. 147-153.

[7] P.G. Mulgaonkar, Multiview Image Acquisition and Address-Block Location for Parcels, Proceedings of the USPS Advanced Technology Conference, Washington, D.C., November 5-7, 1990, pp. 5-17.

[8] P.G. Mulgaonkar, A. Bergman, Address Block Location: The SRI Approach, Proceedings of the USPS Advanced Technology Conference, Washington, D.C., October 21-32, 1986, pp. 161-168.

[9] A. Bergman, P.G. Mulgaonkar, Neural Networks for Address-Block Ranking: A Comparison with Classical Techniques, Proceedings of the USPS Advanced Technology Conference, Washington, D.C., May 3-5, 1988, pp. 736-750.

[10] A. De Gaetano, F. Morando, Real-Time Address Block Location using concurrent expert systems in a parallel architecture, Proceedings of the USPS Advanced Technology Conference, Washington, D.C., November 5-7, 1990, pp. 58-72.

[11] M. Caviglione, G. Musso, B. Ortolani, An Advanced Approach to the Address Block Finding Problem, Proceedings of the USPS Advanced Technology Conference, Washington, D.C., October 21-32, 1986, pp. 85-96.

[12] C. Ronse, P.A. Devijver, Connected Components in Binary Images: the Detection Problem, John Wiley & Sons, New York, 1984.

[13] P. Palumbo, P. Swaminathan, S.N. Srihari, Document Image Binarization: Evaluation of Algorithms, Proceedings of the SPIE Symposium on Applications of Digital Image Processing IX, Bellingham, WA, pp. 278-285.

[14] R.C. Gonzalez, P. Wintz, Digital Image Processing, Addison-Wesley, Reading, MA., 2nd edition, 1987.

[15] J.T. Tou, R.C. Gonzalez, Pattern Recognition Principles, Addison-Wesley, Reading, MA., 1974.

[16] L.A. Fletcher, R. Kasturi, A Robust Algorithm for Text String Separation form Mixed Text/Graphics Images, IEEE Trans. Pattern Anal. Machine Intell., vol. 10, no. 6, Nov. 1988, pp. 910-918.

[17] S.N. Srihari, Document Image Understanding, Proc. IEEE Comput. Soc. Fall Joint Comput. Conf., Dallas, TX, Nov 2-6, 1986.

[18] K.Y. Wong, R.G. Casey, F.M. Wahl, Document Analysis System, IBM J. Res. Develop., 26(6):647-56, 1982.

[19] J.L. Fisher, S.C. Hinds, D.P. D'Amato, A rule-based system for document image segmentation, Proceedings of the 10th International Conference on Pattern Recognition, 1990. pp. I-567-72.

[20] O. Iwaki, H. Kida, H. Arakawa, A Segmentation Method Based on Office Document Hierarchical Structure, Proc. IEEE Int. Conf. Syst., Man. Cybern., Alexandria, VA, Oct. 20-23, 1987, pp. 759-763.

[21] G. Nagy, S.C. Seth, S.D. Stoddard, Document Analysis with an Expert System, Proc. ACM Conf. Document Processing Systems, Santa Fe, NM, Dec. 5-9, 1988, pp. 169-176.

[22] R.O. Duda, P.E. Hart, Pattern Classification and Scene Analysis, John Wiley & Sons, 1973, p. 237.

[23] C.T. Zahn, Graph-Theoretical Methods for Detecting and Describing Gestalt Clusters, IEEE Trans. Comp., Vol. C-20, No. 1, pp. 68-86, Jan. 1971.

[24] M.P. Grzech, A Backpropagation Neural Network as part of a Rule-Based System for Document Image Segmentation, Fifth International Conference on Methodologies for Intelligent Systems, Selected Papers, Knoxville, Tennessee, October 24-27, 1990, pp. 237-248.

[25] J.L. McClelland, D.E. Rumelhart, Explorations in Parallel Distributed Processing, 3rd Ed. MIT Press, 1988. ISBN 0-262-63113-X.

[26] D.J. Burr, Experiments on Neural Net Recognition of Spoken and Written Text, IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol. 36, No. 7, pp. 1162-1168, July 1988.

[27] Y.S. Abu-Mostafa, Information Theory, Complexity, and Neural Networks, IEEE Communications Magazine, pp. 25-28, November 1989.

[28] A. Wieland, R. Leighton, Geometric Analysis of Neural Network Capabilities, presented at the IEEE Int. Conf. Neural Networks, San Diego, CA, June 1987.

[29] Georgia Institute of Technology Research Institute (GTRI), Automated Processing of Irregular Parcel Post: Flat Statistical Database, Prepared for the USPS Office of Operations Research and Systems Requirements, March 14, 1986.

[30] J.W. Cooley, P.A.W. Lewis, P.D. Welch, The Fast Fourier Transform and its Applications, IEEE Trans. Educ., vol E-12, no 1, pp. 27-34.

[31] The HIPS Picture Processing Software, Version 6, Psychology Department, New York University, December 26, 1982.

[32] Aspirin for MIGRAINES, Version 4.0, MITRE Corp., McLean, VA, MITRE Paper MP-90W00044, September 1990.

[33] MICROTEK 300GS User's Manual, Microtek Lab, Inc., 16901 South Western Avenue, Gardena, California 90247.

[34] L. Uhr, Parallel Computer Vision, Academic Press, Inc., Orlando, Florida, 1987.

APPENDIX A

Detailed description of Address Block Location source code

The program consists of the function *main()* and six function subroutines. The function subroutines are *extract()*, *fourtr()*, *fft_2dgen()*, *fftn()*, *reduce2()*, and *nn_class()*. These functions will be discussed in detail in the following sections.

A.1 The function *main()*

The function *main()* is listed in Appendix B.2. The function *main* accepts a set of input arguments, processes these arguments, and produces three file reports. (An output is also sent to the standard IO. This output is used by the script file to extract and display the destination address block.) There are seven input arguments required in the following order:

- File name of original input image in HIPS format
- File name of 90 degree rotated original input image in HIPS format (rotated in the script file -- see Section 4.4)
- Vertical sub-image increment distance (integer number of pixels)
- Horizontal sub-image increment distance (integer number of pixels)
- Name of first output file: original image rough-scan output response (a two-dimensional array of floating point

values where each value represents the neural network output response to a preprocessed sub-image at this coordinate location)

- Name of second output file: ninety degree rotated original image rough-scan output response (array of floating point sub-image output responses for rotated image)
- Name of third output file: the orientation (original or rotated image) and coordinates of the address block

The magazine image is represented by two input files in the input argument string -- the original and ninety degree rotated versions. These files are processed in sequence by the program. The horizontal and vertical sub-image scan parameters come after the input image file names in the input argument string. The final arguments are the user specified output file names.

The program begins by opening the first input image and striping the file header. This header contains information about the number of pixel rows and columns in the image (the size of the image). This information is used to determine the maximum row and column coordinates during sub-image rough scan. Sub-images are then systematically located, using the increment input parameters, and extracted for evaluation. Extracting the sub-images and evaluating them requires the use of the function subroutines previously listed. The output of the rough scan for

the first image processed is written to the first output file. The ninety degree rotated input image is processed in the same manner as the original image. The output of the rough scan for this image is written to the second output file. After the rough scan is performed, a fine scan is performed on the three candidate regions. Information about these three candidate regions were saved in a location structure during rough scan. This location structure stores the neural network output value of the sub-image, the sub-image coordinates, and the orientation of the image (the original image is represented by orientation 0 , and the rotated image is represented by orientation 1). This structure is updated with a new sub-image candidate if its neural network output value is greater than any other in the structure. The result of the fine scan is printed in the third output file.

The first and second output files show the floating point output values that the neural network returned for the extracted sub-images in the original and ninety degree rotated images during the rough scan. The third output file lists the orientation, coordinates, and neural network value of the selected destination address block sub-image after a fine scan of the three candidate areas (the three candidate areas are represented by the three highest values in the floating point outputs of the first and second output files). For the image processed in Figure 3, the contents of the first output file are

in Figure 13, the contents of the second output file are in Figure 14, and the contents of the third output file are in Figure 15.

A.2 The function *extract()*

The function *extract()* is listed in Appendix B.3. This function is used by *main()* to extract 64 by 256 pixel sub-images from either the original or rotated magazine images. The coordinates of the sub-image to be extracted are passed to this function. Since the input image is not normalized (the input image is bit-packed where the bits are either 0 or 1), the extracted sub-images are normalized such that a binary 1 becomes a floating point value of 1.0, and a binary 0 becomes a floating point value of -1.0 (see Section 4.3.3 for the definition of normalization). The sub-image is then passed back to *main()* for further processing.

A.3 The function *fourtr()*

The function *fourtr()* is listed in Appendix B.4. This function computes the complex FFT of the input sub-image. This function then returns the magnitude of the FFT. The complex two-dimensional FFT is computed using another function, *fft_2dgen()*.

A.4 The functions *fft_2dgen()* and *fftn()*

The two-dimensional FFT is implemented by performing a series of one-dimensional FFTs on the rows of a real image to

-0.499999	-0.499999	-0.499999
-0.499999	-0.499999	-0.499999
-0.499999	-0.499999	-0.499999
-0.498960	-0.067937	-0.131594
-0.325590	0.453622	0.371475
-0.474629	-0.275945	-0.391055
-0.481875	0.316406	0.490366
-0.405322	0.075588	0.489592
-0.013668	-0.190976	0.391055
-0.253467	-0.477197	0.037041
-0.493759	-0.422482	-0.275945
-0.498858	0.366132	0.428926
0.424688	0.497659	0.494657
0.482151	0.476129	0.491084
0.348972	0.459459	0.499137
0.264892	0.381615	0.485608
0.479511	0.229520	-0.146014
0.484694	0.314053	0.281331
0.458226	0.360604	0.270465
-0.273216	0.029263	-0.466156
-0.499552	-0.497508	-0.499662
-0.499999	-0.499999	-0.499999
-0.499999	-0.499999	-0.499999
-0.499999	-0.499999	-0.499999

Figure 13: Output floating point values from rough scan of image (orientation 0)

-0.499999	-0.499416	-0.498390	-0.499640	-0.499999
-0.499538	-0.494900	-0.492244	-0.495635	-0.498706
-0.499611	-0.498259	-0.498706	-0.497834	-0.499485
-0.499573	-0.496146	-0.498534	-0.499999	-0.499999
-0.499999	-0.499999	-0.499999	-0.499999	-0.499999
-0.499531	-0.496321	-0.499611	-0.499999	-0.499999
-0.497508	-0.490803	-0.499623	-0.499999	-0.499999
-0.499552	-0.498463	-0.498992	-0.497348	-0.499999
-0.499999	-0.499646	-0.494979	-0.496803	-0.499646
-0.499508	-0.497800	-0.476129	-0.472204	-0.499096
-0.498726	-0.492364	-0.430962	-0.456957	-0.499164
-0.499317	-0.478219	-0.456309	-0.499038	-0.499629
-0.499999	-0.498463	-0.498556	-0.499999	-0.499999
-0.499999	-0.498313	-0.497901	-0.499999	-0.499999
-0.499999	-0.495208	-0.464052	-0.497389	-0.499999
-0.499999	-0.497469	-0.497585	-0.495354	-0.499999
-0.499999	-0.498415	-0.498176	-0.499468	-0.499999
-0.499999	-0.499999	-0.499999	-0.499999	-0.499999

Figure 14: Output floating point values from rough scan of 90 degree rotated original image (orientation)

```
FINE ANALYSIS
output value = 0.499137
location is row 448, col 256
orientation is 0
```

Figure 15: Output of fine scan analysis gives image orientation, location in pixels relative to the upper left corner (coordinates, {0,0}), and the output floating point value

produce an imaginary image with real and complex components. Then, a series of one-dimensional FFTs on the columns of the resultant imaginary image is performed to complete the process (the process can also start with the columns of the real image). The function *fft_2dgen()* uses the function *fftn()* to perform the FFT operations on individual rows and columns. The function *fft_2dgen()* is listed in Appendix B.5, and the function *fftn()* is listed in Appendix B.6.

A.5 The function *reduce2()*

The function *reduce()* is used to perform a 2 by 2 spectral averaging (a 2 by 2 averaging mask) on the magnitude of the FFT. It was found in previous research on textual images [24] that a 2 by 2 spectral averaging of the FFT significantly increases learning speed without a loss in classification performance. The 2 by 2 spectral averaging reduces the input requirements of the neural network from 16K inputs (256 x 64) to 4k inputs (128 x 32). This function is listed in Appendix A.7.

A.6 The function *nn_class()*

The function *nn_class()*, which is listed in Appendix A.8, classifies the reduced input sub-image spectrum. Depending on the sub-image input, this function returns a value between -0.5 and 0.5 to the *main()* function. This function uses four *Aspirin* routines. The first of these functions is *network_initialize()*

which initializes the network using the description in the "DUMPFILE" (see *Aspirin* documentation [30] and *Makefile* in Appendix B.9). The network inputs are set using the function *flat_set_input()*. The inputs are propagated through the weight space using the function *flat_propagate_forward()*. Finally, the neural network value is returned to the function *main()* using the function *flat_get_output()*. (The three functions *flat_set_input()*, *flat_propagate_forward()*, and *flat_get_output()*, are specific to this application. The neural network software *Aspirin* was used to create these functions, along with the file *Network.save* which stores the network weights.)

APPENDIX B: Address Block Location source code

B.1 Address Block Location *UNIX* script file

```
#!/bin/csh

set RM = "/bin/rm -f"

# The input file must have a HIPS header
set INFILE  = "$1"

set INCH    = "$2"
set INCW    = "$3"
set OUT1    = "$4"
set OUT2    = "$5"
set OUT3    = "$6"

bunpack < ${INFILE} | rotate90 |bpack -m > XxXrotate

abl_nn_class75demo ${INFILE} XxXrotate ${INCH} ${INCW} \
                   ${OUT1} ${OUT2} ${OUT3} > XxXstuff

set ORIENT = `awk '{print $1}' XxXstuff`
set ROW    = `awk '{print $2}' XxXstuff`
set COL    = `awk '{print $3}' XxXstuff`

if (${ORIENT} == 0) then
    extract 64 256 ${ROW} ${COL} < ${INFILE} | sunv -Wp 700 410
else if (${ORIENT} == 1) then
    extract 64 256 ${ROW} ${COL} < XxXrotate | sunv -Wp 700 410
else
    echo "Woe shag - wrong value for ORIENT.  It was
>>${ORIENT}<<. Bye."
    exit
endif

${RM} XxX*
#
```

B.2 Address Block Location *main()* source code

```
/* abl_nn_class75demo.c */

#include <hipl_format.h>
#include <stdio.h>
#include <fcntl.h>
#include <math.h>
#include HEADER

#define pi 3.14159265358979

static float sub_image[16384],spectrum[16384],red_spec[4096];
static float *cconst,*iconst;
static int storesize = 0, constsize = 0;

FILE *outfp1, *outfp2, *outfp3;
char Progname[] = "abl_nn_class75demo";

struct Location {
     float value;
     int row, col;
     int orientation;
};

float nn_class();

main (argc,argv)

int argc;
char *argv[];
{
     struct header hd;
     struct Location location[3];
     unsigned char *frame,*rframe;
     char infile[80], rotated[80], outfile1[80], outfile2[80],
outfile3[80];
     int inch, incw, max_row, max_col, rmax_row, rmax_col,
row_test, col_test;
     float value;
     int row,col,rndcol,rrow,rcol,rrndcol,fr,fc;
     int fine_inch, fine_incw, max_r, max_c, min_r, min_c,
skip_row, skip_col;
     register int i;
     int fp;
```

```
    if (argc != 8) {
      printf ("\nusage: abl_nn_class infile
rotated_infile_90_degree inch incw outfile1 outfile2 outfile3");
      exit (1);
    }

    strcpy (infile, *(argv+1));
    strcpy (rotated, *(argv+2));
    inch = atoi(*(argv+3));
    incw = atoi(*(argv+4));
    strcpy (outfile1, *(argv+5));
    strcpy (outfile2, *(argv+6));
    strcpy (outfile3, *(argv+7));

/* ----------- open input image ----------- */
    fp = open (infile, O_RDONLY);
    if (fp == -1) {
      printf ("\ncould not find input file, %s", infile);
      exit (1);
    }
/* ----------- read header and information from header ----------
*/
    fread_header(fp, &hd);

    if (hd.pixel_format != PFBYTE)
      perr("input sequence must be bytes");
    if (hd.bits_per_pixel != 1)
      perr("input must be 1 bits per pixel");
    if (hd.bit_packing == 0)
      perr("input must be bit packed");
    hd.bit_packing = 0;
    hd.bits_per_pixel = 8;
    row = hd.rows;
    col = hd.cols;
    rndcol = (col+7)/8;
/* ----------- allocate space for input frame ------------- */
    frame = (unsigned char *) halloc(row*rndcol,sizeof (unsigned
char));
/* ----------- read input handle into temp frame input image
------------- */
    if (pread(fp,frame,row*rndcol*sizeof(unsigned char)) !=
        row*rndcol*sizeof(unsigned char))
      perr("unexpected end-of-file");
    close (fp);
/* ----------- compute the "max_row" and "max_col" for sub-image
extraction -------------- */
```

```
    max_row = max_col = 0;
    row_test = row-64-inch;
    col_test = col-256-incw;
    while (max_row <= row_test) max_row += inch;
    while (max_col <= col_test) max_col += incw;
/* ----------- start loop where we extract sub-image, perform
fourtr, reduce, and finally nn_class ----------- */

    location[0].value = -0.5;

    fr = 0;
    outfp1 = fopen (outfile1,"w");
    while (fr <= max_row) {
      fc = 0;
      while (fc <= max_col) {
        extract (sub_image, frame, rndcol, fr, fc);
        fourtr (spectrum, sub_image);
        reduce2 (red_spec, spectrum);
        value = nn_class (red_spec);
        fprintf(outfp1,"%f ",value);
        if (value > location[0].value) {
          location[2].value = location[1].value;
          location[2].row = location[1].row;
          location[2].col = location[1].col;
          location[2].orientation = location[1].orientation;
          location[1].value = location[0].value;
          location[1].row = location[0].row;
          location[1].col = location[0].col;
          location[1].orientation = location[0].orientation;
          location[0].value = value;
          location[0].row = fr;
          location[0].col = fc;
          location[0].orientation = 0;
        }
        fc += incw;
      }
```

```
      fr += inch;
      fprintf(outfp1,"\n");
    }
    fclose (outfp1);
/* ----------- open rotated input image ----------- */
    fp = open (rotated, O_RDONLY);
    if (fp == -1) {
      printf ("\ncould not find rotated input file, %s",
rotated);
      exit (1);
    }
/* ----------- read header and information from header ----------
*/
    fread_header(fp, &hd);

    if (hd.pixel_format != PFBYTE)
      perr("input sequence must be bytes");
    if (hd.bits_per_pixel != 1)
      perr("input must be 1 bits per pixel");
    if (hd.bit_packing == 0)
      perr("input must be bit packed");
    hd.bit_packing = 0;
    hd.bits_per_pixel = 8;
    rrow = hd.rows;
    rcol = hd.cols;
    rrndcol = (rcol+7)/8;
/* ----------- allocate space for rotated input frame
------------- */
    rframe = (unsigned char *) halloc(rrow*rrndcol,sizeof
(unsigned char));
/* ----------- read input handle into temp frame input image
------------- */
    if (pread(fp,rframe,rrow*rrndcol*sizeof(unsigned char)) !=
        rrow*rrndcol*sizeof(unsigned char))
      perr("unexpected end-of-file");
    close (fp);

/* ---- start loop on rotated image where we extract sub-image,
perform fourtr, reduce, and finally nn_class --- */
    rmax_row = rmax_col = 0;
    row_test = rrow-64-inch;
    col_test = rcol-256-incw;
    while (rmax_row <= row_test) rmax_row += inch;
    while (rmax_col <= col_test) rmax_col += incw;

    fr = 0;
    outfp2 = fopen(outfile2,"w");
```

```
    while (fr <= rmax_row) {
      fc = 0;
      while (fc <= rmax_col) {
        extract (sub_image, rframe, rrndcol, fr, fc);
        fourtr (spectrum, sub_image);
        reduce2 (red_spec, spectrum);
        value = nn_class (red_spec);
        fprintf(outfp2,"%f ",value);
        if (value > location[0].value) {
          location[2].value = location[1].value;
          location[2].row = location[1].row;
          location[2].col = location[1].col;
          location[2].orientation = location[1].orientation;
          location[1].value = location[0].value;
          location[1].row = location[0].row;
          location[1].col = location[0].col;
          location[1].orientation = location[0].orientation;
          location[0].value = value;
          location[0].row = fr;
          location[0].col = fc;
          location[0].orientation = 1;
        }
        fc += incw;
      }
      fr += inch;
      fprintf(outfp2,"\n");
    }
    fclose (outfp2);

/* ------------------ do a fine analysis of the three highest
regions ---------------- */
    fine_inch = inch/2;
    fine_incw = incw/2;

    for (i=0;i<3;i++) {
      skip_row=location[i].row;
      skip_col=location[i].col;
      if ((min_r = location[i].row-fine_inch) < 0) min_r=0;
      if ((min_c = location[i].col-fine_incw) < 0) min_c=0;
      if (location[i].orientation==0) {
        if ((max_r = location[i].row+fine_inch) > max_row)
          max_r = max_row;
        if ((max_c = location[i].col+fine_incw) > max_col)
          max_c = max_col;
      }
      else {
        if ((max_r = location[i].row+inch) > rmax_row)
```

```
    max_r = rmax_row;
  if ((max_c = location[i].col+incw) > rmax_col)
    max_c = rmax_col;
}
if (location[i].orientation==0) {
  fr = min_r;
  while (fr <= max_r) {
    fc = min_c;
    while (fc <= max_c) {
      if ((skip_row != fr) || (skip_col != fc)) {
        extract (sub_image, frame, rndcol, fr, fc);
        fourtr (spectrum, sub_image);
        reduce2 (red_spec, spectrum);
        value = nn_class (red_spec);
        if (value > location[0].value) {
          location[0].value = value;
          location[0].row = fr;
          location[0].col = fc;
          location[0].orientation = 0;
        }
      }
      fc += fine_incw;
    }
    fr += fine_inch;
  }
}
else {
  fr = min_r;
  while (fr <= max_r) {
    fc = min_c;
    while (fc <= max_c) {
      if ((skip_row != fr) || (skip_col != fc)) {
        extract (sub_image, rframe, rrndcol, fr, fc);
        fourtr (spectrum, sub_image);
        reduce2 (red_spec, spectrum);
        value = nn_class (red_spec);
        if (value > location[0].value) {
          location[0].value = value;
          location[0].row = fr;
          location[0].col = fc;
          location[0].orientation = 1;
        }
      }
      fc += fine_incw;
    }
```

```
            fr += fine_inch;
          }
        }
      }
      outfp3 = fopen(outfile3,"w");
      fprintf(outfp3,"FINE ANALYSIS\n");
      fprintf(outfp3,"output value = %f\n",location[0].value);
      fprintf(outfp3,"location is row %d, col
%d\n",location[0].row,location[0].col);
      fprintf(outfp3,"orientation is
%d\n",location[0].orientation);
      fclose (outfp3);
      printf("%d %d %d",location[0].orientation,
location[0].row,location[0].col);
      }

free (frame); free (rframe);
} /* ------------------------------------------------------------ */
```

B.3 The function subroutine *extract()*

```
extract (ofr, ifr, packed_c, fr, fc)
/* extract a 64 row by 256 col float sub_image from the following
r,c coordinates */

unsigned char *ifr;
float *ofr;
int packed_c, fr, fc;
{
     register int i,j;
     register unsigned char *ip_byte, *ip_row_start;
     register float *op_float;
     int ip_bit, first_col;
     static char bit[9] = { 0000, 0001, 0002, 0004, 0010, 0020,
0040, 0100, 0200 };

     ip_row_start = ip_byte = ifr + fr*packed_c + fc/8;
     first_col = 8-(fc%8);

     op_float = ofr;

#define NEXTip_bit \
     if (!(--ip_bit)) { \
       ip_bit = 8; /* move to next input bit ... */       \
       ip_byte++;  /* ... which is in next input byte */  \
     }

     i=64;
     while(i--) {
       ip_bit=first_col;
       j=256;
       while(j--) {
         if (*ip_byte & bit[ip_bit]) *op_float = 1.0;
         else *op_float = -1.0;
         ++op_float;
         NEXTip_bit;
       }
       ip_byte = (ip_row_start += packed_c);
     }
}
/* ------------------------------------------------------ */
```

B.4 The function subroutine *fourtr()*

```
fourtr (ofr, ifr)
/* spectral fft on 256 wide by 64 tall binary image */

float *ofr, *ifr;
{
     register float *spectrum;
     register float *rvec, *ivec;
     register float a,b;
     register int r, j;
     int  nrows=64,ncols=256;
     int  index,offset,logrows=6,logcols=8,hnrows=32,hncols=128;

     spectrum = ofr;
     rvec = ifr;

     ivec=(float *)halloc(16384,sizeof(float));

     storesize=0;constsize=0;
     fft_2dgen(rvec,ivec,logrows,logcols);

     for(r=0;r<nrows;r++) {
       index = ((r+hnrows)%nrows)*ncols;
       for(j=0;j<ncols;j++) {
         offset = (j+hncols)%ncols;
         a= *(rvec+index+offset); b= *(ivec+index+offset);
         *spectrum++ = (sqrt(a*a+b*b))/16384;
       }
     }
     free (ivec);
}
/* ---------------- a subroutine of fourtr --------------- */
```

B.5 The function subroutine *fft_2dgen()*

```
fft_2dgen(rvec,ivec,logrows,logcols)

float *rvec,*ivec;
int logrows,logcols;

{
     int i,rows,cols,size;

     rows = 1<<logrows;
     cols = 1<<logcols;
     size = rows * cols;
     for (i=0;i<size;i+=cols)
       fftn(rvec+i,ivec+i,logcols,1);
     for (i=0;i<cols;i++)
       fftn(rvec+i,ivec+i,logrows,cols);
}
/* ------------------- a subroutine of fft_2dgen ---------------
*/
```

B.6 The function subroutine *fftn()*

```
fftn(rvec,ivec,loglen,nskip)

float *rvec,*ivec;
int loglen,nskip;

{
     register int i,j,k,l,n;
     int nv2,nm1,le,le1,c,nle;
     register float *rveci,*rvecj,*iveci,*ivecj,t;
     float ur,ui,wr,wi,tr,ti ;


     if(loglen==0)
       return;
     n=1<<loglen ;
     nv2=n >> 1 ; nm1=n-1 ; j=0 ;
     if (storesize<nv2) {
       if ((0==(cconst=(float *)calloc(nv2,sizeof(float)))) ||
           (0==(iconst=(float *)calloc(nv2,sizeof(float)))))
         perr("Not enough core for fftn");
       storesize = nv2;
     }
     if (constsize!=nv2) {
       constsize = nv2;
       wr =  cos(2*pi/n);
       wi = -sin(2*pi/n);
       cconst[0] = 1.;
       iconst[0] = 0.;
       for (i=1;i<nv2;i++) {
         cconst[i] = wr*cconst[i-1] - wi*iconst[i-1];
         iconst[i] = wr*iconst[i-1] + wi*cconst[i-1];
       }
     }
     for (i=0;i<nm1;i++) {
       if(i<j) {
         rveci=rvec+i*nskip ; rvecj=rvec+j*nskip ;
         t=(*rvecj) ; *rvecj=(*rveci) ; *rveci=t ;
         iveci=ivec+i*nskip ; ivecj=ivec+j*nskip ;
         t=(*ivecj) ; *ivecj=(*iveci) ; *iveci=t ;
       }
       k=nv2 ;
       while (k<=j) {
         j-=k;
         k>>=1;
```

```
        }
        j+=k ;
      }
      le=1 ;
      for (l=0;l<loglen;l++) {
        le1=le ; le+=le ; c = 0; nle = n/le;
        for (j=0;j<le1;j++) {
          for (i=j;i<n;i+=le) {
            if(i+le1>=n)
              perr("fftn: strange index=%d",i+le1);
            rveci=rvec+i*nskip ; rvecj=rvec+(i+le1)*nskip;
            iveci=ivec+i*nskip ; ivecj=ivec+(i+le1)*nskip;

            if (c==0) {
              tr = *rvecj;
              ti = *ivecj;
            }
            else {
              tr = *rvecj*cconst[c] - *ivecj*iconst[c];
              ti = *rvecj*iconst[c] + *ivecj*cconst[c];
            }
            *rvecj = *rveci - tr;
            *ivecj = *iveci - ti;

            *rveci += tr;
            *iveci += ti;
          }
          c += nle;
        }
      }
}
/* ------------------------------------------------------------
*/
```

B.7 The function subroutine *reduce2()*

```
reduce2 (ofr, ifr)
/* reduce the fft spectrum by a factor of 2 */

float *ofr, *ifr;

{
     register float *red_spec, *spectrum;
     register int r,c,roffset,coffset;
     int rrough,rprecise,crough,block;

     red_spec = ofr;
     spectrum = ifr;

     for (r=0;r<4096;r++) *(red_spec+r) = 0.0;

     for (r=0;r<32;r++) {
       rrough = r*2;
       for (c=0;c<128;c++) {
         crough = c*2;
         block = r*128+c;
         for (roffset=0;roffset<2;roffset++) {
           rprecise = (rrough+roffset)*256;
           for (coffset=0;coffset<2;coffset++)
             *(red_spec+block) +=
*(spectrum+rprecise+crough+coffset);
         }
       }
     }

     for (r=0;r<4096;r++) *(red_spec+r) /= 4;
}
/* ------------------------------------------------------------ */
```

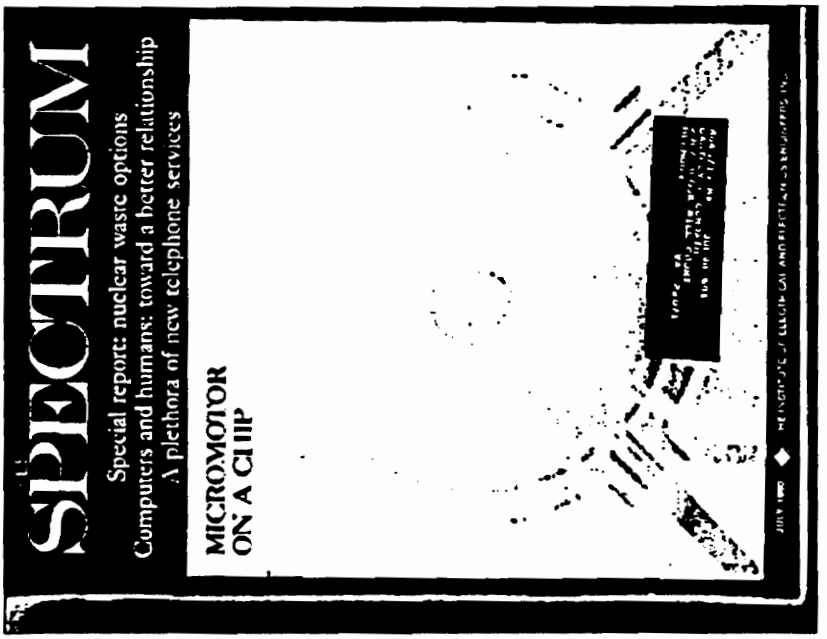
B.8 The function subroutine *nn_class()*

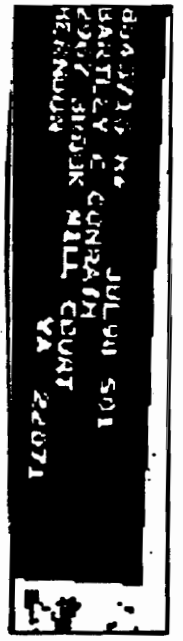
```
float nn_class (red_spec)
    /* neural network classification for a 128 by 32 reduced fft
of a 256 by 64 input binary image */

float *red_spec;

{
    network_initialize(DUMPFILE,0);

    /****** nn_stuff *******/
    flat_set_input(red_spec);
    flat_propagate_forward();
    return (*(flat_get_output()));
}
```

B.9 The program Makefile

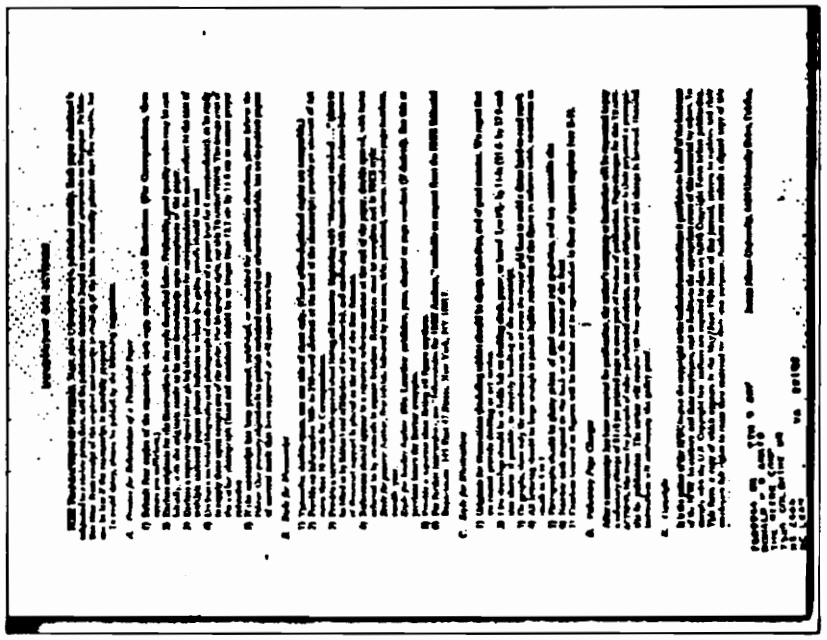
```
NNDIR = Your_working_directory

NETWORK = $(NNDIR)/flat
# "flat" is my Aspirin neural network description file

#Include Aspirin libraries
LIBS= $$NNTOOLS/lib$(TARGET_ARCH)/bpSim.a -lm

# Include my prototype description file
HEADER = '"$(NNDIR)/flat.h"'

# Include my weight space file "Network.save"
DUMPFILE = '"$(NNDIR)/Network.save"'

# More include files
STUFF = -I$$NNTOOLS/include -DHEADER=$(HEADER)
-DDUMPFILE=$(DUMPFILE)

# Make the file
abl_nn_class75demo : abl_nn_class75demo.c $(NETWORK).o
     cc -O4 -fsingle $(STUFF)  abl_nn_class75demo.c -o
abl_nn_class75demo -I/usr/local/include -lhipl $(NETWORK).o
$(LIBS)
```

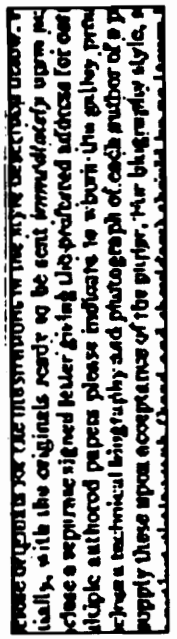
APPENDIX C: TEST IMAGES

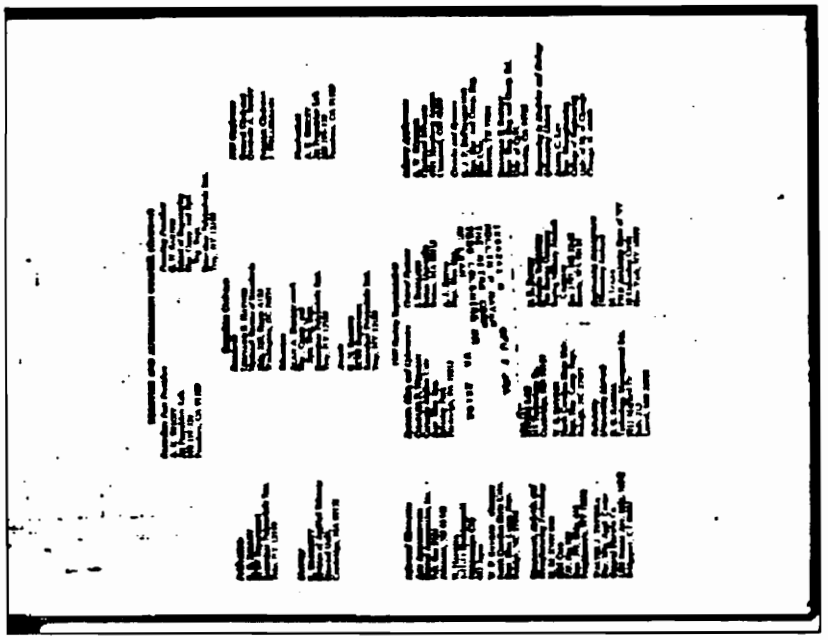


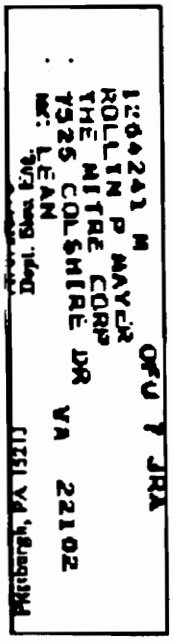


APPENDIX C: TEST IMAGES (continued)

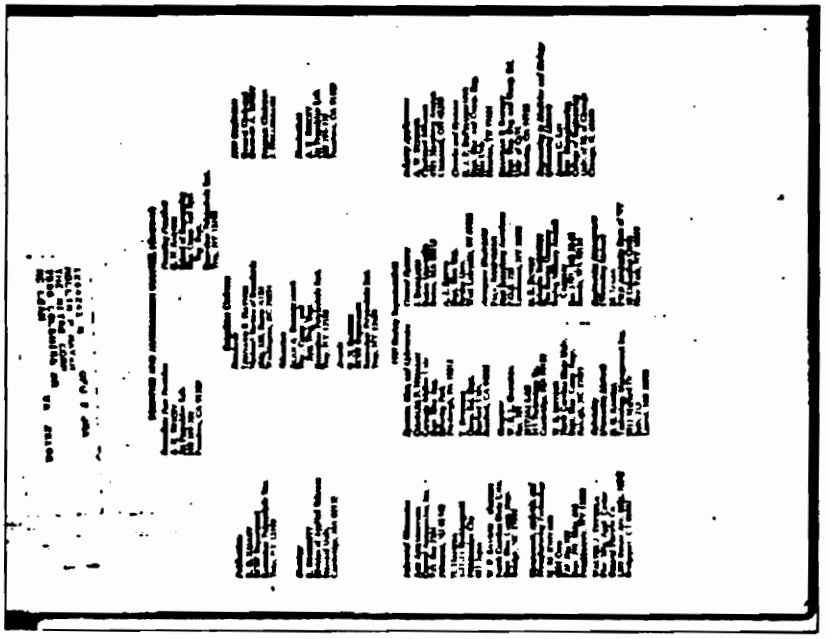




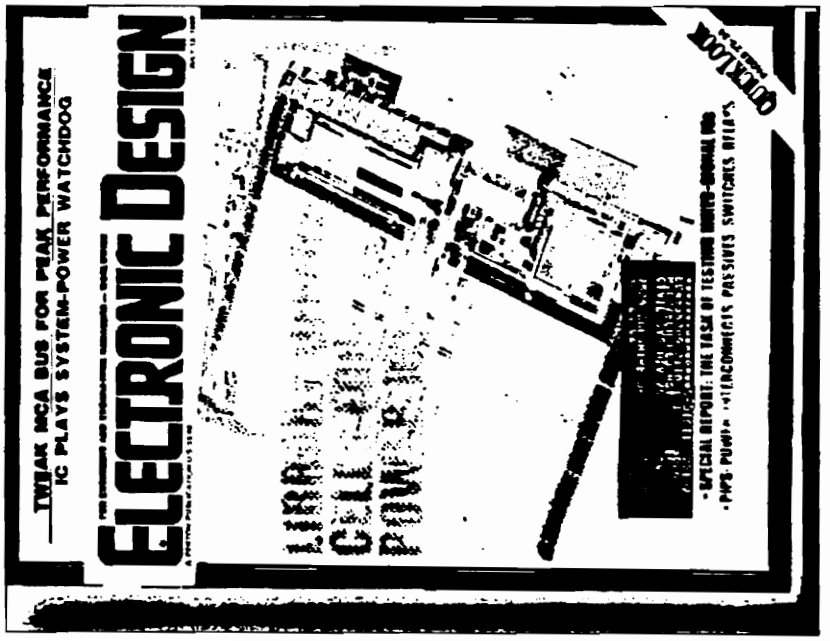




APPENDIX C: TEST IMAGES (continued)

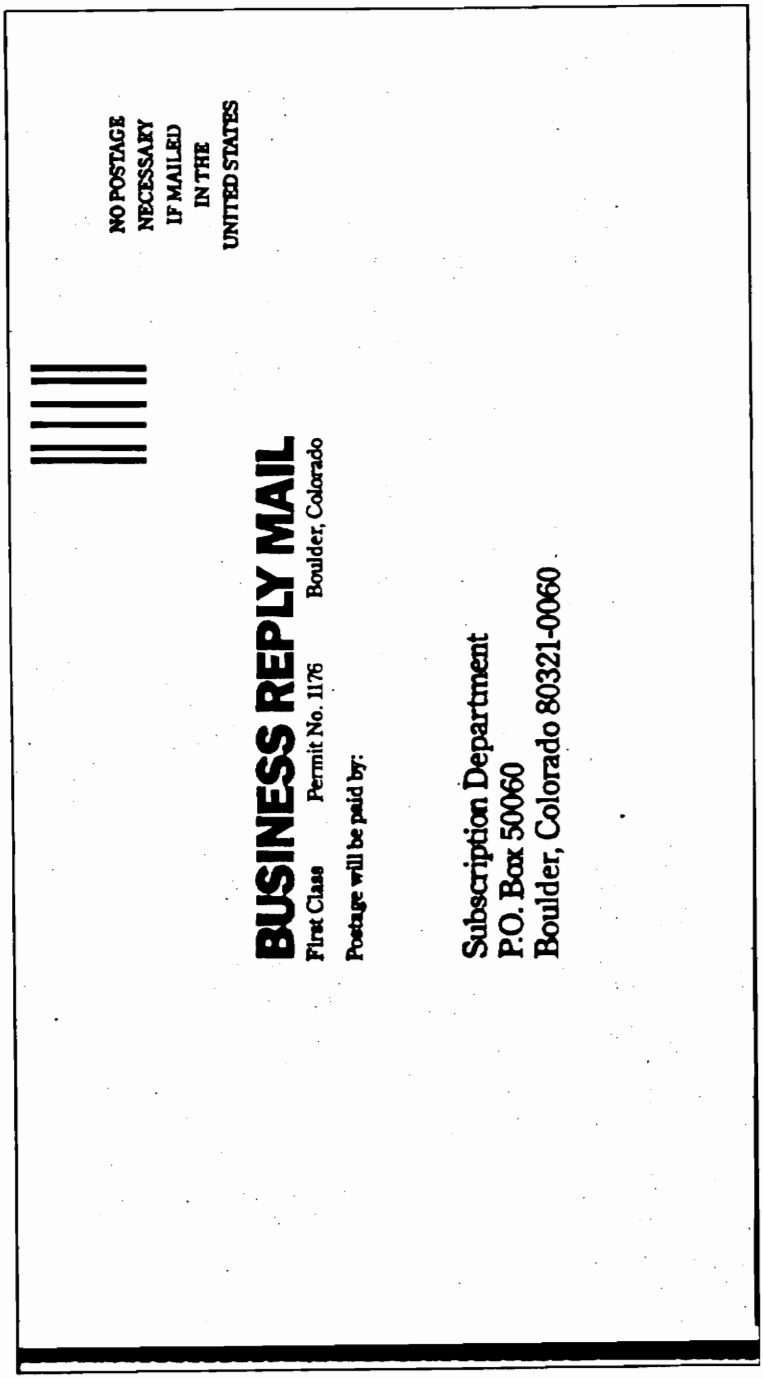


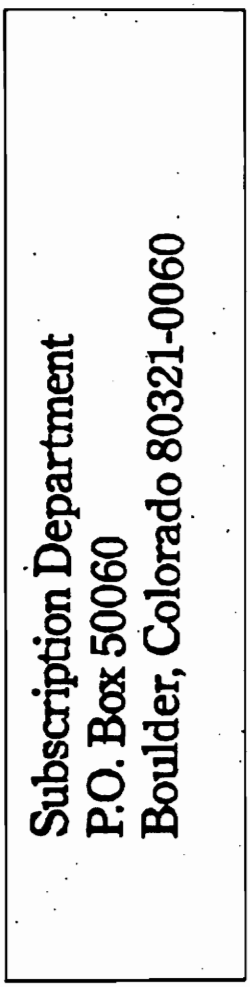
```
OFU 7 JRA
1:04241 M
ROLLIN P MAYER
THE MITRE CORP
7525 COLSHIRE DR
MC LEAN VA 22102
```



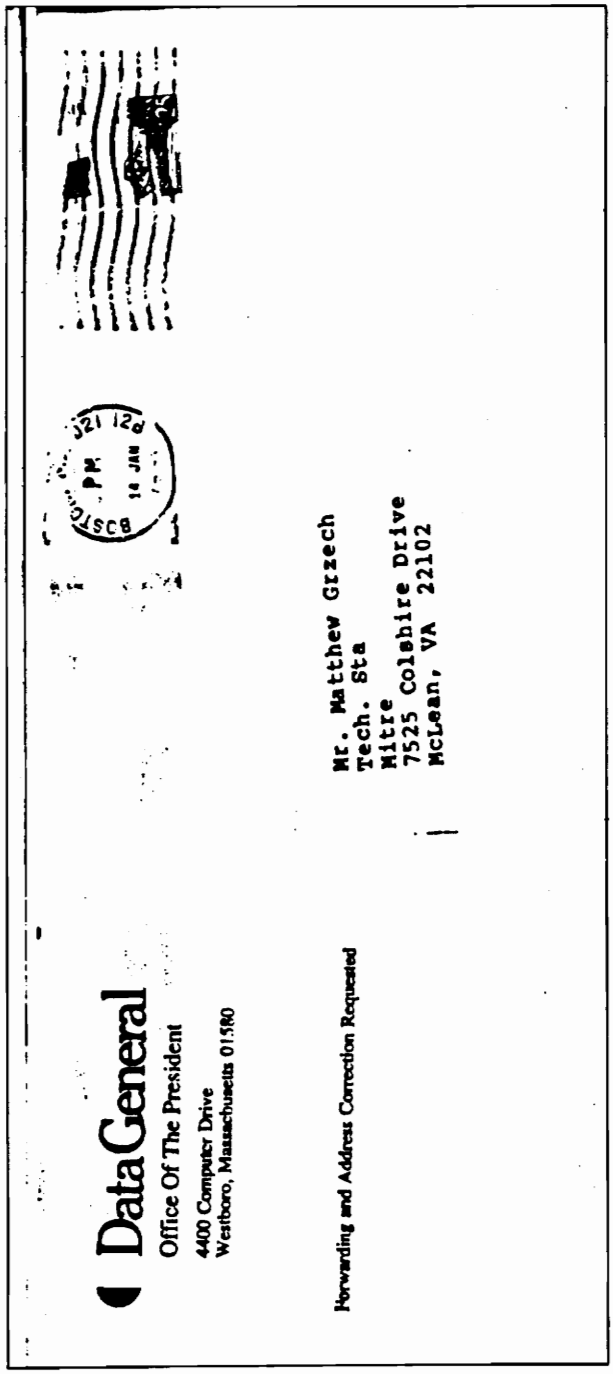

```
*****5-DIGIT 22102
EC4555209 2110 171
M:C27:CH MEO MGT
CIVIL SYS DIV 25
7525 COLSHIRE DR
```

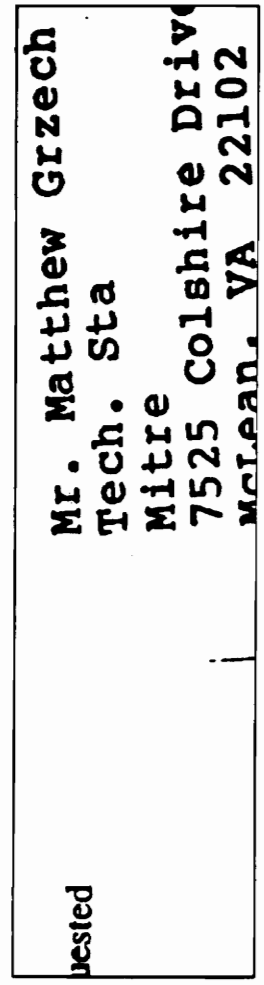
APPENDIX C: TEST IMAGES (continued)



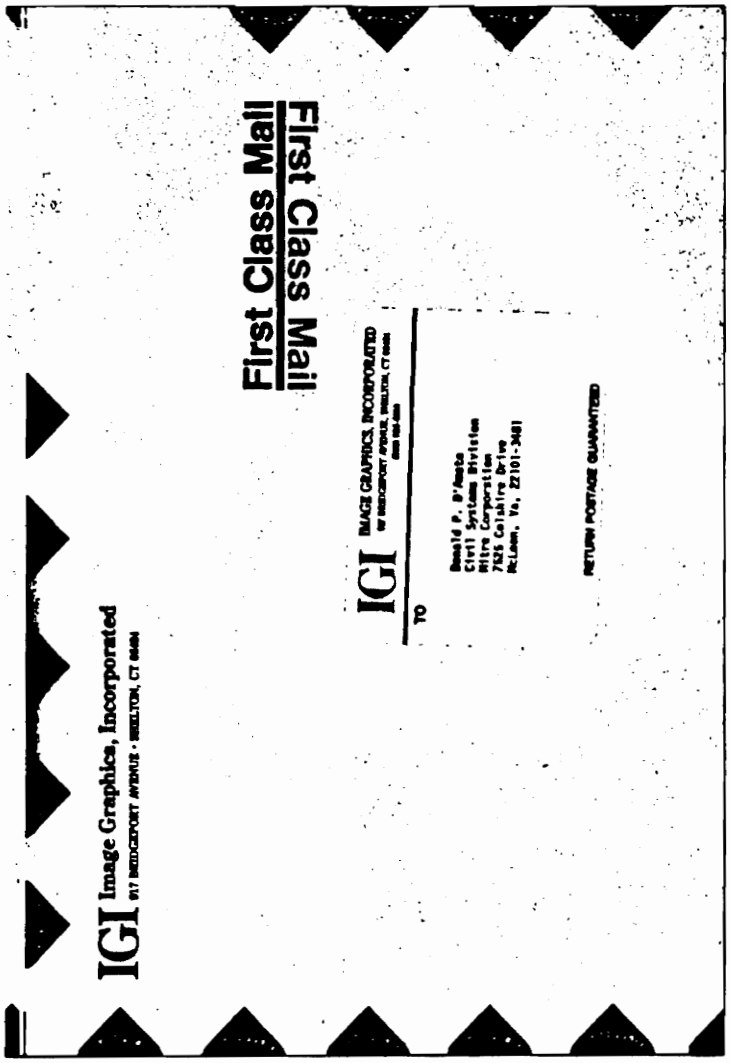


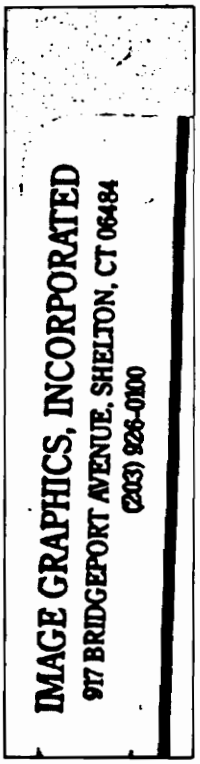
APPENDIX C: TEST IMAGES (continued)





APPENDIX C: TEST IMAGES (concluded)





VITA

Matthew P. Grzech was born in San Diego, California on February 15, 1962. He received his B.S.E.E. degree from The George Washington University in 1985. He is a Member of the Technical Staff of The MITRE Corporation. Recently, Mr. Grzech designed an experimental Address Block Location image processing system on MITRE Sponsored Research (MSR) funds. Prior to this, he worked in morphological image processing of binary document images. He also worked in Very Large Scale Integrated (VLSI) design where he implemented a MITRE patented Content Addressable Memory (CAM) that has variable string length matching capabilities. His professional interests include image processing, neural networks, and VLSI design.

Matthew P. Grzech

Matthew Phillip Grzech