1989

The Application of neural networks to character recognition based on primitive feature detection

Michael Pistacchio

Rochester Institute of Technology
Computer Science Department

The Application of Neural Networks to Character Recognition Based on Primitive Feature Detection

by

Michael Pistacchio

A thesis, submitted to
The Faculty of the Computer Science Department in partial fulfillment of the requirements for the degree of
Master of Science in Computer Sciences

Approved by: _____
Prof. Peter G. Anderson

_____
Prof. Stanislaw P. Radziszowski

_____
Prof. John A. Biles

December 27, 1989

The Application of Neural Networks to Charater Recognition Based on Primitive Feature Detection

ABSTRACT

This thesis investigates a character recognition method inspired by the premise that humans recognize shapes using their ability to assimilate a set of primitive features. These features collectively create a higher level shape of a certain category. The primitive features employed in our method include horizontal, vertical, diagonal lines, and corners of various orientations positioned at various places within a character. Combinations of these features form categories of characters to be recognized. The basic approach consists of preprocessing a character bitmap, extracting primitive features to form a feature vector. The feature vector is then input to a classification neural net. Based on weights derived during training, the system selects the character most closely identified by the feature vector. The advantages of this approach are the speed of training and recognition (as opposed to methods which continually iterate to the final solution), and robustness of the "blurring" effect realized by transforming a character bitmap to an array of features, rather than attempting template matching at the bitmap or pixel level.

To support this study, a graphics workstation based environment has been developed, equiped with 3000 16X16 pixel characters, suitable for experimentation.

TABLE OF CONTENTS

1. INTRODUCTION

In this thesis we investigate a method using neural networks to solve character recognition problems. It is based on the observation of a set of "pre-wired" features used by the mammalian vision system, not developed by learning, but by a million years of evolution. The learning which results during an individual's lifetime to facilitate the classification of image patterns is based on these existing "primitive" features [2]. The natural perception of images is derived from or composed of these primitive features, rather than of pixels. This thesis will present a neural model which attempts to mimic natural perception by "feature processing" the image pattern, which results in a set of predefined line-like features of different lengths and orientations. The evolutionary learning process will be modeled by a "pre-wired" feature extraction net. The classification net models the part of the process that must be subsequently learned by humans. This net must be capable of associating a set of primitive features with identifiable character classes. The network parameters are derived during a training or learning process which is described below.

The algorithm performed very well with respect to lateral translation, 10° rotation, and the application of random noise (10% and 20%). Recognition accuracy for these tests was in the range of 80 - 99% correct first choice. Successful results were achieved with a minimal amount of training samples.

2. CHARACTER RECOGNITION ALGORITHMS AND APPLICATIONS

2.1 Applications of Character Recognition

A reliable character recognition mechanism has many industrial and commercial applications. The banking industry is a front-runner in this regard. Character recognition has been used for years in the sorting of bank checks and other documents, where an account number is recognized, and subsequent processing is initiated. The post office uses character recognition to sort mail. Although a significant percentage of recognition attempts result in characters which are unreadable, the system's cost effectiveness can still be demonstrated due to the immense volume of mail. A bandwidth reduction for facsimile systems could be achieved by recognizing characters before they are transmitted over the communication channel. The 8-bit ASCII code could be transmitted rather than the character bitmap if the character could be confidently recognized; otherwise the bitmap information is sent. Character recognition systems can be employed as reading aids for the blind.

Errors of two types can exist in character recognition systems: rejection and substitution. A substitution error occurs when a character is erroneously identified as another (the other character usually possesses similar shape characteristics). A rejection error occurs when the system cannot accumulate enough evidence to confidently identify a particular character sample. Obviously, a system in which tight controls

are placed on the generation of characters, which causes a reduction in the variation between samples of the same class, will be more accurate than an environment in which sample variation is much larger (e.g., handwritten characters). Automatic recognition systems are much faster than human recognition, but humans can recognize a wider variety of characters, and tolerate more noise and character distortion. Humans also use a broader context ("world knowledge") which aids their recognition performance.

2.2 Statistical Character Recognition

In statistical character recognition (or "decision-theoretic"), classification is based on selected features extracted from the input pattern, rather than a simple application of template matching [7]. Therefore, character recognition is accomplished by solving two fundamental problems. First, determine the features to extract from a character. Second, choose an algorithm to process the features, and classify the character. The selection of initial features, while crucial to the performance of the character recognition system, remains largely empirical and ad hoc, and draws upon the designer's knowledge about the problem [7]. The number of features should be as small as possible, and invariant to minor deviations. The classification problem is regarded as a mapping from feature space to decision space. N measured features produce an N-vector X for an input character. Each possible value of X must then be mapped to a class in the decision space.

Linear Discriminant Functions employ a linear combination of feature measurements, x(1), x(2), ... x(N), and weights, w(i,1), w(i,2), ... w(i,N); w(i,k) represents the weight associated with class i and feature vector element k. The decision function for class i can be expressed as:

$$D(i,X) = \sum_{k=1}^{N} [\ w(i,k)\ x(k)\ ] + \text{constant}(i),$$

The Minimum Distance Classifier is a technique which measures the distance between a sample character, and a set of reference characters, and selects the closest class. The distance metric is defined as $|\ X - R(i)\ |$, where X is the sample feature vector and R(i) is the feature vector for the reference class. Hamming distances, geometric norms, and Manhattan distances could subsequently be calculated.

Another statistical approach, Bayes Parametric Classification, assumes x(1), x(2), ... x(N) are random variables, where x(i) is the noisy measurement of the ith element in the feature vector. Given m character classes, c(1), c(2), ... c(m), assume that the multivariate probability distributions are known. Then the probability of X occuring given c(i), P[X/c(i)], and the probability of classes c(i) occuring, P[c(i)], can be discerned. As a result, the decision rule utilized to select the correct class for the current feature vector X is to determine the value of i for which P[c(i)]*P[X/c(i)] is maximized. The above equation is derived from Bayes law of conditional probabilities:

P(A/B) = P(A&B) / P(B), therefore

P(A&B) = P(A/B) * P(B)

which reads, the probability of A and B occuring simultaneously is equal to the probability of A given B, times the probability of B. The success of this approach heavily depends upon the initial underlying assumptions of probability density functions.

Another statistical tool, the Nearest Neighbor Rule, is based on the k nearest neighbors of a sample vector X. Let S = [X(i),c(i)] be a set of sample feature vectors, X(i) belonging to class, c(i). Given a new sample X', the decision rule is based on the number of nearest neighbors within S for each class, c(i), out of the k nearest neighbors to X'. Of course, the success of this technique is based on the assumption that S is a representative sample of the population.

2.3 Syntactic Character Recognition

Syntactic character recognition is based on the premise that a character pattern is composed of several subpatterns or primitive patterns. Syntactic character analysis consists of selecting primitive components from a set of training samples, and formulating a structural description (using a grammar) based on these primitives. Syntactic recognition of a sample consists of extracting a character's primitive components, and performing a syntax analysis based on those components, and the grammar which may be derived during training. The grammars used are similar to those used in the theory of compiler construction.

Recognition algorithms exist which test for the occurrence of certain sets of subpatterns in various combinations. Classification decisions are made based on the success of these tests.

It could be argued that our neural net approach is a hybrid of the statistical and syntactic philosophies, because the feature vector is composed of primitive character shapes, which collectively synthesize a particular character class. Once these primitive components or features have been extracted, recognition is based on a Linear Discriminant Function.

2.4 Character Parameterization Techniques

An approach to character recognition exists in which ad hoc measurements are obtained from sample characters, and then input to recognition algorithms. For example, the number of intersections on a random line, or the ratio of lengths of character limbs may be useful in discriminating characters.

A technique using character profiles exists which encloses a character in a bounding rectangle, and measures the distance from each of the four bounding sides to the first black pixel. The result is a complete character profile which could be input to a minimum distance classifier.

Methods such as these are not addressed in our neural net algorithm. A suggestion for future study is to couple these ad hoc measurements with the shape features evaluated to improve accuracy.

2.5 Neural Network Approaches to Character Recognition

Artificial neural net models have been applied to many types of pattern recognition problems, including character recognition. One approach consists of applying a bitmap generated from a binary input pattern directly to a neural net system designed to classify or categorize the input. The Hamming and Hopfield networks described in [1] work in this manner. The three layer perceptron using the Back-Propagation learning algorithm will initially accept the bitmap, and subsequently extract its own features before a classification is attempted.

The Hopfield net typically has N bipolar (+1 and -1) nodes. The output of each node is connected to every other node in the system. To store M patterns in such a network, the weights, t(i,j), from node i to node j are initialized to:

$$t(i,j) = \sum_{s=1}^{M} [\, x(i,s) * x(j,s) \,] \text{ , with } t(i,i) = 0$$

where x(i,s) is the ith pixel of exemplar class s and is associated with node i in the network. Character recognition is achieved by applying an unknown input pattern to the net, and iterating until convergence using the following formula (f[] is a non-linearity, such as the signum, applied to the final result):

$$u(j,T+1) = f\left[\sum_{i=1}^{N} \left(t(i,j)*u(i,T) \right)\right],\ j = 1 \text{ to } N$$

where u(j,T) denotes the output of node j at time T.

Our neural net system differs from this approach in three respects:

* the character bitmap is preprocessed before a classification attempt is made (e.g., features are extracted)
* weights are modified as a result of the analysis of sample characters during training (not just initialized)
* the algorithm studied in this thesis does not iterate to arrive at the final conclusion (e.g., recognition is achieved after one pass)

The Hopfield net, as described above, is very unstable if several classes share many bits in common (e.g., the exemplar database contains correlated patterns).

The Hamming net bases its "classification scores" on Hamming distances, the number of pixels that do not match a particular exemplar. The weights, w(i,j), from input i to node j are intialized to:

$$w(i,j) = x(i,j) / 2,$$

where x(i,j) = the value of the ith element of exemplar j. Node values and inputs are equal to +1 or -1. For all practical purposes, the weights in a Hamming net store the values of the exemplar characters. The Hamming distance is found by multiplying the input vector by the weight vector for each class:

$$\text{output of node } j = \sum [\, w(i,j) * y(i) \,],$$

where y(i) is the ith element of the input vector. The class with the minimum Hamming distance, e.g. the maximum correlation, is the one that is selected. However, a human could perceive a given pattern as a very good match for a given exemplar, yet the bitmap of the input pattern may be a large Hamming distance from the correct exemplar. The Hamming and Hopfield nets both would indicate a poor match (Figure 1). These nets deal with noise but not distortion.

This problem is solved by the neural net system studied in this thesis. It attacks the problem of character recognition in a manner similar to the Hamming net. Weights are initialized in a similar manner. Instead of applying the character bitmap to the net, and calculating the Hamming distance or maximum score, features are extracted and stored in a one dimensional feature vector, which is subsequently applied to the net in an attempt to evaluate the class with the minimum distance. Another difference is that weights are further modified as a result of the analysis of sample characters during training.

The Hamming and Hopfield neural net approaches deal specifically with binary input, as does our network. The three layer perceptron using the Back-Propagation learning algorithm is an example of a neural net structure which handles continuous valued input. Despite this dissimilarity, it possesses one common denominator worth mentioning. The three layer perceptron is a feed forward structure, where the output of each node is calculated as follows:

$$\text{output of node } j = f\,[\, \sum (\, w(i,j) * x(i) - C(j) \,) \,],$$

where f = a non-linearity

w(i,j) = the weight of the connection from input i or lower neuron i and neuron j

x(i) = the value of input i or the output of lower neuron i

C(j) = the threshold constant of node j.

Using Back-Propagation, the error which results from the highest layer of neurons, during the supervised learning phase, is used to adjust the weights proportionately. In many cases, thousands of iterations are required to train the system. As we will show below, the neural net algorithm studied in this thesis learns in a fraction of the iterations. The second, or hidden layer in the multilayer perceptron structure, assumes a role similar to that of the one dimensional feature vector used in our neural net. This middle layer can be interpreted as the "feature layer", synthesized during training. Each node in this middle layer represents one particular feature. The features obtained using this approach were selected by the algorithm, in response to the sample characters applied during training. The major disadvantage of this approach is the time necessary for the network to learn which features are important in recognition.

3. PROJECT DESCRIPTION

3.1 Overview of System Architecture

Figure 2 depicts the architecture of our system. The image bitmap is input to the primitive feature detector, designed to process raw data and output predefined features. The feature detector can be described as a set of subnets which fully operate in parallel, where each subnet represents a particular feature. The software which simulates this neural net does not operate in parallel. Parallelism is easily achieved by implementing the net in hardware. Each subnet takes its input from a predefined set of pixel locations which are used to detect one specific feature. The subnets detect features as a result of neurons firing in parallel, when their threshold values are exceeded. Each subnet consists of multiple layers of neurons, where the highest layer neurons fire to indicate the presence of a particular feature. The primitive features, or outputs of the highest layer neurons in the primitive feature detector net, are input to another neural model which classifies the pattern in terms of a set of predefined exemplars or categories. This classifier net must participate in a supervised learning algorithm, such that its weights are adjusted according to an applied training set, and the association of this training set to a finite set of predefined categories. An association matrix W is made up of weights, W(i,j), which measure the degree to which a particular feature, F(i), discriminates for a specific category, C(j). W(i,j) is an

indication of the relationship between feature i and category j. In general, if W(i,j) > 0, then feature i tends to be present in category j; otherwise feature i tends not to be present in category j. The magnitude of this weight determines the feature's relative importance in discriminating for the current category. As explained below, the training algorithm strives to accentuate the differences in the set of categories, while not inhibiting the similarities [2].

3.2 Primitive Feature and Sub-Feature Detection

3.2.1 The Window Location Feature

The bitmap for each character that is presented to the system is composed of a 16X16 pixel matrix. A logical 1 indicates a black pixel, and a logical 0 indicates a white pixel. A coarse spatial coordinate location is associated with each primitive feature, which indicates where that particular feature exists within the 16X16 bitmap. A spatial coordinate location is defined as an NXN "window" of pixels currently being considered for feature extraction. Several windows exist, each at a specific location, spanning the entire character matrix. To correct for possible errors occuring due to features which exist on window boundaries, the windows overlap by two pixels in the horizontal and vertical directions. 5X5, 6X6, and 7X7 window sizes (in pixels) are examined in this study. The original 16X16 character matrix will be padded by one or two pixels on either

side, to accomodate window sizing. Figure 3 depicts the 6X6 window geometry. Pixels used for padding will be assigned values of logical 0 (white). Note that the window dimensions dictate the total number of windows required to span the entire character (e.g., nine 7X7 windows cover the 16X16 character). The feature extractor net will be applied to each window, extracting pertinent features associated with the window's location.

3.2.2 Horizontal Sub-Features

A sub-feature is defined as a particular instance of a feature within a given window. By definition, the existence of one or more sub-features of a given feature type implies the existence of that feature.

Several horizontal sub-feature locations have been defined as shown in Figure 4. The structure is similar for each window size. Each sub-feature location is composed of two pixel rows within the window, and can exist in two possible lengths, long horizontal lines (L), and short horizontal lines (S). The first step in the calculation of a particular sub-feature is to OR the value of pairs of pixels in the same column. The purpose of ORing pixels is to increase the robustness of the feature extraction process, so that slight deviations from the ideal feature shape can be tolerated. Subfeatures extracted from the feature extractor subnet are collectively ORed as shown.

Figure 5 shows the structure of the feature extractor subnet. All neurons presented in this thesis have activation values of either 0 or 1. If the sum of the weights multiplied by the associated activation values of lower layer neurons (or initial inputs) exceed the threshold value of a higher layer neuron for which a series of connections exist, its associated activation value becomes 1 (fires), otherwise it is set to 0. Threshold values are explicitly designated for each node. These values are fixed, and most are a function of window size and feature length. The neural subnet attempts to extract short horizontal lines employing a set of neurons with threshold values of 2.5. Each of these neurons is connected to 4 sets of ORed pixels. Either of these next layer neurons will fire if 3 out of the set of 4 ORed pixels fire. This is accomplished by setting each threshold value to 2.5 (the weight of each connection is 1). Therefore short horizontal lines have lengths of 3 or 4 pixels. All sets of ORed pixels are connected to another neuron designed to extract a long horizontal line. Its threshold value is equal to 4.5, implying that at least 5 of the set of ORed pixels must fire to detect a long horizontal line. The presence of a long horizontal line precludes the existence of a short horizontal line for the sub-feature pairs HSx and HLx. However, it is possible for features HLx and HSy (where x is not equal to y) to coexist. This implementation is accomplished by the negative, or inhibitory link from the neuron which extracts the long horizontal line to the neuron which ultimately indicates the existence of a short horizontal line.

3.2.3 Vertical Sub-Features

The extraction of vertical sub-features completely parallels that of its horizontal counterpart. Several vertical sub-feature locations have been defined for each window size. Each vertical sub-feature location is composed of two pixel columns within the window, and can exist in 2 possible lengths, similar to horizontal lines. These sub-features are referred to as VS1/VL1, VS2/VL2, VS3/VL3, etc. Pixels in the same row are ORed together in a similar fashion, as in the horizontal case. The structure of the neural subnet which extracts vertical sub-features is identical to the one which extracts horizontal sub-features.

3.2.4 Diagonal Sub-Features

Diagonal lines (+/- 45° orientation) are also output during feature extraction. DPS and DPL are terms which represent +45° diagonal lines (P for Plus) of short and long lengths, respectively. DNS and DNL are used to designate -45° lines. Like horizontal and vertical sub-features, short lines are composed of 3 or 4 pixels, and long lines are derived from 5 to 7 pixels. Adjacent diagonal rows are not ORed in the extraction of diagonal features, unlike horizontal and vertical sub-features. Each diagonal row is considered individually. Figure 6 depicts the location and nomenclature

regarding the subfeatures of DPS and DPL. All diagonal sub-features can potentially exist in two locations within one window (due to symmetry), except for the diagonal line that extends from the lower left corner of the window, to the upper right corner of the window. Feature extraction is implemented utilizing subnets similar to the horizontal case (Figure 5).

3.2.5 Corner Sub-Features

3.2.5.1 Corner Sub-Features Derived from Horizontal / Vertical Lines

When a short horizontal and vertical line intersect, a corner sub-feature, CHSx exists (where x defines the orientation from 1 - 4). Figure 7 displays four orientations, identified by the relative location of the vertex. CHLx defines the long version of this sub-feature, and is extracted in the same manner, regardless of window size. Certain combinations of HL and VL derive particular orientations of CHLx.

The calculation of sub-feature CHSx (the short version of this type of corner) is achieved utilizing a subnet which identifies the vertex of a corner for a given orientation. Feature extraction subnets exist for each orientation. CHLx and CHSx cannot both be present within a given window.

3.2.5.2 Corner Sub-Features Derived from Diagonal Lines

When a +45º and -45º diagonal line intersect, a corner sub-feature, CDSx exists (where x defines the orientation from 1 - 4). Figure 8 defines four orientations for this sub-feature, identified by the relative location of the vertex. This feature has no long (L) version, since it is impossible for this corner to be derived from two long lines of 5 pixels or greater (within the boundaries of the defined windows). The calculation of sub-feature CDSx is achieved utilizing a subnet which identifies the vertex of a corner for a given orientation. Feature extraction subnets exist for each orientation. This feature is extracted in the same manner, regardless of window size.

3.3 Feature Vector Construction

Once the sub-features are extracted from the image, a feature vector is then constructed. This vector consists of a one-dimensional array, where each element of the array represents an individual feature. If the feature is present in the character image then its value equals +1, otherwise -1 is used to indicate the absence of a particular feature.

A feature vector is composed of 20 bipolar elements (features) per window location. They are as follows:

HS, HL, VS, VL, { horizontal / vertical lines }

DPS, DPL, DNS, DNL, { +—45° diagonal lines }

CHS1, CHL1, CHS2, CHL2, { corners formed by Hor/Ver lines}

CHS3, CHL3, CHS4, CHL4,

CDS1, CDS2, CDS3, CDS4 { corners formed by diag lines }

Sub-features are ORed to determine the existence of features. For example, sub-features HL1, HL2, HL3, etc. would be ORed to produce a value for feature HL. Other features are calculated in a similar fashion. As a result, as long as one instance of the sub-feature exists within a given window, the feature exists within that window. The size of the feature vector can be found by multiplying the number of windows X 20. The smaller the window size, the more windows needed to span a character, hence the larger the feature vector (a good reason why window size should be maximized). For example:

5X5 case => 20 features/window X 25 locations =

500 features in feature vector

6X6 case => 20 features/window X 16 locations =

320 features in feature vector

7X7 case => 20 features/window X 9 locations =

180 features in feature vector

A feature vector is constructed in this manner to accomodate slight variations in sample characters. The system determines if a particular feature exists, and coarsely, where this feature exists. For example, it will suffice to know that a long vertical line exists (VL = +1), as opposed to knowing which of the sub-features (VL1, VL2, VL3,

etc.) caused VL to trigger. As a result, the system is able to tolerate slight variations concerned with the position of various sub-features within the bitmap. Sub-features that exist near window boundaries may be detected in both windows, due to window overlap. Therefore, to some extent, the positioning of various sub-features will affect the feature extraction results of neighboring windows.

3.4 The Learning Process

Our learning algorithm is modeled after a strategy developed by Krishnan and Walters [2], which was derived from a method by Anderson [3]. The fundamental objective of the algorithm is to develop an association matrix of weights (W(i,j)) which relates a specific feature, F(i), to a given category, C(j). A category is one of the defined exemplars for the character set being identified. W(i,j) > 0 if F(i) tends to exist in C(j); W(i,j) < 0, if not. The magnitude of W(i,j) reflects the degree to which F(i) discriminates for C(j). A large positive W(i,j) indicates that F(i) is a distinguishing feature for C(j). A large negative W(i,j) indicates that the presence of F(i) in an input image strongly suggests that C(j) is not its correct classification. Also, the absence of F(i) indicates the possible presence of a C(j) character, when W(i,j) has a large negative value.

During the training process, a feature vector, F(1..N), obtained from a sample character, is associated with its known category, C(k). By

definition, for j = k, C(j) = +1, and for all j != k, C(j) = -1. The learning rule is as follows:

Case	F(i)	C(j)	W(i,j)	Learning Rule (delta W(i,j))
0.	- 1	- 1	<= 0	0
1.	- 1	- 1	> 0	B*F(i)*C(j)
2.	- 1	+ 1	<= 0	A*F(i)*C(j)
3.	- 1	+ 1	> 0	A*F(i)*C(j)
4.	+ 1	- 1	<= 0	B*F(i)*C(j)
5.	+ 1	- 1	> 0	0
6.	+ 1	+ 1	<= 0	A*F(i)*C(j)
7.	+ 1	+ 1	> 0	A*F(i)*C(j)

Note: F(i) = - 1 signifies feature i IS NOT present in the sample character bitmap

F(i) = + 1 signifies feature i IS present in the sample character bitmap

C(j) = - 1 signifies that this category IS NOT being associated with the sample character bitmap

C(j) = + 1 signifies that this category IS being associated with the sample character bitmap

W(i,j) is initialized to +1 if F(i) IS contained

within the exemplar of C(j), and -1 otherwise

"A" and "B" or Alpha and Beta refer to gain factors

As mentioned above, this learning rule tends to accentuate the differences between several categories, while not inhibiting common features [2]. Assuming A=B=1, all cases, except for 0 and 5, modify W(i,j) by the product F(i)*C(j), over all i and j. Case 0 does not alter W(i,j), since the product F(i)*C(j) would make W(i,j) less negative. This would be undesirable because one of the goals of the training algorithm is not to inhibit common features. If within a sample character, F(i) does not exist, and its known category is other than j (e.g., C(j) = - 1), and W(i,j) < 0, then the commonality between C(j) and the current sample character is the fact that F(i) is not in either. This is because, since W(i,j) < 0, F(i) tends not to be in C(j). By making W(i,j) less negative, the affect of F(i) in C(j) would be lessened. For case 1, W(i,j) > 0, F(i) is in C(j), yet not in the sample character. W(i,j) will become more positive, hence this difference is accentuated. In considering cases 2 and 3, W(i,j) will become more negative for the known category of the sample character, since F(i) is not present in the sample character. Case 4 is another example of differences between the sample's known category, and C(j) being accentuated. F(i) is present in the sample character, yet tends not to exist in C(j), since W(i,j) < 0. As a result, W(i,j) becomes more negative. Case 5 is another special situation where, in an attempt not

to inhibit similiarities, the value of W(i,j) is left unchanged. F(i) is present in the sample character, and in C(j) (a category other than the known category), yet the product F(i)*C(j) would make W(i,j) less positive, thus lessen its affect. Cases 6 and 7 would make W(i,j) more positive for the sample's known category, since F(i) is present in the sample character.

During training, the order in which sample characters are applied to the system is completely randomized, to minimize any bias which may be attributed to sample order.

3.5 The Recognition Process

After extracting the features from the sample character, the feature vector is input to the classifier net, which computes the "scores" for each category. This net is depicted in Figure 9. Each neuron, C(1) through C(m), represents a predefined category. The weights associating the feature vector inputs, F(1..N), and the categories, are obtained from the association matrix, which was derived during training. The value output from each category neuron is calculated by summing the product, F(i)*W(i,j), over all i. The approach is similar to that in the Hamming net [1]. A large positive score will occur when F(i) and W(i,j) frequently have the same sign. In other words, the weight W(i,j) indicates whether or not F(i) is present in C(j). If the inputted feature vector suggests a similar pattern (e.g., indicates feature not present when W(i,j) < 0, and feature present when W(i,j) > 0), then a

high score will result. This correlation is similar to the Hamming distance. The outputs from the category neurons are calculated, and the maximum score reflects the chosen category.

3.6 Simulation/Testing

3.6.1 Measurement Statistics

To investigate system performance, several parameters were chosen and selectively varied. Recognition accuracy (the percentage of correctly recognized characters) was calculated for each combination of parameters. A sample character is correctly recognized when the class with the highest score matches the true class of the sample. It is also important to know which character classes were "runners-up" in the scoring. This information provides a measure of confidence in the choice, and also would be useful to a system which possessed context information about a certain character. The context information may serve to pick the correct character out of the set of characters with the X highest scores. For this reason, the system will tabulate the four highest scores, and calculate the frequency (expressed as a percentage) of the correct character existing within the top X scores (X = 1..4).

Another important statistic, related to the degree of confidence that can be placed on recognizing a given character, is calculated as follows:

(1st place score-2cd place score)/(1st place score) X 100

This statistic measures the delta difference between the 1st and 2cd place score. The average of this statistic over a set of tests is refered to as the "Confidence Factor". In a system where the price of an erroneous recognition error is severe, it may be benefitial to require that this statistic exceed a certain threshold. If not exceeded, the system could flag the given character as unrecognizable.

3.6.2 Parameters

The following is a list of parameters and their potential values which were evaluated to determine their affect on recognition accuracy:

Parameter	Values
1. Window Dimension (# pixels X # pixels)	* 5X5 (25 locations) * 6X6 (16 ") * 7X7 (9 ")

2. Features Dropped - the following sets of features were selectively dropped (e.g., ignored) during various training and testing exercises:

    * none

* length (xxL OR xxS = xxL, do not distinguish between Long and Short)
* short (drop all Short features, xxS)
* DFL,DFS,DNL,DNS (drop all diagonal lines)
* CHSx,CHLx,CDSx (drop all corners)
* HL, HS (drop all horizontal lines)
* VL, VS (drop all vertical lines)
* HL, HS, VL, VS (drop horizontal and vertical lines)

3. Training samples/character

4. Thinning algorithm applied or not applied

5. Selectively varying the gain factors "A" and "B" (Alpha/Beta):

   * A=1 B=4
   * A=1 B=1
   * A=4 B=1

6. Variations in training samples applied to the system during training. Note that each mode of training refers

to operations that are applied to the exemplar bitmap, prior to the feature selection process. Random noise was simulated by changing each black pixel to white with a certain probability. For example, 30% random noise indicates that each black pixel will become white with a probability of 0.30. Each type of training consists of 10 samples/class.

- TRAIN1 - pure exemplar training (no alterations of exemplar characters)

- TRAIN2 - rotation of 10° clockwise and counterclockwise, 10% random noise, 1 unit of lateral translation (up, down, right, and left), 3 pure exemplars

- TRAIN3 - rotation of 30° clockwise and counterclockwise, 30% random noise, 2 units of lateral translation (up, down, right, and left), 3 pure exemplars

- TRAIN4 - rotation of 30° clockwise and counterclockwise (3 of each), 4 pure exemplars

Note: The input character set we have used to evaluate this system consists of 16X16 bit maps (roughly 3000 characters), taken from a Chinese word-processing project at the Shanghai University of Technology.

3.6.3 User Interface

The user interface is derived from the Sunview Windowing package of library routines. Upon system invocation, three main windows appear on the display (Figure 10). The top left window is the control panel which houses the command buttons to perform the following:

* set system parameters, such as window size, dropped features, test or train mode
* reset test or reset train statistics
* display the batch processing subwindow
* show the current feature vector components for the last character that was evaluated
* display recognition accuracy scores for the last character tested (1st - 4th place)
* global system reset
* set rotation direction (clockwise or counterclockwise)
* rotate current character the specified number of degrees (10°, 30°, or 45°)
* apply random noise at indicated percentage to current character (10, 20, 30, 40, 50)
* translate current character per indicated direction (up, down, left, or right one unit)

* partially "cover" the lower right section of the character bitmap (set all ON pixels to OFF in a region defined by a "cover" rectangle of a fixed size - 6 pixels by 6 pixels). This removes an area of pixels - e.g. converts black to white.
* apply thinning algorithm to character
* commit character bitmap, which performs the following:
    - if in train mode, extract shape features and adjust weights
    - if in test mode, extract shape features and attempt to classify (display the top four scores and associated class bitmaps)
* cancel current "fat-pixel" display (see below) of sample character (sets all pixels to OFF)

The upper right window displays a close-up view of the previously selected character, refered to as a "fat-pixel" display. A "fat-pixel" is defined as a 16X16 matrix of pixels which are either all ON or all OFF. This entire matrix represents one pixel of the sample character. The result of executing any of the above character modification procedures (rotation, translation, etc.) is graphically depicted in this window. The user is also able to explicitly toggle individual "fat-pixels" by clicking on the appropriate location. A black fat-pixel will turn white, and a white fat-pixel will turn black. Clicking on the "commit" button will apply the indicated

bitmap to the system for feature extraction, etc. (as described above).

The bottom window contains the set of exemplar character classes (e.g., the nominal shape of the indicated class). Clicking on any one class causes the "fat-pixel" window to display the indicated character. This becomes the sample character to train or test, depending on the current mode. This window, as well as the "fat-pixel" window, are only useful in the interactive state (see below).

The system is in one of two modes, training or testing (recognition). Training mode consists of specifying the category (by clicking on an exemplar), extracting features, and then applying the sample feature vector to the learning algorithm. Each sample directly modifies the association (weight) matrix. In testing mode, the system attempts to classify the character as described in section 3.5, and updates the recognition statistics described in section 3.6.1. The desired set of parameter values (e.g., train/test mode, window size, and features dropped) must be selected before the system learns. The same sets of characters can be processed (trained and tested) with differing sets of parameter values, so researchers can evaluate their affect on recognition accuracy.

Each mode can be exercised in an interactive or batch state. In the interactive state, the character bitmap is obtained and displayed in the fat-pixel window by clicking a mouse cursor on one of the exemplar classes. Once the character has been put into the fat-pixel window, various operations which manipulate the bitmap can be applied

(as described above). Thus, characters can be trained, and then tested in an interactive manner. Clicking on "reset train" causes the association matrix (W(i,j)) to re-initialize. Clicking on "reset test" causes the recognition statistics to clear. "Global reset" causes a reset test and train, as well as setting a flag which allows the modification of parameters (window size, etc.).

Clicking on the "batch" button will display the batch processing subwindow. This window contains a set of four training strategies, and a set of eight (TEST1-8) testing strategies. Clicking on any one of these buttons will cause the corresponding batch routine to execute. The TRAINx routines (described in section 3.6.2) randomly select character classes from the set of all classes, until all classes have been appropriately sampled. The set of all samples from each class undergo the same set of transformations before features are extracted. Character order is completely randomized. The TESTx routines operate on each class in a predefined order, until all classes have been sampled. Recognition statistics are tallied during this process. These routines perform as follows:

* TEST1 - selects each exemplar sample for classification (no operations are applied to the sample)
* TEST2 - each exemplar class is translated one unit (four separate classification tests occur, one for each direction - U,D,R,L)
* TEST3 - each exemplar class is rotated 10° (two tests occur per class - clockwise and counterclockwise)
* TEST4 - each exemplar class is rotated 30° (two tests occur per class - clockwise and counterclockwise)
* TEST5 - each exemplar class is exposed to 10% random noise
* TEST6 - each exemplar class is exposed to 20% random noise
* TEST7 - each exemplar class is exposed to 30% random noise
* TEST8 - each exemplar class is "covered" in the lower right section of the character

3.6.4 Simulation Environment

Hardware: Sun Workstation 3/50 or 3/60

Software: SunOS (Ver 4.0), SunView Windows Library
C language.

4. RESULTS AND CONCLUSIONS

4.1 Introductory Comments

To analyze the performance of our character recognition system, we performed several batch test sets on a constant set of 100 characters (Figure 10). The character set chosen consisted of various characters from a cross section of languages: English, Greek, Russian, Japanese, and Chinese. Each test set was composed of eight individual batch tests, as described above. The following sections present the results and conclusions.

The raw data for batch test #4 (e.g., 30° rotation) was eliminated due to the poor performance of 30° rotation, which was partially caused by the rotation algorithm itself. Rounding errors were introduced because of the need to select the closest pixel in the 16X16 pixel bitmap, which tended to distort the character. The recognition algorithm was largely unable to overcome this deficiency. A smoothing algorithm was applied to the rotation algorithm in an effort to alleviate the rounding problem. Smoothing would turn ON the neighboring pixel if the absolute difference between the rounded integer pixel coordinate location (the new rotated pixel location) and the calculated real number representing the new pixel location, was greater than or equal to 0.4. A round up would turn ON the location one unit less, otherwise the location one unit greater would be turned ON. For example, if the calculation for the new rotated

column value = 5.4, the integer column value gets rounded down to 5. The difference is 0.4, therefore column 6 would also have its pixel turned ON (holding the integer row value constant). The same treatment applies to the rounding of row values. This smoothing algorithm did slightly improve recognition accuracy approximately 2 percentage points, yet performance was still poor relative to the other tests.

During a set of tests designed to analyze a particular parameter, all other parameters are set to their nominal values as follows:

Parameter	Nominal Value
* training set type	TRAIN2
* features dropped	none
* window size	6X6
* thinning algorithm	not applied
* Alpha = Beta	1
* samples per character	10

In general, the algorithm performed very well with respect to lateral translation (up, down, right, and left), 10^{o} rotation, and the application of random noise (10%, 20%). Recognition accuracy for these tests was in the range of 80 - 99% correct first choice. Random noise of 30% dropped this accuracy statistic by roughly 40%.

4.2 Training Set

This particular set of tests were geared towards analyzing the sensitivity of the algorithm with respect to the training set. The objective was to apply all four training sets to the system, and compare results. From the descriptions of each of the training sets, it is evident that training numbers are assigned according to the degree of sample deviation from the exemplar. For example, TRAIN2 performs operations on character samples before training, whereas TRAIN1 does not alter any of the exemplar characters. The operations performed by TRAIN3 are more extreme than those employed by TRAIN2 (e.g., 30° rotation as opposed to 10° rotation). TRAIN4 implements the most sample character distortion, since it only rotates sample characters 30° (no lateral translation). It is assumed, and confirmed by the test results, that lateral translation is less of a problem for the recognition algorithm than 30° rotation. It should be noted that all training sets consist of 10 training samples per class, which include pure exemplar characters (unaltered) randomly mixed within each training set. Tables 1 - 4 contain the recognition accuracy results for each of the four training sets.

The results indicate that TRAIN2 resulted in a slight improvement in recognition accuracy. As sample deviation increased (e.g., TRAIN3 to TRAIN4), performance decreased, yet some deviation from perfect exemplar training (e.g., TRAIN2 as opposed to TRAIN1)

did help to improve the algorithm's robustness. The reason for the slight variation in performance is that the association (weight) matrix is initialized according to the class exemplar. For example, W(i,j) is initialized to +1 if F(i) exists for an exemplar character of class j, otherwise it is set to -1. This has the effect of "jump starting" the weight matrix, and models the theory that humans are born with an evolutionary learning capability, in addition to continued environmental learning (modeled by the supervised learning algorithm). The learning algorithm is such that the evolutionary state of the weight matrix heavily influences the outcome of later environmental or supervised learning. This is because the sign of W(i,j) is set at initialization (the evolutionary part), which is totally derived from the class exemplars. This sign determines the subsequent affect that training samples from other classes will have upon a given class (see section 3.4), and thus minimizes the affect of training with non-exemplar characters (e.g., rotated, translated, etc.). For example, given $W(i,j) \gg 0$, it is of little consequence if W(i,j) is subsequently decremented by 1, because a distorted sample from class j did not contain this popular feature.

The significance of the exemplar initialization of the weight matrix, following feature extraction, will be fully disclosed in section 4.7.

4.3 Alpha (A) / Beta (B) Gain Factors

Gain factors were analyzed by selecting various values for Alpha and Beta, which directly control the incremental change applied to a weight, W(i,j), by the product A*F(i)*C(j) or B*F(i)*C(j). Gain factor Alpha is used when C(j) is the known class of the sample character, otherwise Beta is employed. In other words, if C(j) = +1, Alpha is used, and if C(j) = -1, Beta is used.

Analysis of the results reveal that the recognition algorithm is fairly insensitive to the values of Alpha and Beta. Increasing Alpha slightly degraded performance, yet increasing Beta slightly increased the level of confidence. This improvement does, however, level off as Beta is increased further (maximum improvement is obtained at Beta = 4). In any event, the changes in recognition accuracy, and confidence were insignificant. Altering the values of Alpha and Beta served only to change the magnitude of the final scores achieved by each class during a recognition session in a proportional manner. It did not significantly enhance the systems ability to discriminate characters.

4.4 Window Size

Overall, performance slightly increased with window size, however in this case, statistical averages can be misleading (Tables 5 - 7). The effect of window size depends upon the type of transformation applied to a sample character (e.g., translation,

rotation, etc.). For example, the 5X5 window size was the worst performer during the lateral translation test (TEST2), and the 7X7 window size performed best. This result is consistent with the way window features are incorporated into a feature vector. A lateral translation of a test character will result in more deviation of the feature vector when the window size is smaller, since it is more conceivable that a feature which exists at a given window position, will be translated to the adjacent window location.

The results for a 10^{o} rotation were quite different. The 5X5 window size performed the best, and performance slightly degraded as the window size increased. The random noise tests (TEST 5-7) experienced a slight improvement in performance as window size became larger. The cover test (TEST 8) performed well in the 5X5 case, and was worst in the 6X6 case. It is difficult to draw any conclusions concerning this test, since its success greatly depends on the exemplar character set. This is due to the fact that the character is "covered" in the same place each time. The amount of information lost depends on each exemplar character bitmap. Randomizing this position (upper lefthand corner of the "cover" rectangle) will still cause the results to be somewhat dependent on the exemplar character set.

4.5 Character Thinning

Tests were executed subsequent to the application of a character thinning algorithm, applied to each character prior to the feature extraction process. Thinned characters were also used to train the system. The thinning algorithm searches for a set of predefined patterns, eliminating redundant pixels.

The pattern

```
* *
* *
  *
```

is searched for,

resulting in:

```
* *
  *
  *
```

starting from the top leftmost pixel, moving in a left to right direction (row by row). The pattern may appear rotated 90°, 180°, or 270°. For example:

```
* *                                 * *
* *                                   *
* *                                   *
* *            will result in         *
* * * * * *                           *
* * * * * *                         * * * * * *
```

The algorithm is not perfect (a couple of stray pixels remain "unthinned"), yet the affect on recognition accuracy is negligible. Speed is the major advantage, due to its simplicity, as opposed to more complicated, and hence slower algorithms, which would not affect recognition accuracy.

Tests were executed utilizing all three window sizes. Recognition accuracy was virtually unchanged for lateral translation, yet was degraded for rotation, random noise, and cover testing with respect to a similar set of tests applied to characters which were not thinned, and where system training was performed in the absence of thinning. This was especially prevalent in regard to random noise and cover testing. The slip in accuracy can be attributed to the fact that both the random noise and cover tests have the effect of reducing the number of ON pixels in the bitmap (toggling them OFF). Given this loss of information, coupled with the fact that a thinned character initially has a reduced number of redundant ON pixels (as compared to the regular font), it is clear that a set of "lost" pixels (turned OFF) would result in more recognition failures for thinned characters.

4.6 Dropped Features: Determining Which Features are the Most Powerful

Table 8 contains the results of the tests designed to evaluate system performance in regard to the omission of selected shape features (system parameters are set to nominal values - e.g., window size = 6X6). "Dropping short (xxS)" refers to omitting features ending with the "S" mnemonic. "Dropping Length" translates to ignoring the concept of feature size altogether. In other words, xxS OR xxL = xxL. Similar tests were performed for the 7X7 and 5X5 window sizes.

Overall, dropping corner features tended to slightly increase recognition accuracy under nominal conditions. After a detailed examination of the results, it was concluded that recognition accuracy did increase for random noise testing (except for the 7X7 window size), yet for rotation testing, recognition accuracy decreased (for all window sizes). The effect of dropping corner features for translation testing depended upon the window size as follows:

7X7 => decreased accuracy

6X6 => accuracy remained the same

5X5 => increased accuracy

Corner features play a much more important role for the larger window size.

The average results were fairly indicative of each individual test for the other "feature drop" cases. In other words, rotation, translation, and random noise tests all resulted in decreased recognition accuracy as these features were dropped.

Examination of the frequency of occurrence for the features in the given character set might serve to explain the results in TABLE 8. This information is contained in Table 9 (expressed as a percentage of the total number of features extracted in the character set), and tends to correlate with TABLE 8. It suggests that features which occur more frequently, have more of an adverse affect on recognition accuracy when they are omitted. For example, since more vertical than horizontal lines were extracted, recognition accuracy

was affected more when vertical lines were dropped. Omitting "short" features had a drastic affect upon recognition accuracy. Table 9 also helps to explain why corner features play a more important role for the 7X7 window size.

4.7 Training Sample Size

Tests were performed using TRAIN1 learning with 1, 2, 4, and 10 samples per character class. Recognition accuracy, and the confidence factor was invariant to sample size. Similar tests were executed using TRAIN2 learning, with the same results.

At this point it was decided to experiment with the system's ability to recognize characters without the supervised learning algorithm. This was in effect, a direct test of the "evolutionary" part of the model, which assigns weight W(i,j) a value of +1 if F(i) is present in class j, or -1 otherwise. This phase occurs during system initialization. The problem of recognition has now been reduced to extracting shape features from a sample test character, and applying this feature vector to a Hamming Net [1] with weights assigned as described above. The chosen character class is the one with the minimum Hamming distance, which of course results in the maximum score. Recognition accuracy was as good as the results achieved when the learning algorithm was employed, but the confidence factor was, on the average, 1/3 of what was achieved with the learning algorithm.

4.8 General Conclusions

Based on the above results, several conclusions can be discerned concerning our character recognition system. The driving force behind character recognition is the feature extraction mechanism, coupled with the assignment of initial weights as described above. The learning algorithm was instrumental in increasing the level of confidence (by a factor of 3) in a decision. In other words, the distance was increased between the first and second choice score. This is extremely significant for systems which require this distance to exceed a certain threshold before a decision can be made. If the threshold is not exceeded, then the system will render the sample character unrecognizable. Such systems exist, when the cost of an incorrect decision is high relative to no decision.

In an attempt to provide additional evidence to support the above conclusions, the system was exercised on a larger collection of characters obtained from the Shanghai University of Technology database. Every fourth character was selected, and written to a separate file. This resulted in a total of 725 characters (80% of these were Chinese characters). The major objective was to determine system behavior with respect to a larger group of characters. The test results (utilizing a 7X7 window size) are contained in Tables 10 - 12. The outcome was similar to the results obtained with the original 100 character set. As before, the driving force behind

character recognition was feature extraction, coupled with the assignment of initial weights. The learning algorithm, again caused the level of confidence to significantly increase. TRAIN2 learning slightly improved recognition accuracy for rotated characters (TEST3). Recognition accuracy for rotated characters was reduced in comparison to the original character set. This can be attributed to the fact that the 725 character set was composed of 80% Chinese characters, which undergo more distortion during rotation.

A final analysis was performed on the original 100 character set, aimed at adjusting the learning algorithm, and comparing the adjusted algorithm's results with the outcome of the original learning algorithm. The first adjustment consisted of applying the following weight modification:

Modification A

$W(i,j) = W(i,j) + A*F(i)*C(j)$ if $C(j) = +1$ or

$W(i,j) = W(i,j) + B*F(i)*C(j)$ if $C(j) = -1$

in all cases, regardless of the original W(i,j) value.

Cases 0 and 5 in section 3.4 would result in the weight being modified as described above (recall that in the original learning algorithm these cases did not modify W(i,j)). Modifying W(i,j) for case 5, where F(i)=+1, C(j)=-1, and W(i,j)>0, results in common features being inhibited. Since the product F(i)*C(j) < 0, the value of W(i,j) is reduced, even though F(i) tends to exist in C(j). The

consequences of modifying case 0 are even more severe. For the average character (e.g., not Chinese), there is a greater number of features that do not exist, as opposed to features that do exist (e.g., a lot of white space). Therefore, for case 0, where F(i)=-1, C(j)=-1, and W(i,j)<=0, the feature does not exist in the sample character, or in the class indicated by C(j). This case occurs quite frequently. Since the product F(i)*C(j) > 0, W(i,j) is increased. This results in large positive weights for features which commonly do not exist (e.g., F(i)=-1), thus the classification score is driven to a large negative number for all classes. Recognition accuracy was moderately degraded, and the confidence factor was drastically reduced (less than 1% as compared to an average of 23% for the original learning algorithm).

The second attempt at modifying the original learning algorithm consisted of the following:

Modification B

W(i,j) = W(i,j) + F(i), for C(j) = +1

W(i,j) = W(i,j), for C(j) = -1

Only the weights from the "known" class are modified, per sample, during the learning phase. In this case the recognition accuracy was slightly worse, compared to the original learning algorithm, and the average confidence factor decreased to 6.02%. This is significantly

less than the average confidence (23%) obtained from the original learning algorithm.

In summary, neither attempt at adjusting the original learning algorithm was successful. Both accuracy and confidence were decreased. The following information summarizes the average results obtained from each learning modification (system parameters were set to their nominal values):

Learning Algorithm	Avg Rec Accuracy	Conf Factor
original	80.29%	23%
mod A	75.14%	0.22%
mod B	73.29%	6.02%

5. RECOMMENDATIONS FOR FUTURE RESEARCH

This study was limited to line-like features. For a 16X16 bitmap, nonlinear features are a concatenation of linear features. For example, an arc can be approximated by a series of horizontal, vertical, and diagonal lines. By employing more sophisticated feature extraction algorithms, possibly invariant to window dimensions, and capable of attaining non-linear features (e.g., radius of curvature, arc length, etc.), similar methods could be employed for characters of many different sizes and fonts. In addition to shape features, one might

incorporate features which are invariant to rotation, and improve the system's performance. These features might be as simple as the sum total of ON pixels in a particular window, or as complex as a central moment calculation [8].

The effectiveness of a set of features in recognizing characters may depend on the characters themselves. One way to select additional features is to investigate characters which are often classified incorrectly. Table 13 illustrates a "confusion matrix" for the characters A through Z, which depicts the normalized classification scores for each character. Classification scores were normalized by dividing the actual score by the sum of all actual scores in the row, and then multiplying by 100. Values were rounded to the nearest integer. Training consisted of 1 sample per character. Each row represents the normalized scores from the tested character. This table can help identify characters which are easily confused, thus providing clues to additional features which may aid in the discrimination process.

Generally, more sophistication also results in an algorithm which takes longer to execute, or consumes more resources. A possible solution is to utilize our neural net system hierarchically. The first test would reduce the search space of candidate characters, and subsequent tests would depend upon the subset. As a result, we are taking advantage of the speed and simplicity of our neural net system, and employing a more costly algorithm on the smaller subset.

6. REFERENCES

[1] R. P. Lippmann, "An Introduction to Computing with Neural Nets", IEEE ASSP, April 1987

[2] G. Krishnan and D. Walters, "Psychologically plausible features for shape recognition in a neural-network", Proc. IEEE Conf. on Neural Networks 1988, p. II-127 - II-134

[3] J. A. Anderson and M. Mozer, "Categorization and Selective Neurons", Parallel Models of Associative Memory, Hillsdale, 1981, p. 213-236

[4] D. Walters, "Selection of image primitives for general-purpose visual processing", Computer Vision Graphics and Image Processing, Vol. 37 No. 2, 1987, p. 261-298

[5] D. E. Rumelhart and J. L. McClelland, Parallel Distributed Processing, Vol. 1, MIT Press, Cambridge, Ma., 1986

[6] L. D. Jackel, H. P. Graf, W. Hubbard, J. S. Denker, and D. Henderson, "An Application of Neural Net Chips: Handwritten Digit Recognition", Proc. IEEE Conf. on Neural Networks 1988, p. II-107 - II-115

[7] K. S. Fu, Applications of Pattern Recognition, CRC Press, 1982, p. 1-40 and 197-235

[8] M. James, Pattern Recognition, J. Wiley and Sons, 1988, p. 1-30 and 102-129

[9] R. Duda and P. Hart, Pattern Classification and Scene Analysis, J. Wiley and Sons, 1973, p. 10 - 34 and 130 - 156

APPENDIX A - FIGURES

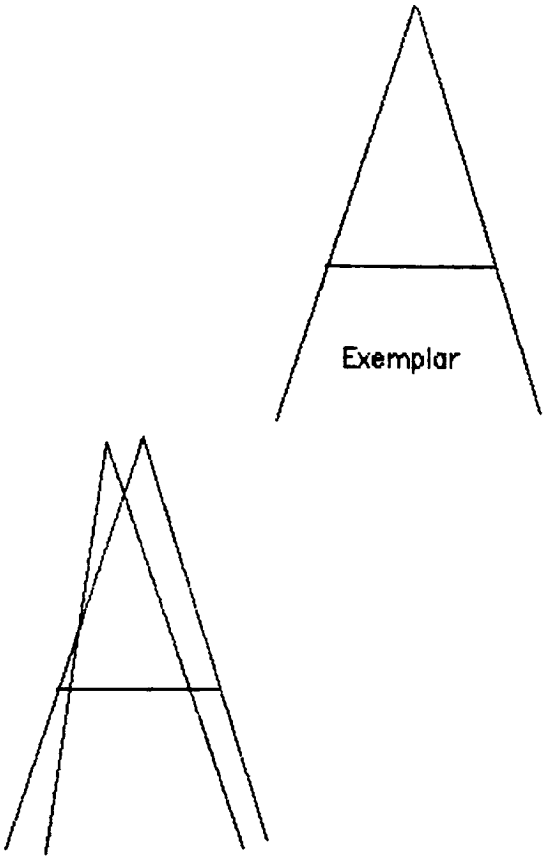

Figure 1a.

Two different "A" fonts superimposed

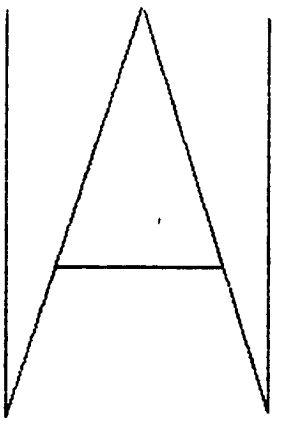

Figure 1b.

"W" superimposed on "A"

Note that the Hamming distance between superimposed characters is greater for Figure (a.)

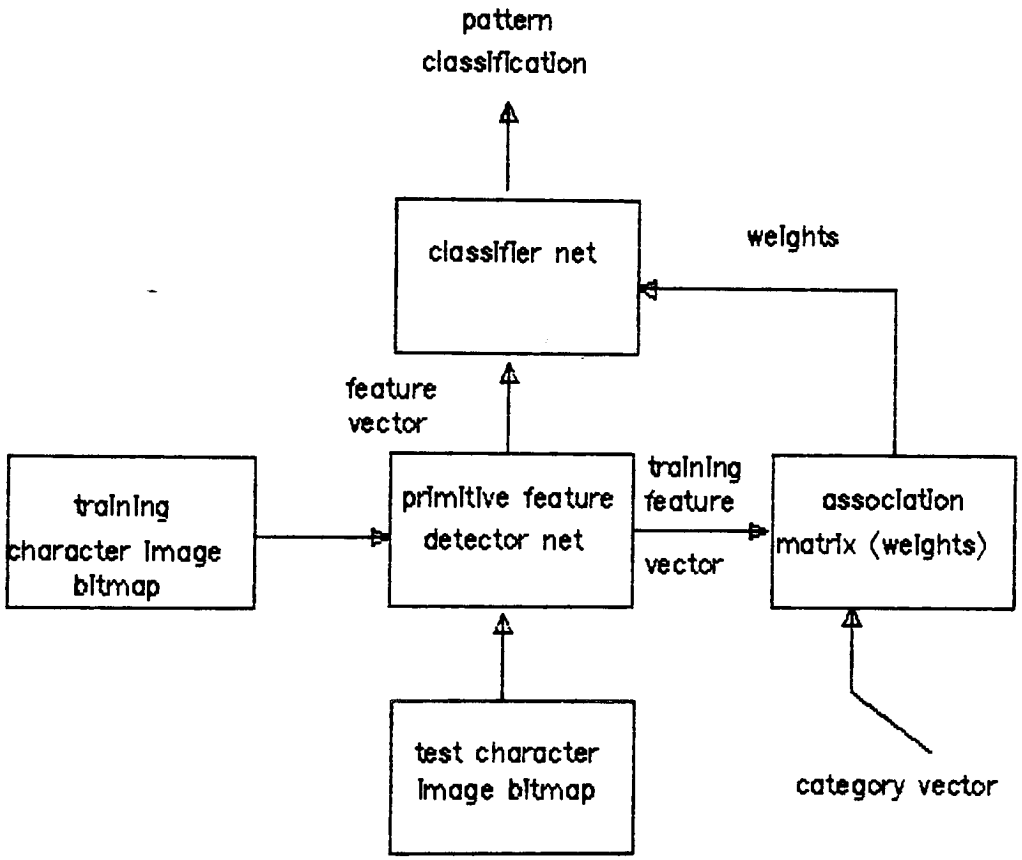

Figure 2. System Architecture

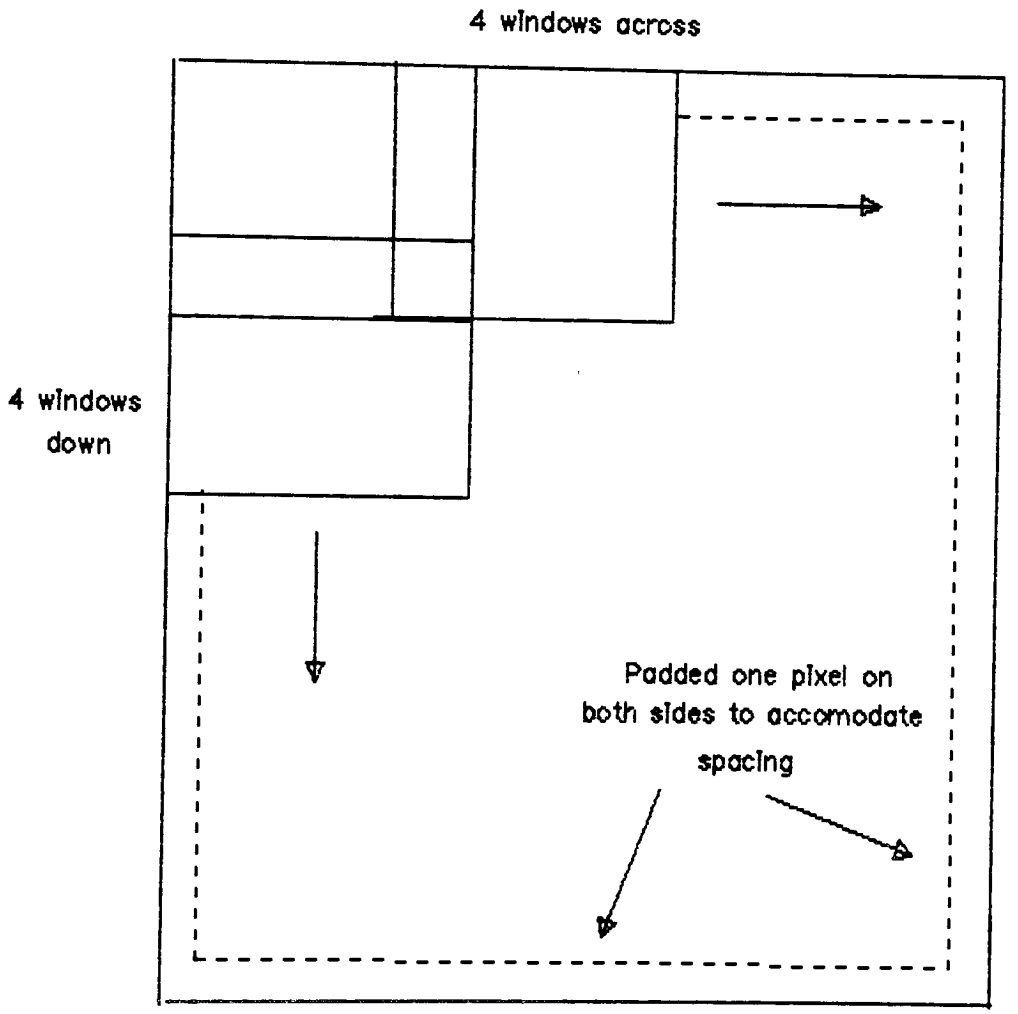

Figure 3 – 6X6 Window Geometry

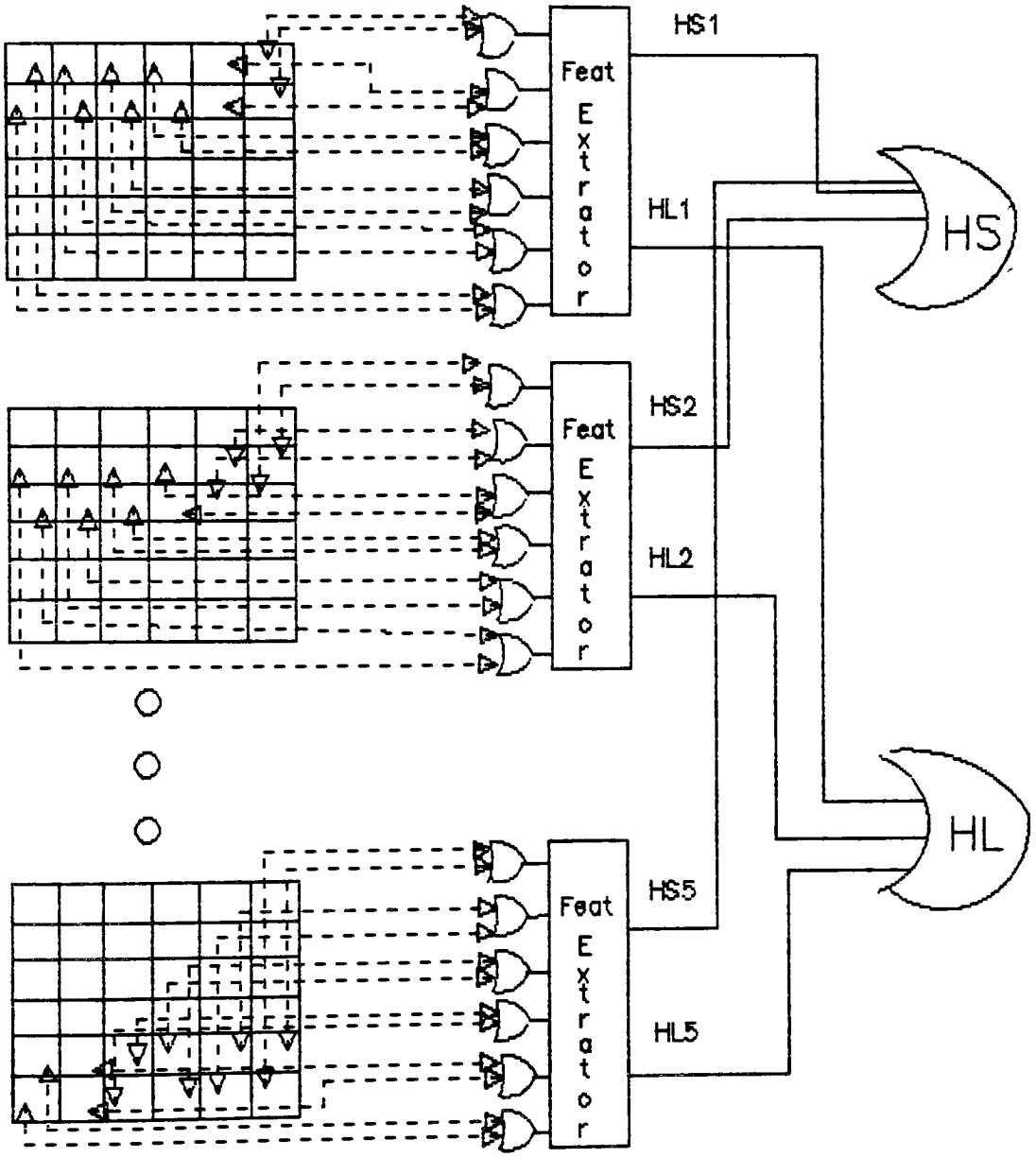

Figure 4 – Horizontal Line Feature Extraction (6X6 Window)

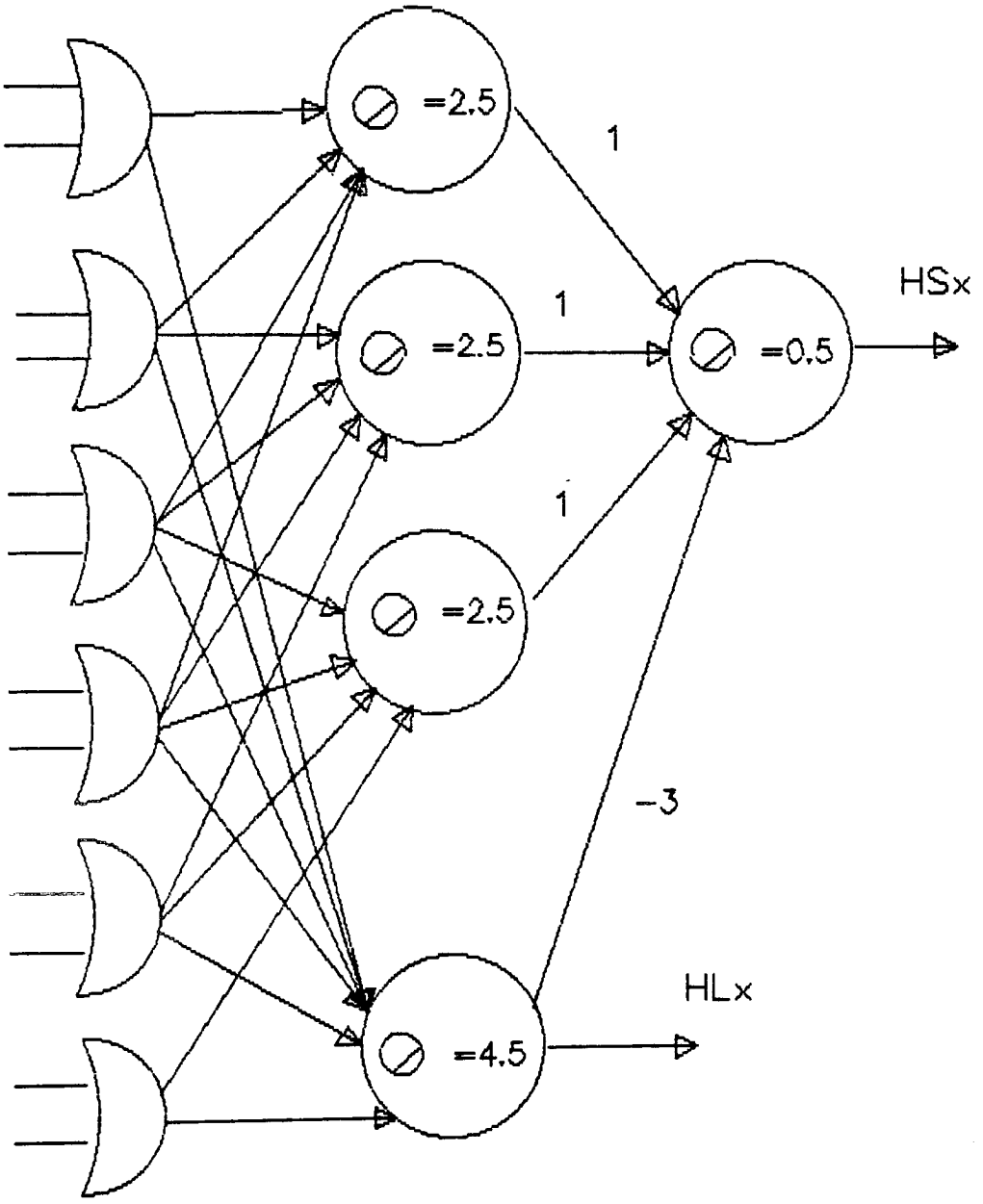

Figure 5 – Horizontal Feature Extractor Subnet (6X6)

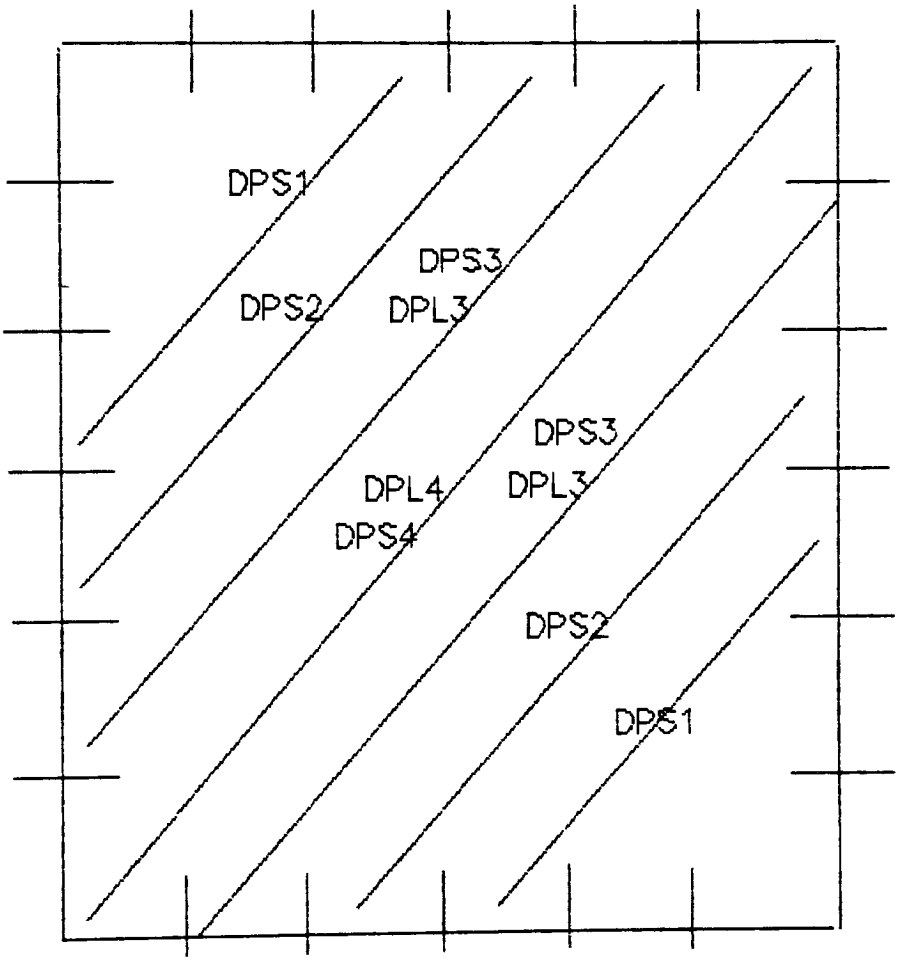

Figure 6 – Positive Diagonal Sub–Features (6X6)

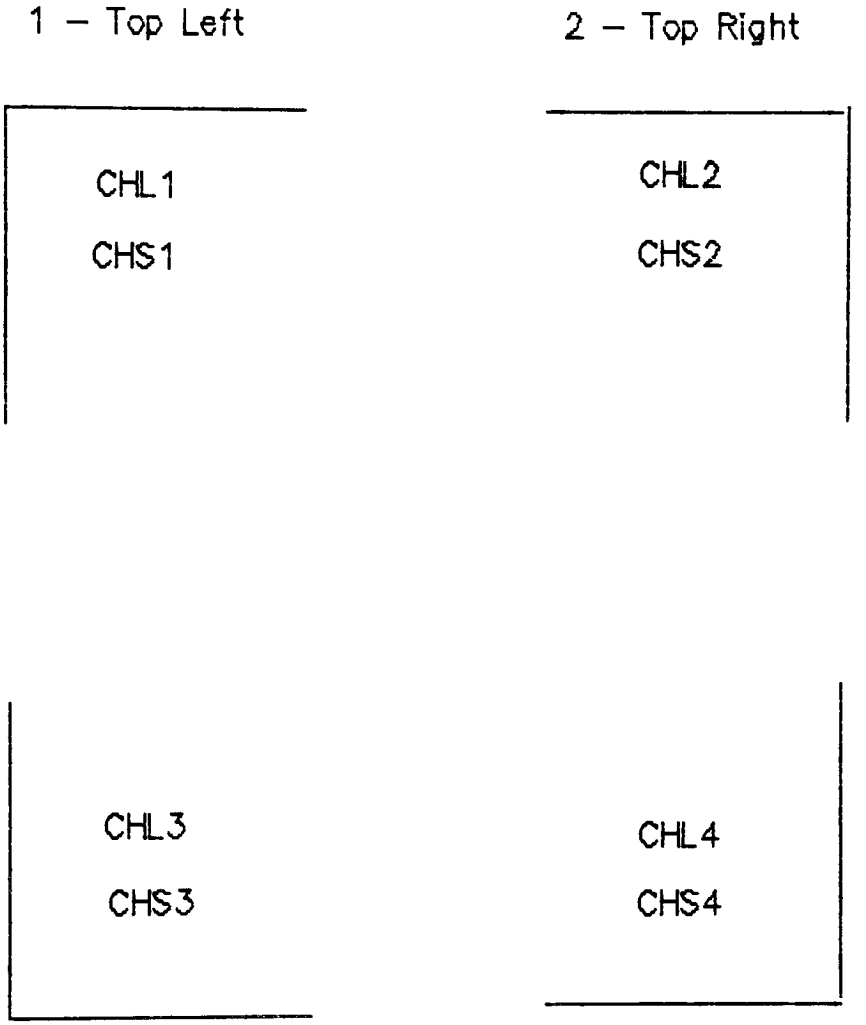

Figure 7 – Four Orientations of CHSx / CHLx

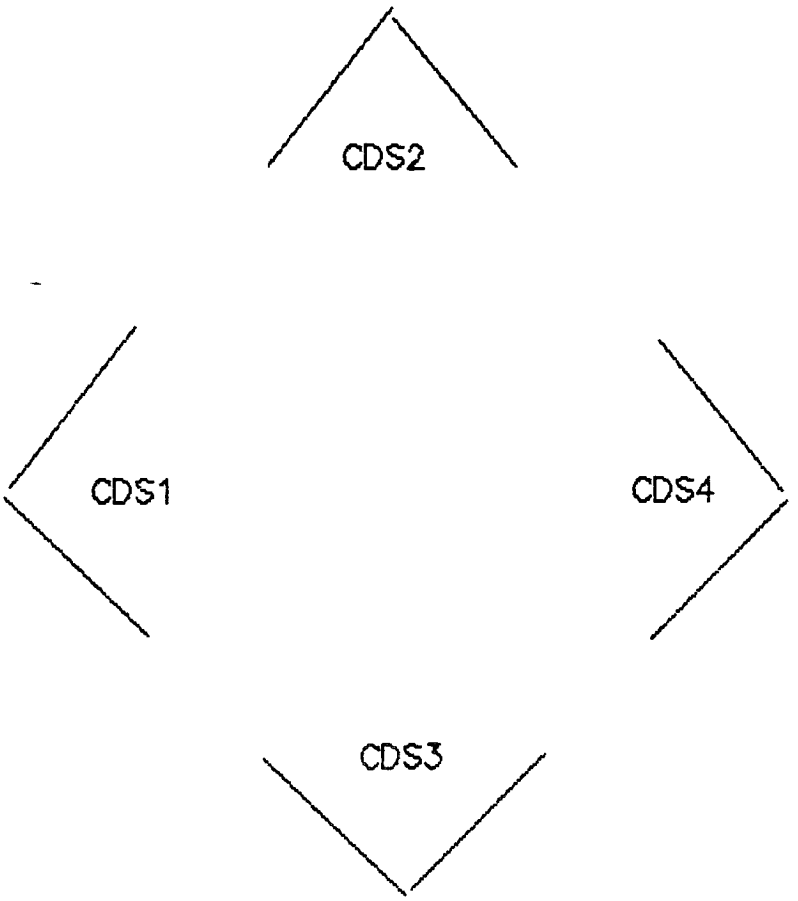

Figure 8 – Four Orientations of CDSx

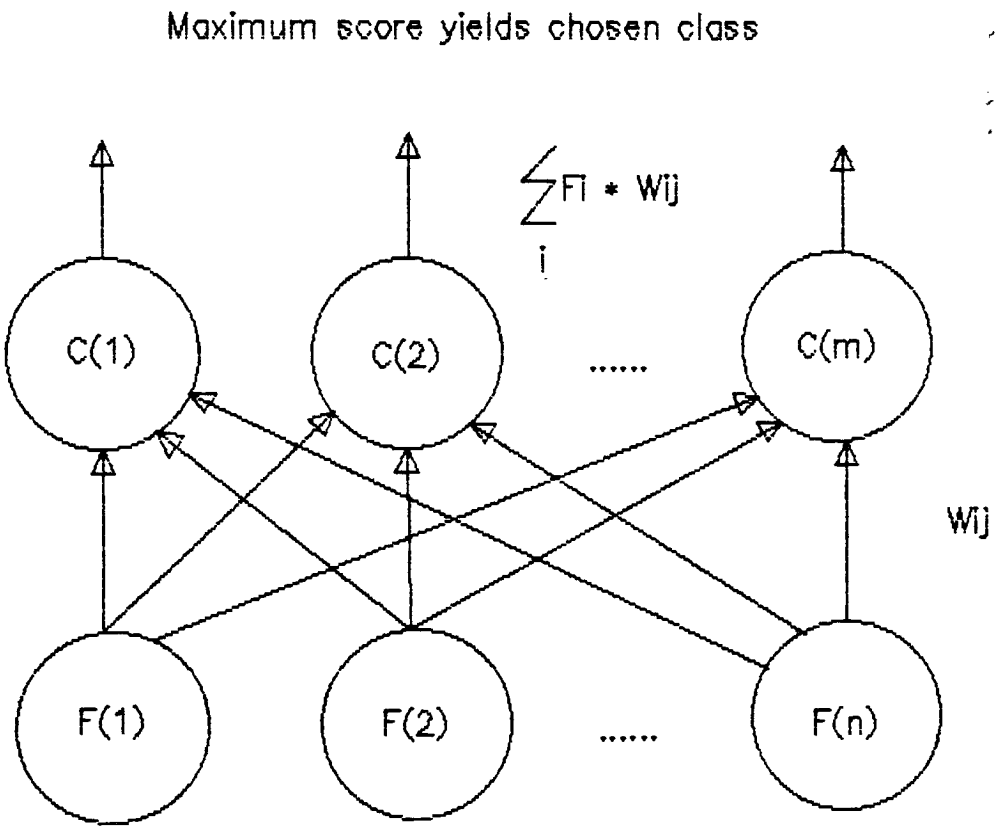

Figure 9. The Classifier Net

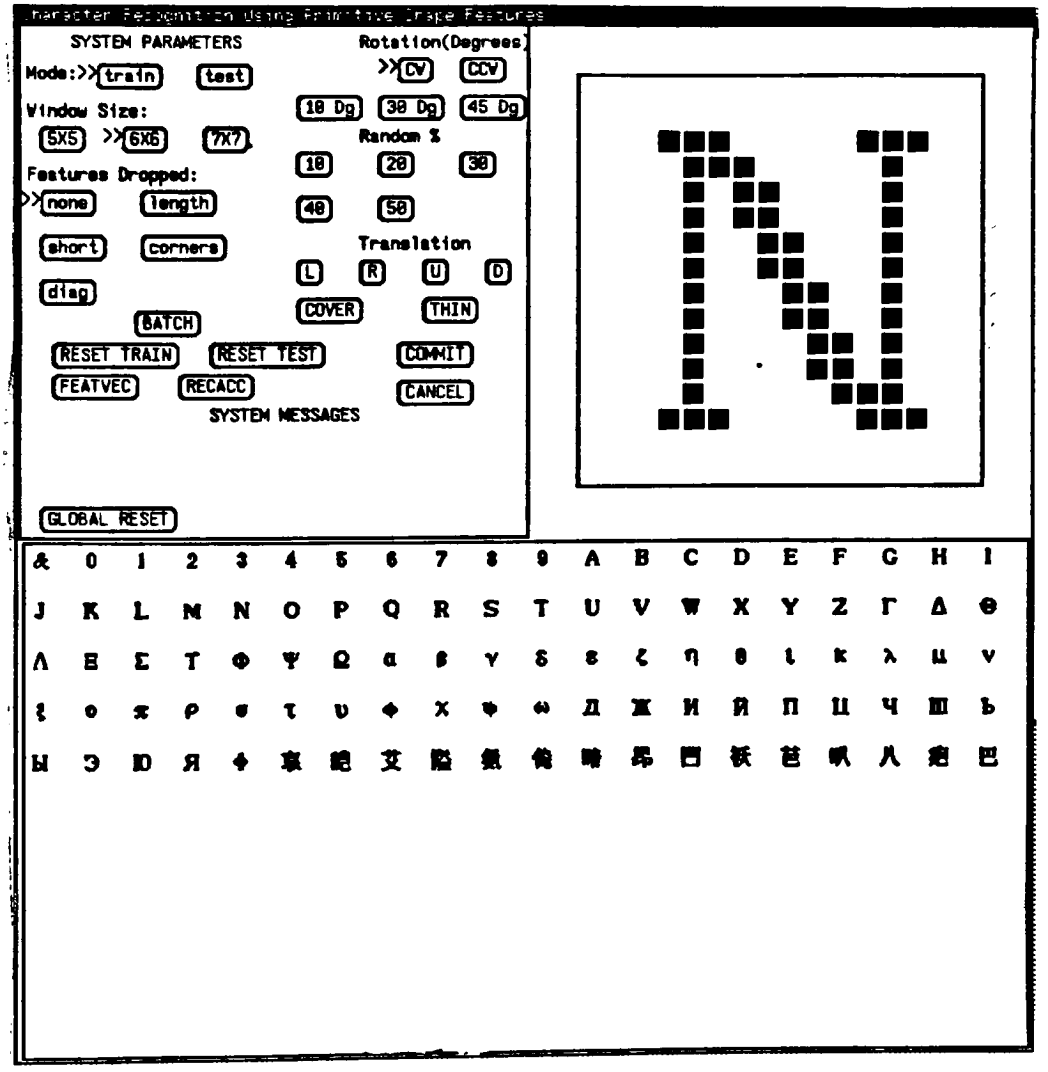

FIGURE 10 - USER INTERFACE

APPENDIX B - TABLES

TRAIN1

RECOGNITION ACCURACY (%)

TEST TYPE	TOP 1	TOP 2	TOP 3	TOP 4	CONFIDENCE
TEST1	100	100	100	100	41.20
TEST2	87	94	97	97	23.09
TEST3	85	94	97	97	20.17
TEST4	11	19	26	30	10.37
TEST5	96	99	100	100	31.32
TEST6	78	87	89	92	19.88
TEST7	38	44	48	52	9.59
TEST8	68	78	83	86	19.49
AVERAGE	78.85	85.14	87.71	89.14	23.53

TABLE 1.

Tables 1 through 4 show the results of four training methods. See sections 3.6.2 and 3.6.3 for a description of TRAIN1 .. TRAIN4 and TEST1 .. TEST8.

TRAIN2

RECOGNITION ACCURACY (%)

TEST TYPE	TOP 1	TOP 2	TOP 3	TOP 4	CONFIDENCE
TEST1	100	100	100	100	39.60
TEST2	87	95	97	98	21.97
TEST3	88	95	97	97	19.75
TEST4	11	19	27	29	10.02
TEST5	99	100	100	100	30.86
TEST6	79	86	89	91	19.26
TEST7	43	54	61	63	8.91
TEST8	66	76	81	86	18.17
AVERAGE	80.28	86.57	89.28	90.71	22.65

TABLE 2.

TRAIN3

RECOGNITION ACCURACY (%)

TEST TYPE	TOP 1	TOP 2	TOP 3	TOP 4	CONFIDENCE
TEST1	100	100	100	100	38.83
TEST2	87	94	97	97	21.60
TEST3	87	95	97	98	18.77
TEST4	11	21	29	32	10.22
TEST5	93	97	98	99	29.93
TEST6	75	86	90	91	19.34
TEST7	41	47	54	58	7.72
TEST8	68	76	82	85	18.05
AVERAGE	78.71	85.00	88.28	89.71	22.03

TABLE 3.

TRAIN4

RECOGNITION ACCURACY (%)

TEST TYPE	TOP 1	TOP 2	TOP 3	TOP 4	CONFIDENCE
TEST1	99	99	99	100	36.12
TEST2	85	93	96	97	19.59
TEST3	85	94	96	96	18.23
TEST4	12	20	30	32	9.95
TEST5	97	97	98	99	27.64
TEST6	71	81	85	88	14.82
TEST7	35	44	51	54	7.36
TEST8	68	76	81	86	15.91
AVERAGE	77.14	83.12	86.57	88.57	19.95

TABLE 4.

Window Size = 5X5

RECOGNITION ACCURACY (%)

TEST TYPE	TOP 1	TOP 2	TOP 3	TOP 4	CONFIDENCE
TEST1	100	100	100	100	41.70
TEST2	80	89	94	97	24.25
TEST3	89	95	97	97	24.84
TEST4	13	19	24	27	7.82
TEST5	94	97	98	98	32.36
TEST6	72	80	85	89	20.26
TEST7	34	44	48	52	8.48
TEST8	81	88	91	94	21.79
AVERAGE	78.57	84.71	87.57	89.57	24.81

TABLE 5.

Tables 5 through 7 show the results of each window size with system parameters set to nominal values.

Window Size = 6X6

RECOGNITION ACCURACY (%)

TEST TYPE	TOP 1	TOP 2	TOP 3	TOP 4	CONFIDENCE
TEST1	100	100	100	100	39.60
TEST2	87	95	97	98	21.97
TEST3	88	95	97	97	19.75
TEST4	11	19	27	29	10.02
TEST5	99	100	100	100	30.86
TEST6	79	86	89	91	19.26
TEST7	43	54	61	63	8.91
TEST8	66	76	81	86	18.17
AVERAGE	80.28	86.57	89.28	90.71	22.65

TABLE 6.

Window Size = 7X7

RECOGNITION ACCURACY (%)

TEST TYPE	TOP 1	TOP 2	TOP 3	TOP 4	CONFIDENCE
TEST1	100	100	100	100	37.66
TEST2	90	96	98	99	23.44
TEST3	84	93	97	97	19.36
TEST4	14	21	30	35	9.51
TEST5	95	100	100	100	30.15
TEST6	84	90	90	93	20.71
TEST7	44	52	57	58	10.73
TEST8	73	83	89	90	18.44
AVERAGE	81.42	87.71	90.14	91.00	22.93

TABLE 7.

	AVERAGE RECOGNITION ACCURACY (%)			
DROPPED FEATURES	TOP 1	TOP 2	TOP 3	TOP 4
corners	80.57	88.29	91.86	93.57
none	80.28	86.57	89.28	90.71
diagonal lines	75.43	82.86	85.86	87.57
horizontal lines	73.30	80.57	83.71	85.57
vertical lines	73.00	79.86	82.14	84.00
horiz & vertical	65.00	71.71	74.71	77.29
length	57.00	65.43	70.14	74.29
short	53.43	63.00	67.71	72.29

TABLE 8.

This table depicts the results obtained when selected features were dropped (see section 4.6).

	WINDOW SIZE		
FEATURES	5X5	6X6	7X7
short (xxS)	74.9	74.8	73.6
VS,VL	29.7	29.0	23.7
HS,HL	24.3	21.8	19.1
DPS,DPL	10.6	11.5	11.7
DNS,DNL	10.9	11.1	12.3
CHS1,CHL1	5.3	5.4	6.2
CHS2,CHL2	5.5	5.5	6.1
CHS3,CHL3	5.5	5.5	6.5
CHS4,CHL4	5.0	5.3	5.5
CDS1	0.9	1.6	2.2
CDS2	0.9	0.8	2.2
CDS3	0.6	1.0	2.2
CDS4	1.0	1.5	2.3

TABLE 9.

This table displays the frequency of feature occurrence expressed as a percentage of the total number of features extracted in the character set (see section 4.6).

NO TRAINING

RECOGNITION ACCURACY (%)

TEST TYPE	TOP 1	TOP 2	TOP 3	TOP 4	CONFIDENCE
TEST1	99	100	100	100	33.07
TEST2	84	91	93	94	14.11
TEST3	39	49	53	56	5.96
TEST5	97	98	99	99	24.19
TEST6	77	83	85	87	11.92
AVERAGE	79.20	84.20	86.00	87.20	17.85

TABLE 10.

Tables 10 through 12 depict the test results (utilizing a 7X7 window size) which determine system behavior with respect to a larger group of characters (see section 4.8). "No training" implies that weights were not modified after initialization.

TRAIN1 - 1 sample per char

RECOGNITION ACCURACY (%)

TEST TYPE	TOP 1	TOP 2	TOP 3	TOP 4	CONFIDENCE
TEST1	99	100	100	100	43.97
TEST2	84	91	93	94	21.13
TEST3	39	49	53	56	8.92
TEST5	97	99	99	99	35.39
TEST6	80	83	86	87	19.11
AVERAGE	79.80	84.40	86.20	87.20	25.70

TABLE 11.

TRAIN2 - 10 samples per char

RECOGNITION ACCURACY (%)

TEST TYPE	TOP 1	TOP 2	TOP 3	TOP 4	CONFIDENCE
TEST1	99	100	100	100	43.77
TEST2	85	91	94	95	21.00
TEST3	41	50	55	58	8.99
TEST5	98	99	99	99	34.37
TEST6	72	79	82	84	18.05
AVERAGE	79.00	83.80	86.00	87.20	25.24

TABLE 12.

CONFUSION MATRIX

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	19	0	6	2	0	0	6	3	5	6	2	1	3	7	6	0	4	2	4	4	5	4	2	5	0	4
B	1	16	4	8	10	8	3	6	0	1	4	2	-3	2	4	9	0	10	7	0	4	1	3	0	0	2
C	3	0	14	5	4	5	11	0	5	5	-1	6	-1	2	10	4	4	1	4	4	7	6	0	0	2	2
D	1	6	6	14	6	5	5	2	4	4	0	6	-1	2	10	6	4	5	4	4	7	2	0	-2	1	1
E	0	7	5	7	14	11	4	5	2	3	4	6	-1	1	2	8	0	6	5	3	5	3	0	-1	0	4
F	-1	5	5	5	11	14	3	5	4	3	3	7	-1	2	2	10	-1	6	5	4	5	4	-1	0	1	3
G	4	0	13	6	4	3	16	0	3	3	0	4	0	3	11	2	6	0	4	3	6	6	0	0	1	3
H	3	5	2	3	6	7	1	15	5	4	3	6	0	3	2	7	1	7	5	4	7	2	0	1	0	1
I	3	-2	5	3	1	4	3	3	14	10	0	8	-1	3	4	3	2	1	2	13	8	5	-2	3	5	4
J	4	-1	6	4	2	4	3	3	10	15	-1	7	-1	2	6	2	2	0	4	10	7	4	0	0	2	6
K	4	4	3	3	7	6	3	5	4	2	18	3	1	6	2	6	0	8	1	2	3	2	0	5	1	2
L	0	0	6	5	5	7	4	4	8	6	0	14	0	2	4	6	0	1	3	8	9	6	-4	0	3	3
M	6	-4	4	3	0	2	3	1	4	3	2	4	24	13	6	0	4	2	0	3	6	7	0	6	3	1
N	6	0	4	3	1	3	3	2	5	3	3	3	7	17	4	2	4	3	2	4	7	6	2	7	4	0
O	3	1	11	10	1	2	10	0	4	5	-2	3	0	3	15	3	8	1	5	4	8	4	0	-1	1	1
P	0	7	5	6	8	11	3	7	4	3	3	6	-2	2	4	15	0	10	4	4	4	2	-1	-2	0	1
Q	4	0	8	7	0	0	9	1	5	4	-2	2	1	5	12	0	19	0	5	4	9	5	0	0	1	2
R	2	8	3	6	7	8	2	8	3	2	6	3	0	3	3	11	0	15	4	3	4	1	0	1	0	0
S	3	6	6	5	6	6	5	5	3	5	-1	4	-2	2	7	4	4	3	16	4	6	4	-1	0	0	3
T	2	-2	5	3	2	4	2	2	13	9	0	8	-1	3	4	3	2	1	2	15	7	4	0	2	4	5
U	1	1	6	6	3	3	4	4	7	5	-1	7	0	4	7	2	5	1	4	6	13	6	0	1	4	1
V	2	-2	7	2	3	4	5	0	6	4	-1	6	2	5	5	1	3	-1	3	5	8	16	4	3	8	3
W	5	4	5	3	2	2	4	1	2	3	0	0	0	5	4	1	1	1	2	3	6	8	21	5	6	5
X	5	0	3	0	0	2	3	2	6	3	4	2	4	10	2	0	1	2	1	5	5	6	3	18	8	6
Y	1	-1	6	3	2	3	3	0	7	4	0	5	1	5	4	1	1	0	1	7	7	10	4	8	17	3
Z	4	1	5	3	5	4	5	1	6	8	0	4	-1	1	3	2	2	0	4	7	4	4	2	4	3	17

TABLE 13.

Table 13 illustrates a "confusion matrix" for the characters A through Z, which depicts the normalized classification scores for each character. Classification scores were normalized by dividing the actual score by the sum of all actual scores in the row, and then multiplying by 100. Each row represents the normalized scores from the tested character (see section 5).