[14] H. Xingui, "Weighted fuzzy logic and its applications," in *Proc. 12th Annu. Int. Computer Software Application Conf.*, Chicago, IL, 1988, pp. 485–489.
[15] L. A. Zadeh, "Fuzzy sets," *Inform. Contr.*, vol. 8, pp. 338–356, 1965.
[16] H. J. Zimmermann, *Fuzzy Set Theory and Its Applications.* Dordrecht, The Netherlands: Kluwer-Nijhoff, 1991.
[17] R. Zwick, E. Carlstein, and D. Budescu, "Measures of similarity among fuzzy sets: A comparative analysis," *Int. J. Approx. Reas.*, vol. 1, pp. 221–242, 1987.

Handwritten Word Recognition with Character and Inter-Character Neural Networks

Paul D. Gader, Magdi Mohamed, and Jung-Hsien Chiang

Abstract— **An off-line handwritten word recognition system is described. Images of handwritten words are matched to lexicons of candidate strings. A word image is segmented into primitives. The best match between sequences of unions of primitives and a lexicon string is found using dynamic programming. Neural networks assign match scores between characters and segments. Two particularly unique features are that neural networks assign confidence that pairs of segments are compatible with character confidence assignments and that this confidence is integrated into the dynamic programming. Experimental results are provided on data from the U.S. Postal Service.**

I. Introduction

An off-line, handwritten word recognition algorithm has two inputs: a digital image (assumed to be an image of a word), and a list of strings called a lexicon, representing possible identities for the word image. The goal is to assign a match score to each candidate in the lexicon.

A variety of approaches have been reported since 1990. Several researchers [1]–[7] have used hidden Markov models. Others have tried to use "wholistic approaches" in which a word is recognized as an entity. These algorithms do well at providing auxiliary information, but not as stand-alone recognizers [8]–[13]. Some of the most successful results have come from segmentation-based techniques that rely on dynamic programming [5], [14]–[20]. In these approaches, an optimal segmentation is generated for each lexicon string.

Our baseline system is based on dynamic programming and is illustrated in Fig. 1. A word image is segmented into subimages called *primitives* without using a lexicon. Each primitive ideally consists of a single character or a subimage of a single character. A segment is defined as either a primitive or a union of primitives and a segmentation as a sequence of segments using all the primitives. Dynamic programming is used to find the segmentation that matches a given string best. The match score is assigned by matching each segment in the segmentation to the corresponding character in the string using a character recognizer that returns confidence values.

This approach does not consider important inter-character relationships. For example, in Fig. 2, a segmentation of the word

Manuscript received December 22, 1993; revised August 12, 1995. This work was supported by the U.S. Postal Service through the Environmental Research Institute of Michigan.

The authors are with the Department of Electrical and Computer Engineering, University of Missouri, Columbia, MO 65211 USA (email: gader@sunpg.ece.missouri.edu).

Publisher Item Identifier S 1083-4419(97)00798-X.

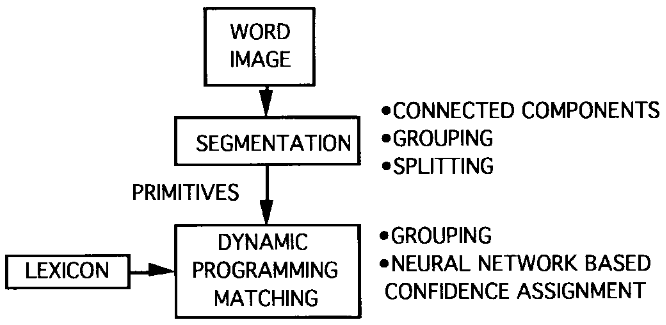

Fig. 1. Overview of the word recognition system.

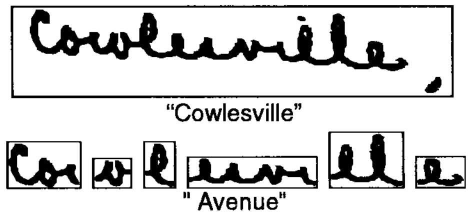

Fig. 2. The character recognition scores of the individual fifth and sixth segments match well against the characters "u" and "e," but the sizes of the segments are not spatially compatible.

"Cowlesville" matches well to the string "avenue". The fifth and sixth segments together do not look much like "ue" since the fifth segment is much larger than the sixth segment and "u" and "e" are about the same size. However, as individual characters, the fifth segment looks very much like "u" and the sixth segment very much like "e". Of course, the "ue" hypothesis is possible and should be assigned some nonzero confidence.

The spatial relationships and relative sizes between segments are cues that should be considered in assigning a match score between a word image and a lexicon string. One method for doing so is to use a post processor that modifies the match score after dynamic programming. This approach cannot correct for segmentation errors caused by bad matches.

The novel approach described here builds the confidence modification due to spatial relationships into the dynamic programming. A compatibility score is assigned to pairs of adjacent segments using a neural network. This compatibility score is combined with the character recognition score to assign match scores between segments and characters. A related concept was used by Obaidat and Macchairolo who used time intervals between typed characters to identify computer users [21]. We now describe the system and then provide experimental results for the character recognition and compatibility modules and the entire system.

II. Segmentation

The segmentation module is very similar to that described in [22] and we therefore do not discuss it much here. The segmentation process initially detects connected components. Some simple grouping and noise removal is performed. The results are referred to as the initial segments. An element of an initial segmentation is generally a significant connected component in the word, or a grouping of connected components. Those components which are not "bars" (such as the top of a "T" or the vertical bar in a "D") are sent to a splitting

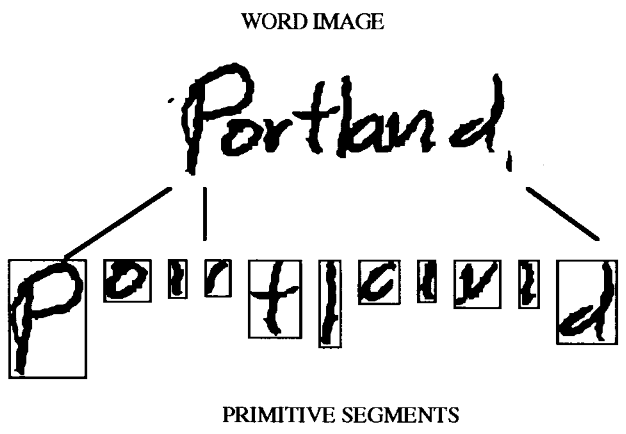

Fig. 3. A word image and its primitive segments.

process which is designed to split connected components consisting of multiple characters into primitive segments. The need for aggressive splitting due to ambiguity of characters is well documented elsewhere and will not be discussed here [22]–[26]. The results of splitting are then used to form the primitive segments. A word image and the resulting primitive segments are shown in Fig. 3.

The segmentation module described in [22] differs from the segmentation module described here in four ways. The first is the output format. In the current approach, dynamic programming is used to generate optimal segmentations "on the fly" but in the referenced work, multiple segmentations are formed in the segmentation process and they remain static during the matching process. The second major difference is that many of the rules for grouping and splitting have been made less restrictive in order to handle unconstrained words rather than just handprinted words. The earlier segmentation algorithm included a strict recognition module: an initial segment with a high recognition score was not sent to the splitting process. This module was removed because multiple characters, such as "tt" could receive high scores as single characters such as "H". Finally, in the earlier version, the initial segmentation was matched against the lexicon before splitting. If a high match score was attained, then no further processing was performed. This module has also been removed.

III. Character Confidence Assignment Neural Networks

Character confidence assignment refers to the following process: given an image s, and a character class c, assign a value to s that indicates the degree to which s represents c. This differs from the character recognition model which is: given an image s, and a set of character classes $\{c_1, c_2, \cdots, c_n\}$, determine which class s belongs to. This process is required in a lexicon driven system such as ours. We use a character recognition module that returns confidence values for character confidence assignment. There is a difference in the philosophy for training. The objective of character recognition is to achieve high classification rates. The objective of a character confidence assignment module is to accurately reflect possible class memberships.

The latter objective is difficult to model. We use the notion of fuzzy set membership to train multilayer, feedforward neural networks for character confidence assignment [20]. Specifically, we use a variation of the fuzzy k-nearest neighbor algorithm to estimate fuzzy set memberships for each sample in our training set [27]. We use these fuzzy set memberships as the target output values for each class during backpropagation training.

Our word recognition system uses four multilayer, feedforward networks for character confidence assignment trained with backpropagation. We have two types of features, the bar features and the transition features. For each feature type, there is an upper case network and a lower case network. The network architectures all consist of two hidden layers, and input and output layers. Each network has 27 outputs, one for each class and one for a noncharacter class. The noncharacter class is included to train the network to produce low responses when segments consisting of noncharacters (such as multiple characters or pieces of characters) are used as inputs. The bar feature networks each have 120 inputs, 65 units in the first hidden layer, and 39 units in the second hidden layer. The transition feature networks each have 100 inputs, 65 units in the first hidden layer, and 39 units in the second hidden layer. The number of hidden units was not optimized but has performed well in testing. In fact, the bar-feature networks trained with these architectures did very well in an international competition sponsored by NIST [28].

Both feature types operate on binary images which need not be a fixed size. The bar features have been described in detail elsewhere so we do not discuss them here [20], [29], [30]. The transition features have not been described in detail except in a technical report [30]. We describe them here. The idea is to compute the location and number of transitions from background to foreground along horizontal and vertical lines. This transition calculation is performed from right to left, left to right, top to bottom, and bottom to top. Since a constant dimension feature is required as input to the network, an encoding scheme was developed.

In the first stage of feature extraction, the transitions in each direction are calculated. Each transition is represented as a fraction of the distance across the image in the direction under consideration. These fractions are computed in decreasing order. For example, when calculating the location of transitions from left-to-right, a transition close to the left edge would have a high value and a transition far from the left edge would have a low value as illustrated in Fig. 4.

A maximum number of transitions, M, are counted on each line. If there are more than M transitions in a line, then only the first M are counted, the rest are ignored. M is set to 5. If there are less than M transitions on a line, then the "nonexistent" transitions are assigned a value of 0.

More precisely, by a line we mean a row or a column of the character image. Let h be the height of the image and w be the width of the image. We assign exactly M values to each line, say $t_1, t_2, \cdots t_N$. Assume there are n transitions on a line located at (x_i, y_i) for $i = 1, 2, \cdots, n$. The algorithm for calculating the transition locations can be represented as follows:

```
For $i = 1$ to min$(n, M)$
        If the line is a row then
                $p_i = x_i$ and $d = w$
        else
                $p_i = y_i$ and $d = h$
        end if
$t_i = 1 - p_i/d$
End for
If $n < M$ then
        For $i = n + 1$ to $M$
                $t_i = 0$
        end for
end if.
```

The transitions are resampled to a 5-point sequence for each direction and assembled into a feature vector. The five transitions for each row (column) are represented as a two-dimensional (2-D) array, $t = [t_{ij}]$ for $i = 1, \cdots, h(w)$ and $j = 1, \cdots, 5$. A 2-D array

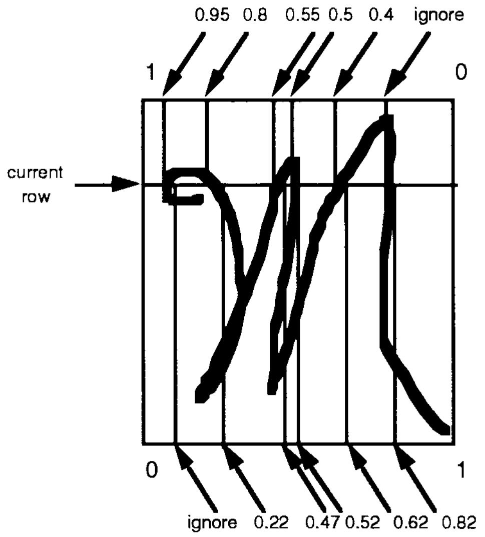

Fig. 4. The first stage of transition feature extraction shown for transitions from the left and from the right on one row of the image.

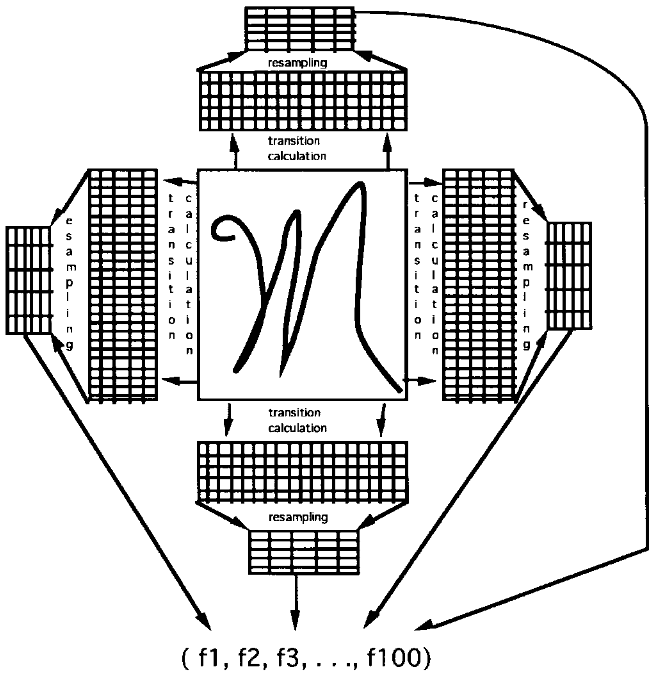

Fig. 5. The second stage of transition feature calculation consists of resampling the transition locations onto fixed size grids.

of size 5×5 from is constructed from t by local averaging on the columns of t. Since there are four directions and 5 stroke transitions along each line, this process results in a $5 \times 4 \times 5 = 100$ element feature vector. This process is illustrated in Figs. 4 and 5.

IV. Compatibility Assignment Neural Networks

In this section, we describe the approach used to assign compatibility scores to pairs of characters. The idea is to assign a compatibility measure to pairs of segments that are adjacent in a word. For example, if the character confidence assignment module is matching segment 1 to an upper case character class and segment 2 to a lowercase "g", then the compatibility measure measures the degree with which the pair of images "looks like" an ascender-type character followed by a descender-type character.

Characters are categorized into three categories: ascender (A), core (C), and descender (D). Character pairs are therefore categorized into nine classes: AA, AC, AD, CA, CC, CD, DA, DC, and DD. Samples of character pairs were collected and assigned character pair class memberships. Neural networks were trained to classify character pairs. Two different networks were trained, each with different features as inputs. We refer to these networks as compatibility networks.

The character categories are defined as follows:

$$\begin{aligned}\text{ASCENDER} &= \{\text{All upper case characters and}\\ &\qquad \text{b, d, f, h, k, l, t}\},\\ \text{CORE} &= \{\text{a, c, e, i, m, n, o,}\\ &\qquad \text{r, s, u, v, w, x, z}\},\\ \text{DESCENDER} &= \{\text{f, g, j, p, q, y}\}.\end{aligned}$$

Samples of pairs of characters from each class are shown in Fig. 6. These character pairs consist of adjacent pairs of character that were extracted from handwritten city and state words.

The character pairs were used to train two neural networks. Each neural network has nine outputs, one for each character pair type. One neural network used transition-like features. The other neural network used bounding box measurements and distances to the center-line of the word images as features. We refer to the first set of features as the transition-like features and to the latter set as the bounding box features. The bounding box features are much faster to compute and perform almost as well.

The features are computed separately on each of the individual components in a pair image. The pair image is considered within the context of the word image it was extracted from. First, the center-line of the word is estimated using the horizontal projection. The center-line of the word is used to account for differences in the positions of characters within a word depending on whether a descender is present or not. A lower-case "a" may be at the bottom of a word image if there is no character of type descender in the word, but may be in the middle of the word image if there is. The two samples of the pair "al" in the CA class shown in Fig. 6 provide good examples of this problem. The center-line is used to embed the word image in a virtual bounding-box that provides space for descenders even if they do not appear. This center-line and virtual bounding box are illustrated in Fig. 7.

The transition-like features are computed on the virtual bounding-box of each of the individual segments of a pair of segments from a word image. Only horizontal transitions are considered, at most two per line per segment. Transitions are considered from the left and from the right. The positions of the transitions are recorded at eight equally spaced rows of the image rather than resampling the positions over a number of rows as is done for the character recognition transition features. The length of the resulting feature vector is $64 = 2$ segments * 8 lines per segment * (2 left-to-right transitions per line $+$ 2 right-to-left transitions per line). The compatibility network based on the transition-like features has 64 inputs, nine outputs, and one hidden layer with 54 units.

The bounding box features are the normalized distances above and below the word center line of each segment, the width of the overlap of the segments, the widths of the parts of each segment that do not overlap the other segment, and the relative offsets of the tops of the segments and the bottoms of the segments. The result is nine features altogether, compared to 64 for the transition-like features. A neural network was trained with nine inputs, 12 hidden units, and nine outputs. Word recognition results using these features are provided in Section VI.

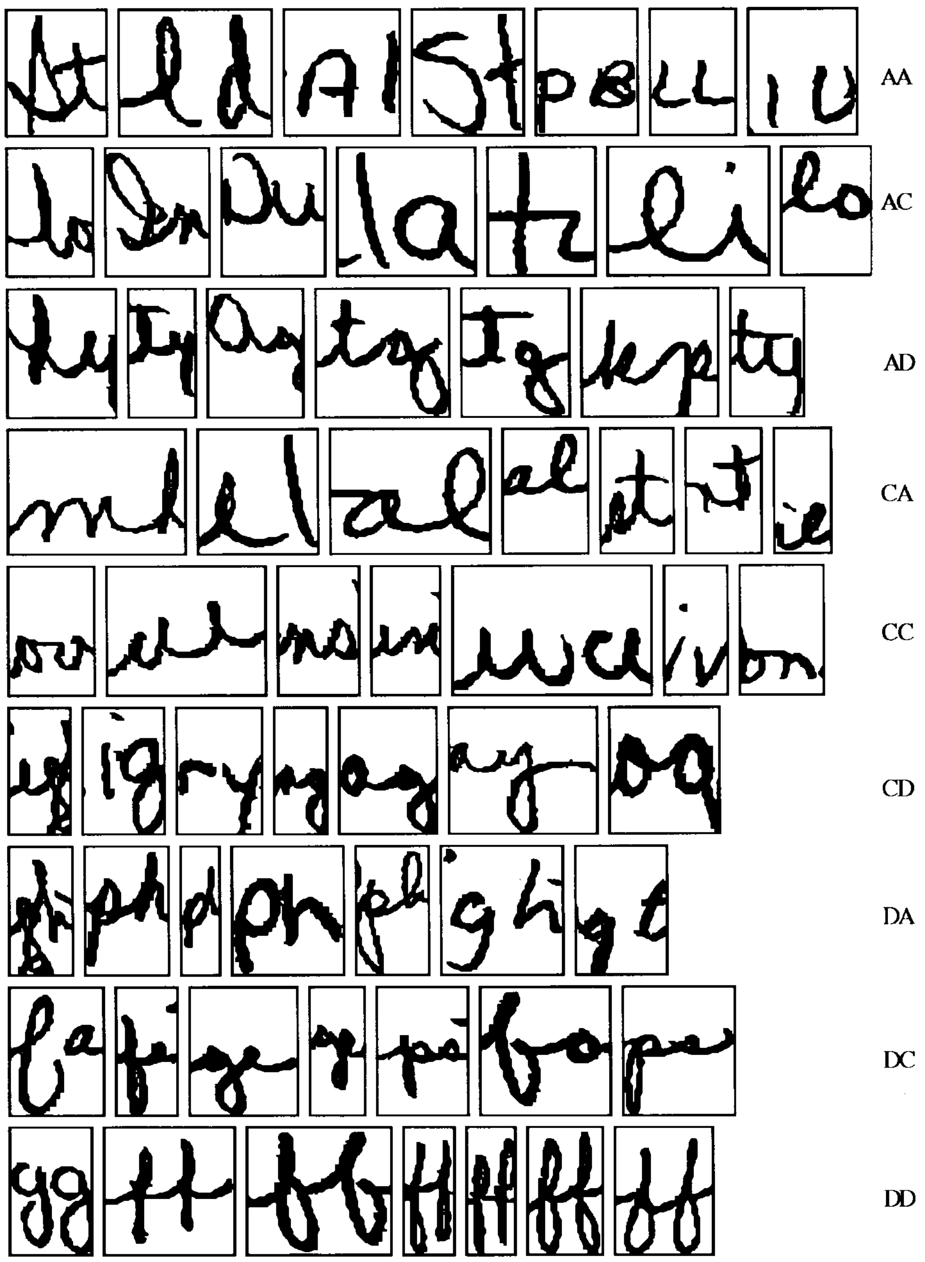

Fig. 6. Samples of character pairs used for training compatibility neural networks.

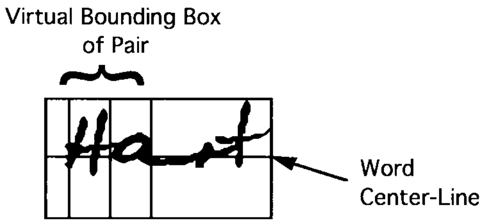

Fig. 7. Illustration of word center line and virtual bounding box.

V. Dynamic Programming Matching

The core of the dynamic programming algorithm is the module that takes a word image, a string, and a list of the primitives from the word image and returns a confidence value between 0 and 1 that indicates the confidence that the word image represents the string. Dynamic programming is used to find the best path through the space of primitives and legal unions of primitives. The best path depends upon the method to evaluate each node in the path. The value of each node here can be provided solely by the character neural networks or by combining the character network and compatibility network outputs. The value of a path is computed by averaging the values of the nodes.

The algorithm is implemented using a match matrix approach. We describe it first for the case that only the character neural networks are used. For each string in the lexicon, an array is formed. The rows of the array correspond to the characters in the string. The columns of the array correspond to primitive segments. The (i, j) element in the array is the value of the best match between the first i characters in the string and the first j primitive segments. This value may be negative infinity if there is no legal match. A path array can also be maintained to keep track of the best matches.

Let the primitive segments of the image be denoted by $S_1, S_2, \cdots, S_p$. Let the characters in the string be denoted by

$C_1, C_2, \cdots, C_w$. Let match(c, s) be a function that takes a character c and a segment image s as input and computes the confidence that s represents c. For each pair $m, n \in \{1, 2, \cdots, p\}$, let

$$S_{mn} = \bigcup_{h=m}^{n} S_h$$

and let Legal$p(S_{mn})$ be a Boolean function that returns TRUE if the union image S_{mn} is "character-like" and FALSE otherwise. The (i, j) element of the match array (called value) is computed as follows:

```
IF $i = 1$, (that is we are matching against
        the first character) THEN
    IF Legal$p(S_{1j})$ THEN
        value $(1, j)$ = match $(S_{1j}, C_1)$
    ELSE
        value $(1, j) = -\infty$
    END IF
ELSE IF Legal$p(S_{ij})$ THEN
    value $(i, j)$
        = $\max_k$ {value $(i-1, k)$+ match
        $(S_{k+1,j}, C_i) | i \leq k < j$ and Legal$p(S_{kj})$}
    ELSE
        value $(i, j) = -\infty$
    ENDIF
ENDIF
IF value $(i, j) > -\infty$ THEN
    path $(i, j)$
        = $\operatorname{argmax}_k$ {value $(i-1, k)$+ match
        $(S_{k+1,j}, C_i) | i \leq k < j$ and Legal$p(S_{kj})$}.
```

Once the match array has been computed, the best match can be found using the path array. This algorithm requires that we define the match function and that we define criteria for deciding when a union of primitive segments is legal.

A picture of all the legal unions of primitives for the word image from Fig. 3 is shown in Fig. 8. The criteria for legality of unions are based on measuring the complexity of the union in various ways. If a union becomes too complicated in terms of number of stroke transitions, or if it contains a large gap in the vertical projection (indicating that two characters have been grouped), or if the union extends across too many connected components of the word image, then it is determined to be not legal. These rules are clearly heuristic and depend upon thresholds, but they reduce the computational requirements considerably. The thresholds are set low to place most of the decision making power in the dynamic programming match function.

We can define the match function using only the isolated character networks as follows: Let S denote a binary image and let C represent a character class. Let $BF_U(S, C), BF_L(S, C), TF_U(S, C)$, and $TF_L(S, C)$ denote the output activations of the output node associated with class c for the upper and lower case Bar and Transition feature networks. Define the match between image S and character class C to be the maximum of the averaged output activations

$$\text{match}(S, C) = \tfrac{1}{2}\max(BF_U(S, C) + TF_U(S, C), BF_L(S, C) + TF_L(S, C)).$$

If we use compatibilities, the algorithm becomes more complicated. Assume we are using one of the compatibility networks. We take the match function to depend upon the value of the match of the current segment to the current character class under consideration and the compatibility of that match with matching the previous segment to the previous character class. More precisely, let $CN(S; T; p)$ denote the output activation of the pth node of the compatibility network

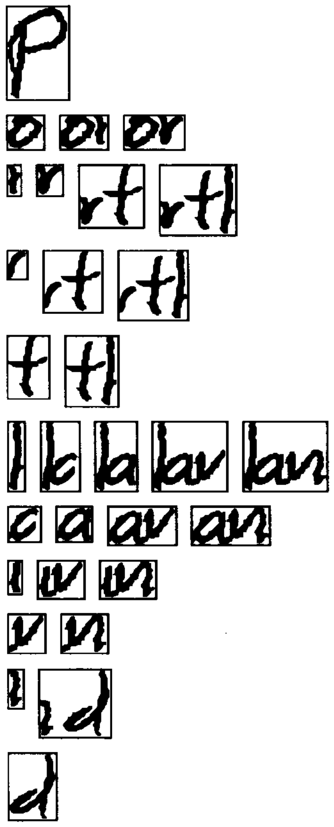
Fig. 8. The set of all legal unions of primitives for the word image of "Portland" shown in Fig. 3.

for $p \in \{\text{AA, AC, AD, CA, CC, CD, DA, DC, DD}\}$ given (feature vectors computed from) images S and T as inputs. Suppose we are currently trying to compute the best match between the first i characters in the string and the first j primitive segments. If $i = 1$ then the matching is the same as before. If $i = 2$, then for each $k < j$ such that Legal$p(S_{1k})$ and Legal$p(S_{k+1,j})$ are TRUE, let $(i-1, k)$ denote the position of a previous match. This means that the segment S_{1k} was used to match to the first character, C_1, and that we are currently considering the match between S_{k+1}, j and C_2. Let p_{12} denote the compatibility class associated with the character class pair (C_1, C_2). The match score is then given by

$$\text{value}(2, j) = \max_k \{\text{value}(i-1, k) + \text{match}(S_{k+1}, j, C_i) + a * CN(S_{1k}; S_{k+1}, j; p_{12}))\}$$

and the path pointer is given by

$$\text{path}(2, j) = \operatorname{argmax}_k \{\text{value}(i-1, k) + \text{match}(S_{k+1}, j, C_i) + a * CN(S_{1k}; S_{k+1,j}; p_{12}))\}.$$

In this formula, match is as defined above and a is a scaling factor that weights the relative contribution of the compatibility.

For the general case, if $i > 2$ then for each $k_1 \leq j$ such that Legal$p(S_{k_1} + 1, j)$ is true, consider $(i-1, k_1)$ and $(i-2, k_2)$ where $k_2 = \text{path}(i-1, k_1)$. If Legal$p(S_{k_2}, k_1)$ is true then, by definition of the path matrix, the segment S_{k_2}, k_1 was used to match to the previous character, C_{i-1}. For each such k_1, we compute the compatibility of the match between S_{k_2}, k_1 and C_{i-1} with the match between $S_{k_1} + 1, j$ and C_i, weight it by a scaling factor, and add it to the character confidence assigned to the match between $S_{k_1} + 1, j$ and C_i. The maximum value attained is the match value at (i, j).

TABLE I
COMPARISON OF CHARACTER RECOGNITION RATES

Network	UC (Train)	UC (Test)	lc (Train)	lc (Test)
Bar feature	88.03 %	85.14 %	83.45 %	79.62 %
Transition feature	86.31 %	82.72 %	83.84 %	80.17 %
Averaging	91.14 %	86.24 %	88.27 %	83.45 %

that is, let p_{i-1}, i denote the compatibility class associated with the character class pair (C_{i-1}, C_i). The match score is then given by

$$\begin{aligned}\mathrm{value}(i,j) = \max_{k1} &\{\mathrm{value}(i-1,k_1)\\ &+ \mathrm{match}(S_{k_1}+1, j, C_i\\ &+ a * CN(S_{k_2,k_1}; S_{k_1+1,j}; p_{i-1,i}))\}\end{aligned}$$

and the path array entry is the argmax of the expression being maximized as before. Actually, the case of $i = 2$ can be considered as part of the latter case, but it is easier to understand by treating it separately.

The final segmentation and match score are computed after the value and path arrays have been filled. Let w denotes the number of characters in the string being matched and let p denote the number of primitive segments. We generate indexes $\{k_0, k_1, k_2, \cdots, k_{w-1}, k_w\}$ according to

$$\begin{aligned}k_w &= p\\ k_{w-1} &= \mathrm{path}(w,p)\\ ki &= \mathrm{path}(i, k_{i+1}) \text{ for } i = w-2 \text{ downto } 1\\ k_0 &= 1.\end{aligned}$$

The final segmentation is then $\{S_{k_0,k_1}, S_{k_1+1,k_2}, S_{k_2+1,k_3}, \cdots, S_{k_{w-1}+1,k_w}\}$ and the match score is the average of $\{\mathrm{value}(1,k_1), \mathrm{value}(2,k_2), \mathrm{value}(3,k_3), \cdots, \mathrm{value}(w-1, k_{w-1}), \mathrm{value}(w,p)\}$.

VI. EXPERIMENTAL RESULTS

A variety of experiments were performed. All the data used came from images of addresses from the USPS mail. For each module, we first describe the data and then the results.

A. Character Recognition Results

The training and testing sets for the character recognition modules consisted of 250 characters from each class in each set. Characters were extracted from images of words. The horizontal locations of the characters were manually marked at the Environmental Research Institute of Michigan. These characters were automatically extracted by our group and then manually screened for accuracy. Some classes (such as vowels) had several thousand samples, while others (such as "q" and "j") had less than ten.

We constructed balanced training and testing sets from these samples. Each set had 250 characters from each class. For those classes with less than 500 samples, we constructed sets of 500 per class by randomly resizing the existing characters. For those classes with between 500 and 800 samples, we randomly selected 500 samples. For those with more than 800 samples, we used a clustering scheme to select samples. Each class was clustered into 250 clusters using the k-means algorithm with Euclidean distance. Clusters with a small number of samples were discarded. In order to represent the typical character shapes in proportion to their distribution in our data, we sampled from the remaining clusters in proportion to their size. Thus, a large cluster yielded more samples than a small cluster. This

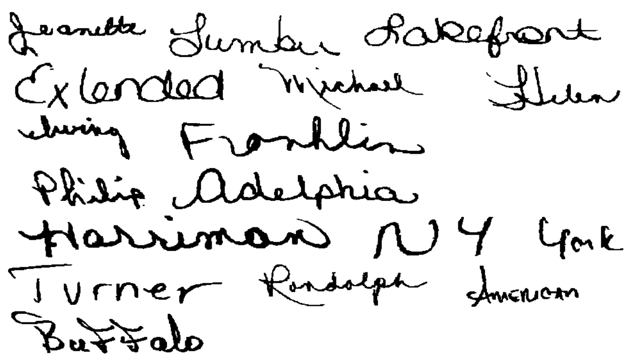

Fig. 9. Some sample word images.

process was performed separately for the training and testing sets. The training and testing sets were disjoint.

The networks were trained for 12 000 epochs using a special purpose computer system (the ANSIM system developed by SAIC) for neural network training. The training and testing rates are shown in Table I. We note that these results are different than those shown in [20] because we used different character sets.

B. Compatibility Recognition Results

The data for the compatibility training and testing sets were constructed similar to the character recognition sets. Samples are shown in Fig. 6. The compatibility networks were trained using 1000 samples per class for the transition like-compatibilities and 250 per class for the bounding-box compatibilities. The recognition rates for training were 90.6% for the transition-like compatibilities and 81.8% for the bounding-box compatibilities.

C. Word Recognition Results

The data set used for word recognition experiments consisted of 1000 city and state words selected at the Environmental Research Institute of Michigan. Each word had a lexicon of size 100 associated with it. The lexicons always contained the correct string. Samples of the words are shown in Fig. 9. The words used for word recognition experiments were distinct from the words used to extract characters and pairs of characters.

The following experiments were performed.

Experiment 1:

Word Recognition Using the Bar and Transition Feature Character Neural Networks only.

Experiment 2:

Word Recognition Using the Bar and Transition Feature Character Neural Networks and the Transition-like Compatibility Neural Networks. We used $a = 0.25$.

Experiment 3:

Word Recognition Using the Bar and Transition Feature Character Neural Networks and the Bounding-box Compatibility Neural Networks. We used $a = 0.25$.

The results are shown in Table II.

The table shows that the use of the compatibility networks significantly improves performance and that the bounding-box compatibility features perform essentially the same as the transition-like compatibility features. This last result is interesting since the bounding-box compatibility network is much simpler and the training classification rate is much lower than the transition like compatibility. This results reinforce a view that we hold which is that recognition performance is not the best measure of how well a confidence assignment module will work in the context of word recognition. We believe that accurate

TABLE II
PERCENTAGE OF CORRECT STRINGS IN THE TOP N RANKED STRINGS FOR SET OF 1000 HANDWRITTEN WORDS. THE LEXICON SIZE WAS 100 STRINGS FOR EACH WORD IMAGE AND THERE WAS A DIFFERENT LEXICON FOR EACH WORD IMAGE

N	Experiment 1 Character Networks Only	Experiment 2 Transition Like Compatibility	Experiment 3 Bounding Box Compatibility
1	79.5%	85.8%	85.3%
2	85.2%	91.0%	90.3%
3	88.6%	92.9%	92.7%

reflection of ambiguity is more important, although it is not clear how to measure accurate reflection of ambiguity.

VII. CONCLUSION

A word recognition algorithm employing dynamic programming, neural-network-based character recognition, and neural-network-based inter-character compatibility scores has been presented. It has been shown that the inter-character information can lead to a significant improvement in performance.

ACKNOWLEDGMENT

The authors would especially like to thank A. M. Gillies, D. Hepp, M. Ganzberger, and M. Whalen of the Environmental Research Institute of Michigan (ERIM), J. Tan of Arthur D. Little, Inc., and C. O'Connor of the U.S. Postal Service for their support of this work.

REFERENCES

[1] A. Gillies, "Cursive word recognition using hidden markov models," in *Proc. U.S. Postal Service Advanced Technology Conf.*, Washington, DC, 1992, pp. 557–563..

[2] M. Y. Chen, A. Kundu, and J. Zhou, "Off-line handwritten word recognition using hidden Markov model-type Stochastic network," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 16, pp. 481–496, May 1994.

[3] M. Y. Chen and A. Kundu, "An alternative to variable duration HMM in handwritten word recognition," in *Proc. 3rd Int. Workshop Frontiers in Handwriting Recognition*, Buffalo, NY, 1993, pp. 82–92.

[4] M. Y. Chen, "Off-Line handwritten word recognition using a hidden Markov model-type Stochastic network," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 16, pp. 481–496, May 1992.

[5] M. A. Mohamed and P. D. Gader, "Handwritten word recognition using segmentation-free hidden Markov modeling and segmentation-based dynamic programming techniques," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 18, no. 5, pp. 548–554, May 1996.

[6] M. A. Mohamed, "Handwritten word recognition using generalized hidden Markov models," Ph.D. dissertation, University of Missouri-Columbia, 1995.

[7] H. Park and S. Lee, "Off-line recognition of large-set handwritten hangul with hidden Markov models," in *Proc. 3rd Int. Workshop Frontiers in Handwriting Recognition*, Buffalo, NY, 1993, pp. 51–62.

[8] S. Madvanath, "Using holistic features in handwritten word recognition," in *Proc. U.S. Postal Service Advanced Technology Conf.*, Washington DC, 1992, pp. 183–199.

[9] S. Madvanath and V. Govindaraju, "Holistic lexicon reduction," in *Proc. 3rd Int. Workshop Frontiers in Handwriting Recognition*, Buffalo, NY, 1993, pp. 71–82..

[10] A. W. Senior and F. Fallside, "An off-line cursive script recognition system using recurrent error propagation networks," in *Proc. 3rd Int. Workshop Frontiers in Handwriting Recognition*, Buffalo, NY, 1993, pp. 132–141.

[11] B. Plessis, "Isolated handwritten word recognition for contextual analysis reading," in *Proc. U.S. Postal Service Advanced Technology Conf.*, Washington DC, 1992, pp. 579–593.

[12] J. Hull, T. Ho, Favata, J., V. Govindaraju, and S. Srihari, "Combination of segmentation-based and holistic handwritten word recognition algorithms," in *Proc. 2nd Int. Workshop Frontiers in Handwriting Recognition*, Chateau de Bonas, France, 1992, pp. 229–240.

[13] J. Favata and S. Srihari, "Recognition of general handwritten words using a hypothesis generation and reduction methodology," in *Proc. U.S. Postal Service Advanced Technology Conf.*, Washington DC, 1992, pp. 237–253.

[14] F. Kimura, M. Shridhar, and N. Narasimhamurthi, "Lexicon directed segmentation—Recognition procedure for unconstrained handwritten words," in *Proc. 3rd Int. Workshop Frontiers in Handwriting Recognition*, Buffalo, NY, 1993, pp. 122–132.

[15] F. Kimura, M. Shridhar, S. Tsuruoka, and Z. Chen, "Context directed handwritten word recognition for postal service applications," in *Proc. U.S. Postal Service Advanced Technology Conf.*, Washington, DC, 1992, pp. 199–214.

[16] E. Lecolinet and J. Crettez, "A grapheme-based segmentation technique for cursive script recognition," in *Proc. 1st Int. Conf. Document Analysis and Recognition*, Saint Malo, France, 1991, pp. 740–748.

[17] C. Nohl, C. Burges, and J. Ben, "Character-Based handwritten address word recognition with lexicon," in *Proc. U.S. Postal Service Advanced Technology Conf.*, Washington DC, 1992, pp. 167–180.

[18] P. D. Gader, M. A. Mohamed, and J. M. Keller, "Dynamic programming based handwritten word recognition using the Choquet Fuzzy Integral as the match function," *J. Electron. Imaging*, vol. 5, no. 1, pp. 15–25, Jan. 1996.

[19] P. D. Gader and M. A. Mohamed, "Multiple classifier fusion for handwritten word recognition," in *Proc. IEEE Conf. Systems, Man, and Cybernetics*, Vancouver, B.C., Canada, 1995.

[20] P. D. Gader, M. A. Mohamed, and J. Chiang, "Comparison of crisp and fuzzy character neural networks in handwritten word recognition," *IEEE Trans. Fuzzy Syst.*, vol. 3, no. 3, pp. 357–364, Aug. 1995.

[21] M. S. Obaidat and D. T. Macchairolo, "A multilayer neural network system for computer access security," *IEEE Trans. Syst. Man, Cybern.*, vol. 24, pp. 806–813, 1994.

[22] P. D. Gader, M. P. Whalen, M. J. Ganzberger, and D. Hepp, "Hand-printed word recognition on a NIST data set," *Mach. Vis. Applicat.*, vol. 8, pp. 31–40, 1995.

[23] P. D. Gader, A. M. Gillies, and D. Hepp, "Handwritten character recognition," in *Digital Image Processing Methods*, E. Dougherty, Ed. New York: Marcel Dekker, 1994, pp. 223–261.

[24] P. D. Gader, J. M. Keller, R. Krishnapuram, J. H. Chiang, and M. A. Mohamed, "Neural and fuzzy methods in handwriting recognition," *Computer*, submitted for publication.

[25] P. D. Gader, J. M. Keller, and J. Cai, "A fuzzy logic system for the detection and recognition of street number fields on handwritten postal addresses," *IEEE Trans. Fuzzy Syst.*, vol. 3, no. 1, pp. 83–96, 1995.

[26] P. D. Gader and J. M. Keller, "Applications of fuzzy set theory to handwriting recognition," in *Proc. 3rd IEEE Int. Conf. Fuzzy Systems*, Orlando, FL, 1994, pp. 910–917.

[27] J. M. Keller, M. R. Gray, and J. A. Givens, "A fuzzy k-nearest neighbor algorithm," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-15, no. 4, pp. 580–581, 1985.

[28] R. Wilkenson, J. Geist, S. Janet, P. Grother, C. Burges, R. Creecy, B. Hammond, J. Hull, N. Larsen, T. Vogl, and C. Wilson, *The First Census Optical Character Recognition Systems Conference*, National Institute of Standards and Technology, Gaithersburg MD, NISTIR 4912, Aug. 1992.

[29] J. Chiang, "Hybrid fuzzy neural systems for robust handwritten word recognition," Ph.D. dissertation, University of Missouri-Columbia, 1995.

[30] A. M. Gillies, D. Hepp, M. Ganzberger, R. Rovner, and P. D. Gader, "Handwritten address interpretation," report from ERIM to U.S. Postal Service, Office of Advanced Technology, 1993.