Recognition on Handwritten Digits Based on Their Topological and Morphological Properties

Vladimir Delevski[1], M.Sc. E.E., and Srdjan Stankovic[2], Ph. D. E.E.

[1]Institute "Mihajlo Pupin", Belgrade, Yugoslavia
vlada@l100.imp.bg.ac.yu
[2] The Faculty of Electrical Engineering, Belgrade University, Yugoslavia
SStankovic@mailer.scu.edu

Abstract – This paper presents a system for off-line handwritten numeral recognition based on topological properties of the digits. The first step in recognition algorithm is graph evaluation, obtained from the RLC of the digit, the second is measuring the geometrical properties of the elements of the graph and classification based on that measures. Algorithm was trained and tested on CEDAR database and it achieved correct recognition rate of 99,74%.

1 Introduction

The computer recognition of handwritten numerals is still a very challenging problem for many researchers. Basic intention is to achieve the level of recognition that humans have. For that purpose many algorithms are developed, which could be generally separated in two major classes: structural [1],[2],[3],[4], and statistical [5],[6],[8],[10],[11]. Algorithms of both classes often require additional preprocessing as skeletonization, picture normalization, or filtering.

Algorithms of the first class consider some geometrical properties of the digits and try to explain the digits in some meaningful features such as lines, arcs, contours, graphs, etc. Classification methods could be statistical or structural (like decision trees).

The system we present here belongs to the first class of algorithms and it is based on analysis of topological and geometrical properties of the picture. By processing the picture which comes in RLC (Run Length Code) format, we evaluate its graph representation (LAG, [3] or GR, [9]), which is the base for the further recognition. Proposed method of evaluation of graph as the representative of digit's topology is very robust and captures a lot of information in a small feature vector. This system goes step further than similar algorithms ([1], [3], [9]) in defining the exact feature vector for graph of singular runs of the picture's RLC.

The basic feature for classification is the *graph*, i.e., topological representation of the digit. When we evaluate the graph of the digit, further analysis is performed in

order to make correct classification. We distinct horizontal (if we go rightward), and vertical graphs (if we perform downward scan of the digit). By inspecting horizontal and vertical graphs, as well as their components, we obtain a significant information how digits look like, trying to imitate human's way of recognizing.

Section 2 presents the input data format. In Section 3 we describe evaluation of graph representation of digits and define the basic structures we need. Graph filtering is described in Section 4. Formatting the feature vector of graph and its properties are explained in Section 5. Section 6 deals with further classification (for the certain graph), Section 7 presents the results, and Section 8 concludes the paper. The system has been trained and tested on the CEDAR datasets (BR and GOODBS samples, respectively).

2 Input Data Format

As the input to the system we use already segmented digits, prepared in such way to consist of only one connected geometric figure (e.g., two-stroke 5's were connected on nearest place to form one connected figure, which is not a problem if the digit has already been segmented correctly). This connecting was the only preprocessing in the system, if we do not consider that a segmentation.

Input data for graph finding part was in format of array of segments (continuos runs of black pixels), i.e. RLC, where each segment was defined by three coordinates (x, y, z), where (for vertical segments – i.e., horizontal graph), x is the x coordinate, and y and z are the y coordinates of the start and the end of the segment. Dimensions of the input picture are equal to the minimum picture bounding rectangle, and they are completely arbitrary.

3 Definition of Basic Structures

The basic structure we start from in graph finding is the *segment*. Segment is continuos run of black pixels in the single column of the picture matrix (for horizontal graph). Segments that belong to the same column we note as

$$S_{i,j} \in L_i, 1 \le j \le k; 1 \le i \le M \tag{1}$$

where k is the number of segments in the i-th column (L), and M is the number of columns in the picture matrix. Segment (i,j) is made of pixels

$$a_{k,i} = 1; S_{i,j}^{start} \le k \le S_{i,j}^{stop} \tag{2}$$

Two segments (i,j) and (k,l) are neighbors if they satisfy:

$$abs(k-i) = 1 \wedge S_{i,j}^{start} \le S_{k,l}^{stop} + 1 \wedge S_{i,j}^{stop} \ge S_{k,l}^{start} - 1, \tag{3}$$

i.e. L_i and L_k are neighbor columns and projections of these two segments on the y-axis are overlapping or touching.

If we define binary operation on segments, that for two neighbor segments $S_{i,j} \bullet S_{k,l} = 1$, we could define *regular* and *singular* segments.

For the *singular* segment $S_{i,j}$ holds

$$\sum_{k} S_{i-1,k} \bullet S_{i,j} \neq 1 \vee \sum_{m} S_{i+1,m} \bullet S_{i,j} \neq 1, \tag{4}$$

and such a segment we call the *node*. Any segment which is not a node is the *regular* segment.

Depending on number of neighbor segments on the left and right side, there are four different types of nodes:

$$i = 1 \vee \sum_{p=1}^{K_{i-1}} S_{i,j} \bullet S_{i-1,p} = 0, \tag{5}$$

$$i = M \vee \sum_{p=1}^{K_{i+1}} S_{i,j} \bullet S_{i+1,p} = 0, \tag{6}$$

$$\sum_{p=1}^{K_{i-1}} S_{i,j} \bullet S_{i-1,p} = 2, \tag{7}$$

$$\sum_{p=1}^{K_{i+1}} S_{i,j} \bullet S_{i+1,p} = 2, \tag{8}$$

Node types for these singular segments are *start, end, joint,* and *fork*, respectively.

There is a very simple solution to limit the number of neighbors at two for segments that really have more than two neighbors (explanation is given in the description of branches of zero length). This limitation is very important because it limits the possible number of branch types as well.

Segment can be double node - start and end, start and fork, joint and fork or joint and end, or (if it really has three neighbors) double joint or double fork.

The *branch* of the graph is the union of the segments between two nodes for which stands:

$$\forall S_{i,j}, S_{i+1,k} \in G \Rightarrow \sum_{p=1,K_{i+1}} S_{i,j} \bullet S_{i+1,p} = 1 \wedge \sum_{p=1,K_i} S_{i,p} \bullet S_{i+1,k} = 1 \tag{9}$$

All regular segments between two nodes belong to the branch, as well as nodes which are branch ends, if they have exactly one neighbor on the branch side (forking segment belongs to the branch which is coming into it, and joining segment belongs to the branch which is coming out from it, while for starting and ending segments it is obvious which branches they belong to).

As a result of the previous definition we have branches of zero length - that is obvious if the branch start is the fork, and the branch end is the joint (in neighbor lines). Special cases of branches of zero length occur at really triple joints or forks, where segment of triple forking (joining) is the double node, causing that at forking, the first and the second neighbor segments in next line are the upper forks, and at

joining, the second and the third neighbor segments in previous line are the lower joints.

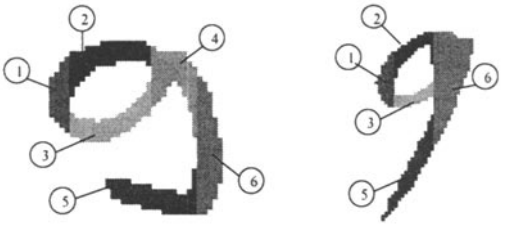


Fig. 1. Zero length branch example (branch 4 on the right); both digits have the same graph

The *type* of the branch is determined by the type of starting node and type of ending node. Possible types are line start (LS), fork upper (FU), fork lower (FL) and joint (J) for starting type; and line end (LE), joint upper (JU), joint lower (JL), and fork (F) for ending type.

Combination of four starting and four ending types results in sixteen different branch types.

The *graph* is the set of the singular segments (nodes) of the picture with defined connections (branches) between nodes. It could be mapped in different ways (Section 5).

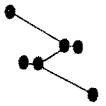
4 Noise Filtering

In order to eliminate errors caused by a great number of branches which are the result of the small holes in the black neighborhood or small peninsulas of black pixels (caused by improper quantization or by pencil shaking errors), we perform the generalization of the segments and the branches at the following stages: segments at the segment determining time, and branches after branch finding. For segments, if two adjacent segments in the same line are closer than certain threshold, they are joined by ignoring white gap between them. For branches, we delete "limb" branches (i.e., branches which begin at fork and finish with end, and branches which begin with start and finish at joint) if they are shorter than certain threshold and also thinner than certain threshold, and reconnect branches which are their neighbors (their starting and ending types are adjusted). Contours (simple or complex) shorter than certain threshold are filled, with proper adjustment of neighbor branches. This filtering drastically reduces the number of different graphs for observed samples, and it is very quick, because it is performed on very small amount of data.

5 Feature Vector Formatting

Graph could be mapped on matrix of connections between graph nodes. One realization could be square upper triangular matrix with dimensions equal to number

of nodes sorted in increasing x – order, where rows represent starting nodes, and columns represent ending nodes of branches. Nonzero elements of matrix are 1 (for branches of type LS-LE, LS-F, J-LE, or J-F). U(pper) and L(ower) (for appropriate connections between starts and joints, or between forks and ends), as well as two-character combination of U and L (for connections between fork and joint).



	S1	S2	F	J	E1	E2
S1	●			U		
S2		●	1			
F			●	UL		L
J				●	1	
E1					●	
E2						●

	S1	S2	F	J	E1	E2
S1	●		1			
S2		●		U		
F			●	UL		L
J				●	1	
E1					●	
E2						●

Fig. 2. Example of two isomorphic graphs and their matrix representations

By linear transformation of rows and columns of presented matrix, while preserving the order of nodes on every graph path (i.e., we cannot swap connected nodes), we obtain matrices of isomorphic graphs (two matrices on Fig. 2).

All isomorphic graphs have the same number of nodes of same types, and also have same paths between nodes of type start an nodes of type end. By description of the graph with model that contains paths between starting and ending nodes, we obtain feature unique for all isomorphic graphs.

Although this model gives us unique feature for all isomorphic graphs, such description is not comfortable for computer processing, due to its multi-dimensional nature (any path can have different number of nodes, and graphs can have different number of paths). Instead of that model we use feature vector which consists of number of occurences of branches of different type on the graph. Due to the fact that we have 16 different branch types (Sect. 3), this feature vector has length of 16.

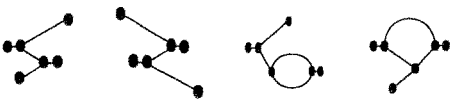


Fig. 3. Examples of the different graphs (horizontal)

Feature vectors for graphs shown at Fig. 3 are 0011100001001000, 0101001010001000, 0001110000111000, and 0011010001001010, respectively. The mapping of isomorphic graphs into this feature vector is unique, but the opposite mapping is not always unique (some different graphs have the same feature vectors). For graphs which have a small number of nodes, (usual case for digits), we can almost always get unique graph from its feature vector. However, even for feature vectors which are the same for different graphs, we can always make the right conclusion based on information (saved in the system) about neighbor nodes.

Using the complete combination of horizontal and vertical graphs (feature vector with dimension of 32), we got 949 different features for the training set (18450 samples). If we finish recognition at this stage (i.e., if we recognize the digit as the most frequent digit for that feature), we would yield the recognition rate of 89.60%

(with 10,40% misclassification). In order to decrease the number of features, we use the combination of horizontal and vertical graphs in such way that for horizontal graphs with 2 or 3 "native" nodes (forks and joints), feature is horizontal graph, and for all other horizontal graphs, feature is vertical graph. Hence, the feature vector has the dimension of 17, where the first digit is added to indicate what kind of graph is explained by next 16 digits: horizontal (1), or vertical (2, 3, 4, or 5) where horizontal graph was: without forks and joints (2), with exactly one fork and no joints (3), with one joint and no forks (4), or with more than 3 forks and joints (5). By this way of graph representation we found 170 different features for the training set, with recognition rate (if stopped at this stage) of 87,18%.

6 Classification of Digits for Single Graph

When we have found the graph of the incoming digit, we have not done yet all the job we need for correct classification of digit, because two or more different digits can appear in that graph with their representatives. Further distinction between different digits in the same graph is achieved by comparison of various morphological properties of the graph: relative branch lengths, branch angles, branch positions, etc. In order to decrease the processing time, and speed up the classification algorithm, as well as to avoid possible errors caused by similarity of certain branches, only the most important properties of the graph are taken into consideration in this part of the algorithm (i.e., we consider only the branches (branch types) that differ most for different digits in the same graph). As the result of such analysis, we make the conclusion what digit it is, i.e., what class in that graph.

Fig. 4. Example of horisontal and vertical parts on the same branch

For some graphs it is necessary to perform an additional analysis of branch width in order to make a conclusion about presence of different components (horizontal and vertical) in one branch [3], which is not the case for the largest number of graphs where we usually look every branch as a single line component (horizontal or vertical). In that sense we could form the feature vector of appearances of "thick" and "thin" parts, with additional information how "thin" (i.e., vertical, corresponding to the Fig. 4.) parts are placed regarding to the "thick" (horizontal) parts. Such analysis is performed usually for the graphs with the small number of branches (all digits on Fig. 4 have the same horizontal (single joint) and vertical graph (no joints or forks)).

Common reasons for varying between different graphs for samples of same digit are non-closed contours (0's, 6's, 8's, 9's) or degenerated (filled) contours (6's, 8's),

as well as different angle of writing mixed with the line width (2's, especially 3's and 4's), additional strokes (ending stroke down left for 5's), or simply different writing style (European 1's are written by two strokes and are different from American, European 7's usually have horizontal line at the middle, American 2's have contour in lower left corner, etc.).

During the classification we also perform the graph simplification, i.e., reduction of the complicated graphs to basic graphs, by filtering them with the increased threshold for keeping branches on the graph. This technique was very efficient, because it helped us to recognize near 3% of total number of samples, which occupy near 80 (almost 50%) graphs in simpler graphs. For some graphs (more than 30) we concluded that only one digit can appear in that graph. Hence, the number of graphs for "real" classification is reduced to less than 60.

Once formed, this system does not have the ability of self-learning, but it is designed to cover all major graph occurrences for ten digits. Unknown graphs result in rejection of the samples.

7 Test Results

Table 1. Overall accuracy of the recognition system (on CEDAR samples).

	Correct	Substituted	Rejected	Reliability
Training set	18401 (**99,74%**)	13 (**0,07%**)	36 (**0,19%**)	**99,93%**
Testing set	2199 (**99,37%**)	8 (**0,36%**)	6 (**0,27%**)	**99,64%**

Program is also tested on more than 700 Yugoslavian post cheques, with more than 2700 additional characters, also with accuracy of over 99%.

Processing speed on the Pentium 133MHz under MS-DOS operating system was 5.4 ms per sample for graph finding, 50 μs per sample for graph filtering, and 220 μs per sample for classification. This means about 5.7 ms for a complete processing of the digit, which is over 170 characters per second for digits of average dimension of 45x57 pixels (width x height). Additional reading from the database files (from the hard disc) slows down the system to process almost 140 characters per second.

8 Conclusion

We developed very robust system for recognition of handwritten digits based on topological properties of digitized picture of the digit. Introduced feature vector is insensitive to large variations in writing style (for example, more than 90% of samples of digit '5' have the same graph), and to changes in sample size and smaller variations of writing angle. The system needs minimal preprocessing, which means operating with already reduced input data set (graph filtering), in comparison with various algorithms that require skeletonization or normalization at specific size.

We tried to imitate human's way of recognition during the classification stage. Because of the way of feature interpretation, as well as classification in certain graph, the system also behaves very well on the samples significantly different from samples it was trained on (Fig. 6).

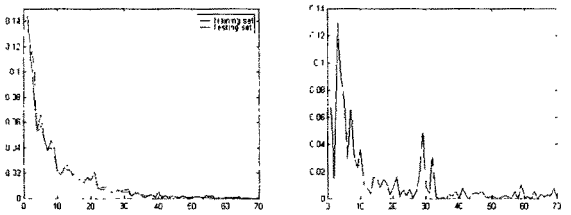


Fig. 5. Feature distribution of samples: left - CEDAR samples; right - Yugoslavian cheques

Speed and accuracy are very good and competitive with these presented by other authors, tested on the same or different datasets ([6], [7], [8], [10], [11]).

References

1. P. G. Perotto, "A new method for automatic character recognition," *IEEE Transactions On Electronic Computers*, vol. EC-12, pp. 521-526, Oct. 1963.
2. F. Ali and T. Pavlidis, "Syntactic recognition of handwritten numerals," *IEEE Transactions On System, Man, and Cybernetics*, vol SMC-7, pp. 537-541, July 1977.
3. T. Pavlidis, "A vectorizer and feature extractor for document recognition," *Computer Vision, Graphics, and image processing*, vol. 35, pp. 111-127, 1986.
4. M. Shridhar and A. Badreldin, "Recognition of isolated and simply connected handwritten numerals," *Pattern recognition*, vol. 19, pp. 1-12, 1986.
5. L. Stringa, "A new set of constraint-free character recognition grammars", *IEEE Transactions on pattern analysis and machine intelligence*, vol. 12, pp. 1210-1217, Dec. 1990.
6. Ching Y. Suen, Christine Nadal, Raymond Legault, Tuan A. Mai, and Louisa Lam, "Computer recognition of unconstrained handwritten numerals," *Proceedings of the IEEE*, vol. 16, pp. 1162-1180, July 1992.
7. T. Wakahara, "Shape matching using LAT and its application to handwritten numeral recognition," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 16, pp. 618-629, June 1994.
8. S. J. Smith, M. O. Bourgoin, K. Sims, and H. L. Voorhees, "Handwritten character clasification using nearest neighbor in large databases," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 16, pp. 915-919, Sept. 1994.
9. S. Di Zenzo, L. Cinque, and S. Levialdi, "Run-based algorithms for binary image analysis and processing," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 18, pp. 83-89, Jan. 1996.
10. Paul. D. Gader and Mohamed Ali Khabou, "Automatic feature generation for handwritten digit recognition," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 18, pp. 1256-1261, Dec 1996.
11. T. M. Ha and H. Bunke, "Off-line, handwritten numeral recognition by perturbation method," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 19, pp. 535-539, May 1997.