THE TEACHABLE LETTER RECOGNIZER

James Geller
Department of Computer Science
State University of New York at Buffalo
Buffalo, NY 14260
geller%buffalowcsnet-relay

ABSTRACT

The "Teachable Letter Recognizer" (TLR) is a program for letter learning which uses a method not vet described in the character recognition literature. TLR has two main characteristics: (1) It uses a discrimination tree as a knowledge representation. The discrimination tree is a quadtree with some additions. (2) The two types of global features that are used to characterise a letter are the density of pixels and the overall "color", white, black or grey. TLR is invariant to shifts and shows several interesting effects which are related to human behavior, for example occasionally it becomes confused when learning new letters. TLR's importance as a letter recognition program lies in its ability to recognize some distortions of letters which it has never seen before, and for which it also does not have a transformation algorithm.

1 INTRODUCTION

This paper describes the Teachable Letter Recognizer" (TLR). TLR belongs to the class of programs which are intended to know nothing about the domain of usage and acquire their knowledge by interaction with a "teacher". TLR owes some of its goals, as well as its name, to the 'Teachable language Comprehended (Quillian, 1969). Instead of programming' knowledge into TLR 1 want to build a general knowledge representation and then teach letters by showing them to TLR. The other important A! source for TLR was the idea of using a discrimination net as a knowledge representation. This has been influenced by EPAM (Leigenbaum, 1963).

TLR is conceptually different from classical approaches to letter recognition. Nevertheless TLR shares with some graphics programs the use of a quadtree as a hierarchical data-structure. Eor a quadtree oriented approach to shape recognition see (Chien and Aggarwal, 1984). The reader is referred to (Samet, 1984) for a survey about quadtrees.

To show the contrast between TLR and classical methods of character recognition 1 want to compare it with two of the major techniques that have been discussed in the literature use of low level features and template matching. Eor a review of some prin ciples of character recognition see (Harmon, 1972). A psychological treatment of letter recognition can be found for instance in (McClelland and Rumelhart, 1981).

One of the classical approaches in character recognition is to employ some method of low level feature extraction. Examples of low level features include "line segments", "intersections", "corners", etc. Without questioning the idea of using features per se it seems disturbing that a vision system should have to do a vast amount of computation before it is able to say anything significant about the object that it is "watching". Even though there is a lot of evidence that early processing in the human brain is not accessible to introspection, it still seems obvious that we can say something about the appearance of an object *before* we have perceived all its little details. Therefore we have to look for different types of features that are more "global" or "structural".

The fact that 1 want to look for "structural" as opposed to local features does not exclude a bottom up approach. Recognition of the global features should therefore not require the computation of small features, this means that "structural features" should be easily accessible "on the surface". In other words, 1 am looking for a characterization of an object which can be obtained by *ignoring* details.

The other approach that 1 want to mention as different from TLR has been referred to by Harmon as a "parallel processing" technique, namelv template matching. Template matching does not create the same problem as local feature extraction because it is definitely a global process. However, template matching seems to make immediate use of all the information available in the picture instead of accessing the easily available structural features first and looking at details later.

TLR gains a lot of cJantv bv using a very simple control mechanism namelv depth first tree search.

II TLR

A. Input Data Format

Conceptually the reader can think of the input data as consisting of arrays of binarv values with each arrav representing one letter. Due to the fact that the program is implemented in LISP a letter is represented as a list structure of b's and w's. Figure 1 shows a reduced (8x8) binarv matrix and the corresponding list representation of the letter "1".

B.

Informally a quadtree is a tree representing a picture where every node is either a leaf or has exactly four sons. A leaf contains either a "w" (white) or a "b" (black) and represents a square

```
0 0 0 0 0 0 0 0     (w w w w w w w w)
6 0 0 0 0 0 0 0     U w w w w w w w)
0 0 0 1 0 0 0 0     W w w b w w w w)
0 0 0 0 0 0 0 0     W w w w w w w w)
0 0 0 1 0 0 0 0     (w w w b w w w w )
0 0 0 1 0 0 0 0     (w w w b w w w w)
0 0 0 1 0 0 0 0     (w w w b w w w w)
0 0 0 1 0 0 0 0     (w w w b w w w w) )
```

Figure 1. Matrix and List Representation of " i "

of the picture in the given color. A non leaf is said to represent a grey area and is obtained by combining four sons in a way that a square of double the length of the "son squares" is built, the root represents the whole picture. If the root is a leaf, then it describes an entirely black or entirely white square. The precise method of representing a picture with a quadtree is explained in (Samet, 1984).

The quadtree used in this paper adds three aspects to a normal quadtree, (l) The representations of all known letters are integrated into a single tree structure. (2) All nodes contain information about the objects they belong to. (3) Every node contains information about the relative densities of black pixels of the area it is representing (relative to the father area).

Item 3 from above is the source for the name given to this type of quadtree, namely "density-quadtree". Items 1 - 3 seem to justify the treatment of a density quadtree as a form of knowledge representation as opposed to simply a data structure. I will refer to the additional information in the quadtree as "object knowledge'. Figure 2 shows a BNE definition of a density-quadtree.

```
<dens-q-tree>     ::=  <node>
<node>            ::=  (<son> <son> <son> <son>)
<son>             ::=  (<w-seg> <b-seg> <m-seg>)
<w-seg>           ::=  (w <value-hint-l> )|nil
<b-seg>           ::=  (b <value-hint-l> )|nil
<m-seg>           ::=  (m <node> <value-hint-l>)|nil
<value-hint-l>    ::=  <value> <hint-list> ....
                       .... <value> <hint-list>
<hint-list>       ::=  (<hint> .... <hint>)
<hint>            ::=  a|b|c|d .... z
<value>           ::=  0.00 .... 1.00
```

Figure 2. BNT Definition: Density-quadtree

"0.00 ... 1.00 in Figure 2 stands for all rational numbers between 0 and 1 with a precision of two digits after the period. This is an arbitrary limit which is easily changeable.

In order to explain the meaning of the different parts of a density-quadtree a small example follows. Assume the following <son>:

```
((w 0.3 (a e o) 0.4 (g q))
 ml
 (m (....) 0.3 (j l k)0.41 (m n)))
```

The line containing "w" indicates that the given node is a leaf node for the letters (a e o g q). This means that all these letters have a purely white sub-area corresponding to this node. The line consisting only of "nil" indicates that this node is not a leaf for any letter with a corresponding purely black area. Finally the line containing an "m" indicates that the letters (j 1 k m n) are grey in the given area. Therefore they have to be described by a recursive sub-tree which is indicated by (....).

The number 0.4 is the approximate density of black pixels at this node relative to the number of black pixels of the area corresponding to the immediate parent node for the letters "g" and "q".

c Learning

In the learning phase letters given in a format similar to Figure 1 are subjected to a density-quadtree analysis. The tree structures of all letters are integrated into *one* single density-quadtree. The process of doing this and the very structure of the tree itself are designed to permit a high degree of structure sharing between objects that are looking similar.

On the other hand two letters create a more complicated tree if they are very different, because they can share less information. The important characteristic of a letter is its *structural distribu tion* of black, white and grey values, and letters share structural distributions if their two-dimensional appearances are similar.

The second measure used to ensure that the tree is not growing unnecessarily is to keep the number of <value>s in the <value-hint-l'> small by using a learning scheme with a "competing letter". Any time TLR is asked to learn a new letter it first tries to *recognize* this new letter. If the recognition gives a different letter name than the one told by the teacher, then there is a potential danger of repeating this mistake at the next recognition process. Therefore the letter obtained bv the recognition process is called the competing letter' and given as an additional parameter to the tree building function.

If the current new letter has a density value that is slightly different from all the values at the currently analyzed node, then the tree building function will not add a new density value, *unless* this would put a letter and a competing letter together into the same <hint hst>. If the new letter was already in the <hint-list>, then nothing will be added at all. (This could hap pen if two slightly different handwritten versions of the same letter are taught).

Finally it is necessary to mention that there is a preprocessor for "learn*' that removes the "white strips" around the letter before quadtree analysis starts. This makes the recognition process invariant towards shifts but unfortunately forces the pro gram frequently to deal with areas of odd numbered size which cannot be broken down evenly. In this case the analysis is contin ued with four areas of different size.

D. Recognition

In the recognition phase an unknown letter is analyzed in the same way as in the learn phase. However, while analyzing the area, the integrated quadtree is also traced through its corresponding nodes. If the unknown letter happens to have a black area at the given level, then analysis stops for this branch of the quadtree and the <hint-list> of the <b-seg> is added to a result list. (The analogue thing happens for a purely white area).

If the unknown letter is grey, and the given node contains an <m-seg>, then analysis is continued. The four sons of the subtree are recursively analyzed relative to the four sub-areas of the area. <hint-list>s for the computed densities are added to the result list. This use of the quadtree bears a strong resemblance to the use of a discrimination tree. After the whole tree has been searched in the described way, the result list might look like ((a b) (a) (a c) (c) (a) (a)) for a quadtree which contained pictures of "a", "b" and "c".

The most important characteristic of the result list is that it contains sublists of varying length. Sublists that were contributed by the top nodes in the density quadtree tend to be long, because on the top levels most letters look the same. This sounds like a strong statement, but on the top level all letters are grey and therefore similar.

TLR decides what letter it saw by counting the number of occurrences of each letter in the result list. However before this is done the result list is pruned by removing all sub lists that are longer than a so called "effort" value. If the effort value is set to a low value (1-3) then the evaluation of the result list can be done comparably fast. It is not possible, however, to rely strictly on the short sub-lists derived from the leaves because the analysis of a new distortion might bottom out at an earlier level. Therefore in some cases short sub-lists alone might not be sufficient and a higher effort \alue would be necessary.

After learning more and more letters the length of <,hmt list>s will get longer even at the leaves which will also require the user to raise the effort of recognition.

III RLSLLTS OK WORKING WITH TLR

In this section two series of tests of TLR will be described. The goal of both of them was to investigate TLR's most important feature: Its ability to recognize some distortions that it has not been confronted with before.

The first experiment used onlv 4 letters, namelv "a", "b". "g" and "o"; "a" and "o" were chosen because they look similar in many handwritings, Fvery letter was shown in one original form and ten different distortions. That means TLR was shown eleven sets of the letters "a", "b", "g" and "o" in precisely this order. The first set of lour letters could of course not be recog ni/ed, because TLR is started without any prior knowledge. The results of this test run were quite encouraging.

Thirt\ one distortions out of fortv were immediately recognized. Nevertheless, all of them were presented a second time to TLR in order to improve its knowledge. Lor three of the letters TLR suggested two or more possible solutions with almost identical hint counts. In these cases correct solutions could be obtained by raising the "effort level". In six cases TLR confirmed wrong results. After teaching these letters again TLR finally recognized them.

Test data consisted of hand printed lower case letters of one subject that were "digitized" with an editor. A bias to the handwriting of the subject cannot be excluded, however an effort was made to make distorted letters really different.

Lnforcing knowledge of one letter can weaken the memory of another letter, because the relative number of hints for the second letter diminishes in the tree. To test for this possibility all forty letters were presented again for recognition after the experiment was finished. Four letters were recognized incorrectly, and three out of these four required unexpectedly high effort values (approximately 8) to be recognized correctly. This stands in contrast to the case where TLR returns two letters with similar hint counts, because in that case the necessity of a higher effort level is immediately obvious.

One positive result of this experiment was that the number of errors that TLR made became smaller with greater knowledge (cf. Table 1.); however this gain was not accompanied by a degradation of the recognition times. It will require tests with larger data sets to find out whether a saturation effect of learning can be achieved.

Learning times for a letter were usually less than one minute real time on a VAX-750. Recognition times varied between several seconds and almost one minute, however were mostly in the order of fifteen seconds.

Cycles	Letters	Errors
1	4	0
2	8	3
3	12	4
4	16	5
5	20	6
6	24	7
7	28	9
8	32	9
9	36	9
10	40	9

Table 1. Number of accumulated errors

The second experiment involved a test with the complete alphabet. Two sets of the entire alphabet were used, one learning set and one set of distortions. This time letters were not presented again after successful recognition. Seventeen out of twenty six letters were recognized immediately, one after adjusting the effort level, and eight letters required additional teaching.

It should be noted that some of the distortions that were recognized were of different size or consisted of strokes of double width. These two types of variations could be recognized *without* the use of a scaling algorithm or a thinning algorithm. However TLR was not designed to deal with rotated letters.

IV CRITICISM OF TLR

When TLR finds its own competing letter to differentiate it from a new letter that has to be learned, then some of its choices would surprise a human observer. So for example "i" was taken as an obvious competing letter for "j", but "g" was chosen as "competing letter" for "k", although people do not consider these letters to be similar. A detailed analysis of the density quadtree should show similar density values for these two letters. One reason why we see "g" and "k" as different and TLR does not is that we have a literal base line in mind, when we look at the letters. We also might think of words where "g" occurs and "g" clearly extends below the word but "k" does not. TLR never saw a "base line", and it does not know words.

V CONCLUSIONS

This paper introduced a letter recognition program called the "Teachable Letter Recognizer which uses a quadtree with added object knowledge as a discrimination tree and knowledge representation. Global structural features, namely the b/w/grev structure and relative density values are used for the recognition process. An intelligent learning process that avoids unnecessary information and is based on "competing letters" is used. An effort value permits TLR to prune a list of hint-lists. Some distortions which are not known to the program beforehand and for which *no Hans formation algotithm* is implemented are nevertheless recognized.

ACKNOWLEDGEMENTS

I am indebted to Jon Hull, David Shapiro, Stuart Shapiro, Soon Chun and three anonymous referees for valuable contributions.

REFERENCES

Chien, *C.W.* and Aggarwal, J.k., "A Normalized Quadtree Representation', *(Computet \ision, Graphics and Image Process ing*$_t$ Vol. 26, 1984, pp. 331 346.

Feigenbaum, E.A., "The Simulation of Verbal Learning Behavior", in: Feigenbaum, L.A. & 1 eldman, J. (Eds.) *Computers and Thought,* McGraw-Hill, 1963.

Harmon, L.D, "Automatic Recognition of Print and Script", *Proceedings of the IEEE,* October, 1972, pp. 1165-1176.

McClelland, J.L. and Rumelhart, D.E., "An Interactive Activation Model of Context Effects in letter Perception: Part 1. An Account of Basic Findings', *Psychological Review,* Vol. 88, N. 5, 1981, pp. 375-407.

Quilhan, M.R., The Teachable Language Comprehender: A Simulation Program and the Theory of Language**, *Communications of the ACM,* Vol. 12, 1969, pp. 459-476.

Samet, H. The Quadtree and Related Hierarchical Data Structures" *ACM Computing Surveys,* Vol. 16, No. 2, June. 1984.