Hand-Printed Character Recognition System Using Artificial Neural Networks

Adnan Amin and W. H. Wilson

School of Computer Science & Engineering, University of New South Wales
P.O. Box 1, Kensington, NSW 2033, Australia

ABSTRACT: This paper proposes a new technique for the recognition of hand-printed Latin characters using Artificial Neural Networks together with conventional techniques. One advantage of this technique is that it combines rule-based (structural) and classification tests. Another advantage is that it is more efficient for large and complex sets. The technique can be divided into five major steps: (1) digitization of the image; (2) thinning of the binary image using a parallel thinning algorithm; (3) tracing of the skeleton of the image and construction of a binary tree; (4) extraction of features from the structural information; and (5) classification of the segmented descriptions as particular characters by a feedforward neural network trained by backpropagation.

Keywords: Latin characters, Parallel thinning, Feature extraction, Structural classification, Neural Networks.

1. Introduction

Autonomous recognition of printed or written text, was one of the first goals of early research into pattern recognition. To date few products are commercially available to perform such tasks. Furthermore, their abilities are limited to recognition of typed text in a restricted number of fonts. Products to perform hand-printed text recognition are not available, although many architectures have been proposed. Indeed there has recently been much interest in applying neural networks to solve the problem [Hinton 91, Le Cun 91, Fang 90].

It is well known that the computer recognition of hand-printed text is a difficult issue because of the writing style and the shapes vary enormously. The recognition of hand-printed text is widely applicable in areas such as postal code recognition, for office automation; and for a large variety of other business and scientific applications [Amin 84, Wang 81].

Most of the development in neural networks research during the past decade makes use either Pattern matching or statistical approach for features extraction. Pattern matching is useful for printed character sets but will not work for hand-printed character sets. A number of workers have recently applied neural network techniques to hand-printed character recognition [Lisboa 91, Burr 88] by using a statistical approach. However, these have generally been confined to relatively small character sets, usually digits only. This approach is usually more expensive and has not proven very efficient for large and complex sets. In the project described in this paper, neural network methodology has been applied only in the final classification step; the computation-intensive earlier stages being done by more classical approaches. Thus this project takes a hybrid approach to character recognition.

2. Digitization

A considerable amount of work has been undertaken which is concerned with digitization, or binarization [Otsu 79, Weszka 79]. In contrast to such work, however, the algorithm adopted in this paper is similar to that which appears in [Amin 89, Amin 92]. A 300 dpi scanner is used to digitize the image.

3. Preprocessing

The preprocessing step involves operations on the digitized image intended to reduce noise and increase the ease of extracting structural features, by cleaning and thinning the image.

3.1. Pre-Thinning

The aim of this step is to minimize the noise in the image due to the shading effect or unevenness of the grey scale. Each pixel, P, is surrounded by eight pixels numbered P0 to P7 in a clockwise fashion starting from the north neighbor; see diagram below:
Let \( B(P) = P_0 + P_2 + P_4 + P_6 \). Any black pixel \( P \) with \( B(P) < 2 \) is to be whitened, any white pixel \( P \) with \( B(P) > 2 \) is to be blackened, and \( P \) is unchanged if \( B(P) = 2 \).

3.2. Thinning
There is a large volume of literature on thinning algorithms of both the sequential and parallel nature. Parallel approaches allow a thinning of the entire image simultaneously while sequential algorithms use a process on the pixels which are dependent on previous operations. The algorithm implemented in this system is a parallel method based on the Guo and Hall [Guo 89] A1 method.

4. Construction of Binary Tree
This step builds, from the pre-processed image, a binary tree which contains information describing the structure of the image.

4.1. Tracing
An algorithm implementing a 3x3 window is used to trace along the path of the skeleton, recording the structural information of the traced path. A path is described as a tracing between junction or end points, where an end point has a single neighbor and a junction point has 2 neighbors.

This path is stored in a node of the binary tree, where a choice of path to trace exists, a left and right node are formed beneath the current one and their respective paths traced out. A priority system is used which favours certain directions over others (without this, the window would trace the skeleton in random direction). It is obtained by experience and experimental results.

The starting point for tracing the skeleton is based on several criteria. The image is divided into 3 horizontal regions and the top and bottom region are searched for end points or junction points. This ensures that the starting point does not split a path into two subpaths. If no such points are found, as with the letter "o", the left most pixel of the image is used as starting point.

This path is stored in a node of the binary tree, where a choice of paths to trace exists, a left and right node are formed beneath the current one and their respective paths are traced out.

4.2 Structural Information
The structural information for each path traced is saved as follows:

1. Freeman code chain: an 8-directional code [Free 68] describing the tracing of the path.
2. Frame: co-ordinates describing a minimal-size frame which contains the path.
   - used to calculate size of loops in the image (e.g. 'a' = small, 'D' = large) as well as approximate centre.
3. Positional: co-ordinates describing the start and ending points of the path.
   - used to determine positional relationship between loops and touching paths (e.g. 'b' = left touching path, 'd' = right touching path).
4. Loop: pointer indicating paths rejoining previously explored section (e.g. 'e', 'G').
5. Points: location of points in the image, defined as isolated paths of very short length (e.g. 'j', 'i')

The completed tracing results in the segmentation of the character into paths or strokes which will latter be formed into primitives.

4.3. Smoothing
Upon completion of the binary tree, a smoothing step allows redundancies and noise to be removed from the tree, a smoothing of the binary tree is designed to minimize the number of nodes in the tree and minimize the Freeman code chain. Loops whose paths contain multiple nodes are identified and then compressed to single node. Redundancies in the Freeman chain code are smoothed and noise is reduced. At points of change in the Freeman code, a vector-averaging algorithm produces results as shown in the table below. The result is a minimal smoothing function which produces segment-like code with a minimum unit length of 3 and eliminates paths of length 1.

5. Feature Extraction
The structural information in the binary tree allows the formation of pattern primitives, or sub-patterns, which are used to describe the original image. There are two main primitives described in this system: straight lines and curves. A path may be described by a single primitive or by multiple primitives. The structural information in the tree is converted to these primitives using the following definitions.

Breakpoint (Separator): divides a path into sub-paths more easily described by primitives. A breakpoint has at least one of two possible conditions:
   - inflection point: a change in curvature, a posi-
tive (clockwise) curve followed by a negative (anti-clockwise) curve, or vice versa.
* cusp point: a sharp change in direction, two segments form an acute angle <= 90.

Straight Line: has its usual geometric definition as two points in sequence within a path. A point in a Freeman chain can be defined as a change in the Freeman code. Lines can be distinguished from curves in two ways: the length of a line segment is significant in comparison to the length of the path, or the path contains only two points.
Curve: these are formed by at least three segments of nearly equal length (usually small in relation to the length of the path) with no breakpoint. Two types of curve exist: Open and Closed.
* Open: there are only 4 open curves useful in describing Latin characters. They face the 4 main points of the compass, e.g. U, S, C.
* Closed (Loop): described by three sizes (small, medium, and large) and also includes the double loop, e.g. a, R, D, 8.

6. Character Classification
The characters have now been represented as a collection of segments, together with information about how the segments are related (do they touch? how are the segments laid out spatially?)
The remaining problem is to classify these character representations. Handwritten characters will vary in their representations: as an example, different 6's might be represented as a (closed) loop plus a curve (as in the printed version shown here), a loop with a sloping straight line at the top, or a loop with a vertical line at the left side (rather like a sanserif "b").

Rather than try to list all the possible variants, a reasonable approach is to take a large sample of actual handwritten characters (with their "correct" classifications) and apply a machine-learning technique to devise a classification algorithm. Such methods include symbol-processing-style algorithms such as the ID3 and C4.5 systems [Quinlan 1986], and backpropagation neural networks [Rumelhart 86]. As we are familiar with neural networks, we use this method.

We use a 3-layer network with one output unit for each possible character, a hidden layer, and inputs of a relatively complex nature, described below and in Figures 2-5. The overall network is shown in Figure 1.

Each component of the segmented representation is classified as a dot, line, curve, or loop. In each case, the characteristics of the component are determined: if a line, what is its orientation and its size relative to the character frame - short, medium or long. One input neuron is used to encode each of these possible choices (short/medium/long) and each of four possible orientations for a line. See Figure 2 for details.

**Figure 1:** Overall structure of neural network

<table>
<thead>
<tr>
<th>Object encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>null</td>
</tr>
<tr>
<td><strong>details</strong></td>
</tr>
</tbody>
</table>

**Details for four object types**

<table>
<thead>
<tr>
<th>DOT: NIL</th>
</tr>
</thead>
<tbody>
<tr>
<td>LINE</td>
</tr>
<tr>
<td><img src="image_url" alt="Image" /></td>
</tr>
</tbody>
</table>

**Figure 2:** Object encoding for input layer.

**Object inter-relationship matrix for ≤ 5 objects**

The (i, j) entry relates object i to object j. Each entry is a segment relative position record - see below.

**Segment relative position record**

![Image](image_url) (9 bits)

**Figure 3:** Representation of object interrelationships.

The relationships between the components are encoded in the object inter-relationship matrix (see Figure 3). This matrix records, for each pair of distinct components, what direction one has to go from the first to reach the second, and also whether the objects touch.
The overall design of the input layer is shown in Figure 4 for a network capable of handling characters with a maximum of 5 components. It uses a total of 150 input neurons: an \( n \)-object net would use \( 12n + 9n(n-1)/2 \) neurons. Figure 5 shows the representation of an (imaginary) character using this input layer design: black squares represent 1 bits, white squares represent 0 bits, and grey sections are not used in this example.

![Figure 4: Full representation of character using a maximum of 5 objects.](image)

When the network is trained, the output layer, in response to an input pattern with which it is familiar, or which resembles a familiar pattern, activates the neuron corresponding to this character classification.

7. Conclusion

The technique which has been adopted for the recognition of hand-printed Latin text using Neural Networks combines a purely structural method (based on structural primitives such as curves, straight lines, etc. in a similar manner to which human beings describe characters geometrically) and a classification test. In addition, this approach is efficient for feature extraction and recognition.

Algorithmic techniques work well for certain parts of the character recognition process. For the classification step, however, it is difficult to devise foolproof characterisations of each character to be recognized. While such characterizations might work for printed fonts, the large number of variants and incompleteness likely with handwritten characters means that a machine learning approach which infers rules, hopefully robust ones, from a wide sample of actual hand-written characters, offers a solution to this problem.

References


