Determining Character Confidences using Neural Network-based Recognition Techniques for Handwritten Word Segmentation

Michael Blumenstein & Brijesh Verma
Determining Character Confidences using Neural Network-based Recognition Techniques for Handwritten Word Segmentation

Michael Blumenstein\(^1\) and Brijesh Verma\(^2\)

Abstract

Character confidence calculation can provide useful information for segmentation-based handwritten word recognition systems. This research describes neural network-based techniques for segmented character recognition that are applied to the segmentation and recognition components of an off-line handwritten word recognition system. Two neural architectures along with three different feature extraction techniques were investigated. A novel technique for character feature extraction is discussed and compared with others in the literature. Recognition results above 70% are reported using characters automatically segmented from the CEDAR benchmark database.

Keywords: Handwriting Recognition, Cursive Word Segmentation, Neural Networks, Character Recognition, Character Confidences

---

\(^1\) Email: M.Blumenstein@mailbox.gu.edu.au
\(^2\) Email: B.Verma@mailbox.gu.edu.au
1 Introduction

The literature is replete with high accuracy recognition systems for separated handwritten numerals and characters [1]-[3]. However, research into the recognition of characters extracted from cursive and touching handwriting has not had the same measure of success [4]-[6]. One of the main problems faced when dealing with segmented, handwritten character recognition is the ambiguity and illegibility of the characters. Although a difficult problem, the accurate recognition of segmented characters is important in the context of segmentation-based, word recognition.

Traditionally, character confidences have been used in conjunction with dynamic-programming matching-based approaches for handwriting recognition [5]. These approaches calculate the best path through a graph of primitives and primitive unions based on the confidence values generated through the recognition of segmented character patterns. In this research, various neural architectures for character recognition are investigated and applied to confidence calculation in the context of handwritten word segmentation. Three feature extraction techniques were tested including a novel technique that extracts features based on the angle of line segments within a character image.

The remainder of this paper is broken down into 5 sections. Section 2 discusses previous and related work, Section 3 describes the neural-based character classifiers, Section 4 provides experimental results, a discussion of the results takes place in Section 5, and finally Section 6 presents future research and conclusions.

2 Previous Work and Related Research

In an earlier paper [7] a successful technique for over-segmenting hand-printed/cursive handwritten words to assist in the overall process of word recognition was proposed. A further technique for decreasing the number of erroneous segmentation points in over-segmented handwriting called Segmentation Point Validation (SPV) was also discussed. It was posited that an accurate technique of this sort could be invaluable for use with a lexicon-based dynamic matching procedure or some other word recognition process.

Through experimentation it was found that SPV was quite accurate on its own. It removed a high proportion of incorrect segmentation points. The main problem that presented itself was the removal of correct segmentation points. Whereas incorrect segmentation points may always be removed through further post-processing (lexicon-based matching and the union of primitives), it is very difficult to recover from a loss of correct segmentation points. The problem that remained was to develop a technique that could be used to generate all possible correct segmentation points in a word whilst keeping the number of incorrect points to a minimum. The traditional response in the literature seems to suggest the use of tightly coupled segmentation-recognition systems to increase segmentation and hence recognition accuracy [8]. The technique proposed in a later paper [9] went along these lines by examining information outside of the local areas investigated thus far and attempted to include input from a recognition component to strengthen segmentation decisions.

In the aforementioned paper, the "recognition component" proposed for handwritten word segmentation involved three neural experts. The first was the SPV network as discussed above and the second two experts were the neural classifiers proposed for segmented character recognition (upper case and lower case). On their own, these three experts seemingly served a different purpose, however following confidence fusion, it was found that they were a powerful tool for validation of segmentation zones and even for final word recognition. This paper will discuss in detail the investigation and experimentation of the two
neural character classifiers that assisted in providing high segmentation decisions for handwritten word recognition.

3 Segmented Character Recognition

Our segmentation technique for handwritten words [9] relies on character confidence assignment/classification. A number of techniques were investigated and tested on real-world handwritten characters extracted from words located on the CEDAR CD-ROM. Two neural classifiers were developed for experimentation and are described in this section.

3.1 Character Extraction

Prior to detailing the techniques used for feature extraction, it may be necessary to describe the methods employed for extracting character matrices from handwritten words and the preprocessing of these characters. Character Matrix Extraction refers to the process of locating and storing an area in a given word image delineated by a specified boundary. The extracted area, composed of a matrix of background and foreground pixels, represents the proposed character image. A short description of this extraction process is detailed in this section.

The boundaries for character extraction were defined in terms of x-coordinates generated by our heuristic segmenter [7] in conjunction with boundary information. To summarise the character extraction process, our technique first proceeds to sequentially locate all non-cursive/printed character components through the use of character component analysis. Finally, x-coordinates (vertical segmentations) for each connected character component (defined by the heuristic segmenter) were used to define the vertical boundaries of each character matrix. To locate the horizontal boundaries (top and bottom of the character matrix), the area bounded vertically (via x-coordinates or the boundaries found as a result of connected component analysis), is examined from the top and bottom. The first instances of foreground pixels located by searching from the top or bottom are deemed as the top-most and bottom-most y-coordinates for the character matrix respectively.

The character sets used for training and testing the chosen classifiers were extracted from words in the training and test directories of the CEDAR CD-ROM. As the words had already been globally preprocessed for slant, noise etc., it was not deemed necessary to further preprocess the individual extracted characters. However, due to the nature of the classifiers used (neural networks) one type of preprocessing was essential: re-scaling. We were required to create input vectors of uniform size and hence it was decided that for our earlier experiments, it was necessary to employ nearest-neighbour interpolation for this task. This type of re-scaling was mainly used in conjunction with the density feature extraction technique (to be discussed below). However, for the remaining two techniques, no re-scaling was performed. Instead, a form of local averaging was performed on the extracted feature vectors.

3.2 Feature Extraction

The literature describes a inexhaustible number of feature extraction techniques for character recognition. They may be loosely categorised into global and local (topological) feature extraction methods. Global methods are very simple to implement and model the global characteristics of a character. One of its main advantages is that it ignores local noise or distortions in the character image. Conversely, topological feature extraction techniques examine the geometry and topology of the character e.g. stroke direction, convexities, junction points etc. In fact, topological feature extraction techniques have proven to be the most popular amongst researchers for handwritten character recognition. The sections below
describe three feature extraction techniques that were investigated in this research. Specifically, the third technique has been investigated here for the first time. The first is a global technique whereas the second and third are based on the extraction of topological features.

3.2.1 Density features

The first feature extraction technique that was tested was similar to that described in [11]. In this technique, the extracted character image (raw or re-scaled), is segmented into windows of equal size. Density values (Number of foreground pixels divided by the total number of pixels) are obtained for each window. All density values are used to form the input feature vector for a particular character pattern. There were two main sets of experiments conducted using density feature extraction. The first used a re-scaling technique to initially create a character matrix of uniform size prior to feature extraction. The second set does not employ re-scaling but instead performs density feature extraction on the original character image. As mentioned previously, in either case, the result must be a set of vectors with uniform inputs to the neural network classifier. Density feature extraction based on the raw character image is discussed in detail here.

Upon character extraction, the matrices vary in dimension. Therefore, to create density features of uniform size, the original character dimensions (number of columns and number of rows) are each divided by the desired dimensions of the extracted feature vector. The final dimensions for the feature vector were experimentally chosen to be of equal value i.e. 10×10. If the number of original columns and rows could not be equally divided by the Final Dimensions (FDs), then the remainder for each was calculated and stored. To ensure that the character matrix was equally divisible by the FDs, the original matrix was padded with zeros on the bottom and on the right. The remainder values found in the previous calculation were used to decide the number of columns/rows to use for padding. Following padding, new values for the dimensions of the matrix were recorded. Finally, these newly calculated dimensions (number of rows and columns) were each divided by the FDs to determine the window size that could be used for density feature calculation. Therefore, through this process, each character matrix would have different window dimensions computed to compensate for the difference in character size and to allow the output of uniform feature vectors. Density values for each window were subsequently calculated as described above.

3.2.2 Transition features

The second feature extraction technique employed was based on the calculation and location of transition features from background to foreground pixels in the vertical and horizontal directions. A number of researchers have proposed feature extraction techniques based on transition information, examples may be found here [12],[5].

3.2.3 Stroke Direction (Angle) Features

In this section, a novel feature extraction approach, based on the direction of stroke components in a character image, is presented. It is a simple technique that employs zoning to partition the image into possible stroke segments. It then makes use of each stroke segment’s centroids to identify the angles at which the stroke components are directed. In this technique the ‘zones’ are represented by windows that have had their dimensions calculated based on the size of the extracted character matrix (as per density features). With regards to preprocessing, there is no thinning or contour extraction required in this technique. The directional values are obtained from strokes in the image regardless of width.

After the size of each window is defined, the pixels in each window are analysed. The operations that are applied to each window are identical. Therefore, an overview of the
feature extraction algorithm is presented in terms of features extracted from a single zone or window. Each operation on a single window is described in detail below.

**Calculation of centroids**

Each window examined is divided into two equal, upper and lower zones. It is assumed that each half contains a distribution of foreground pixels representing a fragment of a character stroke. The aim is to first locate the centroids (also called centres of gravity) of each half, and then to join each half with a line going through each centroid location (a centroid is represented by x and y coordinates). The equations below calculate the x and y centroid coordinates for a given window half:

\[
X_{Centroid} = \frac{\sum_{j=0}^{NR-1} \sum_{i=0}^{NC-1} x_{ij} \times i}{nfp} \quad (1)
\]

\[
Y_{Centroid} = \frac{\sum_{j=0}^{NR-1} \sum_{i=0}^{NC-1} x_{ij} \times j}{nfp} \quad (2)
\]

where,
- \( NC \) - Number of Columns is the width of the window
- \( NR \) - Number of Rows is the height of the window
- \( x \) - \{0,1\} Indicates the value of the current pixel being examined (background/foreground)
- \( y \) - \{0,1\} Indicates the value of the current pixel being examined (background/foreground)
- \( nfp \) - Number of foreground pixels

Once each centroid has been calculated, the coordinates \((x\_centroid, y\_centroid)\) denote the centre of the stroke fragment being currently examined. Therefore, the calculation of x and y centroid coordinates should be repeated for the upper and lower halves of the window being examined. This may be denoted below:

\[(UxC, UyC); (LxC, LyC)\]  \quad (3)

where,
- \( UxC \) - Upper x-coordinate Centroid,
- \( UyC \) - Upper y-coordinate Centroid,
- \( LxC \) - Lower x-coordinate Centroid,
- \( LyC \) - Lower y-coordinate Centroid.

The above pertains to the ideal situation that both halves of the window contain sufficient foreground pixels. However, if the number of foreground pixels is insufficient to calculate centroids in either of the halves, then it is necessary to look for x and y centroid coordinates in the left and right halves of the window instead (see below).

**Angle calculation**

Once the x and y centroid coordinates have been successfully located in each window half, it is possible to hypothesise a line that joins the two sets of coordinates. The angle that the line makes with the x-axis is then returned. To calculate the angle \( \theta \) (in degrees), the following equation is employed:
The equations above attempt to assign values between 0 and 360° denoting a character stroke’s angle. As defined above, if the upper x centroid value is greater than the lower x centroid value, the inverse "tan" function computes a positive angle in the range of 0 and 180°. Conversely, if the upper x centroid is smaller, the computed result from the inverse "tan" function is set to its negative equivalent and a value of positive 360° is added to it. This produces an angle value in the range of 180 to 360°. The latter rule was devised to increase the range of angles that could be calculated. Therefore angles calculated in the range of 0 and 180° were obtained when upper x centroid > lower x centroid because this denoted a hypothetical line pointing upwards. Whereas when upper x centroid < lower x centroid, this denoted that the line was pointing downwards hence an angle in the range of 180 to 360° would be the computed result.

**Detection of vertical or horizontal stroke segments**

In certain cases, an angle of 0° may be returned if the centroids computed indicate a horizontal or vertical line. This result may also be encountered if the region being examined is almost completely filled with foreground pixels and hence represents neither of the above. Therefore additional rules were added to identify these special cases. A measurement of the foreground pixels in the x and y directions of the window, provided information regarding the orientation of the line. Therefore, if the measurement of foreground pixels in the x direction proved to be larger than the measurement in the y direction, it was concluded that the pixels formed a horizontal line and were assigned a value of 90°. Conversely, if the y-direction measurement was greater, then a value of 180° was assigned instead. If the measurements were equal, a value of 0° was assigned.

**Centroid calculation for right and left window halves**

One further eventuality existed when calculating the x and y centroid coordinates as specified above. It was possible that the stroke being examined did not appear at all in either the top or bottom half of a particular window. In this case, one pair of (x,y) centroid coordinates would not be obtained and therefore no hypothetical line could be measured. To overcome this problem, one final check was performed. In the eventuality that the stroke was hidden only in the bottom or top half of the window, an examination of the right and left hand sides of the window would follow. This examination would therefore find any near horizontal strokes that were not found in the first steps of the examination. The x and y centroid coordinates would therefore be calculated for each half of the window and the hypothetical line (representing a stroke) would again be measured so that an angle could be computed and used as a possible feature.

To finally obtain the feature value for the window, the angle was divided by 360 to obtain a decimal value between 0 and 1. Each window produced a similar value and all were assembled into a feature vector that could serve as input to a classifier. Figure 1 below illustrates and summarises the process of angle calculation and obtaining a feature value for a particular window in a character image.
3.3 Configuration of the neural classifiers

In the sections above, three feature extraction techniques were detailed for the purpose of providing meaningful local/global character features that would be useful as inputs to a classifier. The classifiers chosen for this task was a feed-forward neural network. For experimentation purposes, the architectures were modified varying the number of inputs, outputs, hidden units, hidden layers and the various learning terms.

The number of inputs to each network was associated with the size of the feature vector for each image. From the very beginning it was decided that the vector size be kept as small as possible. One effect that a large input vector may have on neural network training is that learning can become quite slow (this is especially true if there are thousands of input vectors). Therefore various small vector dimensions were investigated for experimentation. The most successful vector size was of size 100.

The number of neural network outputs was altered quite often throughout experimentation. This seems an unlikely parameter to alter considering that the number of character classes (a-z, A-Z) was always constant. However, a number of strategies for varying the number of outputs were attempted to achieve the highest classification rates. In preliminary experiments for character recognition [10] (mainly using the pixel density feature), the number of outputs was set to 52. In this particular case the outputs represented all 26 lower case characters as well as 26 upper case characters. Therefore the network was presented with feature vectors representing lower case and upper case characters and it attempted to output a character class confidence based on 52 classes. In later experiments, two neural networks were trained with 26 outputs each. Therefore one neural network was trained with upper case characters and the other with lower case characters. Further evolution of the network architecture saw the inclusion of "reject neurons" to deal with primitives and sub-characters.

Originally each network was fitted with three reject neurons. The first reject category was exclusively implemented to identify character patterns that were smaller than approximately half a regular character i.e. a very small character fragment. The second category identified characters that had been split approximately in half. While the final reject category was used to identify character components that had not been correctly split i.e. multiple characters. As might be expected, the classification of what constitutes a quarter or half of a character
pattern is quite subjective. Therefore, to label each character class, a human operator was required to distinguish between the 52 character classes as well as the six types of reject patterns (3 per network). The procedure of labelling each input vector with a desired target vector was necessary for the supervised learning algorithms used for neural network training.

After conducting experiments with two neural networks and three reject outputs per network, another alternative was to limit each network to only having one reject neuron. The sole reject neuron was trained to output a confidence for any primitives/character components that were not an entire character. Therefore this included small character components i.e. 1/4 character and 1/2 character components as well as components that contained multiple characters (more than one character due to poor segmentation).

Two learning algorithms were used. The first was the popular backpropagation algorithm with momentum and the second was a standard Radial Basis Function (RBF) network. When using the backpropagation algorithm, the number of hidden units could be varied to obtain the most favourable results. Similarly, when using the RBF network, the number of centres were varied to provide the best results. When using the backpropagation algorithm, experiments were conducted using one and two hidden layers. The learning rate (\( \eta \)) and the momentum term (\( \alpha \)) were also varied in the range of 0 and 1 to produce the best results. Finally, the number of iterations was also varied to discover the best weight configuration for the backpropagation-based network.

### 3.4 Preparation of training data for the neural classifiers

The character training and test sets for the various neural networks were obtained from training and test words on the CEDAR CD-ROM. As pointed out earlier, both the neural classifiers used were trained with supervised algorithms. This meant that for all the different neural configurations (especially when modifying the number of output neurons), it was necessary to adjust the training data. For example, when 52 outputs were used in the neural configuration, it was necessary to include training samples for each type of character (a-z, A-Z). The training/test files needed to be manually prepared, however character matrix boundaries were determined based on the output of our heuristic segmenter. Each extracted character was viewed by a human operator and was labelled manually as belonging to a particular character class. This was a difficult process as certain characters were very ambiguous and in some cases it was necessary to verify the context from which the character was extracted. In the case where three reject neurons were used, the human operator was instructed to subjectively decide what constituted a 1/4 of a character, half a character and a multiple character component as well as deciding the class that each character belonged to.

The process of labelling a particular character or sub-character pattern was automated as much as possible with the use of a software application. The human operator was able to indicate a character class at the press of two keys. For example, to indicate that a particular character being viewed on screen was a lower case "c", the operator was instructed to simply press the "c" key on the keyboard followed by the "enter" key. For capitals, the user would type a capital letter in the normal fashion and press "enter". For the various sub-character and multi-character components, a number of symbols were assigned for each eventuality. For example, an upper case, multi-character component was indicated by a "@" and lower case 1/4 and 1/2 character components were indicated by a "(" and a ")" respectively. Upon pressing a key and thereby making a decision regarding a particular character class, the computer application would automatically send the binary character matrix to a file and accompany it with the relevant desired output vector. This process was executed for each character/sub/multi-character component. Once completed, the raw characters in the file could be converted to their equivalent feature representations and once again paired with their corresponding desired outputs.
4 Experimental Results

For experimentation of the character recognition techniques detailed in Section 3, we used handwritten words from the CEDAR benchmark database [13]. In particular we used word samples contained in the “BD/cities” directory of the CD-ROM.

Characters were obtained by automatically segmenting each word using the heuristic algorithm and extracting characters bounded by the determined segmentation points (refer to Section 3 for details). Characters for testing were extracted in much the same way, except that SPV was conducted following heuristic segmentation.

In preliminary experiments [10], a backpropagation neural network with a 52 output architecture was employed. For these experiments, a re-scaling procedure was employed along with the pixel density feature extraction technique. This pilot study yielded top recognition rates of 56.19% on the test set. These were considered quite low, and therefore a different strategy was posited whereby two neural networks would be trained (one for lower case characters and the second for upper case ones).

The results in this research are displayed in tabular form for each set of experiments. Table 1 presents top results using a three reject neuron architecture for three feature extraction techniques. Separate experiments were conducted for lower case and upper case character patterns. A total of 18655 lower case and 7175 upper case character patterns were generated for training. A further 2240 lower case and 939 upper case patterns were used for testing.

Table 2 similarly presents results for the single reject neuron architecture. The number of lower case and upper case training patterns used was 13123 and 4825 respectively. The number of testing patterns used for lower case and upper case characters remained at 2240 and 939 respectively.

Finally, Table 3 presents results employing the Radial Basis Function network employing each of the feature extraction techniques discussed previously. These experiments were conducted solely using the single reject neuron architecture.

In the case of the backpropagation neural networks, many experiments were performed varying settings such as the number of iterations, the number of hidden units, learning rate and momentum. For the Radial Basis Function architecture, the number of centres was adjusted to provide the best results. The tables below show the top results in each case.

### Table 1. Character recognition rates using the backpropagation algorithm and a three reject neuron architecture

<table>
<thead>
<tr>
<th>Density</th>
<th>Transition</th>
<th>Angle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase</td>
<td>56.07</td>
<td>60.63</td>
</tr>
<tr>
<td>Uppercase</td>
<td>60.92</td>
<td>67.94</td>
</tr>
</tbody>
</table>

### Table 2. Character recognition rates using the backpropagation algorithm and a single reject neuron architecture

<table>
<thead>
<tr>
<th>Density</th>
<th>Transition</th>
<th>Angle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowercase</td>
<td>59.60</td>
<td>63.13</td>
</tr>
<tr>
<td>Uppercase</td>
<td>62.83</td>
<td>69.33</td>
</tr>
</tbody>
</table>
Table 3. Character recognition rates using an RBF network and a single reject neuron architecture

<table>
<thead>
<tr>
<th></th>
<th>Lowercase</th>
<th>Uppercase</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Density</strong></td>
<td>60.89</td>
<td>67.20</td>
</tr>
<tr>
<td><strong>Transition</strong></td>
<td>64.29</td>
<td>71.46</td>
</tr>
<tr>
<td><strong>Angle</strong></td>
<td>55.67</td>
<td>60.38</td>
</tr>
</tbody>
</table>

5 Discussion of Results

5.1 The effect of reject neurons and decreased vector size

A major difference in strategy that was undertaken in this research which contrasted with our previous research [10] was that of neural architecture. Previously, one network was employed to classify 52 character patterns (upper and lower case). Results were not entirely satisfactory and hence a strategy of two networks to deal with both cases of letters was investigated. It was found that by splitting the character patterns into these two separate classes, it eliminated the problems of ambiguity faced when similar upper/lower case characters were presented to a single network. As an example, frequent ambiguity resulted when the neural network was presented with a "c" and a "C". Also, the number of classes that the single network needed to learn was reduced and hence increased accuracy and training time.

Another observation that was made whilst conducting our investigation was the influence on character vector size on the overall recognition rate. Pilot experiments saw the use of the pixel density feature extraction technique whereby a training vector size of $15 \times 15$ (225 inputs) was produced. The number of outputs for that neural network was set to 29 ($a$-$z$ and 3 reject neurons). For the settings detailed above, the lower case network produced results of up to 51% for the test set. The aforementioned results were not considered satisfactory and an adjustment to the input vector size was considered to possibly increase accuracy. All settings were identical to those detailed immediately above, except that the feature vectors were set to a smaller size: $10 \times 10$ (100 inputs). As may be seen in Table 1, the lower case character recognition rate was boosted by 5% when the number of inputs to the network was decreased.

5.2 Three reject neurons versus one

It may be observed that the networks employing a single reject neuron have produced better results on average than those employing three reject. It may therefore be posited, that it was more difficult for the network containing three reject neurons to differentiate between the various reject classes than it was for the single reject network to learn all three types of reject patterns. It may be noted that the difference in performance between the two types of networks (three reject/single reject) is at most 3%.

5.3 Backpropagation and RBF networks

All experiments using RBF networks were conducted with a single reject neuron architecture. The reason for this was that the networks containing a single reject neuron provided the highest recognition accuracy when using the backpropagation algorithm. It was therefore decided to focus on RBF training solely using the most successful architecture. As may be seen from Tables 2 & 3, the results for all feature types using an RBF network were superior to those that used a backpropagation network. This may be attributed to the fact that the Gaussian function in the hidden layer of the RBF network was more successful at distinguishing between character and non-character (garbage) patterns than was the backpropagation network.
5.4 Comparison of character recognition results with other researchers in the literature

It is always a difficult task to compare results for handwritten character recognition with other researchers in the literature. The main problems that arise are differences in experimental methodology, different experimental settings and difference in the handwriting database used. The comparisons presented below have been chosen for two main reasons. The handwriting database (CEDAR CD-ROM) used by the researchers is identical to the one used in this research and the results are some of the most recent in the literature.

Yamada and Nakano [3] presented a handwritten word recognition system that included a character recogniser. Their classifier was trained on segmented characters from the CEDAR benchmark database. The classifier was trained to output one of 52 classes (a-z, A-Z). They recorded recognition rates of 67.8% and 75.7% for the recognition of characters where upper case letters and lower case letters were distinguished and not distinguished respectively. In other words the first recognition rate refers to when an "a" is distinguished from an "A" (case sensitive) and the second recognition rate relates to the two types of characters not being distinguished (non-case sensitive). The difficulty that may be observed is that for the results presented in this research, the best recognition rates were obtained when two separate neural networks were trained with upper case and lower case characters. However, if the top lower case (64.29%) and upper case (71.46%) character recognition scores in this research are averaged, a recognition accuracy of 67.88% is obtained. This recognition rate compares well with their results.

Another example where a 52-output classifier is used for segmented character recognition is in research presented by Kimura et al. [5]. They used neural and statistical classifiers to recognise segmented CEDAR characters. For case sensitive experiments, their neural classifier produced an accuracy of 73.25%, which was comparable to the lower case and upper case average of 67.88%.

Singh and Hewitt [14] employed the modified Hough Transform on characters from the CEDAR CD-ROM. They obtained a recognition rate of 67.3% using a Linear Discriminant Analysis-based classifier. Our best results compare favourably with their top recognition rate.

6 Conclusions and Future Research

This paper presented an investigation of character classification techniques that may be applied to character confidence calculation for handwritten word recognition. A stroke direction-based (angle) feature extraction technique for character recognition was compared to two others. Upon comparison, it was found that it did not produce the top recognition accuracy, however it was comparable to the other two techniques. The feature extraction techniques were tested on real world handwritten data that were used to train and test two neural classifiers. The results for handwritten character classification were favourable, especially considering the difficulty of the character data set. It was observed in practically all experiments, that the single reject neuron networks outperformed the three-reject neuron networks by up to 3%. In this research, the top character recognition rates were obtained employing RBF networks with a single reject neuron. Upon averaging the results from the upper case and lower case, single reject networks, a recognition accuracy of 67.88% was obtained. This compared favourably, with the results presented by other researchers in the field.

In future research a number of considerations will be addressed in an attempt to boost character recognition rates. Firstly, a better preprocessing methodology shall be used. It may include the introduction of a character slant correction module. This was not applied to the
characters in this research as it was assumed that slant correction applied to the entire word would be sufficient. Other strategies may include character smoothing prior to feature extraction.

With regards to training samples, a less automated approach to character image generation may be employed. This may assist in providing the most optimum test set recognition results. Also, a wider variety of global and local feature extraction techniques shall be tested on the character images. One particular strategy that may be attempted is that of proposing new procedures for combining the character classifiers i.e. character recognition with multiple experts. Many different strategies for expert combination have been applied to numeral recognition, and it is possible that their success can pass on to segmented characters. Finally, the examination of different classification techniques shall also be explored i.e. statistical classifiers and other neural-based classifiers.

References


