A Novel Approach For Structural Feature Extraction: Contour vs. Direction

Brijesh Verma
School of Computing and Information Systems
Central Queensland University
b.verma@cqu.edu.au

Michael Blumenstein
School of Information Technology
Griffith University
m.blumenstein@griffith.edu.au

Moumita Ghosh
School of Information Tech & Mathematical Sciences
University of Ballarat
m.ghosh@ballarat.edu.au

Abstract

The paper presents a novel approach for extracting structural features from segmented cursive handwriting. The proposed approach is based on the contour code and stroke direction. The contour code feature utilises the rate of change of slope along the contour profile in addition to other properties such as the ascender and descender count, start point and end point. The direction feature identifies individual line segments or strokes from the character's outer boundary or thinned representation and highlights each character's pertinent direction information. Each feature is investigated employing a benchmark database and the experimental results using the proposed contour code based structural feature are very promising. A comparative evaluation with the directional feature and existing transition feature is included.

1. Introduction

Interest in the recognition of off-line handwritten cursive words has been ongoing for over four decades. Off-line handwriting recognition refers to the problem of computer-based reading of handwritten characters or words that have been written on a common surface (i.e. paper). This is significantly different to on-line handwriting recognition whereby words are generally written on a pressure sensitive surface from which real time information, such as the order of strokes made by the writer, is obtained and preserved [1].

One of the first investigations in the literature that confronted the cursive word recognition problem was that of Frishkopf and Harmon [2]. In their research, the authors investigated a letter-by-letter or segmentation-based approach in addition to a holistic approach for reading cursive script. To this very day, the main approaches that exist for off-line cursive word recognition may be divided into segmentation-based and holistic ones. Generally, the former utilises a strategy based on the recognition of individual characters whereas the latter deals with the recognition of the word image as a whole [1]. Since Frishkopf and Harmon's seminal work, research into the recognition of cursive handwriting still continues to be intense. This continued motivation may be attributed in part to the challenging nature of the problem as well as the countless number of commercial areas that it may be applied to [3]. Applications that have received particular attention in recent times include: Postal Address Recognition [4][5], Bank Cheque Processing [6][7] and Forms Processing [8].

In the segmentation-based strategy for handwritten word recognition, the objective is to over-segment the word a sufficient number of times to ensure that all appropriate letter boundaries have been dissected. To determine the best segmentations, a set of hypotheses are tested by merging segments of the image and invoking a classifier to score the combinations. Most techniques employ an optimisation algorithm making use of some sort of lexicon-driven, dynamic programming technique and possibly incorporating contextual knowledge. The basic approach described above was proposed simultaneously by a number of researchers [5][9][10][11][12].

A crucial component of the segmentation-based strategy is the development of a classification system for scoring individual characters and character combinations. The literature is replete with high accuracy recognition systems for separated handwritten numerals [13][14][15], however the same measure of success has not been attained for segmented or cursive characters [11][16][17][18][19][20][21][22]. There are three main problems faced when dealing with segmented, handwritten character recognition. The first relates to the ambiguity of the character without the context of the entire word i.e. an 'l' may look very similar to an 'e'. The second problem relates to the illegibility of certain characters due to the nature of cursive writing i.e. ornamentation, distorted character shape etc. [23]. Finally, the process of segmentation may itself introduce some anomalies depending on the algorithm used. Certain algorithms may not locate the segmentation path or anchorage point accurately and may sometimes dissect adjacent character components [24].

In order to address the problems discussed above, researchers have explored two main approaches: 1) Determining features best suited for recognition and 2) investigation of different classification schemes [19]. Yamada and Nakano [16] investigated a standard technique for feature extraction based on direction histograms in character images. They used a multi-template strategy with clustering for the recognition of segmented characters from words in the CEDAR database [25]. Kimura *et al.* investigated a similar feature extraction technique calculating local histograms based on chain code information in segmented handwritten characters. They used statistical and neural classifiers for the recognition of segmented CEDAR characters. Gader *et al.* have proposed a feature extraction technique utilising transition information [11]. Their technique is based on the calculation and location of transitions from background to foreground pixels in the vertical and horizontal directions. The authors used neural networks trained with the backpropagation algorithm for recognising characters obtained from U.S. postal words. Other recent studies by Camastra and Vinciarelli [20][22] have proposed feature extraction techniques generating local and global features. The local features are obtained from sub-images of the character including foreground pixel density information and directional information. The global features used included the fraction of the character appearing below the word baseline and the character's width/height ratio. The authors used Learning Vector Quantization (LVQ) [20] and a combination of neural gas and LVQ classifiers [22] for the recognition of segmented (cursive) characters from the CEDAR database.

In this research, a novel approach is presented for extracting structural features from segmented handwritten characters. The approach is based on the character's contour code and stroke direction. The contour code feature is based on information extracted from each characters contour profile such as slope change, direction change, starting points and end points. In contrast, the direction feature obtains information based on the identification of individual line segments (or strokes) and distinguishes each segment in terms of its normalised direction. The approach is tested using a benchmark database of handwritten characters and the results are promising.

The remainder of this paper is broken down into 4 sections. Section 2 describes the handwriting recognition system along with the proposed feature extraction approach, Section 3 provides experimental results, a discussion of the results takes place in Section 4, and finally Section 5 presents conclusions and future research.

2. Overview of Handwriting Recognition System

The techniques described in this research form part of a larger handwriting recognition system as shown in Figure 1 below.

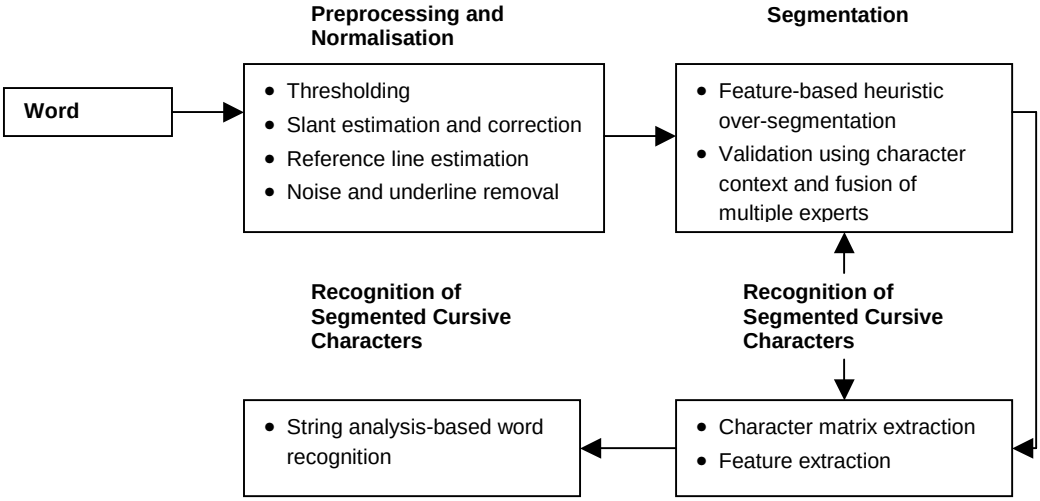

Figure 1 Overview of Handwriting Recognition System

The main components to be discussed in this section are 1) Preprocessing and Normalisation 2) Segmentation and 3) Recognition of Segmented Cursive Characters.

2.1 Preprocessing and Normalisation

The following sections describe the databases chosen and the operations performed on them to facilitate the recognition of cursive characters.

2.1.1 Cursive Character Datasets

The first character data set used for training and testing was extracted from words in the training and test directories (CITIES/BD) of the CEDAR CD-ROM [25]. These directories contain 3106 training words and 317 test words. 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. This will be referred to as the CEDAR Automatically Segmented (CAS) data set. The techniques employed for segmentation will be discussed in later sections. The second data set was comprised of pre-segmented, Binary Alphanumeric Characters (BAC) from the CEDAR CD-ROM found in the BINANUMS/BD & BL directories. The BAC dataset contained a total of 19145 characters for training and 2183 characters for testing.

2.1.2 Thresholding

The words that were obtained from the CEDAR database were in a grey-scale format. It is common for grey-level images to be thresholded prior to further processing in most handwriting recognition systems. Thresholding is advantageous as it is easier to manipulate images with only to levels of colour, processing is faster, less computationally expensive and allows for more compact storage. Otsu's method [26] was employed to threhsold the handwritten words used in this research. The characters in the BAC dataset were already in a binary format and did not require thresholding.

2.1.3 Slant estimation and correction

Arguably, slant estimation and correction may be one of the most important steps in word preprocessing and normalisation. It is helpful in transforming handwritten words into a

normalised form that more easily facilitates segmentation and the extraction of features. A large proportion of real-world handwritten words exhibit some sort of slant usually in the forward direction. The "cities" directory of the CEDAR database is no exception. The Slant estimation and correction strategy employed in this research is a slightly modified version of that described in Bozinovic and Srihari [27]. Through correction of all words in the CEDAR database it was found that the slant was adequately corrected in most cases.

2.1.4 Baseline computation

Baseline computation is another important technique in handwriting recognition. Baselines are used for size normalization, correcting rotation, extracting features etc. The baseline detection algorithm incorporated in this research is very simple and is similar to those employed by other researchers in the area. It is briefly described below and an illustration is shown in Figure 2:

Step 1: Calculate the horizontal histogram row by row.
Step 2: Calculate changes between the histogram and store the relevant values.
Step 3: Find the maximum value of the histogram and its corresponding row.
Step 4: Start from the row that has the maximum histogram value, look for the biggest change in the histogram density or until the bottom of the image is reached. The row that has the biggest change in histogram density will be the baseline.
Step 5: Similarly repeat for the upper baseline working towards the top of the image

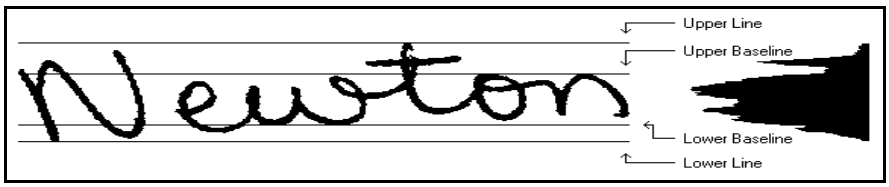

Figure 2 Reference line estimation based on horizontal density histogram

2.1.5 Noise and underline removal

A set of rules was implemented to detect and eliminate noise elements that were either prominent in the original word images or that were introduced at the thresholding stage. Two components were introduced for removing anomalies that were found to be prominent within the handwritten images.

The first was a technique to remove salt and pepper noise as well as small foreground objects such as interfering strokes that intercepted the word's bounding box. This was achieved by first detecting all the segregated components in the image through connected component analysis. The perimeters of contours belonging to each connected component were recorded, summed and averaged. Any connected component contours that were smaller than the calculated average were deemed to be either noise or irrelevant stroke fragments.

The second noise removal technique was a simple algorithm designed to remove underlines from the word images. It was noted that a small number of words in the database had underlines present and it was later found that segmentation and further processing was impeded due to their presence. This prompted the implementation of a simple underline removal algorithm. The algorithm searched the word images horizontally for the presence of underlines. Upon detection of a relatively long unbroken horizontal line of foreground pixels, a potential underline had been identified. Its length was recorded, and if its length exceeded that of a threshold that had been calculated earlier based on the size of the word, then it was marked as an underline and removed. This simple algorithm worked for some underlines but did not perform well on some of the more difficult, erratic and skewed underlines that were

present in some word images. The remainder of undetected underlines were removed manually to facilitate further processing.

2.2 Segmentation

This section describes the segmentation algorithm and sub-algorithms incorporated employed in the handwriting recognition system. An overview of is shown in the stepwise algorithm below and also in Figure 3.

Step 1. Locate features (such as upper and lower word contours, holes, upper and lower contour minima, vertical density histograms, etc.) and over-segment the word using the heuristic, feature-based segmenter.
Step 2. Obtain confidence values from character and Segmentation Point Validation (SPV) neural networks.
Step 3. Fuse all confidence values.

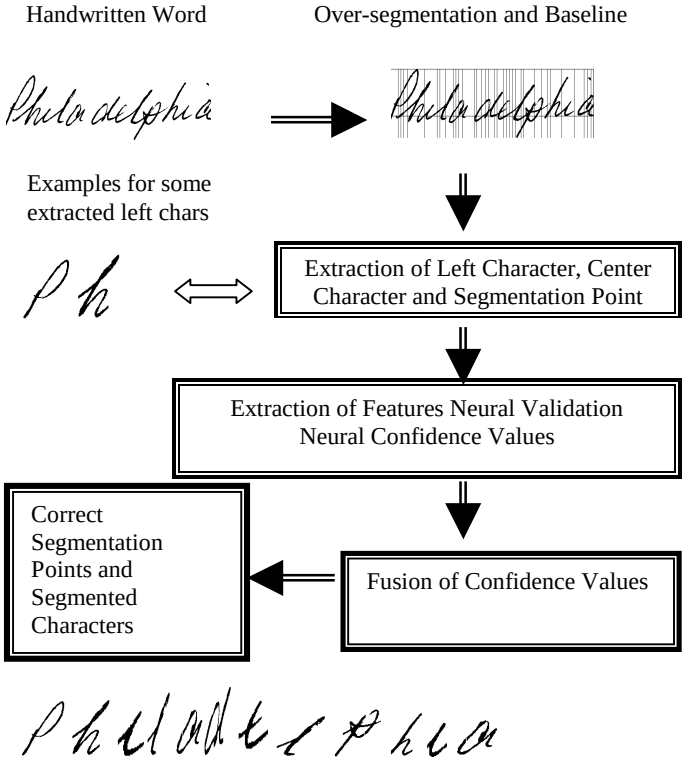

Figure 3 An Overview of the Segmentation Algorithm

2.2.1 Over-segmentation

To assign Prospective Segmentation Points (PSPs) that could be validated through further processing, a heuristic, over-segmentation algorithm was developed [28][29]. Initially, a number of preliminary processing steps were required prior to segmentation point assignment. Following these steps, an algorithm was devised to propose segmentation points based on various handwriting features such as upper and lower word contours, holes, upper and lower contour minima, vertical density histograms and confidence assignment.

2.2.1.1 Preliminary Processing

The first step of processing required connected components in the image to be determined. These components represented isolated characters or sections of joined or cursive handwriting. The larger components would be split into smaller components in further steps. Conversely, small connected components were deemed as being "noise" in the word image and were hence removed to facilitate the segmentation process.

The next steps of processing required calculation of certain measurements that would assist in locating segmentation points: 1) Stroke width estimation, 2) Word height estimation and 3) Average character width estimation. These measurements were invaluable for serving as threshold values when assigning segmentation points (refer to Section 2.2.1.2).

2.2.1.2 Segmentation Point Assignment

Segmentation points in each word were assigned based on confidence values derived from examining various features located in the word image. Confidence values for PSPs were increased if the particular area under examination contained minima in the upper and lower contour (i.e. possible ligatures in cursive writing) and areas in the word that exhibited a low vertical pixel density. Finally, a PSP's confidence was decreased if it was located in an area of the word that contained "holes" (i.e. found in such letters as "a" and "o"). The PSPs bearing the highest confidences were retained for further examination i.e. redundant segmentation point removal, and equally distributing segmentation points throughout the word. Hence the PSPs generated where used as input to the next phase of the entire segmentation procedure. The reader may refer to [30] for a detailed description of the over-segmentation algorithm.

2.2.2 Neural Confidences

Following heuristic segmentation it is necessary to discard "incorrect" segmentation points while preserving the "correct" points. This is achieved by calculating a number of confidences for each segmentation point generated by the heuristic segmenter. Three neural networks are used for this step. Firstly, a neural network is trained with features extracted from segmentation areas originally located by the heuristic algorithm. The neural network verifies whether each particular area is or is not characteristic of a segmentation point [28]. If an area is positively identified as a segmentation point, the network outputs a high confidence (> 0.5). Otherwise the network will output a confidence close to 0.1.

Two other neural networks trained with handwritten characters (upper case and lower case) are then used to confirm the first neural network's output. Each network is presented with areas immediately centred on/adjacent to each segmentation point. Area width is calculated based upon average character width. If for example, the area immediately to the left of the PSP proves to be a valid character, the network will output a high confidence (LCC) for that character class. At the same time, if the area immediately centred on the segmentation point provides a high confidence for the reject neuron (CCC), then it is likely that the PSP is a valid segmentation point. The "reject" output is specifically trained to recognise non-character patterns (ie. joined characters, half characters or unintelligible primitives). If this neuron gives a high confidence, this will usually indicate that the particular area being tested is a good candidate for a segmentation point. Otherwise, if any valid characters are given a high confidence (in the centre character area), it is unlikely that that particular area should be segmented.

2.3 Contour features for segmented character recognition

2.3.1 Contour extraction

This section describes the proposed methodology for the extracting the contour between the two segmentation points. The contour between two consecutive segmentation points is extracted using the following few steps. In the first step disconnect the pixels near the first segmentation point; disconnect the pixels near the second segmentation point. Find the nearest distance of the first black pixel from the first segmentation point and the three baselines. Follow the contour path across that baseline having minimum distance is closest. Find the connecting contour. Mark it as visited once it is visited. If the contour is already visited then discard that, take the other part if any.

2.3.2 Proposed contour code based structural feature

A novel feature extraction technique is proposed to extract the feature between the two contours. The values for feature extracted are structural feature from the contour profile. So the feature is named as contour code feature. The rate of change of slope along with the point where it is changing is extracted. With the contour slope, a few additional values that count number of ascenders, number of descenders, start point, end point, etc. are taken into consideration to capture the structural properties of the contour profile. The contour code feature vector size is taken as 25. The contour code feature is described below.

Slope: The slope of the consecutive points is estimated using linear regression. The rate of change of slope is used as the main feature. The feature is described below in Figure 4.

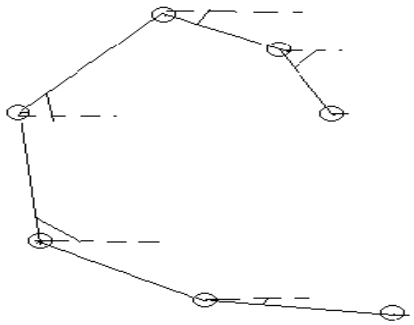

Figure 4 Contour code feature

The input to the contour code feature extraction module is the set of coordinate (x, y) of the contour extracted from the contour extraction phase. With the coordinate the slope of two consecutive points are estimated. The slope estimation is done using linear probing. Linear regression attempts to explain this relationship with a straight line fit to the data. The linear regression model postulates that

$$Y = a + bX \tag{1}$$

The coefficients *a* and *b* are determined by the following equations.

$$b = \frac{2\left(x_1 y_1 + x_2 y_2\right) - \left(x_1 + x_2\right)\left(y_1 + y_2\right)}{2\left(x_1^2 + x_2^2\right) - \left(x_1 + x_2\right)^2} \tag{2}$$

$$a = \frac{\left(y_1 + y_2\right) - b\left(x_1 + x_2\right)}{2} \tag{3}$$

The slope is between the two points (x1, y1) and (x2, y2) is represented by the parameter b. The following values are calculated and stored as a contour code in a single dimension vector.

Point of Change: The point with respect to the main body of the contour where the slope is changing is taken.

Direction Change (Up/Down): The point with respect to the main body of the contour where the direction is changing is taken. The change of direction is denoted by the contour, which is changing the direction upwards to downward or vice versa.

Number of Ascenders: The number of point above the upper baseline is counted and stored.

Number of Descender: The number of point below the lower baseline is counted and stored.

Start Point: Start point of a character (position with respect to baselines) is detected and stored.

End Point: End point of a character (position with respect to baselines) is detected and stored.

2.4. Proposed Direction based structural feature

The second technique (direction feature) sought to simplify each character's boundary or thinned representation through identification of individual stroke or line segments in the image. Next, in order to provide a normalized input vector to the neural network classification schemes, the new character representation was broken down into a number of windows of equal size (zoning) whereby the number, length and types of lines present in each window was determined. The line segments that would be determined in each character image were categorised into four types: 1) Vertical lines, 2) Horizontal lines, 3) Right diagonal and 4) Left diagonal. Aside from these four line representations, the technique also located intersection points between each type of line.

To facilitate the extraction of direction features, the following steps were required to prepare the character pattern:

1. Starting point and intersection point location
2. Distinguish individual line segments
3. Labeling line segment information
4. Line type normalization

2.4.1 Starting point and intersection point location

To locate the starting point of the character, the first black pixel in the lower left hand side of the image is found. The choice of this starting point is based on the fact that in cursive English handwriting, many characters begin in the lower, left hand side. Subsequently, intersection points between line segments are marked (these locations are a component of the final feature vector). Intersection points are determined as being those foreground pixels that have more than two foreground pixel neighbours.

2.4.2 Distinguish individual line segments

As mentioned earlier, four types of line segments were to be distinguished as compromising each character pattern. Following the step described in Section 2.3.1, neighbouring pixels along the thinned pattern/character boundary were followed from the starting point to known intersection points. Upon arrival at each subsequent intersection, the algorithm conducted a

search in a clockwise direction to determine the beginning and end of individual line segments. Hence, the commencement of a new line segment was located IF:

1. The previous direction was up-right or down-left AND the next direction is down-right or up-left OR
2. The previous direction is down-right or up-left AND the next direction is up-right or down-left OR
3. The direction of a line segment has been changed in more than three types of direction OR
4. The length of the previous direction type is greater than three pixels

The above rules were consulted for each set of pixels in the character pattern. The thresholds in rules 3 and 4 above were determined by manually inspecting a subset of the character database.

Firstly, it was noted that the object of “line type normalization” (Section 2.4.4.) was to normalize individual lines in a character pattern matching one of the four line types described earlier; specifically pixels which did not significantly deviate from one single direction type. Hence, it was posited that a line changing in more than three directions could not be called a single line.

Secondly, in rule 4 above, the threshold of three was chosen as it was found through visual inspection that a suitable “minimum line length” could be deemed as being one composed of at least four pixels all marked with the same direction value.

2.4.3 Labelling line segment information

Once an individual line segment is located, the black pixels along the length of this segment are coded with a direction number as follows: Vertical line segment - 2, Right diagonal line - 3, Horizontal line segment - 4 and Left diagonal line - 5. Figure 5 illustrates the process of marking individual line segments.

2.4.4 Line type normalization

Following the steps described in Sections 2.4.2 and 2.4.3, line segments were composed of marked direction pixels (Figure 5).

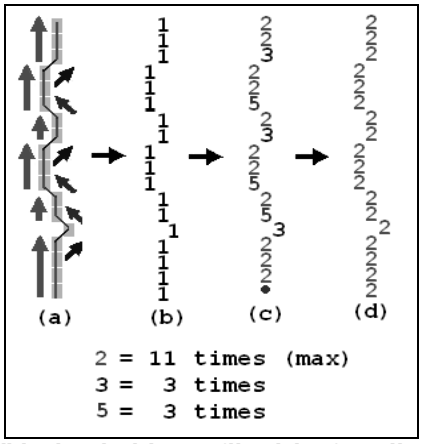

Figure 5 (a) Original line, (b) Line in binary file, (c) After distinguishing directions, (d) After direction normalization

As line segments were marked by following either the character boundary or thinned image pattern on a pixel-by-pixel basis, some spurious direction values may have been marked in any particular line segment due to the presence of anomalous pixels introduced through the thinning or boundary extraction process. Hence to "normalize" a particular line segment (discarding spurious direction values), the number of values belonging to each

direction type was tallied in a particular line. The direction value most frequently represented in a particular line segment was used to replace spurious direction values (Figure 5).

2.4.5 Formation of feature vectors through zoning

As neural classifiers require vectors of a uniform size for training, a methodology was developed for creating appropriate feature vectors. In the first step, the character pattern marked with direction information was zoned into windows of equal size (the window sizes were varied during experimentation). If the image matrix was not equally divisible, it was padded with extra background pixels along the length of its rows and columns. In the next step, direction information was extracted from each individual window. Specific information such as the line segment direction, length, intersection points, etc. were expressed as floating point values between –1 and 1.

The algorithm for extracting and storing line segment information first locates the starting point and any intersections in a particular window. It then proceeds to extract the number and lengths of line segments resulting in an input vector containing nine floating-point values. Each of the values comprising the input vector was defined as follows: **1.** The number of horizontal lines; **2.** The total length of horizontal lines; **3.** The number of right diagonal lines, **4.** The total length of right diagonal lines; **5.** The number of vertical lines; **6.** The total length of vertical lines; **7.** The number of left diagonal lines; **8.** The total length of left diagonal lines and **9.** The number of intersection points (Figure 6).

As an example, the first floating point value represents the number of horizontal lines in a particular window. During processing, the number starts from 1.0 to represent “no line” in the window. If the window contains a horizontal line, the input decreases by 0.2. The reason a value commencing at 1.0 and decreasing by 0.2 was chosen was mainly because in preliminary experiments, it was found that the average number of lines following a single direction in a particular window was 5. However in some cases, there were a small number of windows that contained more than 5 lines, and hence in these cases the input vector contained some negative values. Hence values that tallied the number of line types in a particular window were calculated as follows:

$$\text{value} = 1 - ((\text{number of lines}/10) \times 2) \qquad (4)$$

For each value that tallied the number of lines present in a particular window, a corresponding input value tallying the total length of the lines was also stored. To illustrate, the horizontal line length can be used as an example. The number starts at 0 to represent “no horizontal lines” in a particular window. If a window has a horizontal line, the input will increase by the length of the line divided by the maximum window length or window height, (depending on which one is the largest) multiplied by two. The reason this formula is used, is because it is assumed that the maximum length of one single line type is two times the largest window size. As an example, if the line length is 7 pixels and the window size is 10 pixels by 13 pixels, then the line length will be 7 / (13 × 2) = 0.269.

$$\text{length} = \frac{\text{number of pixels in a particular direction}}{(\text{window height or width}) \times 2} \qquad (5)$$

The operations discussed above for the encoding of horizontal line information must be performed for the remainder of directions. The last input vector value represents the number of intersection points in the character. It is calculated in the same manner as for the number of lines present. Figure 6 illustrates the process of input vector creation.

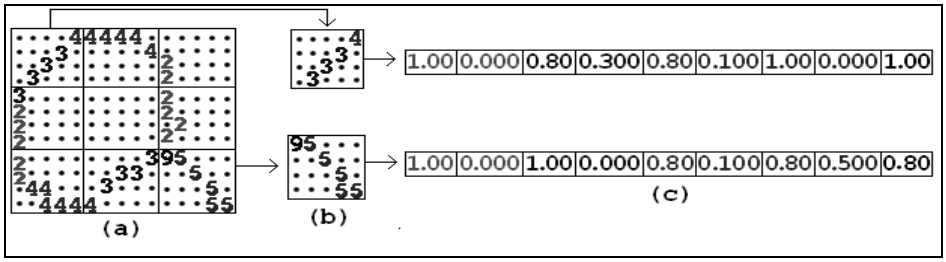

Figure 6: (a) Processed image, (b) Zoned windows, (c) Input vector components

2.5 Configuration of the neural classifiers

The neural classifiers chosen for the task of character recognition were Back-Propagation (BP) and Radial Basis Function (RBF) networks. 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. The most successful vector configurations were of size 25 for the contour code feature, 81 for the direction feature and 100 for the transition feature. For the BAC data set (whilst testing the contour code feature), two separate neural networks were trained for lowercase and uppercase characters. The number of outputs considered for this data set was 26 for each (characters a-z and A-Z). With the CAS data set, two separate neural networks were trained for lowercase and uppercase characters. This time the number of outputs considered for this data set was 27 (characters a-z and 1 reject neuron for non-character patterns). The non-character category was specifically used to identify character patterns that were smaller than approximately half a regular character and those character components that had not been correctly split i.e. multiple characters. For the BAC data set (whilst testing the direction feature), a single neural network architecture was considered with 36 outputs similar to that proposed by Singh and Hewitt [18].

2.6 Preparation of training and testing data

For neural network training it was necessary to include samples for each type of character (a-z, A-Z). In the case of the CAS data set, training and test files needed to be manually prepared, however character matrix patterns were determined based on the output of our heuristic segmenter. To summarise the character extraction process, our technique first proceeded to sequentially locate all non-cursive/printed character components through the use of character component analysis. Next, x-coordinates (vertical segmentations) for each connected character component (defined by the heuristic segmenter [28][29]) were used to define the vertical boundaries of each character matrix. The horizontal boundaries of the character matrix were deemed as the top-most and bottom-most y-coordinates of the connected component being examined. Each extracted character was viewed by a human operator and was labelled manually as belonging to a particular character class.

3. Experimental results

The first set of character recognition experiments tested the contour code feature using a back propagation neural network. The number of characters used for training and testing were 1500 and 1200 respectively from the BAC dataset. The results obtained for character recognition using the contour code feature were compared to the performance of the transition feature and are shown in Table 1.

Table 2 presents top results using the CAS dataset, the BP algorithm and the direction and transition feature extraction techniques described in Section 2. Separate experiments were once again 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.

Finally, Table 3 presents results for the entire BAC dataset processed by the direction feature extraction technique using the BP network for non-resized patterns. The BAC dataset contained a total of 19145 characters for training and 2183 characters for testing. Resized patterns were excluded from these experiments on the basis that in preliminary experiments with the CAS dataset, the results suggested that on average the non-resized dataset outperformed the resized one. The results are once again compared to those obtained using the transition feature extraction technique.

Table 1 Top character recognition rates using the BP network and the BAC dataset

	Recognition Rate [%]	
	Lowercase	**Uppercase**
Contour Code	**86.84**	**85.34**
Transition	83.46	84.64

Table 2 Top character recognition rates using the BP network and the CAS dataset

	Recognition Rate [%]	
	Lowercase	**Uppercase**
Direction (resized thinned)	**69.78**	78.70
Direction (resized boundary)	69.73	77.32
Direction (non-resized thinned)	69.02	**80.62**
Direction (non-resized boundary)	67.19	79.98
Transition	67.81	79.23

Table 3 Top character recognition rates using the BP network and the BAC dataset

	Recognition Rate [%]
Direction (non-resized thinned)	83.10
Direction (non-resized boundary)	**83.65**
Transition	82.82

4. Discussion of results

4.1 Contour vs Transition feature

The novel contour code feature extraction technique was compared with the transition feature and it was found that the proposed contour code feature produced, on average, much higher recognition rates.

4.2Direction vs Transition feature

Across all experiments that were performed, it was found that the direction feature outperformed the transition feature in all cases. However, whilst using the BP network, small differences in recognition rate may sometimes be attributed to variations in the starting conditions of the network. Hence, to confirm the veracity of the recognition rates attained, extra experiments, using the BP network configuration that provided the top result for direction and transition features, were conducted. For each feature type, a total of six experiments were conducted and the average recognition rate was calculated. In each case the direction feature outperformed the transition feature technique by 2% for lowercase characters and almost 7% for uppercase characters.

4.3 Contour vs Direction feature

From the experiments, it was found that the contour feature outperformed direction feature. The average recognition rate was slightly higher using the contour code feature at 86.09% in comparison to 83.65% for the direction feature.

4.4 General discussion

As may be seen from the results in Table 2, the use of non-resized patterns resulted in a comparable or slightly higher recognition rate. Whereas the difference in character classification rate when features were extracted from thinned character images and the character boundary was negligible.

4.5 Character recognition results

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, experimental settings and the handwriting database used. The comparisons presented below have been chosen for two main reasons. The handwriting database (CEDAR) 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 [15] presented a handwritten word recognition system, which was trained on segmented characters from the CEDAR benchmark database. 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. Therefore, if the top lower case (86.84%) and upper case (85.34%) character recognition scores in this research are averaged, a recognition accuracy of 86.09% is obtained, which compares well with their results. Kimura *et al.* [17] 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 86.09%. Singh and Hewitt [18] employed the modified Hough Transform on characters from the CEDAR. They obtained a recognition rate of 67.3%, our best result using their network configuration (83.65%) compares favourably with their top recognition rate. Finally, the techniques proposed in this research compared favourably to those presented by Camastra and Vinciarelli [20][22].

5. Conclusions and future research

We have presented a novel contour code and direction feature based approach for the recognition of segmented handwritten characters. Both techniques were compared to other popular techniques in the literature. In general, the proposed techniques outperformed the transition feature technique and were comparable to other techniques in the literature. The contour code technique seems to be very promising producing top results. In future research the contour code will be integrated into an off-line handwritten word recognition system.

6. References

[1] R. Plamondon and S. N. Srihari, 2000, On-Line and Off-Line Handwriting Recognition: A Comprehensive Survey, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **22**, 63-84.

[2] L. S. Frishkopf and L. D. Harmon, 1961, Machine Reading of Cursive Script, *Information Theory*, C. Cherry (ed.), Butterworth, London, 300-316.

[3] C. Y. Suen, and R. Legault, C. Nadal, M. Cheriet, and L. Lam, 1993, Building a New Generation of Handwriting Recognition Systems, *Pattern Recognition Letters*, **14**, 305-315.

[4] C. J. C. Burges, J. I. Be and C. R. Nohl, 1992, Recognition of Handwritten Cursive Postal Words using Neural Networks, *Proceedings of the 5th USPS Advanced Technology Conference*, 117-124.

[5] F. Kimura, S. Tsuruoka, M. Shridhar and Z. Chen, 1992, Context-Directed Handwritten Word Recognition for Postal Service Applications, *Proceedings of the 5th USPS Advanced Technology Conference*, 199-213.

[6] T. Paquet and Y. Lecourtier, 1991, Handwriting Recognition: Application on Bank Cheques, *Proceedings of the 1st International Conference on Document Analysis and Recognition*, St Malo, France, 749-757.

[7] D. Guillevic and C. Y. Suen, 1998, HMM-KNN Word Recognition Engine for Bank Cheque Processing, *Proceedings of the 14th International Conference on Pattern Recognition*, Brisbane, Australia, 1526-1529.

[8] T. Bruel, 1994, Design and Implementation of a System for Recognition of Handwritten Responses on US Census Forms, *Proceedings of the IAPR Workshop on Document Analysis Systems*, Kaiserlautern, Germany, 237-264.

[9] F. Kimura, S. Tsuruoka, Y. Miyake and M. Shridhar, 1994, A Lexicon Directed Algorithm for Recognition of Unconstrained Handwritten Words, *IEICE Trans. Information and Systems*, **E77-D**, 785-793.

[10] P. D. Gader, M. Magdi and J-H Chiang, 1992, Segmentation-based Handwritten Word Recognition, *Proceedings of the 5th USPS Advanced Technology Conference*, 215-226.

[11] P. D. Gader, M. Mohamed and J-H. Chiang, 1997, Handwritten Word Recognition with Character and Inter-Character Neural Networks, *IEEE Trans. Systems, Man, and Cybernetics-Part B: Cybernetics*, **27**, 158-164.

[12] G. Kim and V. Govindaraju, 1997, A Lexicon Driven Approach to Handwritten Word Recognition for Real-Time Applications, *IEEE Trans. Pattern Analysis and Machine Intelligence*, **19**, 366-379.

[13] J. Cai and Z-Q Liu, 1999, Integration of Structural and Statistical Information for Unconstrained Handwritten Numeral Recognition, *IEEE Trans. Pattern Analysis and Machine Intelligence*, **21**, 263-270.

[14] S-W. Lee, 1996, Off-Line Recognition of Totally Unconstrained Handwritten Numerals using Multilayer Cluster Neural Network, *IEEE Trans. Pattern Analysis and Machine Intelligence*, **18**, 648-652.

[15] S-B. Cho, 1997, Neural-Network Classifiers for Recognizing Totally Unconstrained Handwritten Numerals, *IEEE Trans. on Neural Networks*, **8**, 43-53.

[16] H. Yamada and Y. Nakano, 1996, Cursive Handwritten Word Recognition Using Multiple Segmentation Determined by Contour Analysis, *IEICE Trans .Information and Systems*, **E79-D**, 464-470.

[17] F. Kimura, N. Kayahara, Y. Miyake and M. Shridhar, 1997, Machine and Human Recognition of Segmented Characters from Handwritten Words, *4th International Conference on Document Analysis and Recognition (ICDAR '97)*, Ulm, Germany, 866-869.

[18] S. Singh and M. Hewitt, 2000, Cursive Digit and Character Recognition on Cedar Database, *International Conference on Pattern Recognition, (ICPR 2000)*, Barcelona, Spain, 569-572.

[19] B. Lazzerini and F. Marcelloni, 2000, A Linguistic fuzzy recogniser of off-line handwritten characters, *Pattern Recognition Letters*, **21**, 319-327.

[20] F. Camastra and A. Vinciarelli, 2001, Cursive character recognition by learning vector quantization, *Pattern Recognition Letters*, **22**, 625-629.

[21] M. Hanmandlu, K. R. Murali Mohan, S. Chakraborty, S. Goyal, D. Roy Choudhury, 2003, Unconstrained handwritten character recognition based on fuzzy logic, *Pattern Recognition*, **36**, 603-623.

[22] F. Camastra and A. Vinciarelli, 2003, Combining neural gas and learning vector quantization for cursive character recognition, *Neurocomputing*, **51**, 147-159.

[23] M. Blumenstein, B. K. Verma and H. Basli, 2003, A Novel Feature Extraction Technique for the Recognition of Segmented Handwritten Characters, *7th International Conference on Document Analysis and Recognition (ICDAR '03)*, Edinburgh, Scotland, 137-141.

[24] M. Blumenstein, and B. K. Verma, 2001, Analysis of Segmentation Performance on the CEDAR Benchmark Database, *6th International Conference on Document Analysis and Recognition, (ICDAR '01)*, Seattle, U.S.A, 1142-1146.

[25] J. J. Hull, 1994, A Database for Handwritten Text Recognition, *IEEE Trans.Pattern Analysis and Machine Intelligence*, **16**, 550-554.

[26] N. Otsu, 1979, A Threshold Selection Method from Gray-Level Histograms, *IEEE Transactions on Systems, Man, and Cybernetics*, **SMC-9**, 62-66.

[27] R. M. Bozinovic, and S. N. Srihari, 1989, Off-Line Cursive Script Word Recognition, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **11**, 68-83.

[28] M. Blumenstein and B. Verma, "A New Segmentation Algorithm for Handwritten Word Recognition", *Proceedings of the International Joint Conference on Neural Networks, (IJCNN '99)*, Washington D.C., 1999, pp. 2893-2898.

[29] M. Blumenstein and B. Verma, 1999, "Neural Solutions for the Segmentation and Recognition of Difficult Words from a Benchmark Database", *Proceedings of the Fifth International Conference on Document Analysis and Recognition, (ICDAR '99)*, Bangalore, India, pp. 281-284.

[30] M. Blumenstein, and B. Verma, 2001, "Analysis of Segmentation Performance on the CEDAR Benchmark Database", *Proceedings of the Sixth International Conference on Document Analysis and Recognition,* (*ICDAR '01)*, pp. 1142-46.

[31] M. Shridhar and A. Badreldin, "High Accuracy Character Recognition using Fourier and Topological Descriptors", *Pattern Recognition*, Vol. 17, 1984, pp. 515-524.