Chapter 6

Development of Character Segmentation Techniques

To recognize a word or a sentence it is necessary to recognize each alphabet individually and so segmentation of the handwritten characters of the text must be done accurately. The proper segmentation of the characters from the text directly puts an impact on the success of the handwriting recognition method. This dilemma is known as chicken-egg relationship [99]. The easiest way to segment out the characters is to identify the gaps present between the characters and this is well and good for those handwriting styles which contains isolated characters and printed text [107]. Extraction of individual characters in fused and joined handwritten text [14] does not follow this method. It is a challenging task, because different individuals have different handwriting styles.

In cursive type of handwriting, it is too difficult to find the boundary between two consecutive characters. The problem of joined characters arises in many non English regional scripts like Gurmukhi [174, 14 and 102], Urdu and Bengali [137]. Joined handwriting characters may be written using one or more strokes [192]. This may create ambiguities while segmentation which is known as character-within-character problem. For example, the character ‘d’ can sometimes be recognized as ‘cl’, if written in separate strokes. Actually, character segmentation is a different field of research and lot of work has already been carried out in this field. Still, many researchers are presently working in this field to achieve maximum efficiency.

Two character segmentation methods i) Slider Drifting Method (SDM) and ii) Baseline Pixel Burst Method (BPBM) has been developed and presented here. The first method is quite simple. It has been developed to extract lines from the text, words from the lines and finally characters from the words. The method has been designed to extract out isolated characters having gaps in between the characters. The second method extracts out characters from isolated as well as cursive and joined type of handwritings efficiently.

6.1 Extraction of English Alphabets from Words and Sentences Using Slider Drifting Method (SDM)

In order to recognize handwritten characters, it is very important to extract out single characters from the word/sentence. The approach is to extract out individual characters, by forming a rectangular border around the word or the sentence and drifting a vertical slider from the beginning of the word or sentence towards the end of the text, to find out the alphabet delimiters and enclose the characters within the box.

Slider Drifting Method (SDM) is one type of gap finding method. Gaps are found between words and sentences by using a vertical slider which is nothing but a virtual column of white pixels. The number of white pixels in the slider is equal to the height of the contour. Height of the contour is number of pixels present in any column of the character enclosed inside the contour.

The overall method has been divided into two parts. The first part forms a boundary tightly enclosing the sentence to be segmented out. The second part extracts out individual characters from the sentence.

6.1.1 Methodology of SDM

A sentence has been written on a piece of A4 size paper using a black marker. The sentence has been scanned using a high definition scanner as shown in Figure 6.1. The scanned sentence can be enclosed within a rectangular boundary by using a contour tracing method [15]. A contour is a rectangular boundary enclosing the paragraph image. Here a contour tracing method has been developed as given in Algorithm 6.1. This method has been tested with a sentence having disjoint characters and then with a sentence having joined and fused characters.

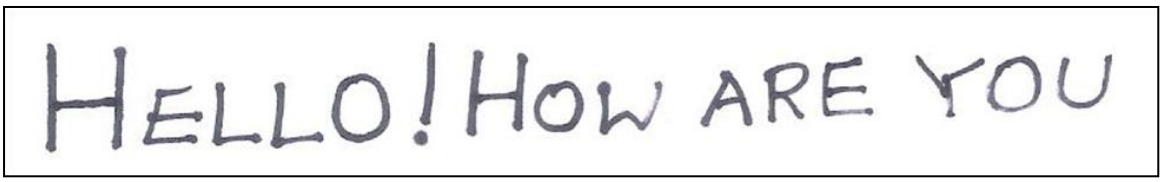

Figure 6.1: Scanned Sentence

Algorithm 6.1(Text_Encage): Encaging the Sentence in a Rectangle with the Image Fit to Boundaries

STEP 1. Read image matrix ‘A’.

STEP 2. Initialize the variable ‘j’ to 1. [Where, ‘j’ is the location of the first column from the left touching the sentence boundary.]

STEP 3. Initialize the variable ‘counter’ to ‘n’. [Where, ‘n’ is number of rows in the matrix.]

STEP 4. Repeat while counter is equal to n:

Initialize the variable ‘counter’ to 0.

Initialize the variable ‘i’ to 1.

Repeat while ‘i’ less than equal to n:

If A(i, j) equals 0 [Where, 0 indicates the white pixel.]

Increment ‘counter’ by 1.

End of If

End of while

If counter equals ‘n’

Increment ‘j’ by 1.

End of If

End of Step 4 loop

STEP 5. Set matrix ‘B1’=A(1 to n, j to m). [Where, the variable ‘m’ represents the number of columns in the image matrix.]

STEP 6. Initialize the variable ‘k’ to ‘m’. [Where, ‘k’ is the location of the last column from the right touching the sentence boundary.]

STEP 7. Initialize ‘counter’ to ‘n’.

STEP 8. Repeat while ‘counter’ is equal to ‘n’:

Initialize ‘counter’ to 0.

Initialize ‘i’ to 1.

Repeat while ‘i’ is less than equal to ‘n’:

If A(i,k) is equal to 0 [Where, 0 indicates white pixel.]

Increment counter by 1.

End of If

End of while

If ‘counter’ equals ‘n’

Decrement ‘k’ by 1.

End of If

End of Step 8 loop

STEP 9. Set matrix ‘B2’ = A(l to n, j to k).

STEP 10. Initialize the variable ‘l’ to 1. [Where, ‘l’ is the location of the first row from the top touching the sentence boundary.]

STEP 11. Initialize ‘counter’ to ‘m’.

STEP 12. Repeat while ‘counter’ is equal to ‘m’:

Initialize ‘counter’ to 0.

Initialize ‘i’ to 1.

Repeat while ‘i’ is less than equal to m:

If A(l, i) equals to 0 [Where 0 indicates white pixel.]

Increment ‘counter’ by 1.

End of If

End of while

If ‘counter’ equals ‘m’

Increment ‘l’ by 1.

End of If

End of Step 12 loop

STEP 13. Set matrix ‘B3’ = A(1 to n, j to k).

STEP 14. Initialize the variable ‘p’ to ‘n’. [Where, ‘p’ is the location of the last row from the bottom touching the sentence boundary.]

STEP 15. Initialize ‘counter’ to ‘m’.

STEP 16. Repeat while ‘counter’ is equal to ‘m’:

Initialize ‘counter’ to 0.

Initialize ‘i’ to 1.

Repeat while ‘i’ is less than equal to ‘m’:

If A(p,i) equal to 0, where 0 indicates white pixel

Increment ‘counter’ by 1.

End of If

End of while

If ‘counter’ equals ‘m’

Decrement ‘p’ by 1.

End of If

End of Step 16 loop

STEP 17. Set matrix ‘B4’ = A(1 to p, j to k).

STEP 18. STOP.

Matrix ‘B4’ represents the final image matrix en-caging the sentence, fit to boundaries as shown in Figure 6.2. It can be observed that the initially scanned image matrix is en-caged in a rectangular box which consists of unwanted space other than the text.

Figure 6.2: Image obtained after Applying Algorithm 'Text_Encage'

Algorithm 6.1 has been used to remove the unwanted space and enclose the text in exactly inside a rectangle. The rectangle boundaries touch the edges of the largest character, reducing the unwanted white area. Algorithm 6.2 has been applied to extract out individual characters as shown in Figure 6.3.

Algorithm 6.2(Extract_Alphabet): Slider Drifting Algorithm to Extract out Alphabets from the Sentences

STEP 1. Initialize the variables 'k' to 1 and 's_k' to 'j'. [Where, 's_k' locates the column representing the alphabet delimiter and j is the starting location from where the column starts drifting in order to find out the alphabet delimiter.]

STEP 2. Initialize 'counter' to 0.

STEP 3. Repeat while 'counter' is not equal to 'n': [Where, 'n' is the number of rows in the image matrix encapsulating the sentence.]

STEP 4. Initialize 'counter' to 0.

STEP 5. Initialize 'i' to 1.

STEP 6. Repeat while 'i' is not equal to 'n':

STEP 7. If A(i, s_k) equals 0

‘counter’ = ‘counter’ + 1.

End of If

End of Step 6 loop

If ‘counter’ is not equal to ‘n’

‘s_k’ = ‘s_k’ + 1

End of If

End of Step 3 loop

STEP 8. Set ‘s_k’= A(1 to m, j to s_k-1). [Where, matrix ‘s_k’ encages the first alphabet of the sentence matrix.]

STEP 9. Assign ‘s_gap’ to ‘s_k’. [Where, ‘s_gap’ locates the gap between two alphabets in the sentence.]

STEP 10. Initialize ‘counter’ to ‘n’.

STEP 11. Repeat while ‘counter’ is equal to ‘n’:

STEP 12. Initialize ‘counter’ to 0.

STEP 13. Initialize ‘i’ to 1.

STEP 14. Repeat while ‘i’ is not equal to ‘n’:

STEP 15. If A(i,s_gap) equals 1

‘counter’ = ‘counter’ + 1

End of If

End of Step 14 loop

STEP 16. If ‘counter’ equals ‘n’

‘s_gap’ = ‘s_gap’ + 1

End of If

End of Step 11 loop

STEP 17. Assign ‘s_k’+1 to ‘s_gap’. [Where, ‘s_k’+1 is the starting position of the next alphabet in the sentence.]

STEP 18. Test stopping condition: [Where, stopping condition is the length if the sentence]

If stopping condition is false go to Step 1.

STEP 19. STOP.

The column which finds out the alphabet delimiter is treated as a vertical slider that is why this method is named as Slider Drifting Method (SDM).

Figure 6.3: Image Obtained after Applying Algorithm ‘Extract_Alphabets’

It can be observed that Algorithm 6.2 finds out the delimiters of the characters by using a vertical slider as shown in Figure 6.3. Algorithm 6.1 has been applied to the individual characters to remove the extra space created above the small sized characters and en-cage the characters to fit into the boxes touching the character boundaries as shown in Figure 6.4.

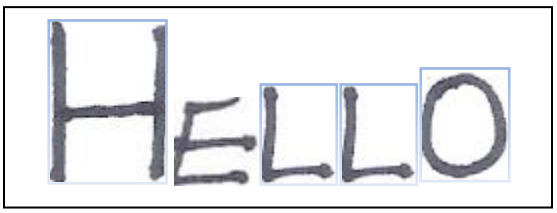

Figure 6.4: Image of Individual Characters Obtained after Applying Algorithm 'Text_Encage'

The image matrix formed for different characters are of different sizes. Already existing function in the MATLAB software can be used to resize the images of different sizes to some standard sized matrix. The standard sized matrix images have been used for the training purpose and presented to the net.

6.1.2 Performance Analysis of SDM

It has been found that the method is very efficient to extract out the characters from the sentences which consists of disjoint characters. A sentence which consists of fourteen disjoint alphabets has been considered to test the performance of SDM. It has been found that this method successfully extracted out all the alphabets. The accuracy of the method to test the presented text has been found 100%. A sentence extracted out of a paragraph containing joined and fused characters has been presented to SDM. SDM extracts out the joined characters as individual characters as shown in Figure 6.5. To overcome this situation Baseline Pixel Burst Method (BPBM) has been developed and presented in the next section.

Figure 6.5: A Sentence having few Joined Characters

6.2 Handwritten English Character Segmentation by Baseline Pixel Burst Method (BPBM)*

Baseline Pixel Burst Method (BPBM) is a character segmentation approach where a paragraph has been taken from an individual whose handwriting consists of both isolated and joined characters. For the isolated ones the segmentation has been done very easily by identifying the gaps between characters but it becomes cumbersome to segment out joined characters as there is no standard method to segment out characters from the joined words. The method developed and used here actually draws baselines on various rows of the image word matrix whereas vertical lines try to find out character boundaries by analyzing the densities of the pixels near the baselines.

Some sort of preprocessing may be required before and after segmentation of the characters in order to remove or reduce the ambiguities. The approach is to simplify the method by segmenting as many as characters by applying a suitable gap finding procedure and to try the joined ones.

BPBM is an approach where the joined and fused characters can be segmented by drawing baselines at different rows of the word or part of the word image matrix containing joined characters and analyzing baseline pixel densities to find out the locations on the baselines, where a vertical line can be drawn to extract out the single characters.

The overall method has divided into five phases. Paragraph, containing the text to be segmented out has been obtained and converted into a two dimensional binary matrix in Phase I. Paragraph has been segmented into separate lines of text in Phase II. Lines have been segmented into separate words in Phase III. A suitable gap finding method has been used to extract out the isolated characters as well as chunks of words containing joined characters in Phase IV. Joined characters have been extracted out finally, in Phase V. Figure 6.6 shows the conceptual diagram of BPBM.

*** Based on author's Publication no. 14 [Appendix B]**

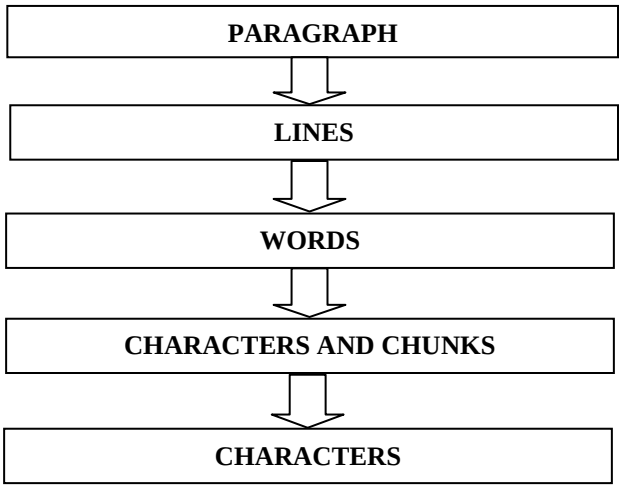

Figure 6.6: Conceptual Diagram of BPBM

6.2.1 Phase I - Preparing the Input

A high definition scanner has been used to scan a handwritten paragraph, initially. The paragraph shown in Figure 6.7 has been converted into a two dimensional binary image matrix.

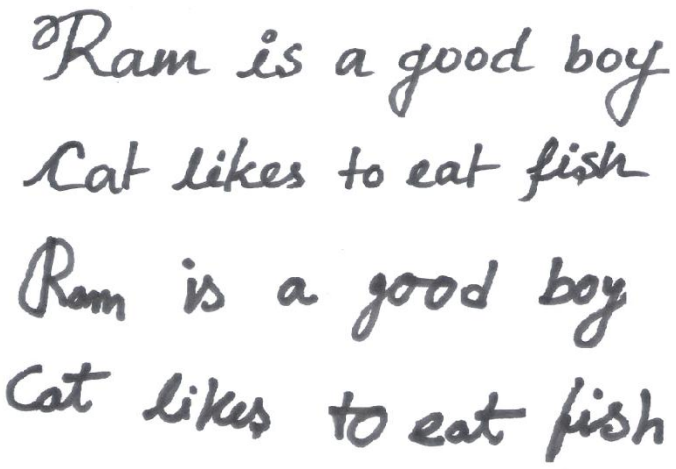

Figure 6.7: Initial Paragraph

The contour tracing method used in SDM has been used here to enclose the text paragraph within a rectangular boundary.

6.2.2 Phase II - Segmentation of the Lines from a Text Paragraph

The binary paragraph image matrix obtained in phase I has been considered for the segmentation of lines. Algorithm 6.3 extracts the lines from the paragraph. Figure 6.8 shows one such line. The process repeats till the end of the paragraph, extracting out all the lines of the paragraph.

Algorithm 6.3 (Segment_Line): Segmentation of Lines from a Text Paragraph

STEP 1. Read the paragraph image stored in bmp format.

STEP 2. Convert the bmp format of the image to a two dimensional binary matrix.

Figure 6.8: First Line of the Paragraph

STEP 3. Initialize ‘i’ to 1 and ‘counter’ to l. [Where, ‘i’, represents the location of the current row of the matrix and l is the number of columns in the matrix.]

STEP 4. Repeat Steps 5 to 8, while ‘counter’ is equal to l:

STEP 5. Initialize ‘counter’ to 0.

STEP 6. Repeat Step 7, for ‘j’ = ‘l’ to 1. [Where, ‘j’ represents the location of the column of the matrix.]

STEP 7. If P[i,j] is equal to 1

Increment ‘counter’ by 1.

End of If

End of Step 6 loop

STEP 8. If ‘counter’ is equal to l

Increment ‘i’ to 1.

End of If

End of Step 4 loop

STEP 9. Set the variable ‘s’ equal to ‘i’. [Where, ‘s’ is the beginning of the first pixel of the line found from the top of the line.]

STEP 10. Set ‘counter’ to 0.

STEP 11. Repeat Steps 12 to 15, while ‘counter’ is not equal to l:

STEP 12. Initialize ‘counter’ to 0.

STEP 13. Repeat Step 14, for ‘j’ = ‘l’ to 1. [Where, ‘j’ represents the location of the column of the matrix.]

STEP 14. If P[i,j] is equal to 1

Increment ‘counter’ by 1

End of If

End of Step 13 loop

STEP 15. If ‘counter’ is not equal to l

Increment ‘i’ to 1.

End of If

End of Step 11 loop

STEP 16. Set the variable ‘e’ equal to ‘i’- 1. [Where, ‘e’ is the end pixel of the first line of the paragraph]

STEP 17. Set the matrix ‘P1’ equal to P[s: e, 1: l]. [Where, ‘P’ is the matrix representing the paragraph, ‘P1’ is the matrix representing the first line.]

STEP 18. Repeat Steps 1 to 17 for all the lines in the paragraph.

STEP 19. STOP.

6.2.3 Phase III - Segmentation of the Words from the Lines of the Text Paragraph

The lines obtained in phase II has been considered for the segmentation of words.

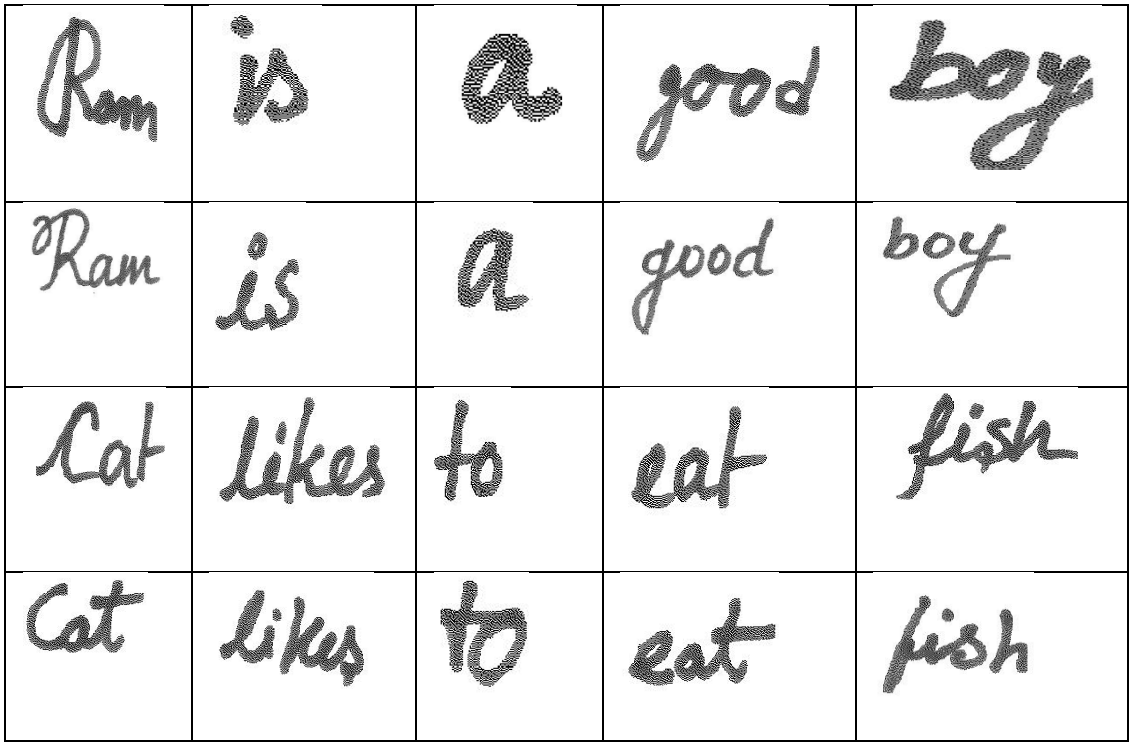

Figure 6.9: Segmented Words from the Lines of the Paragraph

Algorithm 6.4 extracts the words as shown in Figure 6.9, from the lines of the paragraph

Algorithm 6.4 (Segment_Word): Segmentation of the Words from the Lines of the Text Paragraph

STEP 1. Repeat Steps 2 to 14 for all the lines in the paragraph:

STEP 2. Initialize 'j' to 1 and 'counter' to 'w_k'. [Where, 'j' is the location of the current column of the line and 'w_k' is the width of the line 'k' of the paragraph.]

STEP 3. Repeat Steps 4 and 5 while 'counter' is equal to 'w_k':

STEP 4. Repeat Step 5 for 'i' = 1 to 'w_k': [Where, 'i' is the location of the row of the line.]

STEP 5. Set 'counter' to zero.

If $P_k[i,j]$ is equal to 1 [Where, $P_k[i,j]$ is matrix element of i^{th} row and j^{th} column of k^{th} line]

Increment 'counter' to 1.

End of If

End of Step 4 loop

If counter is equal to w_k

Increment j to 1

End of If

End of Step 3 loop

STEP 6. Initialize the variable 'sw' to 'j'. [Where, 'sw' is the start of the word of the line.]

Initialize ‘counter’ to 0 and ‘gap’ to 0. [Where, ‘gap’ is the parameter measuring the ‘gap’ between the words which also decides the end of the word]

STEP 7. Repeat Steps 8, 9, 10, 11 and 12 while ‘gap’ is not equal to 3: [Where, value 3 is taken at random]

STEP 8. Repeat Step 9, 10 and 11, while ‘counter’ is not equal to ‘w_k’:

Set ‘counter’ to 0.

STEP 9. Repeat Step 10 for ‘i’ = 1 to ‘w_k’.

STEP 10. If $P_k[i,j]$ is equal to 1

Increment ‘counter’ to 1.

End of If.

End of Step 9 Loop.

STEP 11. If ‘counter’ is not equal to ‘w_k’

Increment ‘j’ to 1.

End of If.

End of Step 8 loop

STEP 12. Set ‘counter’ to 0.

Increment ‘gap’ to 1

End of step 7 loop

STEP 13. Set ‘ew’ to ‘j’-‘gap’. [Where, ‘ew’ is the location of the end of the word.]

STEP 14. Set 'WD_m' = P_k[1:w1,sw:ew]. [Where, 'WD_m' is the m_{th} word]

STEP 15. STOP.

6.2.4. Phase IV - Segmentation of the Isolated Characters and Chunks from the Words of the Lines

The words obtained in phase III has been considered for the segmentation of isolated characters and the chunks containing more than one characters joined together and which appear as a single character. These words are presented to Algorithm 6.5 and the result is shown in Figure 6.10.

Algorithm 6.5 (Segment_Chunk): Segmentation of the Isolated Characters and Chunks from the Words present in the Lines

STEP 1. Initialize 'j' to 1 and 'counter' to 'w_1'. [Where, 'j' is the current location of the column and 'w_1' is the number of rows in the matrix containing the current word.]

STEP 2. Repeat Steps 3, 4, 5 and 6 while 'counter' is equal to 'w_1':

STEP 3. Set 'counter' to 0.

STEP 4. Repeat Step 5 for 'i'=1 to 'w_1'

Step 5. If P_1[i,j] is equal to 1
Increment 'counter' to 1.
End If
End of Step 4 loop

STEP 6. If 'counter' is equal to 'w_1'
Increment 'j' to 1.
End If
End of Step 2 loop

STEP 7. Set 'sc' to 'j'. [Where, 'sc' is the start of the character and 'j' is the current location of the column from where the character starts.]

STEP 8. Set 'counter' to 0.

STEP 9. Repeat Steps 10, 11, 12 and 13 while 'counter' is not equal to 'w_1':

STEP 10. Set 'counter' to 0.

STEP 11. Repeat Step 12 for 'i' = 1 to 'w_1':

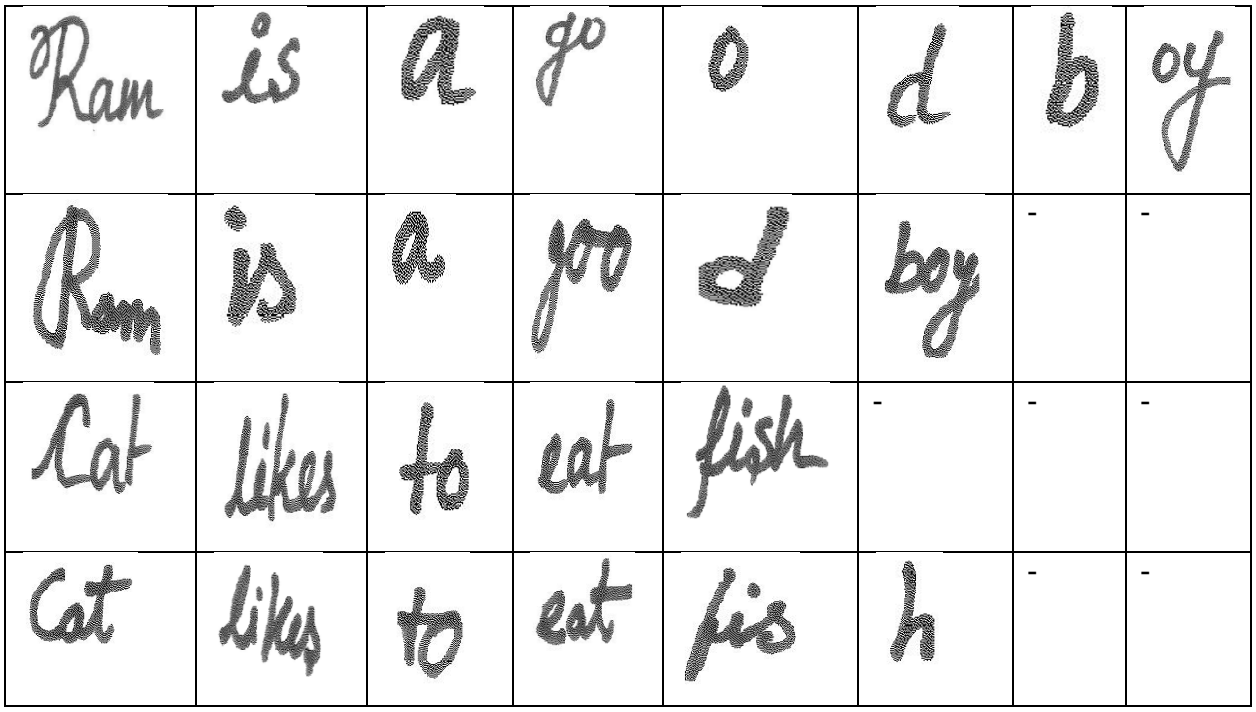

Figure 6.10: Segmented Chunks from Different Words of the Paragraph

STEP 12. If $P_1[i,j]$ is equal to 1

Increment 'counter' to 1.

End If

End of Step 11 loop

STEP 13. If 'counter' is not equal to 'w_1'

Increment 'j' to 1.

End If

End of Step 9 loop

STEP 14. Set 'ec' to 'j'-1 [Where, 'ec' is the end of the character.]

STEP 15. Set matrix 'C_1' = P_1[1:w1,sc:ec] [Where, C_1 is the matrix which represents the first segmented character.]

STEP 16. STOP.

6.2.5. Phase V - Segmentation of the Characters from the Chunks which appear as Characters

The chunks and words obtained in phase IV have been considered for the segmentation of the individual characters. Algorithm 6.6 breaks the chunks and words into individual characters as shown in Figure 6.11.

Figure 6.12, represents the extraction of single characters from a chunk of three joined characters using Algorithm 6.6.

Algorithm 6.6 (Segment_Character): Segmentation of the Characters from the joined words which appear as Characters

STEP 1. Read the portion of the word, having joined characters, as a two dimensional Matrix.

STEP 2. Repeat Steps 3 to 14, for all the characters or chunks, present in the matrix.

STEP 3. Drift upwards, starting from the bottom row of the matrix, taking one row at a time, stop at the location, where, the row finds its first black pixel and draw a horizontal baseline at that location and store the location using a variable.

STEP 4. Drift towards the rightmost position, starting from the leftmost column taking one column at a time, stop at the location, where the column finds its first black pixel and draw a vertical solid line on that location and store the location using a variable. [Where, vertical solid line shows the beginning of the character.]

STEP 5. Drift towards the rightmost position, starting from the current location of the solid line, taking one column at a time, stop at the location, where the column finds a single burst of black pixels near the baseline and draw a vertical line at that location and store the location using a variable. [Where, vertical line may be the end of the character. While drifting towards rightmost position, the succeeding columns, show bursts of black pixels, showing the presence of the character image, the end of the character image shows the pixel burst only at the joint which is mostly found near the baseline.]

STEP 6. Draw a dotted line with end points at the middle of the starting and ending lines of the suspected character.

STEP 7. If numbers of bursts of black pixels on the dotted line is greater than 1, then convert the vertical line into a solid line and go to step 12 else go to Step 8. [Where, the character may be ‘i’ or a portion of ‘u’, ‘w’, ‘y’ or ‘v’.]

STEP 8. Drift downwards from the current location of the upper horizontal line, stop at the first black pixel and draw another horizontal line at that location, with end points at the starting and ending position of the character under observation.

STEP 9. Drift downwards from the current location of the horizontal line, stop at the location having no black pixel on the horizontal line and draw another horizontal line at that location.

STEP 10. Drift downwards from the current location of the horizontal line, stop at the

first black pixel and draw another horizontal line at that location. [Where, Steps 8, 9 and 10 show the presence of character ‘i’ or ‘j’.]

Figure 6.11: Segmented Characters from the Different Joined Chunks which Appears as Characters

STEP 11. If Steps 8, 9 and 10 are met then convert the vertical line ending at the character having single burst of pixel into a solid line else go to Step 4.

STEP 12. Start from the current location and repeat Step 3 to find the starting location of the next character. Draw a solid line on that location.

STEP 13. Start from the current location and repeat Step 4 to find the end of the next character and draw a solid line on that location

STEP 14. Extract out all the suspected characters using MATLAB.

STEP 15. STOP.

6.2.6. Performance Analysis of BPBM

The experiment produces satisfactory results for the test paragraph as given in Figure 6.7. Following are the details of the results produced by the method after presenting the test paragraph:

Number of paragraphs tested = 1
Number of lines present in the paragraph = 4
Number of lines identified by applying Algorithm ‘Segment_Line’ = 4
Accuracy of identifying the lines = 100 %
Number of words present in the paragraph = 20
Number of words obtained after applying Algorithm ‘Segment_Word’ = 20
Accuracy of identifying the words = 100 %
Number of chunks present in the paragraph = 25
Number of chunks obtained after applying Algorithm ‘Segment_Chunk’ = 25
Accuracy of identifying the chunks containing joined characters from words = 100 %
Number of characters present in the paragraph = 60
Number of characters identified by Algorithm ‘Segment_Character’ = 57
Accuracy of identification the characters = 95%

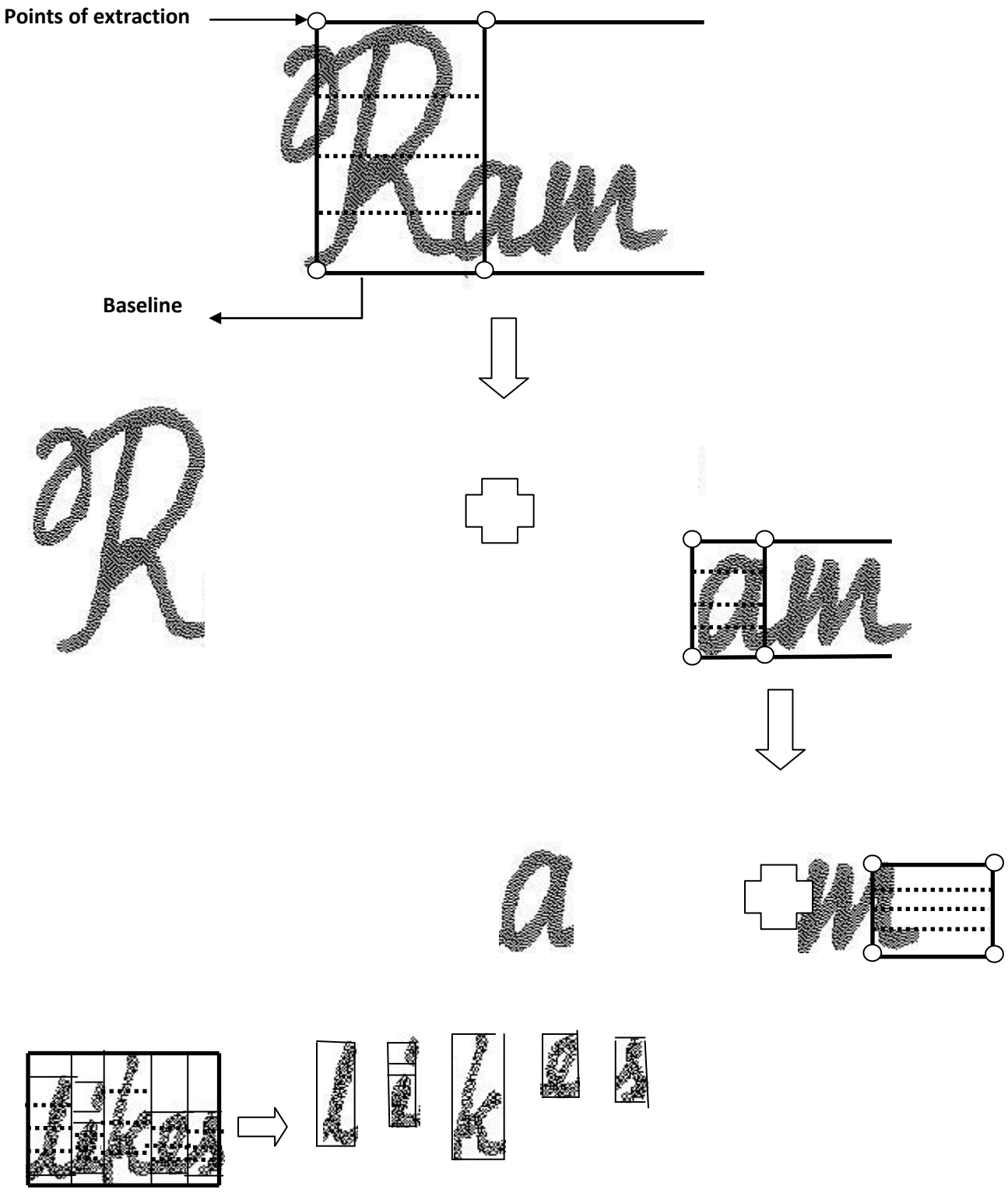

Figure 6.12: Extraction of Individual Characters from Joined Chunks

The method BPBM has also been tested with two sample sets of paragraphs for accuracy measurement. These sample sets are used to train and test the developed ANNs. Each set contains 10 paragraphs i.e. 10 pages. Each paragraph contains a sentence written by an individual ten times on a piece of paper i.e. 10 sentences. Each sentence contains different joined and isolated characters. 10 such distinct characters are present in each sentence. The overall accuracy of the BPBM method has been measured on the identification of these 2000 characters present in two sets of paragraph, each containing 1000 characters. A sample page Paragraph of Sample Set-1 and a sample page Paragraph of Sample Set-2 are shown in Figure 6.13 and Figure 6.14 respectively.

The performance of BPBM is given below:

Number of characters present in paragraph set-1 = 1000

Numbers of characters properly segmented = 886

Accuracy in paragraph set-1 = 88.6%

Number of characters present in paragraph set-2 = 1000

Numbers of characters properly segmented = 820

Accuracy in paragraph set-2 = 82%

Average accuracy of two paragraph sets = 85.3%

6.3 Discussion

SDM has been applied for the sentences containing characters having spaces in between. BPBM may be efficiently applied to the handwritten text paragraphs, which may or may not have spaces in between.

Cursive type handwritings may produce faulty results if the input is noisy. As for example, it is difficult to find the existence of single ‘w’ or double ‘u’ in case of the presence of two consecutive ‘u’s. Preprocessing of the input may help to remove the noise.

Again efficiency of the methods is greatly dependent on the style of handwritings as is found in Section 6.2.6 by different experiments.

The average efficiency of the method BPBM is found to be around 85.3%.

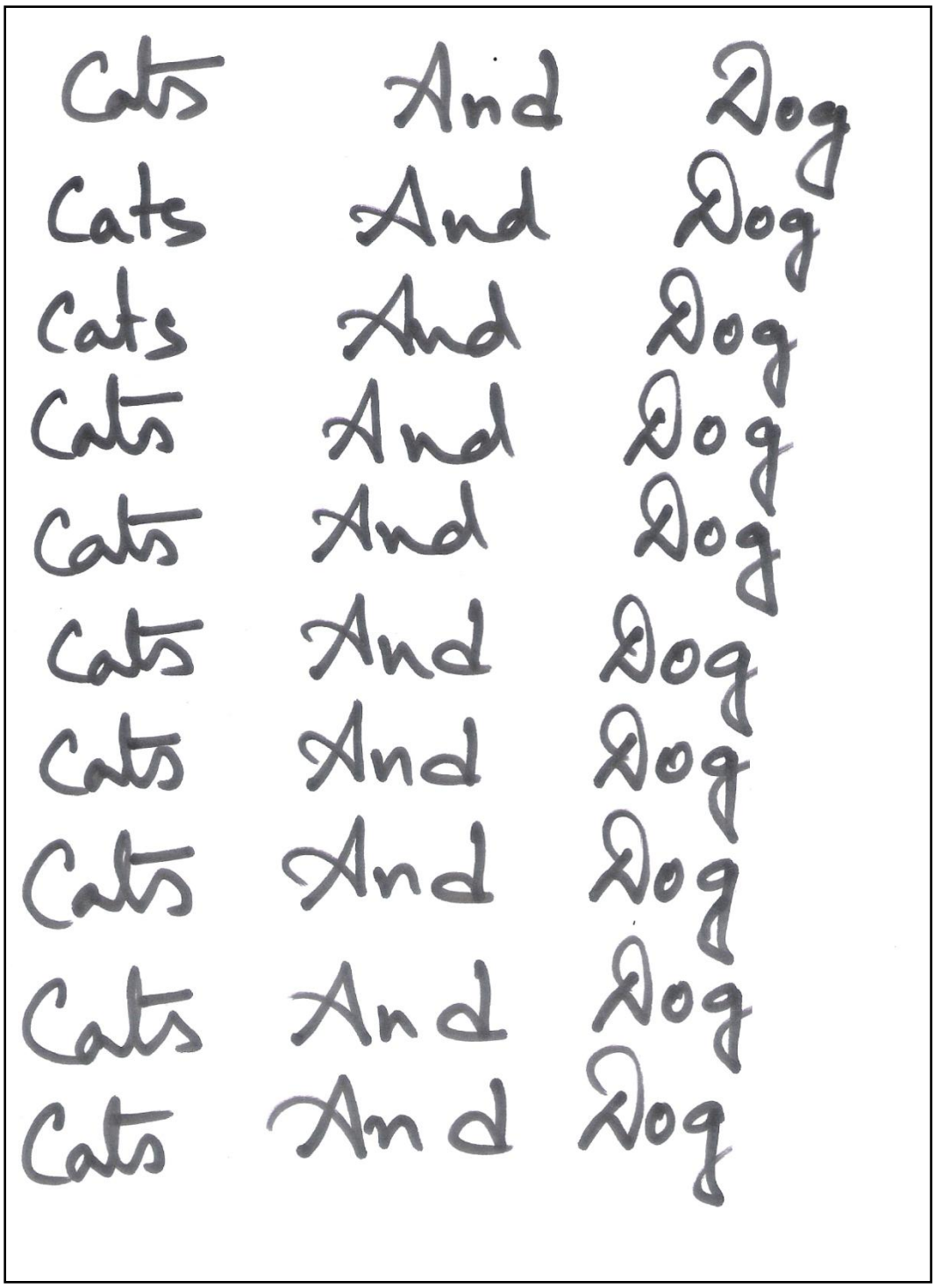

Figure 6.13: One Paragraph of Sample Set-1

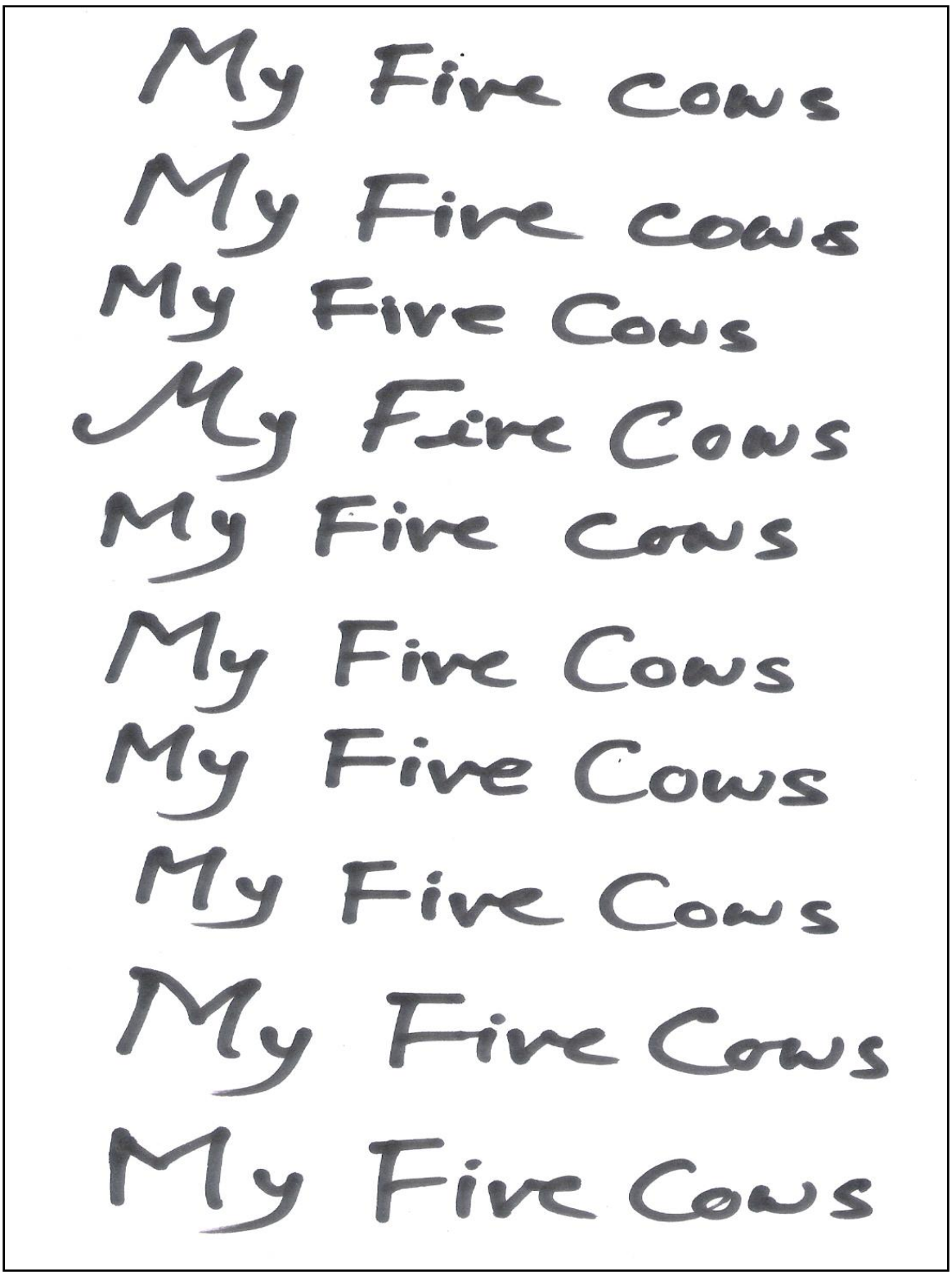

Figure 6.14: One Paragraph of Sample Set-2

6.4 Conclusion

Segmentation of individual characters from the handwritten text plays an important role in the development of methods for identification of handwritten documents. The process of extracting out the characters is a complex task and difficulties vary from language to language as well as person to person. Two developed simple methods of segmentation (SDM and BPBM) have been presented here with their limitations. Average accuracy of Baseline Pixel Burst Method (BPBM) and Slider Drifting Method (SDM) comes out as 85.3% and 92 %(with limitations) respectively.