SEGMENTATION OF TOUCHING AND FUSED DEVANAGARI CHARACTERS

Veena Bansal and R. M. K. Sinha
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur 208016 India

Abstract-Devanagari script is a two dimensional composition of symbols. It is highly cumbersome to treat each composite character as a separate atomic symbol because such combinations are very large in number. This paper presents a two pass algorithm for the segmentation and decomposition of Devanagari composite characters/symbols into their constituent symbols. The proposed algorithm extensively uses structural properties of the script. In the first pass, words are segmented into easily separable characters/composite characters. Statistical information about the height and width of each separated box is used to hypothesize whether a character box is composite. In the second pass, the hypothesized composite characters are further segmented. A recognition rate of 85 percent has been achieved on the segmented conjuncts. The algorithm is designed to segment a pair of touching characters.

Key Words: Devanagari Script Character/Text Recognition Prototype Construction Character fusion Character fragmentation Character segmentation/decomposition.

1 Introduction

Segmentation of a document into lines and words, and of words into individual characters and symbols constitute an important task in the optical reading of texts. Presently, most recognition errors are due to character segmentation errors [(1)]. Very often, adjacent characters are touching, and may exist in an overlapped field [(1)]. Therefore, it is a complex task to segment a given word correctly into its character components.

In this paper, we have considered the problem of segmenting printed text in Devanagari. Devanagari is the script for Hindi (the official language of India), Sanskrit, Marathi and

Nepali languages, among others. It is used by more than 400 million people across the globe. Devanagari is a derivative of ancient Brahmi, the mother of all Indian scripts. It is a logical script composition of symbols in two dimensions. The two dimensional composition has to be decomposed into symbols that are meaningful in the script.

2 Background

There has always been a dilemma in a segmentation task whether to segment first and then classify, or classify while segmenting. Casey and Lecolinet [2] while reviewing strategies for segmentation have argued that the segmentation is interdependent with the local decisions of shape similarity and with global contextual acceptability. They conclude that the strategies for segmentation can be classified into three *pure* elemental strategies as follows:

1. The classical approach where segments are identified based on *character-like* features;
2. Recognition-based segmentation, in which a search is made for image components that match with the character classes in the alphabet; and
3. A holistic approach that attempts to recognize the word as a whole.

Our approach to Devanagari script segmentation is a hybrid of the classical and recognition-based segmentation.

Many algorithms have been proposed for segmentation of touching characters [1, 3, 4, 5, 6] for the Roman script. All of these algorithms generate multiple segmentation points.

Most of these methods have been investigated for the Roman script. The OCR work on Indian scripts [7, 8, 9, 10, 11] have not considered the problem of touching and fused character segmentation. Although algorithms for preliminary segmentation and isolation of shadow characters have been developed earlier [8] and similar techniques have been used by Choudhary et al. [12, 13, 14], no attempt has been made so far in isolating the touching and fused composite characters of Devanagari script. This work represents the first of its kind, to the best of our knowledge, in this direction.

In section 3, an example illustration of the segmentation process is presented. This is followed by a brief description of Devanagari script features that are relevant from an OCR viewpoint in section 4.

This is followed by a presentation of the details of the preliminary segmentation process in section 5. In section 6, the algorithm for isolation of touching characters is presented. Isolation of shadow character pair and lower modifier symbols are discussed in sections 7 and 8 respectively. In section 9, we present results of our investigation. This is followed by some conclusions that can be drawn from this work. The conclusions are given in section 10.

3 An Illustration of Devanagari segmentation process

Let us introduce Devanagari script segmentation process with the help of the word
that is pronounced as 'unmulan'. The word image is shown in figure 1(a). The subimages after ideal segmentation of the example word is shown in figure 1(b). The most dominating horizontal line in the word is referred to as the *header line* that glues the characters of the word together. Let us remove it from the word. The resulting image is shown in figure 1(c). Now we extract the subimages that are separated from their neighbours by vertical white space. These subimages are shown in figure 1(d). These subimages may contain more than one connected component. For instance, the second subimage has two connected components. The second subimage of figure 1(d) that is replicated in figure 1(e) has a lower modifier. We now separate the lower modifier and extract the vertically separate subimages. The resulting subimages are shown in figure 1(f). However, the subimage shown in figure 1(g) contains two consonants that are touching each other. Such a consonant pair is referred to as *conjunct*, *touching characters* or *fused characters*. There are over 100 conjuncts currently in use in Devanagari script. An optical character recognizer (OCR) can treat conjuncts as a unit or segment it to yield the constituent consonants. Since conjunct is a logical composition of its constituent consonants, it may not be logical to treat it as a unit. We segment all conjunct images into their constituent consonant images. The segmentation of the subimage of figure 1(g) is shown in figure 1(h).

The segmentation process that extracts constituent symbols images from a word for Devanagari script thus needs to perform the following tasks:

1. Remove the header line.

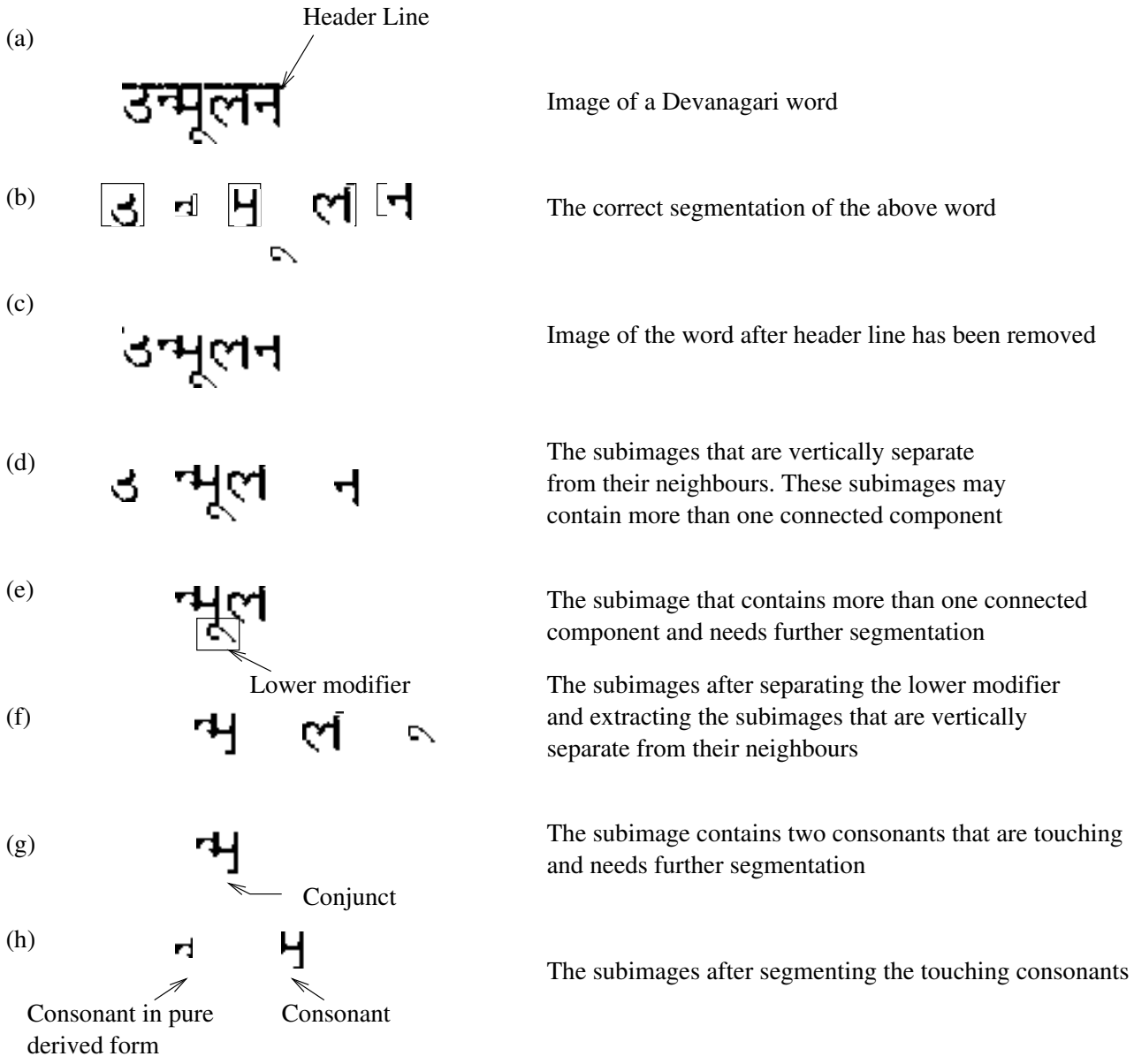


Figure 1: Image of a Devanagari word and its segmentation (Please refer to section 3 for the discussion.)

2. Extract subimages that are vertically separate from their neighbours. These subimages may contain more than one connected component.

3. Select the subimages that need further segmentation due to the presence of lower modifiers. A selection criterion is required for selecting these.

We describe our algorithms for tasks 1-3 in section 5.

4. Separate the lower modifiers from the subimages selected in step 3.

We describe our algorithms for task 4 in section 8.

5. Select the subimages that contain conjuncts or shadow characters. A selection criterion is required for selecting these.

We describe our algorithms for task 5 in section 5.

6. Segment the conjunct subimages selected in step 5 into constituent consonant subimages.

We describe our algorithms for task 6 in section 6.

7. Segment the subimages of shadow characters selected in step 5 into constituent character subimages.

We describe our algorithms for task 7 in section 7.

One may notice that the nature of segmentation required for Devanagari script is different from that of Roman script. In Devanagari script, about 5 percent of the characters are conjuncts. The percentage of the consonants that have lower modifiers is also about 5 percent. In Roman script, ink spread causes merging/touching of characters. Whereas, every Devanagari document, including the ones with absolutely no ink spread, will have about 10 percent touching characters.

Our segmentation process consists of two distinct stages. In the first stage, a preliminary segmentation is performed that executes line and word segmentation followed by tasks 1 and 2. This process leads to the isolation of subimages corresponding to characters that are vertically separate from their neighbours. These subimages may contain more than one connected component. Statistical information about height and width of the subimages obtained by the preliminary segmentation is used to mark the subimages that need further segmentation, i.e. tasks 3 & 5. Some of the marked character boxes may contain characters under *shadow* of each other. A character is said to be under the shadow of another character when the characters do not physically touch each other but it is not possible to separate

them merely by drawing a vertical line. Some of the image boxes contain lower modifiers and some may contain conjuncts. All of these image boxes are segmented in the second stage, i.e. tasks 4 & 6 of the list. In our present OCR system, the second stage is invoked only after the classification process has tried to classify the image boxes and has failed to classify them into any of the known classes. This type of strategy is referred to as loose coupling between segmentation and classification [2]. Our approach relies heavily on the structural properties of the individual characters and on observation about the nature of the characters that touch and fuse, i.e. 10 percent of the characters that are normally encountered in printed Devanagari text.

4 Some relevant features of Devanagari script from OCR viewpoint

Devanagari script has about 11 vowels and 33 consonants. The vowels and the consonants are shown in figure 2(a) and figure 2(d) respectively. One may notice the significant difference in the dimensions of these characters. In English as well as in Hindi, the vowels are used in two ways:

1. They are used to produce their own sounds. For instance, in word *insurance* in English, the letter *i* is used for producing its own sound.

The vowels shown in figure 2(a) are used for this purpose in Devanagari.

2. They are used to modify the sound of a consonant. For instance, in word *his* in English, the letter *i* is used for modifying the sound of preceding consonant *h*.

However, instead of juxtaposing the vowel to modify the sound of the preceding consonant as in case of English, a different mechanism is used in Devanagari. There is a symbol corresponding to each vowel that we refer to as modifier symbol. The modifiers symbols corresponding to the vowels are shown in figure 2(b). In order to modify the sound of a consonant, we attach an appropriate modifier in an appropriate manner to the consonant. In English, one can write more than one vowel following a consonant, i.e. *oo, ie, au, ee*. In Devanagari, only one vowel modifier can be attached to a consonant at any time. The vowel set and the corresponding modifier set is richer in Devanagari. It contains single vowels and corresponding modifiers for producing sounds corresponding to English vowel sequences *ea,*

oo, ie, au, ee etc. Each modifier has been attached to the first consonant of the script (see figure 2(c)). A visual inspection of figure 2(c) reveals that some of the modifier symbols are placed next to the consonant (core modifiers), some above (top modifiers) and some are placed below (lower modifiers) the consonant. Some of the modifiers contain a core modifier and a top modifier, the core modifier is placed before or next to the consonant; the top modifier is placed above the core modifier.

In Devanagari, a consonant is pronounced as a sequence of its own sound and the implied sound corresponding to vowel . Whenever we want to write a consonant that is deprived of the implied vowel sound, we use the *pure form* of the consonant. Devanagari script has a pure form for most of the consonants. The pure form of each consonant is shown in figure 2(e). For the consonants that do not have a pure forms, the same effect is achieved by attaching the special symbol (called *halant*) below the consonant. Whenever, the last character of a word is a consonant without any vowel sound, the script rules require that we write the consonant and attach *halant* even if the pure form of the consonant exists. A consonant in pure form always touches the next character, yielding conjuncts, touching characters, or fused characters. Figure 2(f) shows the conjuncts formed by writing pure form consonants followed by consonant . We can use almost any consonant in place of and write over 100 conjuncts.

4.1 Composition of Characters and Symbols for Writing Words

A horizontal line is drawn on top of all characters of a word that is referred to as the header line or *shirorekha*. It is convenient to visualize a Devanagari word in terms of three strips: a core strip, a top strip and a bottom strip. The core and top strips are separated by the header line. Figure 3 shows the image of a word that contains five characters, two lower modifiers and a top modifier. The three strips and the header line have been marked. No corresponding feature separates the lower strip from the core strip. The top strip has top modifiers and the bottom strip has lower modifiers whereas the core strip has the characters and core modifier ‘ ’. If no consonant of a word has a top modifier, the top strip will be empty. Similarly, if no character of a word has a lower modifier, the bottom strip will be empty. It is possible that either of the bottom or top strips may be present or both may be

(a) Vowels

(b) Modifier Symbols corresponding to the above vowels

(c) The modifier symbols have been attached to the first consonant to indicate their placing

(d) Consonants

(e) Pure Form of Consonants

(f) Some sample Conjuncts

Figure 2: Characters and Symbols of Devanagari Script.

Figure 3: Three strips of a Devanagari word

Figure 4: Sample Hindi text written in Devanagari Script.

present.

A few sentences written in Hindi language in Devanagari script are presented in figure 4. Sample text lines show the placing of the header lines and modifiers. Some of the conjuncts are also used in this text.

4.2 Script Composition Grammar

It is evident form the above discussion that word formation in Devanagari (as in other Indian scripts) follows a definite script composition rule for which there is no counterpart in an English text. A simplified Devanagari script composition grammar in Backus-Naur Form (BNF) is summarized in the figure 5.

$$< word >:= < composite\_char. >^{+}(\text{"}shiro\_rekha\text{"})$$

$$< composite\_char. >:= < vowel >^{*} | < conjunct >^{*}$$
$$| < conjunct >^{+} < \text{"}matra\text{"} >$$

$$< conjunct >:= < consonant >$$
$$| < consonant >< Reph >$$
$$| < consonant >< diacriticalmarks >$$
$$| < pure\_consonant >^{*} < conjunct >^{+}$$

Figure 5: A simplified Devanagari script composition grammar in BNF

5 Preliminary Segmentation of Words

Before we describe the preliminary segmentation of words, let us first define the following two operators:

1. **Definition 1: Vertical Projection:** For a binary image of size H * W where H is the height of the image and W is the width of the image, the vertical projection has been defined by [(1)] as

   $$VP(k), k = 1, 2, \ldots, W$$

   This operation counts the total number of black pixels in each vertical column.

2. **Definition 2: Horizontal Projection:** For a binary image of size H * W where H is the height of the image and W is the width of the image, the horizontal projection is defined as

   $$HP(j), j = 1, 2, \ldots, H$$

   This operation counts the total number of black pixels in each horizontal row.

The preliminary segmentation consists of the following five steps:

Step i: Locate the text lines A text line is separated from the previous and following text lines by white space. The line segmentation is based on horizontal histograms of the document. Those rows, for which $HP[j]$ is zero; $j = 1, 2, \ldots, H$; serve as delimiters between successive text lines.

Step ii: Locate the words The segmentation of the text line into words is based on the vertical projection of the text line. A vertical histogram of the text line is made and white space are used as word delimiter.

Step iii: Locate the header line After extracting the subimages corresponding to words for a text line, we locate the position of the header line of each word. Coordinates of the top-left corner are (0,0) and bottom-right corner are (W, H) where H is the height and W is the width of the word image box. We compute the horizontal projection of the word image box. The row containing maximum number of black pixels is considered to be the header line. Let this position be denoted by *hLinePos*. Figure 6(a) shows image of a word and figure 6(b) shows its horizontal projection. The row that corresponds to the header line has been marked as *hLinePos*.

Step iv: Separate character/symbol boxes of the image below the header line: To do this, we make vertical projection of the image starting from the *hLinePos* to the bottom row of the word image box. The columns that have no black pixels are treated as boundaries for extracting image boxes corresponding to characters. Figure 6(c) shows the vertical projection. The columns corresponding to white space between successive characters have been marked. The extracted subimages have been shown in figure 6(d).

Step v: Separate symbols of the top strip To do this, we compute the vertical projection of the image starting from the top row of the image to the *hLinePos*. The columns that have no black pixels are used as delimiters for extracting top modifier symbol boxes. Figure 6((e) shows the vertical projection. The extracted subimages have been shown in figure 6(f).

Our selection criteria for marking the subimages that need further segmentation due to the possible presence of lower modifiers is based on statistical analysis of height of the subimages extracted from the word image below the header line of the entire text line obtained after preliminary segmentation. We check the height of all the image boxes corresponding to characters extracted from a text line and denote the maximum height as *MaxCharHt*. The

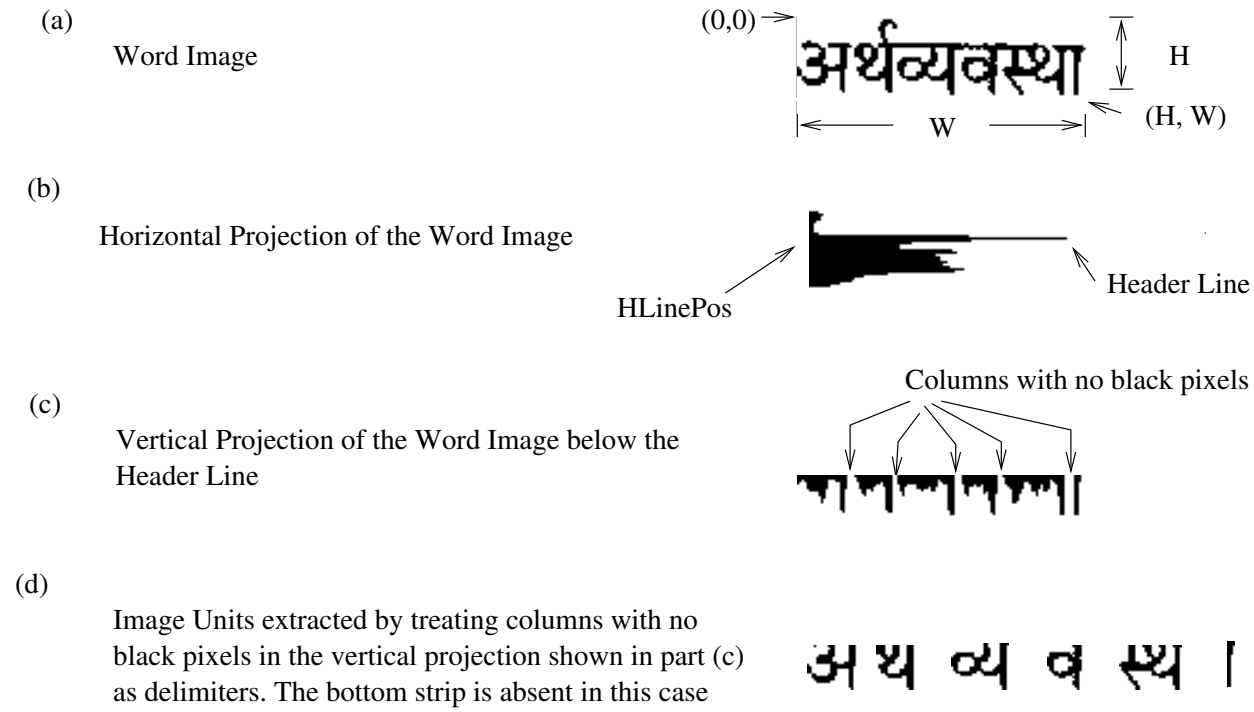


Figure 6: A Devanagari word image and its preliminary segmentation; (a) Word Image; (b) horizontal projection of the Word Image; (c) vertical projection of the word image below the header line; (d) Image units extracted; (e) vertical projection of the word image above the header line; (d) Image units extracted from the top strip.

image boxes are divided into three bins. The first bin contains those image boxes whose height is 80 percent or more of the *MaxCharHt*. The second bin contains the image boxes whose height is less than 80 percent of *MaxCharHt* but more than 64 percent of *MaxCharHt*. The remaining image boxes go in the third bin. The number of image boxes in each bin is counted. The bin that contains maximum image boxes becomes the bin that contains image boxes that need no further segmentation. We find the average height of an image box in this bin and denote it as *threshold character height*. All image boxes that have height more than the *threshold character height* are marked as image boxes that may need further segmentation due to the presence of lower modifiers. These figures of 80 percent and 64 percent correspond to the ratio of heights of three zones in Devanagari.

In the same manner, we check the width of all the image boxes corresponding to characters extracted from a text line and denote the maximum width as *MaxCharWd*. The image boxes are divided into three bins. The first bin contains those image boxes whose width is 80 percent or more of the *MaxCharWd*. The second bin contains the image boxes whose width is less than 80 percent of *MaxCharWd* but more than 64 percent of *MaxCharWd*. The remaining image boxes go in the third bin. The number of image boxes in each bin is counted. The bin that contains maximum image boxes becomes the bin that contains image boxes that need no further segmentation. We find the average width of an image box in this bin and denote it as *threshold character width*. All image boxes that have width more than the *threshold character width* are marked as image boxes that may need further segmentation as they may contain shadow characters or conjuncts. The preliminary segmentation and threshold computation constitutes pass-I of the segmentation process.

6 Segmentation of Conjunct/Touching Characters

An image box marked for further segmentation due to its width may contain either a conjunct or a shadow character pair. An outer boundary traversal is initiated from the top-left most black pixel of the image. During the traversal, the right most pixel visited will not be on the right boundary of the image box if the image box contains a shadow character pair. If the image contains a conjunct, the right most pixel visited will be on the right boundary of the image box. We discuss the shadow character pair segmentation in section 7. In this section,

end bar characters

middle bar characters

non-bar characters

Figure 7: Three classes of core characters based on the vertical bar feature

we discuss segmentation of a conjunct image into subimages corresponding to its constituent characters.

A conjunct is segmented vertically at an appropriate column to yield the constituent characters. In order to locate this column, we make use of two structural properties of Devanagari characters that are explained in next two subsections.

6.1 Vertical Bar and its Location:

Devanagari characters can be divided into three groups based on the presence and position of vertical bars, namely: *no bar characters, end bar characters* and *middle bar characters*. Some of the characters belonging to each of these classes are shown in figure 7. A vertical bar does not occur at the left end of a character. The position of the vertical bar is the left most column where number of black pixels is 80 percent or more of the character height. We divide the character image in three equal vertical zone and compute vertical projection for each zone. The column that contains desired number of pixels corresponds to the vertical bar column.

6.2 Continuity of the Horizontal Projection:

We explain this property with the help of figure 8. Figure 8(a) shows images of three Devanagari characters - a middle bar character, a no bar character, and an end bar character. The horizontal projections of the character images are shown in figure 8(b). We delete the vertical bar from each of the character image, if it is present, and the image on the right of the vertical bar. The resultant character images are shown in figure 8(c) and the corresponding

horizontal projections are shown in figure 8(d). Notice that every row in the horizontal projections in each case has at least one black pixel. Please recall that each time, the image is enclosed by a minimum inscribing rectangular box. The horizontal projection is made using only the pixels in this box. Let us now chop a few columns from the left of the character images of figure 8(c). The resultant images are shown in figure 8(e) and the corresponding horizontal projections are shown in figure 8(f). Notice that some of the rows in the horizontal projections of figure 8(e) are now without any black pixels. By inspecting the horizontal projection of the character image, from which the vertical bar and the image on the right of the vertical bar has been removed, we can find out if the character image has been chopped on the left side. In other words, when we approach the segmentation column of the conjunct from its right in steps of one column, we can find out if the left boundary of the second constituent character has been reached. Instead of making the horizontal projection, we make a *collapsed horizontal projection* of the image and check for its *continuity*, both of that we define next. We also define *pen width*.

Definition 3: Collapsed Horizontal Projection: The collapsed horizontal projection is defined as

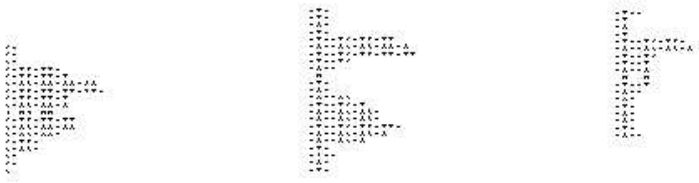
$$CHP(k) = \begin{cases} 1 & \text{If there exists an } i, i = 1, \ldots, W \\ & \text{such that Image[k, i] = 1} \\ 0 & \text{otherwise} \end{cases}$$

H and W are the height and width of the Image respectively and $k = 1, 2, \ldots, H$. CHP(k) denotes the presence of at least one black pixel in the horizontal row k. This operator is cheaper than the horizontal projection operator since it stops scanning the image the moment a black pixel is hit.

Definition 4: Continuity of the Collapsed Horizontal Projection: We scan the collapsed horizontal projection $CHP(k)$ in the sequence $CHP(1), CHP(2), \ldots, CHP(H)$. Let the position of first row for which $CHP(k)$ is 1 be R_1. We continue the scan and find the position of the subsequent row for which $CHP(k)$ is 0. Let this position be R_2. We continue the scan and find the position of the subsequent row for which $CHP(k)$ is 1. Let this position be R_3. We will always find R_1. For R_2 and R_3 the following three possibilities exist:

(a) Image of three Devanagari characters:
Middle Bar character No Bar characters End Bar characters

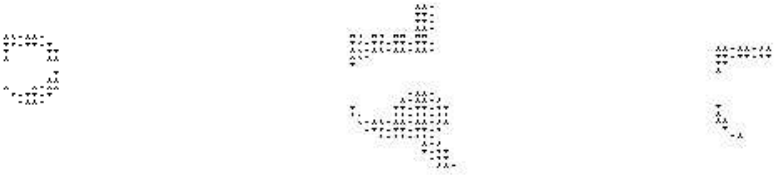
(b) Horizontal Projection of the Character Images of (a)



(c) Images after the vertical bar and image to its right has been removed

(d) Horizontal Projection of the Character Images of (c)

(e) Images of (c) after they are chopped from the left side



(f) Horizontal Projection of the Chopped Characters of (e)

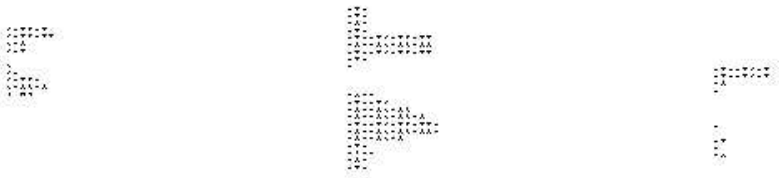


Figure 8: Continuity of Horizontal Projection of Devanagari Characters

1. R_2 and R_3 both do not exist;

2. R_2 and R_3 both exist; or

3. R_2 exists but R_3 does not exist.

The collapsed horizontal projection $\{CHP(k), k = 1, 2, \ldots, H\}$ has no discontinuity if R_2 and R_3 both do not exist or R_2 exists but R_3 does not exist. The difference $R_2 - R_1$ is referred to as the height of the collapsed horizontal projection.

Definition 5: Pen Width: In Devanagari characters, the pen width is same as the thickness of the header line. The thickness of the header line is obtained during the preliminary segmentation process from the horizontal projection of the words (see figure 6).

6.3 Locating the Segmentation Column:

Now we present our algorithm for locating the segmentation column. We locate a segmentation column, say C1, by examining the right part of the conjunct image in step 1 explained below. We locate another segmentation column, say C2, by examining the left part of the image in step 2 explained below. Then columns C1 and C2 are co-related to yield the final segmentation column C.

Step 1. Locate segmentation column C1: Let us say, the conjunct image has the left and the right co-ordinates - CONJ_TOP and CONJ_BOTTOM. We compute height of the conjunct image which is (CONJ_BOTTOM - CONJ_TOP), referred to as CONJ_HEIGHT. Recall that the coordinates of upper-left corner are (0,0). We denote left_one_third and middle column positions of the conjunct image as CONJ_LEFT_ONETHIRD and CONJ_MID respectively. Figures 9(a) shows image of the conjunct. We have marked CONJ_TOP, CONJ_BOTTOM, CONJ_RIGHT and CONJ_LEFT. We have also marked columns corresponding to CONJ_LEFT_ONETHIRD and CONJ_MID. These steps have been presented in the form of an algorithm in figure 11.

Let us now delete the vertical bar from the right end of the conjunct image. The image after deleting the vertical bar is shown in figure 9(b). Let us refer to the right boundary of this image as CONJ_RIGHT. The column that is one pen width to the left of CONJ_RIGHT is our candidate segmentation column. Let this column be C1. We have also marked the

column C1 in figure 9(b). We inscribe the image between C1 and CONJ_RIGHT and compute the collapsed horizontal projection. Let this collapsed horizontal projection be called CHP(k). These steps are presented as an algorithm in figure 11. If CHP(k) has no discontinuity and the height of CHP(k) is more than 1/3rd of CONJ_HEIGHT, C1 is our segmentation column. Otherwise, we shift C1 to the left by one column and recompute CHP(k). After a few iterations, CHP(k) will have desired properties and we would have found our segmentation column C1. Figure 9(c)i - iv depict the iteration process. In each figure, we have shown the image box under consideration, corresponding CHP(k) as well as R1, R2 and R3. The CHP(k)shown in figure 9(c)iv has no discontinuity and its height is more than 1/3rd of CONJ_HEIGHT. Therefore, the process is terminated and no more iterations are performed.

Step 2. Locate segmentation column C2: We put pure consonants in two classes based on their height:

(i) Height of the pure consonant is 80 percent or less than the corresponding consonant: Characters of this class are: . This class is referred to as class H_1.

(ii) Height of the pure consonant remains unchanged: Characters of this class are: . This class of characters is referred to as class H_2.

The classes H_1 and H_2 as outlined above are used for identifying the column C2. First, we inscribe the image between CONJ_LEFT and CONJ_LEFT_ONETHIRD and compute collapsed horizontal projection of the image in the box. Let this collapsed horizontal projection be CHP2(k).

If the height of the CHP2(k) is more than 80 percent of CONJ_HEIGHT, the first constituent character of the conjunct belongs to class H_2. Otherwise, it belongs to class H_1. The image of our example conjunct is shown in Figure 10(a). We have marked columns CONJ_LEFT and CONJ_LEFT_ONETHIRD. The collapsed horizontal projection, referred to as CHP2(k), of the image inscribed by columns CONJ_LEFT and CONJ_LEFT_ONETHIRD is shown in figure 10(b). The height of CHP2(k) is less than 80 percent of CONJ_HEIGHT. Therefore, the subimage on the left side belongs to class H_1. After identifying the class of the left subim-

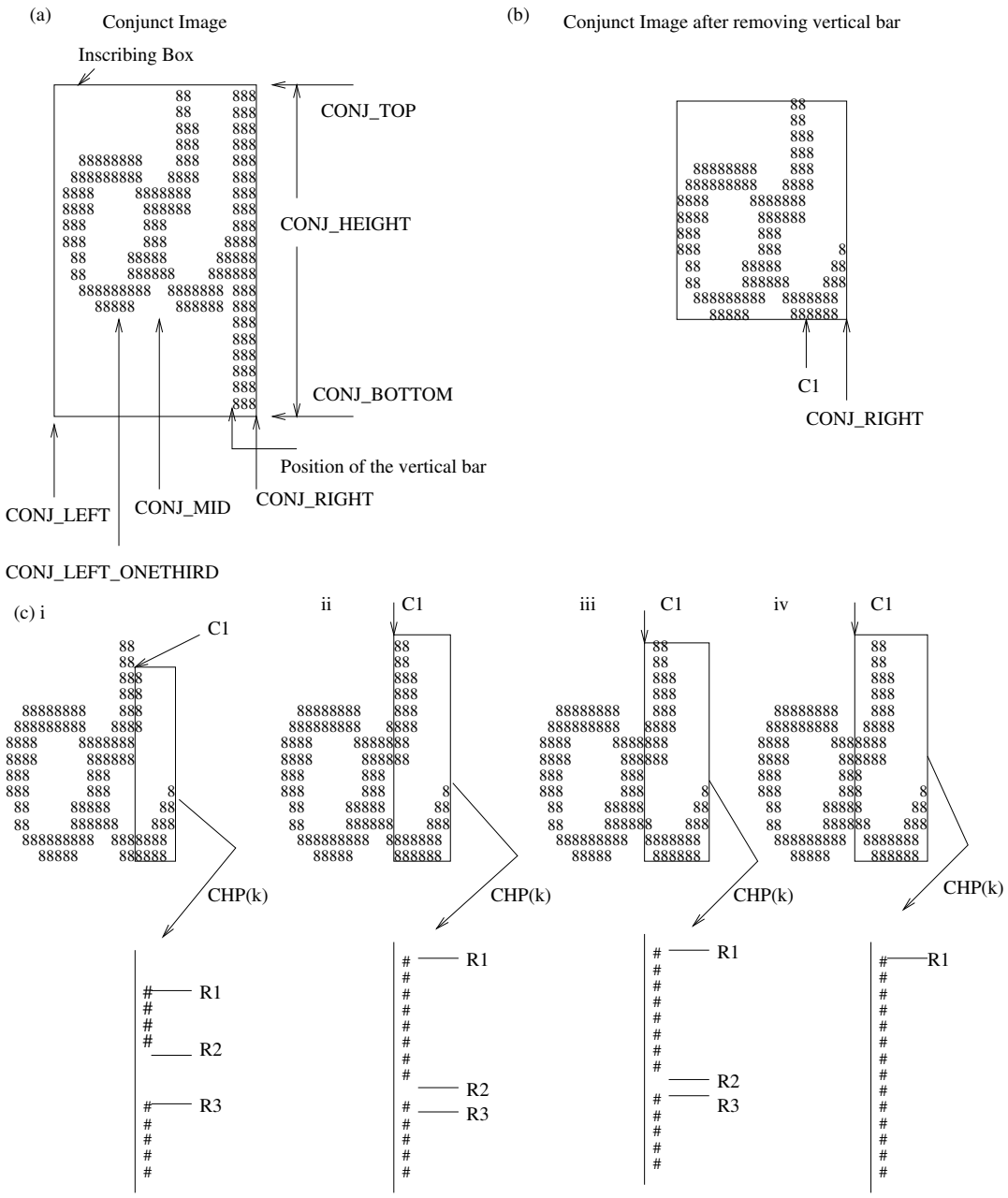


Figure 9: Locating column C1; (a) An example conjunct image that has been labeled to show its boundaries, inscribing box, CONJ_HEIGHT, CONJ_LEFT_ONETHIRD, CONJ_MID and vertical bar column; (b) The conjunct image after vertical bar has been removed; (c)i-iv: The image inscribed between the columns C1 and CONJ_RIGHT, corresponding CHP(k) that has been labelled by R1, R2 and R3.

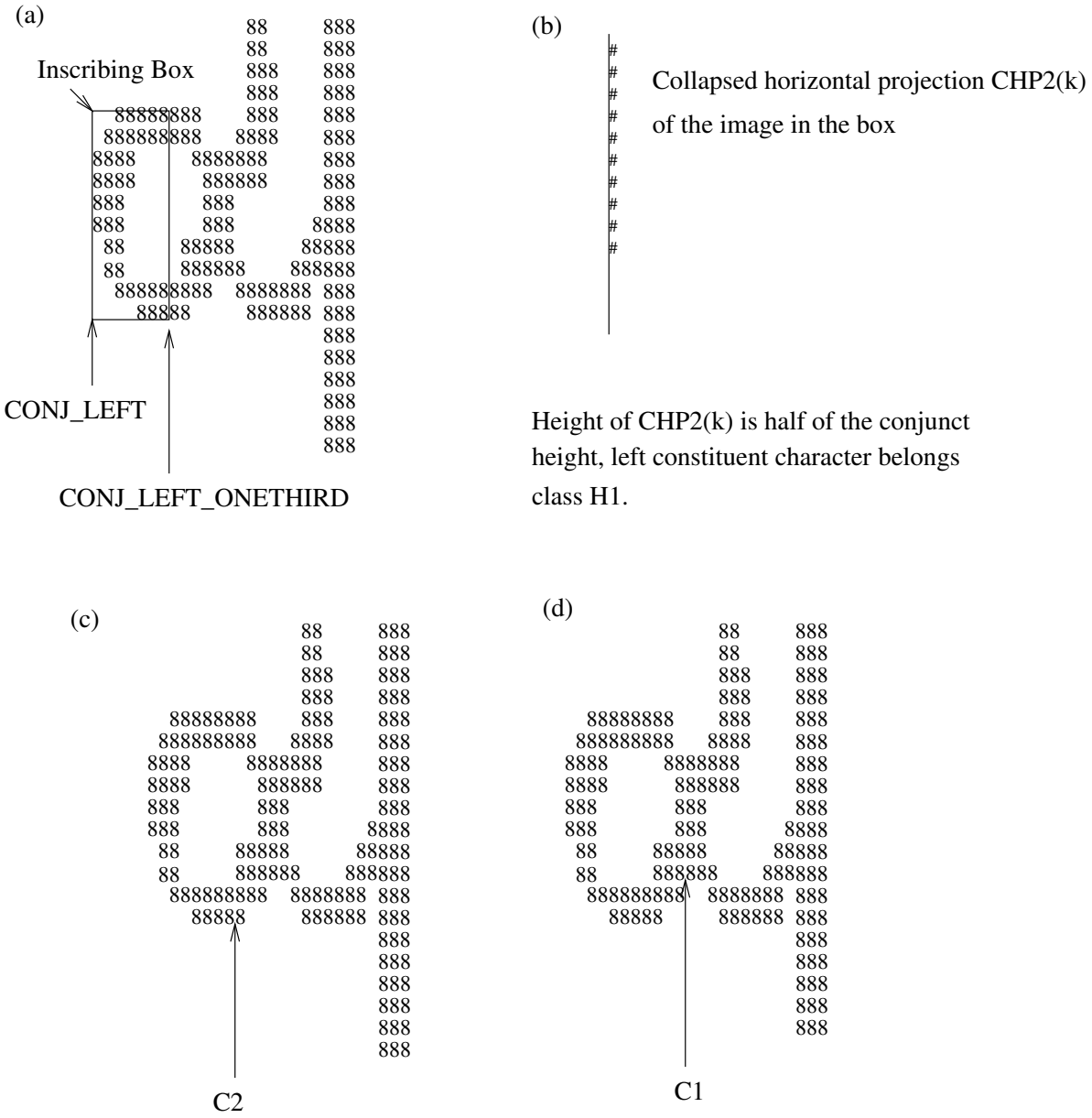


Figure 10: The image of the conjunct and column C2; (a) image inscribed by CONJ_LEFT and CONJ_LEFT_ONETHIRD is used for computing the collapsed horizontal projection for identifying the class; (b) Horizontal Collapsed Projection of the image in the box; (c) The column C2; (d) The column C1.

age of the conjunct, we set C2 to CONJ_LEFT_ONETHIRD. We compute the height of the image inscribed by CONJ_LEFT and C2. If the height is less than 1/3rd of CONJ_HEIGHT for class H_1 characters, we move C2 to the right in steps of one column provided the pixel strength of the new column C2 is not more than the pixel strength of the present column C2. We have marked column C2 in figure 10(c) for our example conjunct. The height of the image is more than 1/3rd of CONJ_HEIGHT and we stop the process.

For class H_2 characters, we inscribe the image between CONJ_LEFT and CONJ_LEFT_ONETHIRD. We then compute the vertical projection of the image in this box and locate the right most column with maximum number of black pixels. Let this column be C2.

We move C2 in steps of one column to the right as long as the pixel strength of the next column is not more than the pixel strength of the present column. C2 is not allowed to go beyond CONJ_MID. These steps have been presented in form of an algorithm in figure 11.

Relating C1 and C2: We now make an attempt to relate C1 and C2. While comparing C1 and C2, the following three possibilities are encountered:

(a) C1 is less than C2: both the segmentation columns are ignored and no segmentation is done. This situation occurs when a character does not require segmentation and has been wrongly marked for further segmentation.

(b) C1 and C2 are almost same: If C1 and C2 are the same or the difference between them is less than the *pen width*, the segmentation column C1 is moved to the left by the difference. Column C1 is our segmentation column C.

(c) The difference between C1 and C2 is more than the pen width. This situation occurs when the image has more than two characters, we can probably extract the first constituent character using C1 and then attempt to segment the remaining image. However, this remains to be done in future.

7 Segmentation of Shadow Character

Two adjacent characters are in shadow of each other if they do not touch each other and cannot be separated by a single vertical line due to the overlapping region. In order to extract the subimages corresponding to each character, we locate the left most black pixel

```
Algorithm: Segment Conjunct
Input:
  conjunct image
  CONJ_LEFT    /* left boundary of the conjunct image */
  CONJ_RIGHT   /* right boundary of the conjunct image */
  CONJ_TOP   /* top boundary of the conjunct image */
  CONJ_BOTTOM   /* bottom boundary of the conjunct image */
  PENWIDTH: thickness of the header line
Output:
  C1: left boundary of the second constituent character
  C2 :  right boundary of the first constituent  character
Procedure:
  0. CONJ_LEFT_ONETHIRD = CONJ_LEFT + (CONJ_RIGHT - CONJ_LEFT)/3;
     CONJ_MID = CONJ_LEFT + (CONJ_RIGHT - CONJ_LEFT)/2;
     CONJ-HEIGHT = CONJ_BOTTOM - CONJ_TOP
1. Remove vertical bar if any and image to its right from the second half
      of the conjunct image.  Let the new right boundary be CONJ_RIGHT
2. Initial guess for the segmentation column is
     C1 = CONJ_RIGHT -  PENWIDTH
3. Inscribe the image between C1 and MOD_RIGHT
     and compute  its collapsed horizontal projection: CHP(k)
4. If ( CHP(k) has no discontinuity and height of CHP(k) > 1/3 of CONJ_HEIGHT )
        the C1 is the segmentation column.
        go to step 5
   else
        C1 =  C1 - 1;
        If ( C1 > CONJ_RIGHT_ONETHIRD )
          go to step 3
        else
          go to step 5
5. Inscribe the image between CONJ_LEFT and CONJ_LEFT_ONETHIRD
     and make its collapsed horizontal projection: CHP2(k)
6. If ( height of CHP2(k) >= .8 * CONJ_HEIGHT )
       CLASS = H2;
     else
       CLASS = H1;
```

Figure 11: Algorithm for locating columns C1, C2 and the segmentation column C (contd).

```
7. If ( CLASS = H1 )
   7.1 C2 = CONJ_LEFT_ONETHIRD + 1
   7.2 Inscribe the image between CONJ_LEFT and C1
         and make its collapsed horizontal projection: CHP3(k)
   7.3
       7.3.1 If ( height of CHP3(k) <= .3 * CONJ_HEIGHT  and
                                  C2 < CONJ_MID )
                C2 = C2 + 1
              go to step 7.2
       7.3.2 If ( height of CHP3(k) >= .3 * CONJ_HEIGHT  and
                                  C2 < CONJ_MID )
         7.3.2.1 If ( number of black pixels in C2 column <=
                      number of black pixels in C2 + 1 column )
                    C2 = C2 + 1
                    If ( C2 < CONJ_MID )
                      go to step 7.3.2.1
                    else
                      go to step 7.4
   7.4 C2 is the segmentation column
8. If ( CLASS = H2 )
   8.1 Locate the rightmost column in the image enclosed by
          CONJ_LEFT and CONJ_LEFT_ONETHIRD  that has
          maximum number of black pixels; let this column be C2
   8.2 C2 = C2 + 1
   8.3 If ( C2 < CONJ_MID )
       8.3.1 If ( number of black pixels in C2 column <=
                     number of black pixels in C2 + 1 column )
                C2 = C2 + 1
                If ( C2 < CONJ_MID )
                  go to step 8.3.1
                else
                  go to step 8.4
   8.4 C2 is the segmentation column
9. (a) If ( C1 -  C2 <= PENWIDTH)
          C = C1 - PENWIDTH
   (b) If ( C1 <  C2 )
          NO SEGMENTATION IS POSSIBLE
   (c) NO SEGMENTATION IS POSSIBLE
```

Figure 11: Algorithm for locating the columns C1, C2 and the segmentation column C

in the image, say P1. Figure 12(a) shows image of two characters that are in shadow. We have also marked the left most black pixel in the image. Next, we do an outer boundary traversal starting at pixel P1. The extreme left, right, top and bottom image points visited during the traversal form the inscribing box for the subimage of the first character of the pair. Figure 12(b) shows the inscribing box for the the first character. It is clear from the figure 12(b) that a small part of the image of the second character overlaps with the inscribing box for the first character. In order to extract the second subimage, another outer boundary traversal is initiated from a pixel, say P2 that is outside the first character image box. Figure 12(c) shows the pixel P2 that is the left most black pixel outside the box inscribing the first character subimage. The outer boundary traversal initiated from P2 yields the inscribing box for the second character subimage. Figure 12(d) shows the box inscribing the second character. During this traversal, the extreme left, right, top and bottom points that are visited in the territory of first image box form the overlap region. The overlap region is cleaned from each image box to obtain proper subimages. Figure 12(f) shows the extracted subimages.

8 Segmentation of Lower Modifiers

In section 5, we marked some of the image boxes for further segmentation suspecting the presence of lower modifiers based on the *threshold character height*. Devanagari script has characters of varying height. Character has almost same height as that is character with the lower modifier . Therefore, a character suspected to have a lower modifiers may not actually contain a lower modifier. Therefore, we need to carefully examine the image box for the presence of a lower modifier. When a lower modifier is placed below a core character, one of the three possible joining patterns is formed. These three joining patterns are:

i. Weak Joining A lower modifier below a middle or end bar character forms a weak join with the core character. Some of the non-bar character that have a small bar in the lower region also form a weak joining point with the lower modifier. Some examples are , , .

ii. Thick Joining There are some characters that touch the lower end of the core strip at

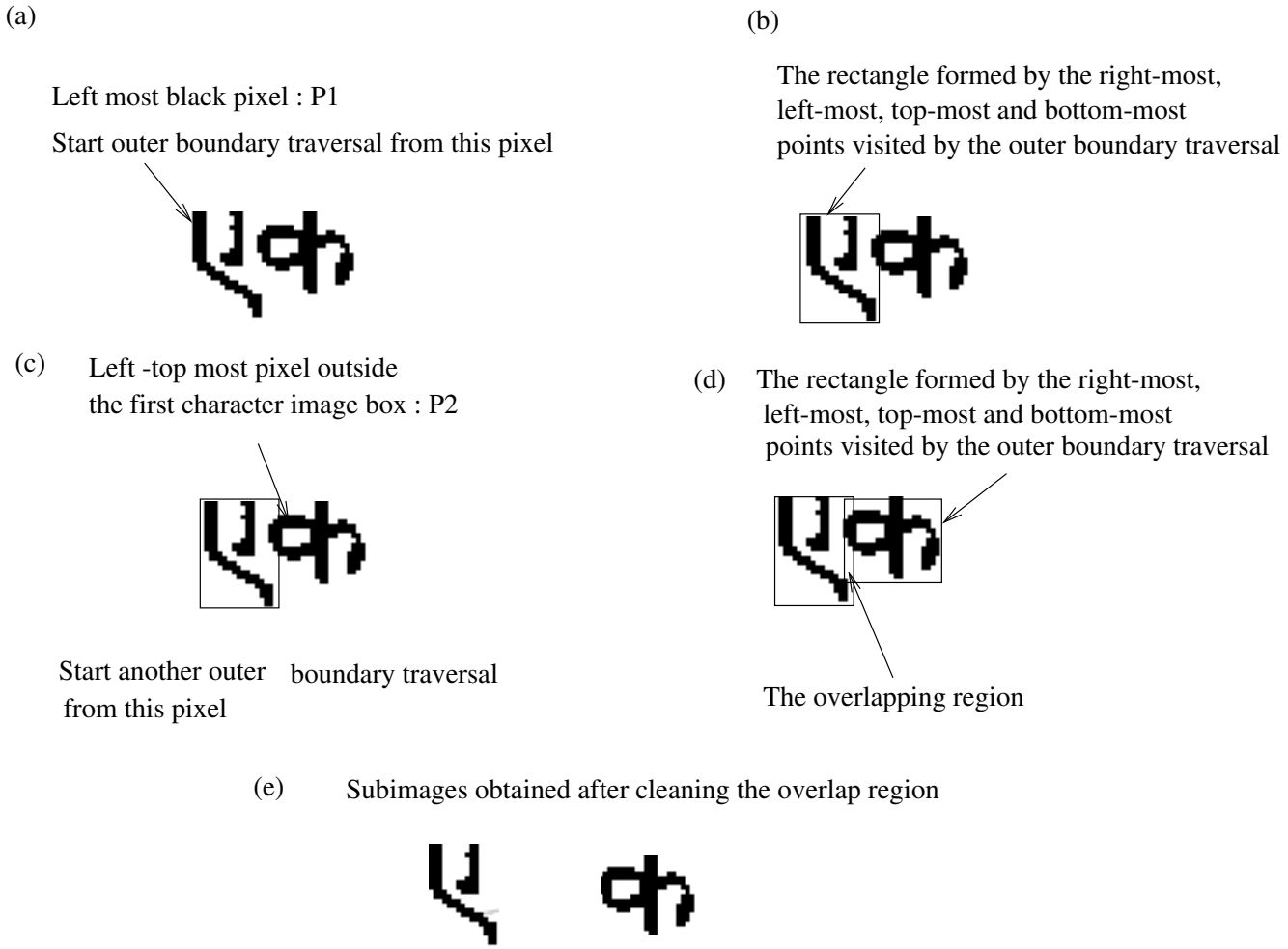


Figure 12: Two Characters in Shadow and their Segmentation; (a) The image of Characters in Shadow; Left most pixel is also marked (b) The box inscribing the image of first character; (c) Left most black pixel outside the first image box; (d) The box inscribing the second character; the overlapping region has also been marked; (e) the subimages for the constituent characters after cleaning the overlapping region.

more than one point. A lower modifier placed below such character forms a thick join with the core character, such as , .

iii. **Gap between the character and the modifier** Sometimes a modifier doesn't touch the core character at all. The lower modifier called 'nukta' (represented as a dot(.) in the bottom) usually does not touch the core character. Some examples are: etc.

The horizontal segmentation row for separating a lower modifier is below the core strip. Height of the core strip is estimated to be *threshold character height* (see section 5). We explore the rows below the core strip for a possible segmentation row. If we find a row that contains no black pixels, we use this row to separate the lower modifier. Otherwise, the row with minimum number of black pixels below the core strip is located. The pixels below this row are checked whether they have sufficient height and width to qualify for a lower modifier symbol by making a horizontal projection of the selected image. The height of a lower modifier is usually 1/5th or more of the character height. In case, enough pixels are present, we use the located row for segmenting the lower modifier. However, in case of a thick joining pattern, some adjustment is made based on the profile information of the joining region. An image that has been marked for further segmentation has been shown in figure 13(a). We have shown the inscribing boxes for the subimages corresponding to the core character and the lower modifier. The threshold character height has also been marked. The subimage inscribed by the box shown in figure 13(b) is used for making horizontal projection. The row that contains minimum number of black pixels has also been marked. We extract the subimages shown in figure 13(c) using this row. These subimages correspond to the core character and the lower modifier respectively.

The character is a long character and has a high frequency of occurrence in text. The image below the *threshold character height* resembles the lower modifiers *halant* very closely. Our lower modifier segmentation algorithm removes lower part of and puts it in a lower modifier box. The chopped character is essentially a new character and it is treated like a new character. However, this lower modifier does not make a syntactically valid character with character . Since the trainer and recognizer both use the same segmentation algorithm, character is always chopped. The chopped character is added to the set of known

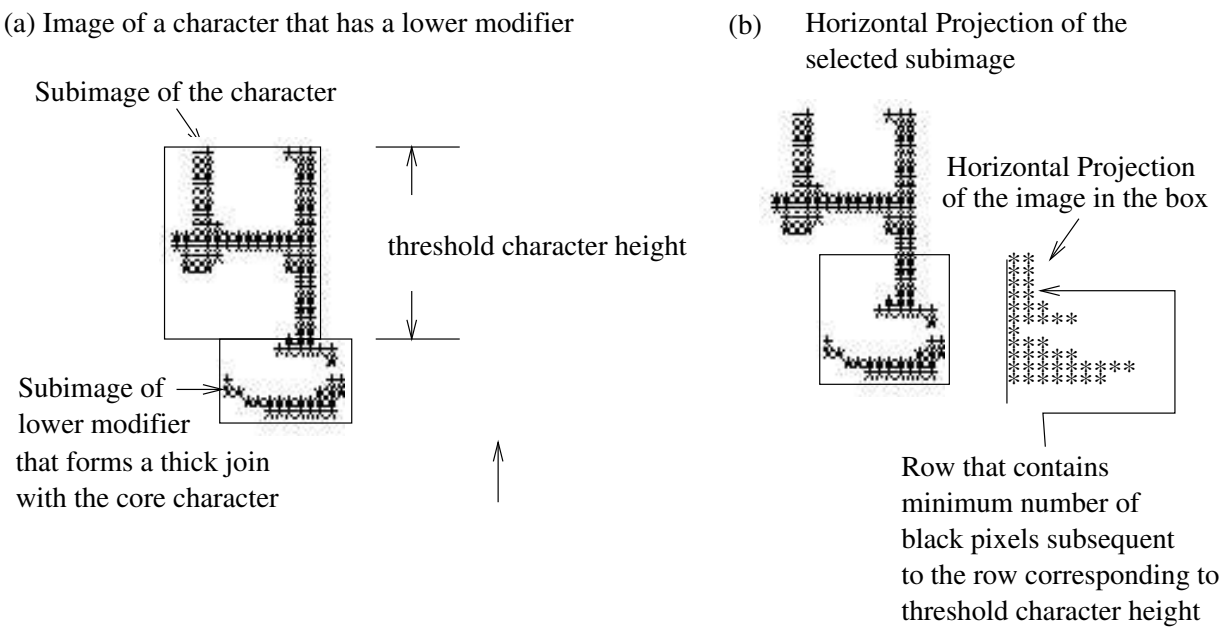


Figure 13: Lower Modifier Segmentation; (a) The image of a character containing a lower modifier; (b) Horizontal projection of the selected image area; (c) The subimages for the core character and the lower modifier.

Composite characters Constituent characters

Figure 14: Composite Characters not invoked for segmentation.

characters. The composition grammar is augmented to compose the chopped and lower modifier *halant* into character [9, 16].

9 Results

The structural segmentation algorithm has been tested on 18 document pages from two different magazines. The number of conjuncts in the test documents is about 5 percent. Out of all the conjunct and composite characters, about 90 percent conjuncts have been marked for further segmentation based on the output of the classification process (see [17, 18]) and the threshold width and height of the characters. During the testing, it was observed that sometimes a composite character was substituted by another Devanagari character. This happened when both constituent characters were relatively thin. The composite character and get mapped to character . Some of the composite characters that were not selected for segmentation are shown in figure 14. The recognition of the constituent characters obtained after segmentation of touching character is about 88 percent that matches with the overall performance of the system. There are approximately 12 percent substitution errors. The results of touching character segmentation are summarized in tables 1 and 2. However, the algorithm is capable of segmenting all the conjuncts of figure 14 when suggested to do so.

A sample text page and the text after recognition is shown in figure 15. The segmented image after preliminary segmentation and lower modifier segmentation are shown in figures 16 and 17 respectively. The image after conjunct segmentation and the OCR output after post-

FONT I

	total number of chars.	overall recognition	number of touching chars.	touching char. recognition	touching chars. substitution
Doc I	2229	2001	186	174	12
Doc II	1785	1605	84	69	15
Doc III	1500	1263	90	81	9
Doc IV	1419	1230	63	60	3
Doc V	1926	1653	99	84	15
Doc VI	2517	2259	117	90	27
Doc VII	2310	2019	141	114	27
Doc VIII	2679	2232	162	126	36
Doc IX	2412	2136	144	111	33
Doc X	2079	1764	86	72	14
Overall	20856	18162	1172	981	191
		87.08%	5.61%	83.70%	17.29%

Table 1: Performance of the system for Font I.

FONT II

	total number of chars.	overall recognition	number of touching chars.	touching char. recognition	touching chars. substitution
Doc I	2386	2122	98	78	20
Doc II	1740	1466	72	64	8
Doc III	2258	1930	128	112	16
Doc IV	2052	1834	90	78	12
Doc V	2038	1758	120	98	22
Doc VI	2136	1854	110	86	24
Doc VII	1820	1538	88	78	10
Doc VIII	1926	1857	96	84	12
Overall	16356	14359	802	678	124
		87.79%	4.90%	84.53%	15.46%

Table 2: Performance of the system for Font II.

The input text

Figure 15: The input text

The image after preliminary segmentation (lines 1-5)

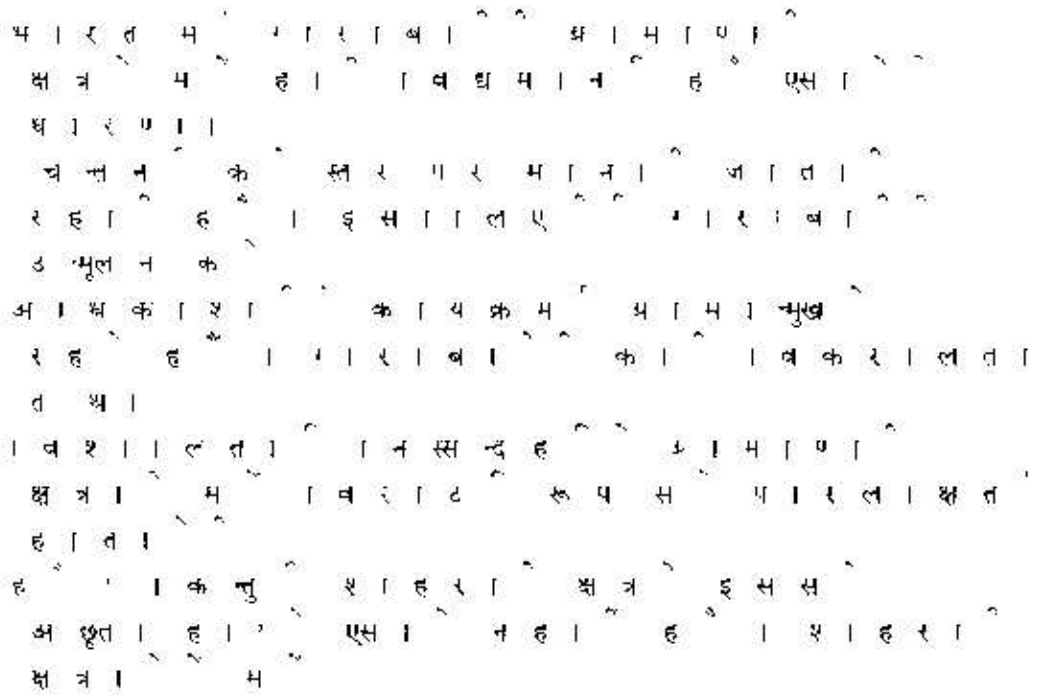


Figure 16: The image after preliminary segmentation (lines 1-5); lower modifiers and conjuncts have not yet been segmented.

The image after lower modifier separation (lines 1-5)

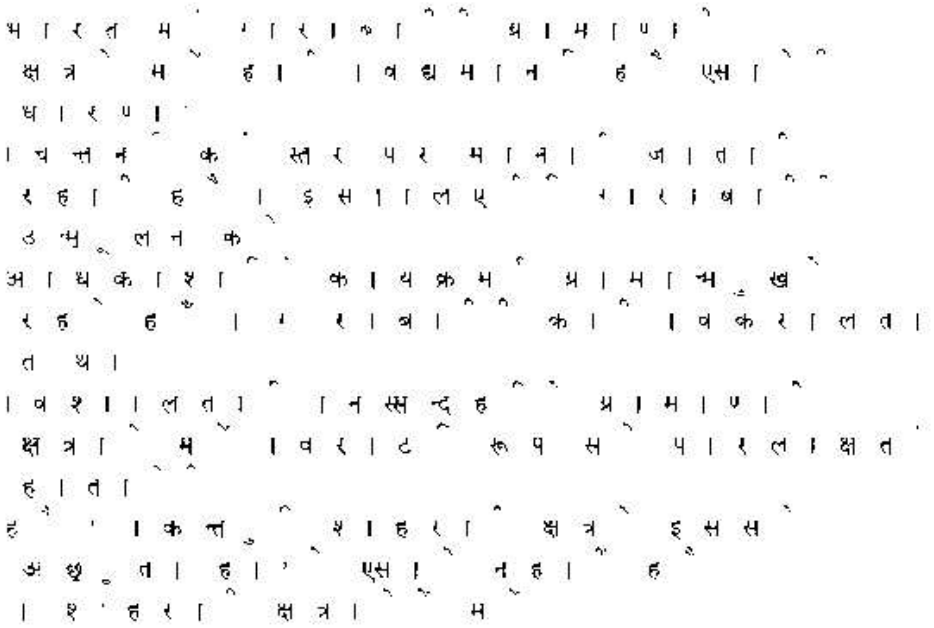


Figure 17: The image after lower modifier separation (lines 1-5); conjuncts have not yet been segmented.

The image after conjunct segmentation (lines 1-5)

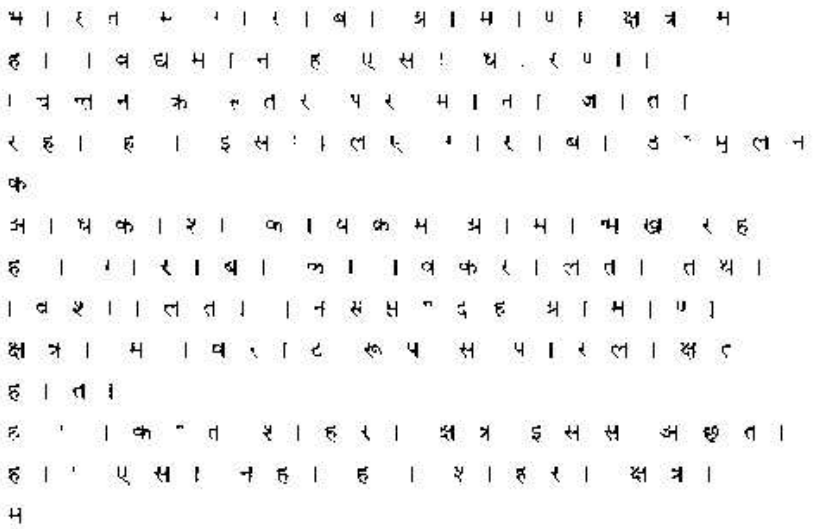


Figure 18: The image after conjunct segmentation (lines 1-5)

Output of OCR

{ }

Figure 19: The output of the classification process; the word has been underlined if the true word is the second or third choice; incorrect words are shown in { }

processing [19] are shown in figures 18 and 19.

10 Conclusion

In this paper, we have presented a complete method for segmentation of text printed in Devanagari script, a script used by more than 400 million people on the globe.

A preliminary segmentation process extracts the header line and delineates the upper-strip from the rest. This yields vertically separated character boxes, that may be conjuncts, touching characters , characters with lower modifier attached to it, shadow characters or a combination of these. In pass-I of the segmentation process, statistical information about these boxes is collected. In pass-II, this statistical information id used to select the boxes on which further segmentation is attempted. Besides separating the lower strip from the core strip the constituent character boxes in the conjunct are identified in the core strip. We extensively use the structural properties of the characters in the segmentation process. The presence and relative location of vertical line, and the nature of constituent pure consonant form in the conjunct, provide valuable clues about segmentation boundaries. Our segmentation approach is a hybrid approach, wherein we try to recognize the parts of the conjunct

that form part of a character class. A performance of 85 percent correct recognition has been achieved.

Devanagari script is a derivative of the ancient Brahmi script that is also the mother of all other Indian scripts. The methodology outlined here are equally applicable to many other scripts like Bangla, Asamiya, Gujarati and Gurumukhi that are similar in structure.

Acknowledgements

A part of this work was supported by Department of Electronics, Government of India.

References

[1] Su Liang, M. Shridhar and M. Ahmad, Segmentation of Touching Characters in Printed Document Recognition, *Pattern Recognition*, 27, pp. 825-840, 1994.

[2] R. G. Casey and E. Lecolinet, A survey of Methods and Strategies in Character Segmentation, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18, pp. 690-706, 1996.

[3] R. G. Casey and G. Nagy, Recursive segmentation and classification of composite character patterns, *Proc. 6th International Conference Pattern Recognition, Munich, Germany*, pp. 1023-1026, 1982.

[4] S. Kahan, T. Pavlidis and H. S. Baird, On the recognition of printed characters of any font and size, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9, pp. 274-287, 1987.

[5] Shuichi Tsujimoto and Haruo Asada, Solving Ambiguities in Segmenting Touching Characters, *Structured Document Image Analysis*, H. S. Baird et al., eds., Springer-Verlag, U.S.A., 1992.

[6] Jairo Rocha and Theo Pavlidis, A Shape Analysis Model with Applications to a Character Recognition System, *IEEE Transactions on PAMI*, 16(4) pp. 393-403, 1994.

[7] S.S. Marwah, S. K. Mullick and R. M. K. Sinha, Recognition of Devanagari characters using a hierarchical binary decision tree classifier, *IEEE International Conference on Systems, Man and Cybernetics*, October 1994.

[8] R. M. K. Sinha and H. N. Mahabala, Machine Recognition of Devanagari Script, *IEEE International Conference on Systems, Man and Cybernetics*, pp. 435-441, 1979.

[9] R.M.K.Sinha, Rule based contexual post-processing for Devanagari text recognition, *Pattern Recognition*, 20(5), pp. 475-485, 1987.

[10] I.K. Sethi and B. Chatterjee, Machine recognition of handprinted Devanagari Numerals, *Journal of Institution of Electronics and Telecommunication Engineers*, India, vol. 22, pp 532-535, 1976.

[11] J.C. Sant and S. K. Mullick, Handwritten Devanagari script recognition using CTNNSE algorithm, *International Conference on Application of Information Technology in South Asian Language*, February 1994.

[12] B.B. Chaudhuri and U. Pal, Skew Angle detection of Digitized Indian Script Documents, *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 19(2), pp. 182-186, Feb 1997.

[13] B.B. Chaudhuri and U. Pal, A Complete Printed Bangla OCR System, *Pattern Recognition*, 31(5), pp. 531-549, 1997.

[14] B.B. Chaudhuri and U. Pal, An OCR system to read two Indian language scripts: Bangla and Devanagari, *Proceedings of 4th International Conference on Document Analysis and Recognition*, pp. 1011-1015, 1997.

[15] Pan Bao-Chang, Wu Si-Chang and Yan Guang-Yi, A method of Recognizing Handprinted Characters, *Computer Recognition and Human Production of Handwriting*, Eds. R. Plamondon, C. Y. Suen and M. L. Simner, World Scientific Publ. Co., pp. 37-60, 1989.

[16] R.M.K.Sinha, Visual text recognition through contexual processing, *Pattern Recognition*, 21, pp. 463-479, 1988.

[17] R. M. K. Sinha and Veena Bansal, On Devanagari Document Processing, *IEEE International Conference on Systems, Man and Cybernetics, Vancouver, Canada* (1995).

[18] R.M.K. Sinha and Veena Bansal, On automating trainer for construction of prototypes for Devanagari text recognition, *Technical Report*, TRCS-95-232, I.I.T. Kanpur, India, (1995).

[19] V. Bansal and R.M.K. Sinha, Partitioning and Searching Dictionary for Correction of Optically-Read Devanagari Character Strings, *Proceedings of the international conference on Document Analysis and Recognition (ICDAR-99)*, Bangalore, India, pp. 653-656, 1999.