Multimedia Tools and Applications (2019) 78:19361–19386
https://doi.org/10.1007/s11042-019-7286-0

Handwritten multilingual word segmentation using polygonal approximation of digital curves for Indian languages

Deepika Gupta[1] · **Soumen Bag**[1]

Received: 13 July 2018 / Revised: 11 January 2019 / Accepted: 27 January 2019 /
Published online: 11 February 2019

Abstract
Multilingual Optical Character Recognition (OCR) is difficult to develop as different languages exhibit different writing and structural characteristics and it is very difficult to generalize their segmentation process. Character segmentation plays an important role in developing OCR for handwritten languages. The exactness of character segmentation is the integral factor of OCR. In this paper, we exploit this limitation and propose a approach based on the polygonal approximation of the word, which works on more than one Indian languages. This work depicts the novel approach for script independent character segmentation of handwritten text utilizing basic structural properties of the languages. Digitally straight line segments (DSS) of the word is obtained by applying Polygonal approximation to the word. The segmentation of character is language independent and works considerably with skew words as well. Experiments are carried out with four popular Indian languages, Hindi, Marathi, Punjabi, and Bangla. The average success rate for character segmentation of four languages is 90.07% which is satisfactory compared with other existing methods. We use shadow and cumulative stretch feature set with random forest, support vector machine (SVM), multi-layer perceptron (MLP), and convolutional neural network (CNN) classifiers for character recognition. On experimentation, it is observed that our proposed method provided good accuracy for character segmentation and recognition.

Keywords Character segmentation · Deep learning · Handwritten · Indian languages · Multilingual · OCR · Script independent

✉ Deepika Gupta
deepika.guptaa19@gmail.com

Soumen Bag
bagsoumen@gmail.com

[1] Department of Computer Science and Engineering, Indian Institute of Technology (ISM), Dhanbad 826004, India

1 Introduction

Optical character recognition (OCR) is the electronic conversion of the printed or handwritten scanned document into the machine understandable text. It is the process of changing the physical text information to digital form so that it can be edited, stored, or used for many other machine processes. The main advantage of this process is that it is helpful in entering the data from printed data records without human intervention. Basic OCR system includes various steps of pre-processing, feature extraction and classification.

Character segmentation is a pre-processing step in OCR. It is the process of separating the word image into individual character images. It is the fundamental and critical step of OCR. The performance of OCR degrades due to incorrect character segmentation. Segmentation of characters is a complex task in Indian scripts of cursive handwriting style. Character segmentation becomes difficult with the variation in writing styles, skew variability etc.

A large number of works are done on various scripts like Roman, Chinese, and Arabic [11, 15, 16, 24] etc. Various strategies in character segmentation are consolidated in an early study by Casey and Lecolinet [12]. Various techniques for character segmentation for Indian scripts are proclaimed in [34]. Methodologies used by researchers for segmenting Hindi and Bangla languages are united in [4]. A survey on recognition of Devanagari script is given in [23]. Work from the 1970s of printed and handwritten OCRs are consolidated in the survey. Bansal and Sinha [6] have proposed a two pass algorithm for segmenting Devanagari script. In the first pass, the word is segmented into easily separable characters. Separated characters are classified into composite and non-composite characters based on statistical information about the height and width of the characters obtained after the first pass. Characters classified into the category of composite characters are segmented in the second pass of the algorithm. Hanmandlu and Agrawal [18] have used structural properties of Hindi language for segmentation. Sarkar et al. [44] have proposed a non-linear fuzzy membership function for header line estimation. Then used a non-linear fuzzy function to identify the segmentation points on the header line. An adaptive Hindi OCR is proposed in [28]. In this, Hindi scripts are first identified from bilingual and multilingual documents based on structural properties and then character segmentation is applied on the identified word. A horizontal projection approach is used for dissecting the word into characters. Another work of handwritten Hindi text segmentation based on header line removal is reported in [36]. In this, the header line is detected by estimating the average line height and then the characters are segmented. Srivastava and Sahu [47] have also used the header line detection and removal approach for handwritten Hindi text segmentation. Ramteke et al. [41] have reported segmentation of Marathi handwritten text based on header line detection. One of the early research on Bangla character segmentation is based on recursive contour following [10]. Pal and Dutta [35] have used a concept based on water reservoir principle for character segmentation of Bangla. Bag et al. [3] have proposed a method based on vertex characterization of outer isothetic polygonal covers. Roy et al. [42] have presented a approach for skew detection, correction, and character segmentation. In this work, segmentation points are extracted on the basis of some patterns observed in the handwritten words. Basu et al. [7] have proposed a two step fuzzy technique for segmentation of Bangla words. In a recent work of handwritten Bangla OCR [1], distance based segmentation approach is used for line, word, and character segmentation. Lehal and Singh [27] have developed a Gurmukhi OCR system for machine printed text. The recognition of the text is done at sub-character level. The word is divided into sub-characters in the segmentation phase and in the recognition phase these sub-characters are classified and combined to form Gurmukhi

characters. Lehal [26] has presented a complete OCR for printed multi-font Gurmukhi script. Kumar and Sengar [25] have proposed a bilingual character segmentation comprising Gurmukhi and Devanagari script based on the horizontal and vertical projection of the word. Sharma et al. [45] have proposed an iterative approach based on the header line, aspect ratio and vertical and horizontal projection profiles for handwritten Gurmukhi text. Mangala and Kaur [29] have proposed an end detection algorithm for segmentation of touching and broken characters in handwritten Gurmukhi words. They also used horizontal and vertical profile projection for isolated character segmentation. Number of works have been done for character segmentation for Indian languages, but most of the methods work on removal of the header line followed by post-processing to segment the characters of the words [13, 28, 31, 36, 37, 46] as discussed above.

OCR of Indian scripts is an ongoing topic of research as not much of work is done on multilingual OCR of Indian scripts. However, it is troublesome for skewed word images. Implementing a generalized character segmentation method for more than one language is a trivial task due to the variation in writing styles and structural properties of languages. Few research works have been done for multilingual character segmentation for Indian scripts [2, 17, 22, 30]. But all of them, use different approaches for segmenting different languages and combine these approaches to make a single module. All these works are based on script identification and then applying character segmentation for the corresponding script.

Implementing a generalized method for character segmentation without removing header line, which works on more than one language and also handles skewed words motivates us for this work. The novelty of the work lies on the fact that our proposed method does not identify the language before segmentation and handles all the languages using a single segmentation approach, unlike previous works. Following key features of the proposed method illustrate the novelty of the proposed work.

- Proposed method does not identify language before segmentation.
- It uses a single approach to segment the words of different languages.
- It performs character segmentation without removal of header line.
- Works well on skewed words up to an angle of $\pm 10^{\circ}$.
- Works well on words with broken header line.
- Proposed method segments broken words correctly.

This is significant as a single method can be used to develop multilingual OCR for Indian languages. To our best knowledge, only a single work is done by Bhattad and Chaudhuri [8] on using a single approach for bilingual character segmentation of Bangla and Devanagari. But this work does not perform segmentation of modifiers. In this paper, we propose a method to segment multilingual handwritten words into characters using the polygonal approximation of the word.

The architecture of rest of the paper is as follows. Section 2 discusses characteristics of all the four languages, Hindi, Marathi, Punjabi, and Bangla. Section 3 extends the proposed methodology for character segmentation. Experimental results and analysis of results are discussed in Section 4. Concluding remarks and scope for future work are given in Section 5.

2 Properties of languages

The Government of India have recognized 22 languages namely, Assamese, Bengali (Bangla), Bodo, Dogri, Gujrati, Hindi, Kannada, Kashmiri, Konkani, Maithili, Malayalam, Marathi, Manipuri, Nepali, Odia, Punjabi, Sanskrit, Santali, Sindhi, Tamil, Telugu, and Urdu

as official languages. Among these four popular languages are Hindi, Marathi, Punjabi, and Bangla collectively spoken by approx 723 million people worldwide.

2.1 Properties of Hindi language

Hindi language is written in Devanagari script. It is the most prominent language spoken in India. It consists of 13 vowels and 33 consonants.

Like English language, there is no case of uppercase and lowercase of characters. Writing mode of this language is from left to write. Two or more characters are combined to form a word by joining header lines of individual characters.

2.2 Properties of Marathi language

Marathi language is also written in Devanagari script. Over 73 million people speaks Marathi in India resulting it as the fourth most spoken language in India and ranks nineteenth in the list of most spoken languages in the world [19]. As it is also written in Devanagari scripts, so it has the same properties as of Hindi language.

2.3 Properties of Punjabi language

With over 100 million speakers worldwide, it is ranked tenth most widely spoken language in the world [20]. With 30 million speakers in India, it is ranked 11th most spoken language in India. Punjabi is most widely spoken language in Pakistan. Gurmukhi script is used to write the Punjabi language. It comprises 10 vowels and 41 consonants. Punjabi words are also connected by header line. Its writing mode is from left to right.

2.4 Properties of Bangla language

Bangla language is written using Bangla script. The basic character set of Bangla comprises 11 vowels and 39 consonants. It is the second most spoken language in India and seventh most spoken language in world [21]. Over 80 million people in India speaks Bangla. It is also the national language of Bangladesh. For OCR point of view Bangla is very significant language.

Our algorithm works on exploiting the common structural properties (Fig. 1) of the above mentioned four languages. The comparative analysis of the above four mentioned languages is given in Table 1.

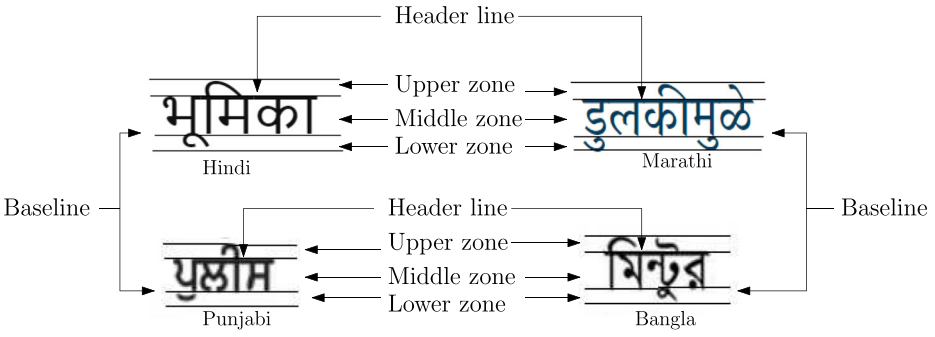

Fig. 1 Structural properties of languages

Table 1 Comparison of the four Indian languages

Property	Hindi	Marathi	Punjabi	Bangla
Language family	Indo-European	Indo-European	Indo-European	Indo-European
Native to	India	India	India and Pakistan	India and Bangladesh
Region	Northern India	Maharashtra	Punjab	Bengal
Script	Devanagari	Devanagari	Bangla	Gurmukhi
Writing Mode	Left to right	Left to right	Left to right	Left to right
Zonal division of words	Divided into 3 zones	Divided into 3 zones	Divided into 3 zones	Divided into 3 zones
Uppercase letters	Absent	Absent	Absent	Absent
Lowercase letters	Absent	Absent	Absent	Absent
Headerline	Present	Present	Present	Present
Invisible Baseline	Present	Present	Present	Present
Number of consonant	33	33	41	39
Number of vowels	13	13	10	11
Number of speakers in India (in million)	120	73	30	80
Official language in	India	India	India	India and Bangladesh

3 Proposed method

In this work, we have considered header line and baseline for character segmentation and modifiers' segmentation. In our proposed method header line of the word is not removed to perform character segmentation. The whole method is divided into four phases. First phase is preparatory phase in which input word is transformed in the required form for the algorithm. In second phase, polygonal approximation is done to find out the straight line segments of the word. On these segments, we perform the traversing to find out the segmentation points on the header line. On the basis of these segmentation points, word is divided into the characters. In the next two phases, upper and lower modifiers, if any, are segmented using statistical information about the individual components of words. Figure 2 depicts the system architecture of the proposed method.

3.1 Preparatory stage

Optical scanning of the document results into raw input image for OCR system. Pre-processing is the first phase of document image analysis. The purpose of this stage is to improve the quality of the image. Two pre-processing methods are applied to make input ready for the next phase.

- This stage reads the input image of word and then performs binarization using Otsu's method [33]. Otsu's method works on fixed global thresholding. Converting gray scale image to binary image is the first step of OCR.
- On the binarized input, thinning is applied using [49]. This is a fast parallel algorithm. In thinning, the image regions are reduced to one-pixel width characters.

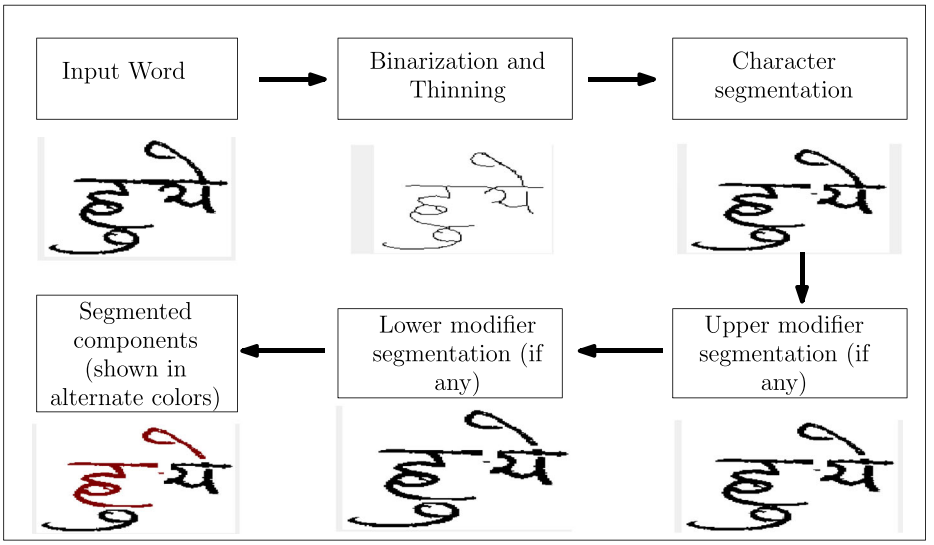

Fig. 2 System architecture of the proposed method

These mentioned pre-processing techniques, namely binarization and thinning are important as our proposed method solely depends on these two methods. Polygonal approximations of a digital curve are based on chain code property. As chain code works on single pixel width characters, so for this purpose thinning is applied. To perform thinning, image binarization is an important pre-processing step. If any stroke is lost in the binarization process, then the approximate shape of the character will be inaccurate, so as the segmentation of the word which in result affects the performance of OCR.

3.2 Character segmentation

Figure 3 outlines the proposed method for character segmentation. Following steps describe the process in detail.

1. Binarized thinned word obtained in Section 3.1 is the input for this process (Fig. 3b).
2. Polygonal approximation [9] is applied on the binarized thinned input word image to do the character segmentation of word. This algorithm determines digital straightness of the one-pixel width word obtained after thinning. It describes digital straight line segments (DSS) from digital curve. The number of such segments required to cover the word is few. As a result, the data set required to represent the word is reduced to a large extent. Since, our proposed method of segmentation is based on the structural properties of the four mentioned languages so, it requires storing the information of the structure and shape of the words in efficient manner. The structure of the word can be represented with very less data after applying the polygonal approximation and can be accessed efficiently. It transforms the huge pixel data of words to simple graph with considerable less data points and makes the traversal of points simple and possible. Polygonal approximation gives the approximate points of segments (Fig. 3c). When we combine these approximated points, we get the approximated shape of the word as shown in Fig. 3e.

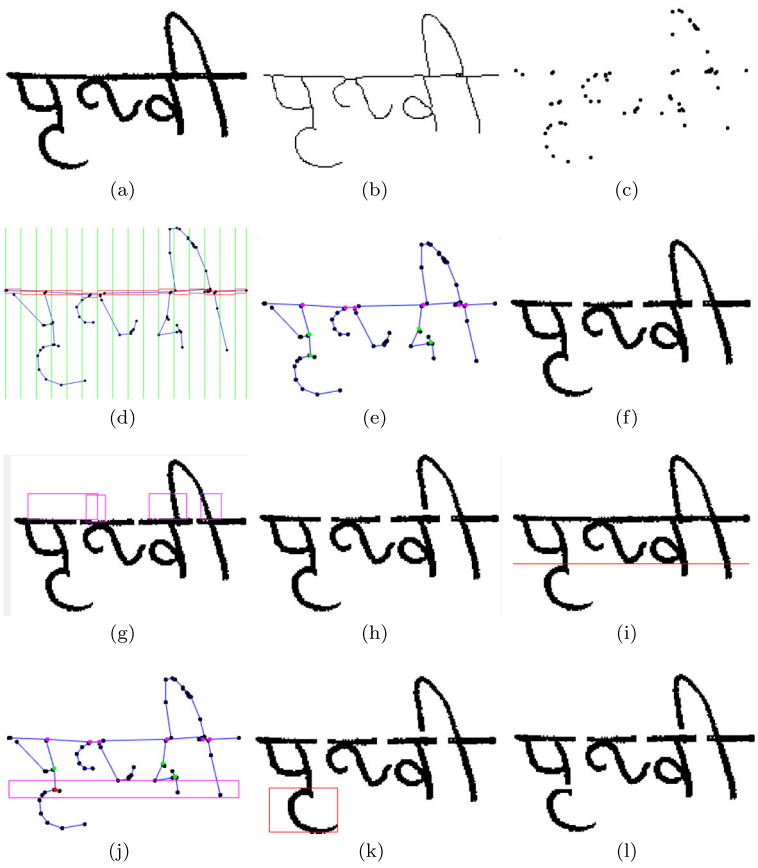

Fig. 3 The process of character segmentation **a** input word image, **b** binarized thinned word, **c** detected approximated points, **d** detection of header line, **e** different points considered for segmentation (junction points are shown in green color are header junction points are shown in magenta color), **f** result of character segmentation (modifiers remain connected), **g** bounding boxes for upper modifier segmentation, **h** result of upper modifier segmentation, **i** baseline detection, **j** baseline junction point detection (rectangular box is baseline region and red point in this region is baseline junction point), **k** bounding box (shown in red color) for lower modifier segmentation, and **l** result of lower modifier segmentation and final segmented output

3. Junction point is defined as the point which have more than two neighbors. We identify all the junction points among all the approximated points. The green points are the junction points of the word (Fig. 3e).
4. We define header junction points as the junction points residing on the header line. We identify all the header junction points out of all the junction points. All the junction points which are on the header line are detected even if the word is skewed upto an angle of $\pm 10°$. In this process, the word is divided into stripes vertically (Fig. 3d). In each vertical section, row with maximum object pixel density is detected as header line. Red boxes in the figure show the header line for that particular region. Thus header line is detected irrespective of the skewness of the word so as the header junction points are determined. Magenta points in Fig. 3e are the header junction points.

5. We combine the approximated points to get the approximated shape of the word which resemble a graph structure (Fig. 3e). These points are traversed to determine the segmentation points. Traversing is done for all the header junction points $header\,JP$ obtained in step 4. If the end point or another header junction point is reached while traversing from source header junction point S, then S is marked as segmentation point. Following steps describes traversing in detail:

   Step 1: Initialize a queue Q as empty and source S as a $header\,JP$.

   Step 2: Each approximated point is connected to its neighbors, so, we check the neighbors. If neighbor point p of S is a junction point and its not visited yet, then p is marked as visited and inserted to queue Q.

   Step 3: If it is a $header\,JP$, then we do not need to traverse further for S. Point p is marked as visited and as segmentation point. Traversing is terminated for point S and Q is made empty.

   Step 4: If Q is empty then, no more points are remaining for traversal, it ensures that all the junction points reaching from header junction point $header\,JP$ are traversed.

   Step 5: If Q is not empty, then all the points reaching from $header\,JP$ are not traversed. So, point is deleted from Q, and this becomes new source S for traversal and steps 2–5 are repeated.

   Step 1–5 are repeated for each $header\,JP$.

   Figure 4 explains the traversing method to determine the segmentation points on the example word image. Both the cases of traversal described above are shown in the example. In this example, four points J_1, J_2, J_3, and J_4 are the header junction points and E_1, E_2, and E_3 are the end points. Green points are the junction points of the word ('c' becomes junction point due to the presence of small cut). We start traversing from J_1 following the direction shown by arrows and reach to the point E_1 via 3 points 'a', 'b', and 'c'. This is one path of traversal (Fig. 4a). On reaching point E_1, queue becomes empty, so, traversing is completed for header junction point J_1. Another path of traversal is possible starting from J_1 to E_1 following the direction via points 'a', 'c', and 'b' (Fig. 4b). In this path, we reach to end point E_1 through junction point 'a'. Since, all the other points reaching from 'a' are not discovered yet so, we backtrack and traversing continues following the direction of arrows as shown in Fig. 4b. On reaching to junction point 'b', traversing stops as all the points are discovered and queue becomes empty. We got the end point and did not reach to any other header junction point in this traversal so, header junction point J_1 is marked as segmentation point. Similarly, starting from J_2, we reach to E_2. Whereas, starting from header junction point J_3, we get to end point E_3 following the direction of the arrows via 2 junction points 'd' and 'e'. But on reaching to E_3, queue is not empty as junction point 'e' reaching from J_3 is not traversed completely (neighbor point of 'e' is still not visited). So, traversing backtracks from E_3 and another header junction point J_4 is reached, which is the part of same character. Therefor, among J_1, J_2, J_3, and J_4, header junction J_1, J_2, and J_4

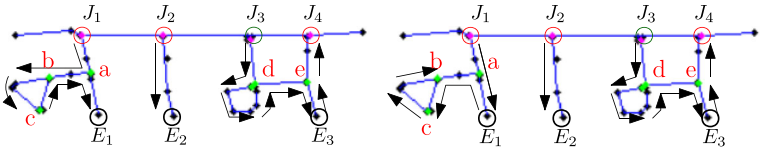

Fig. 4 Detection of segmentation points using ***PointTraversal*** procedure

are considered for segmentation while J_3 remains connected to the character. Even presence of small cuts and merges does not create any problem in traversal as data set required to represent the word is few after applying the polygonal approximation so as the cuts and merges are few in the word. All the points follows the same procedure and results into character segmentation (as explained in the given example).

6. We segment the word into individual characters after we get the segmentation points. The word is divided into individual characters with modifiers attached to it (Fig. 3f). We perform the segmentation of both upper and lower modifiers one by one. In the next sections, we explain the process of modifiers segmentation.

Algorithm 1 SegmentUpModifier.

```
1:  Initialize headerJP[1 : N] = < junction points lying on header line>
2:  Initialize W_c[1 : N] = < width of characters associated with headerJP in the word>
3:  for i ← 1 to N do
4:      Define a bounding box B_bi of width w_bi and of height h_bi over headerJP[i];   ▷
    w_bi = W_ci and h_bi is calculated using (1).
5:      Calculate number of pixels n_pi present inside B_bi;
6:      if n_pi ≥ θ_u then                                          ▷ modifier is present
7:          if W_ci ≥ θ_w then                                      ▷ θ_w is threshold for width
8:              Segment upper modifier associated with jeaderJP[i];
9:          else
10:             Upper modifier remain connected with headerJP[i];
11:         end if
12:     end if
13: end for
```

3.3 Upper modifier segmentation

After the character segmentation, the statistical information of the word width is utilized to do the upper modifier segmentation. Our proposed method for upper modifier segmentation is advantageous as all the upper modifiers are segmented using this single logic. We do not categorize the modifiers to segment them as reported in earlier literatures [5].

Procedure ***SegmentUpModifier*** in Algorithm 1 explains the approach for segmentation of upper modifiers.

1. We determine the presence of upper modifier with the help of bounding box. A rectangle bounding box B_b (Fig. 3g) is drawn over each header junction point $headerJP$ which belongs to the character (Line 1–4). The width of this box is equal to the width of the character and height (h_b) of the box is calculated using (1).

$$h_{bi} = |headerJP_i^x - \texttt{mid\_point}_i| \tag{1}$$

where,

$$\texttt{mid\_point}_i = (\texttt{start\_row} + headerJP_i^x)/2 \tag{2}$$

where, `start_row` is the start row of word image.

2. Next, to detect the upper modifier, we calculate the number of pixels n_p present inside each B_b. If $n_p \geq \theta_u$, upper modifier is present (Line 5–6). We determine the value of θ_u as 30 on the basis of experimental analysis (Section 4.4).

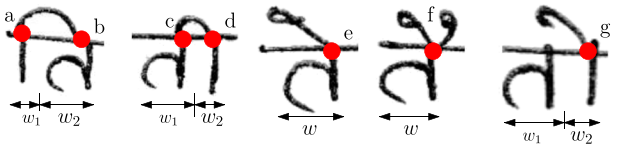

Fig. 5 Explanation of upper modifier segmentation with example

3. After we identify the modifier, we compare the width of character W_c associated with the header junction point over which the modifier is connected (Line 7).
4. The upper modifier is segmented from the character if $W_c \geq \theta_w(threshold)$, where W_c is width of character (Line 8–10). Otherwise, upper modifier stays associated to the character.

This process is explained in Fig. 5. In first two figures, two modifiers are shown. In first case, $w_1 < \theta_w$ while $w_2 > \theta_w$ so, upper modifier is segmented at point b but remains connected to a. The reverse of this is true in second case. So, upper modifier is segmented at point c. In next two figures, $w > \theta_w$, so upper modifier is segmented at points e and f. In last figure, modifiers at points g is segmented as $w_2 < \theta_w$. We have taken the value of θ_w as 10 on the basis of experimental analysis (Section 4.4). Figure 3h shows the result of upper modifier segmentation of the input word. Some image results for upper modifier segmentation for four different languages are shown in Fig. 6. In case upper modifier is not connected properly to the character of the middle zone and

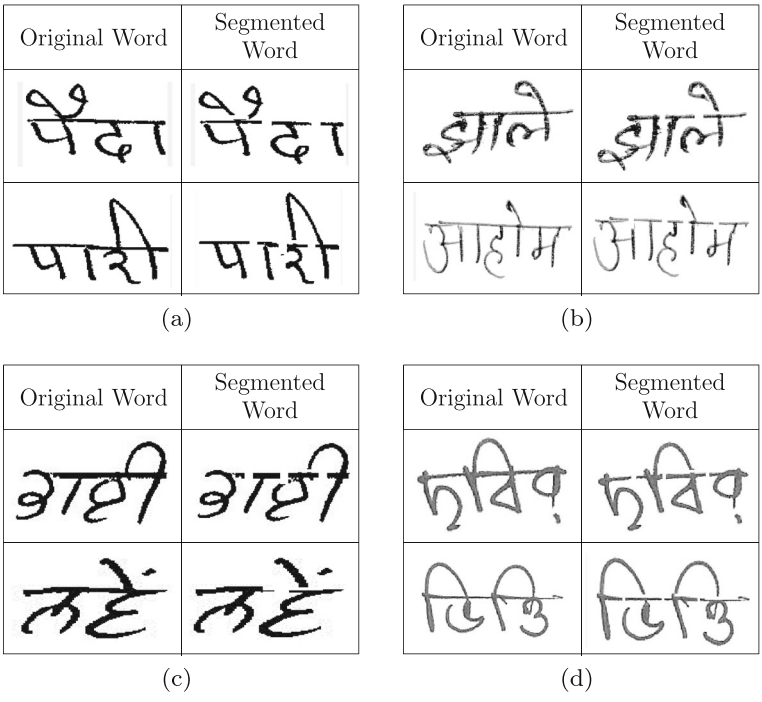

Fig. 6 Image examples for segmentation of upper modifier for **a** Hindi, **b** Marathi, **c** Punjabi, and **d** Bangla languages

appears partly on top of two characters, width of the nearest character is compared as the upper modifier automatically comes in the bounding box over the character which is nearer to the modifier (Fig. 7). In the given example of Hindi language, point 'p' is nearer to header junction point 'a' than point 'b' so width of the character associated with header junction point 'a' is compared resulting in to the proper segmentation of upper modifier.

Algorithm 2 SegmentLowModifier.

1: Baseline ← DetectBaseline()
2: Initialize $baseJP[1:N]$ = <junction points lying on baseline region>
3: Initialize $W_c[1:N]$ = < Width of characters associated with $baselineJunctionPoints$ in the word>
4: **for** $i \leftarrow 1$ to N **do**
5: Define a bounding box in lower zone LB_{bi} of width w_{bi} and of height h_{bi} over $baseJP[i]$; ▷ $w_{bi} = W_{ci}$ & h_{bi} is calculated using (3).
6: Calculate number of pixels n_{p_i} present inside LB_{bi};
7: **if** $n_{p_i} \geq \theta_l$ **then** ▷ modifier is present
8: Segment lower modifier associated with $baseJP[i]$;
9: **end if**
10: **end for**

3.4 Lower modifier segmentation

The method used to segment lower modifiers is similar to the upper modifier segmentation method as discussed in Section 3.3. In case of upper modifiers, we work on the junction points lying on header line. Similarly, for lower modifier we consider the the junction points lying on the baseline region. Baseline is the invisible line that separates the middle zone from lower zone of the word (Fig. 1). The advantage of this method is that all the modifiers are handled using this single approach only. The method is explained in procedure ***SegmentLowModifier*** in Algorithm 2.

1. This procedure starts with detecting the baseline using procedure ***DetectBaseline*** in Algorithm 3. To detect the baseline, sudden changes in horizontal density of object pixel from middle row to end row of image is considered (Line 1–4). It is observed that sudden changes in density is found at the end of lower zone but it is not correct if the modifier is present (Fig. 3i). So, to generalize the baseline detection, concept of pixel density of word in different zone is considered. The first detected baseline is named as $Baseline_{end}$ (Line 6). To overcome this problem, second baseline $Baseline_{second}$ is detected (Line 7). We calculate the difference $n_{p_{diff}}$ of number of pixels lying in both

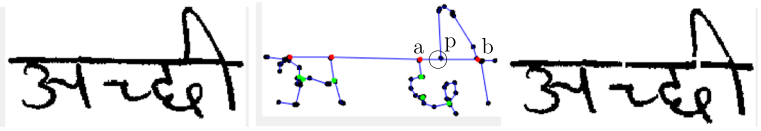

Fig. 7 Examples of upper modifier segmentation for Hindi in case of modifier appears partly between two characters **a** Input word, **b** approximated points of word, and **c** segmented output

rows of $Baseline_{second}$ and $Baseline_{end}$ (Line 9–11). If $n_{p_{diff}} < 5$ (value taken as per experimental analysis) then $Baseline_{end}$ is considered as baseline (Line 12–13); otherwise $Baseline_{second}$ is the baseline (Line 15).

Algorithm 3 DetectBaseline.

1: Find $start\_row$, end_row of word image;
2: Calculate $mid\_row = (start\_row + end\_row)/2$;
3: Get horizontal density of number of object pixels from mid_row to end_row;
4: $Baseline \leftarrow$ row with sudden change in density; ▷ value taken is 20
5: **if** $Baseline$ is closer to end_row **then** ▷ value taken is 5
6: $\quad Baseline_{end} = Baseline$;
7: $\quad Baseline_{second} =$ row with next sudden change in density;
8: **end if**
9: Calculate number of pixels $n_{p_{end}}$ of row $Baseline_{end}$;
10: Calculate number of pixels $n_{p_{second}}$ of row $Baseline_{second}$;
11: Calculate difference $n_{p_{diff}} = |n_{p_{second}} - n_{p_{end}}|$;
12: **if** $n_{p_{diff}}$ is significantly less **then** ▷ value taken is 5
13: $\quad Baseline \leftarrow Baseline_{second}$;
14: **else**
15: $\quad Baseline \leftarrow Baseline_{end}$;
16: **end if**

Figure 8 shows the two cases for detecting the baseline. In first word of image $n_{p_{diff}}$ is *less* as both the rows $Baseline_{end}$ and $Baseline_{second}$ lying in middle zone which is highest zone of object pixel density. On the other hand, in case of presence of lower modifier (second word of image), row $Baseline_{end}$ is in lower zone (low object pixel density) and row $Baseline_{second}$ is in middle zone (high object pixel density). Due to which $n_{p_{diff}}$ will be *higher* in this case. So, in case of first word of Fig. 8, $Baseline_{end}$ is $Baseline$ and in second word, $Baseline_{second}$ is $Baseline$. All the values which are considered are based on experimental analysis.

2. We define baseline junction points $baseJP$ as the junction points lying in the baseline region (Fig. 3j). We identify all the baseline junction points $baseJP$ after the baseline is detected.
3. Bounding box are drawn for each $baseJP$ (Fig. 3k) (Line 4–5). Width of bounding box is equal to the width of the character associated with the $baseJP$ and height h_b is calculated using (3).

$$h_{bi} = |baseJP_i^x - \texttt{end\_row}| \tag{3}$$

where, `end_row` is the end row of the word image.

4. Calculate the total number of pixels inside the bounding box. If number of object pixels are greater than the threshold θ_l then modifier is present. We disconnect the lower

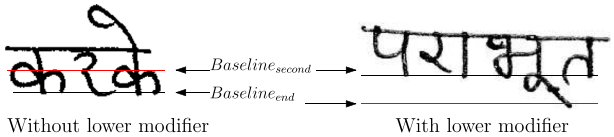

Fig. 8 Example of baseline detection

modifier from that baseline junction point (Line 6–9). We set the value of θ_l as 30 (on the basis of experimental analysis)(Section 4.4). Figure 3l shows the result of lower modifier segmentation for the input image of Fig. 3a. Figure 3l is the final segmented output of the input word. Image results of lower modifiers' segmentation for all the four languages are shown in Fig. 9.

4 Experimental results and discussion

We discuss the experimental results and analysis of results of our proposed method in this section.

4.1 Experimental dataset

We have taken about 3000 handwritten words of each of the four language for our experimental purpose resulting 12000 handwritten words in total. Out of four languages, dataset for Hindi and Bangla are obtained from [43] and dataset for Punjabi is obtained from [32]. Marathi dataset is collected from 20 individuals of different age group. Each Marathi image is digitized at 300 dpi resolution using Cannon Pixima E560 flat-bed scanner. The implementation has been done on MATLAB.

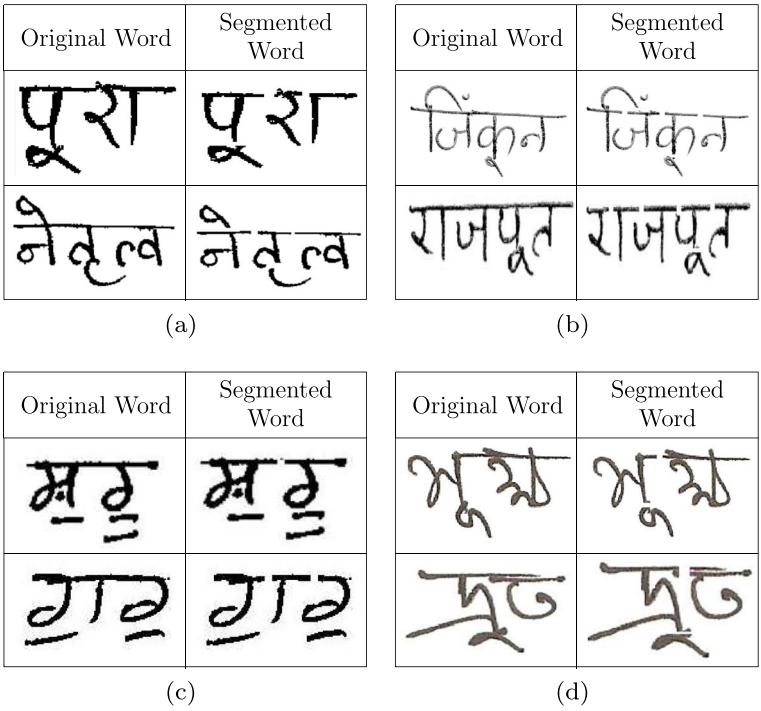

Fig. 9 Examples of lower modifier segmentation for **a** Hindi, **b** Marathi, **c** Punjabi, and **d** Bangla languages

4.2 Results for character segmentation

Method described above performs segmentation for four Indian languages. Some of the image results for Hindi and Marathi languages are shown in Fig. 10 and for Punjabi and Bangla languages are shown in Fig. 11. The proposed methods handles some special cases like broken words, words without header line and skewed words as well. These cases are described as follows:

Character Segmentation of Skewed Words: As described in Section. 3.2, methods works well on skewed words. Image result for skewed words for all four different languages are shown Fig. 12.

Character Segmentation of Broken Words:

In handwritten text, the word can be broken in two ways. Many time while writing in flow, some of the characters are broken from the words. This makes character segmentation difficult. Few examples of broken words are given in Fig. 13. This is first type of broken word. Our proposed method works well on input with broken characters. Segmentation is

(a) Hindi (b) Marathi

Fig. 10 Experimental results for Hindi and Marathi languages. Alternate colors of gray and black is used to represent the segmented components

(a) Punjabi (b) Bangla

Fig. 11 Experimental results for Punjabi and Bangla languages. Alternate colors of gray and black is used to represent the segmented components

performed on the remaining connected part of the word (Fig. 14). The broken parts of the word do not require any segmentation and can be considered as they are for classification.

In second type of broken word, header line is broken in middle resulting two or more parts in a single word. These type of words are found mainly in Bangla only. To handle these kind of cases, connected components are found in the word image and the proposed method is applied for each component resulting into segmented individual characters. Figure 15 shows the image example for words broken from header line.

Character Segmentation of Words without Header line: Many writers do not put header line while writing in flow. Due to the absence of header line, segmentation becomes difficult. These type of cases are also found mainly in Bangla language. To overcome this problem, we estimated header line of the word using a linear curve fitting-based method proposed in [39]. After estimating the header line, character segmentation is applied as discussed in Section 3. Some of the image example for Bangla language are given in Fig. 16.

Detailed experimental results for all four languages: Hindi, Marathi, Punjabi, and Bangla are reported in Table 2, Table 3, and in Table 4 for character segmentation, upper modifiers'

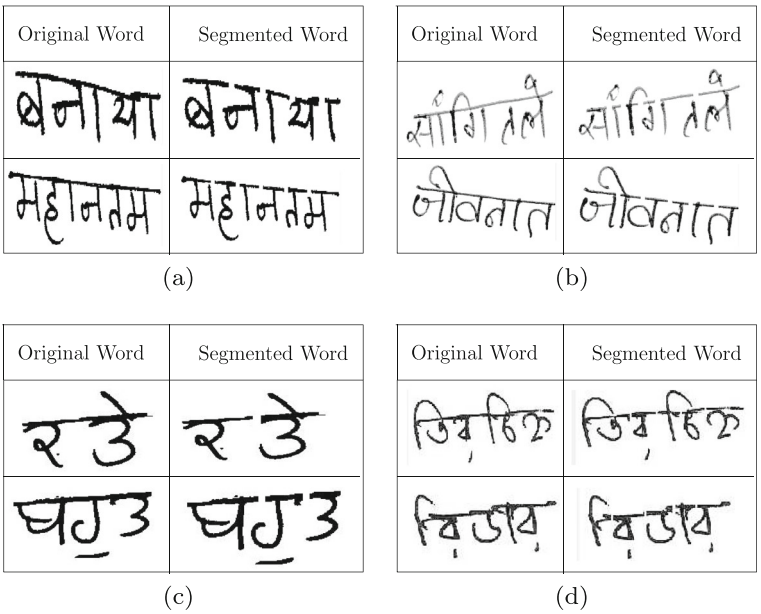

Fig. 12 Examples of character segmentation of skewed words for **a** Hindi, **b** Marathi, **c** Punjabi, and **d** Bangla languages

segmentation, and lower modifiers' segmentation respectively. Lower modifiers in Punjabi language are not connected to the characters. This is the reason for the 100% accuracy for lower modifier segmentation for Punjabi language. The combined average accuracy for

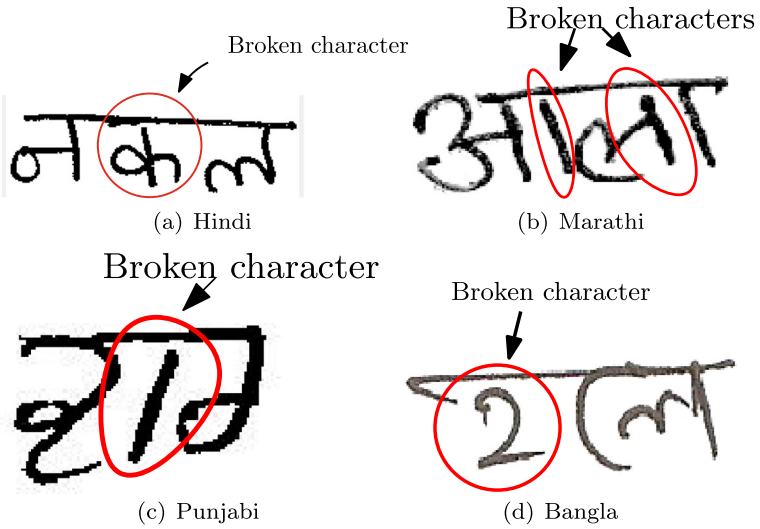

Fig. 13 Broken word image examples of Type-I

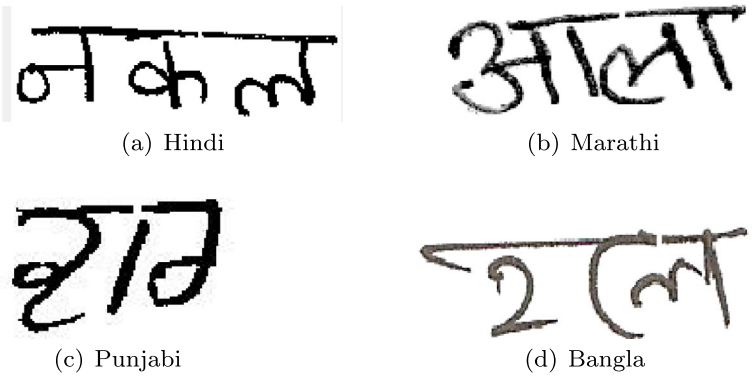

Fig. 14 Segmentation of broken words of Fig. 13

character segmentation obtained for all the languages is 90.07%. For upper modifiers, the average accuracy for all the four languages is 90.33%, and in case of lower modifiers, the obtained average accuracy is 93.99%.

4.3 Character recognition result

In this Section, we discuss the extracted features and classifiers used for character recognition purpose.

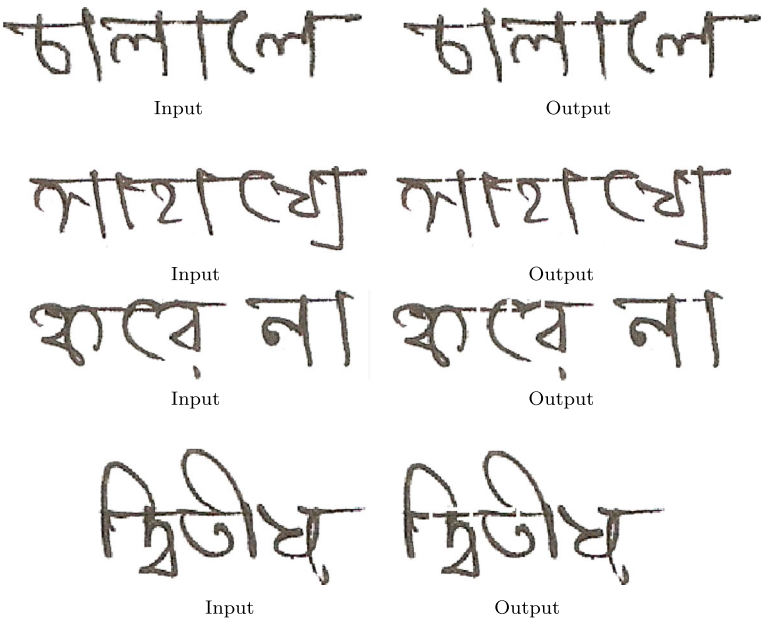

Fig. 15 Broken word image example of Type-II

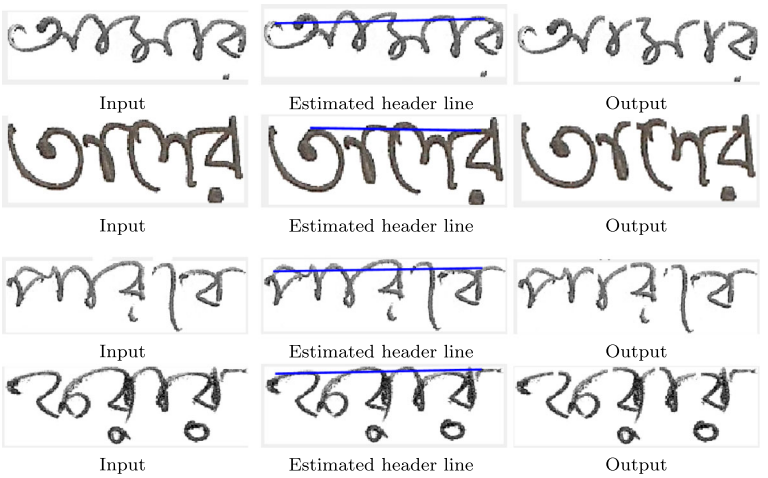

Fig. 16 Image examples of broken header line in word in Bangla language.

4.3.1 Feature extraction

Two categories of features [40] are extracted for each separated binary component of the word as given below.

i. Cumulative stretch in four different direction, and
ii. Shadow features [14].

Cumulative stretch features are extracted for four directions, namely horizontal, vertical, left diagonal, and right diagonal. For shadow features extraction, each component is divided into 8 octant and each octant has 3 sides. Light is projected from 3 different directions and length of the shadow is computed on each side of octant. Total of 84 cumulative stretch features and 120 shadow features are obtained resulting into a total of 204 features for each component.

4.3.2 Classifiers

We have applied the extracted features to three classifiers, namely Multilayer Perceptron (MLP), Support Vector Machine(SVM), and Random Forest.

MLP: MLP is also known as feed-forward neural network which consists of multiple layers of computational units interconnected in feed-forward way. MLP maps the input to the predicted output data. It uses the back propagation learning technique in which output

Table 2 Character segmentation result of all four languages

Language	Words	Total no. of components	Correctly detected components	Accuracy (%)
Hindi	3000	9034	8208	90.86
Marathi	3000	14437	12986	89.95
Punjabi	3000	9160	8307	90.69
Bangla	3000	7305	6485	88.77

Table 3 Upper modifiers' segmentation result of all four languages

Language	Words	Total no. of components	Correctly detected components	Accuracy (%)
Hindi	3000	2501	2278	91.08
Marathi	3000	2388	2151	90.08
Punjabi	3000	2461	2257	91.72
Bangla	3000	1740	1539	88.45

values are compared to the correct answer to get the value of mean square error. This error is then fed back to the network and weights of the network are updated in order to reduce the mean square error between the actual and desired outcome of the network. The network is trained when this value to minimized. In the present work, we use one-hidden layer MLP.

This classifier is widely used for classification in various areas. Pramanik and Bag [38] have used MLP for classification of Bangla compound characters, Zhang and Sun [50] have used single layer feed forward neural network to develop a angiosperm genus classification system. Wang et al. [48] have also used single layer feed forward neural network for recognition of facial emotions.

SVM: SVM is a set of supervised learning methods that analyzes data and recognizes patterns that is used for classification. In SVM model each data item is plotted as a point in n-dimensional space, mapped so that the points of the different categories are divided by a clear gap. In more formal way, a SVM model constructs a hyperplane or set of hyperplanes in a high or infinite dimensional space that differentiate the two classes. The hyperplane achieves a better separation if data points of any class are as far as possible. Larger difference between the hyperplane and the data points leads to the lower generalization error of the classifier. In this work we have used polynomial kernel in SVM model for classification.

Random Forest: Random forest is a supervised learning method that operates by constructing multiple decision tree classifiers during training phase. To classify a new object on the basis of attributes, the decision of the majority of the trees is chosen by the classifier as the final decision. The random forest classifier avoids the problem of over-fitting by using multiple trees which re-samples the data during training phase and by changing the features over different classifier. Moreover, random forest classifier handles the missing values which makes it more robust to noise. These two advantages of random forest classifier make this a potential choice for classification.

The recognition of characters is done using above mentioned classifiers. Table 5 shows the accuracy of each language for all the three classifiers. Among all tested languages, and classifiers, the Hindi language shows the highest accuracy of 96.09% for MLP classifier. Marathi and Punjabi languages also show the best accuracy for MLP classifier scoring 90.50% and 91.21% respectively. Bangla language shows relatively less accuracy of 85.74%

Table 4 Lower modifiers' segmentation result of all four languages

Language	Words	Total no. of components	Correctly detected components	Accuracy (%)
Hindi	3000	517	484	93.62
Marathi	3000	631	579	91.62
Punjabi	3000	419	419	100
Bangla	3000	435	394	90.57

Table 5 Character recognition accuracy for each language

Language	Classifier	Accuracy (%)
Hindi	Random Forest	95.10
	SVM	95.57
	MLP	**96.09**
Marathi	Random Forest	87.24
	SVM	89.63
	MLP	**90.50**
Punjabi	Random Forest	88.05
	SVM	90.85
	MLP	**91.21**
Bangla	Random Forest	84.12
	SVM	**85.74**
	MLP	83.77

for SVM classifier in comparison to the other three languages. Vast variation in the writing style of Bangla language results in ample deviation in the dataset which makes the recognition of characters difficult and consequently affects the recognition accuracy for this language. Best accuracies obtained by all four languages among all three classifiers are shown in bold numbers.

We also applied a deep learning architecture for recognition of characters. A deep convolutional neural network (CNN) is a feed-forward multilayer network trained in supervised mode. CNN consists of a number of layers namely, convolutional layers, pooling layers, and fully connected layers. There can be any number of fully connected layers after the convolutional layers and pooling layers as in a standard multi-layer neural network. The CNN model takes the whole binary image of dimension of 32×32 as input in a single 1 dimensional array. So the single 1 dimensional array of size 1024 is selected as features. From the total data of each language, 80% data is used for training of model and rest of the data is used for testing. The training data is randomized beforehand to avoid the problem of data overfitting. This model is trained with the batch training size of 100 and learning rate of 0.001. The classifier provides the recognition accuracy for deep learning for all the languages. For our case, CNN classifier obtains accuracy of 94.71% for Hindi language which is less than the accuracy obtained by all the other three classifiers. For Marathi and Punjabi languages CNN shows better accuracy of 89.16% and 89.84% in comparison to random forest classifier for the same. For Bangla language CNN obtains the accuracy of 83.77% which is same as for MLP classifier.

4.4 Parameter tuning

Figure 17 provides a tuning for choosing the optimal threshold θ_u for checking the presence of upper modifier as discussed in Algorithm 1. We have taken 400 random word images, 100 from each language containing upper modifiers to do the parameter tuning. Figure 17 shows the graph of accuracy for different values of θ_u for all the four languages. Accuracy of upper modifier segmentation is different for all the languages for various values of θ_u, but all the languages except Marathi, obtained the best accuracy for upper modifier segmentation for

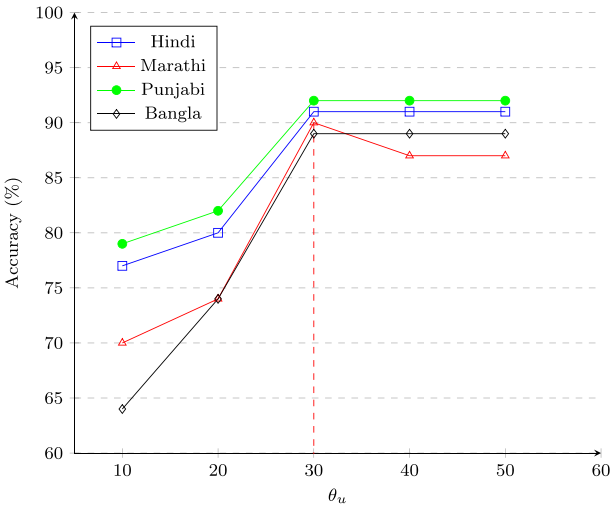

Fig. 17 Tuning of parameter θ_u used in Algorithm 1

$\theta_u \geq 30$. Marathi shows a slight decrement in accuracy for threshold > 30. So, choosing θ_u as 30 is the best accuracy for upper modifier segmentation as depicted in Fig. 17.

Similarly, Fig. 18 depicts the tuning of threshold value θ_w of Algorithm 1 for all the four languages. Same data samples of 400 images have been tested for various threshold values starting from $\theta_w = 2$. Languages obtained best accuracy for $\theta_w \geq 10$ except for Punjabi language as shown in the graphs. Punjabi achieved its best accuracy for $\theta_w \geq 8$. So, threshold $\theta_w \geq 10$ have been chosen as the common threshold for all the four languages.

Tuning of threshold value θ_l of Algorithm 2 is shown in Fig. 19 for all the four languages. Graph for Punjabi language is a straight line at 100%. This is due to the fact that lower

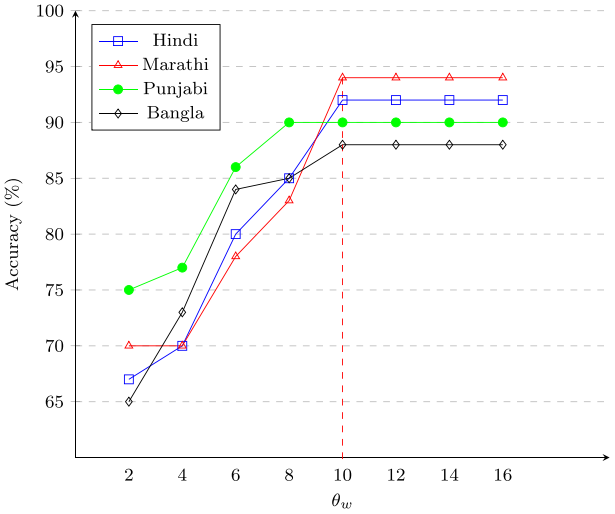

Fig. 18 Tuning of parameter θ_w used in Algorithm 1

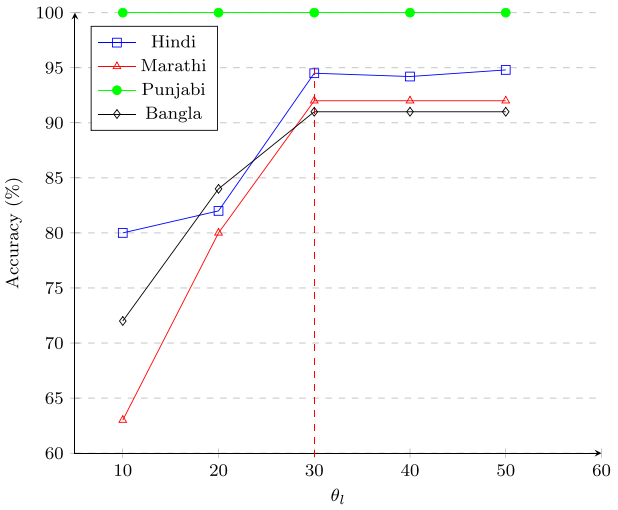

Fig. 19 Tuning of parameter θ_l used in Algorithm 2

modifiers are not connected to the character of the middle zone. Due to which segmentation accuracy is 100% for any value of pixels. Other than Punjabi all the languages obtained their best accuracy for $\theta_l \geq 30$.

4.5 Comparison with other methods

As per our best knowledge, only a single work of Bhattad and Chaudhuri [8] is reported for character segmentation for two different languages using a single approach. So to show the competency of our proposed method we compared character segmentation accuracy of individual language against state-of-the-art approaches for each dataset. We compared our method with recent works of Srivastav and Sahu [47] and Sarkar et al. [44] method for Hindi language, Ramteke et al. [41] for Marathi language, Mangla and Kaur [29] for Punjabi language, and Arefin et al. [1] for Bangla language. As the databases used in the above papers are not available, so to do the comparative analysis, we implemented the above methods and tested the same on 500 words of each database as discussed in Section 4.1. Table 6 shows the character segmentation accuracy for the above mentioned methods and our proposed method. We observe that the performance of character segmentation is much better than the existing methods for all the languages. The reason behind this is that the compared methods fail to segment the skewed words. In method proposed by Bhattad and Chaudhuri [8] lower modifiers are not segmented resulting into the decrement in the accuracy. The segmentation of upper modifiers are followed by post processing of reconnecting them to original characters. But our proposed method does not require any post processing for modifier segmentation. Our proposed method is tested for four languages and gives satisfactory results, whereas [8] is tested only for two languages. The success rate of our proposed method is 91.04% and 89.47% for the Hindi and Bangla languages respectively which is much better than 80.59% and 82.26% for Hindi and Bangla of method proposed by Bhattad and Chaudhuri [8]. It is also better than the methods proposed by Sarkar et al. [44] and Srivastav and Sahu [47] for Hindi language and by Arefine et al. [1] for Bangla language. Arefine et al. [1]

Table 6 Comparison of character segmentation accuracy of the proposed method with the state-of-the-art methods

Method	Language	True segmentation (%)		
		Upper modifier	Constituent character	Lower modifier
Bhattad and Chaudhuri [8]	Hindi	–	80.59	–
	Bangla	–	82.26	–
Srivastav and Sahu [47]	Hindi	80.92	82.28	–
Sarkar et al. [44]	Hindi	82.40	89.34	88.64
Ramteke et al. [41]	Marathi	–	73.23	–
Mangla and Kaur [29]	Punjabi	–	76.39	–
Arefin et al. [1]	Bangla	–	78.31	–
Proposed Method	Hindi	**92.42**	**91.04**	**92.09**
	Marathi	–	**88.72**	–
	Punjabi	–	**91.82**	–
	Bangla	–	**89.47**	–

have also not mentioned the segmentation of modifiers, so this cannot be compared with our proposed method. For Marathi and Punjabi languages, our method gives the accuracy of 88.72% and 91.82% which is better than the corresponding methods. Upper and lower modifier segmentations are not mentioned separately in the above papers, so they also cannot be compared. Only Sarkar et al. [44] and Srivastav and Sahu [47] have provided the segmentation accuracy for the modifiers for Hindi language. So, the segmentation accuracies for these methods are compared with our proposed method. Accuracies obtained by proposed method are shown in bold numbers to show the efficacy of the method.

A comparison of the average process speed of above compared method is given in Table 7. This comparison is carried out with the methods on the same language datasets as specified above. The method proposed by Bhattad and Chaudhuri [8] is computationally less expensive but shows poor performance. The process speed of our proposed method depends on the word structure and length of the word. Our proposed method uses polygonal approximation which is based on chain code, so for this reason, our proposed method takes much time for processing. Our method shows better computational efficiency for Marathi and Punjabi language as compared to other methods. Best average process speed for each language among all the methods is shown in bold.

Table 7 Average process speed (in second) taken by different character segmentation methods

Method	Hindi	Marathi	Punjabi	Bangla
Bhattad and Chaudhuri [8]	**0.26**	–	–	**0.13**
Srivastav and Sahu [47]	0.29	–	–	–
Ramteke et al. [41]	–	0.55	–	–
Mangla and Kaur [29]	–	–	0.51	–
Arefin et al. [1]	–	–	–	0.15
Proposed method	0.44	**0.49**	**0.19**	0.31

5 Conclusion

We have proposed a novel script independent character segmentation technique based on polygonal approximation. The use of structural properties of all the languages helps us to generalize the segmentation process. This method is tested on four popular Indian languages and has given promising results for all the four languages resulting average segmentation accuracy of 90.07%. We have also performed the character recognition with random forest, SVM, MLP, and CNN classifiers with satisfying segmentation and recognition results. In future, we shall extend our work to improve the accuracy by handling the failure cases and to use this method for character segmentation of more languages.

References

1. Arefin N, Hassan M, Khaliluzzaman M, Chowdhury SA (2017) Bangla handwritten characters recognition by using distance-based segmentation and histogram oriented gradients. In: IEEE Region 10 humanitarian technology conference, pp 678–681
2. Arya D, Jawahar C, Bhagvati C, Patnaik T, Chaudhuri B, Lehal G, Chaudhury S, Ramakrishna A (2011) Experiences of integration and performance testing of multilingual OCR for printed Indian scripts. In: Joint workshop on multilingual OCR and analytics for noisy unstructured text data, 9
3. Bag S, Bhowmick P, Harit G, Biswas A (2011) Character segmentation of handwritten Bangla text by vertex characterization of isothetic covers. In: National conference on computer vision, pattern recognition, image processing and graphics, pp 21–24
4. Bag S, Harit G (2013) A survey on optical character recognition for Bangla and Devanagari scripts. Sadhana 38(1):133–168
5. Bag S, Krishna A (2015) Character segmentation of Hindi unconstrained handwritten words. In: International workshop on combinatorial image analysis, pp 247–260
6. Bansal V, Sinha R (2002) Segmentation of touching and fused Devanagari characters. Pattern Recogn 35(4):875–893
7. Basu S, Sarkar R, Das N, Kundu M, Nasipuri M, Basu DK (2007) A fuzzy technique for segmentation of handwritten Bangla word images. In: International conference on computing: theory and applications, pp 427–433
8. Bhattad AJ, Chaudhuri B (2015) An approach for character segmentation of handwritten Bangla and Devanagari script. In: International conference on advance computing conference, pp 676–680
9. Bhowmick P, Bhattacharya BB (2007) Fast polygonal approximation of digital curves using relaxed straightness properties. IEEE Trans Pattern Anal Mach Intell 29(9):1590–1602
10. Bishnu A, Chaudhuri B (1999) Segmentation of Bangla handwritten text into characters by recursive contour following. In: International conference on document analysis and recognition, pp 402–405
11. Bunke H (2003) Recognition of cursive Roman handwriting: past, present and future. In: International conference on document analysis and recognition, pp 448–459
12. Casey RG, Lecolinet E (1995) Strategies in character segmentation: a survey. In: International conference on document analysis and recognition, vol 2, pp 1028–1033
13. Chaudhuri B, Pal U (1997) An OCR system to read two Indian language scripts: Bangla and Devnagari (Hindi). In: International conference on document analysis and recognition, vol 2, pp 1011–1015
14. Das N, Das B, Sarkar R, Basu S, Kundu M, Nasipuri M (2010) Handwritten Bangla basic and compound character recognition using MLP and SVM classifier. arXiv:1002.4040
15. Dershowitz N, Rosenberg A (2014) Arabic character recognition. In: Language, culture, computation. Computing-theory and technology, pp 584–602
16. Gao Y, Yang Y (2004) Survey of unconstrained handwritten Chinese character segmentation. Comput Eng 5:052
17. Garain U, Chaudhuri B (2002) Segmentation of touching characters in printed Devnagari and Bangla scripts using fuzzy multifactorial analysis. IEEE Trans Syst Man Cybern Part C Appl Rev 32(4):449–459

18. Hanmandlu M, Agrawal P (2005) A structural approach for segmentation of handwritten Hindi text. In: International conference on cognition and recognition, pp 589–597
19. https://en.wikipedia.org/wiki/Marathi_language. Accessed 23 Jan 2018
20. https://en.wikipedia.org/wiki/Punjabi_language. Accessed 23 Jan 2018
21. https://en.wikipedia.org/wiki/Bengali_language. Accessed 23 Jan 2018
22. Jawahar C, Kumar MP, Kiran SR (2003) A bilingual OCR for Hindi-Telugu documents and its applications. In: International conference on document analysis and recognition, pp 408–412
23. Jayadevan R, Kolhe SR, Patil PM, Pal U (2011) Offline recognition of Devanagari script: a survey. IEEE Trans Syst Man Cybern Part C Appl Rev 41(6):782–796
24. Khorsheed MS (2002) Off-line Arabic character recognition–a review. Pattern Anal Applic 5(1):31–45
25. Kumar V, Senegar PK (2010) Segmentation of printed text in Devnagari script and Gurmukhi script. Int J Comput Appl 3:24–29
26. Lehal GS (2009) A complete machine-printed Gurmukhi OCR system. In: Guide to OCR for Indic scripts, pp 43–71
27. Lehal GS, Singh C (2000) A Gurmukhi script recognition system. In: International conference on pattern recognition, vol 2, pp 557–560
28. Ma H, Doermann D (2003) Adaptive Hindi OCR using generalized Hausdorff image comparison. ACM Transactions on Asian Language Information Processing 2(3):193–218
29. Mangla P, Kaur H (2014) An end detection algorithm for segmentation of broken and touching characters in handwritten Gurumukhi word. In: International conference on reliability, infocom technologies and optimization, pp 1–4
30. Mohanty S, Dasbebartta HN, Behera TK (2009) An efficient bilingual optical character recognition (English-Oriya) system for printed documents. In: International conference on advances in pattern recognition, pp 398–401
31. Nawab NB, Hassan M (2012) Optical Bangla character recognition using chain-code. In: International conference on informatics, electronics & vision, pp 622–627
32. Obaidullah SM, Halder C, Santosh K, Das N, Roy K (2017) Phdindic_11: page-level handwritten document image dataset of 11 official Indic scripts for script identification. Multimed Tools Appl: 1–36
33. Otsu N (1979) A threshold selection method from gray-level histograms. IEEE Trans Syst Man Cybern 9(1):62–66
34. Pal U, Chaudhuri B (2004) Indian script character recognition: a survey. Pattern Recogn 37(9):1887–1899
35. Pal U, Datta S (2003) Segmentation of Bangla unconstrained handwritten text. In: International conference on document analysis and recognition, pp 1128–1132
36. Palakollu S, Dhir R, Rani R (2012) Handwritten Hindi text segmentation techniques for lines and characters. In: World congress on engineering and computer science, vol 1, pp 24–26
37. Patel C, Desai A (2010) Segmentation of text lines into words for Gujarati handwritten text. In: International conference on signal and image processing, pp 130–134
38. Pramanik R, Bag S (2018) Shape decomposition-based handwritten compound character recognition for Bangla OCR. J Vis Commun Image Represent 50:123–134
39. Pramanik R, Bag S (2017) Linear curve fitting-based headline estimation in handwritten words for Indian scripts. In: International conference on pattern recognition and machine intelligence, pp 116–123
40. Pramanik R, Raj V, Bag S (2018) Finding the optimum classifier: Classification of segmentable components in offline handwritten Devanagari words. In: International conference on recent advances in information technology, pp 1–5
41. Ramteke S, Gurjar A, Deshmukh D (2016) Automatic segmentation of content and noncontent based handwritten Marathi text document. In: International conference on global trends in signal processing, information computing and communication, pp 404–408
42. Roy A, Bhowmik TK, Parui SK, Roy U (2005) A novel approach to skew detection and character segmentation for handwritten Bangla words. In: Digital image computing: Techniques and applications, pp 30–38
43. Sarkar R, Das N, Basu S, Kundu M, Nasipuri M, Basu DK (2012) Cmaterdb1: a database of unconstrained handwritten Bangla and Bangla–English mixed script document image. Int J Doc Anal Recognit 15(1):71–83
44. Sarkar R, Sen B, Das N, Basu S (2015) Handwritten Devanagari script segmentation: A non-linear fuzzy approach. arXiv:1501.05472
45. Sharma DV, Lehal GS (2006) An iterative algorithm for segmentation of isolated handwritten words in Gurmukhi script. In: International conference on pattern recognition, vol 2, pp 1022–1025
46. Shinde AB, Dandawate YH (2014) Shirorekha extraction in character segmentation for printed Devanagri text in document image processing. In: Annual IEEE India conference, pp 1–7

47. Srivastav A, Sahu N (2016) Segmentation of Devanagari handwritten characters. Int J Comput Appl 142(14)
48. Wang SH, Phillips P, Dong ZC, Zhang YD (2018) Intelligent facial emotion recognition based on stationary wavelet entropy and Jaya algorithm. Neurocomputing 272:668–676
49. Zhang T, Suen CY (1984) A fast parallel algorithm for thinning digital patterns. Commun ACM 27(3):236–239
50. Zhang YD, Sun J (2018) Preliminary study on angiosperm genus classification by weight decay and combination of most abundant color index with fractional Fourier entropy. Multimed Tools Appl 77(17):22671–22688

Deepika Gupta received B.Tech degree in information technology in 2008 from Uttar Pradesh Technical University and M.Tech degree in 2011 from National Institute of Technology, Jaipur. She held Senior Software Engineer position in Samsung Electronics, India, (2011-2015). Currently, she is pursuing her Ph.D from Indian Institute of Technology (Indian School of Mines), Dhanbad. Her research interests include OCR for Indian Scripts and Image processing.

Soumen Bag received B.E. and M.Tech. degree in Computer Science & Engineering from NIT Durgapur in 2003 and 2008 respectively. He received his Ph.D. degree from IIT Kharagpur in 2013. Presently, he has been working as an Assistant Professor in the department of Computer Science & Engineering in IIT (ISM) Dhanbad. He is the recipient of Institute Gold medal for First Class for his Master's degree. He is also the recipient of different Fellowships/Scholarships from National and International Societies. He acts as an Organizing and Programme Committee members for different International Conferences. He is enlisted in the Marquis Who's Who in the World, USA, 32nd Ed., 2015. He is a life time member of IUPRAI. He is the author of several reputed International Journals and Conferences. His research interests are in the areas of OCR for Indian Scripts, Document Image Analysis, Image Processing, and Pattern Recognition.