Adaptive Hindi OCR Using Generalized Hausdorff Image Comparison

HUANFENG MA and DAVID DOERMANN
University of Maryland, College Park

We present an adaptive Hindi OCR implemented as part of a rapidly retargetable language tool effort. The system includes: script identification, character segmentation, training sample creation, and character recognition. In script identification, Hindi words are identified from bilingual or multilingual documents based on features of the Devanagari script or using Support Vector Machines. Identified words are then segmented into individual characters in the next step, where the composite characters are identified and further segmented based on the structural properties of the script and statistical information. Segmented characters are recognized using generalized Hausdorff image comparison (GHIC) and postprocessing is applied to improve the performance. The OCR system, which was designed and implemented in one month, was applied to a complete Hindi–English bilingual dictionary and a set of ideal images extracted from Hindi documents in PDF format. Experimental results show the recognition accuracy can reach 88% for noisy images and 95% for ideal images. The presented method can also be extended to design OCR systems for different scripts.

Categories and Subject Descriptors: I.5.2 [**Pattern Recognition**]: Design Methodology—*Classifier design and evaluation and pattern analysis*; I.7.5 [**Document and Text Processing**]: Document Capture—*Optical character recognition (OCR)*

General Terms: Design, Documentation, Experimentation, Languages

Additional Key Words and Phrases: Optical character recognition (OCR), generalized Hausdorff image comparison, script identification, document processing

1. INTRODUCTION

Digital document processing is gaining popularity for application to office and library automation, bank and postal services, publishing houses, and communication technology. An important task of automatic document processing is the reading of text. The procedure of automatically processing the text components of a complex document which contains text, graphics, and/or images can be divided into three stages: (1) region extraction and text region classification using document layout analysis; (2) text line, and possibly word (glyphs separated by white space) and character segmentation; and (3) optical character recognition (OCR). Typically, the OCR classifier stage needs to be redesigned

Author's address: H. Ma and D. Doerman, Language and Media Processing Laboratory, 3348 A.V. Williams Building, University of Maryland, College Park, MD 20742, email: hfma@umiacs.umd.edu.

for each new script, while the other stages are easier to port. OCR technology for some scripts like Roman and Chinese is fairly mature and commercial OCR systems are available with accuracy higher than 98%, including *OmniPage Pro* from ScanSoft or *FineReader* from ABBYY for Roman and Cyrillic scripts, and *THOCR* from Tsinghua University for Chinese.

Although commercial systems are available for Roman, Cyrillic, far east and many middle eastern languages, such systems for Indian scripts, as well as many low density languages are still in the research and development stage. In some cases, this is due to technical challenges, but more often it is due to a lack of a commercial market. Nevertheless, there is a real need for OCR in these languages.

The DARPA TIDES program is supporting a project at the University of Maryland that is focused, in part, on rapidly acquiring resources from printed resources such as bilingual dictionaries. With the large number of different languages all over the world, obtaining OCR systems of all these languages is unrealistic. We need to be prepared to retarget OCR systems to be able to deal with specific tasks including new languages or new scripts.

During a recent 'Surprise Language' task for TIDES that focused on Hindi, we were faced with the challenge of rapidly acquiring Hindi OCR capabilities. Since no feasible commercial OCR system was available, we set out to develop one as rapidly as possible. In this paper, we present a Devanagari (Hindi) OCR system using GHIC that was developed and trained in less than a month. Trained using character samples extracted from different documents, the OCR system can be easily adapted to perform Devanagari OCR of other fonts. Details of a complete system for segmenting, parsing, and tagging bilingual dictionaries can be found in Ma et al. [2003].

1.1 Background

Devanagari, an alphabetic script, is used by a number of Indian languages, including Sanskrit, Hindi, and Marathi. Many other Indian languages use close variants of this script. Although Sanskrit is an ancient language and is no longer spoken, written material still exists. Hindi is a direct descendant of Sanskrit through Prakrit and Apabhramsha, and has been influenced and enriched by Dravidian, Turkish, Farsi, Arabic, Portuguese, and English. It is the world's third most commonly used language after Chinese and English, and there are approximately 500 million people all over the world that speak and write in Hindi. Thus, research on Devanagari script, mainly the Hindi language, attracts a lot of interest. In the rest of this paper, Hindi, the language, and Devanagari, the script are used interchangeably.

Unlike English and other Roman script languages, Hindi has few, if any, commercial OCR renders; and the ones that have products provide only custom enterprise solutions. Chaudhuri and Pal proposed a Devanagari OCR system that was ultimately purchased and is being marketed as a custom solution, but is not yet available as an off the shelf product. The basic components of the system, however were described in the literature [Chaudhuri and Pal 1997a; Chaudhuri and Pal 1997b]. After word and character segmentation, a feature

based tree classifier was used to recognize the basic characters. Error detection and correction based on a dictionary search brought the recognition accuracy of the OCR to 91.25% at the word level and 97.18% at the character level on clean images. Bansal [1999], in his Ph.D. thesis, designed a Devanagari text recognition system by integrating knowledge sources, features of characters such as horizontal zero crossings, moments, aspect ratios, pixel density in nine-zones, number, and position of vertex points, with structural descriptions of characters. These were used to classify characters and perform recognition. After correction, based on dictionary search, the average accuracy was about 87% at the character level for scanned document images.

It should be noted that both of the OCR systems mentioned above need vast amounts of training data with ground truth to achieve acceptable levels of performance. Collection and ground-truthing of data is time consuming and labor intensive. Even so, before feeding a new font of a Hindi document to the OCR, the system must be retrained to obtain reasonable accuracy. In our application, we benefit from the need for only a small number of fonts for any given dictionary. In this paper, we propose an approach to quickly build a Hindi OCR. The segmentation of characters is similar to the approach proposed in Bansal [1999] with minor changes, and the recognition is based on the GHIC, which is like a template matching method but overcomes some disadvantages of the traditional template matching approach. This OCR does not need to be trained using a large number of training samples, and is easily adapted to different types of documents.

The rest of this paper is organized as follows: Section 1.2 describes the system architecture. Section 2.1 addresses how to identify the Devanagari words from bilingual or multilingual documents. Section 2.2 describes the procedure of character segmentation. Section 2.3 addresses the procedure of character recognition and some postprocessing techniques. Section 3 provides experimental results. Summary, conclusion, and future work are discussed in Section 4.

1.2 System Design

Our Hindi OCR, designed to work on pure Devanagari, or bilingual and multilingual document images with one script Devanagari, is shown in Figure 1. The system contains three different functional components: (1) document image preprocessing including denoising and deskewing; (2) segmentation and script identification at the word level; and (3) a classifier. In the following sections, we will briefly describe the word level script identification and focus on the design of the Hindi classifier.

The system first scans pages of Hindi text at 300 or 400 DPI. Images are first preprocessed with denoising and deskewing [Ittner and Baird 1993; Hull 1998]. An implementation of DOCTRUM [O'Gorman 1993] is applied to the preprocessed images to segment them into zones, text lines, and words. Components of the page are segmented into entries based on the functional features of documents using the approach described in Ma and Doerman [2003a]. Figure 2 shows the segmented dictionary entries. Script identification is applied to the segmented word images to identify Devanagari script and Roman

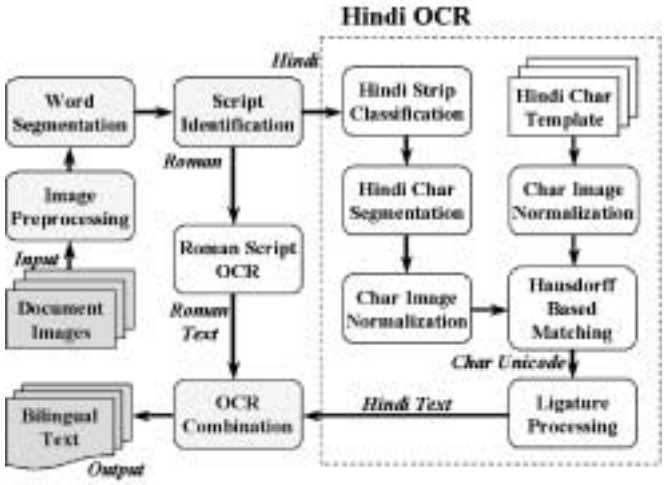


Fig. 1. System architecture.

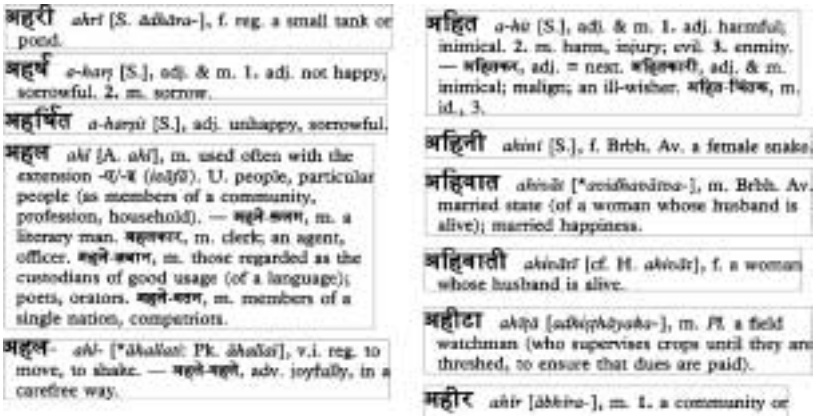


Fig. 2. Segmented entries of the Hind-English dictionary.

script words (including symbols neither Roman nor Devanagari). The identified Roman script word images are fed into a commercial English OCR, while the Hindi word images are first segmented into characters, and the character images are fed into a classifier to perform classification and recognition. After postprocessing, the output of the Hindi OCR is combined with the OCR output of Roman script to provide a complete result. The details of the approach and results are described in the following sections.

2. TECHNICAL APPROACH

2.1 Devanagari Script Identification

Before describing what types of features can be used to identify Devanagari script words from bilingual or multilingual document images, we examine the

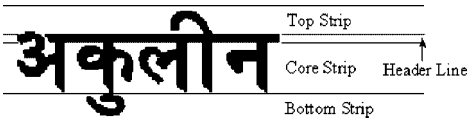


Fig. 3. Three strips of a Hindi word.

appearance of Devanagari script. Regular Hindi words can typically be divided into three strips: top, core, and bottom. For the Hindi word अकुलीन, for example, the five-character-word image is shown in Figure 3, where three strips are illustrated. The top strip and core strip are always separated by the header line, while there is no corresponding feature to separate the bottom strip and core strip. The top strip contains the top modifiers, and the bottom strip contains the lower modifiers. In a Hindi word, the top and bottom strips are not always necessary, but depend on the top and lower modifiers.

We proposed an approach to identify scripts at the word level based on the texture features, where the features were extracted using Gabor filters [Ma and Doerman 2003b]. The performance of the approach can be improved by applying Supported Vector Machines (SVM), as found in Ma and Doerman [2004]. The approaches in Ma and Doremann [2003b; 2004], however, are designed to be used to identify scripts under the assumption that the operator knows nothing about the non-Roman script, and the only feature used to do script identification is texture features extracted in 16 Gabor channels. For a specific script, some nontexture features can be used to improve the performance, if the system knows these features. The occurrence of a header line in a Hindi word is such a powerful feature that it can be used to identify Hindi words from bilingual or multilingual document images. For each segmented word with width W and height H, we compute the horizontal projection (denoted HP) of this word and then find the maximum value $HP_{\max}$ and the position $PS_{\max}$ of the maximum value. A word could be identified as a Hindi word if and only if

$$HP_{\max} > 0.8W$$
$$PS_{\max} > 0.5H$$

In some documents, single Roman character such as ***E***, ***e***, ***R***, ***T***, ***t***, ***I***, ***P***, ***D***, ***F***, ***l***, ***Z***, ***z***, or ***B*** in a specific font face may also satisfy the above two criteria, so these misidentification cases must be handled to improve the performance. Considering the fact that in a regular document, all these characters except ***I*** seldom appear as a single character, while ***I*** is usually much narrower than a single Hindi character, these misidentifications are removed by setting a word width threshold which depends on the font size and resolution of the document image. Figure 4 shows the performance comparison of two script identification approaches, SVM and the above-mentioned *script-oriented* approach on 20 randomly picked pages. The results show both of the approaches work effectively with average accuracy higher than 93%, and the latter is much higher with average accuracy of 98.94%.

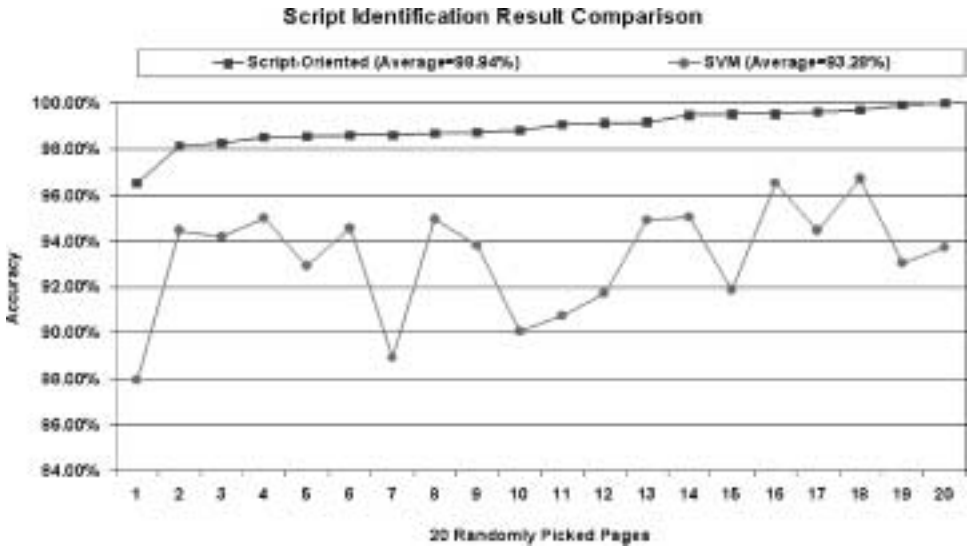


Fig. 4. Accuracy comparison of two script identification approaches (random pages, sorted by script-oriented accuracy).

Table I. Vowels and Corresponding Modifiers

Vowels	अ	आ	इ	ई	उ	ऊ	ऋ	ए	ऐ	ओ	औ
Modifiers		ा	ि	ी	ु	ू	ृ	े	ै	ो	ौ

2.2 Character Segmentation

2.2.1 *Devanagari Script Overview*. Devanagari has about 11 vowels (shown in the first row of Table I) and about 33 consonants (shown in Table II). Each vowel except अ corresponds to a modifier symbol as shown in the second row of Table I. In Hindi, when consonants are combined with other consonants, the consonant with a vertical bar may appear as a *half form*. Except for the characters क and फ, the half forms of consonants are the left part of original consonants with the vertical bar and the part to the right of the bar removed. These half consonants are shown in Table III, where the order of characters corresponds to the character order in Table II. Table IV gives some examples of combinations of half consonants with other consonants. The combination of half consonants and other consonants are not always left-right structured. Sometimes the combination orientation is top-down, or even reorganized to become a new character. Some of these examples of special combinations are shown in Table V. In addition to these special combinations, some special Hindi symbols are shown in Table VI. It should be noted that the list of special combinations is far from complete, so handling all these cases needs to be addressed. In Section 2.3.5, we will address how to deal with these special cases with an operator's feedback.

Table II. Consonants

क	ख	ग	घ	ङ	च	छ	ज	झ	ञ	ट
ठ	ड	ढ	ण	त	थ	द	ध	न	प	फ
ब	भ	म	य	र	ल	व	श	ष	स	ह

Table III. Half-Forms of Consonants With a Vertical Bar

क	ख	ग	घ		च		ज	झ	ञ	
			ण	त	थ		ध	न	प	फ
ब	भ	म	य		ल	व	श	ष	स	

Table IV. Examples of Combination of Half-Consonants and Consonants.

क क्क	क ल क्ल	घ न घ्न	च ञ च्ञ	ञ च ञ्च	त न त्न	प त प्त	प ल प्ल
ब व ब्व	भ न भ्न	म ल म्ल	ल ल ल्ल	श न श्न	श ब श्ब	श ल श्ल	स न स्न

Table V. Examples of Special Combination of Half-Consonants and Consonants

क ष क्ष	ज ञ ज्ञ	ट ट ट्ट	ट ठ ट्ठ	त र त्र	द द द्द
द ध द्ध	द व द्व	द व र द्व	श र श्र	द भ द्भ	द य द्य

Table VI. Special Symbols

क़	ख़	ग़	ज़	फ़	ड़	ढ़	ं	ँ	ः	ा	ऽ	ॅ

Fig. 5. The procedure of Hindi character segmentation.

2.2.2 *Hindi Character Segmentation.* The procedure to segment a Hindi word into characters (including core characters, and top and lower modifiers) is illustrated in Figure 5 using the segmentation of the Hindi word स्वीकृति as an example. The numbered arrows in Figure 5 represent the steps of segmentation, and the characters with solid bounding boxes are the final segmentation results.

The procedure to perform character segmentation can be described as follows:

Step 1: Locate the header line and separate the core-bottom strip that contains the core strip and bottom strip, and a top strip that contains the header line and the top modifiers.

Step 2: Identify the core strip and the bottom strip from the core-bottom strip, and extract the lower modifiers.

Step 3: Separate the core strip into characters that may contain conjunct/shadow characters.

Step 4: Segment the conjunct/shadow characters into single characters.

Step 5: Remove the header line from the top strip and extract the top modifiers.

Step 6: Put the header line back to the segmented core characters.

The details for each step are described below. We denote the width of the Hindi word bounding box as W, the height as H, and the coordinates of the left-top corner are set to be (0,0).

Step 1: *Separate the top strip and the core-bottom strip.* The separation of the top strip and the core-bottom strip is based on the location of the header line. For each word, we compute the HP and find the row (with Y-coordinate y) having the maximum value of HP. This is the candidate of the header line position. A header line candidate can be the real header line if $y \leq 0.4H$. If this condition is not satisfied, then set the HP value of this row to 0 and research the row with the maximum value until a real header line position is located. The maximum HP value is marked as $HP_{\max}$, and the position of the header line is marked as *hPosition*. Setting *hPosition* as the center, traverse the adjacent 10 HP values at each side of hPosition, and find the continuous rows whose HP values are all greater than $0.8\ HP_{\max}$. The number of these continuous rows is the stroke width of this word, which is marked as *StrokeWidth*, and important for the postprocessing of segmentation. hPosition is updated as the first row's Y-coordinates of the header line. The header line separates the Hindi word into the top strip, including the header line, and the core-bottom strip. This procedure is shown in step 1 of Figure 5.

Step 2: *Identify the core strip and the bottom strip from the core-bottom strip.* This procedure is briefly shown in step 2 of Figure 5. Denoting the width and height of the core-bottom strip obtained in the last step as W_{cb} and H_{cb}, and using the Hindi word खुशनुमा, which contains two lower modifiers, the detailed procedure is shown in Figure 6, by dividing it into the following steps:

(1) Compute the vertical projection (VP) VP_{cb} of the core-bottom strip (Figure 6(a)).

(2) The columns with no black pixels separate the Hindi word into several character candidates, which may contain conjunct/shadow characters or even incorrectly segmented characters (Figure 6(b)).

(3) Find the maximum height of these characters and denote it $H_{\max}$. The separated characters are divided into three groups. The first group contains all characters with height greater than $0.8H_{\max}$, the second group contains

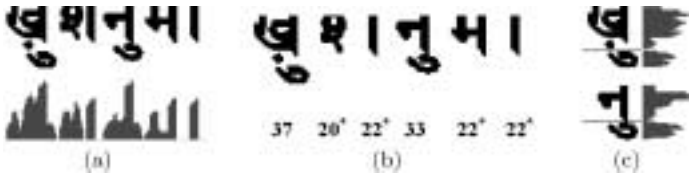


Fig. 6. Extraction of the lower modifiers from the core-bottom strip. (a) The core-bottom strip and its VP; (b) Separated characters based on the VP, the number under each character is its height, and numbers with '*' are used to compute the threshold $h_{\mathrm{Th}} = 22$. Note that the second character is segmented into two characters incorrectly, while that does not affect the final result; (c) Two characters with a lower modifier and their HPs, where the two straight lines denote the separation positions.

characters whose height is between $0.8H_{\max}$ and $0.64H_{\max}$, and the remaining characters are put into the third group. The group with the maximum number of members is considered to contain normal characters without the lower modifiers, and the maximum height of members in this group is set as a threshold h_{Th}. If $(H_{\mathrm{cb}} - h_{\mathrm{Th}}) \geq H_{\mathrm{cb}}/4$, the word contains at least one lower modifier (Figure 6(b)).

(4) For each separated character with a lower modifier, compute its horizontal projection HP_{cb}.

(5) In the HP_{cb} obtained in the last step, setting h_{Th} as the center, traverse the adjacent five values at each side of h_{Th}. The row with the minimum HP_{cb} value is the boundary which segments the core-bottom strip character into the core character and the lower modifier (Figure 6(c)).

Step 3: *Separate the core strip into characters.* In this step, the core strip is separated into characters, and the conjunct/shadow characters, which need further segmentation, will be determined as well. We borrow the definition of shadow character from Bansal and Sinha [2002]. A character is said to be under the shadow of another character if they do not physically touch each other but it is impossible to separate them merely by drawing a vertical line. In their paper, Bansal and Sinha proposed an approach to separate the core strip into characters and determine the conjunct/shadow characters based on the statistical information (such as average width, minimum and maximum width) of characters on the text line. The approach obviously cannot model our case because in the bilingual documents, Hindi words and English words are usually interlaced. It is impossible to obtain one Hindi text line that contains words with the same size. Therefore, we separate the Hindi word into characters and determine conjunct/shadow characters based on the statistical information obtained from the current Hindi word. Before extracting the information, it must be noted that the modifier ◌ृ in the Hindi word has a much smaller width than the regular characters after removing the Header line, so this character cannot be applied in the computation of statistical information of character width. Fortunately, this character is easily located based on the obtained stroke width in the first step. The separation of the core strip and the determination of conjunct/shadow characters are shown in step 3 of Figure 5, where one conjunct

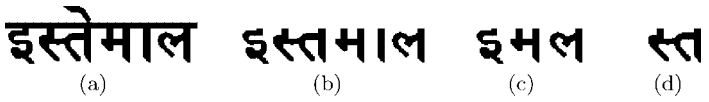


Fig. 7. Conjunct/shadow character determination. (a) Original word image (located header line provides StrokeWidth = 6); (b) Five characters separated based on the VP, with width 26, 51, 28, 7, 32 respectively; (c) Three characters used to compute the average width, with width 26, 28, 32 respectively, where $W_{\min} = 26$ and $W_{\text{avg}} = 28.7$; (d) Detected conjunct character (with width 51).

character is located. Taking the segmentation of another Hindi word इस्तेमाल as an example, the detailed procedure is shown in Figure 7 and can be describe as

(1) Using the method in step 2, separate the core strip into characters based on the VP (Figure 7(b)).
(2) For each separated character, if its width is smaller than 2 StrokeWidth, consider this as ा and remove it.
(3) Find the minimum width of remaining characters and denote this as $W_{\min}$.
(4) For each remaining character, if its width is greater than $1.5W_{\min}$, remove this character because it may be a conjunct/shadow character which can affect the statistical information significantly (Figure 7(c)).
(5) After removing the too-narrow and too-wide characters, compute the average width of the remaining characters and denote it as W_{avg}.
(6) Traverse all the separated characters, each character with width wider than $1.2W_{\text{avg}}$ is considered a conjunct/shadow character which needs further segmentation (Figure 7(d)).

Step 4: *Segmentation of the conjunct/shadow character.* The segmentation of a conjunct character is complicated, and because of the different characteristics of conjunct and shadow characters, the segmentation operations of the conjunct character and the shadow character are different. They are described as follows.

Segmentation of the conjunct character. The basic idea to segment the conjunct character is to find the segmentation column from both the right and the left sides of the word image and then determine the final segmentation position by comparing these two segmentation columns. After examining all the consonants, we found the following four observations:

(1) In each conjunct character, the right part is a full consonant that is wider than the left part, and the left part is always a half consonant.
(2) For each consonant that can be combined with a half consonant to create a conjunct character, after removing the header line, the vertical bar and the right part to the right of the vertical bar (if there is a vertical bar), the *HP* of the remaining part is always connected without any discontinuity.
(3) Neither of the two parts of a conjunct character can be too short.
(4) The pixel strength in the touching column of the two characters is usually less than that of other columns.

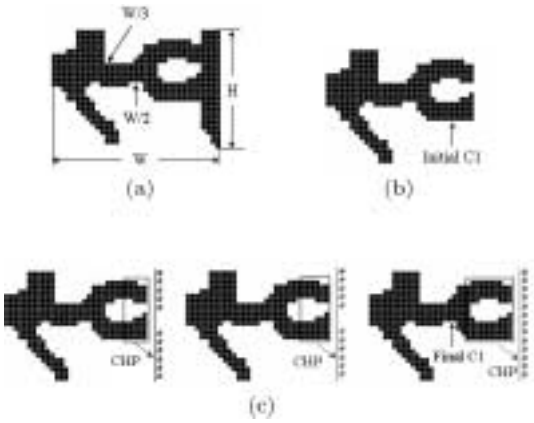


Fig. 8. Segmentation of the conjunct character (to find C1). (a) The conjunct character image; (b) The remaining character image with vertical bar removed; (c) Steps to search for C1.

So the design of the segmentation algorithm that contains three steps is based on the above four observations. In the first step, segmentation column C1 is located by examining the right part of the conjunct character image (based on observations (1), (2), and (3)). Then in the second step, segmentation column C2 is located by examining the left part of the conjunct character image (based on observations (1), (3), and (4)). In the last step, the final segmentation column C is determined by comparing C1 and C2.

In this paper, we use the same idea of computation of the collapsed horizontal projection (CHP), which was defined by Bansal and Sinha in their paper to detect the continuity of an inscribed image. The basic idea of CHP is: for each row of the inscribed image, if one foreground pixel can be found, then set the projection of this row 1; otherwise set the projection of this row 0. The detailed definition of this concept can be found in Bansal [1999] and Bansal and Sinha [2002]. The operations in the three steps are described in the following.

Locate the segmentation column C1: Illustrated in Figure 8, the procedure to locate the segmentation column C1 can be described as

(1) Check if there exists a vertical bar in the right part of the image by computing the VP of the right half part of the conjunct image.
(2) If there is a vertical bar, the bar and the image part to the right of this bar are removed from the conjunct image. The new image is shown in Figure 8(b).
(3) Suppose the right boundary of this new image is *nRight*, then the initial C1 is set at one stroke width to the left of nRight. Inscribing the right part of the image between C1 and nRight, the CHP of the inscribed image is computed.
(4) If the CHP has no discontinuity and the inscribed image is higher than $H/3$, C1 is the segmentation column and stop the searching procedure.
(5) Otherwise, shift C1 one column left and repeat the above computation until the C1 that satisfies the above criteria is found.

Fig. 9. Segmentation of a conjunct character (to find C2).

Fig. 10. Example of conjunct character with two vertical bars.

This searching procedure is shown in Figure 8(c), where the first two inscribed images have discontinuity, and the final C1 is shown in the last image.

Locate the segmentation column C2: As we mentioned in Section 2.2.1, most of the half consonants have no vertical bar, but there exist two consonants क and फ whose half forms also have a vertical bar. The appearance of the vertical bar can affect the final segmentation column searching result. So before setting the initial segmentation column C2, we need to check the occurrence of the vertical bar in the left half part of this conjunct image based on the VP. If no vertical bar is found in the left part of the image, the initial C2 is set at $W/3$, where W is the width of this conjunct image. The searching of the segmentation column C2 is described as

(1) Suppose the left boundary of this conjunct image is *nLeft*, the height of the inscribed image between *nLeft* and $W/3$ is computed.
(2) If the computed height is less than $H/3$, then C2 is shifted one column right. If the inscribed image is higher than $H/3$, then C2 is shifted one column right only if the pixel strength of the new column is not greater than the present column. The pixel strength of the column is defined as the number of black pixels in this column.
(3) Iterate the above steps until a segmentation column C2 that meets the requirement is located.

This procedure is shown in Figure 9, where the inscribed image is always higher than $H/3$, and the strength for the next column is shown in all three subfigures.

If there is a vertical bar in the left part of the image, suppose the right column of the bar has location *bRight*, then the initial segmentation column C2 is set as *bRight*+1. Since the height of the inscribed image between *nLeft* and C2 is always higher than $H/3$ in this case, C2 is shifted one column right only if the pixel strength of the new column is not greater than the present column. One example is shown in Figure 10.

Considering the observation (1), C2 must be smaller than $W/2$, which puts another stop condition for the determination of C2.

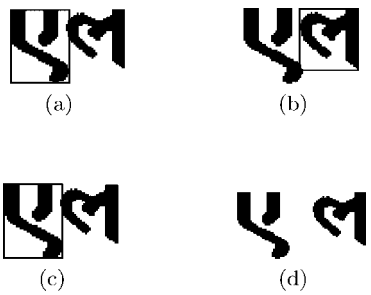


Fig. 11. Segmentation of the shadow character. (a) Determination of the shadow character; (b) Bounding box of the connected component (the right character); (c) Bounding box of the left character; (d) Segmented characters.

Determine the segmentation column C by comparing C1 and C2: If a detected conjunct character is a real conjunct character, the found segmentation columns C1 and C2 should be very close. And considering the stop conditions of the searching iterations of C1 and C2, C1 cannot be less than C2 for a real conjunct character. So in this step, the decision of segmentation column C is made based on the following three situations that could be encountered:

(a) C1 is less than C2: Detected character is not a real conjunct character, so no further segmentation is needed.
(b) C1 is greater than C2 and C1, C2 are very close: If the difference between C1 and C2 is less than the stroke width, then the segmentation column C is set as the average of C1 and C2.
(c) C1 is one or more than one stroke width larger than C2: The segmentation column C is set as the column that is one stroke width left of C1, and only the right part will be extracted. The remaining left part will be considered as a new conjunct character image and further segmentation is needed.

Segmentation of the shadow character. The detection of a shadow character is straightforward. First, we find the left most pixel of the character image. Then the connected component starting from this pixel is detected and the bounding box of this connected component is computed. If the right value of this bounding box is less than the right value of the original character image, then this character is considered a shadow character, which needs further segmentation. The segmentation of a shadow character is clearly shown in Figure 11. There are not many shadow cases in the Hindi words. Usually in the shadow character image, the right character is a character that can be represented as one single connected component. So the segmentation of a shadow character starts from the right side of the image. First, we find the right most black pixel of the image, then find the connected component starting from this pixel using 8-neighbor tracing. The connected component is considered the right character and separated from the original image. The left character is the remaining part with the detected connected component removed.

It should be noted that the above mentioned segmentation can also be extended to the segmentation of the shadow top modifiers (the lower modifiers

Fig. 12. Examples of shadow top modifiers.

usually do not have shadow situations). Three examples of shadowed top modifiers are shown in Figure 12.

Step 5: *Extract the top modifiers.* The extraction of the top modifiers from the top strip (show in step 5 in Figure 5) is simple and straightforward. The header line is removed from the top strip first, then the VP of the remaining strip is computed. The boundary of a top modifier is located based on the column without any black pixels. There are some special cases that two top modifiers may touch each other, and they are separated as one single top modifier. Further segmentation of the top modifiers will be handled as a special case, which is described in Section 2.3.5.

Step 6: *Put the header line back into the segmented characters.* This step is straightforward, where the header line is put back to each segmented core character for the recognition in the next step.

In the above description of operations in all steps, there are some constants defined. Some of the constants depend on the natural characteristics of the Hindi character, such as the factors 0.4, 0.8, 0.64, 1.2, 1/4, 1/3, and 1/2, which are usually fixed even for different fonts or different sizes of Hindi words. There are also some constants (such as the 5 and 10 when traversing the projection profile), which depend on the sizes of Hindi words, we chose these constants based on the experimental results, they can be fixed as long as the font has the standard font size used in regular documents, or they can be changed based on the new font size.

2.3 Recognition

In a typical OCR system, feature extraction is probably the most important step to achieve high performance of character recognition. Devijver and Kittler [1982] defined feature extraction as the problem of "extracting from the raw data the information which is most relevant for classification purposes, in the sense of minimizing the within-class pattern variability while enhancing the between-class pattern variability." Considering the fact that there typically exist many variations of the same character, a good feature for character recognition should be invariant to transformations such as scale and rotation. Some feature extraction methods work on grayscale images, whereas others work on binarized images, vector images (thinned skeletons), or outer symbol contour. Trier et al. presented an overview of feature extraction methods for recognition of segmented (isolated) characters [Trier et al. 1996]. OCR approaches can be briefly classified into template matching, transform, zoning, or moment based.

The recognition of Hindi characters is based on the Hausdorff image comparison. Huttenlocher et al. [1993] proposed efficient algorithms of computing Hausdorff distance to compare the resemblance between two binary images with the assumption that there is only a translation between two images. In this paper, the GHIC is applied to determine the resemblance of one segmented character (a point set) to another character (a template point set), by examining

the fraction of points in one set that lie near points in the other set, and vice versa. There are two parameters used to determine the degree of resemblance of two point sets: (i) the maximum distance that points can be separated and still be considered close together; and (ii) what fraction of the points in one set are at most this distance away from points of the other set. Hausdorff-based distance measures are different from the correspondence-based matching techniques in that there is no pairing of points in the two sets being compared [Huttenlocher et al. 1993]. Often in matching and recognition problems, the two images are allowed to undergo some kind of geometric transformation in the matching process. In this case we are concerned with finding the transformations of one image (character image) that produces good matches to the other image (template image). In the following subsections, we give a brief introduction of GHIC.

2.3.1 *Generalized Hausdorff Image Comparison.* Given two sets of points $A = \{a_1, \ldots, a_m\}$ and $B = \{b_1, \ldots, b_n\}$, the Hausdorff distance is defined as

$$H(A, B) = \max(h(A, B), h(B, A)),$$

where $h(A, B) = \max_{a \in A} \min_{b \in B} \|a - b\|$. The function $h(A, B)$ is called the "directed Hausdorff distance" from A to B (this function is not symmetric and thus is not a true distance). It identifies the point $a \in A$ that is farthest from any point of B, and measures the distance from a to its nearest neighbor in B. Thus the Hausdorff distance, $H(A, B)$, is actually used to measure the degree of mismatch between two image point sets, as it reflects the distance of the point of A that is farthest from any point of B and vice versa.

The Hausdorff distance is very sensitive to even a single *outlying* point of A or B. For example, consider $A = B \cup x$, where the point x is some large distance D from any point of A. In this case $H(A, B) = D$, which is determined solely by the point x. Considering the fact that scanning images are often noisy with different quality, directed Hausdorff distance is not powerful enough to provide a satisfying match between two character images. Therefore, when performing the recognition, rather than using $H(A, B)$, a generalization of the Hausdorff distance (which does not obey the metric properties on A and B, but does obey them on specific subsets of A and B) is used. This generalized Hausdorff measure is given by taking the kth ranked distance rather than the maximum, or largest ranked one:

$$h_k(A, B) = k^{\text{th}}_{a \in A} \min_{b \in B} \|a - b\|,$$

where kth denotes the kth ranked value (or equivalently the quantile of m values). For example, when $k = m$, then kth is max. When $k = m/2$, then the median of the m individual point distances determines the overall distance. Therefore, this measure generalizes the directed Hausdorff measure, by replacing the maximum with a quantile.

For character recognition, we are interested in using the Hausdorff distance to measure the similarity of one image bitmap I (the character image) with some 'model' bitmaps M (character templates or samples), under the assumption that only translation transformation can exist between these two matched bitmaps. In other words, we seek all translations $t \in R^2$ such that $h_k(M + t, I) \leq \delta$. The

parameter k tells us how many of the model points should be near image points in order to classify a given translation as a potential matching instance of the model (i.e., we allow $m-k$ of the m model points to be outliers). The parameter δ tells us how close each nonoutlying model point must be to some image point. In our work, the parameter k is dependent on the scanning image quality, while parameter δ is determined based on the shape variability of same characters in different context.

In order to find each translation such that $h_k(M+t, I) \leq \delta$, we form $I' = I + C_\delta$ and then compute the correlation of M with I'. For each translation t of M with respect to I', the correlation determines p, how many points of $M+t$ are superimposed with I' (the logical *and* of $M+t$ and I'). If the point number p of one translation is greater than or equals k (i.e., $p \geq k$), then the current model M with translation t is considered a match to I, that is, $h_k(M+t, I) \leq \delta$ is satisfied. We refer to p/m as the Hausdorff fraction for a given translation t (at some fixed δ). The Hausdorff fraction measures the percentage of $M+t$ that lies near (within δ) points of I, which in some sense provides the degree of similarity of two image point sets and can be used to provide the confidence value of recognition.

The computation of $h_k(M+t, I) \leq \delta$ alone does not necessarily find good matching of M in I, rather it finds portions of I that could contain M plus some other points. For instance, with black the foreground pixel, a totally black image will match any model at any translation. Thus, we absolutely need to use the other direction of the generalized Hausdorff measure, $h_k(M+t, I)$, to 'verify' those translations where it was found that $h_k(M+t, I) \leq \delta$. This reverse Hausdorff fraction, ensures that a given portion of the points in the image (covered by the model array) are actually near points of $M+t$. It thereby rules out situations such as a totally black image match any other images.

The details of the GHIC can be found in Huttenlocher et al. [1993]. In our recognition, the forward and reverse distance thresholds are specified to compare the resemblance between a character and the templates. Under the constraint of the two thresholds, there may still be more than one character that satisfies the conditions, thus the forward and reverse Hausdorff fractions are used to prune the character candidates. Furthermore, in our work, we also set two thresholds of these two fractions. The sum of the forward and reverse Hausdorff fraction is used to compute the confidence (half of the sum which has value between 0.0 and 1.0) of character recognition, and the confidence value is used for follow on processing, which will be discussed in Section 4.

2.3.2 *Normalization of Template and Character Images.* Since we only consider the translation when computing the generalized Hausdorff image measure to perform recognition, there is no way that the same character in different sizes could be matched. One solution to solve the scaling problem is image normalization, that is, the template and character images fed into the system must be normalized before computing the Hausdorff distance. In our work, we employ a simple normalization based on the long edges of images, which normalize all core characters into images with long edge 32, and normalize all top and lower modifiers into images with long edge 16 while preserving the aspect ratio.

Table VII. Classes of Core Hindi Characters

Open header	अ थ ध भ
One conjunction end bar	च ज ञ त न च ज्ञ
More conjunctions end bar	ख घ झ प म य ष स ख़
Middle bar	ऋ क फ क़ फ़
No bar	इ उ ऊ ए ङ छ ट ठ ड ढ द र ह ड़ ढ़
Special case	ग ण श ग़

Denoting the image's dimensions as W (width) and H (height), the normalization procedure is:

$D_{\max} = \max(W, H)$
if low or top modifiers, **then**
Factor $= 16/D_{\max}$
else
Factor $= 32/D_{\max}$
$NW = W \times$ Factor, $NH = H \times$ Factor
Resize the image into a new image with size $NW \times NH$

The constants 16 and 32 are not very important in the normalization step, we can also change this value into a variable which is dependent on the average size of character images.

2.3.3 *Classification of Characters.* A vertical bar does not appear at the left end of a Hindi character. If a vertical bar is present, it either appears at the right end (End Bar) or in the middle (Middle Bar) of a Hindi character. Based on the presence and position of the vertical bar and the conjunction number of the character with the header line, all the core Hindi characters can be divided into the following six groups shown in Table VII.

The four characters ग, ण, श, ग़ in the *Special Case* class are characters with an end bar, but after removing the header line and computing the VP, each of these four characters will be split into two parts. The over-segmentation is handled in the next subsection. The character ए in the *No Bar* class is also special because it is the only *No Bar* character which has more than one conjunction with the header line.

2.3.4 *Over-Segmentation Processing.* In the above character segmentation procedure, there could exist over-segmentation of characters, that is, one single character could be segmented into two or more parts. Based on the direction of over-segmentation, we divide this into two types, horizontal and vertical direction, and handle them separately.

The over-segmentation in the vertical direction only happens to long characters such as ए ह and other strongly combined forms of consonants such as छ, द्ध, द्व, द्ध, and ङ्ग. After being segmented into two parts, the bottom part is usually rejected or incorrectly recognized during the procedure of recognition. This type of over-segmentation is handled by adding the over-segmented top part into the templates and assigning them special codes. Once recognized, we know this is an over-segmented character, its following part will be put back.

(a) (b)

Fig. 13. Examples of over-segmented characters added in the template. (a) Over-segmented characters; (b) Original characters.

Fig. 14. Examples of touching modifiers.

For example, Figure 13(a) shows some of the over-segmented characters we added to our templates, and characters in Figure 13(b) are the original complete forms.

The handling of over-segmentation in the horizontal direction is a little more complex because of the half consonant. Taking the four characters in the *Special Case* class, their half forms happen to be the left part of the over-segmented parts. The determination of an over-segmented character depends on the result that if the following character is character ा or not. In these four cases, if they are over-segmented, then the next character can only be character ा, but this cannot be determined until the next character has been recognized. We leave the handling of this type of over-segmentation to the 'Ligature processing' section.

2.3.5 *Dealing with Special Characters.* As mentioned above, it is not possible to add all of the special characters to the template because many of them are rarely used. In addition, adding more templates will significantly slow down the recognition. In our work, we provide a very simple scheme which allows the operator to add new special characters easily, which means the recognizer can easily be tuned to adapt to new cases. If the operator finds one new character, he first classifies this character (end bar, middle bar, no bar, and so on) and puts this into the corresponding file that includes the template name and unicode of this character. Then he simply cuts this character from the image, saves it into a TIFF file and puts the filename into the training template directory. The recognizer will automatically read templates from the file and perform recognition. The recognition of all of the special characters listed in Tables V and VI were handled in our system.

Due to the quality of scanned document images, two or more top modifiers may touch each other, which cannot be segmented using the approach described above. Some of these touched top modifiers are shown in Figure 14. These touched top modifiers are considered as special top modifiers, which are added to the class of top modifiers. Fortunately, there are not many such cases, so this case can be easily handled.

2.4 Ligature Processing

Devanagari characters, like characters from many other scripts, can be combined or change shape depending on their context. A character's appearance is affected by its relation to other characters, the font used to render the character,

and the application or system environment. These special characters are handled in Section 2.3.5.

Additionally, a few Devanagari characters may result in a change in the order of the displayed characters. This reordering is not commonly seen in non-Indic scripts. One such character is ि, which is always displayed one consonant left of its real position. When exporting the codes of one Hindi word with such a character, the codes of characters must be reordered.

The Devanagari script is noted for a large number of consonant conjunct forms that serve as orthographical abbreviations (ligatures) of two or more adjacent letter forms. This abbreviation takes place only in the context of a consonant cluster.

Some independent characters such as ई, ऐ, ओ, औ, अं, ो, and ौ have a top strip which is segmented and classified as top modifiers. When exporting the encoding of these characters, the top modifier should be put back into the corresponding core character to generate a correct code.

Independent characters such as आ, ओ, औ will be segmented into अ plus ा, ो, ौ. When exporting the encoding of these characters, the separated parts should also be put back to generate one correct single code.

All the above schemes are handled in our system. Since most of them are caused by the ligature of Devanagari script, we discuss this in the 'Ligature processing' part.

3. EXPERIMENTAL RESULTS

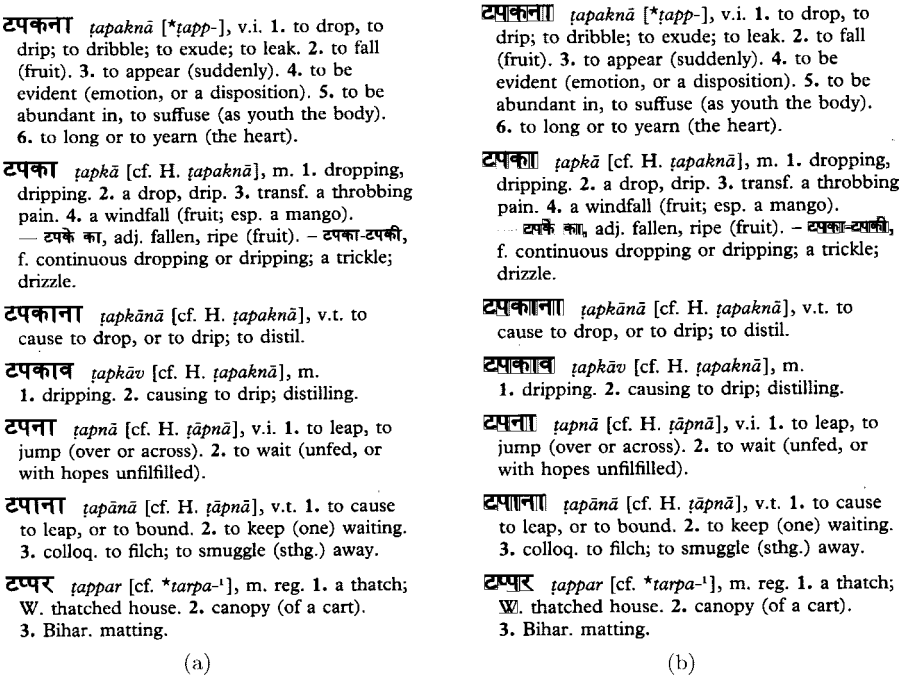
3.1 OCR Evaluation

The proposed system was applied to the 1083 pages of the Oxford Hindi–English dictionary [McGregor 1993] and to a collection of PDF-converted Hindi document images. The dictionary binding was burst and scanned at 400 DPI. The PDF-converted Hindi document images are obtained directly from the PDF without the introduction of scanner noise, so they are considered ideal images.

An example of the scanned dictionary image is shown in Figure 15, where (a) is the original image, and (b) shows the identified Hindi words (with errors) and the segmented characters. The OCR result which combines the Hindi OCR and the Roman OCR is shown in Figure 16.

To evaluate the accuracy of this OCR, we randomly chose seven pages and counted the number of Hindi words and characters recognized. The result evaluation is shown in Table VIII.

From Figure 15, it can be seen that the identified Hindi words can be correctly segmented into isolated characters using the proposed approach. The evaluation result shown in Table VIII shows the recognition accuracy at the character level reaches 87.75%, while the accuracy at the word level reaches about 67%. The experiment was done on scanned images, which obviously contain noise, and the result is the pure recognition result without any spell checking and word correction based on dictionary search. We strongly believe, with the availability of Hindi language constraints and electronic text, correction techniques



टपकना *ṭapaknā* [**ṭapp-*], v.i. **1.** to drop, to drip; to dribble; to exude; to leak. **2.** to fall (fruit). **3.** to appear (suddenly). **4.** to be evident (emotion, or a disposition). **5.** to be abundant in, to suffuse (as youth the body). **6.** to long or to yearn (the heart).

टपका *ṭapkā* [cf. H. *ṭapaknā*], m. **1.** dropping, dripping. **2.** a drop, drip. **3.** transf. a throbbing pain. **4.** a windfall (fruit; esp. a mango). — टपके का, adj. fallen, ripe (fruit). – टपका-टपकी, f. continuous dropping or dripping; a trickle; drizzle.

टपकाना *ṭapkānā* [cf. H. *ṭapaknā*], v.t. to cause to drop, or to drip; to distil.

टपकाव *ṭapkāv* [cf. H. *ṭapaknā*], m. **1.** dripping. **2.** causing to drip; distilling.

टपना *ṭapnā* [cf. H. *ṭāpnā*], v.i. **1.** to leap, to jump (over or across). **2.** to wait (unfed, or with hopes unfilfilled).

टपाना *ṭapānā* [cf. H. *ṭāpnā*], v.t. **1.** to cause to leap, or to bound. **2.** to keep (one) waiting. **3.** colloq. to filch; to smuggle (sthg.) away.

टप्पर *ṭappar* [cf. **tarpa*-[1]], m. reg. **1.** a thatch; W. thatched house. **2.** canopy (of a cart). **3.** Bihar. matting.

(a)

टपकना *ṭapaknā* [**ṭapp-*], v.i. **1.** to drop, to drip; to dribble; to exude; to leak. **2.** to fall (fruit). **3.** to appear (suddenly). **4.** to be evident (emotion, or a disposition). **5.** to be abundant in, to suffuse (as youth the body). **6.** to long or to yearn (the heart).

टपका *ṭapkā* [cf. H. *ṭapaknā*], m. **1.** dropping, dripping. **2.** a drop, drip. **3.** transf. a throbbing pain. **4.** a windfall (fruit; esp. a mango). — टपके का, adj. fallen, ripe (fruit). – टपका-टपकी, f. continuous dropping or dripping; a trickle; drizzle.

टपकाना *ṭapkānā* [cf. H. *ṭapaknā*], v.t. to cause to drop, or to drip; to distil.

टपकाव *ṭapkāv* [cf. H. *ṭapaknā*], m. **1.** dripping. **2.** causing to drip; distilling.

टपना *ṭapnā* [cf. H. *ṭāpnā*], v.i. **1.** to leap, to jump (over or across). **2.** to wait (unfed, or with hopes unfilfilled).

टपाना *ṭapānā* [cf. H. *ṭāpnā*], v.t. **1.** to cause to leap, or to bound. **2.** to keep (one) waiting. **3.** colloq. to filch; to smuggle (sthg.) away.

टप्पर *ṭappar* [cf. **tarpa*-[1]], m. reg. **1.** a thatch; W. thatched house. **2.** canopy (of a cart). **3.** Bihar. matting.

(b)

Fig. 15. One example of the bilingual dictionary. (a) Original image; (b) Identified Hindi words and character segmentation.

टपकना *tapaknd [*tapp-], v.i. 1. to drop, to drip; to dribble; to exude; to leak. 2. to fall (fruit). 3. to appear (suddenly). 4. to be evident (emotion, or a disposition). 5. to be abundant in, to suffuse (as youth the body). 6. to long or to yearn (the heart).*

टपका *tapka [cf. H. tapakna], m. 1. dropping, dripping. 2. a drop, drip. 3. transf. a throbbing pain. 4. a windfall (fruit; esp. a mango).*

- टपके का, *adj. fallen, ripe (fruit).* - टपकाटपकी, *f. continuous dropping or dripping; a trickle; drizzle.*

टपकाव *tapkav [cf. H. tapaknd], m. 1. dripping. 2. causing to drip; distilling.*

टपना *tapnd [cf. H. tdpnd], v.i. 1. to leap, to jump (over or across). 2. to wait (unfed, or with hopes unfilfilled).*

टपाना *tapand [cf. H. tapna], v.t. 1. to cause to leap, or to bound. 2. to keep (one) waiting. 3. colloq. to filch; to smuggle (sthg.) away.*

टप्पर *tappar [cf. *tarpa-'], m. reg. 1. a thatch; श्श thatched house. 2. canopy (of a cart). 3. Bihar. matting.*

Fig. 16. OCR results of images shown in Figure 15 (Reconstructed by combing Hindi and Latin results).

Table VIII. Result Evaluation of the Hindi–English Dictionary

Pages	Chars	Recognized	Correct	A1	A2	Words	Correct	A
p0098	451	443	407	90.24%	91.87%	110	79	71.82%
p0160	317	311	272	85.80%	87.46%	73	54	73.97%
p0179	480	477	409	85.21%	85.74%	113	67	59.29%
p0401	294	290	264	89.80%	91.03%	71	53	74.65%
p0799	437	451	379	86.73%	84.04%	80	50	62.50%
p0987	405	402	359	88.64%	89.30%	67	39	58.20%
p1023	343	338	303	88.34%	89.64%	64	44	68.75%
Total	2727	2712	2393	**87.75%**	**88.24%**	578	386	**66.78%**

"A1" is the character accuracy with respect to "Chars" and "A2" is the character accuracy with respect to "Recognized". "A" is the word accuracy.

can be applied to the postprocessing stage of this system to improve the performance.

As we mentioned in Section 1.1, Bansal proposed a Hindi text recognition system by integrating knowledge sources. After correction, based on dictionary search, the average accuracy is about 87% at the character level for scanned document images. This result is comparable with the performance of our system without any correction. The recognition accuracy of the Hindi OCR system proposed by Chaudhuri and Pal can achieve 91.25% at the word level and 97.18% at the character level. However, this accuracy was obtained on clean images with error detection and correction based on dictionary search.

To test the effectiveness of the proposed approach working on clean images, we processed PDF converted ideal clean images and evaluated them. The accuracy is about 95% (with 2584 characters and 2450 correctly recognized for one page). Figure 17 shows part of one converted clean image and its OCR result.

3.2 Discussion

In examining the data, we found that a number of factors contributed to the incorrect recognition including:

(1) *Incorrect word segmentation*. The word segmentation performance is dependent on the quality of document images. Noise may cause the incorrect under- or over-segmentation of words or merging of words and other symbols. Incorrect word segmentation can further affect the script identification result, which leads to the degradation of OCR performance.

(2) *Incorrect character segmentation*. Segmentation is a challenging task, especially for Hindi scanned images. Due to the appearance of noise, average width and height of characters may not be obtained accurately. During segmentation, many decisions such as identifying conjunct/shadow characters, determining lower modifiers, and determining stroke width, are made based on these statistics. Incorrect statistics can significantly degrade the performance of character segmentation, and furthermore degrade the final performance of recognition.

(3) *Missing punctuation such as commas, periods, and parentheses.* Often the space between the words and the punctuation that follows is small.

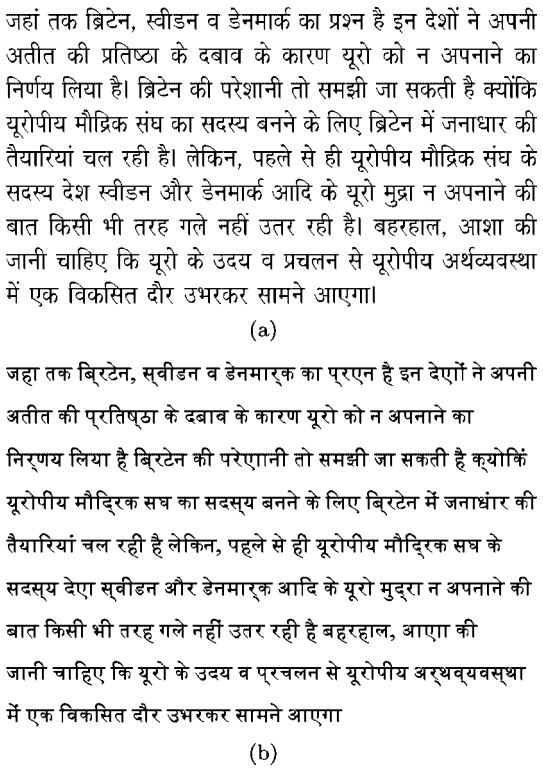


Fig. 17. OCR result of the PDF converted ideal image. (a) Original image; (b) OCR result.

Punctuation can therefore be merged with Hindi words during word segmentation. The direct result is these symbols can negatively influence the distribution of the features used to perform segmentation. Although we tried to handle this by detecting these symbols first, there are still some cases that were not detected correctly.

(4) *Character misclassification due to noise.* This typically happens with *Open Header* and *Middle Bar* characters. Noise may cause an *Open Header* character to become a closed header character, or cause the detected vertical bar in a *Middle Bar* character to shift. The classes typically do not overlap, so if one character is misclassified, it most likely will not be recognized correctly.

(5) *Character similarity.* There exist Hindi characters with similar appearances. Sometimes the scanning noise makes them almost indistinguishable. During recognition, they may have the same confidence value, which can then make the final selection of OCR output of this character ambiguous. Higher level context or language models may help fix this problem.

(6) *Special symbols which are noise-like.* There are some special symbols (including notations) which are visually noise-like, although the position of these symbols relative to the rest of the text provides strong context. Sometimes these symbols are removed as noise, othertimes noise is left resulting

in incorrect modifiers. This causes incorrect classification and degrades the performance of the OCR.

To improve the performance of the OCR, we need to consider all the factors discussed above and strive to remove their effects. Since a majority of the incorrect recognition was caused by noise, applying new denoising techniques such as the approach in Zheng et al. [2003], or making the parameters more flexible with varying image quality can improve the performance. For the incorrect recognition caused by misclassification, more rules, or new classes could be added to make the classification more accurate.

As a postprocessing step, there are a number of known ways to increase the accuracy of OCR for free text, and most center around the use of either general or domain specific lexicons. The recognized terms are looked up in the lexicon and if they are present, no further analysis is required. If they are not present, however, we must typically assume there is an OCR error and take steps to try to select the correct term. Often this is accomplished by using a distance measure between terms, typically based on character recognition confusion probabilities.

In our application, we are using OCR to process bilingual dictionaries and these types of documents introduce several inherent problems for OCR correction. First, we are dealing with a source that by nature has very few instances of some words, yet is fairly complete in its coverage of the language. In the case of Hindi, the coverage of the lexicons we have is not complete enough to warrant the use of statistical correction approaches. Although we have significant amounts of electronic text that can be used to generate a lexicon, it is well known that most naturally occurring text provides only a limited coverage of the language. In our case we have performed experiments with a limited lexicon, but not surprisingly, the overall recognition rates decrease. When a term is spelled correctly, but does not appear in the lexicon, we actually introduce more errors by mapping these words to incorrect ones.

The second problem is somewhat of a chicken and egg problem. Since we do not have ground truth information (and it was not feasible to get in the 30 days of this effort), it is difficult to estimate the OCR confusion probabilities needed for intelligent distance measures. We are currently exploring various character level correction schemes, but they also rely on a statistical distribution of character bi- or tri-grams that may not be accurate for dictionaries.

In general, if we are trying to produce OCR systems for low density languages, large amounts of electronic text may not be available, so we are exploring other ways to semi-interactively identify common confusions.

4. CONCLUSION AND FUTURE WORK

We have presented an adaptive Hindi OCR system which uses a GHIC implemented as part of a rapidly retargetable language tool effort. The system includes three stages: (i) script identification; (ii) character segmentation; and (iii) training sample creation and character recognition. Based on the GHIC, the system is easy to retarget to different Hindi fonts or even a different script, provided the segmentation can be applied using the same or a similar approach. The OCR is also designed to handle unknown special characters, and provides

a simple interactive interface to allow the user to add them. The OCR (designed and implemented in one month) was applied to a complete Hindi–English bilingual dictionary and a set of ideal images extracted from Hindi documents in PDF format. Experimental results show the average recognition accuracy was 87.82%, while for ideal images, the accuracy was 95%, both at the character level, without any spell checking.

A major thrust of future work will be to perform OCR correction or to resolve ambiguity among the candidates. One advantage of our approach is that we give confidence as a side effect of recognition. By setting two thresholds on the Hausdorff distance, the forward and reverse Hausdorff fractions under the constraint of these two thresholds can be used to compute the confidence of a character and word recognition. This confidence is a real value between 0.0 and 1.0, which is much more intuitive and usable than current commercial OCR software such as *ScanSoft Developer's Kit 2000* from ScanSoft and *FineReader Engine* from ABBYY. For these packages, the confidence of each character is a boolean value, which gives the same weight to all characters when computing the confidence of a word.

Given the confidence of characters and words, we can further consider the word correction based on dictionary search. The word correction engine would determine whether a word needed to be replaced with another correct word coming from the dictionary, which can significantly improve the recognition performance at both the character and the word level. Since there is the possibility that the recognized result may be over-corrected, the word correction can also provide a probability for the replacement, which makes the correction easily tuned. Details of this correction can be found in Kolak and Resnik [2002] and Kolak et al. [2003].

Another advantage of our approach is that we can adapt to different image qualities. For example, if the scanned document image quality is poor, the Hausdorff thresholds can be set to lower values, which makes the classifier and recognizer more tolerant to allow more choices. While if the image quality is high, the thresholds can be set to higher values, which can speed up the recognition process.

The next step for the recognition phase is to apply new, possibly multiclassifier techniques, and combine them with the current Hausdorff classifier to provide improved performance. The approach assumes we can train the system using a small number of samples, so the new classification techniques must also have this property.

Our OCR system was designed under the assumption that no vast amount of training samples are available, so it can be easily extended for the recognition of other languages or scripts under the following two conditions: (i) the language uses symbols of the same basic class; (ii) the words of this new language can be segmented into characters. Once glyphs are segmented, the same classifiers can be trained and used for recognition. Segmentation, however, is very different between Hindi and other non-Indic languages. For Chinese, segmentation is straightforward because character spacing is fixed. For many Latin fonts, kerning must be considered, while language such as Arabic must consider touching characters during segmentation. Under the two assumption conditions, the

segmentation can be changed based on the characteristics of the new script, while the recognition will be adapted through exemplars, if necessary. The user can create new rules to classify characters, obtain samples from document images, set output codes for each character, set thresholds for the Hausdorff distance, and perform recognition. If it is difficult to extract features, which can be used to classify characters, the operator can put the whole set of characters into one single class, compute the Hausdorff distance between the segmented characters and each character in the single class, and then perform recognition. Although the system was designed specifically to deal with Hindi text, it was modularized so that we can pull out different components to use for other languages. Overall, our goal is to build a toolkit of components that can be reused for rapidly building OCR capabilities for new languages.

Dealing with special characters can also be extended for the recognition of some symbols in cases that are difficult to segment correctly (such as complex ligatures). The user can add these parts into the samples and perform recognition. The only thing the user needs to do before performing recognition is to make sure the encoding of these special characters is correct.

Finally, a key to making our system generally adaptive is to consider how to allow system parameters to dynamically adjust for changes in image quality.

ACKNOWLEDGMENTS

The support of this research under DARPA cooperative agreement N660010028910 and DOD contract MDA90402C0406 is gratefully acknowledged. Thanks to Vishal Khandelwal and Armrita Misra for doing the evaluation and explaining the encoding of special Hindi symbols.

REFERENCES

BANSAL, V. 1999. Integrating Knowledge Sources in Devanagari Text Recognition. Ph.D. thesis, Indian Institute of Technology, Kanpur, India.

BANSAL, V. AND SINHA, R. 2002. Segmentation of touching and fused Devanagari characters. *Pattern Recognition 35*, 875–893.

CHAUDHURI, B. AND PAL, U. 1997a. An OCR system to read two Indian language scripts: Bangla and Devanagari (Hindi). In *Proceedings of the Fourth International Conference on Document Analysis and Recognition* (Germany). 1011–1016.

CHAUDHURI, B. AND PAL, U. 1997b. Skew angle detection of digitized Indian script documents. *IEEE Transactions on Pattern Analysis and Machine Intelligence 19*, 2, 182–186.

DEVIJVER, P. AND KITTLER, J. 1982. *Pattern Recognition: A Statistical Approach*. Prentice-Hall, London.

HULL, J. J. 1998. Document image skew detection: Survey and annotated bibliography. In *Document Analysis Systems II*, J. J. Hull and S. L. Taylor, Eds. Word Scientific, Singapore.

HUTTENLOCHER, D. P., KLANDERMAN, G. A., AND RUCKLIDGE, W. J. 1993. Comparing images using the Hausdorff distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence 15*, 9, 850–863.

ITTNER, D. J. AND BAIRD, H. S. 1993. Language-free layout analysis. In *IAPR Second International Conference on Document Analysis and Recognition* (Tsukuba Science City, Japan). 336–340.

KOLAK, O. AND RESNIK, P. 2002. OCR error correction using a noisy channel model. In *Human Language Technology Conference (HLT 2002)*.

KOLAK, O., RESNIK, P., AND BYRNE, W. 2003. A generative probabilistic OCR model for NLP applications. In *HLT-NAACL 2003*, in press.

MA, H. AND DOERMANN, D. 2003a. Bootstrapping structured page segmentation. In *SPIE Conference Document Recognition and Retrieval* (Santa Clara, CA). 179–188.

MA, H. AND DOERMANN, D. 2003b. Gabor filter based multi-class classifier for scanned document images. In *Seventh International Conference on Document Analysis and Recognition (ICDAR)* (Edinburgh, Scotland). 968–972.

MA, H. AND DOERMANN, D. 2004. Word level script identification for scanned document images. In *SPIE Conference Document Recognition and Retrieval* (San Jose, CA), in press.

MA, H., KARAGOL-AYAN, B., DOERMANN, D., WANG, J., AND OARD, D. 2003. Parsing and tagging of bilingual dictionaries. *Traitement Automatique Des Languages*, in press.

MCGREGOR, R. 1993. *The OXFORD Hindi–English Dictionary*. Oxford University Press, Delhi, ISBN 0-19-864339-X.

O'GORMAN, L. 1993. The document spectrum for page layout analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence 15*, 11, 1162–1173.

TRIER, Q. D., JAIN, A. K., AND TAXT, T. 1996. Feature extraction methods for character recognition—A survey. *Pattern Recognition 29*, 4, 641–662.

ZHENG, Y., LI, H., AND DOERMANN, D. 2003. Text identification in noisy document images using markov random field. In *Seventh International Conference on Document Analysis and Recognition (ICDAR)* (Edinburgh, Scotland). 599–605.

Received August 15, 2003; revised September 28, 2003; accepted October 24, 2003