Pergamon

Pattern Recognition, Vol. 31, No. 5, pp. 531–549, 1998
© 1998 Pattern Recognition Society. Published by Elsevier Science Ltd
All rights reserved. Printed in Great Britain
0031–3203/98 $19.00 + .00

PII: S0031-3203(97)00078-2

A COMPLETE PRINTED *BANGLA* OCR SYSTEM

B. B. CHAUDHURI* and U. PAL

Computer Vision and Pattern Recognition Unit, Indian Statistical Institute,
203 B. T. Road, Calcutta 700 035, India

(*Received* 13 *May* 1996; *accepted* 3 *July* 1997)

Abstract—A complete Optical Character Recognition (OCR) system for printed *Bangla*, the fourth most popular script in the world, is presented. This is the first OCR system among all script forms used in the Indian sub-continent. The problem is difficult because (i) there are about 300 *basic*, *modified* and *compound* character shapes in the script, (ii) the characters in a word are topologically connected and (iii) *Bangla* is an inflectional language. In our system the document image captured by Flat-bed scanner is subject to skew correction, text graphics separation, line segmentation, zone detection, word and character segmentation using some conventional and some newly developed techniques. From zonal information and shape characteristics, the basic, modified and compound characters are separated for the convenience of classification. The basic and modified characters which are about 75 in number and which occupy about 96% of the text corpus, are recognized by a structural-feature-based tree classifier. The compound characters are recognized by a tree classifier followed by template-matching approach. The feature detection is simple and robust where preprocessing like thinning and pruning are avoided. The character unigram statistics is used to make the tree classifier efficient. Several heuristics are also used to speed up the template matching approach. A dictionary-based error-correction scheme has been used where separate dictionaries are compiled for *root word* and *suffixes* that contain morpho-syntactic informations as well. For single font clear documents 95.50% word level (which is equivalent to 99.10% character level) recognition accuracy has been obtained. Extension of the work to *Devnagari*, the third most popular script in the world, is also discussed. © 1998 Pattern Recognition Society. Published by Elsevier Science Ltd. All rights reserved

Optical character recognition Document processing Language processing Spell-checker Indian language

1. INTRODUCTION

The object of optical character recognition (OCR) is automatic reading of optically sensed document text materials to translate human-readable characters to machine-readable codes. Research in OCR is popular for its various application potentials in banks, post-offices and defense organizations. Other applications involve reading aid for the blind, library automation, language processing and multi-media design.

The origin of character recognition can be found in 1870 when Carey invented the retina scanner, an image-transmission system using a mosaic of photo-cells.[1] Later in 1890 Nipkow invented the sequential scanner which was a major breakthrough both for modern television and reading machines. Character recognition as an aid to the visually handicapped was at first attempted by the Russian scientist Tyurin in 1900.

The OCR technology took a major turn in the middle of 1950s with the development of digital computer and improved scanning devices. For the first time OCR was realized as a data processing approach, with particular applications to the business world. From that perspective, David Shepard, founder of the Intelligent Machine Research Co. can be considered as a pioneer of the development of commercial OCR equipment. Currently, PC-based systems are commercially available to read printed documents of single font with very high accuracy and documents of multiple fonts with reasonable accuracy. The book edited by Wang[2] contains a list of eight prominent vendors of commercial systems.

However, most of the available systems work on European scripts which are based on Roman alphabet.[3–7] Research reports on oriental language scripts are few, except for Korean, Chinese and Japanese scripts.[8]

We are concerned here with the recognition of *Bangla*, the second most popular script and language in the Indian sub-continent. About 200 million people of Eastern India and Bangladesh use this language, making it the fourth most popular in the world. A few reports are available on the recognition of isolate Indian script character[9–11] but none deal with a complete OCR for printed documents. From that standpoint this paper is a pioneering work among all Indian scripts. Moreover, as described in Section 9, most of the methods developed here can be directly applied for the recognition of *Devnagari* script (in which Hindi, the third most popular language in the world, is written). This paper will be useful for the computer recognition of several other Indian script

* Author to whom correspondence should be addressed. E-mail: bbc@isical.ernet.in.

forms since these scripts, having the same ancient origin (see Section 2.1), show some structural similarities.

In *Bangla*, the number of characters is large and two or more characters combine to form new character shapes called *compound or clustered characters*. As a result, the total number of characters to be recognized is about 300. Also, for the inflectional nature of this language, it is difficult to design a simple OCR error-correction module. Thus, *Bangla* OCR development is more difficult than that of any European language script.

Feature matching and template matching are two basic approaches to character recognition. Our method is predominantly a feature-based one, although a sophisticated template matching approach is used at some stage to detect some compound characters.

The organization of this paper is as follows. In Section 2 properties of *Bangla* script and language as well as some statistical analysis of *Bangla* text corpus are presented. Text digitization and skew-correction techniques are described in Section 3. Section 4 deals with line, word and character segmentation. Initial classification of characters in three groups and feature selection and detection are elaborated in Sections 5 and 6, respectively. Section 7 describes the character recognition procedure. The error detection and correction method is presented in Section 8. Lastly, the conclusion and scope of further work are described in Section 9.

2. PROPERTIES OF *BANGLA* SCRIPT AND WORD MORPHOLOGY

Since *Bangla* is an oriental script form, we describe some of its important properties for the benefit of wider readership. In this perspective some statistical analyses are also presented here.

2.1. *Properties of* Bangla *script*

- The *Bangla* script is derived from the ancient *Brahmi* script through various transformations. Other Indian scripts also have the same origin, making many of them similar in shape.
- There are 11 vowel and 39 consonant characters in modern *Bangla* alphabet. They are called *basic characters*. The basic characters and ASCII codes used to represent them are shown in Fig. 1. The concept of upper/lower case is absent in *Bangla*.
- From Fig. 1 it is noted that for many characters there exit a horizontal line at the upper part. We shall call it as *head line*. The head line is an important feature to locate the script line, to segment the characters in a word and to classify the characters using our tree classifier. Some characters have a signature extended above the head line (e.g. ই ঈ ঊ), which is also useful for character classification.
- The vowel A(অ) following a consonant character is never printed in a word. Thus, A(অ) can occur only at the beginning of the word.
- A vowel (other than A(অ)) following a consonant takes a modified shape, depending on the position of the vowel: to the left, right (or both) or bottom of the consonant. These are called *modified characters or allographs*. See Fig. 2a for modified vowel shapes and their attachment with a consonant character K(ক). The modified shape of vowel O(ও) and (AU) (ঔ) have two parts. One sits to the left and other to the right of a consonant (or compound) character (e.g. see the last two columns of Fig. 2a). Thus, the

অ A	আ A,	ই I	ঈ I,	উ U	ঊ U,	ঋ R.	এ E	ঐ (AI)	ও O
ঔ (AU)	ক K	খ KH	গ G	ঘ GH	ঙ N,	চ C	ছ CH	জ J	ঝ JH
ঞ N+	ট T,	ঠ T,H	ড D,	ঢ D,H	ণ N*	ত T	থ TH	দ D	ধ DH
ন N	প P	ফ PH	ব B	ভ BH	ম M	য J,	র R	ল L	শ S.
ষ S,	স S	হ H.	ড় R,	ঢ় R*	য় Y	ৎ T.	ং M,	ঃ H,	ঁ N.

Fig. 1. *Bangla* alphabet basic shape with codes used to describe them. (The first 11 characters are vowels while others are consonants.)

Vowel	আ	ই	ঈ	উ	ঊ	ঋ	এ	ঐ	ও	ঔ
Modified Shape	া	ি	ী	ু	ূ	ৃ	ে	ৈ	ো	ৌ
With number	1	2	3	4	5	6	7	8	9	10
When attached to Consonant ক	কা	কি	কী	কু	কূ	কৃ	কে	কৈ	কো	কৌ

(a)

Consonant	গ	র	শ	হ	র	হ
Vowel	উ	উ	উ	উ	ঊ	ঋ
Compound character	গু	রু	শু	হু	রূ	হৃ

(b)

Consonant	য	র	র
Modified shape	্য	্র	◌
with number	1	2	3
When attached to	দ+য	দ+র	র+দ
consonant দ	দ্য	দ্র	র্দ

(c)

Fig. 2. Examples of vowel and consonant modifiers: (a) vowel modifiers, (b) exceptional cases of vowel modifiers and (c) consonant modifiers.

Bangla script is not strictly a left to right alphabetical sequence. If the first character of the word is a vowel then it retains its *basic* shape. For two consecutive vowels in a word, the second one also retains its *basic* shape. The vowels U(ঊ), U,(ঊ) and R.(ঋ) may take different modified shapes when attached to some consonant characters. They also change the shape of some consonant characters to which they are attached. For illustration see Fig. 2b.

- In some situations as shown in Fig. 2c a consonant following (preceding) a consonant is represented by a modifier called *consonant modifier*.
- A word may be partitioned into three zones. The *upper zone* denotes the portion above the head line, the *middle zone* covers the portion of basic (and compound) characters below head line and the *lower zone* is the portion where some of the modifiers can reside. The imaginary line separating middle and lower zone is called the *base line*. A typical zoning is shown in Fig. 3.
- More formally speaking, modifiers are those symbols which do not disturb the shape of the basic characters (in middle zone) to which they are attached. If the shape is disturbed in the middle zone, we call the resultant shape as *compound character* shape. Compounding of two constants is most abundant although three consonants can also be compounded. There are about 250 compound characters of which a sub-set of 100 characters are shown in Fig. 4.
- The total number of basic, modified and compound characters is about 300. Although the writing style is from left to right, a word may not be partitioned vertically into a left–right character sequence (e.g. for the modifier of vowel O(ও) one part sits to the left and one to the right of the consonant). Thus, a delayed decision is necessary during recognition of some characters.
- As compared to *Roman*, the font and style variations in *Bangla* script are few. Popular font for

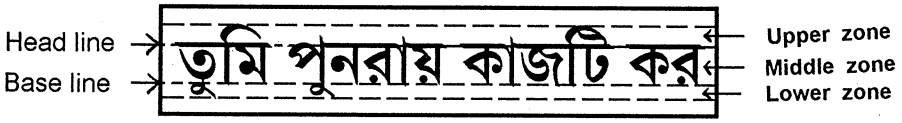


Fig. 3. Different zones of a *Bangla* text line.

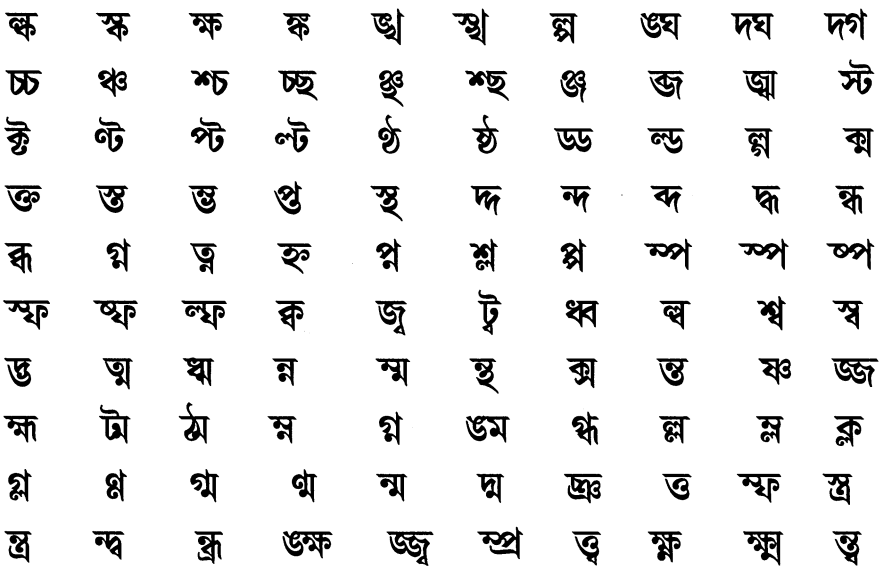


Fig. 4. A subset of 100 compound characters. Compound characters of last row are formed by combining of three consonants.

mass printing media is *Linotype*. Recently, some fonts are proposed for desk top publication (DTP) using PC, but none of them popular. The size of characters in most *Bangla* documents range between 8 and 16 points of which 12 points is most popular. Among style variations, occasionally bold, expanded and rarely italicized style are used.

2.2. *Brief* Bangla *word morphology*

Bangla is an inflectional language. In the text, a *Bangla* word may consist of an optional prefix, a simple or compounded stem of root words, zero or more internal affixes and an optional termination declension. The set of words that can be used in *Bangla* text may be called *surface words*. The words having no *suffix* may be called *root words*. The major affix classes in *Bangla* are verbal and nominal declensions called *Bibhaktis*, verbal and other internal affixes called *Pratyayas*, nominal suffixes and prefixes called *anusargas* and *upasargas*.

Because of inflectional nature, a huge number of surface words can be formed out of a moderate number of root words. For example, in a dictionary of 50,000 root words there are about 800 single worded verbs. There are about 125 verb suffixes (neglecting dialectical variations) of which a particular verb can accept about 100 suffixes to form valid *surface words*. Thus, about 80,000 surface verb words can be formed. The number of non-verb suffixes is also of similar order. Naturally, the storage and search time could be reduced if we could have separate *root word dictionary* and *suffix dictionary* files. However, surface word formation from root and suffixes, as well as surface word parsing into root words and suffixes are both difficult because a large number of rules and their exceptions should be considered. For our OCR error detection and correction, which is based on dictionary search, we simplified the procedure to a large extent by making the dictionary such that

(1) If a root word can accept a prefix then the prefixed root word is considered as another root word. Conjoining of root words by *Sandhi* and *Samasa* are also considered as new root words;
(2) If morphological deformation is caused in a root word in order to accept a set of particular suffixes, then the deformed root word is considered as another root word.

Thus, we have our root words and suffixes such that any surface word is a simple concatenation of a single root word string followed by (optional) single suffix string. In doing so, we can avoid a large number of rule verification during OCR error detection and correction without appreciable increase in the size of root word dictionary.

2.3. *Some statistical analysis of* Bangla *script*

Statistical analysis from a corpus of *Bangla* text is useful and important in designing the OCR apart from other applications in Natural Language Processing and Cryptography. Since no such analysis is available in the literature, we collected two sets of data, the first set containing 100,000 words from popular *Bangla* books, newspapers and juvenile literatures and the second set containing 60,000 words from a *Bangla* dictionary.[12] Each datum is a word converted into ASCII codes of Fig. 1. A typical example is A/NT//RGH/A,TMU,LK. Here, /NT/ denotes that N and T together make a compound character. The words in the data sets were stored in alphabetical order. For the first set of data each distinct word is stored with its frequency of occurrence.

At first, global statistics of characters were computed. Table 1 shows some interesting comparisons. In commonly used language (as in current newspaper, literature) the vowel occurrence is larger and the compound character occurrence is smaller than that in the dictionary. In other words, we have a tendency of using words which are easy to spell and mellowed to listen. Also, in the present day *Bangla* text the compound character occurrence is in the range of 4–6% only, the rest being basic characters and modifiers. Hence, compared to basic characters we can assign secondary importance to compound character recognition in our OCR design. The average word-length as well as the entropy of characters were also computed.

Next, global occurrence of (vowel, consonant and compound) characters were computed. When represented as a fraction of the total number of characters in the data set, it gives an estimate of *a priori* probability of the character. In Table 2 these estimates are ranked and shown in percentage of occurrence. It gives an idea about which characters should be recognized more accurately to obtain higher recognition rate. Also, this statistics guides us to design a tree classifier for character recognition, as discussed in Section 7.

To see how dictionary look-up approach can be effective for error correction in *Bangla* OCR, we grouped the character pair that are graphemically identical. The OCR errors lie mostly within these pairs. Now, if a character of a word is misrecognized as another identical character and if such a character makes a valid word, then dictionary look-up cannot detect the error. By a computer simulation we noted that the total number of such cases is about 1.01% only and hence the dictionary look-up error detection technique can be quite effective for *Bangla* OCR.

Some phonetic statistics of *Bangla* words and a comparative study with graphemic statistics were

Table 1. Global character statistics of 1st and 2nd set of data

	Global character statistics	1st set of data	2nd set of data
1	Vowel characters	38.70%	37.10%
2	Consonant characters	61.30%	62.90%
3	Compound characters	4.10%	7.20%
4	Entropy of the alphabet	4.07(bits/char)	3.54(bits/char)
5	Average word length	5.73 (char)	4.99(char)
6	Words with suffix	69.80%	0.00%

Table 2. First 50 frequently occurring characters with their rank (from 1st data set)

Rank	% of occur.	Char. code	Rank	% of occur.	Char. code	Rank	% of occur.	Char. code
1	12.888	A,	18	1.501	I,	35	0.406	/KS,/
2	9.068	E	19	1.284	J	36	0.405	U,
3	7.542	I	20	1.260	S.	37	0.403	M,
4	7.257	R	21	1.236	G	38	0.375	R.
5	4.215	N	22	1.019	C	39	0.321	TH
6	4.037	K	23	1.007	H.	40	0.319	S,
7	3.454	B	24	0.948	BH	41	0.279	D,
8	3.340	L	25	0.850	CH	42	0.277	/TR/
9	3.025	T	26	0.814	A	43	0.254	GH
10	3.006	U	27	0.774	KH	44	0.224	(AI)
11	2.766	S	28	0.730	R,	45	0.224	T,H
12	2.663	M	29	0.659	/PR/	46	0.187	/BY/
13	2.549	O	30	0.559	N*	47	0.174	/RB/
14	2.327	P	31	0.559	DH	48	0.173	/KT/
15	1.996	D	32	0.542	N.	49	0.165	/NT/
16	1.875	Y	33	0.441	J,	50	0.154	/STH/
17	1.581	T,	34	0.414	PH			

made.[13] This study is useful in text to speech synthesis and talking OCR design which is also a part of our project goals. For brevity, the results are not presented here.

3. TEXT DIGITIZATION AND SKEW CORRECTION

To use an OCR system, at first the document is scanned and a digital image is formed. Next, noise is cleaned and the image skew, if any, is corrected. Also, for a complex document the text regions are separated from the non-text regions. The procedures used in our system for these tasks are given below.

3.1. Text digitization and noise cleaning

Text digitization can be done either by a Flat-bed scanner or a hand-held scanner. The resolution of an optical reader varies from 100 to 900 dots per inch (dpi). Hand-held scanner typically has a low resolution range. We preferred a Flat-bed scanner (manufactured by Truvel, Model number TZ-3BWC) for digitization because the output of hand-held scanner is affected by hand movement creating local fluctuations. Also, most of the commercial OCRs use at least a Flat-bed scanning device.

The digitized images are in gray tone and we have used a histogram-based thresholding approach to convert them into two-tone images. For a clear document the histogram shows two prominent peaks corresponding to white and black regions. The threshold value is chosen as the midpoint of the two histogram peaks. The two-tone image is converted into 0–1 labels where 1 and 0 represent object and background, respectively.

We have tried other thresholding techniques like those described by Kapur, Sahoo and Wong.[14] Otsu,[15] etc. and note that our simple approach gives a result similar to or better than these methods. See Fig. 5, for comparison.

The digitized image shows protrusions and dents in the characters as well as isolated black pixels over the background, which are cleaned by a logical smoothing approach.

3.2. Skew detection and correction

Casual use of the scanner may lead to skew in the document image. Skew angle is the angle that the text lines of the document image makes with the horizontal direction. Skew correction can be achieved in two steps, namely, (a) estimation of skew angle θ_s and (b) rotation of the image by θ_s in the opposite direction. Skew correction is necessary for the success of any OCR system.

There exist a wide variety of skew detection algorithms based on projection profile,[16,17] Hough transform,[18] Docstrum,[19] line correlation,[20] etc. We propose a new approach based on the observation of head line of *Bangla* script. If a digitized document is skewed then the head line also satisfies the properties of a skewed *digital straight line* (DSL) which, if detected, gives estimate of the skew angle. Compared to others, this approach is accurate, robust and computationally efficient. An advantage of this approach is that the head line information can be subsequently used for detection of word upper zone, segmentation of characters, etc.

Our technique starts with connected component labeling. For each labeled component its boundary box (minimum upright rectangle containing the component) is defined. We select only those components whose bounding box widths are greater than the average of bounding box width of all components in the document. Hence, small and irrelevant components like dots, punctuation markers and characters without head line, etc. are mostly filtered out. See Fig. 6b for illustration. Let the set of such selected components be S_1.

Next, we find the upper envelope of the components of S_1. Consider a component with label G. From each pixel of the uppermost row of its bounding box, we perform a vertical scan and as soon as the pixel labeled G is encountered, we convert its label to U. The set of pixels labeled U denotes the upper envelope of the component. See Fig. 6c where the upper envelopes of the components of Fig. 6b are shown. The upper envelope image contains head line information, as well as some non-linear parts due to some modified and basic characters. To delete the non-linear parts, we find and select a super-set of digital straight line (SDSL) segments from the upper envelopes using the following three properties:[21] (1) In a DSL there exist runs of pixels in at most two directions which differ by 45°, (2) for runs of two directions, the run lengths in one of the directions is always one, (3) the run length in the other direction can have at most two values differing by unity (e.g. n and $n + 1$).

For each component only the longest SDSL is chosen. Figure 6d shows the SDSL of the components. Let the set of such SDSLs in the document be R_1. We can take Hough transform on R_1 to find an estimate of the skew angle θ_s. However, a quicker approach is as follows. Find the longest element L of R_1. From L find the normal distance to a reference (say, leftmost) pixel of the members of R_1. The SDSLs of a particular text line will have nearly equal normal distances, and hence they can be clustered in terms of their normal distances. Find the leftmost pixel A and rightmost pixel B in each cluster and find the angle of $\overline{AB}$ with respect to the horizontal direction. The average of these angles over all clusters gives an estimate of skew angle θ_s. We call it as QUICK-SKEW algorithm.

The image is then rotated by θ_s in the opposite direction so that the script lines become horizontal. This is done by simple rotation transforms and truncation to the nearest integer value to get the new coordinate.

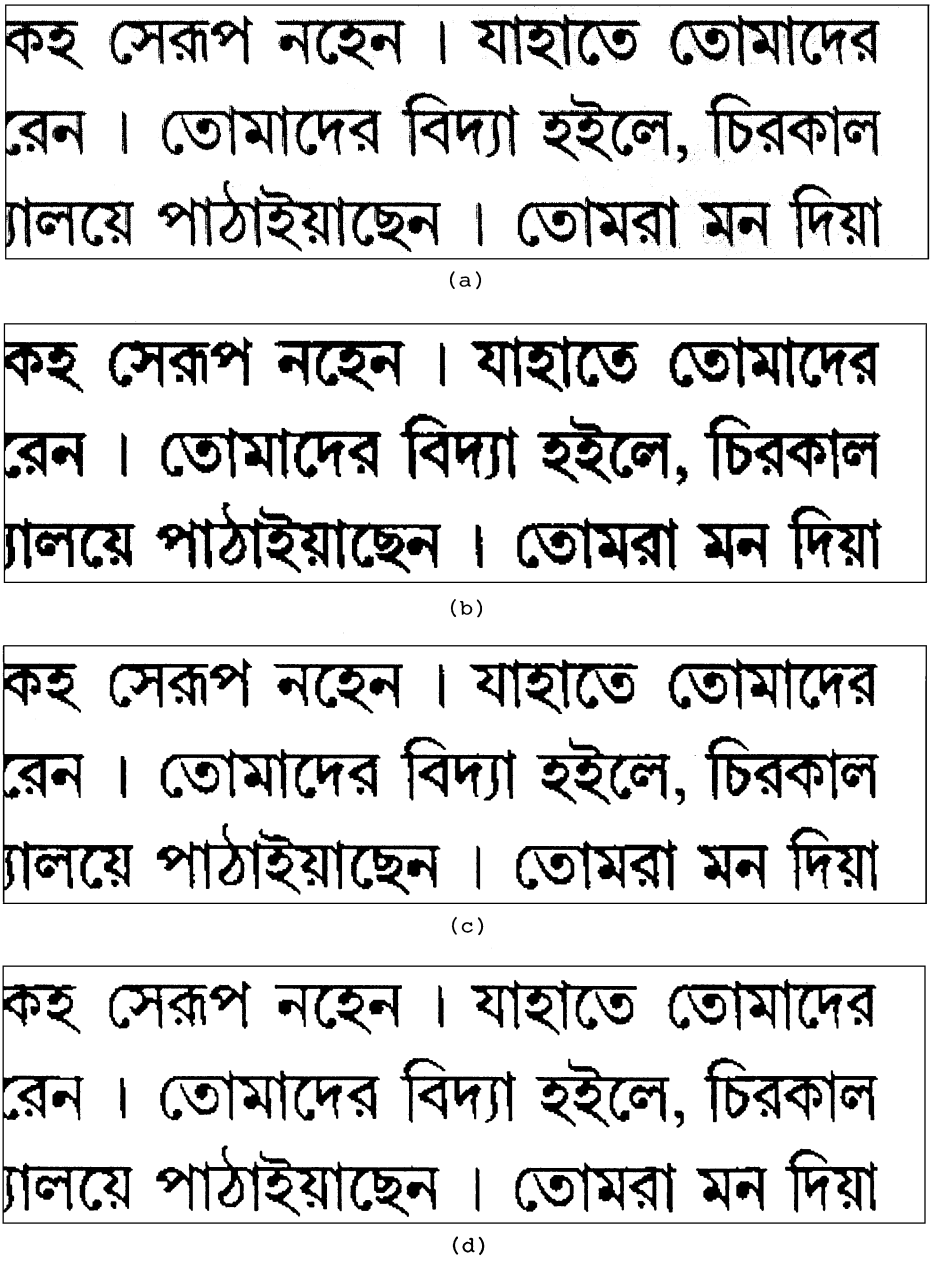


Fig. 5. Comparison of different thresholding methods: (a) original image, (b) method due to Kapur *et al.*, (c) method due to Otsu and (d) our method.

Font and size variation do not affect the proposed skew detection and correction method since the presence of head line is independent of character shape and size. Also, the approach is not limited to any range of skew angle. The algorithm was tested on several skew angles of each of the documents. The mean and standard deviation of the estimated skew angles obtained using Hough transform as well as using QUICK-SKEW are presented in Table 3. Here the statistics has been taken over 30 images for each skew angle.

3.3. Script block detection

If a document contains multiple columns of text lines as well as graphics and halftone pictures, it is necessary to separate the text blocks before feeding it to the OCR system. There exist various techniques of text block segmentation. One approach is based on the use of Gabor filterbanks where the response of the filterbank in text regions is different from that in graphics regions.[22] Another approach is based on the "docstrum" analysis of the document.[19] Fletcher

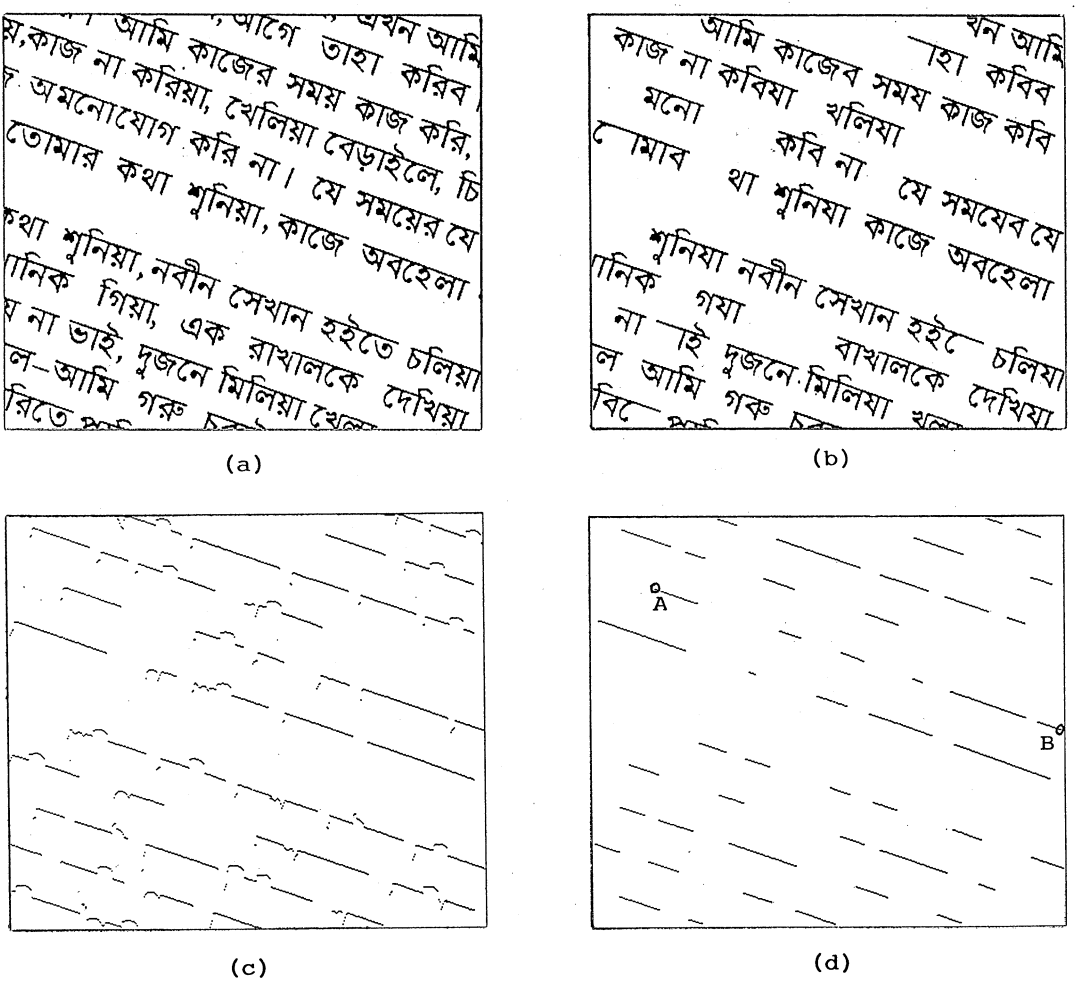


Fig. 6. Skew correction approach: (a) an example of skewed image, (b) selected components of Fig. 6a, (c) upper envelopes of components of Fig. 6b and (d) SDSL components of Fig. 6c.

Table 3. Mean and standard deviation (SD) of estimated skew angles. (For each true skew angle 30 images have been tested.)

True skew angle (in deg)	Estimated skew angle			
	Hough method		QUICK-SKEW method	
	Mean	SD	Mean	SD
30	29.13	0.264	29.86	0.415
20	19.89	0.432	20.12	0.596
10	10.13	0.368	10.21	0.386
5	5.04	0.365	5.02	0.516
3	3.07	0.163	2.96	0.396

and Kasturi[23] applied Hough transform for text separation from mixed text/graphics images. Pavlidis and Zhou[16] used a correlation-based technique to segment the text regions as well as to detect the logical text blocks. We have adopted this method with some modifications and found that reasonably good results are obtained on various documents.

4. LINE, WORD AND CHARACTER SEGMENTATION

For convenience of recognition, the OCR system should automatically detect individual text lines as well as segment the words and then the characters accurately. Since *Bangla* text lines can be partitioned into three zones (see Fig. 3), it is convenient to distinguish these zones. Character recognition becomes easier if the zones are distinguished because the lower zone contains only modifiers and *halant* marker, while the upper zone contains modifiers and portions of some basic characters.

4.1. Text line detection and zone separation

The lines of a text block are segmented by finding the valleys of the projection profile computed by a rowwise sum of gray values. The position between two consecutive head lines where the projection profile height is least denotes one boundary line. Note that the head line is already detected during skew correction. A text line can be found between two

Fig. 7. Text line segmentation approch. (Dotted lines are the separators of text lines.)

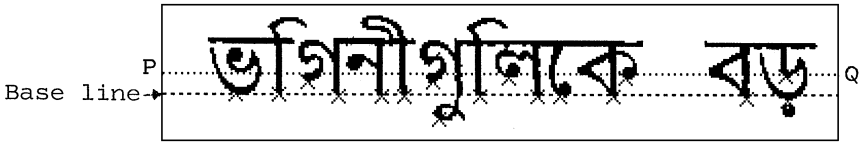


Fig. 8. Lower boundary detection approach.

consecutive boundary lines. For example, see Fig. 7. Here it is assumed that the text block contains a single column of text lines only.

From Fig. 3 it is clear that the upper zone can be separated from the middle zone of a text line by the head line, which is already detected. To separate the middle and lower zone consider an imaginary straight line $\overline{PQ}$ that horizontally partitions the text line region into equal halves. Consider only the connected components below $\overline{PQ}$ and connected to $\overline{PQ}$. For each of these components, the lowermost pixel is labeled by $\times$. The horizontal line which passes through maximum number of $\times$ labeled pixels is the separator line (base line) between the middle and lower zones. See Fig. 8 for an illustration.

Note that separation of middle and lower zones is relatively less accurate. Moreover, the tails of some characters fall in the lower zone. Care is needed to distinguish them from the vowel modifiers, *halant* sign, etc. staying in this zone. This problem is addressed in Section 7. We include the head line, whose thickness is 6% of the height of the text line, within the middle zone. The relative heights of upper, middle and lower zone in *Bangla* script are nearly 27%, 52% and 21%, respectively, for the popular fonts like *Monotype* or *Linotype*.

The upper zone may contain parts of basic and compound characters, parts of vowel modifiers, nasalization sign, apostrophe sign and consonant modifier like *ref*(✓). The lower zone may contain *vowel modifiers*, *consonant modifiers*, *halant sign*, etc.

4.2. *Word and character segmentation*

After a text line is segmented, it is scanned vertically. If in one vertical scan two or less black pixels are encountered then the scan is denoted by 0, else the scan is denoted by the number of black pixels. In this way a vertical projection profile is constructed as

shown in Fig. 9. Now, if in the profile there exist a run of at least k_1 consecutive 0's then the midpoint of that run is considered as the boundary of a word. The value of k_1 is taken as half of the text line height. The algorithm is robust and accurate since characters of a word are mostly connected through the head line.

To segment individual characters of a word we consider only the middle zone. The basic approach here is to rub off the head line so that the characters get topologically disconnected, as shown in Fig. 10b. To find the demarcation line of the characters a linear scanning in the vertical direction from the head line is initiated. If during a scan, one can reach the base line without touching any black pixel then this scan marks a boundary between two characters. For some *kerned* characters[24] a piecewise linear scanning method has been invoked (see Fig. 10b). Using this approach nearly 98.6% characters have been properly segmented. The error was due to touching characters arising out of gray-tone to two-tone conversion. For the segmentation of these touching characters we have used the method due to Kahan *et al.*[7]

Occasionally, characters like G(গ) and N*(ণ) can be split into two sub-segments by the deletion of head line. See Fig. 11b where the character G(গ) has two sub-segments, the left one being enclosed by a box. To join the sub-segments we note that any character in the middle zone touches the base line. Since here the left sub-segment does not touch the base line, the algorithm combines it with the right sub-segment.

In actual practice the head line is not rubbed off to segment the characters. The lowermost row of the head line is noted. A linear or piecewise linear scan starting from this noted row downwards in the middle zone segments the characters, as in Fig. 10b.

Once the segmentation is done, simple features such as bounding box, zonal information, etc. are computed on each of them. The bounding box is computed for the character portion in the middle zone only. In the upper or lower zone about the bounding box, a character may have a portion of its own or portions of vowel modifiers (and occasionally consonant modifiers). They are used for both initial grouping and final recognition of the characters.

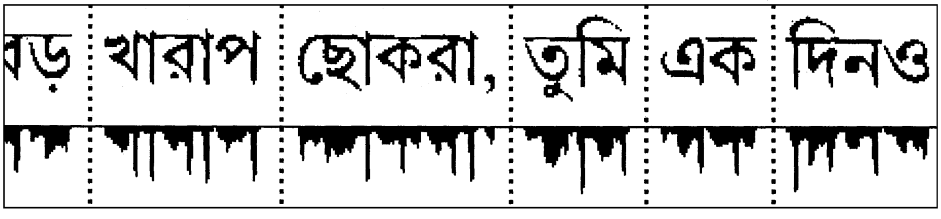
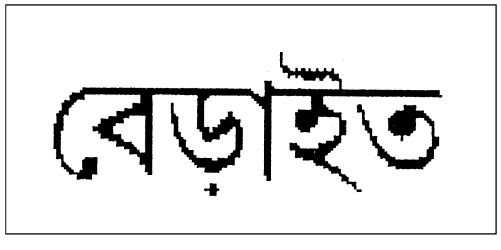
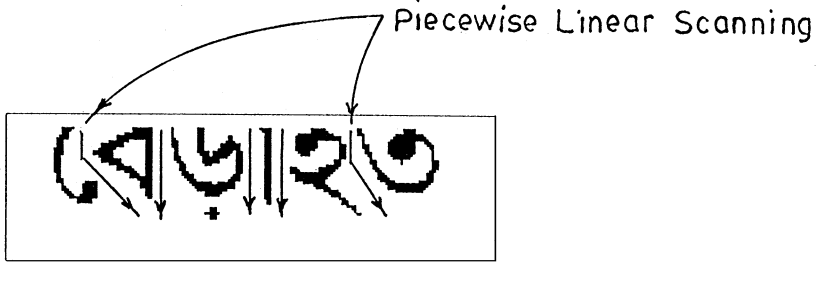


Fig. 9. Word segmentation approach. (Dotted lines are the separators of words.)



(a)



(b)

Fig. 10. Character segmentation approach: (a) a *Bangla* word, (b) scanning for character segmentation after deletion of head line from middle zone.

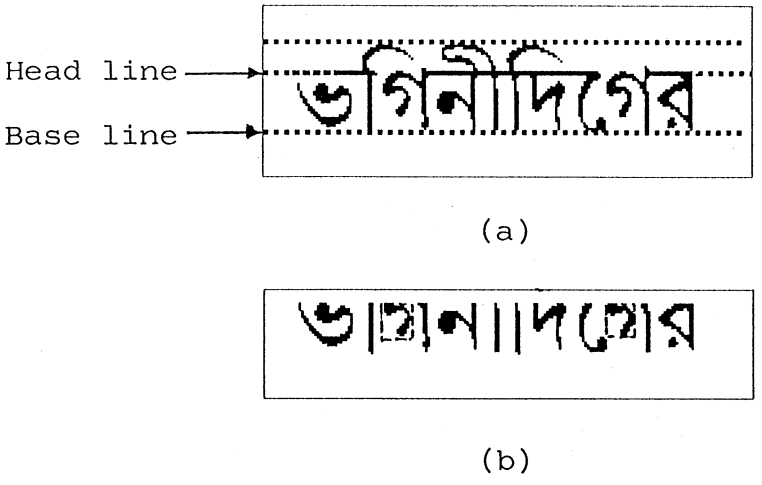


Fig. 11. Detection of two parts of a wrongly segmented character: (a) a *Bangla* word and (b) wrongly segmented parts of the word are shown in a square box (see text).

5. PRIMARY GROUPING OF CHARACTERS

Before going for actual classification, the basic, modified and compound characters are distinguished. One motivation of this step is that although there are a large number of compound characters, they occupy only 4–6% of the text corpus. If we can isolate them, then the small sub-set of basic characters can be recognized by a fast, accurate and robust technique while we could use a slower technique to recognize the compound characters. We have used a feature-based approach for basic and modifier character recognition, and a combination of feature-based and template-matching approach for the compound character recognition. The primary grouping of characters are done with the following observations.

The middle zone of a script line may contain the vowel modifiers that have very small width. The width of the basic characters vary from character to character, being the least for character D(ি) and the most for character N + (ে) in almost all fonts. On the other hand, some compound characters like /DG/(দ্গ), /R,G/(র্গ) have very large width.

Another important point to note is that the compound characters are, in general, more complex-shaped than the modifier and basic characters. In general, a compound character will have large border length and high curvature along the border than a basic character. Thus, using the following three features a character may be identified as belonging to one of the three groups representing modifier, basic and compound characters.

- Feature f_1. Bounding box width.
- Feature f_2. Number of border pixels per unit width, which is computed by dividing the total number of character boarder pixels by the width of the character.
- Feature f_3. Accumulated curvature per unit width. To compute f_3 the boarder of the character is traversed. The absolute angular change made by traversing from one pixel to the next is added over all pixels of the boarder. This accumulated value is divided by f_1 to get f_3.

Two discriminant planes of the form

$$a_1 f_1 + a_2 f_2 + a_3 f_3 = a_0 \quad \text{and}$$
$$b_1 f_1 + b_2 f_2 + b_3 f_3 = b_0$$

are used for the classification. The parameters of discriminant planes are found from a large number of training data. For 12 point text digitized at 300 dpi the typical values of $a_0, a_1, a_2, a_3, b_0, b_1, b_2, b_3$ are 225.77, 0.30, 0.21, 1.99, 80.83, 0.22, 0.49, 0.67, respectively. On the test data set, an overall classification accuracy of 97.2% was obtained by this classifier. More about the misclassification behaviour is described in Section 7.

6. FEATURE SELECTION AND DETECTION

We considered a few stroke features for initial classification of basic characters. The stroke features are used to design a tree classifier where the decision at each node of the tree is taken on the basis of presence/absence of a particular feature.

In Fig. 12 the principal set of chosen stroke features are shown in the context of the characters. Apart from the *principal features*, some other features are also used at some nodes of the tree classifier. They are described during tree classifier design. The features are chosen with the following considerations; (a) robustness, accuracy and simplicity of detection, (b) speed of computation, (c) independence of fonts and (d) tree classifier design need.

It may be noted that the stroke features of Fig. 12 are simple (mostly linear) in structure and hence quick and easy to detect. Their distortion due to noise can be easily taken care of. They are quite stable with

respect to font variation. At a non-terminal node of a tree classifier, we use one feature which is able to sub-divide the characters in that node into two groups so that the sum of occurrence probabilities of one group is as near as possible to that of the other group. The stroke features of Fig. 12 serve such a purpose reasonable well.

Note that these features should be searched in the text line middle zone only. For the upper and lower zones, we use shape characteristic of the modifiers, described in Section 7.

The methods of detecting the stroke features of Fig. 12 are described below. Here the stroke lengths are standardized with respect to the character middle zone height because this height is constant for characters of single font and size.

The stroke 1, shown in Fig. 12 is essentially the same as the head line of a character. To detect it, we assumed that the length of the stroke l_a is at least 75% of the width of the particular character bounding box. Now, the upper quarter part of the character middle zone is scanned horizontally. If a scan contains continuous black pixels whose number is more than or equal to l_a then the stroke is assumed to be present in that character.

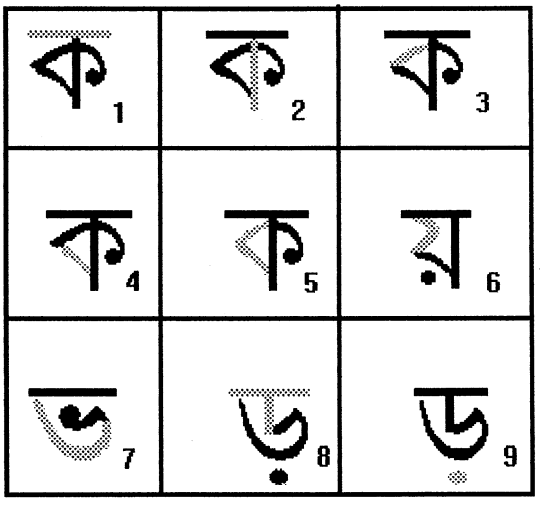


Fig. 12. Stroke features used for character recognition. (Shaded portions in the character represent the features. For easy identification, the features are numbered.)

The method of detection of stroke 2 is similar to that of stroke 1, but here the length of the stroke l_b is 75% of the height of the character middle zone and the scanning mode is vertical.

To detect stroke 3, we assumed that the length of this stroke l_c is 40% of the height of the middle zone of the character. A diagonal scan is made (so that the scanning direction makes $+45^\circ$ with the horizontal) in the upper half of the middle zone of the character. If a scan contains continuous black pixels which is greater than or equal to l_c then stroke 3 is present in the character. Stroke 4 can be detected in a similar way. However, here the scanning is made in the lower half part of the middle zone of the character along 45° direction.

We do not need any extra computation, to detect stroke 5. If the strokes 3 and 4 are both present in the character, we assume that stroke 5 is present in that character.

To detect stroke 6, the length of the arms of the stroke is assumed to be 30% of the width of the middle zone of the character and the angle between them is 315°. A two-way scan is performed where the starting point of the scan is at the left-hand side of the middle of the vertical line of the character. If, in a scan, each arm contains continuous black pixels of length greater than or equal to the length of the stroke then stroke 6 is deemed present in that character.

Stroke 7 is a cup-shaped feature where the bottom of the cup touches the base line. To detect it horizontal scans are performed while leftmost and rightmost black pixels are noted. For each horizontal scan, the x co-ordinates of these two extreme points are of nearly equal distance from P, the midpoint of black run where the cup touches the base line. This distance increases monotonically as we go scanning upwards till the middle of the middle zone. See Fig. 13 where $\overline{PX_1} \simeq \overline{PX_2}$, and both $\overline{PX_1}$ and $\overline{PX_2}$ increase monotonically with scans.

To detect stroke 8 at first we check whether stroke 1 is present. If this stroke is present then a stroke like

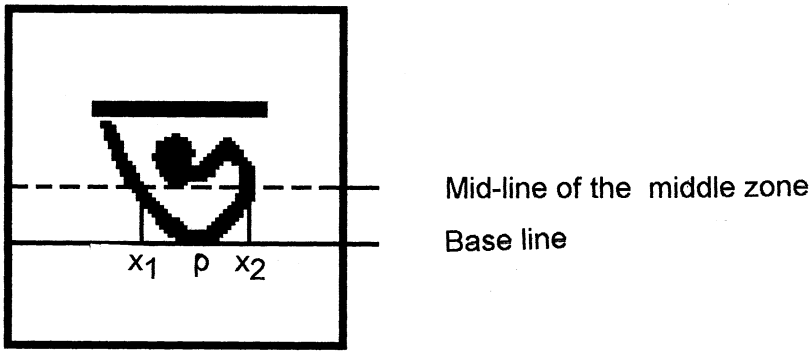


Fig. 13. Detection approach of stroke feature number 7 of Fig. 12.

Table 4. Success rates on the detection of different strokes

Stroke number	Recognition rate
1	99.98%
2	99.70%
3	99.65%
4	99.30%
5	99.45%
6	98.32%
7	98.21%
8	99.55%
9	96.67%

stroke 2, whose length is 40% of the height of the middle zone, is tested at the middle portion of stroke 1. If such a stroke is also present then it is assumed that stroke 8 is present in that character.

For the detection of stroke 9 it is tested in the lower part of the middle zone, whether more than 95% pixels within a circle having diameter equal to 12% of the middle zone height (that means the thickness of head line) are black. If yes, and if it is not a part of another stroke then such a stroke is deemed present in that character. To detect the circle quickly, at first we scan horizontally. If a run of R black pixels is present then from its midpoint O we scan vertically. If a run of R black pixels is present with O as midpoint then we scan diagonally from O. If again successes are obtained for both diagonal directions then we assume that a circle of diameter R is present.

We have noted that in most of the cases the strokes 1, 2, 3, 4, 5 and 8 are correctly detected. The success rate on the detection of different strokes are shown in Table 4.

7. RECOGNITION OF CHARACTERS

7.1. Basic character and modifier recognition

Our recognition scheme of modifiers and basic characters is based on shape identification only. A feature based tree classifier separates the characters. For the compound characters, feature-based tree classifier is initially used to separate them into small groups. Next, an efficient template matching approach is employed to recognize individual characters of each group.

The misclassification in primary grouping of characters (described in Section 5) arise because sometimes basic characters like A(অ), J(জ) and N + (ঞ) fall into compound character category and compound characters like /NN/(ন্ন), /T,B/(ট্ট) and /TT/(ত্ত), etc. fall under basic character category. Similarly, basic characters C(চ) and M,(২) occasionally fall into a modifier group while the basic character H,(ঃ) almost always falls into the modifier group. On the other hand, one modified shape of E(এ) sometimes falls into the basic character group. For correct recognition, we have considered these characters in both the groups they may be classified.

7.1.1. Design of the tree classifier. The design of a tree classifier has three components: (1) a tree skeleton or hierarchical ordering of the class labels, (2) the choice of features at each non-terminal node, and (3) the decision rule at each non-terminal node.

Ours is a binary tree where the number of descendants from a non-terminal node is two. Only one feature is tested at each non-terminal node for traversing the tree. The decision rules are mostly binary e.g. presence/absence of the feature. To choose the features at a non-terminal node we have considered the *Bangla* character occurrence statistics given in Table 2.

If the pattern group of any non-terminal node can be sub-divided into two sub-groups by examining a feature so that the sum of occurrence probabilities of one group is as near as possible to that of the other group then the resulting binary tree is optimum in time complexity, assuming that the time required to test a feature is constant. The resulting tree is identical with the Huffman code[25] tree generated using the character occurrence probabilities.

However, we may not get a set of features to design such an optimal tree. A semi-optimal tree is generated if out of the available features, we select the one for a given non-terminal node that separates the group of patterns best in the above sense. In this way, we used features 1 and 2 near the root nodes. For example, the 'vertical line' i.e. feature 2 in the right side of the character boundary, separates the characters into two groups with sum of probabilities 0.59 and 0.41. Features 1 and 2 partition the basic characters into four nearly equi-probable sub-groups. Fortunately, they are the simplest features to detect too.

Note that our features are positional strokes. Detection of some of them (e.g. feature 4 of Fig. 12) should follow detection of others (e.g. feature 2). In other words, if the feature 2 exists in a character then we will check only for feature 4 in that character. Moreover, as we go down the tree, the number of features to choose from gets reduced. These problems put further constraint in designing the optimal tree classifier.

For an operational tree classifier, the average number of nodes visited is $\sum_{i=1}^{n} n_i p_{c_i}$, where n_i is the number of nodes needed to visit for the recognition of character c_i having probability of occurrence p_{c_i}. This number is the smallest if Huffman tree could be used for classification. As a comparative study, the average number of nodes to be visited in the Huffman (i.e. optimum) tree according to the probability estimates of the basic characters is 4.714 while that obtained in our tree is 5.331. Given the constraints of selecting and using the features described above, our tree is quite close to the optimum one.

The tree classifier for basic characters is partially shown in Fig. 14. Most basic characters can be recognized by the principal features alone. However, as stated earlier, in addition to the principal features some other simple features are also used at some

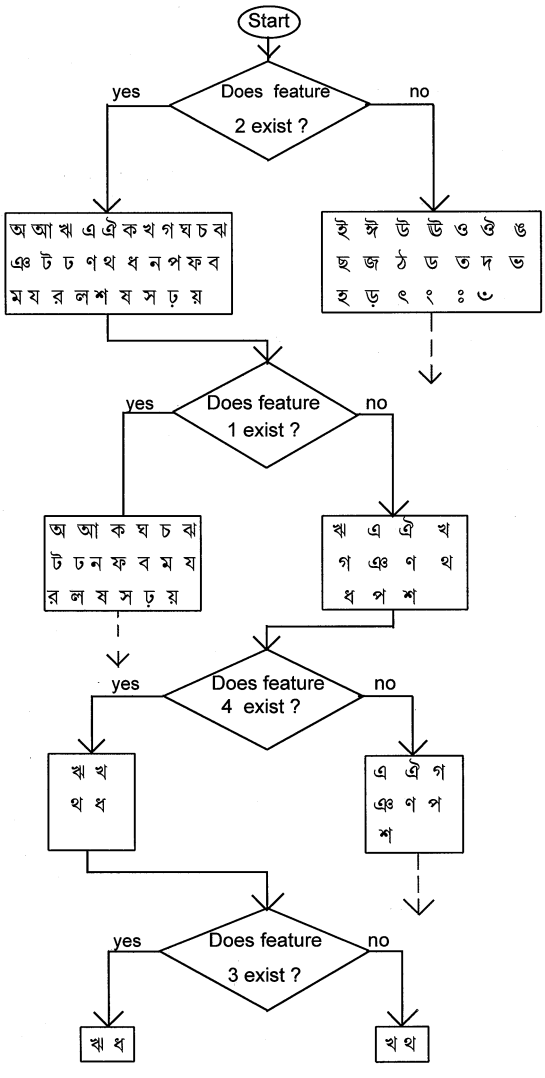


Fig. 14. A flow chart representation of a portion of tree classifier for basic character recognition.

nodes. Consider for example, K(ক) and PH(ফ) which belongs to the same non-terminal node of the tree. To separate them, we test if the vertical line touches the head line or not (in case of PH(ফ) the vertical line does not touch the head line). Similarly, to separate L(ল) and N(ন) which belong to the same node we draw an imaginary line from the central position of the disc (feature 9) upwards till the head line is reached. For N(ন) this line will cross the character boundary pixels thrice while for L(ল) the number of crossings will be four. This *crossing count* is used as a heuristic at some other nodes also.

Detection of two full vertical lines as in JH(ঝ) is used to distinguish some characters. To distinguish P(প) and G(গ) as well as KH(খ) and TH(থ) some other heuristics are used in the tree.

To detect basic characters as well as some modifiers, the upper zone is inspected. Any signature in the upper zone which touches a vertical line in the middle zone denotes the modifier for I(ই), I,(ঈ) or (AU) (ঔ). Other signatures in the upper zone are characteristics of the basic(or compound) characters of the corresponding middle zone. Note that the modifiers for E(এ) and O(ও) are detected by inspecting the middle zone.

To detect other modifiers, the lower zone is inspected. If there is any signature in the lower zone that is not a part of the basic or compound characters in the middle zone then it is assumed to be a modifier following the character. Three such modifiers and halant sign may be encountered in this region. They are separated by some structural primitives.

To check if a signature in lower zone is actually the tail of basic characters, a curvature smoothing test is made. If the signature curvature is continuous and smooth, then it is accepted as the tail of a basic character.

7.2. *Compound character recognition*

The compound character recognition is done in two stages. In the first stage the characters are grouped into small sub-sets by a feature-based tree classifier. At the second stage characters in each group are recognized by a sophisticated run-based template matching approach. We adopted this hybrid approach instead of only tree classifier approach because it is nearly impossible to find a set of stroke features which are simple to compute, robust and reliable to detect and are sufficient to classify a large number of complex-shaped compound characters.

The features used in the tree classifier are head line, vertical line, left slant (i.e. features 1, 2, 3 of the Fig. 12), boundary box width, presence of signature in upper zone etc. A terminal node of this tree corresponds to a sub-set of about 20 characters. These character templates are ranked in terms of their bounding box width and stored during the training phase of the classifier at an address corresponding to this terminal node.

A candidate compound character which has reached a terminal node of the tree is then matched against the stored templates corresponding to this node. Matching is started with the template whose bounding box width is nearest to that of the candidate and continued till the template width is comparable to that of the candidate. Matching decision is speeded up by this process.

To superimpose the candidate on the template for computing the matching score, the reference position is chosen by characteristic features of the candidate. For example, if a vertical line (feature 2 of Fig. 12) is present in the candidate (which is known from the first stage of the classifier) the templates also have vertical lines and the matching score is computed at first by aligning the vertical line of the candidate and template. Scores for small shift about this vertical line are also computed to find the best match. This process

also speeds up the matching process and increase the accuracy to a large extent.

Further speed-up is done by using the concept of black and white runs as illustrated in Fig. 15. Let us consider a black run of the candidate character to be matched with a row of the template character. At position p, the black to black matching score is 4. At position $p+1$, the matching score is also 4. This result can be inferred by noting that pixels A and B in the template have the same colour. In general, we can compute the change in matching score for any run by noting the color of only two pixels in the template. Thus, the order of computation is of the order of number of runs, rather than the number of pixels.

A wide variety of correlation measures are available for template matching applications[26] of which the measure $J_C = n_{11}/(n_{11} + n_{10} + n_{01})$ due to Jaccard and Needham, is found satisfactory for our purpose. Here n_{ij} = number of pixels with input image pixel value i and template pixel value j in the corresponding positions; $i, j = 0, 1$. The J_C measure gives the ratio of total matched object pixels to the sum of the object pixels in template and candidate.

The best matching score for the candidate should exceed a threshold for acceptance as a recognized character. Else, the candidate is rejected as unrecognized.

We noted that a reasonable amount of character size variation can be accommodated by rescaling. If templates for 12 point characters size are stored then it was found that characters of size ranging from 8 to 16 points can be matched by rescaling the candidate without appreciable error. The candidate middle-zone boundary box height is used for rescaling the candidate.

As mentioned before, some basic characters are found to be classified as compound characters during initial grouping. Their templates are also stored and they are matched with the candidate in the same way as above.

The recognition performance over a set of documents (without error correction module) is presented in the next section. The recognition speed, for 12 points text digitized at 300 DPI, is 85–90 characters per minute by SUN 3/60 machine (with microprocessor MC68020 and SUN O.S. version 3.0).

The error correction module described below improves the performance to a reasonable extent.

8. ERROR DETECTION AND CORRECTION

Of the two classes of approaches of error detection and correction[27, 28] based on n-gram and dictionary search, we use the latter approach. Our input is a string of OCR recognized characters between two word boundary markers, called as *test string*. According to the structural shape of *Bangla* characters as well as the OCR technique used, the error can be classified into five categories: substitution, rejection, deletion, run on characters error and split character error. Transposition and insertion error are unlikely for *Bangla* text.

In the error calculation, it is assumed that the original document contains no error i.e. all words are valid surface words. Based on an experiment with 10,000 words it was noted that our OCR system makes an average of 3.17% character recognition error which is equivalent to 15.54% word recognition error. Out of these 15.12% are non-word and 0.42% are real word errors (which cannot be corrected). The relative contribution of different error types is given in Table 5. We have noticed that 29% of single character error words are due to characters of compound class and the rest for modifier and basic characters. Also, most of the rejected characters are compound characters.

8.1. Error detection approach

The OCR system output can be a *partially recognized string* which is inherently erroneous or a *fully recognized string* which should be subject to error detection. However, both types of string should be subject to error correction.

As mentioned before, for an inflectional language it is useful to maintain a root word dictionary and a suffix dictionary. Then for error detection, it is necessary to check if the test string is a valid root

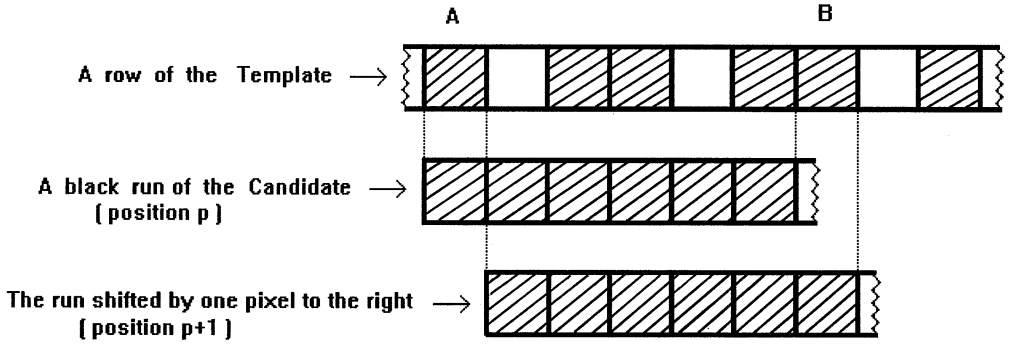


Fig. 15. Run-based template matching method.

word followed by a suffix that grammatically agrees with the root word. The notion of grammatical agreement is explained later on in this section.

Let W be a string of n characters $x_1x_2 \dots x_n$. If W_1 is a string $x_1x_2 \dots x_n$ and if W_2 is another string $r_1r_2 \dots r_p$ then we define the concatenation of the two strings i.e. $W_1 \oplus W_2 = x_1x_2 \dots x_nr_1r_2 \dots r_p$. Let $|W|$ denote the number of characters in string W. Also, if a sub-string W_s of W contains x_1 (i.e. first character of the string) then W_s is called *front sub-string*. Similarly, W_s is a *back sub-string* if W_s contains x_n (i,e last character of the string). Also, we can 'dissociate' a sub-string W_s of W from W given by $W \ominus W_s$. If $W \ominus W_s = W_r$ then $W = W_s \oplus W_r$ if W_s is a front sub-string, while $W = W_r \oplus W_s$ if W_s is a back sub-string.

If W is a valid surface word then either (i) W is a root word or (ii) A sub-string $R_1 = x_1x_2 \dots x_r$ of W is a root word and another sub-string $S_1 = x_{r+1}x_{r+2} \dots x_n$ of W is a suffix which grammatically agrees with R_1.

The segmentation of W into root word and suffix (including null) satisfying these conditions may be called *valid parsing*. Normally, a valid surface word can be parsed into *one* root word and *one* suffix (including null) although multiple parses are sometimes possible.

To parse a string W, we can start from left and find the sub-strings $R_1, R_2, \dots, R_k$ which are valid root words by looking at root word lexicon. Similarly, we can start from right of W and find the sub-strings $S_1, S_2, \dots, S_m$ which are valid suffixes. However, a root suffix pair $\{R_i, S_j\}$ is a candidate for parsing only if $R_i \cap S_j = \phi$ and $R_i \oplus S_j = W$. The situation for a typical string is shown in Fig. 16. Here, the pairs $\{R_1, S_m\}$, $\{R_3, S_3\}$ and $\{R_k, S_2\}$ only are candidates for parsing and if W is a valid surface word then at least one of these candidate pairs has a valid parse. The distribution of candidate root–suffix pairs computed over 60,000 surface words is shown in Table 6. Note that most of the suffixed surface words have only one candidate root–suffix pair.

A root word cannot concatenate with arbitrary suffix to form a valid surface word. The valid concatenation is primarily governed by the *parts of speech* of the root word. For example, if the root word R_i is a "verb", then it can concatenate with suffix S_j if it is an appropriate "verb suffix". In other words, R_i and S_j grammatically agree with each other.

Thus, our algorithm for error detection is as follows:

Step 1: Consider a new string W. Start from left and find all root word sub-strings $R_1, R_2, \dots, R_k$ in W.

Step 2: If $R_k = W$, go to step 5. Else, start from right and find a valid suffix sub-string S.

Step 3: If $W \ominus S = R_i$, for any i, then call the grammatical agreement test. If the test is passed then go to step 5.

Step 4: Find the next valid suffix sub-string. If no such sub-string exists then go to step 6. Else, go to step 3.

Step 5: Report success (the string is a valid surface word). Go to step 1.

Table 5. Distribution of word recognition error

Word error due to single misrecognized character	9.31%
Word error due to single rejected character	3.05%
Word error of other types (Double character error, Run on error, Splitting error, Deletion error, etc.)	3.18%
Total word recognition error	15.54%

Table 6. Prababilities of valid root-suffix pair substrings

Word length	Number of valid root-suffix pair		
	1	2	3
2	1.0000	0.0000	0.0000
3	0.9820	0.0180	0.0000
4	0.8892	0.1108	0.0000
5	0.8703	0.1250	0.0047
6	0.9199	0.0736	0.0065
7	0.9335	0.0666	0.0000
8	0.9800	0.0200	0.0000
9	0.9825	0.0175	0.0000
10	1.0000	0.0000	0.0000

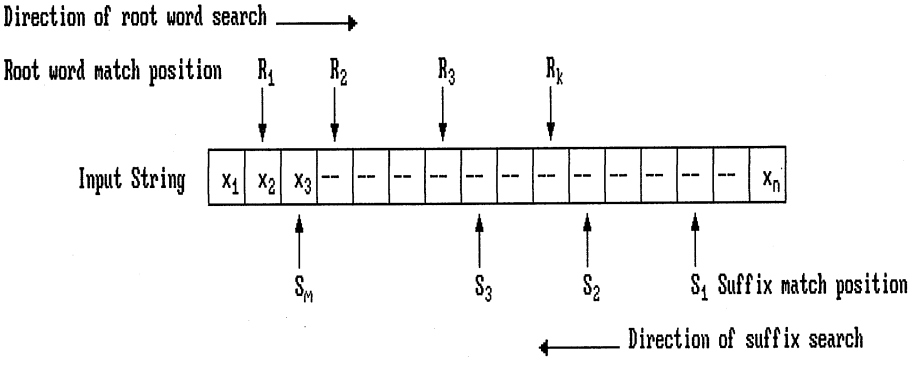


Fig. 16. An example of character string parsing.

Step 6: Report failure. Call error correction algorithm and go to step 1.

8.2. *Lexicon structure*

The root-word lexicon contains all root words as well as the morphologically deformed variants of the root words (mostly verbs) so that simple concatenation with suffix make valid surface words. To organize the lexicon, valid root words of length upto 3 characters as well as the first 3 characters of root words of length greater than 3 are represented in a trie structure. At each leaf node of the trie an address is maintained. Words are kept in the main lexicon in alphabetical order along with the parts of speech and other grammatical informations. Address of a leaf node L_k of the trie points is the first word of the main lexicon of which the first three characters are the same as those we encounter by traversing the trie upto that leaf node L_k. So, if a string is of length four or more (characters) then we can obtain the address from trie using the first three characters and then sequentially searching the main lexicon to see if the string is a valid word. The lexical structure is shown in Fig. 17. The sequential search on an average requires five words to be checked for each input string where the lexicon size is 50,000 words.

The suffix lexicon is maintained in a trie structure where a node representing a valid suffix is associated with a suffix marker and parts of speech marker. If one traverses from the root of the trie to a node having valid suffix marker then the encountered character sequence represents a valid suffix string in reverse order (i.e. last character first). This is so because during suffix search we start from the last character of a string and move leftwards.

8.3. *Error correction approach*

For simplicity, our effort is limited to correcting single character (basic or compound) in a word. At first, let us consider a *fully recognized string*.

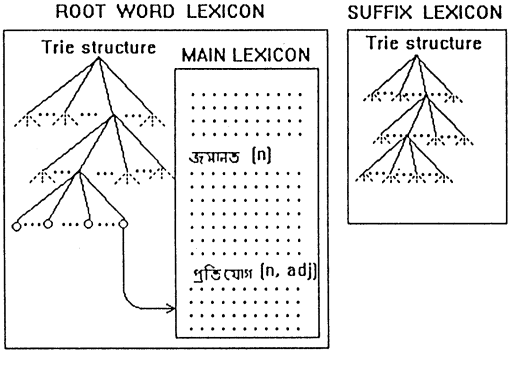


Fig. 17. Error correction lexicon structure. (See examples in main lexicon where parts of speech markers are given in parenthesis.)

The information about all valid root sub-strings $R_1, R_2, \ldots, R_k$ and all valid suffix sub-strings $S_1, S_2, \ldots, S_m$ is recalled from the error detection program module. In general, four cases may arise depending on whether k or m is zero or not. Depending on the situation, error correction is made either on the suffix part or root-word part or on the full string. The root-word or suffix dictionary is opened accordingly.

For *partially recognized string* at first we detect all root words and suffix sub-strings. Here too, depending on the value of k or m four situations may occur. In each case, it is guaranteed that the sub-string to be corrected contains the unrecognized character. Now, the correction is done as follows.

Let C be a character in the alphabet. During the training phase a test is conducted on a large set of data for the performance evaluation of the OCR and let $D(C)$ denote the subset of characters which may be wrongly recognized as C. Members of $D(C)$ may be called confusing characters for C. Now, if a recognized string W contains a character C and if it is a misrecognized character then the correct word for W is a string among the strings obtained when C is replaced by elements of $D(C)$. Such strings can be called *candidate strings*. We can attach a probability to each candidate string depending on the *a priori* knowledge of frequency of a confusing character. Other candidates are generated by taking care of run on, split and deletion error also. The candidates are checked in the respective lexicon and the matched one with the highest probability is accepted as the corrected sub-string. Unmatched string is rejected.

According to the performance of our OCR, the number of alternatives per character rarely exceeds 4. Thus, with an average wordlength of 5 characters, the number of candidate strings per word can be 20 for single position error. For a partially recognized string with a single unrecognized character, note that our OCR knows if it is a basic or compound character. For a basic character the number of candidates is 4 while for a compound character it is about 10 on an average.

We have noted that our error correction module can correct 74.22% of single character error generated by the OCR system. As a result, the overall accuracy of the OCR system is about 95.50% in word level which is equivalent to about 99.10% in character level (assuming a word to consist of an average of 5 characters).

9. DISCUSSIONS

An OCR system to read printed *Bangla* script is proposed for the first time. It is assumed that the document contains Linotype font text, which is valid for most *Bangla* documents. We obtained high recognition score for text printed on clear paper. Performance of the system on documents with a varied degree of noise is being studied.

Table 7. Word recognition rate on different point size images

Point size	8	10	12	14	16
Recognition rate	94.60%	95.65%	96.70%	95.49%	95.11%

Some discussions on the performance of the system on size and style variations of the font are in order. Size and most style variations do not affect the modifier and basic character recognition scheme. Compound character recognition is somewhat affected by the size variation, but the rescaling of templates discussed in Section 7 is effective if the range of size is between 8 and 16 points. The effect of different sizes of single font characters on the recognition system is given in Table 7. For bold style, a set of bold templates yield better results. We have not made elaborate study on italics style since it is rare in *Bangla* documents.

Many popular Indian scripts including *Bangla* and *Devnagari* are derived from *Sanskrit* script and have a lot of common graphemic features including the presence of head line and vertical strokes. Devnagari script is used to write *Hindi*, the third most popular language in the world. Devnagari and *Bangla* have a similar set of compound characters. Moreover, Hindi, like *Bangla* is an inflectional language.

We made a preliminary study of applying our method for the OCR of *Devnagari* script. We noted that skew correction and zone separation as well as line, word and character segmentation modules of our *Bangla* OCR can be directly used for *Devnagari*. The basic and compound character separation can also be done in a somewhat similar way. Also, the decision tree classifier scheme suits well for *Devnagari*, although some of the features are not identical with those used for *Bangla*. Encouraged by these studies, we are building a complete *Devnagari* script recognition system based on our *Bangla* OCR system, which will be reported later on. Our system can act as a model for the OCR development of a few other Indian scripts also.

One of our goals is to attach a speech synthesis module to OCR system so that it can be used for the blind. We have used the diphone concatenation approach where only 750 diphones are sufficient to synthesize any *Bangla* word. Prior probabilities of the diphones are used to make the synthesis efficient in time. Some utterance rules are used for grapheme to phoneme conversion in the OCR recognized word. A word having rejected character(s) is uttered character by character. The speech output is quite intelligible but somewhat flat. We are adding the stress and intonation rules to make the speech more lively.

REFERENCES

1. J. Mantas, An overview of character recognition methodologies, *Pattern Recognition* **19,** 425–430 (1986).
2. P. S. P. Wang, *Character and Handwritten Recognition*, World Scientific, Singapore (1991).
3. R. M. Bozinovic and S. N. Shihari, Off line cursive script word recognition, *IEEE Trans. Pattern Anal. Mach. Intell.* **11,** 68–83 (1989).
4. R. Casey and G. Nagy, Automatic reading machine, *IEEE Trans. Comput.* **17,** 492–503 (1968).
5. K. Y. Wang, R. G. Casey and F. M. Wahl, Document analysis system, *IBM J. Res. Dev.* **26,** 647–656 (1982).
6. S. Mori, C. Y. Suen and K. Yamamoto, Historical review of OCR research and development, *Proc. IEEE* **80,** 1029–1058 (1992).
7. S. Kahan, T. Pavlidis and H. S. Baird, On the recognition of printed character of any font and size, *IEEE Trans. Pattern Anal. and Mach. Intell.* **9,** 274–288 (1987).
8. V. K. Govindan and A. P. Shivaprasad, Character recognition—a survey, *Pattern Recognition* **23,** 671–683 (1990).
9. A. K. Dutta and S. Chaudhuri, Bengali alpha-numeric character recognition using curvature features, *Pattern Recognition* **26,** 1757–1770 (1993).
10. R. M. K. Sinha, Rule based contextual post processing for Devnagari text recognition, *Pattern Recognition* **20,** 475–485 (1985).
11. G. Siromony, R. Chandrasekaran and M. Chandrasekaran, Computer recognition of printed Tamil characters, *Pattern Recognition* **10,** 243–247 (1978).
12. S. Biswas, S. Dasgupta and D. Bhattacharya, Samsad Bangala Abhidhan, *Sahitya Samsad*, Calcutta (1992).
13. B. B. Chaudhuri and U. Pal, Relational studies between phoneme and grapheme statistics in modern *Bangla* language, *J. Acoust. Soc. India* **23,** 67–77 (1995).
14. J. N. Kapur, P. K. Sahoo and A. K. C. Wong, A new method for gray-level picture thresholding using the entropy of the histogram, *Comput. Vision Graphics Image Process.* **29,** 273–285 (1985).
15. N. Otsu, A threshold selection method from gray level histograms, *IEEE Trans. Systems Man Cybernet.* **9,** 62–66 (1979).
16. T. Pavlidis and J. Zhou, Page segmentation and classification, *Comput. Vision Graphics Image Process.* **54,** 484–496 (1992).
17. T. Akiyama and N. Hagita, Automatic entry system for printed documents, *Pattern Recognition* **23,** 1141–1154 (1990).
18. D. S. Le, G. R. Thoma and H. Wechsler, Automatic page orientation and skew angle detection for binary document images, *Pattern Recognition* **27,** 1325–1344 (1994).
19. L. O'Gorman, The document spectrum for page layout analysis, *IEEE Trans. Pattern Anal. Mach. Intell.* **15,** 1162–1173 (1993).
20. H. Yan, Skew correction of document images using interline cross-correlation, *CVGIP: Graphical Models Image Process.* **55,** 538–543 (1993).
21. A. Rosenfeld and A. C. Kak, *Digital Picture Processing*, *vol.* 2, Academic Press, Orlando (1982).
22. A. K. Jain and S. Bhattacharjee, Text segmentation using automatic document processing, *Mach. Vision Appl.* **5,** 169–184 (1992).
23. L. A. Fletcher and R. Kasturi, A robust algorithm for text string separation from mixed text/graphics images, *IEEE Trans. Pattern Anal. Mach. Intell.* **10,** 910–918 (1988).
24. Y. Lu, Machine printed character segmentation—an overview, *Pattern Recognition* **28,** 67–80 (1995).

25. B. B. Chaudhuri and D. Datta Majumder, *Two-Tone Image Processing and Recognition*, Wiley Eastern Limited, New Delhi (1993).
26. J. D. Tubbs, A note on binary template matching, *Pattern Recognition* **11,** 207–222 (1986).
27. K. Kukich, Techniques for automatically correcting words in text, *ACM Comput. Surveys* **24,** 377–439 (1992).
28. H. Takahashi, N. Itoh, T. Amano and A. Yamashita, A spelling correction method and its application to an OCR system, *Pattern Recognition* **23,** 363–377 (1990).

About the Author—B. B. CHAUDHURI received his B.Sc (Hons), B.Tech and M.Tech degrees from Calcutta University, India in 1969, 1972 and 1974, respectively, and Ph.D. degree from Indian Institute of Technology, Kanpur in 1980. He joined Indian Statistical Institute in 1978 where he served as the Project Co-ordinator and Head of National Nodal Center for Knowledge Based Computing. Currently, he is the Head of Computer Vision and Pattern Recognition Unit of the institute. His research interests include Pattern Recognition, Image Processing, Computer Vision, Natural Language Processing and Digital Document Processing including OCR. He has published more than 150 research papers in reputed International Journals and has authored a book titled *Two Tone Image Processing and Recognition* (Wiley Eastern, New Delhi 1993). He was awarded *Sir J. C. Bose Memorial Award* for best engineering science oriented paper published in JIETE in 1986 and *M. N. Saha Memorial Award* (*twice*) for best application oriented research published in 1989 and 1991. In 1992 he won the prestigious *Homi Bhabha Fellowship* for working on OCR of the Indian Languages and computer communication for the blind. He has been selected for the *Hari Om Ashram Prerit Dr. Vikram Sarabhai Research Award* for the year 1995 for his achievements in the fields of Electronics, Informatics and Telematics. Also, he received *C. Achuta Menon Prize* in 1996 for his work on computational linguistics and Natural Language Processing. As a Leverhulme visiting fellow, he worked at Queen's University, U.K. Also, he worked as a visiting faculty member at GSF, Munich and guest scientist at the Technical University of Hannover during 1986–1988 and again in 1990–1991. He is a Senior member of IEEE, member secretary (Indian Section) of the International Academy of Sciences, Fellow of National Academy of Sciences (India), Fellow of the Indian National Academy of Engineering. He is serving as a associate editor of the journals *Pattern Recognition* and *Vivek* as well as guest editor of a special issue of *Journal IETE* on Fuzzy Systems.

About the Author—U. PAL received his M.Sc. degree in Mathematics from Jadavpur University, India in 1990. He joined as a research student at the Indian Statistical Institute in 1991. Currently, he is a faculty in the Computer Vision and Pattern Recognition Unit of the Indian Statistical Institute, Calcutta. His fields of interest include Digital Document Processing and Optical Character Recognition. In 1995, he received student best paper award from Computer Society of India and a merit certificate from Indian Science Congress Association in 1996. He has about 20 papers on Digital Document Processing and Optical Character Recognition. He is a member of IUPRAI, Computer Society of India and Indian Science Congress Association.