

OCR Error Correction of an Inflectional Indian Language Using Morphological Parsing

U. PAL, P. K. KUNDU AND B. B. CHAUDHURI

Computer Vision and Pattern Recognition Unit

Indian Statistical Institute

203, B. T. Road, Calcutta

700035 India

E-mail: {umapada, bbc}@isical.ac.in

This paper deals with an OCR (Optical Character Recognition) error detection and correction technique for a highly inflectional Indian language, Bangla, the second-most popular language in India and fifth-most popular language in the world. The technique is based on morphological parsing where using two separate lexicons of *root words* and *suffixes*, the candidate root-suffix pairs of each input string, are detected, their grammatical agreement is tested and the root/suffix part in which the error has occurred is noted. The correction is made to the corresponding error part of the input string by means of a fast dictionary access technique. To do so, the information about the error patterns generated by the OCR system are examined, and some alternative strings are generated for an erroneous word. Among the alternative strings, those satisfying grammatical agreement in root and suffix are finally chosen as suggested words. In the list of suggested words generated by the system, the desired word is available in 84.22% cases.

Keywords: OCR (Optical Character Recognition), error detection, error correction, Indian language, morphological parsing, suffix, inflectional language

1. INTRODUCTION

The problems of automatic error detection in words and automatic correction are great research challenges. Finding solutions will be very important for text and code editing, computer aided authoring, OCR, machine translation and natural language processing (NLP), database retrieval and information retrieval interfaces, speech recognition, text to speech and speech to text conversion, communication systems for the disabled, e.g. the blind and deaf, computer aided tutoring and language learning, desktop publication, and pen-based computer interfaces. This paper is a report on our OCR [1, 2] error detection and correction approaches for a major Indian language (Bangla) text.

Word errors belong to either of two distinct categories, namely, *non-word errors* and *real word errors*. Invalid words are non-words. By real word error, we mean a valid but not intended word in a sentence, thus making the sentence syntactically or semantically ill-formed or incorrect. In both cases, the problem is to detect the word error and either suggest correct alternatives or automatically replace it with the appropriate valid word. The detection of real word errors needs higher level knowledge compared to the detection of non-word errors. Also, for real word errors, it is often not possible to separate the problem of error detection from that of correction.

Received November 13, 1998; revised October 18, 1999; accepted March 14, 2000.
Communicated by Zen Chen.

Here we are concerned with non-word errors only. We assume that the document subject to OCR system contains only valid words, and that the sentences are syntactically and semantically well formed. Of course, the OCR still can create real word errors, but as will be shown later, such errors are rare.

We are concerned here error correction of Bangla, the second-most popular language in India and fifth-most popular language in the world. The complex character grapheme structure of Bangla script creates difficulty in error detection and correction. There are 11 vowels and 39 consonants in the Bangla alphabet. These are known as *basic characters*. The basic characters are shown in Fig. 1. But the vowels, depending on their position in a word, take different shapes, called *modifiers*. Vowel modifiers and their shapes when attached to a consonant are shown in Fig. 2(a). Moreover, two or more basic characters combine to form a new complex shaped character, called a *compound character*. Examples of some compound characters are shown in Fig.2(b). The total number of characters is about 300, making recognition as well as error detection and correction more difficult than, say, that of *Roman* script.

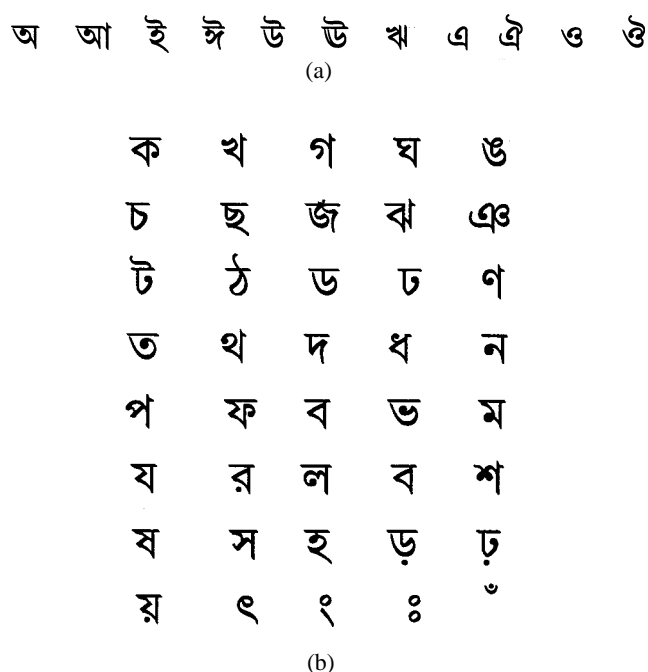


Fig. 1. Basic characters of the Bangla alphabet: (a) vowels (b) consonants.

There are two classes of word error detection techniques, which are based on (1) n-gram verification [3] and (2) dictionary look-up [4]. Although some of the n-gram based techniques are relatively fast, those supported by dictionary look-up are more robust and reliable. Our approach is based on dictionary search. A reason for choosing the dictionary search technique is that an n-gram of 300 characters would have a huge number of entries even for a moderate value of n, (say $n = 3$). Also, since Bangla text is not strictly alphabetical, there exist some difficulty in defining the n-gram.

Vowel	আ	ই	ঈ	উ	ঊ	ঋ	এ	ঐ	ও	ঔ
Modified Shape	।	ি	ী	ু	ূ	ৃ	ে	ৈ	ো	ৌ
When attached to consonant ক	কা	কি	কী	কু	কূ	কৃ	কে	কৈ	কো	কৌ

(a) vowel modifiers

Consonant Characters		Compound Characters
ক + র	=	ক্র
ক + ষ	=	ক্ক্ষ
গ + ষ	=	গ্ক্ষ
চ + চ	=	চ্চ
শ + চ	=	শ্চ
ক + ষ + ণ	=	ক্ক্ষ্ণ
ন + ত + ব	=	ন্তব
ন + ত + র	=	ন্তর

(b) Compound Characters

Fig. 2. Examples of some modifiers and compound characters in Bangla.

However, there is a major problem in using the dictionary look up approach for the Bangla language. *Bangla* is a highly inflectional and agglutinating language. We computed some statistics (see Table 1) based on a Bangla corpus of 2.5 million words and found that nearly 70% of the words in the corpus were formed by inflecting root-words with suffixes. We found that for *Bangla*, the number of inflected word forms can be about 100 times the number of root words. For example, in a dictionary of 70,000 root words, there are about 800 single word verbs. About 125 verb suffixes are in use, of which a particular verb root can combine with nearly 100 suffixes to form valid surface words. Assuming this number to be the average value, the number of surface verb words that can exist in a text is as high as 80,000. The number of non-verb suffixes is also of similar order. Also, in this language, the possibility of word formation by *euphony* and *assimilation* is very high. If we want to make a dictionary that contains all the distinct surface words, then its size can be on the order of millions of words. Even then the dictionary may not be complete, as new words are generated or borrowed from other languages from time to time.

Table 1. Global characteristics of Bangla characters and words.

	Global characteristics	Computed results
1.	Vowel Characters	39.82%
2.	Consonant characters	54.01%
3.	Compound characters	6.17%
4.	Average word length	4.56 (character)
5.	Words with suffix	69.91%

Using a huge lexicon for error detection and correction is not a very practical proposition. For speed and storage efficiency, it is necessary that the lexicon size be reasonably small while tackling the large number of surface words using a clever suffix handling approach. For this purpose, two lexicons are used in our system, one for the root words and the other for the suffixes. Given a character string, first we look for matches in the root word lexicon, and then we search in the suffix lexicon for suffix matches. When properly done, this approach has an additional advantage in that it performs morphological parsing of words, which is the first stage in any natural language processing (NLP) task. We shall very briefly describe in section 5 how morphological parsing is conducted. Moreover, the basic structure of the approach presented here is very suitable for the design of spell-checkers in word processing applications for Bangla and other Indian languages.

Among the existing reports dealing with OCR errors [4-6], we have not come across any that deals with a highly inflectional language like *Bangla*. We believe that this work may be useful for other inflectional languages also.

This paper is organized as follows. A brief review of the earlier works is given in section 2. Section 3 deals with the error pattern generated by our OCR system. The lexicon structure and error detection procedure are described in section 4 and section 5, respectively. The error correction approach is presented in section 6. Finally, experimental results and discussion are given in section 7.

2. BRIEF REVIEW OF EARLIER STUDIES

As mentioned above, there are two major approaches to non-word error detection, namely, dictionary lookup and n-gram matching, in which either a lexicon file or a database of legal n-grams is used to locate one or more errors. In the dictionary lookup method, a test string is used to search the dictionary, and if no match is found, an error is reported. In the n-gram approach, the n-grams of the test string are found. If all the n-grams are there in the system database, then the string is considered to be a valid word. Otherwise, an error is reported.

To generate candidates for non-word error correction, minimum edit distance techniques, similarity key techniques, rule based techniques, n-gram based techniques, statistically based techniques and neural networks based techniques have been developed [5].

In the first technique, the suggested words are those valid words which have the minimum edit distance from the misspelled string. The minimum edit distance is the minimum number of editing operations (i.e., insertions, deletions and substitutions) required to transform one string into another [7, 8]. A modification of dynamic programming [9], the

reverse minimum edit distance [10] was proposed to speed up the error correction effort. In the reverse edit technique, a candidate set is produced by first generating every possible single-error permutation of the misspelled string and then checking the dictionary to see if any make up for valid word.

In a similarity key based technique, the idea is to map similarly spelled strings into identical or similar keys. When a key is computed for a misspelled string, it provides a pointer to all similarly spelled words in the lexicon which may be accepted as candidates [11].

In a rule based technique, a set of rules derived from knowledge of a common spelling error pattern is used to transform misspelled words into valid words [12].

A clear explanation of the traditional use of n-grams in OCR correction has been provided by Riseman and Hanson [13]. They constructed positional n-gram matrices for each subdictionary by partitioning the directory into subdirectories. Presently, hardware based techniques have been proposed that exploited n-gram databases in parallel [3, 14]. Angell et al. [15] also proposed an approach which is based on the number of non-positional binary trigrams common to a misspelled string and a lexicon word.

Probabilistic models have long been used for OCR error correction. Two types of probabilities, namely, transition probability and confusion probability, have been exploited for this purpose. Hull and Srihari [16] provided a general overview of error correction techniques based on transition and confusion probabilities. Bayes' rule was used by Bledsoe and Browning [17] in text error correction, where the valid word maximizing a posteriori probability was considered as the correct word. Kahan et al. [18] combined confusion probabilities with dictionary look up to speed up the process. Also, transition and confusion probability were combined in a *Viterbi algorithm* [19, 20] for OCR error correction.

Other techniques, such as neural networks, have been tried for isolated word error correction. Kukich [21] as well as Cherkassky and Vassilas [22] used backpropagation algorithms in name correction applications. Techniques for reducing the training time by partitioning the lexicon or by exploiting alternative network architectures are being explored [23].

Most of these papers dealt with English and European language texts, and these approaches cannot be applied directly to Indian languages because of their inflectionality, complex character shapes and language patterns.

3. OCR ERROR PATTERN OF BANGLA TEXT

We will briefly describe our OCR system [1] to aid understanding of the error pattern generated by the system.

A document image captured by flatbed scanner is subjected to initial processing, such as skew correction and line, word and character segmentation. For convenience of recognition, the basic, modified and compound characters are classified into separate groups. Basic and modified character recognition is done by a feature based tree classifier. A decision is made at each node of the tree based on certain feature(s) from the feature set. The compound characters are initially grouped into small subsets by a feature based tree classifier. Characters in each group are then recognized by a run length based template matching approach. Characters which are not recognized with the system are marked by the marker '?'.
?



Fig. 3. Stroke features used for character recognition. (Shaded portions in the character represent the features. For easy identification, the features are numbered).

We have considered a few stroke features for initial classification. The features were used to design a tree classifier, where the decision at each node of the tree was taken on the basis of presence/absence of a particular feature. Fig. 3, the principal features chosen for our classifier are shown in the context of the characters. Apart from the *principal features*, some other features, such as the number of crossings, the position of the stroke with respect to the bounding box of the character, the relative position of one feature with respect to another etc. are also used for final classification at some nodes of the tree classifier. The features are chosen based on the following considerations: (a) robustness, accuracy and simplicity of detection; (b) speed of computation; (c) independence of fonts; and (d) tree classifier design needs.

The design of a tree classifier has three components: (1) a tree skeleton or hierarchical ordering of class labels, (2) a choice of features at each non-terminal node, and (3) a decision rule to be used at each non-terminal node. The decision rules are mostly binary in nature, e.g. based on the presence/absence of the feature. We have set the number of descendant nodes to be two, and the number of features at each non-terminal node to be one. For choosing features at a non-terminal node, we have considered the occurrence statistics of the characters in Bangla for the optimum tree design. If the pattern group of any non-terminal node can be subdivided into two subgroups by examining a feature so that the sum of the occurrence probabilities of one group, is as close 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 to the Huffman code [24] tree generated using the character occurrence probabilities. For character recognition by a 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 to be visited the recognition of character c_i and p_{c_i} is the probability of occurrence of the character c_i . This number is the smallest possible if the Huffman tree can be used for classification [24]. For a comparative study, we have computed the average number of nodes to be visited in the Huffman tree and our classification tree. (The Huffman tree was generated based on the probability estimates of the characters.) We found that the average number of nodes to be visited for character recognition in a Huffman tree (i.e. the optimum tree) is 4.714 while that in our classification tree was 5.331. Given the constraints of selecting and using the features described above, our classification tree is quite close to the optimum one.

As mentioned earlier, for compound character recognition, first compound characters are grouped into small subsets using a tree classifier, and then the template matching approach is applied to these sets. For template matching, a candidate character of width W_c

which has reached a terminal node of the tree is then matched against the stored templates corresponding to this node. Matching is done with templates whose bounding box width is within $W_c \pm d$. The value of d is chosen to be a fraction of W_c . Matching decisions are speeded up by this process.

To superimpose the candidate on the template for computation of the matching score, the reference position is chosen based on characteristic features of the candidate. For example, if strokes numbered 1 and 2 in Fig. 3 are both present in the candidate (which is known from the first stage of the classifier), then the group of templates must also have the same strokes, and the matching score is computed by first aligning the candidate and template along these strokes. After the alignment matching score is computed, scores for small shifts about these strokes are also computed to find the best match. In this way, the matching process is speeded up, and accuracy of matching is also increased.

Further speed-up is obtained by using the concept of black and white runs, as illustrated in Fig. 4. 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 the number of runs, rather than the number of pixels.

The best matching score for the candidate should exceed a threshold for acceptance as a recognized character. Otherwise, the candidate is rejected as un-recognized.

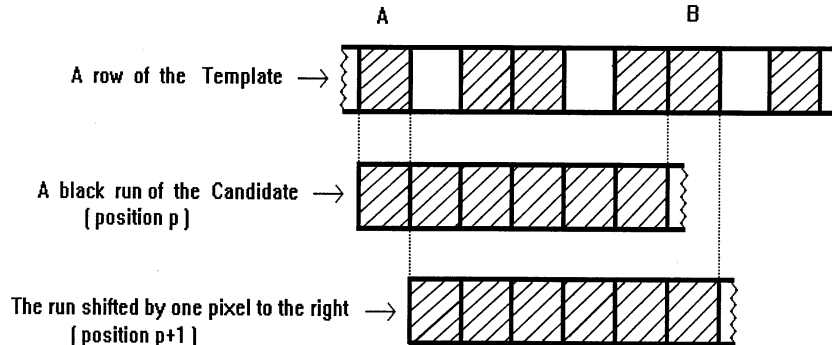


Fig. 4. Run-based template matching approach.

According to the structural shape of *Bangla* characters as well as the OCR technique used, errors can be classified into five categories: substitution, rejection (failure to recognize a character), run on (where two consecutive characters are misrecognized as a single character), split character (where one character is segmented into two parts, one or both parts of which are either misrecognized or unrecognized) and deletion (where two consecutive characters are misrecognized as one of these characters).

As stated above, it is assumed that the original document contains no errors; i.e., all the words are valid surface words. The OCR system exhibits, on average, 3.91% character recognition error, which is equivalent to 19.45% word recognition error. Thus, the system achieves accuracy of 80.55% at the word level with the error correction module. The word

Table 2. OCR error distribution. (The percentage was computed based on the total erroneously recognized words.)

Description	Type	Percentage
Non-word/Real-word Error	Non-word errors	97.3%
	Real-word errors	2.7%
Error Category (among OCR recognized words)	Substitution errors	89.3%
	Deletion errors	6.9%
	Run on errors	2.6%
	Split character errors	1.2%
	Transposition errors	0.0%
Single/Multiple Error	Single OCR recognized character errors	75%
	Single OCR unrecognized character errors	15.8%
	Double character errors	7.3%
	Three or more character errors	1.9%
Corrector Performance	Corrected word = Intended word	84.22%
	Corrected word \neq Intended word	0.38%
	Rejected word	15.4%

recognition error distribution is given in Table 2. The above results were computed from 20,000 OCR recognized words in a single font printed on clear documents. From this table, it can be noted that the single character rejection error of the OCR system is about 15.8%. We have noted that no transposition errors were made by our OCR system for *Bangla*. The reason may be as follows. For recognition, we initially classified the characters into three groups as basic, modifier and compound characters. In our recognition process, a character in one group is rarely recognized as a character in a different group. In Bangla words, a basic character is usually followed by a modified character. Since they belong to different groups, transposition errors are very rare.

We have noted that most of the rejected characters are compound characters. As mentioned above, the OCR system contains a tree classifier; hence, it is possible to memorize the subgroup to which a test character belongs before rejection at the template matching stage so that error correction can be limited to that subgroup of compound characters. We noted that for each character, substitution errors can occur from a set of, at most, 4 other characters. In case of rejection, a rejected compound character can belong to a group of, at most, 15 characters (corresponding to the leaf node of the compound character tree classifier) while a rejected basic character can be one of, at most, 4 basic characters.

Because the errors in our OCR output were mostly single characters per word, our effort was limited to correcting single character in a word. This approach can be extended to correct multiple character errors by increasing the complexity of the system.

4. LEXICON STRUCTURE

As mentioned earlier, there are two lexicons in our system : one for the root words and the other for the suffixes.

The root word lexicon is organized in alphabetical order as follows. It contains all the root words as well as morphologically deformed variants of the root words so that simple concatenation with suffixes produces valid surface words according to our simple word morphology described in section 5. Some of the morphologically deformed variants cannot be used in isolation as root words in the text. This information along with parts of speech and other grammatical information are tagged with each entry of the root lexicon in the form of a number as described in section 5. The valid root words are picked by a dictionary and corpus interaction so that most popular words are in the lexicon.

Root words of length up to 3 characters as well as the distinct strings of root words of length greater than 3 are represented in a trie structure. At each leaf node of the trie an address as well as a boolean flag is maintained. The address of a leaf node L_k (where L_k denotes the k th node of L th level) of the trie points to the first word of the main lexicon, the first three characters of which are the same as those we encounter by traversing the trie starting from the root word and continuing up to that leaf node L_k . Therefore, if a string is of length four or more (characters), then we can obtain the address from the trie by using the first three characters and then sequentially searching the words in main lexicon. The lexicon structure of error correction is shown in Fig. 5. We observed that, on average, five sequential searches were necessary to verify the validity of an input string in the lexicon of 60,000 words. We also observed that the number of nodes in the trie was only 4747 for this lexicon.

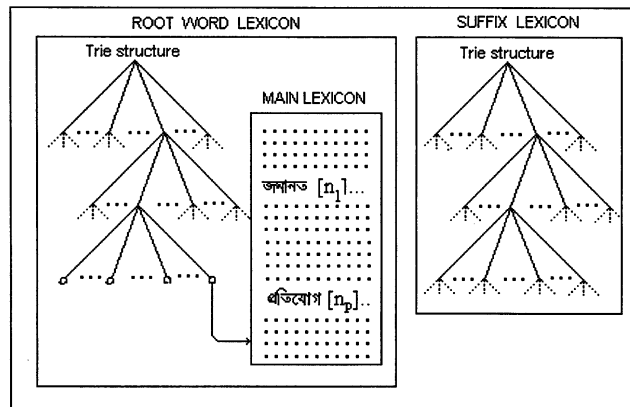


Fig. 5. Lexicon structure for error correction. (See examples in the main lexicon, where parts of speech markers are given in parentheses.)

The boolean flag of a leaf node L_k indicates whether the string of the characters obtained by traversing the trie, starting from the root node and continuing up to L_k , is a valid root word or not. Thus, by means of boolean flags, we can detect the validity of test strings whose lengths are less than or equal to 3, without searching in the root word lexicon.

Our suffix lexicon contains about 800 suffixes. The suffixes were chosen in an interactive way. Initially, about 100 suffixes were used to parse a reasonably large corpus. The words that could not be parsed were manually examined to find new suffixes and were added to the suffix lexicon. In this way, the suffix lexicon was formed.

The suffix lexicon is maintained in another trie structure. Each node in the trie represents either a valid suffix or an intermediate character of a valid suffix. A node representing a valid suffix is associated with a marker needed for a grammatical agreement test, which is described in section 5. If one traverses from the root of the suffix lexicon trie to a node having a 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.

5. ERROR DETECTION APPROACH

For give the reader a better understanding of our approach, the Bangla word morphology is described below.

We will first define some terms which will be used in the subsequent paragraphs. Let a string of characters separated by spaces or punctuation marks be called a *test string*. A test string is a valid word if it has a meaning. *Candidate* words are alternative strings generated by the error correction system for invalid strings. Valid words among the candidates are called *suggested* words. A valid word which has no suffix is known as a *rootword*. The set of suffix parts along with the set of rootword parts constitutes the set of words that may be found in the text. We call them *surface* words.

In the text, a *Bangla* surface 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 major affix classes in *Bangla* are verbal and nominal declensions, called *Bibhaktis*, verbal and other internal affixes, called *Pratyayas*, and nominal suffixes and prefixes, called *anusargas* and *upasargas*. The root words are classified into several clusters like verb-roots, noun-roots etc. The morpho-syntactic rewriting rules are based on these classes. For example, a verb word and a noun word in the text may be conjoining of the form

(Surface)VERB WORD \rightarrow verb root[causational affix]verb declension,
(Surface)NOUN WORD \rightarrow noun root[definiteness affix]case declension.

More complex conjoining such as

Noun root \rightarrow verb root[causational affix]

is also possible. Conjoining is not always a simple concatenation; morphological distortion of the root word is possible. The morpho-phonemic or morpho-graphemic restructuring of symbols at the boundaries of two conjoining morphemes is governed by the *spelling rules* of generative morphology. The spelling rules make the job of detecting the morphemic boundaries more difficult. A spelling rule may affect (or be affected by) the decision of the rule at some other morpheme boundary.

For OCR error detection and correction, we simplified the procedure to a large extent by making the lexicon as follows:

1. If a root word can accept a prefix, then the prefixed root word is considered to be another root word.

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 to be another root word.
3. If two or more root words can conjoin in the form of *Sandhi* or *Samasa*, then the conjoined word is considered to be another root word.

Thus, we have root words and suffixes such that any surface word is a simple concatenation of a single root word string followed by an (optional) single suffix string. In this way, we can avoid a large amount of rule verification during OCR error detection and correction without appreciably increasing the size of the root word dictionary.

We use this simplified morphology in our error detection and correction approach.

From the error detection point of view, the OCR system output can be of two types. In the first type, the *test string* may contain one or more rejected characters. Such a string, called a *partially recognized string*, is inherently erroneous. In the second type, we may have a *fully recognized string* which should be subject to error detection. Since our system is a single error corrector, partially recognized strings having two unrecognized characters are rejected.

As mentioned above, our approach is based on dictionary look-up. However, the dictionary does not contain all the surface words that may be found in the corpus. Therefore, to check if a test string is a valid surface word or not, it is necessary to check if the string is a root word, or if it contains a valid root followed by a suffix that grammatically agrees with the root. The notion of grammatical agreement will be explained later 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 by $W_1 \oplus W_2$, we denote the concatenation of the two strings, i. e., $W_1 \oplus W_2 = x_1x_2\dots x_n r_1r_2\dots r_p$. Let $|W|$ denote the number of characters in string W . Also, if a substring W_s of W contains x_1 (i.e., the first character of the string), then W_s is called a *front substring*. Similarly, W_s is a *back substring* if W_s contains x_n (i.e., the last character of the string). Also we can ‘dissociate’ a substring W_s of W from W . Dissociation is the deletion of substring W_s from left or right of W . Thus, dissociation of W_s from W , namely, $W \ominus W_s = W_r$, results in $W = W_s \oplus W_r$ if W_s is a front substring while it results in $W_r \oplus W_s$ if W_s is a back substring.

If W is a valid surface word, then according to our simplified word morphology, it is possible that

either (i) W is a root word

or (ii) a substring $R_1 = x_1x_2\dots x_r$ of W is a root part and another substring $S_1 = x_{r+1}x_{r+2}\dots x_n$ of W is a suffix which grammatically agrees with R_1 .

If an arbitrary string W satisfies one of the above two conditions, then it may be declared a valid surface word. The segmentation of W into a rootword part and a suffix (including a null suffix) satisfying these conditions may be called *valid parsing*. A valid rootword part with a null suffix implies that the test string is a rootword.

Normally, a valid surface word can be parsed into one rootword part and one suffix (including a null suffix). Very occasionally, however, a string can have multiple valid parses. However, our aim is to get only one valid parse of W so that it can be passed as an error-free word. However, in our NLP version of the algorithm, all parses are detected.

In order to parse an arbitrary string W , we use two separate lexicons, namely, a root word lexicon and a suffix lexicon. To parse a string W , we can start from the left side and find the substrings R_1, R_2, \dots, R_k which are valid root words by looking at the root word

Thus, if $\{R_i, S_j\}$ is a candidate root-suffix pair and if R_i is a verb, then $\{R_i, S_j\}$ is not a valid parse if S_j is not a verb suffix. Moreover, if V denotes the set of all possible verb suffix strings, then the string R_i which is a verb root, can not concatenate with all the members of V to form valid surface words. The valid concatenation is governed by the *parts of speech* and other grammatical information (e.g., morphologically deformed variant or not, etc.) of the root word. To check the validity of the concatenation, the root words are first grouped on the basis of these pieces of information. Each entry in the root lexicon is tagged with its group number. Similarly, suffixes in the suffix lexicon are grouped according to their specific attachment to the roots. Each suffix in the suffix lexicon is tagged with the number of the group to which it belongs.

A table is formed which specifies which groups of suffixes are accepted by the root word of a particular group. Let for verb group A , the acceptable groups of suffixes be a, b, c , and d . Now, for a candidate word, its root group number and suffix group number are checked using the respective lexicons. If the root group number is A , then grammatical agreement is satisfied if and only if the suffix group number is one among a, b, c , and d . Thus, a simple look-up table is necessary to check grammatical agreement.

Since most of the strings are error free (i.e., valid words), the algorithm should be able to check them quickly. Our study on a large corpus showed that if W is a valid surface word and if it has R_1, R_2, \dots, R_k root word substrings, then the probability that a successful parse will be obtained with the largest R_k is highest among R_i 's.

Thus, our algorithm starts from the left side and finds all the root word substrings R_1, R_2, \dots, R_k in W . Next, $W - R_k$ is searched in the suffix lexicon. If success is obtained, and if the suffix grammatically agrees with R_k , then the algorithm is ready to take a new string.

If, however, success is not obtained, then $W - R_{(k-1)}$ is searched in the suffix lexicon, and grammatical agreement with $R_{(k-1)}$ is tested.

In actual practice, the algorithm goes in the suffix lexicon after leaving the root word lexicon (if $R_k \neq W$) and starts searching for valid suffixes in W . The first valid suffix that is equal to $W - R_i$ and that grammatically agrees with R_i signals exit with successful parse. If no such suffix is found, then an error is reported, and the error correction algorithm is called for.

Thus, our algorithm for error detection is as follows:

- Step 1: Initialize with a string W . Start from the left side and find all the root word substrings R_1, R_2, \dots, R_k in W .
- Step 2: If $R_k = W$, go to step 5. Else, start from the right side and find a valid suffix substring S .
- Step 3: If $W \ominus S = R_i$, for any i , then call the grammatical agreement test. If the test is successful, then go to step 5.
- Step 4: Find the next valid suffix substring. If no such substring exists, then go to step 6. Else, go to step 3.
- Step 5: Report success (the string is a valid surface word). Take the next test string and go to step 1.
- Step 6: Report failure. Call the error correction algorithm. Take the next test string and go to step 1.

6. ERROR CORRECTION APPROACH

As mentioned earlier, because of the inflectional structure of surface words in the *Bangla* language, and because the errors in our OCR output are mostly single characters per word, our effort is limited to correcting a single character in a word. Note that, single characters include compound characters that are graphemic combinations of two or more basic characters.

Now, for error correction, we are left with either a *fully recognized string*, which is an erroneous word according to the error detection strategy, or a *partially recognized string* with only one un-recognized character. First, let us consider a *fully recognized string*.

The information about all the valid root substrings R_1, R_2, \dots, R_k and all the valid subscripts S_1, S_2, \dots, S_m is recalled from the error detection program module. Now, the four cases that may arise and their corresponding error hypotheses are described below:

- Case-1: $k = 0, m = 0$, i.e., no substring match is obtained either in the root word lexicon or in the suffix lexicon. Consider the complete test string W as an incorrect root word and correct it using only the root word lexicon.
- Case-2: $k = 0, m > 0$, i.e., no substring match is obtained in the root word lexicon but one or more suffix matches are found in the suffix lexicon (during error detection). It is assumed that one of these suffixes is correct, and that the error occurs in the root word part only. For each suffix S_i , find the substrings R'_i such that $R'_i \oplus S_i = W$ and correct R'_i using the root word lexicon. Look for grammatical agreement with S_i and accept/reject the corrected string.
- Case-2a: If agreement is not obtained for any corrected substring, then correct the whole string W using the root word lexicon.
- Case-3: $k > 0, m = 0$, i.e., one or more substring matches are obtained only in the root word lexicon. It is assumed that one of the root word strings is correct, and that the error occurs in the suffix only. Find the substrings S'_i so that $R_i \oplus S'_i = W$ and correct S'_i using the suffix lexicon. Look for grammatical agreement with R_i as above.
- Case-3a: If agreement is not obtained for any of the corrected S'_i , then correct the whole test string W using the root word lexicon.
- Case-4: $k > 0, m > 0$, i.e., matches are obtained in both lexicons. It is assumed that one of the root word strings is correct (i.e., suffix match is disregarded) and correction is done as in case (3) above.

Note that we have emphasized correction in the suffix for case 3 and case 4 because it is quicker to search the suffix lexicon. It is possible to reduce the computation by neglecting some of the above possibilities. (For example, we could neglect computation of cases 2a and 3a, and reject W as incorrigible.)

Let us now consider the *partially recognized string* which is not subject to error detection. The string W in this case is subject to both root word and suffix substring detection. Here, too, the four situations described above may occur. The error hypotheses are also identical. In each case, however, it is guaranteed that the substring to be corrected contains the unrecognized character.

Whatever the situation might be, either W or its substring should be corrected either for a root word or for a suffix before testing for grammatical agreement. Correction is done as follows.

Let C be a character in the alphabet. A test is conducted a priori on a large set of data for performance evaluation of the OCR, and we let $D(C)$ denote the subset of characters which may be wrongly recognized as C . Members of $D(C)$ may be called confusing characters (similarly shaped characters) for C . See Fig. 7, where misclassification percentages of characters with respect to their confusing characters are shown. Now, if a recognized string W contains a character C , and if it is a misrecognized character, then the correct word for W is one of the strings obtained when C is replaced by elements of $D(C)$. Strings obtained by replacing the character C with the elements of $D(C)$ may be called *alternative* or *candidate* strings. We can attach a probability to each candidate string based on the a priori knowledge of frequency of a confusing character.

Input Char.	Misrecognized as	Percentage of Misclassification	Input Char.	Misrecognized as	Percentage of Misclassification
প	গ	2.13	ক	ফ	2.91
গ	প	3.47	ফ	ক	5.22
ল	ন	2.66	ত	ড	1.22
ন	ল	1.97	ড	ত	3.76
য	য	8.21	য়	য	3.16
য	য	4.11	য	য়	2.94
য	য	5.22	খ	খ	7.39
য	য	7.13	থ	খ	2.79
ডে	টে	1.33	র	ব	2.61
ডে	টে	3.21	ব	র	3.17
ঋ	ঋ	1.14	ঢ	ঢ	3.41
ঋ	ঋ	0.97	ঢ	ঢ	2.83
ড	ড	2.64	এ	এ	1.89
ড	ড	3.79	ঐ	এ	2.32
৳	৳	1.72	শ	ণ	1.21
৳	৳	3.22	ণ	শ	2.17
ম	ম	1.49	ঙ	ঙ	11.13
য়	ম	2.11	ঙ	ঙ	9.14
ঞ	ঞ	7.49	ঙ	ঙ	6.17
ঞ	ঞ	5.33	ঙ	ঙ	8.37

Fig. 7. Some confusing character pairs in Bangla and their percentage of misclassification.

Since we do not know which character in W is misrecognized, we have to generate candidate strings by replacing each character in W with its confusing characters. In generating the candidates, we should be careful about run on, split and deletion errors also. A complete set of candidates is, thus, generated.

According to the performance of our OCR, the number of candidates per character rarely exceeds 4. Therefore, with an average wordlength of about 5 characters (see Table 1), in the worst case, the number of candidate strings per word will be 20. 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 average.

The candidate set thus formed is matched in decreasing order of probability against either the root word lexicon or suffix lexicon. The grammatical agreement of the candidate words is also tested for validity. The valid words are called suggested words. Among these suggested words, we accept the first one. The other suggested words can be used by the user. If there is no match, the test string is rejected.

7. RESULTS AND DISCUSSION

We considered various types of Bangla documents to evaluate the system performance. Books, novels, juvenile literature, newspapers, magazines, question papers etc. were considered for digitization. The print and paper quality was good.

We have noted that our OCR error correction module achieves 84.22% accuracy; i.e., this module can correct 84.22% of words having single character errors. We have also noted that of the 19.45% word error, single character word error is 17.51%. As mentioned earlier, the OCR system achieves 80.55% word level accuracy without an error correction module. As a result, the overall performance of the OCR system is 95.29% at word level with the error correction module. This performance was calculated based on 20,000 OCR recognized words in a single font printed on clear documents with 10-14 point size and scanned at 300 DPI. An example of a Bangla text image is shown in Fig. 8. The document was 12 point *Linotype* font, and it was scanned at 300 DPI.

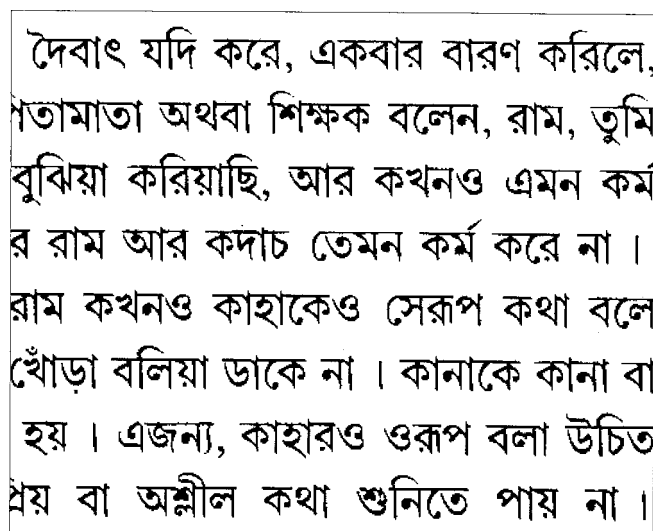


Fig. 8. A typical image of Bangla text.

The error rate depends on the scanning resolution as well as paper quality. We observed that our system provides good results when clear documents with 12 point size are scanned at 300 DPI. We noted that images captured from newspapers generated about 6% more errors than other document images. We also noted that the error rate increased by 9-12% at character level if the document image was captured from an old document.

Some discussion of the performance of the system under size and style variation of the font is in order. Size and most style variations do not affect the modifier and basic character recognition scheme. Compound character recognition is somewhat affected by size variation, but rescaling of templates is effective if the range of size is between 8 and 16 points. For a bold style, a set of bold templates yields better results. We have not studied italics styles in depth since they are rare in Bangla documents. Font variation in Bangla is rare. Most of the publishing houses use the most famous font, *Linotyp*; hence, our OCR system has been developed to handle the Linotype font only.

Our error corrector system uses an identification mark to indicate wrong words which were not corrected by the system. The advantage of this marking scheme is that one can easily find the marked words and perform manual correction.

The original contribution of this work is the development of an OCR detection and correction technique for a highly inflectional language script, in this case, Bangla. An inflectional or agglutinating language has a huge number of 'surface' words in the text as compared to a reasonable number of 'root' words. Direct use of dictionary based spell checking is not an attractive or efficient proposition. We use separate suffix lexicon and root word lexicon in our error detection and correction model. As a result, this approach needs less memory space and is computationally efficient.

From the NLP point of view, our method has an important application in morphological parsing. The proposed spelling detector actually acts as a parser. It parses a given surface word into root parts and suffix parts. All the root parts may not be valid root words. Moreover, more than one valid root word and one or more suffixes may be produced by the parser. Among all the parsed root-suffix pairs, only the ones that satisfy grammatical agreement, as discussed in section 5, are selected as valid parses. As discussed earlier, verb words in Bangla are morphologically deformed because they conjoin with different suffixes. In the case of a morphologically distorted word, the root part may not be a valid root word. These root parts together with their corresponding valid root forms are maintained in the root word lexicon. When such a root part is encountered, the parser outputs the corresponding valid root word.

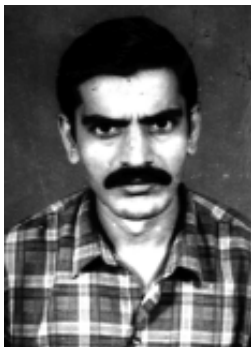
8. CONCLUSIONS

Error detection and correction of inflectional Indian languages is a very difficult problem. To the best of our knowledge, there is no published work dealing with this problem. Here we have developed an OCR single error detection and correction approach for the Bangla language. Our technique can be applied to other inflectional Indian languages as well. The developed error correction module can correct 84.22% of single character errors only. Our future goal is to extend and generalize the work to correct multiple errors. This work has been initiated, and important results will be communicated in a future correspondence. When finished, this work will lead to a spell-checker that can be used in word processing and OCR.

REFERENCES

1. U. Pal, "On the development of an optical character recognition (OCR) system for printed Bangla script," Ph.D. Thesis, Indian Statistical Institute, Calcutta, 1997.
2. B. B. Chaudhuri and U. Pal, "A complete printed Bangla OCR system," *Pattern Recognition*, Vol. 31, No. 5, 1998, pp. 531-549.
3. J. R. Ullmann, "A binary n-gram technique for automatic correction of substitution, deletion, insertion and reversal errors in words," *Computer Journal*, Vol. 20, No. 2, 1977, pp. 141-147.
4. H. Takahashi, N. Itoh, T. Amano, and A. Yamashita, "A spelling correction method and its application to an OCR system," *Pattern Recognition*, Vol. 23, No. 3/4, 1990, pp. 363-377.
5. K. Kukich, "Techniques for automatically correcting words in text," *ACM Computing Surveys*, Vol. 24, No. 4, 1992, pp. 377-439.
6. E. Tanaka, T. Kohashiguchi, and K. Shimamura, "High speed string correction for OCR," in *Proceedings of International Conference on Pattern Recognition*, 1986, pp. 340-343.
7. F. J. Damerau, "A technique for computer detection and correction of spelling errors," *Communications of the ACM*, Vol. 7, No. 3, 1964, pp. 171-176.
8. V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions and reversals," *Sov. Phys. Dokl.*, Vol. 10, No. 8, 1966, pp. 707-710.
9. R. A. Wagner, "Order-n correction for regular languages," *Communications of the ACM*, Vol. 17, No. 5, 1974, pp. 265-268.
10. K. W. Church and W. A. Gale, "Enhanced good-turing and cat-cal: Two new methods for estimating probabilities of English bigrams," *Computer Speech Language*, Vol. 5, No. 2, 1991, pp. 93-103.
11. J. J. Pollock and A. Zamora, "Automatic spelling correction in scientific and scholarly text," *Communications of the ACM*, Vol. 27, No. 4, 1984, pp. 358-368.
12. E. J. Yannakoudakis and D. Fawthrop, "An intelligent spelling corrector," *Information Process Management*, Vol. 19, 1983, pp.101-108.
13. E. M. Riseman and A. R. Hanson, "A contextual postprocessing system for error correction using binary n-grams," *IEEE Transactions on Computer*, Vol. C23, No. 5, 1974, pp.483-493.
14. J. Henseler, J. C. Scholtes, and C. R. Verdoest, "Design of a parallel knowledge-based optical character recognition system," Master of Science Thesis, Dept. of Mathematics and Informatics, Delft University of Technology, 1987.
15. R. C. Angell, G. E. Freund, and P. Willett, "Automatic spelling correction using a trigram similarity measure," *Information Process Management*, Vol. 19, No. 3, 1983, pp. 255-261.
16. J. J. Hull and S. N. Srihari, "Experiments in text recognition with binary n-gram and Viterbi algorithms," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 4, No. 5, 1982, pp. 520-530.
17. W. W. Bledsoe and I. Browning, "Pattern recognition and reading by machine," in *Proceedings of the Eastern Joint Computer Conference*, 1959, pp. 225-232.
18. S. Kahan, T. Pavlidis, and H. S. Baird, "On the recognition of printed character of any font and size," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol.

- 9, No. 2, 1987, pp. 274-288.
19. G. D. Forney, "The viterbi algorithm," in *Proceedings of the IEEE*, Vol. 61, No. 3, 1973, pp. 268-278.
 20. R. Singhal and G. T. Toussaint, "Experiments in text recognition with the modified Viterbi algorithm," *IEEE Transactions on Pattern Analysis Machine Intelligence*, Vol. 1, No. 2, 1979, pp. 184-193.
 21. K. Kukich, "Variations on a back-propagation name recognition net," in *Proceedings of the Advanced Technology Conference*, Vol. 2, 1988, pp. 722-735.
 22. V. Cherkassky and N. Vassilas, "Back-propagation networks for spelling correction," *Neural Networks*, Vol. 1, No. 2, 1989, pp. 166-173.
 23. M. Gersho and R. Reiter, "Information retrieval using self-organizing neural networks," in *Proceedings of International Conference on Neural Networks*, 1990, pp. 405-409.
 24. B. B. Chaudhuri and D. D. Majumder, *Two-Tone Image Processing and Recognition*, Wiley Eastern Limited, New-Delhi, 1993.



U. Pal received his M.S. degree in Mathematics from Jadvpur University in 1990 and Ph.D. degree in Computer Science from Indian Statistical Institute in 1998. He has been with the Indian Statistical Institute, Calcutta, since 1992 and is now a faculty member in the Computer Vision and Pattern Recognition Unit of the Institute. His fields of interest include Digital Document Processing, Optical Character Recognition, Medical Image Analysis etc. In 1995, he received the Student Best Paper Award from the Computer Society of India. He also received a merit certificate from Indian Science Congress Association in 1996. In 1997, he visited GSF, Germany, as a Guest Scientist and worked on Medical Image analysis. He has published about 25 research papers in various reputed national and international journals and conference proceedings. He is a member of IUPRAI (Indian unit of IAPR), the Computer Society of Indian and the Indian Science Congress Association.



P. K. Kundu received his M.S. degree in Mathematics from Calcutta University in 1976. He has been with the Vidyasagar college for Women, Calcutta, since 1984 and is now a Senior Lecturer of Mathematics. His fields of interest include Natural Language Processing and Computer Vision. He has 14 research publications in various reputed journals and conference proceedings.



B. B. Chaudhuri received his B.S. (Hons), B. Tech. and M. Tech. degrees from Calcutta University, India, in 1969, 1972 and 1974, respectively, and Ph.D. degree from the Indian Institute of Technology, Kanpur, in 1980. He joined the Indian Statistical Institute in 1978, where he served as the Project Coordinator and Head of the National Nodal Center for Knowledge Based Computing, funded by the United Nations Development Program and the Dept. of Electronics. Currently, he is the Head of the 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. He has published about

200 research papers in reputed international journals, conference proceedings and edited books. Also, he has authored two books entitled "Two Tone Image Processing and Recognition" (Wiley Eastern, 1993) and "Object Oriented Programming: Fundamental and Applications" (Prentice Hall, 1998). He was awarded the Sir J. C. Bose Memorial Award (twice) for best application oriented papers in 1989 and 1991, the Homi Bhabha Fellowship award in 1992 for OCR of the Indian Languages and computer communication for the blind, the Dr. Vikram Sarabhai Research Award in 1995 for his outstanding achievements in the fields of electronics, informatics and telematics and the C. Achuta Menon Prize in 1996 for computer based Indian language processing. He worked as a Leverhulme visiting fellow at Queen's University, U.K., during 1981-82, as a visiting scientist at GSF, Munich, and a guest faculty member at the Technical University of Hannover during 1986-88 and again during 1990-91. He is a senior member of IEEE, a member secretary (Indian Section) of the International Academy of Sciences, a Fellow of the International Association of Pattern Recognition, a Fellow of the National Academy of Sciences (India), a Fellow of the Institution of Electronics and the Telecommunication Engineering and a Fellow of the Indian National Academy of Engineering. He is serving as associate editor of Pattern Recognition, Pattern Recognition Letters and VIVEK. He also served as guest editor of a special issue of the Journal of Institution of Electronics and Telecommunication Engineering.