WORD RECOGNITION IN INDIC SCRIPTS

Thesis submitted in partial fulfilment
of the requirements for the degree of

MS By Research
in
COMPUTER SCIENCE

by

Naveen T S
201050094
`naveen.ts@research.iiit.ac.in`

CENTER FOR VISUAL INFORMATION TECHNOLOGY
International Institute of Information Technology
Hyderabad - 500 032, INDIA
December 2014

International Institute of Information Technology
Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled "Word Recognition in Indic Scripts " by Naveen T S, has been carried out under my supervision and is not submitted elsewhere for a degree.

Date	Adviser: Prof. C V Jawahar

To LIFE

Acknowledgments

First and foremost, I wish to thank Prof. C V Jawahar for his guidance and supervision during my thesis work. He was always there whenever I needed advice and his encouraging and motivating words have helped me multiple times during the course of this work.

The annotation team comprising of Phani, Rajan and Nandini have helped me more times than I can count. Their immense support in getting annotations done in a fast and efficient way will never be forgotten. Satya have been my go-to person right from day one for any problems. Be it procurement of new machines or helping me during my travel claim submissions, he have always helped me with a smiling face. Anand Mishra, Praveen Krishnan, Ravi Shekhar have all helped me multiple times in past with their comments, suggestions and reviews of my conference submissions.

Teaming up and working with others have given me some of the best moments in my CVIT life and for that I thank Shrey Dutta, Udit Roy, Aman Neelappa and Pramod Sankar. A special thanks to Pramod for his guidance and advice during the two works which we did together. My life at IIIT would not be complete without the mention of my dear friends Basil, Sandhya, Rashad and Rinu. You folks gave me a ton of happy memories during my stay at IIIT and I am grateful for that. Special thanks to the all night *"chai-walas"* who gave me hot tea and breakfast even at 4 in the morning.

Throughout my studies, my parents have given their full support. I thank my parents, relatives and all my other friends who have helped me during my stay at IIIT.

Abstract

Optical Character Recognition (OCR) problems are often formulated as isolated character (symbol) classification task followed by a post-classification stage (which contains modules like UNICODE generation, error correction etc.) to generate the textual representation, for most of the Indian scripts. Such approaches are prone to failures due to (i) difficulties in designing reliable word-to-symbol segmentation module that can robustly work in presence of degraded (cut/fused) images and (ii) converting the outputs of the classifiers to a valid sequence of UNICODES. In this work, we look at two important aspects of word recognition – word image to text string conversion and error detection and correction in words represented as UNICODES.

In this thesis, we propose a formulation, where the expectations on the two critical modules of a traditional OCR (i.e, segmentation and isolated character recognition) is minimized. And the harder recognition task is modelled as learning of an appropriate *sequence to sequence transcription scheme*. We thus formulate the recognition as a direct transcription problem. Given many examples of feature sequences and their corresponding UNICODE representations, our objective is to learn a mapping which can convert a word directly into a UNICODE sequence. This formulation has multiple practical advantages: (i) This reduces the number of classes significantly for the Indian scripts. (ii) It removes the need for a word-to-symbol segmentation. (ii) It does not require strong annotation of symbols to design the classifiers, and (iii) It directly generates a valid sequence of UNICODES. We test our method on more than 5000 pages of printed documents for multiple languages. We design a script independent, segmentation free architecture which works well for 7 Indian scripts. Our method is compared against other state-of-the-art OCR systems and evaluated using a large corpora.

Second contribution of this thesis is in investigating the possibility of error detection and correction in highly inflectional languages. We take Malayalam and Telugu as the examples. Error detection in OCR output using dictionaries and statistical language models (SLMs) have become common practice for some time now, while designing post-processors. Multiple strategies have been used successfully in English to achieve this. However, this has not yet translated towards improving error detection performance in many inflectional languages, especially Indian languages. Challenges such as large unique word list, lack of linguistic resources, lack of reliable language models, etc. are some of the reasons for this. In this thesis, we investigate the major challenges in developing error detection techniques for highly inflectional Indian languages. We compare and contrast several attributes of English with inflectional languages such as Telugu and Malayalam. We make observations by analysing statistics computed from popular corpora and relate these observations to the error detection schemes. We propose a method which can detect errors for Telugu and Malayalam, with an F-Score comparable to some of the less inflectional languages like Hindi. Our method learns from the error patterns and SLMs.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

Optical Character Recognition (OCR) is the process of converting a document image to its corresponding text. Document image could be from magazine papers, newspaper clips or from text books and could be captured using a scanner or via camera. There are several areas where OCR technology is being used. Automatic recognition of vehicle number plates, recognition of road signs, helping visually challenged people read, automatic filling of forms into computer, compression of digital image into UNICODE text etc. are some of the application areas. Several commercial and open-source systems exist which deliver high quality OCR systems.

Commercial OCR engines began to appear during the mid 1960s where recognition was done using template matching techniques [61]. This was followed up by machines that can recognize machine printed and hand-written numerals. During the same time, Toshiba launched the first automatic letter-sorting machine for postal code numbers. During mid 70s and 80s, several OCR engines were developed which could recognize English printed as well as hand-written characters [61, 33].

Recent advancements in both computational power and better scanning devices have lead to the development of several commercial systems [5, 6, 1]. Most of these systems can perform character recognition on complex layouts with sub 1% error rates. Many of these systems are also integrated with portable devices like mobile phones, tablets etc.

Of late, OCR systems have been developed for complex languages like Arabic [63, 30], Chinese [7] etc. Although several such systems exist, none of them can be compared to English OCR systems as they have relatively poorer accuracy when compared against English OCRs. One main reason is that English as a script is less complicated than Arabic or Chinese or Indian scripts.

1.2 Architecture of a Typical OCR

The process of OCR-ing typically begins with a document image. They are obtained typically by scanning or by camera capture. Documents captured using such ways will contain lots of variations and requires a set of pre-processing operations to be performed on them before they can be OCRed. Preprocessing typically includes binarization and noise cleaning followed by skew detection and correction. Binarization refers to the task of converting the original image, which could be in color or grey, to binary format. This process is required as most of the subsequent processes works best with binary images. Popular binarization techniques include Otsu method, Morphology based, Sauvola method etc.. This is followed by correcting the skew of the image as typical scanned input will have a small degree of skewness present in them. There are different methods for correcting the skew and most of them are language specific. For eg: typical skew correction mechanism for Indian language of Devanagari take advantage of the fact that Devanagari words have a head-line (*shiro-rekha*) connecting all the components.

Once the document is processed, a layout engine is used to extract the structure of the document. Typical documents include texts, images, tables etc. in them and it is very important to identify these components. The text content might be split into several columns and layout analysis is required to identify the blocks which needs to be OCRed and also to reconstruct the layout once we generate the OCR output. Several algorithms exist which can be used to extract the lines and words from the text blocks.

Words are then segmented to symbols often by a connected component analysis. Each of these symbols are then recognized/classified. Extracting discriminative features is a key step in any classification problem. Several local and global features can be used in this task. However, using high dimensional features can make the pattern classification process more complicated. Some commonly used features include HoG, SIFT, profile based features, apprarance based features, PCA of raw-pixels etc.

Once we have the features extracted, we use a classifier to recognize them. The purpose of a classifier is to 'classify' the input features into one of the possible output labels. If the input features can be linearly separated, then use of linear Support Vector Machines (SVM) can come in handy. Else, there exists several other classifiers like Neural Networks, HMMs, Random Forests etc. which can perform the classification. One main requirement for any machine learning technique is the availability of considerable amount of raining data. Lack of training data can often generate a classifier model which may not perform as expected. The classifier model generated after training is stored so that it can be used during testing. There are also several approaches wherein the classifier continues to improve itself in a semi-supervised way.

The labels generated by the classifier needs to be converted to corresponding output format. For documents, it will usually be in UNICODE. However, enough care should be taken when performing this conversion as UNICODE requires you to re-order the output in many cases so that it could be rendered correctly by various text editors. Also, there could be cases where multiple connected components needs to be combined together and assigned a single label.

A post-processor module is used to correct any errors which the classifier might have made. They use dictionaries and other language models to perform this task. In this thesis, we focus on the recognizer module and the post-processing module. Figure 1.1 depicts a typical OCR architecture/work-flow.

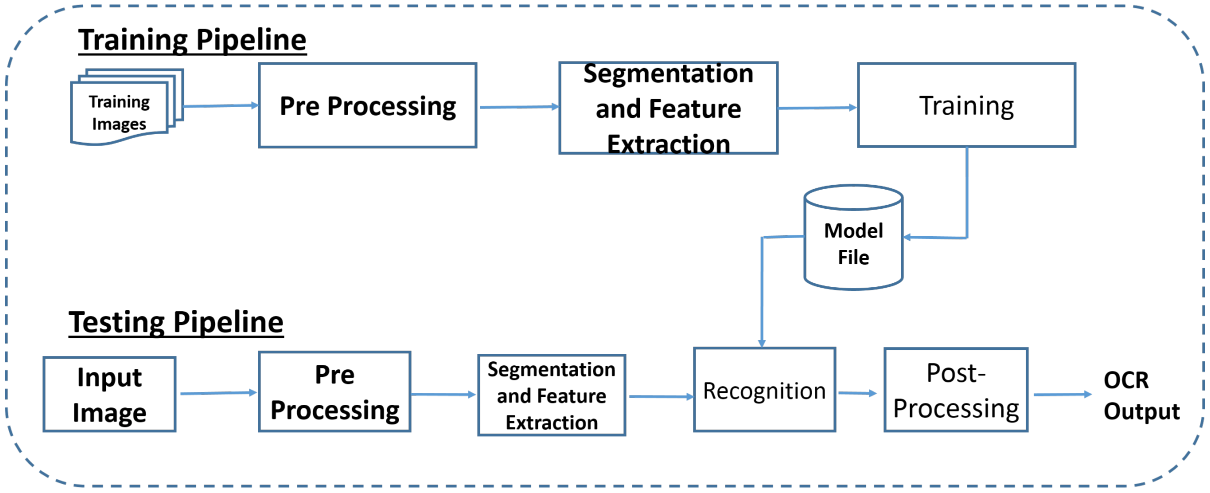


Figure 1.1 Traditional OCR architecture showing the training and testing pipeline

1.3 OCR for Indian Languages

1.3.1 Script and Language

India is a land of several languages. The 2001 official census [4] reports of the existence of 30 languages having more than a million native speakers and 22 of them being granted official language status. Most of these languages have also their own numeral representation systems and are commonly used in print. Broadly, Indian languages have been rationalized into 200+ mother tongues, and grouped under 100+ languages. Table 1.1 shows some details into these language families. An interesting observation from the table is that eventhough Tibeto-Burmese family consist of 66 languages, they are used by only 1% of population.

Language Family	# of Languages	Sample Language	% of total population
Indo-Europian	24	Hindi, Bangla, Afghani, Urdu, Sanskrit	76.89%
Dravidian	17	Kannada, Malayalam, Tamil, Telugu	20.82%
Austro-Asiatic	14	Santali, Nicobarese, Korwa	1.11%
Tibeto-Burmeese	66	Tibetian, Bodo, Ladakhi, Manipuri	1.00%
Semito-Hamitic	1	Arabic	0.01%
Total	**122**	-	**99.83%**

Table 1.1 Table showing major language families in India.

From Table 1.1, we can observe that there are two main language families that exist in India: Indo-Aryan and Dravidian languages, with both the groups having derived from a common ancestor, Brahmi. Indo-Aryan language consists of Bangala, Gujarati, Gurumukhi, Hindi etc. and represent the language set spoken by majority of Indians. They follow the devanagari way of writing. Dravidian language consists of Malayalam, Tamil, Telugu and Kannada and is spoken mostly in south India.

Figure 1.2 shows an example on how a same word is written in different languages. Most Devanagari script based languages can be identified on the basis of *shiro-rekha*, a line passing through top region of the word, connecting all the components. The Dravidian languages, on the other hand, have a unique ‘curvy’ style of writing. One reason for such a style is because of the writing media which was historically in use. Ancient south Indians used iron stylus to write on dried palm leaves. Straight lines would have made cuts in the palm leaves.

Most modern day texts use western punctuations but some scripts use traditional punctuation marks. eg: Devanagari uses *poorna-viranam (DANDA)* (|) to mark the end of sentences. Words are typically separated by spaces.

Number System: All scripts have their own numbering system. However not all scripts use them frequently. Southern scripts like Tamil, Malayalam etc. prefer the use of Arabic/European numerals over their own representation while scripts like Hindi, Bengali etc. heavily use their own scripts. A notable outlier in this is Tamil, which does not use decimal system. So, there is no symbol representing zero. There are however, additional symbols representing 10, 100, 1000. Modern Tamil hence uses European numerals.

The presence of many similar looking glyphs is another interesting aspect of Indic scripts. Every language has a set of glyphs which would cause confusion among each other. Figure 1.3 shows some of those characters present among Indic scripts. A slight degradation can easily make one symbol look like another.

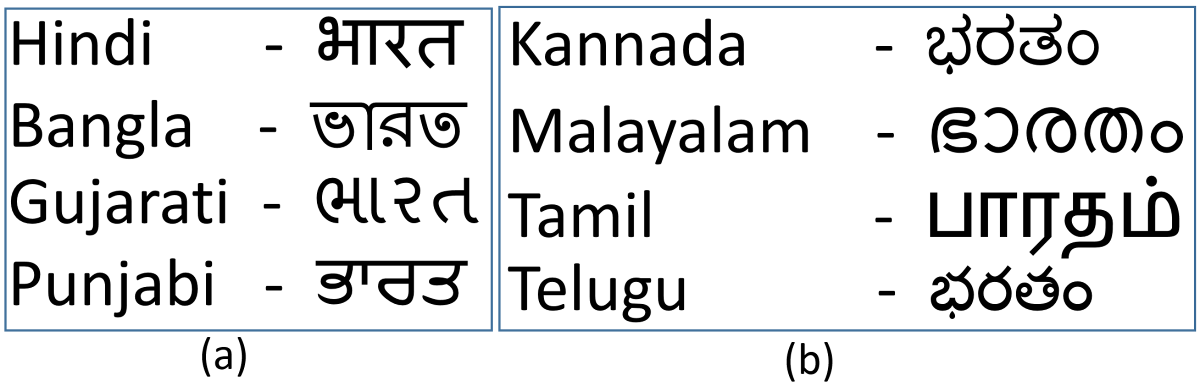

Hindi - भारत
Bangla - ভারত
Gujarati - ભારત
Punjabi - ਭਾਰਤ
(a)
Kannada - ಭರತಂ
Malayalam - ഭാരതം
Tamil - பாரதம்
Telugu - భరతం
(b)

Figure 1.2 Above image shows how a word *Bharat* is written in different languages in India. Figure(a) shows Devanagari script based languages and (b) shows Dravidian languages.

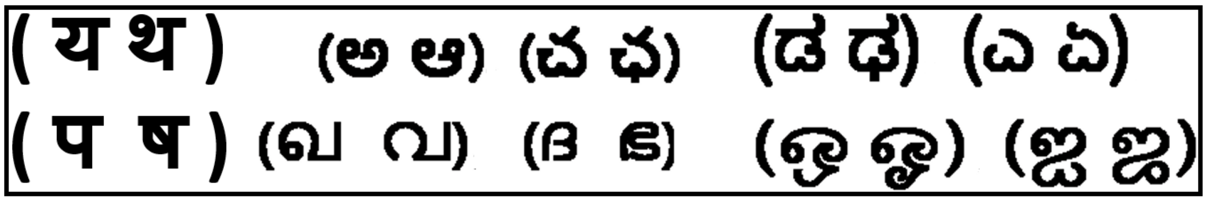


Figure 1.3 Some examples of confusing symbols present in Indic scripts. Notice that with slight degradations, one symbol can easily be confused for another. A larger context needs to be used to correctly identify the symbol.

Script representation : UNICODE is the most popular way of representing script information for Indic scripts. They use a 16 bit representation for each symbol. Every language is assigned a separate block to represents the symbols in case of UNICODE. UNICODE standards for Indic scripts are derived from ISCII (Indian Standard Code for Information Interchange), which used an 8 bit representation. Indian UNICODE script blocks are a superset of ISCII. Every UNICODE script block has the first 85 characters in same order and position as the 1998 ISCII characters for respective scripts. Every single script blocks orders analogous characters in the ISCII range in the same relative locations to the start of the blocks across all major scripts. The next 27 characters are additional UNICODE characters, where each analogous character across the scripts is also assigned to the same code point relative to the beginning of the block. The final column is reserved for script specific characters. There is no special ordering here. Table 1.2 shows the UNICODE range for the major Indian languages.

Language	Unicode Begin	Unicode End
Hindi	0900	097F
Bangla	0980	09FF
Gurumukhi	0A00	0A7F
Kannada	0C80	0CFF
Telugu	0C00	0C7F
Tamil	0B80	0BFF
Malayalam	0D00	0D7F

Table 1.2 UNICODE ranges of major Indian scripts

1.3.2 Challenges in Design of IL-OCR

A significant challenge in designing an Indic OCR is the presence of large number of symbols. While for English, there are primarily 52 characters, most Indic scripts have symbols in excess of 200. This is not counting the numerals and other punctuations which occur commonly in the texts. The structure of the symbols also can cause significant challenges during recognition. Most Indic script glyphs are written in a much complicated way than English or other European scripts. Many of them looks very similar except for some minor changes in the glyph. In Indic scripts, two or more characters can be combined together to form another characters. They are commonly referred to as '*samyuktaksharas*'.

Several challenges exists when the expected output of an OCR is UNICODE. One main challenge is in the area of rendering *'matras'* or vowel modifiers. Such modifiers can exist on top, bottom, left, right or even in some cases, on both sides of the symbol. Hence, script specific rules need to be written which, based on *matra* position and the associated symbol, outputs the UNICODE. The UNICODE standard requires that all combining characters be stored in memory after the base character they are combined with. This is a significant concept as it means that, even if the combining glyph occurs to the left of the base character, it is stored **after** the base character in memory. This is often referred to as logical ordering. Also, a symbol can have multiple matras associated with it, making the job even more difficult. Figure 1.4 shows a typical scenario in Indian language OCR. The word in the figure is split into corresponding symbols and then recognized independently. This is shown in the middle block. As you can observe, the recognition of each symbol is happening according to the order in which the symbols appear in the image. However, for the UNICODE rendering to occur, we need to re-arrange some symbols, combine some other symbols and finally generate the UNICODE. In our example, the first and third component shown in middle block is actually a single UNICODE. While rendering, these two symbols will be placed on either sides of the associated consonant. Also, during UNICODE rendering, the *matra* should be placed **after** the consonant even if it is expected to be seen before the consonant. Hence the need to rearrange the UNICODEs before we output them. An interesting observation in this case is that in the case of the symbols present in the middle block, if either first symbol or third symbol occurs in isolation along with a consonant, then it has a completely different UNICODE value and if both symbols occur together, then the two symbols must be joined. Such set of rules are specific to language and you need to have the necessary language knowledge to write the rules so that the symbols can be converted to UNICODE.

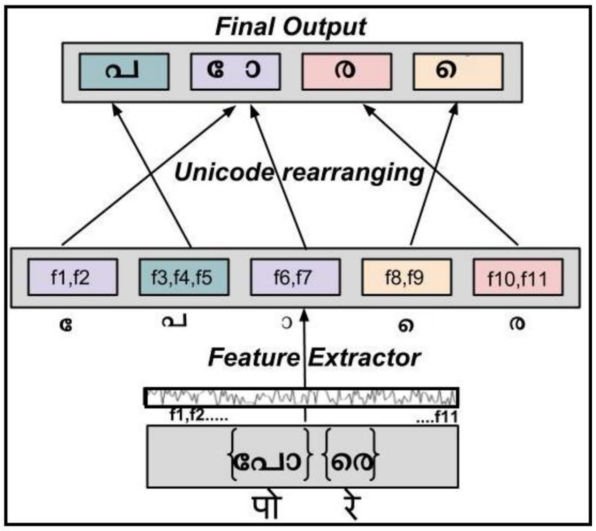


Figure 1.4 Above figure shows how UNICODE rearranging has to occur so that the final output is rendered correctly.

There are several text editors that support UNICODE rendering of Indic scripts. However, lack of consistency among them in the way they render the symbols make the task of recognition even harder. For eg: there are many symbols which are formed by joining two or more symbols, known as *samyuktaksharas*. However, different editors render them differently which in turn means that sometimes the symbols are fused together to look like a single component while some other times, it looks as two distinct symbols.

Another interesting aspect is that there exists multiple symbols having the same UNICODE value. A classic example is the case of Malayalam, a south Indian language spoken in the state of Kerala, where a script revision has occurred which resulted in complete change of how symbols are written. This resulted in cases where same symbol can be written in completely different ways.

1.4 Previous Work related to Indian Language OCR

A complete OCR system for printed Bangla script is presented by Chaudhuri and Pal [19]. In this work, they use stroke based features to design a tree based classifier. These classifiers are used specifically to recognize compound characters. This is followed by a template matching approach to improve accuracy. The character unigram statistics is used to make the the tree classifier efficient. Several heuristics are also used to speed up the template matching approach. A dictionary-based error-correction scheme has been integrated where separate dictionaries are compiled for root word and suffixes that contain Morphys-syntactic information as well.

An Assamese OCR has been developed using Bangla OCR as there are many similar characters present in Assamese and Bangla. Ghosh *et.al.* [36] uses two stage SVMs to improve the performance. They use a combination of directional features which act as local features and some global features like profile based features. The confusing classes from first SVM is sent to a second SVM to improve recognition rate. They also take assistance from a frequently occurring word list to correct some OCR errors.

Antanani and Agnihotri [60] reported character recognizer for Gujarathi script that uses minimum Euclidean distance, hamming distance and KNN classifier with regular and Hu invariant moments. Lehal and Singh reported a complete OCR system for Gurmukhi Script [35]. They use two feature sets: primary features like number of junctions, number of loops, and their positions and secondary features like number of endpoints and their locations, nature of profiles of different directions. A multistage classifications scheme is used by combining binary tree and nearest neighbour classifier. They supplement the recognizer with post-processor for Gurmukhi Script where statistical information of Panjabi language syllable combinations, corpora look-up and certain heuristics have been considered. They reports an accuracy of 96.6% on a test dataset of 100 pages. An OCR system was also reported on the recognition of Tamil and Kannada script [34]. Recognition of Kannada script using Support Vector machine (SVM) has been proposed [69]. To capture the shapes of the Kannada characters, they extract structural features that characterizes the distribution of foreground pixels in the radial and angular directions.

Neeba and Jawahar [49] have performed an empirical evaluation of different character classification scheme for different languages with focus on Malayalam. Malayalam and Telugu are considered relatively tough languages among Indian scripts due to the presence of complex glyphs, large number of symbols etc. Also, for both these languages, the *matras* can appear on left, right, top, bottom or even on multiple sides of the vowels, making the recognition challenging. Shrey *et.al.* [67] have attempted recognition of Malayalam documents using a character n-gram approach. Their approach is robust against character degradations but requires huge memory during execution. They use profile based features along with approximate nearest neighbour classifier for recognition. For Telugu, Rasagna *et.al.* [75] have developed character classification schemes which are robust against font variations and can be scaled to large number of classes. They have tested their approaches in nearly 1.5 million character samples belonging to 359 classes and having 15 font variations. They obtain an accuracy of 96-98% using SVM classifiers. Pavan Kumar *et.al.* [**?**] have improved the OCR accuracy by improving character segmentation algorithm and errors caused due to broken characters. They propose to use the feedback from distance measure used by classifier to handle broken characters.They also use script specific character segmentation methods to improve segmentation accuracy. There have also been other recognition activities in Telugu [50, 17]. A bilingual OCR for recognizing Hindi and Telugu has been proposed by Jawahar et.al. [26].

Similar systems have also been developed for Tamil [14, 40]. There have also been attempts in recognition of Urdu Nastaleeq script using LSTM neural networks [8]. They have used simple RAW pixels as features and have obtained high recognition rates. A more detailed survey on IL-OCR has been done by Pal and Chaudhuri in [53]. A recent comparison and evaluation of state-of-the-art OCR system for Indian languages have been performed in [15]. An overview into the IL-OCR activities for all languages is provided in Table 1.3.

1.4.1 Why newer methods as part of this thesis?

Unfortunately most of the previous methods reported above failed to deliver a working prototype. Implementations are often non-existent now. In many cases, the evaluation was done on authors' private collection and often on tiny sample set.

Our focus is on making a robust OCR system that can work on "real-life" documents. We would like to test on a reasonably large set of documents scanned from books and material collected country-wide.

In addition to the practical aspects, we are also interested in developing methods that advance the current state of the art (in minimizing the handcrafting the solutions).

1.5 Overview of This Work

In this thesis, we propose methods towards addressing the problem of word recognition for Indic scripts. We propose a script independent, segmentation free method which will be evaluated against a large corpora. We also look at the challenges in developing a language post-processor for Indic scripts and possible solutions to them.

Language	Classifier	Features	Accuracy(%)	Reference
Assamese	Two stage SVM	Combination of directional and profile based features	97	[36]
Bangla	Tree Classifier	Stroke based features	94-97	[19]
Gujarati	KNN	Moments	67	[60]
Gurmukhi	Binary Tree Classifiers	Structural features	96.6	[35]
	KNN	Structural features	95	[45]
Hindi	Distance based	Moments, Structure based features	93	[22]
	Multiple Connectionist	Structural, density features	93	[21]
	Tree Classifier	Structural, template features	93	[52]
Kannada	SVM	Structural features	86	[69]
	KNN/ANN	DCT,DWT etc.	93-99	[20]
Malayalam	Approximate nearest neighbour	Profile features	95	[67]
Tamil	Time Delay Neural Network	Gabor Filters	90-97	[40]
	Nearest Neighbour	Moments, DCT	95	[14]
Telugu	Fringe Distance	Fringe Map	92	[17]
	Template Matching	Quantized relative directional features	78-95	[50]
Urdu	LSTM Networks	RAW Pixels	86.5 - 95	[8]
	Tree Classifiers	Structural Features	97.8	[72]

Table 1.3 A Brief Overview of the Major Works in Indian Language OCR

1.5.1 Motivation

We are motivated by the fact:

- most of the Indic OCRs tend to be fragile due to the word to symbol segmentation module. We would like to avoid this step.
- errors in classification percolate to the UNICODE and this creates invalid outputs.
- though the scripts are widely different, we would like to investigate the possibility of a single common architecture for a larger set of scripts.
- often machine learning is used only for the training for classifiers (for symbols). We would like to use the rich annotated training data available for all these scripts.
- we also would like to investigate the post-processing scheme which bypasses two existing directions based on (i) dictionary (Dictionaries could be non-existent, incomplete or infeasible) (ii) word morphology (language processing modules which can analyse or synthesize in presence of noise is missing for most Indian languages.) Our focus is on languages which are highly inflectional.

1.5.2 Scope of the Work

Different scripts have different symbol sets even if they belong to same language family. This results in development of different OCR architectures for different languages as some languages might work better with SVMs while others mght work better with KNN or neural networks. We wish to have uniform architecture for the recognition of all scripts. Such an architecture will help others in developing OCRs for a new language. All we would need would be enough training data and the new OCR should be ready.

We also wish to use machine learning algorithms extensively in our work. Using more formal methods instead of some heuristic based solutions will lead to developing a robust OCR architecture. Training on huge amounts of data, easily adapting our method to new script variations, newer data etc. are some of the objectives which we want to achieve through this work.

Evaluation of our method on large corpora will ensure that our approach is robust against any unseen data. This is significant as the applications of OCRs are many and while others use our OCR, we should ensure that the performance doesnot drop for new data.

We also wish to understand the linguistic challenges of Indian languages from a statistical point of view. This is significant if we wish to develop a statistical language model for Indic scripts. We wish to understand what are the challenges present in our scripts from a linguistic point of view and try to identify how we can overcome those challenges.

1.5.3 Contributions

Extraction of symbols from word posses considerable challenge for Indic scripts. Mostly, heuristic methods based on connected components are used to extract symbols. However, such methods are prone to fail when the input image have cuts or merges. Our work focuses on designing a segmentation free architecture for Indian language OCRs. We will be directly recognizing words instead of symbols. An added advantage of direct word recognition systems is that we do not have to extract individual symbols from words and annotate them for training. There are several automatic methods available which will segment words from lines with minimal manual intervention.

We wish to consider the problem of recognition as that of a sequence transcription problem where we transcribe the input feature sequences into output UNICODE sequences. Such an approach helps us in avoiding the need to know the language specific UNICODE rearranging rules thus making our method a script independent one.

This work will also focus on development of a post-processor module for highly inflectional languages. We try to understand the challenges posed by Indian languages and proposes methods to overcome them.

In short, our contributions are the following:

1. We demonstrate a generic solution to the word recognition in a variety of Indic scripts. Method avoids (i) segmentation (ii) Unicode generation from class labels.

2. We validate on a large real-life corpus. We also compare with the best available solutions.

3. We investigate the challenging post-processing problem and suggest promising directions for highly inflectional languages, where the dictionary based methods need not be the most appropriate.

1.5.4 Organization of Thesis

Chapter 2 introduces us to the commonly used character and word recognition strategies. We shall explore how each of them works and identify an architecture suited for our problem. We then proceed to design our transcription module for Indic scripts in Chapter 3. The results of our transcription module when evaluated using a large corpora is shown in Chapter 4. We also compare our method against other state-of-the-art OCRs. In Chapter 5, we understand the linguistic issues from a statistical point of view and propose a solution to overcome them and develop a post-processor for highly inflectional languages. We finally conclude this thesis in Chapter 6.

Chapter 2

Recognition Architecture

The *brain* of any recognition architecture is the classifier. The classifier decides the OCR output based on the input feature it received. Nearest neighbour classifier, Support vector machines, multi layer perceptron etc. are some of the commonly used classifiers. Similarly, there exists multiple recognition strategies ranging from recognition of isolated characters (symbols) to holistic word recognition or even recognizing entire sentences. This chapter deals with some of the commonly used recognition strategies and the classifiers being used in them.

2.1 Word Recognition by Recognizing Individual Characters

In this approach, we consider a word to be an ordered collection of isolated characters. We recognize these isolated characters and then fuse them together to get the word output. A typical character OCR architecture is shown in Figure 2.1. The biggest advantage of such a method is that the number of classes to be recognized is kept to a minimum. Typically, we extract connected components from the words and they are considered as symbols to be recognized. However, segmentation of word to symbol is a challenging task especially when there are degradations like cuts,merges etc. present in the word image. For certain languages like Hindi, as additional step where we remove the *shiro-rekha* from the word image is also involved. We extract features for each of these symbols and they are send to classifier. Classifier generates latent symbols as its output and they must be processed further to generate word output. e.g: for character ”*i*” the tittle (dot on its top) and the body should be considered as a single component even if they might be recognized as two symbols. Hence, these two symbols must be fused together to form the character *i*.

A major challenge with such a system is that we require annotated symbols for training. Generating many such samples is a tough task as there are no automatic ways to get the symbols annotated.

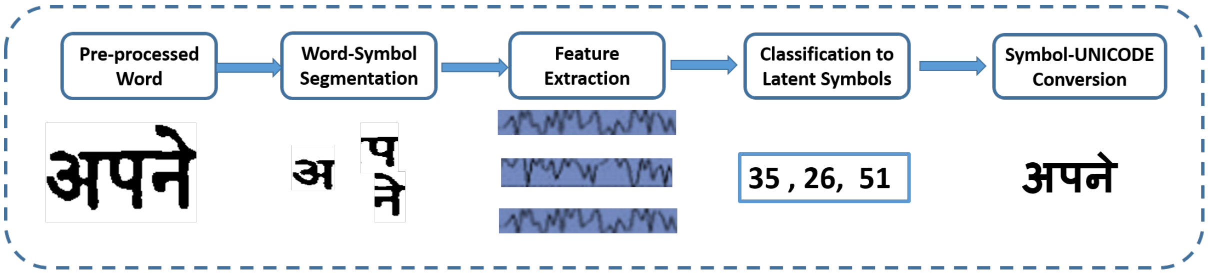


Figure 2.1 General character recognition system pipeline

2.2 Character Pattern Classification

2.2.1 Support Vector Machines

Support Vector Machines are type of classifiers which is used to separate two classes by using a hyperplane between them. It is a supervised learning method where we provide labelled data for training. A learning algorithm is used to generate a model from these training data. Given a test data, we use this model to assign a label to the testing sample. In this sense, SVM is mostly a linear classifier.

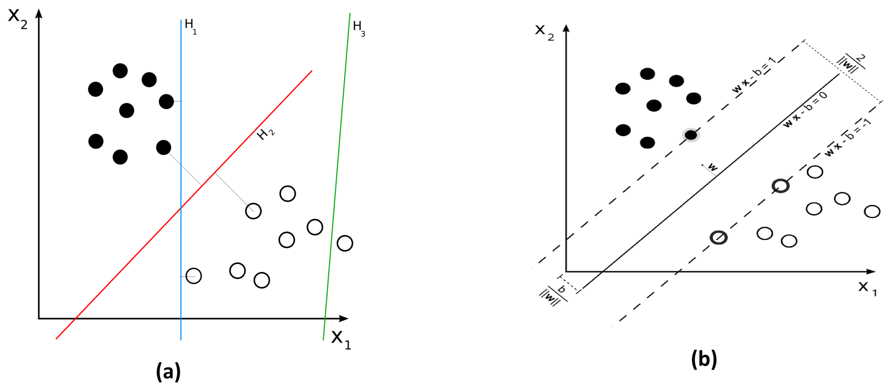


Figure 2.2 Figure (a) showing multiple ways the classes can be separated. Figure (b) shows the optimal line separating the classes.

Given an n dimensional vector, the objective of an SVM learning algorithm is to identify a n-1 dimensional hyperplane which can separate the training classes. Theoretically, there could be many such hyperplanes, as shown in Figure 2.2(a). Our objective is to find a hyperplane which have the largest margin between the training classes. This is shown in Figure 2.2(b). Such a classifier is called a linear classifier as it is assumed that the training classes will be linearly separable in an n dimensional space.

Incase the classes are not linearly separable, we transform the input vector to a higher dimensional space where the points become linearly separable. Such transformation functions are known as kernal functions. Polynomial, RBF, hyperbolic tangent etc. are some of the commonly used kernal functions.

Classification using SVMs : The SVM mentioned above can be directly used only for two-class classification problems. However, most of the general applications in machine learning will have multiple classes to deal with. In these cases, we use a multi-class SVM. The common approach in implementing a multi-class SVM is by dividing the single multiclass problem into multiple binary class problem. There are several ways in which we can implement this. Some of the commonly used approaches are:

- One Vs Rest: We train the binary classifiers to distinguish between one class and the rest of all classes combined. So, for a n-class classification problem (n $>$ 2), there would be n binary classifiers generated. During testing, we use a "winner-takes-all" approach where the classifier with highest output function is identified as the class of testing sample.

- One Vs One: Here, we build a set of binary classifier between every pair of classes. So, there would be $\frac{n.(n-1)}{2}$ classifiers for *n* classes. During testing, we use a "max-win" strategy wherein the class which got the maximum votes is assigned as the class of the testing sample.

- DAG-SVM: DAG-SVM uses a modified version of 1 vs 1 strategy where the binary classifiers are stored as a DAG (Direct Acyclic Graph). Such a structure helps us during testing where-in the sample need not be evaluated using every classifier.

The above mentioned strategies are shown in Figure 2.3. In case of DAG-SVM, we take on of the two possible paths available at any given node based on the classifier output for that node. This helps us in minimizing the number of classifications performed during testing time. Figure 2.3(b) shows all possible classification boundary for class 1. Similarly, we shall have classifications done for classes 2 and 3. Eventhough the number of binary classifiers generated for the above mentioned two strategies are same, the testing time is much less for DAGSVM. Figure 2.3(c) shows the classification boundary between the four classes in a 1 vs rest approach. Black line shows 1 vs rest classifier, blue for 4 vs rest, green for 3 vs rest and red for 2 vs rest.

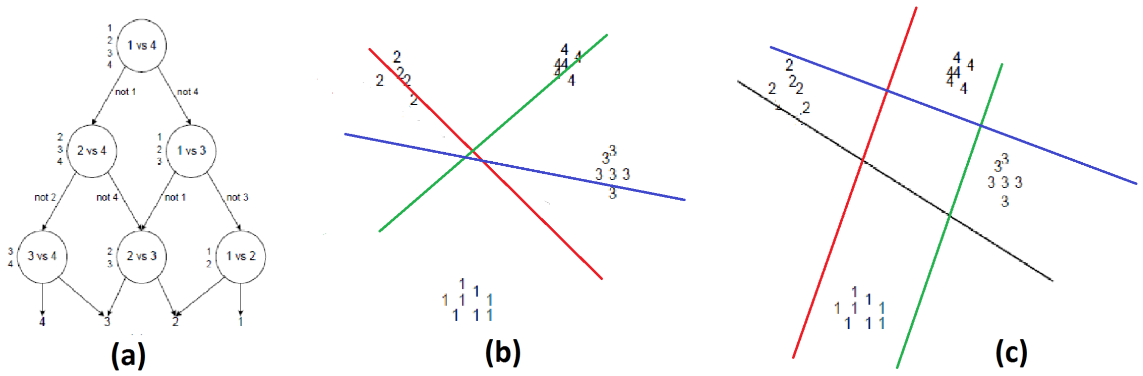


Figure 2.3 Multiple strategies for multi-class SVM. Figure (a) shows DAGSVM, (b) shows 1 Vs 1 and (c) shows 1 Vs rest strategy.

Apart from these methods, there are other algorithms like Crammer-Singer algorithm [41] which converts the multiclass problem into a single optimization problem rather than multiple classification problem.

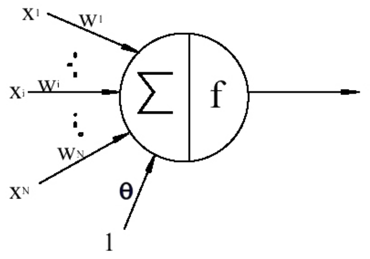
2.2.2 Multilayer Perceptron

An Artificial Neural Network is a system that is loosely based on the learning processes found in the neural networks of the human brain. The human brain consists of a large number of neural cells that have the ability to process information. Each cell consists of a cell body (Soma) that contains the cell nucleus. Connected to the cell body are dendrites for incoming information and a single long axon with dendrites for outgoing information that is passed to connected neurons. Information is transported between neurons along the dendrites in the form of an electrical potential. If the potential reaches a certain threshold, the neuron is said to be activated (fires) and the information is delivered along the neuron's axon to the dendrites, where it is passed on to other neurons.

Like the human brain, an artificial neural network is capable of processing information by means of connecting together relatively simple information processing units with links that allow each unit to communicate with each other by relatively simple signals. Each link has a numeric weight associate with it and is the primary means of long-term storage in the neural network. Weights are also updated during the learning process.

Figure 2.4 illustrates the typical structure of a unit:

As can be seen, an artificial neuron looks similar to a human neuron. Each node has one or more inputs and a single output. The neuron has two modes of operation: the training mode and the using or testing mode. In the training mode, the neuron is trained to respond (whether



$$x = \{x_1, x_2, \cdots, x_N\} \quad \textit{Inputvector}$$
$$\mathrm{w} = \{w_1, w_2, \cdots, w_N\} \quad \textit{Weightvector}$$
$$net = \sum_{i=1}^{N} w_i x_i \quad \textit{Actionpotential}$$
$$v = net - \theta$$
$$f(v) \quad \textit{Activationfunction}$$

Figure 2.4 Single node of an artificial neural network

it should fire or not) to a particular input of patterns. In the using or testing mode, when a taught pattern is received as the input, its associated output becomes the current output. If the input pattern does not belong to the taught list of input patterns, the neuron will then determine whether it will fire or not. Each input signal to the unit has an associated weight. The input information is processed by calculating the weighted sum of the inputs. If this summation exceeds a pre-set threshold value, the neuron will fire. The firing of any neuron is governed by its activation function. There are several activation functions like linear, tanh, logistic etc.

A simple multilayer perceptron has multiple layers with each layer having multiple nodes. Typically, a 3-layered MLP will have an input layer, a hidden layer and an output layer. These networks have fully connected nodes with no cycles. Since the network allows data to be transferred only in one direction, they are also known as feed forward neural networks. A sample MLP is shown in Figure 2.5. The units in a multilayer perceptron are arranged in layers, with connections feeding forward from one layer to the next. Input patterns are presented to the input layer, and the resulting unit activations are propagated through the hidden layers to the output layer. This process is known as the forward pass of the network. The units in the hidden layers have (typically nonlinear) activation functions that transform the summed activation arriving at the unit. Since the output of an MLP depends only on the current input, and not on any past or future inputs, MLPs are more suitable for pattern classification than for sequence labelling.

Multilayer perceptrons have a two pass system for training. In the first pass, known as forward pass, the synaptic weights remain unaltered throughout the network, and the function signals of the network are computed on a neuron-by-neuron basis. The output function is computed as

$$y_j(n) = \varphi(v_j(n))$$

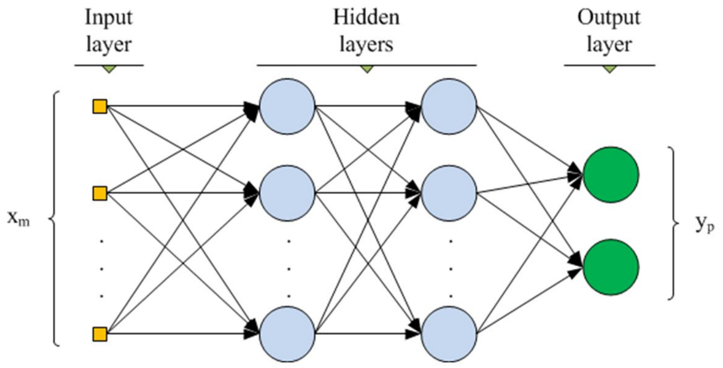


Figure 2.5 A sample multilayer perceptron

where $v_j(n)$ of neuron *j* is defined as

$$v_j(n) = \sum_{i=0}^{m} w_{ji}(n) y_i(n)$$

where *m* is the total number of inputs connected to node *j* and $w_{ji}(n)$ is the synaptic weight connected between nodes *i* and *j*. $y_i(n)$ is the input received at node *j*, which is also the output of node *i*. If node *j* is in the first hidden layer of the network, $m = m_0$ and the index i refers to the i^{th} input node of the network, for which we write

$$y_i(n) = x_i(n)$$

where $x_i(n)$ is the i^{th} element of the input vector. If, on the other hand, node *j* is on the output layer of the network $m = m_L$ and the index j refers to the j^{th} output node of the network, for which we write

$$y_j(n) = o_j(n)$$

where $o_j(n)$ is the j^{th} element of the output vector. This output is compared with the desired output $d_j(n)$ to obtain the error signal $e_j(n)$ for the j^{th} output node. This marks the end of forward pass which begins at the first hidden layer by presenting it with the input vector, and ending at the output layer after computing the error for each output node.

The backward pass, on the other hand starts at the output layer by passing the error signal back to the network, layer by layer, and recursively computing the local gradient for each node. This process is used to update the local weights of each node in each layer. The process continues till all weights in the network are updated. MLPs can also act as classifiers like SVM.

2.3 Word Recognition using HMM

A hidden Markov model (HMM) is a statistical Markov model in which the system being modelled is assumed to be a Markov process with unobserved (hidden) states. HMMs have been used for several applications like speech recognition, handwriting recognition etc. An HMM consists of the following elements:

- Number of states in the model (N). Even-though they are hidden, the state or set of states plays a significant role in the functioning of the model. We can reach a state from any other state in the model. The individual states (S) are defined as

  $$S = S_1, S_2....S_N$$

  For a given time t, the state is given as q_t.

- Number of distinct observation symbols per state (M). They correspond to the physical output of the system being modelled. They are denoted as

  $$V = V_1, V_2.....V_M$$

- The state transition probability distribution $A = a_{ij}$ where

  $$a_{ij} = P[q_{t+1} = S_j | q_t = S_i], 1 \leq i, j \leq N$$

- The observation symbol probability distribution in state j, $B = b_j(k)$, where

  $$b_j(k) = P[V_k^t | q_t = S_j, 1 \leq j \leq N; 1 \leq k \leq M$$

- The initial state distribution $\pi = \pi_i$, where

  $$\pi_i = P[q_1 = S_j, 1 \leq i \leq N$$

Given appropriate values of N, M, A, B and π, the HMM can be used to generate an observable sequence

$$O = O_1 O_2 ... O_N$$

where each observation O_t is one of the symbols from V and T is the number of observations in a sequence. The sequence can be generated using the following steps:

1. Choose an initial state $q_1 = S_i$ according to initial state distribution π

2. Set t=1
3. Choose $O_t = v_k$ according to the symbol probability distribution in state S_i.
4. Transit to a new state according to the state transition probability for S_i
5. Increment t by 1. Continue till $t < T$.

We use $\lambda = (A, B, \pi)$ to denote the complete parameter set of the model. More details into HMMs can be found in [44].

The problem of word recognition can be considered as a problem where we try to maximize $P(O|\lambda)$ given the model parameter λ. In step 1, for each word W of a vocabulary, a separate N-stage HMM will be generated. During training, we take all samples of input word W and build a model. We consider this as a problem of maximizing $P(O|\lambda)$. i.e: Given a set of observation sequences O and a model λ, how do we adjust the model parameters (A, B, π) such that $P(O|\lambda)$ is maximized.

The model which we have generated maynot be the best possible model. We would have to improve the model to generate better outputs. i.e: We need to choose the state sequence Q so that the entire model is in an optimal state. This is done activities like changing the model by introducing more states etc. Since we can never get a ”correct” state sequence, we use an optimality criterion to obtain the best possible state sequence. However, there could be multiple ”optimal” sequences possible. The most commonly used approach is to find the single best state sequence path i.e: to maximize $P(Q|O, \lambda$. We use a dynamic programming based method to find this best sequence and it is called Viterbi algorithm [13].

Once we have the final set of W optimized HMMs, whenever we get an input word to test, we sent it to all the models and take the output of the model which has given the highest likelihood. The most common way to calculate the probability of an observation O given model λ would be to enumerate every possible state sequence. However, such a process will take a very long time to compute. For this, we use a procedure called forward-backward procedure [43].

2.4 Word Recognition using Recurrent Neural Network (RNN)

Recurrent neural network is a type of feed forward neural network (FFNN) where there could be a connection from one of the forward layers to previous layer. i.e: The neurons have feed forward as well as feed back connections. The human brain is considered to be a recurrent

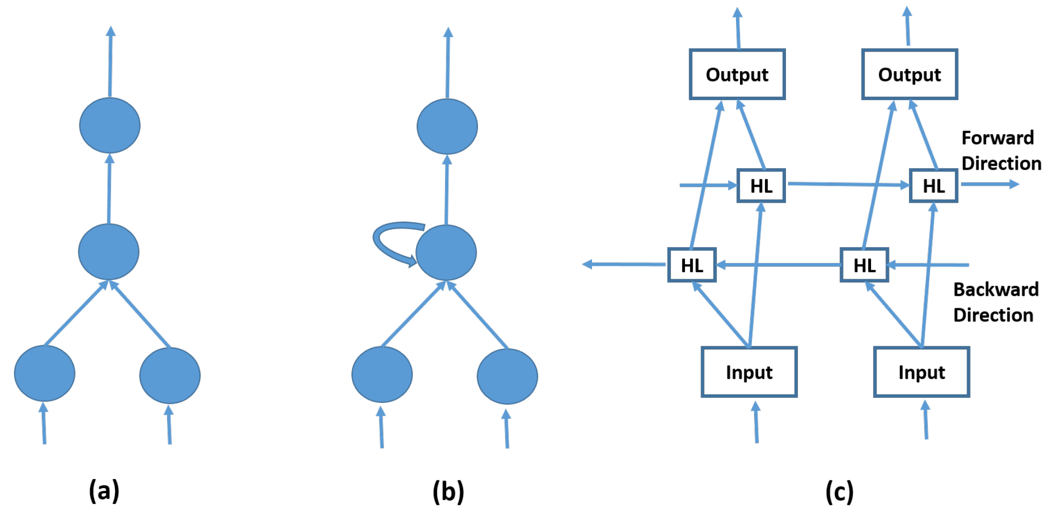


Figure 2.6 Figure (a) shows a sample MLP architecture while (b) shows a sample RNN, having a feedback connection in hidden layer. Figure (c) shows a bi-directional RNN, having ability to observe both past and future context.

neural network. Hence, there is considerable interest in using RNNs for several sequence classification tasks.

Consider the Figure 2.6. Figure 2.6(a) shows a simple FFNN having an input layer, a single hidden layer and an output layer. Figure 2.6(b) also shows a similar FFNN with the exception that the hidden layer has an additional feedback connection to itself. The introduction of such a feedback connection will help the network in identifying the state of the node before it had seen the present input. i.e: It can store the node's previous value in memory. Thus, the RNN have the ability to take into account the context in which the pattern was observed. This greatly increases the efficiency of the network as many pattern classification tasks like handwriting recognition works better if the larger context is known.

The above architecture can only observe the previous contexts. For effective use of context, it would be better if we can observe both past as well as future context. This was the motivation behind introducing a Bidirectional RNN (BRNN) [48]. Figure 2.6(c) shows such an architecture for a specific input at time *t* and *t+1*. BRNN have two separate hidden RNN layers, one for past context observation and another for future context observation. Both of them are connected to the same output layer. The biggest advantage of such an architecture is that without displacing the inputs from corresponding output targets, we obtain the past and future context.

Similar to MLP, RNN also have a forward and backward pass. In forward pass, the hidden layer activations arrive from both current external input and hidden layer activation one timestep before. Consider an input sequence *x* of length T as the input to RNN. The RNN will have I input nodes, H hidden nodes and O output nodes. Then, x_i^t will be the input value at time t for input i. Let y_j^t and z_j^t be the network input to node j at time t and activation output of node j at time t. Then, we define for hidden layers

$$y_h^t = \sum_{i=1}^{T} W_{ih} x_i^t + \sum_{h'=1}^{H} W_{h'h} z_{h'}^{t-1}$$

where

$$z_h^t = \varphi_h(y_h^t)$$

We can use the above two formulas to compute the activation of hidden nodes by varying time (t). For output layers, its input is given as

$$y_o^t = \sum_{h=1}^{H} w_{ho} z_h^t$$

The activation functions which are generally used is same as the ones which we use for MLP, like logistic sigmoid, softmax etc.

For backward pass, we use the well known Backpropagation through time (BPTT) [58] algorithm. Similar to regular back propagation, the objective function for RNNs depends on the influence from the output layer and also from the influence of hidden layer at the next timestep.

$$\delta_h^t = \varphi'(y_h^t) \left(\sum_{o=1}^{O} \delta_o^t w_{ho} + \sum_{h'=1}^{H} \delta_{h'}^{t+1} w_{hh'} \right)$$

where

$$\delta_j^t = \frac{\partial O}{\partial y_h^t}$$

where O is the objective function defined during training.

We can find all values of δ by starting at time t = T and reducing it by one timestep at a time.

2.5 Bidirectional Long Short Term Memory (BLSTM)

As discussed in the previous section, an important benefit of recurrent networks is their ability to use contextual information when mapping between input and output sequences. Unfortunately, for standard RNN architectures, the range of context that can be accessed is limited.

The problem is that the influence of a given input on the hidden layer, and therefore on the network output, either decays or blows up exponentially as it cycles around the network's recurrent connections. In practice this shortcoming (referred to in the literature as the *vanishing gradient* problem) makes it hard for an RNN to learn tasks containing delays of more than about 10 timesteps between relevant input and target events.

There have been several attempts to address this problem and one of the most effective solution was by using the Long Short Term Memory (LSTM) architecture [64]. Every LSTM memory node will have, apart from the input and output, three multiplicative gates known as the input, output and forget gate. The input gates decides when the current activation of the cell should be changed by taking input from network. Similarly, the output gate decide if the activation of the node should be propagated to network. The forget gate helps in resetting the activation value of the node. The use of these gates will help the LSTM system avoid the problem of vanishing gradient. i.e: If the input gate have an activation value near to 0, no new inputs will be available to the code from the network. This will ensure that the current activation value of cell can be made available to the network at a much later stage. Similarly, if the output gate has a low activation value, the present activation of cell is not released to the network.

Similar to bidirectional RNNs, we can use two hidden layers, containing LSTM nodes instead of normal nodes, to have a Bidirectional LSTM (BLSTM) architecture [12, 9]. Such an architecture helps us in recognizing large context from both past as well as future timestamps. Similar to RNNs, the LSTM networks too have forward and backward pass. Three additional parameters, namely α, β and γ are introduced which represent input, output and forget gates respectively. During forward pass, we have for:

- *Input Gate:*

  $$y_\alpha^t = \sum_{i=1}^{I} w_{i\alpha} x_i^t + \sum_{h=1}^{H} w_{h\alpha} z_h^{t-1} + \sum_{c=1}^{C} w_{c\alpha} s_c^{t-1}$$

  and

  $$z_\alpha^t = f(y_\alpha^t)$$

  where the subscripts c refers to one of the C memory cells. s_c^t is the state of cell c at time t (i.e. the activation of the linear cell unit). *f* is defined as the activation function of gates and *g* and *h* are the cell input and output activation functions.

- *Forget Gate:*

  $$y_\gamma^t = \sum_{i=1}^{I} w_{i\gamma} x_i^t + \sum_{h=1}^{H} w_{h\gamma} z_h^{t-1} + \sum_{c=1}^{C} w_{c\gamma} s_c^{t-1}$$

and

$$z_{\gamma}^{t} = f(y_{\gamma}^{t})$$

- *Output Gate:*

$$y_{\beta}^{t} = \sum_{i=1}^{I} w_{i\beta} x_{i}^{t} + \sum_{h=1}^{H} w_{h\beta} z_{h}^{t-1} + \sum_{c=1}^{C} w_{c\beta} s_{c}^{t-1}$$

and

$$z_{\beta}^{t} = f(y_{\beta}^{t})$$

- *Cell:*

$$y_{c}^{t} = \sum_{i=1}^{I} w_{ic} x_{i}^{t} + \sum_{h=1}^{H} w_{hc} z_{h}^{t-1}$$

and

$$s_{c}^{t} = z_{\gamma}^{t} s_{c}^{t-1} + z_{\alpha}^{t} g(y_{c}^{t}$$

- *Cell Output:*

$$z_{c}^{t} = z_{\beta}^{t} h(s_{c}^{t}$$

One major challenge in using RNNs for sequence classification is that we need to provide the training data in a pre-segmented format.i.e: We need to map the input feature vector to the corresponding output targets before training the network. This is because RNNs are trained to consider each classification as an independent event. These independently recognized labels must be joined together post-classification inorder to get the complete output sequence.

Since segmentation of words into corresponding classes is a challenging task which require lots of effort and language knowledge, we decided to use a temporal classifier known as Connectionist Temporal Classification (CTC) along with BLSTM [11]. Use of this layer removes the need for pre-segmenting the training data or post-processing classifier output to get the full sequence output. The basic idea is to interpret the network outputs as a probability distribution over all possible label sequences, conditioned on a given input sequence. Given this distribution, an objective function can be derived that directly maximises the probabilities of the correct labellings. Since the objective function is differentiable, the network can then be trained with standard backpropagation through time (BPTT).

We normalize the output activation functions in such a way that the result is one when they are summed up. This is then treated as probability vector of the characters present at that position. The output layer contains one node for each class label plus a special node ϵ , to indicate “no character”,i.e: no decision about a character can be made at that position. Thus, there are $K + 1$ nodes in the output layer, where K is the number of classes.

The CTC Objective function is defined as the negative log probability of the network correctly labelling the entire training set. For a given training set (S) consisting of pairs of input and target sequences (x,z), the objective function O can be expressed as

$$O = -\sum_{(x,z)\in S} \ln p(z|x).$$

One advantage of having such a discriminative objective function is that we can directly model the probability of the label sequence given the inputs. This has been proven better than an HMM based methods which are generative in nature [57]. Also an HMM based system assume that the probability of each observation depends only on the current state. On the other hand, a BLSTM system can easily model continues trajectories and the amount of contextual information available to such a network can in principle extend the entire input sequence.

2.6 Successful Applications of BLSTM

There are several applications of BLSTM system, mostly in areas which require long contextual information. Handwriting recognition has been one such area where high accuracies have eluded people for long time. However, LSTM networks have been used in recent past to achieve high accuracies [39, 10, 46]. LSTM networks have also been used in multiple handwriting recognition competitions also. Graves *et al.* won three competitions in connected handwriting recognition at the 2009 International Conference on Document Analysis and Recognition (ICDAR), without any prior knowledge about the three different languages (French, Arab, Farsi) to be learned. An interesting point to be noted here is that they used raw pixels (x,y coordinates) as features and still achieved excellent results. Generally, language specific features are used to boost the accuracies. Ciresan *et. al.* won the ICDAR 2011 offline Chinese handwriting recognition contest and again in 2013 ICDAR competition to recognize 3755 Chinese characters [28]. They also have the present lowest error rate in the MNIST dataset [29].

Similar results are also visible in word spotting problems also. Volkmar *et.al.* have used BLSTM for word spotting of handwritten words [31, 32]. Jain *et.al.* have used it for word spotting Devanagari printed words [59]. For OCR tasks, Breuel *et.al* have used LSTM networks to recognize English and Fraktur [70]. Adnan *et.al.* have used the network to recognize printed Urdu characters [8].

2.7 Summary

We looked at various recognition systems in this chapter. We started with symbol/character recognition systems and then towards holistic word recognition systems. Recurrent Neural Networks and their variants known as LSTM networks were analysed in detail. LSTM networks, specifically Bidirectional LSTM networks are found to be more efficient in long sequence classification tasks than many other solutions. When we use Connectionist Temporal Classifier along with BLSTM, the input sequences needn't be segmented and mapped to output labels before training. Such a system makes the job of word recognition even more efficient.

Chapter 3

Word Recognition in Indic Documents

Optical character recognition (OCR) systems for Indian languages have been around for quite some time. However, even the best available systems [15, 5] are unable to report accuracies comparable to their English counterparts. Complexity of the scripts, word-to-symbol segmentation, number of unique symbols (classes) etc. are some of the reasons for this.

Indic OCR systems have achieved considerable progress in the pre-classifier activities like binarization, word segmentation etc. In our recent experience in design of OCRs for Indian languages [15], we have observed that the errors are primarily caused by two modules. The first one, which segments the word into symbols, and the second one which generates valid UNICODE sequence from the classifier outputs. In this chapter, we look into ways to solve these challenges.

3.1 Solution 1: A Segmentation Free ILOCR

The need of a segmentation free OCR system is necessitated mainly due to the quality of Indic documents. Quality of ink, paper, issues related to font rendering, age of documents etc. have all lead to documents having lots of degradations in the form of cuts (single component split into multiple components) and merges (one or more components fused together to form a single components). Both these are challenges faced during isolated character recognition. This was our motivation in using a segmentation free recognition system. Figure 3.1 shows the architecture of our proposed system.

The words written in Devanagari script are connected by a head line, known as *shirorekha*. Vowel modifiers or *matras* can be made by combining a vowel with a consonant. The script also allows joining of multiple vowels and consonants to form a compound character. This results in having the number of unique characters anywhere between 300-800. This number is

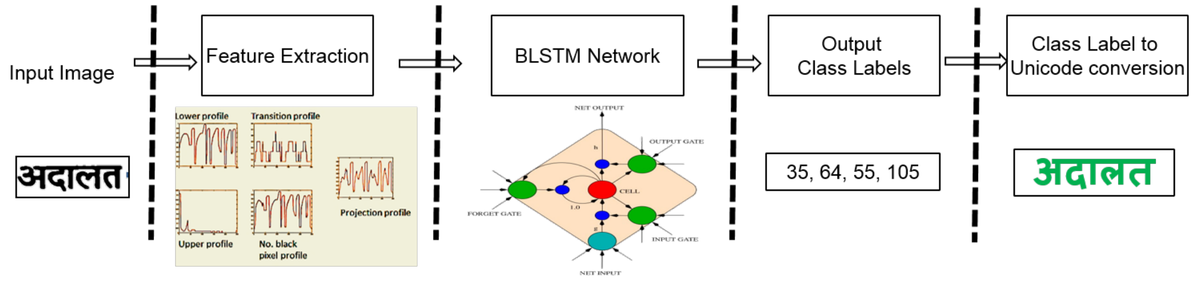


Figure 3.1 Our proposed segmentation free OCR accepts word images as input, extracts features of the word and recognize the entire word without segmenting it into symbols.

dependent on the font and the rendering scheme used. To reduce the number of unique classes, a commonly followed approach is to remove the *shirorekha* and recognize the characters and then recombine those at the end. This is referred to as zoning [18]. Our system is robust enough to handle the large number of classes and hence, we do not perform zoning of words. The number of unique class labels stand at 685 in our case. We assume the input to be a binarized word image.

Network Architecture

For recognition, we use a BLSTM network with a single hidden layer having 100 hidden nodes in it. During training, we extract five profile based features namely upper profile, lower profile, projection profile, transition profile and number of black pixels from every word image. More on the features will be discussed in later sections. We consider every *akshara* to be an output label. i.e: number of *aksharas* = number of class labels. Hence, for every word, we identify the *aksharas* present in them using a rule based system. This set of *aksharas* along with the extracted features are sent for training. Once the network is converged, the network configuration along with the network weights are stored to be used a model file during testing.

For testing, we take the word image, extract the features, and send it to the BLSTM recognizer for classifications. The classifier generates class labels for the word as output. This is shown in Figure 3.1. The output of BLSTM is a set of class labels (shown in the figure as 35, 64, 55, 105). Each class label corresponds to a unique *akshara*. These are intermediate results which must undergo some post processing to be converted to corresponding UNICODE. The post processing steps include re-ordering the class labels, assigning multiple UNICODE to a class label etc. This step is performed by using a rule based system to convert the output class labels to corresponding UNICODE characters.

3.2 Results and Limitations

Dataset: We obtain the word images needed for our experiments from multiple annotated Devanagari books. The ground truth is obtained from manual typing. The training dataset consists of nearly 90K word images and we used around 67K words for testing. The testing dataset was further split into Good and Poor datasets, based on the amount of degradation. Good dataset consists of words which have considerably less degradation upon random visual inspection. These are data on which a state-of-the-art OCR performs well. However, the poor dataset contains words with high percentage of degraded words and a traditional OCR system fails badly for these words.

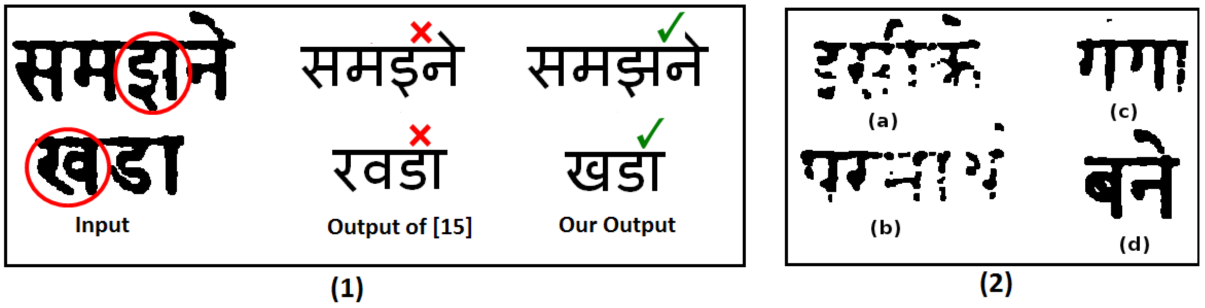


Figure 3.2 Figure showing some words which we successfully recognized (1) along with some words which our architecture failed to recognize correctly (2)

We test our method and compare the results against a state-of-the-art Indian Language OCR [15]. Some of the qualitative results are shown in Figure 3.2(1). The tick(green) marks indicate the word was recognized correctly while the cross(red) mark indicates that the word was recognized wrongly. For these examples, OCR is producing wrong outputs because of the cuts present in word image. The red circle shows the area where cut is present. The cut has resulted in converting a single valid character into another set of valid components, due to which the OCR fails.

	CER		**WER**	
Devanagari	**OCR [15]**	**Ours**	**OCR [15]**	**Ours**
Good	7.63	5.65	17.88	8.62
Poor	20.11	15.13	43.15	22.15

Table 3.1 Character error rate(CER) and Word error rates(WER) for Devanagari dataset. BLSTM based method is compared against a traditional OCR system. BLSTM based system shows considerable improvement in accuracy.

The quantitative evaluation of the entire dataset is done by comparing the labels returned from the neural network against the ground truth data to compute both Character Error Rate

(CER)[1] and Word Error Rate(WER)[1]. Our results are compared against current Devanagari OCR system [15]. The results are provided in Table 3.1. As shown in the table, we show considerable improvements in accuracy for good as well as bad dataset. The word accuracy has improved by more than 9% in case of good data and the improvement is more than 20% when it comes to poor data. The higher improvement in accuracy for bad data can be attributed to the fact that it is these data that require better understanding of context to be recognized correctly which is exactly what is being delivered by our method.

Figure 3.2(2) shows some examples where our method has failed. Most of the failures are due to heavy degradations, resulting in wrong classification. However, the classifier sometimes generated class label sequences which are impossible to occur as a UNICODE. e.g: two vowels occurring together or a consonant occurring in-between two vowels. Such errors result in the rule based system generating UNICODE sequences which, in practice, shall never occur.

3.3 Solution 2: OCR as Transcription

In the second approach, we model the problem of recognition as transcription. By this we mean that, given a sequence of feature vectors, the algorithm should be capable of generating the textual output. There are two important sub-problems to be solved for transcription. (i) Feature vectors need to be translated to corresponding UNICODES. (ii) the transposition that has occurred in the script needs to be compensated and a correct sequence needs to be learned.

There are many possible approaches for solving the first task i.e., of decoding the variable length stochastic sequences. For example, Hidden Markov Models (HMMs) can do this quite comfortably. This capability of HMM is exploited in the speech and handwriting recognition solutions. However, the second step make the problem very complex. This demands two additional characteristics for the algorithm: (i) It should have memory so that it can remember the previous observations and output the label at a later time instance. (i.e., the reversal of ordering is possible). (ii) Network is capable of looking into the future (non-causal) if there is a need.

Our preferred solution to the transcription problem is a recurrent neural network – Bidirectional Long-Short Term Memory (BLSTM) network. BLSTM networks have been successfully used in the past for both printed and handwritten text recognition tasks. The distinctive feature about these networks is their ability to remember long range context over several timesteps. The use of Connectionist Temporal Classification(CTC) Layer as the output layer allows the words to be presented as a sequence of unsegmented features thus doing away with the char-

[1]Evaluation metric is discussed in detail in Chapter 4

acter segmentation issues in Indian Languages. Further the bidirectional nature of the network makes it suitable for UNICODE level learning as it is capable of handling Unicode reordering issues prevalent in Indian scripts.These features make BLSTM a natural choice for developing a document image translation system for Indian languages. Further details into BLSTM architecture can be found in [39, 32, 31].

The task of transcription is often modelled by using Hidden Markov Models. Apart from the assumptions that such a model makes about a language, one major drawback for such an approach is that it does not really capture the context well. BLSTM neural networks are especially suited for transcription tasks for a variety of reasons. Further details into the role of context would be discussed later. Some attributes of the proposed BLSTM network are discussed below.

- **Memory cell:** Being a type of Recurrent neural networks, the BLSTM nodes are presented, along with the output, also their activation from the previous timestep. However, while in normal RNNs the new activation always overwrites the previous activation, thus constraining its memory, BLSTMs have a set of gates that regulate this. By training the weights connecting these gates to the recurrent input, the network can effectively trap almost arbitrary old input and thus do away with the vanishing gradients problem with normal RNNs.

- **Bidirectional:** BLSTM networks parse the sequence both from the front and the back. This allows them to use context on both sides before making any decision. Such information is vital in resolving Unicode reordering issues.

- **CTC Layer:** An important enhancement to BLSTM network is the CTC output layer. This layer essentially makes it unnecessary to segment the input feature based on target labels, a difficult and often buggy task. Further, such segmentation often requires domain knowledge.

The recognition solution presented in [62] for Hindi had a major limitation. It demanded the symbol definition and symbol level annotated ground truth data for training the network. i.e:the set of symbols present in the language must be known beforehand. The ground truth text for each word image should be represented using these set of symbols. However, there is no need to segment the word image to symbols. In contrast, in this work, we directly train at the UNICODE level. At UNICODE level, the network decoding problem is relatively easy. The method proposed in [62] was computationally intensive. It required more than 12 days to obtain a network with training accuracy less than 5%.

However, in our case, we are able to train the network in a couple of days time. This helped us in practically understanding the network and obtaining high performing recognizers for all the major Indian scripts. In the entire process, we innovate the training scheme by defining a stochastic gradient descent method. Such methods are found to be quite effective for scaling machine learning algorithms to large datasets as in the case of Pegasos [65].

3.4 Critical Look at the Architectiure

In this thesis, we argue that one can look at this problem from a different angle. We formulate the problem of OCR as an automatic transcribing task. We formulate this as that of learning a module which minimizes the transcription error (as well as the transposition error) directly. This formulation has multiple practical advantages. This method reduces the number of classes significantly for the Indian scripts. The number of labels that the recognition module need to output is limited to the space of UNICODEs for a language (which is around 150 for Devanagari). This is significantly less than the previous methods, which require more than 800 classes [57]. However, in this case, the output is already in the form which we require it to be. This demands a different type of classifier which is fairly small. Our method removes the need for a word-to-symbol segmentation as popularly done in the case of recognition methods based on Hidden Markov Models(HMM) [57] and Recurrent Neural Networks (RNN) [62]. Since the annotation required for this module is only at the word level, we bypass the requirement of strong annotation of symbols to design the classifiers. Note that the definition of symbols is not yet standard. We do not explicitly use this latent script definition at any stage of our solution. Since we directly generate UNICODE, our output is syntactically correct. Note that a significant percentage of errors in Indian language OCRs is visible as invalid UNICODE sequences (i.e: sequences that can never occur).

3.5 Why IL-OCRs tend to be Fragile?

Figure 3.3(a) depicts a schematic of a typical Devanagari OCR. Input word images are first segmented into symbols in module 1. However, this could become hard due to the definition of symbols, cuts and merges, and formatting/rendering issues in word-processors. Part of these issues (like cuts and merges) are common to almost all the OCRs. However, the similarities of sub-parts across symbols make the problem for Devanagari more challenging. Challenges in Module-3 are possibly more unique. The outputs of the isolated symbol classifiers need to be rearranged and processed to obtain a valid UNICODE sequence. A single error in isolated

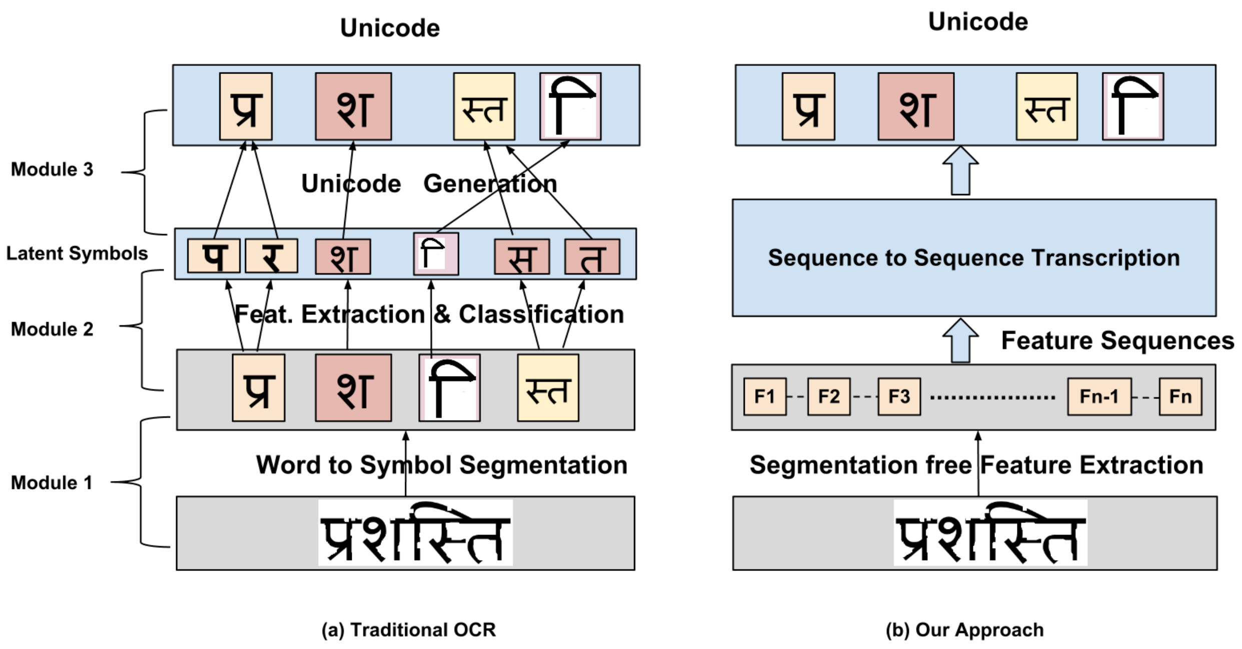


Figure 3.3 Overview of the challenge in Unicode based method. We take an image, recognize the individual components and then need to re-arrange and merge components to get the final Unicode output.

character classification can result in multiple errors in the UNICODE. Because of the specific manner in which Devanagari (and many other Indian languages) gets written, this conversion is not monotonic. i.e., later symbols could output earlier UNICODEs. (Note the intersecting lines in Module 3).

It is observed that, Module-1 and Module-3 contribute to most of the errors in the existing architectures. Module-2 is basically a trainable classifier (e.g. SVM), and is considered as a high-performance (in accuracy and in computational time) module for most scripts. In this thesis, we recast this problem in a nearly complementary manner in Fig. 3.3(b) where Module-1 gets converted to a segmentation-free feature sequence computation. This is less susceptible to the degradations. Similarly the Module-3 is avoided and the learned module is asked to directly output the UNICODE sequence. The harder part of this solution (Figure 3.3(b)) is the sequence to sequence transcription module which converts the feature sequence to a sequence of UNICODE. We model this as a learnable module which can learn this mapping from a set of examples.

Modelling recognition problem as translation and transcription are attempted in the past. Object detection was modelled as machine translation [56] by annotating image regions with words from a lexicon. Translation techniques have also been used by linguists [24] and speech recognition community [47] for modelling recognition. While translation techniques deals in conversion of input to output using specific rules, transcription involves mapping of input to a specified output and is considered as a simpler task. We consider our problem as that of transcription where we start with a set of word images features and obtain a corresponding UNICODE text.

3.6 Design and Training

3.6.1 Computational Complexity

As mentioned above, Indian scripts present a unique set of challenges. In our initial experiments, we faced issues with the BLSTM network taking prohibitively large time for convergence and we needed to investigate further as to find out the factors that lead to this behaviour. Therefore, we subjected the BLSTM network to a series of tests varying different dimensions of the problem to gauge its performance in terms of time taken for convergence. For this purpose we conducted a set of synthetic data experiments, varying the number of training samples, number of target labels and the average length of words in the document image. These were simulations.

Figure 3.4 and Figure 3.5 depict the time taken for convergence. The time taken increases with increase in the number of labels but decreases by using more features. This result has been one of our motivations for choosing UNICODE over class id labels. Number of UNICODE labels are often much smaller than the number of symbols in these languages. Based on a naive calculation, we were concerned of adding more features to the representation (because computations could at least linearly increase with dimensions). However, we found that with more descriptive features, training time reduces.

The plain vanilla BLSTM network is not without its own set of issues. The training time is notoriously large for such recurrent neural networks. A better generalization requires considerably large number of data for training. This added with number of features means that the training time can extend to days if not weeks. Some of the notable reasons for this isitautions are:

- Increase in number of training samples.
- Increase in number of target labels (output classes). Should that be symbols, UNICODE or *Aksharas*
- Increase in length of word. Many of the south Indian scripts have very high word length, measured in UNICODE.

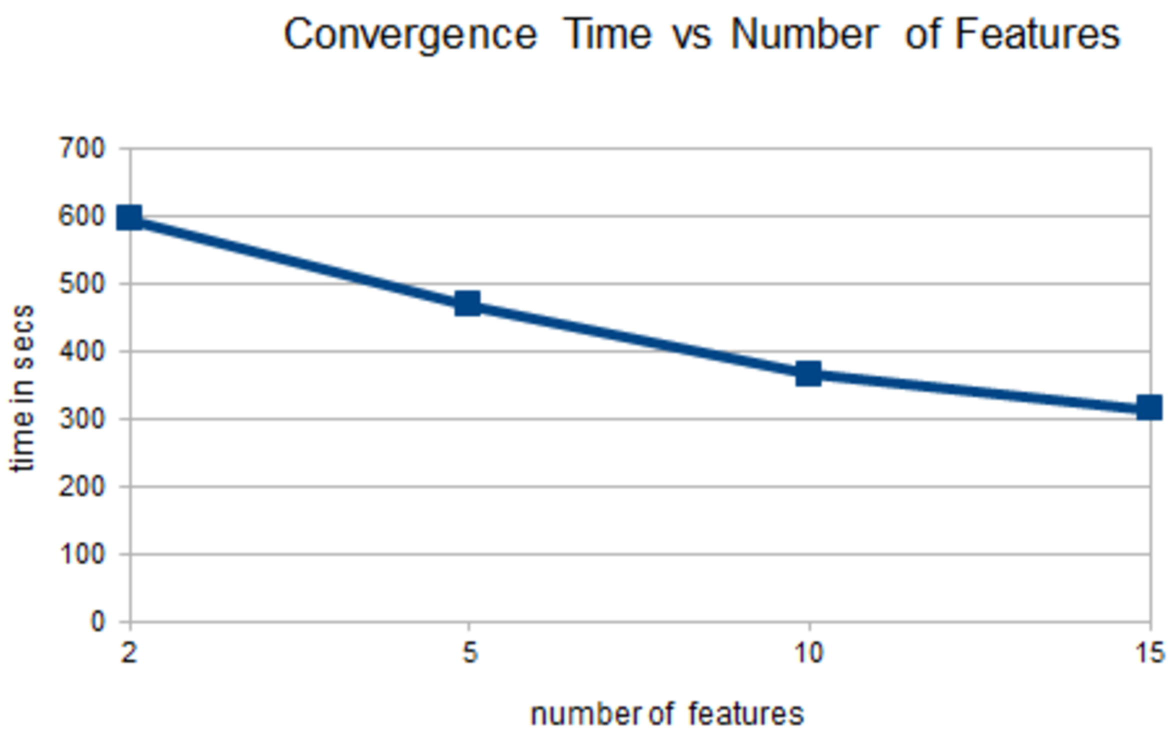


Figure 3.4 Even though the time per epoch increases if we increase the number of features (due to more computations per epoch), faster convergence leads to lower time overall

- Complexity of Scripts(eg: presence of '*samyuktakshars*' and '*matras*'). This forces the network to learn to output the same UNICODE under widely varying situations.

3.6.2 Learnable Transcription Module

The input to the transcription module is a feature sequence and output is a sequence of UNICODEs. Discriminative features play a significant role in classification. Profile based features have been used in past for recognition tasks [62, 67, 39]. For our experiments we have chosen 6 profile features, namely:

- Uppermost ink pixel
- Lowermost ink pixel
- Horizontal profile of ink pixels
- Mean value of ink pixels
- Number of ink-background transitions

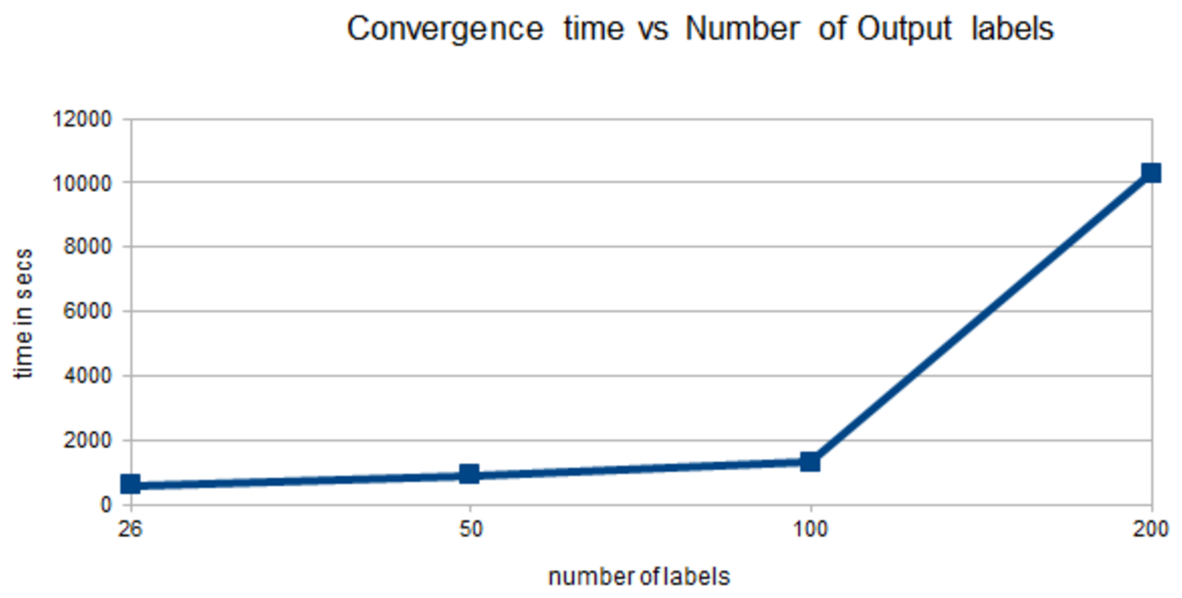


Figure 3.5 The number of labels seem to have a large negative effect on the convergence time

- Standard deviation among pixels.

More details into profile based features can be found in [71].

We use a sliding window to extract features from the word. A window size of 20 pixels with 75% overlap was chosen for each word. All the profiles are normalized with respect to the image height to make them lie between [0,1] which we use for training the transcription module. We also make the features more representative of the given image by dividing the image horizontally into two and computing the above mentioned features for both regions. This results in generating 12 features. This increase in number of features though seems insignificant, have increased the accuracy in practice. This is because there are many symbols which look similar but appear in different areas of word (eg: comma and quote). Splitting the word into two will help in differentiating such symbols.

Some of the features were extracted from grey images instead of binary images as binarization results in the loss of information. For features like top/bottom ink pixel, we applied binarization on the grey word image and extracted the features. A sample feature extracted is shown in Figure 3.6.

Inorder to understand the effect of number of features on accuracy, we performed an experiment. We took different number of features and tested them against different training sizes. Results are shown in Table 3.2. First row represents the percentage of data used for training. First column shows the number of features which were used for training. The table shows that as the number of features increase, the testing error drops.

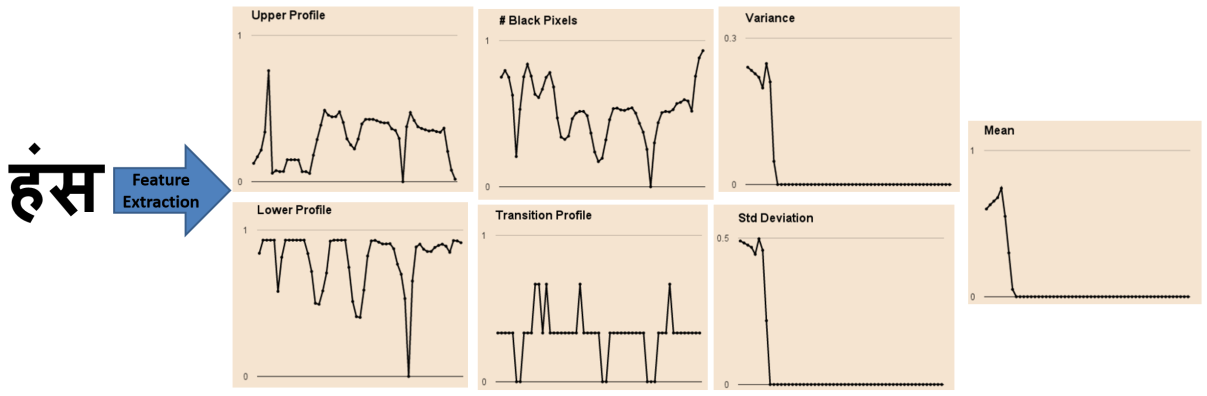


Figure 3.6 The above figure shows the 6 features which we extract from a word image. We divide the word into two based on height. Similar features are computed for both the top half as well as the bottom half for every word image resulting in a total of 12 features for every word.

Table 3.2 Effect of number of features and training size on a dataset. The testing error rate is shown on table below.

$\frac{\#Feat}{train\%}$	6	10	14
5%	16.89	14.22	12.34
10%	14.87	11.38	9.16
20%	10.39	8.36	6.82

3.7 Adaptation to Indic Scripts

Specific concerns relevant to Indic scripts had to be addressed before we could proceed with our experimentations. These included challenges like large number of target classes, significantly large training time and adaptation to newer fonts/styles etc. We address those challenges in this section.

3.7.1 Reduction and Standardization of Labels

Most Indian scripts are rich in script complexity with presence of '*matras*', '*samyuktakshars*', multiple representation of same character etc. Most of the character level recognition systems use a class ID based method where every symbol is given a unique class label. This results in an explosion of number of labels across languages. Languages like Telugu can have more than 350 unique class Id's. Also, use of class Id's make its generation a language-dependent process. Only someone with an understanding of the language can decide on the number of class Id's.

We wish to move away from this traditional process. We propose UNICODE representation as the standard way to represent symbols. Every symbol available in language is converted into one of the UNICODE codepoints. This helps in reducing the number of labels and also provides us with a standard way of representation. Table 3.3 shows the comparison of number of labels when we use UNICODE and class ID's for some Indian languages. As shown, the number of labels have comedown considerably in cases of Kannada, Telugu and Hindi, where the script is more complex. Note that the definition of a class ID can vary from person to person, and so the number of class ID will also vary.

One major challenge associated with this approach is that there will be multiple UNICODEs representing a single symbol. Also, in case of '*matras*', the network should be able to learn the rule associated with each matra. The matra can occur in both sides of a character, and learning such a rule is challenging for a network. Also, for '*samyuktakshars*', the mapping of multiple UNICODEs including the 'Zero-Width-Joiner' or '*Halanth*' to a symbol is to be learned.

3.7.2 Network Configuration

In this work, we analyzed the network performance on various parameter settings. A BLSTM network is mainly characterized by the number of hidden nodes (LSTM Size) it uses, number of hidden layers and the stopping criteria used for training. Table 3.4 shows the recognition rates for one particular language for various parameters. The time complexity for each

Table 3.3 Number of classes at UNICODE and Symbol Level. Telugu and Kannada are having very large number of class ID's.

Language	UNICODE	Symbols
Malayalam	184	215
Tamil	182	212
Telugu	171	359
Kannada	192	352
Hindi	185	612

Table 3.4 Effect of Changing Parameters to the Recognition Rates of the Network. Here we show the performance on the Malayalam language.

BLSTM Architecture		**Training**			**Test Error**
LSTM Size	**# Hidden Layers**	**Epochs Trained**	**Avg. Epoch Time (Hrs)**	**Train Error**	
50	1	70	1.3	2.04	2.2
50	2	63	1.4	0.90	1.31
50	3	47	1.9	0.81	1.12
100	1	38	2.4	1.24	1.6
100	2	18	5.06	0.97	1.25
200	1	10	9.37	1.84	2.08
200	2	5	16.89	1.49	1.61
300	1	4	17.64	2.67	2.67

epoch increases with increasing the LSTM size or the no. of hidden layers. We stop the network training when successive training error rates doesn't drop below a certain threshold. We found that increasing # hidden layers until 3 gave better results. The network with larger LSTM size resulted in poor generalization as can be seen from the table. The best results are obtained from the LSTM size 50 with 3 hidden layers.

The two other parameters which are commonly used in neural network training are learn rate and momentum. A higher learning rate might lead to faster results if the variability in training set is very less. However, most practical training data would have significant variation among the data and hence, we should carefully choose the value so that the value is neither too low, which would result in network converging very slowly, or value being set too large which can lead the network not to learn anything at all.

Similarly, momentum parameter is used to prevent the system from converging to a local minimum or saddle point. A high momentum parameter can also help to increase the speed of convergence of the system. However, setting the momentum parameter too high can create a risk of overshooting the minimum, which can cause the system to become unstable. A momentum coefficient that is too low cannot reliably avoid local minima, and can also slow down the training of the system.

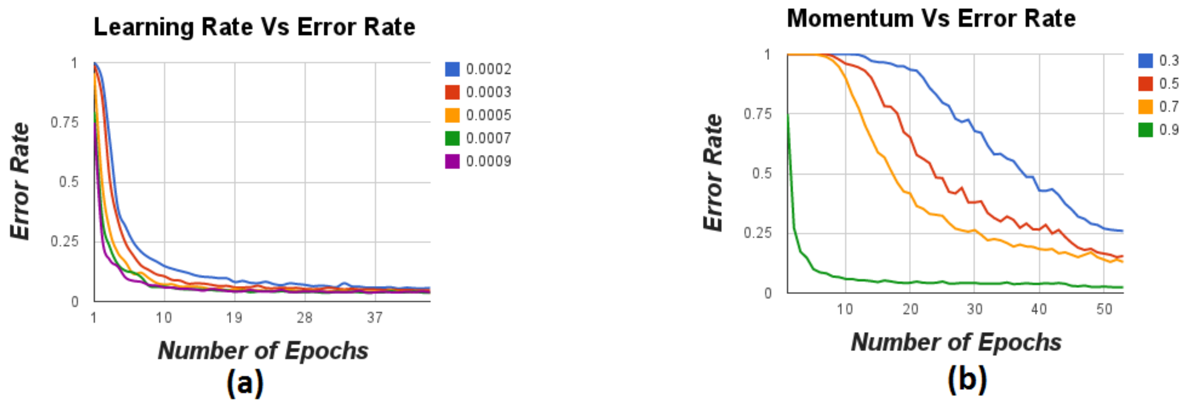


Figure 3.7 Graph showing how error rate varies with changes in learning rate (a) and with changes in momentum (b)

Figure 3.7 shows our results when we changed the learning rate and momentum. We used a small Malayalam corpus to perform these experiments to find out the optimal values. As shown in the graphs, changes made to learning rate do not affect the error rate much. However, changes in momentum have shown how quickly or slowly a system can converge. For our experiments, we have chosen 0.9 as the momentum value and 0.0009 as learning rate.

3.7.3 Training time reduction

Most of the RNNs are typically trained by Back Propagation through Time (BPTT). Being a gradient based method, it is susceptible to being stuck in a local minima. One popular technique to avoid this issue is to use Stochastic gradient descent [65] where the weight updates are based on only a subset of the complete training set. We found this to be extremely useful in our experiments. It was observed while dealing with large quantities of training data that it contains quite a few redundant samples. It seemed possible that a smaller subset of training samples should have the same information content in terms of training the network. Motivated by this we switched to using stochastic gradient descent by solving subsets of the training sets. After each epoch, a fixed number of examples were sampled with replacement. There is another way to view this strategy. The standard method of presenting data to the neural networks consists of doing it in terms of epochs. That is, all the training samples are presented one after the other and the process is repeated till convergence. In the other extreme we can do a random sampling where we sample one example after the other with replacement. In the first case the network is presented with the complete information about the data only in quanta of the complete training set. In the latter case, the network is presented with the same information in any given k examples, for a reasonable size of k. The strategy of sampling a subset of the examples after each epoch is somewhere in the between these two. We provide 20% of training data at a time for training in this mode. This method converges to 5% training error after 42 epochs, taking totally 126 minutes while the normal method takes 10 epoch totalling 190 minutes. We had compared the total time taken for training a dataset so that the training error reaches convergence. As argued, normal approach takes very less number of epochs, but the time taken per epoch is very high. While comparing with stochastic method, it takes nearly an hour more to converge. This difference becomes significant when the training data is very large. Although this method had very little contribution towards final testing accuracy, it helped in reducing the training time considerably and thereby allowing us to perform multiple experiments and obtain better accuracies.

3.8 Summary

In this chapter, we discussed the advantages of using word recognition system for Indic scripts. We proposed two solutions, both segmentation free approaches. While the first solution uses a rule based system to convert latent symbols to UNICODE, the second solution propose to consider the problem of OCR as that of transcription of input feature vectors to output UNICODE text. We proposed the use of BLSTM for implementing such a transcription system.

The features to be used, network parameters to be configured so that the system works best for Indic scripts was found out experimentally. In the next chapter, we test our system on different Indic scripts and compare our results against other state-of-the-art systems.

Chapter 4

Experiments and Results

Formal evaluation is necessary to validate any method and in this chapter, we evaluate our method using dataset from 7 Indian languages. We begin by providing an overview on how the dataset was created and then explain our evaluation parameters. We then proceed to show the experimental results for each of these languages followed by the analysis of results.

4.1 Dataset

4.1.1 Annotated Multilingual Dataset (AMD)

Availability of annotated dataset is an important requirement for performing any machine learning operations. Even though there are several such annotated corpus available for English, annotated content for Indic scripts is still very less. This has had significant impact in development of machine learning applications like OCR. Hence, a dataset containing annotated data for multiple Indic scripts was created by a consortium of Indian universities with the funding from Govt. of India.

We begin the process of annotation by scanning pages from all major languages in India. The scanning was done at multiple resolutions (200 dpi, 300 dpi and 600 dpi) and stored as TIFF file. The text was typed manually in UNICODE. We use a semi-automatic way to annotate the scanned images. The pages were first skew-corrected and binarized using some commonly available tools available in market. The different blocks in the scanned image (text, images, table etc.) was identified manually and was tagged accordingly. Then, the text blocks were segmented to lines and then further to words using a common segmentation method like ML-Segmentation. This is followed by aligning the annotated word boundary with corresponding UNICODE text. More details on this dataset can be found in [25].

Separate datasets for each languages was created for us to evaluate our method. Around 5000 pages from different books were chosen for all the languages which we plan to evaluate. The dataset consists of printed text of moderate degradation. It is commonly used by Indian research community as a common benchmark data. We refer to this dataset as Dataset 1 or AMD. Some of the statistics of the dataset is provided in Table 5.1.

Language	No. of Books	No. of Pages
Hindi	33	5000
Malayalam	31	5000
Tamil	23	5000
Kannada	27	5000
Telugu	28	5000
Gurumukhi	32	5000
Bengali	12	1700

Table 4.1 Dataset details which were used for our experiments

4.1.2 Annotated DLI Dataset (ADD)

We have also annotated 1000 pages from Hindi Digital Library of India (DLI) corpus. DLI is a collection of scanned pages in several languages which has been released to public. It is also a partner in the Million book Project. The quality of scanning is relatively poor for many languages. DLI contains only scanned pages and not the corresponding text. A sample comparison of DLI words with our AMD (Dataset 1) corpus is shown in Figure 4.1.

The Figure shows how the quality of a same word, present in the two different datasets vary in terms of quality. As seen, the quality of words is much poorer in DLI corpus than in AMD corpus.

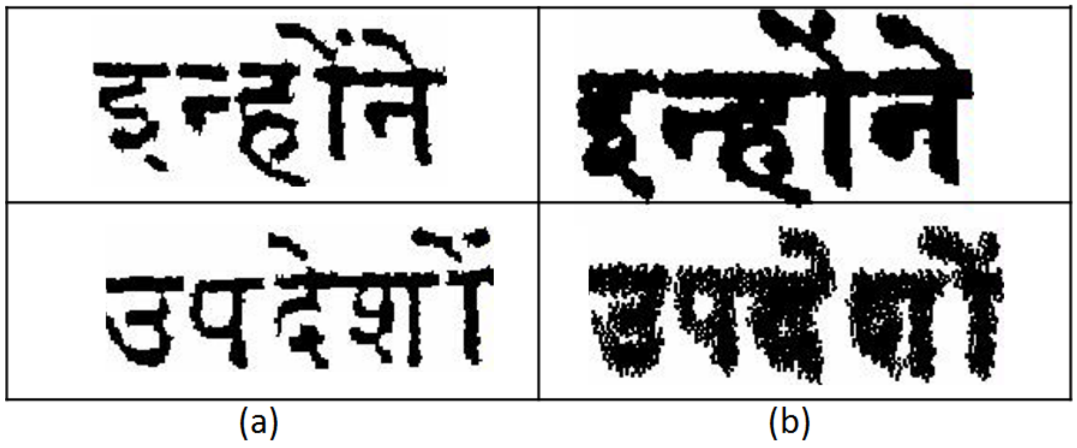


Figure 4.1 Example images from the datasets which we use. We show same images from the two datasets to show the difference in quality of images. Column (a) shows images from Dataset 1 (AMD) and column (b) has images from Dataset 2 (ADD). Dataset 2 consists of words having more degradation than Dataset 1

4.2 Evaluation Protocol

4.2.1 Evaluation Measures

For all experiments using our architecture, we expect the input data to be a word image along with its corresponding UNICODE ground truth. The word images will be grey scaled images and not binarized. The same features are used for all languages, and no script specific pre-processing like shiro-rekha (head-line) removal is performed. Once we get the output, it is evaluated against the corresponding UNICODE ground truth to compute the error. No dictionary/language model was used to improve the accuracy for our method. The output which we evaluate is the output of classifier.

For evaluating Tesseract [5] and Character OCR [15], we provide the grey page images as input and get the corresponding text of entire page as output. This page output is used for evaluation.

For our experiments, we have two main evaluation measures, namely character error rate and word error rate.

4.2.1.1 Character Error Rates (CER)

Character error rate is defined as the percentage of character (UNICODE) errors present in the testing set. To compute the UNICODE errors, we use Levenshtein distance method. Character error rate is computed using the formula :

$$C.E.R = (\frac{Total.Char.Errors}{Total.Characters}) * 100$$

Since we are measuring error, there is a possibility that total number of errors can be more than total number of characters present and the CER value could be more than 100%. This is because sometimes the recognizer recognises a single character as two characters which means the character error could be 2 (one substitution error and one insertion error) for a single character. This happens in cases where there are cuts present in a word or if there are any blobs or other noise present in word image.

4.2.1.2 Word Error Rates (WER)

Similar to CER, word errors can also be computed using Levenshtein distance where instead of comparing characters, we compare words. A word substitution error is one if it has atleast one character error. We define word error rate as:

$$W.E.R = (\frac{Total.Word.Errors}{Total.Words}) * 100$$

Similar to CER, WER can also have a value greater than 100%. A lower CER and a relatively high WER is an indication of presence of large number of word segmentation errors, i.e: error introduced while segmenting words from sentences.

4.3 Recognition Results

For our experiments, we used 80% of the data for training and tested the results on remaining 20% data. Our method took approximately 56 hours for training while running on a mid-level desktop PC having 16GB RAM and a 2.3GHz processor. During testing of our method, recognition of a page took 1.57 seconds on average, considering 250 words in a page. As mentioned earlier, we take input as word image. Hence, this time does not include any preprocessing like noise removal, skew-correction etc.

4.3.1 Malayalam

Malayalam, a Dravidian language, is spoken in the southern state of Kerala. Unlike Hindi, Dravidian languages do not have any head-lines connecting them. The symbols are more curved in nature and can be joined to form samyukthakshars. Though Malayalam has its own number representation, it is hardly used. Roman numerals are used in literature instead.

Malayalam has also undergone a formal script revision in the latter half of 20th century which resulted in introduction of several new symbols. However, till date, there are materials getting printed in both old-script and new-script. This has increased the complexity of recognition for Malayalam. There are not many Malayalam OCRs available at present. Most successful OCR for Malayalam at present is [15]. External plug-ins are available for Tesseract to enable Malayalam recognition.

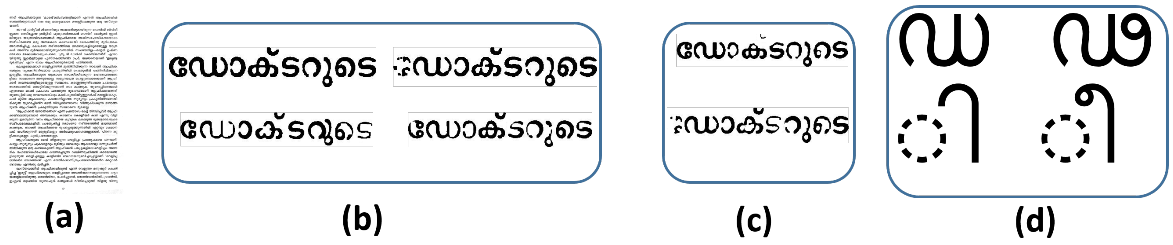


Figure 4.2 In above figure, (a) shows a sample page from our dataset (b) shows some words which we were able to recognize correctly (c) shows some words which was wrongly recognized by our method (d) shows some commonly confusing characters in Malayalam

Figure 4.3 shows a typical training scenario. In the Figure, X axis shows number of epochs and Y axis shows the training error rate. During initial epochs, when training error is high, the words are recognized wrongly. The network learns itself to correctly recognize the word and by the time training is over, it learns to correctly predict the input word.

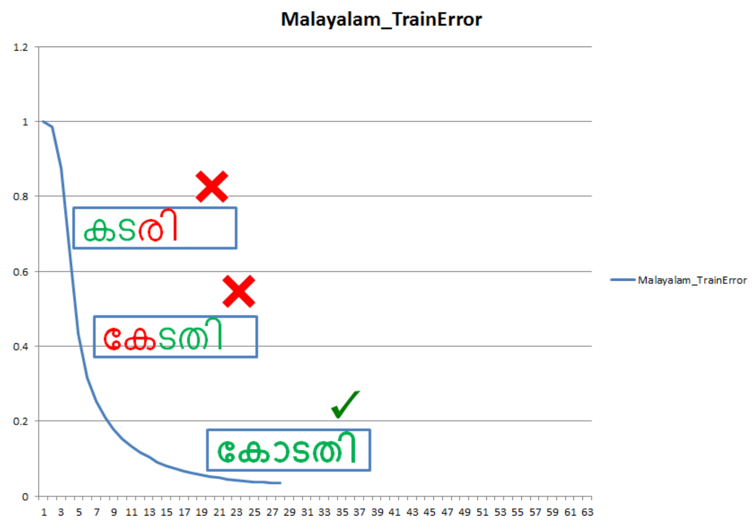


Figure 4.3 Typical training scenario where during initial epochs, words are recognized wrongly. During later stages, the network learns to recognize the word correctly.

For training, we have used the 5000 page annotated dataset and used 80% for training and 20% for testing. The results are shown in Table 4.2. It is quite clear from the table that the effectiveness of Tesseract in recognizing Malayalam is still very limited. We have an extremely high word accuracy, almost touching 90%.

	Character Error Rate(CER)			**Word Error Rate (WER)**		
Language	**Our Method**	**Char. OCR [15]**	**Tesseract [5]**	**Our Method**	**Char. OCR [15]**	**Tesseract [5]**
Malayalam	2.75	5.16	46.71	10.11	23.72	94.62

Table 4.2 Character and Word accuracy for Malayalam being compared against other state-of-the-art systems

Figure 4.2(b) shows some words which our method recognized correctly. There are several errors which have occurred due to *matras* and other similar looking symbols, as shown in Figure 4.2(d). The dataset which we used contain data written only in new script and how the system behaves when we introduce several other old-scripts characters would be an interesting experiment. Errors introduced due to UNICODE rearranging is a common feature in other character recognition systems. We have not yet observed any such re-ordering issues so far in our results.

4.3.2 Hindi

Hindi is written using Devanagari script and is the most commonly spoken language in India. It is a part of Indo-European family of languages. Hindi is spoken widely in northern and central India. Because of its huge popularity, there have been many commercial and popular OCR systems available for Hindi [5, 57, 22].

Hindi characters can be written in isolation or can be combined together to form *samyuktak-sharas*. The words are joined together forming a *shiro-rekha* or headline joining all symbols using a line drawn through top of the word. The end of line for Hindi is denoted using a *purna-viram* (|). Hindi also have its own numeric representation and is used commonly in literature.

For the purpose of evaluation, we used two annotated datasets. The first dataset consists of 5000 page corpus which emerged as a common benchmark data within Indian research community [25]. The pages are from Hindi books with considerably less degradations. This dataset is refereed as *Dataset 1*. We also use 1000 pages from the publicly available Digital Library of India. This corpus contains pages that are scanned and stored as binarized images.

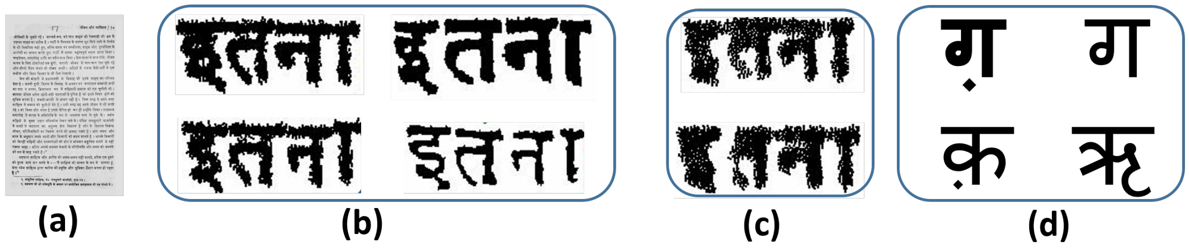


Figure 4.4 In above figure, (a) shows a sample page from our dataset (b) shows some words which we were able to recognize correctly (c) shows some words which was wrongly recognized by our method (d) shows some commonly confusing characters in Hindi

The pages are considerably degraded compared to dataset 1. We refer to this dataset as *Dataset 2*. A common approach in Hindi character recognition is to first remove the shiro-rekha from the word and then recognize the symbols. Our method, however, does not follow this approach.

	Char. Error Rate(CER)			**Word Error Rate (WER)**		
Dataset	**Our Method**	**Char. OCR [15]**	**Tesseract [5]**	**Our Method**	**Char. OCR [15]**	**Tesseract [5]**
Dataset 1	7.06	12.03	20.52	30.52	38.61	34.44
Dataset 2	12.31	53.6	38.2	40.49	83.36	57.9

Table 4.3 Character and Word accuracy for different Hindi corpus. We compare our results against other state-of-the-art OCR systems

We used 80% of our data for training and tested on remaining 20%. The results are shown in Table 4.3. Our method constantly out-performs other state-of-the-art OCR systems. Some of the qualitative results are shown in Figure 4.4(b). However, increase in degradation leads to failure in recognition of words, as shown in Figure 4.4(c).

Similarity in characters is another major factor for the failure of our method. Figure 4.4(d) shows some of the commonly confused characters from our dataset. We have also observed several issues in punctuation recognition and also recognizing *purna-viram*, which denotes the end of sentence , as exclamation character.

4.3.3 Telugu

Telugu is the official language of the state of Andhra Pradesh. One of the five classical languages of India, Telugu ranks third by the number of native speakers in India and thirteenth in the list of most-spoken languages worldwide. Negi *et. al* and C. Bhagavati have developed OCRs for Telugu [17, 15]. Like Malayalam, Telugu also have Tesseract support as a part of 3rd party plug-in.

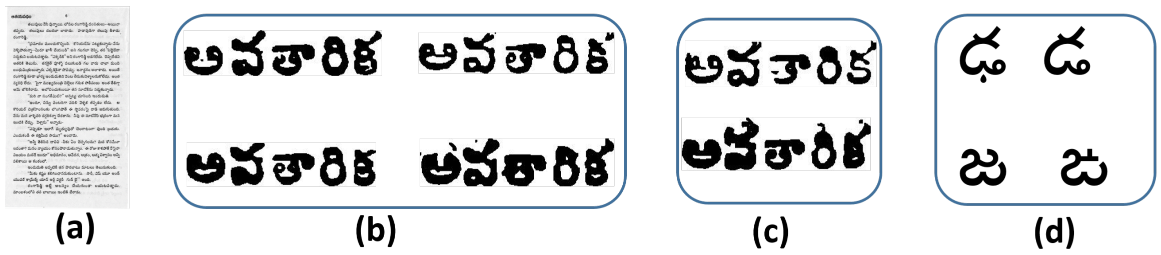


Figure 4.5 In above figure, (a) shows a sample page from our dataset (b) shows some words which we were able to recognize correctly (c) shows some words which was wrongly recognized by our method (d) shows some commonly confusing characters in Telugu

The results of our experiment on 5000 page dataset is shown in Table 4.4. The character OCR based on [15] is having a high error rate of 25% at character level. This might be due to occurrence of several similar looking symbols in Telugu. Also, presence of 'dots' as a part of symbol can cause confusion to a symbol classifier. A larger use of context can be useful in identifying such symbols. Our method shows considerable improvement against the character recognition system. The result is more noticeable in case of word error rate where we have just 16% error rate. Also, while comparing [15] and Tesseract [5], we can observe that even though the character error rate of Tesseract is much higher than [15], similar differences does not exist in word error rate. Multiple errors present in a single word might be the reason for this.

	Character Error Rate(CER)			**Word Error Rate (WER)**		
Language	**Our Method**	**Char. OCR [15]**	**Tesseract [5]**	**Our Method**	**Char. OCR [15]**	**Tesseract [5]**
Telugu	5.68	24.26	39.48	16.27	71.34	76.15

Table 4.4 Character and Word accuracy for Telugu being compared against other state-of-the-art systems

Some of the words which our method detected correctly and words which our method failed to recognize correctly are shown in Figure 4.5(b) and (c). The most commonly confusing classes are shown in Figure 4.5(d). As shown, the presence/absence of a dot below the symbol can change the meaning of the symbol. Larger contextual assistance have helped us in recognizing similar symbols.

4.3.4 Tamil

Tamil is a Dravidian language spoken predominantly by people of Tamil Nadu. Tamil is also a national language of Sri Lanka and an official language of Singapore. It is one of the 22 scheduled languages of India and was the first Indian language to be declared a classical language by the Government of India in 2004. Tamil script consists of 12 vowels and 18 consonants which can be combined together to form many other compound characters. Tamil has also unique symbol representation for 10, 100 and 1000 apart from regular numerals. It also have representations for some other commonly used words like day, month, year etc. Some of the successful OCR systems for Tamil include [14, 15].

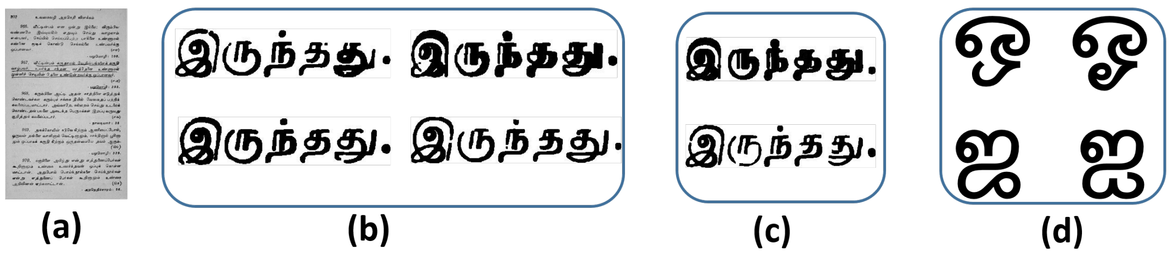


Figure 4.6 In above figure, (a) shows a sample page from our dataset (b) shows some words which we were able to recognize correctly (c) shows some words which was wrongly recognized by our method (d) shows some commonly confusing characters in Tamil

Table 4.5 shows the accuracy comparison between our approach, Tamil character OCR [15] and Tesseract. The tesseract implementation is not at a matured state for Tamil and is evident from more than 90% word error rate. Our method have significant reduction in character and word error rates while comparing with character OCR.

	Character Error Rate(CER)			**Word Error Rate (WER)**		
Language	**Our Method**	**Char. OCR [15]**	**Tesseract [5]**	**Our Method**	**Char. OCR [15]**	**Tesseract [5]**
Tamil	6.89	13.38	41.05	26.49	43.22	92.37

Table 4.5 Character and Word accuracy for Tamil being compared against other state-of-the-art systems

We also show some qualitative results in Figure 4.6(b). Figure 4.6(c) shows two examples where our method failed to recognize the word. Two highly confusing characters are shown in Figure 4.6(d).

4.3.5 Kannada

The southern state of Karnataka has Kannada as its official language. Like all other south Indian languages, Kannada too is part of Dravidian language family. It is recognized as a classical language by Government of India and is one of the official languages of government. Kannada has its own numeric system and symbols can be joined together to form samyuktaksharas (combined characters). The scripts of Kannada and Telugu are derived from common ancestor and hence have lots of similarities. There have been several works in recognition of Kannada scripts. More details can be found in [69, 16, 15]. Tesseract provides a Kannada plug-in but the system is not in a matured state yet.

	Character Error Rate(CER)		Word Error Rate (WER)	
Language	Our Method	Char. OCR [15]	Our Method	Char. OCR [15]
Kannada	6.41	16.13	23.83	48.63

Table 4.6 Character and Word accuracy for Kannada being compared against other state-of-the-art systems

We show the character and word error rates for Kannada in Table 4.6. Tesseract for Kannada had poor accuracies and hence has not been included in the table. We report a reduction of around 25% error rates for Kannada using our method. We have used the 5000 page annotated dataset with 80-20 split for training and testing.

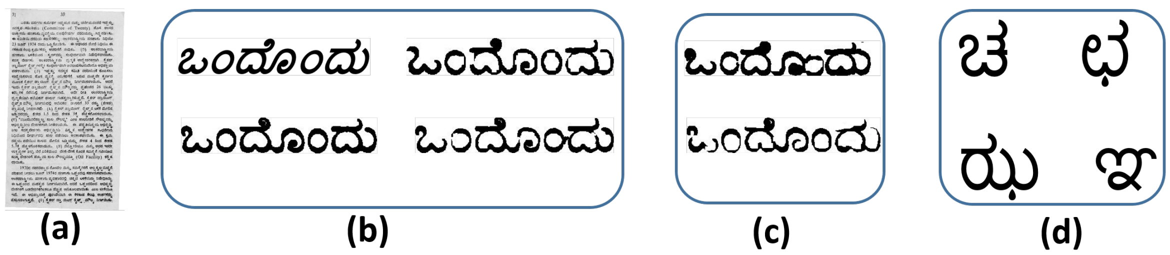


Figure 4.7 In above figure, (a) shows a sample page from our dataset (b) shows some words which we were able to recognize correctly (c) shows some words which was wrongly recognized by our method (d) shows some commonly confusing characters in Kannada

The qualitative results are shown in Figure 4.7. (a) shows a sample page from our dataset while (b) shows some words which our method was able to recognize correctly. As with above languages, we have taken the same word and showed different instances of it getting recognized correctly and wrongly. The commonly confusing classes are shown in Figure (d).

4.3.6 Gurumukhi

Gurumukhi is the script which is used to write Punjabi. It is used mainly in the northern state of Punjab where it is the official language. Gurumukhi is part of the Indo-European family of languages and is spoken widely in many parts of northern India, Pakistan, Canada and England and Wales making it the 9th most widely spoken language in the world. Like Hindi, Gurumukhi words too have a shiro-rekha connecting all symbols of the word. A popular OCR system for Gurumukhi is developed by Lehal and Singh [35].

	Character Error Rate(CER)		Word Error Rate (WER)	
Language	**Our Method**	**Char. OCR [15]**	**Our Method**	**Char. OCR [15]**
Gurumukhi	5.21	5.58	13.65	25.72

Table 4.7 Character and Word accuracy for Gurumukhi being compared against other state-of-the-art systems

The evaluation results of Gurumukhi using our OCR and character OCR is shown in Table 4.7. Since the tesseract plugin for Gurumukhi was not matured enough, we decided not to include them in our results. The character level accuracies of our method and character OCR is almost same for Gurumukhi. However, the word level accuracies are much different. Our method has an error reduction of around 12%. One possible reason for such an improvement might be that for our system, the number of characters that are wrong in a word would be much higher than for the character OCR. In other words, the character errors are concentrated on fewer set of words.

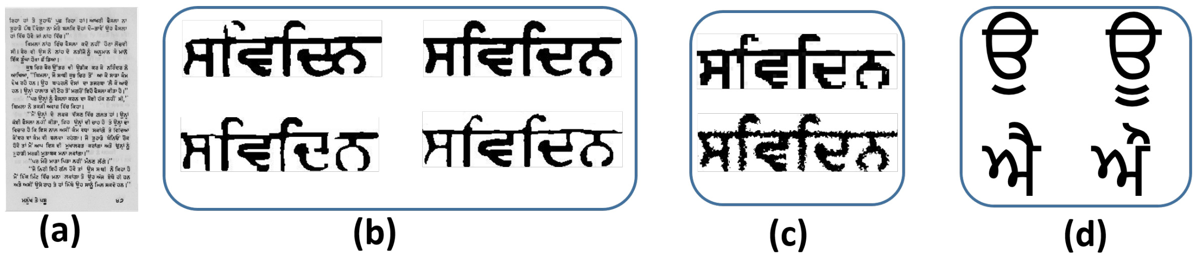


Figure 4.8 In above figure, (a) shows a sample page from our dataset (b) shows some words which we were able to recognize correctly (c) shows some words which was wrongly recognized by our method (d) shows some commonly confusing characters in Gurumukhi

Some of the qualitative results are shown in Figure 4.8. The figure shows the words which our system was able to recognize correctly and some words where our system failed along with couple of confusing classes.

4.3.7 Bangla

Bangla or Bengali is a part of Indo-European family of languages and is spoken commonly in eastern India. It is the official language of the state of West Bengal and also of Bangladesh. The national anthem of both India and Bangladesh is written in Bangla. It is the 7th most spoken language in the world. There are several works on developing OCR for Bangla [18, 19]. Tesseract also supports Bangla as part of its main distribution and not as an independent plug-in.

	Character Error Rate(CER)			**Word Error Rate (WER)**		
Language	**Our Method**	**Char. OCR [15]**	**Tesseract [5]**	**Our Method**	**Char. OCR [15]**	**Tesseract [5]**
Bangla	6.71	5.24	53.02	21.68	24.19	84.86

Table 4.8 Character and Word accuracy for Bangla being compared against other state-of-the-art systems

Table 4.8 shows the result of comparison between our OCR and Bangla character OCR. While Bangla character OCR shows better results than ours at character level, our word recognition results are better. The tesseract for Bangla is having a much poorer accuracy than the other two systems.

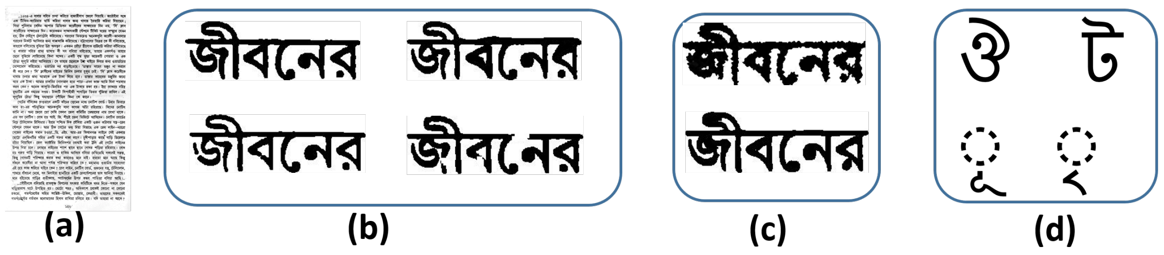


Figure 4.9 In above figure, (a) shows a sample page from our dataset (b) shows some words which we were able to recognize correctly (c) shows some words which was wrongly recognized by our method (d) shows some commonly confusing characters in Bangla

The qualitative results for Bangla is shown in Figure 4.9.

4.3.8 Discussions

We consolidate the results mentioned in previous sections and present them in Table 4.9 and Figure 4.12. We then proceed to stretch our system by trying to identify how much degradation can it handle. We took a word and introduced degradations to it. After every round of degradation, the word was sent back to the recognizer for recognition. This step was repeated till the recognizer failed to recognize the word. The results of this experiment is shown in Figure 4.10. As shown, the recognition architecture is pretty robust towards degradations. Only when the degradations become so severe that it is tough to recognize the words did the recognition fail.

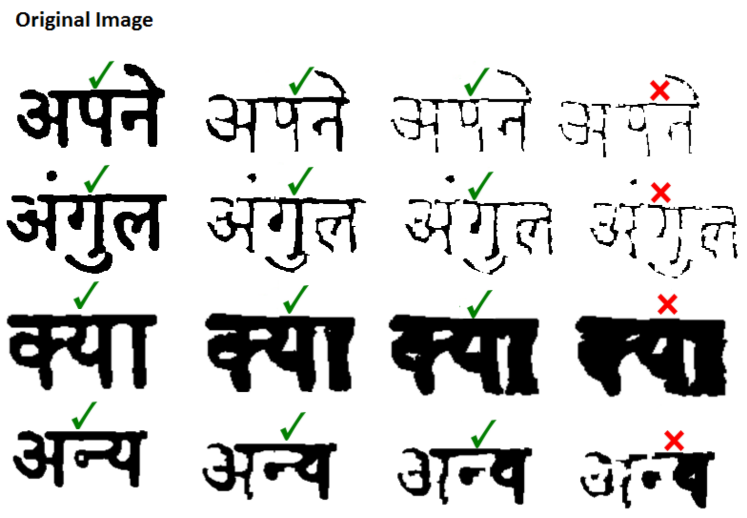


Figure 4.10 A sample word image was taken and subjected to increasing levels of degradation. The words are sent to classifier till the recognition failed.

A significant reason for improvement in accuracies after using BLSTM network is its ability to store context, both past as well as future. The image shown in Figure 4.11 is a typical example that occurs in Hindi. Figure 4.11 (a) and (b) shows two different word containing similar looking segments (highlighted area in words). While the segment shown in (a) consists of two Unicodes, (b) image segment is infact representing a single Unicode. This is because, (b) segment is actually a single glyph but to to degradation or font rendering, it was split into two and could be confused with the segment in (a). Such cases can be recognized only by considering a larger context for recognition. Availability of memory is vital in such cases where past and/or future context can be referred to decide upon the value of present sequence. This is shown in Figure 4.11(c) where we show a common symbol and the associated Unicode for it. While generating Unicode, we need to place them in the proper order so that the text-

	Character Error Rate(CER)			Word Error Rate (WER)		
Language	**Our Method**	**Char. OCR [15]**	**Tesseract [5]**	**Our Method**	**Char. OCR [15]**	**Tesseract [5]**
Hindi	6.38	12.0	20.52	25.39	38.61	34.44
Malayalam	2.75	5.16	46.71	10.11	23.72	94.62
Tamil	6.89	13.38	41.05	26.49	43.22	92.37
Telugu	5.68	24.26	39.48	16.27	71.34	76.15
Kannada	6.41	16.13	-	23.83	48.63	-
Bangla	6.71	5.24	53.02	21.68	24.19	84.86
Gurumukhi	5.21	5.58	-	13.65	25.72	-

Table 4.9 Character and Word accuracy for different language corpus. We compare our results against other state-of-the-art OCR systems

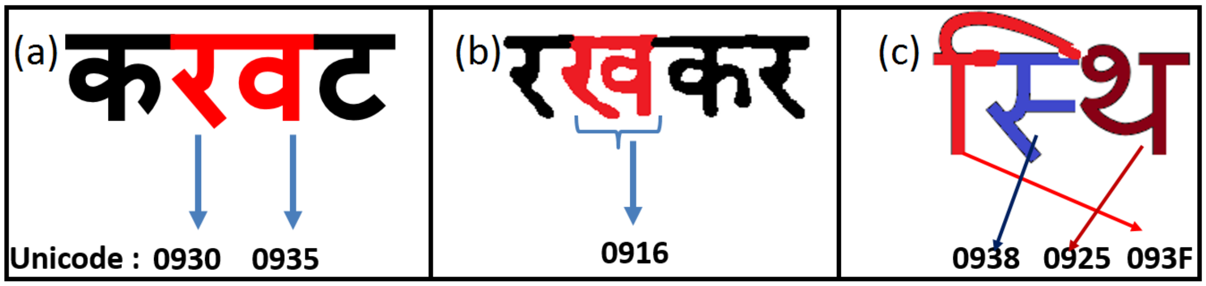


Figure 4.11 The above example shows the significance of context. Two different words with same sub-image is shown with (a) representing two Unicodes and (b) representing a single Unicode. The reason being (b) sub-image should have been joined together but due to degradation, they have been separated. Such words can be detected by considering the larger context

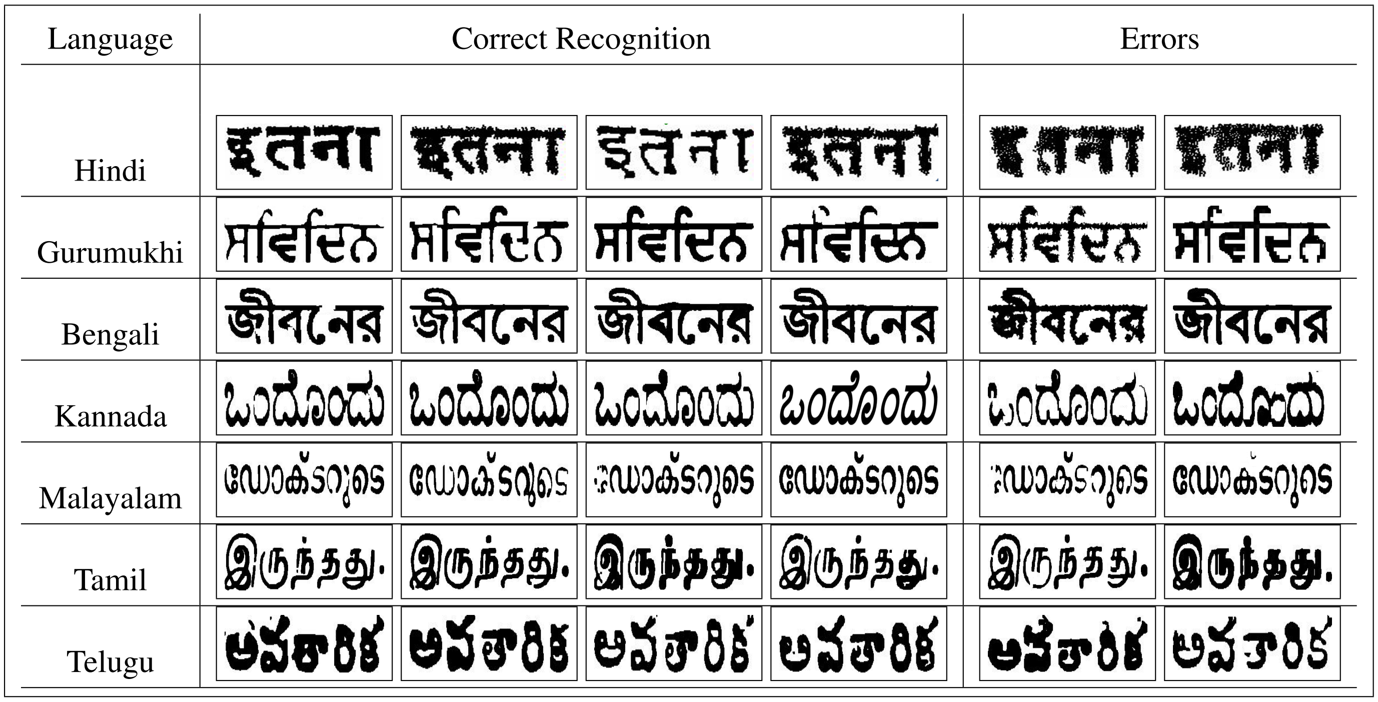

Language
Correct Recognition
Errors
Hindi
Gurumukhi
Bengali
Kannada
Malayalam
Tamil
Telugu

Figure 4.12 Success and failure examples from different languages are shown in above figure. Different samples of same word has been shown in above figure.

processors can render them correctly. So, eventhough symbol 1 (in red) appears first in image, it has to be placed in the end. Such reordering requires the presence of memory so that the future information can be referred to decide the output of present sequence.

Another important requirement for our system was to ensure that UNICODE re-arranging was happening, and that the system should be able to detect them automatically while generating the output. From the outputs, we looked for specific instances where our method solved this problem. Figure 4.13 shows some examples of words whose Unicode outputs needs to be re-arranged to be rendered correctly.

Even-though we were able to achieve high accuracy for all major Indian languages, there were still issues which existed. Degradation is one major challenge which we need to address. Using better features would help us in solving the issue to a certain extend. However, we need to look towards better pre-processing and image enhancement techniques to achieve significant improvement while dealing with heavy degradation. Another significant challenge which we encountered was in the case of special characters, especially small symbols like comma, quotes, fullstops etc. Many a time such symbols were recognized wrongly. Designing a method which specifically tackles special character recognition and coupling it with our existing architecture should be able to bring such errors down.

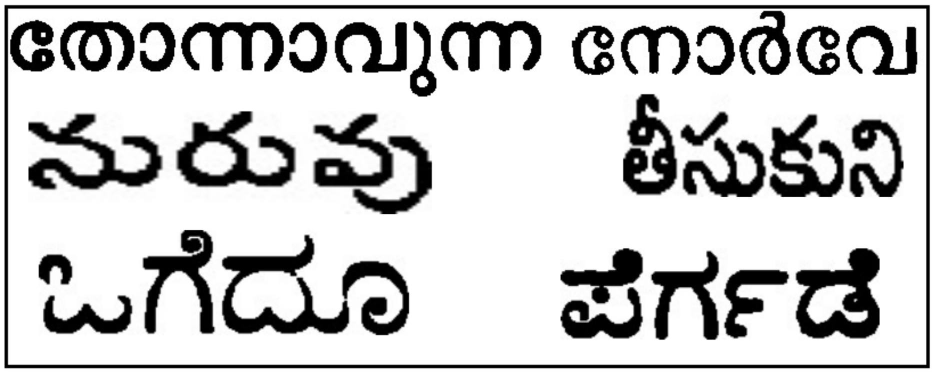

തോന്നാവുന്ന നോർവേ
తీసుకుని
ಒಗೆದೂ ಪೆರ್ಗಡೆ

Figure 4.13 Above figure shows some examples where rearranging of Unicode symbols is required to generate the correct output. Our system was successfully able to recognize these words.

4.4 Summary

In this chapter, we validate our method by evaluating it using large Indian language corpora. We compute character and word error rates and compare it against other state-of-the-art systems available. We also qualitatively show the capabilities of our system w.r.t its ability to handle degradations. UNICODE rearranging, a major challenge especially in Indic scripts, has also been handled using our system. The effect of context in recognition, a major factor in boosting our accuracies, was also discussed in this chapter.

Chapter 5

Error Detection in Indian Scripts

5.1 Introduction

The ability to automatically detect and correct errors in OCR output has been an age old problem. Traditional methods favour using dictionary based models for error detection and correction. For example, Kukich [42] contains several methods in tackling this problem. Such methods tag a word as valid/invalid based on its presence/absence in a dictionary. The effectiveness of such methods largely depends on the size of the dictionary and the size of text corpus from which the dictionary was built. Moreover, such simple methods fail in effectively handling "out of dictionary" words. This limitation of dictionary based method resulted in the popularity of statistical language model (SLMs) for error detection.

In the past, most of the studies in error detection [55, 76] have focussed on English or very few Latin languages like German. In 1992, Kukich [42] performed experimental analysis with merely few thousands of words, while the methods discussed in 2011 by Smith [68] use a corpus as large as 100 Billion words. This growth in linguistic resources has to be attributed to the rapid proliferation of Internet, huge content (amount of text) that has become available publicly, and the increase in computing and storage powers of modern computers. Most of these studies did not have enough impact on many other languages where building and managing large dictionaries were considered to be impractical.

Although efforts have been present in developing post-processing techniques for such inflectional languages [63, 66], it has been perceived and argued that most of the attempts in English have no direct impact on such languages. There are many valid reasons for these arguments: (a) the electronic resources available for many of these languages is very small (eg. the best Telugu corpus is around 4 million words while that of the popularly used English corpus is around 100 Billion words). (b) language processing modules (like a morphological analyser, synthesizer) are still being developed and refined. Though there exist morph analysers for many of the Indian languages, the efficiency of those analysers is nowhere close to what we have for English. They are not possibly ready to be used for error correction in OCR. (c) The unique word lists to get any meaningful results are too high to work with. We broadly agree with most of these observations. However, with significant increase in availability of newly created content for many of these languages on Internet, one can now have better estimate of the complexity of the problem. This led us in revisiting the problem.

The closest to error detection is the literature on (i) post-processor design in OCRs and (ii) spelling correction in word processing. There have been several previous works in this area. Parker *et al.* [51] used hidden markov models (HMMs) in detecting OCR errors while Golding *et al.* [37] have combined part of speech (POS) trigrams with Bayesian methods for error detection. In a recent work, Smith [68] has used a combination of classifier confidence and word nGrams in automatically detecting and correcting word errors. Error detection on inflectional languages have also been attempted before [27, 54]. In this work, we show that by properly adapting the statistical techniques, one can obtain superior performance to those reported in [27, 54] for far more complex languages. Commonly available spell checkers like GNU Aspell are also limited by the small dictionaries they use, and primitive error detection schemes. In general, SLM based methods have shown considerable success in solving this issue. In this thesis, we try to understand why the problem is harder for Indic scripts from a statistical point of view. We comment on possible direction to design effective error detection and correction schemes. We also propose a method for error detection by learning error patterns from a set of examples using a SVM classifier.

5.2 Error Detection: Why is it Hard?

Inflectional languages have an extremely large vocabulary. Words can take forms due to the gender, sense or other contextual factors. Inflectional languages have looked at the problem of understanding words, by building word morphology analysers. However, most morphology analysers assume that the words are correct. One of our primary focus is to work with words which are erroneous, and sometime invalid. Inflectional languages are also morphologically rich and agglutinative in nature with complex structures. Besides, there are many morphophonemic changes in the word formation process. i.e: two or more valid words can be combined to form another valid word. This issue is explained with an example in Figure 5.1(b). For Indian languages, the basic unit to work with is an *akshara*, which is a set of one or more consonants and a vowel. This imply that multiple Unicodes can form a single akshara. We demonstrate this formation in Fig. 5.1(a). An akshara could have 5 or 6 Unciodes in extreme cases.

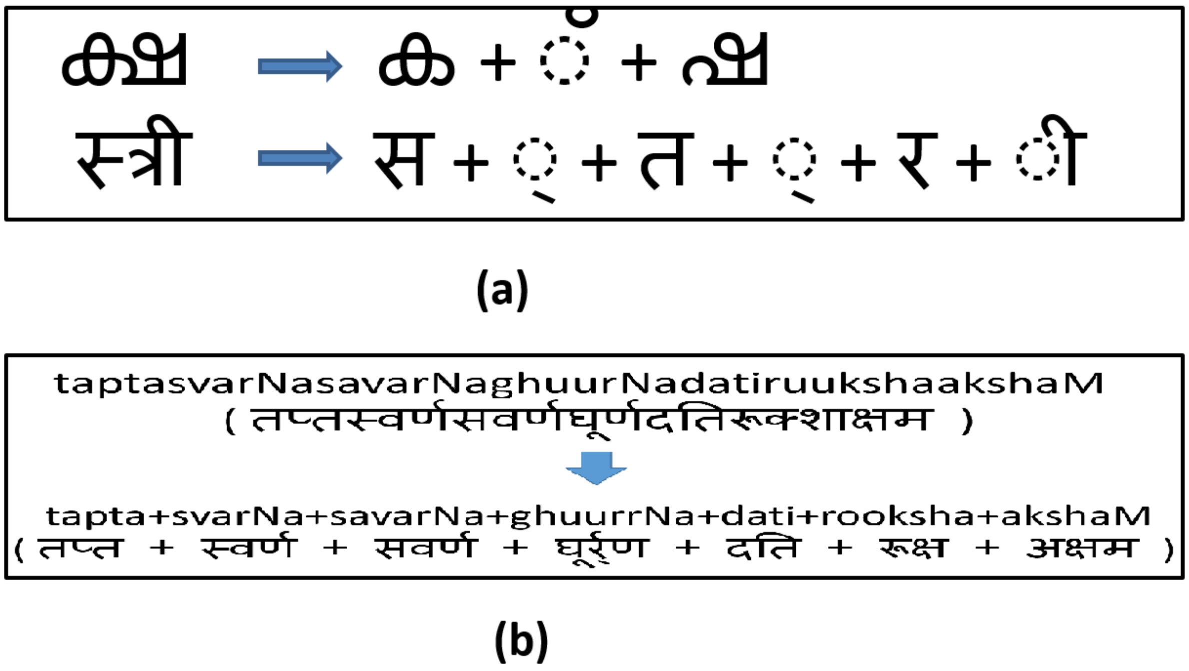

ക്ഷ → ക + ് + ഷ
स्त्री → स + ् + त + ् + र + ी
(a)
taptasvarNasavarNaghuurNadatiruukshaakshaM
(तप्तस्वर्णसवर्णघूर्णदतिरूक्शाक्षम)
tapta+svarNa+savarNa+ghuurrNa+dati+rooksha+akshaM
(तप्त + स्वर्ण + सवर्ण + घूर्ण + दति + रूक्ष + अक्षम)
(b)

Figure 5.1 First image shows some sample *aksharas* and the Unicodes associated with them. As shown, there are many cases where a single akshara is formed using multiple Unicode characters, unlike English. Second figure shows a single valid word formed using combination of multiple other words.

5.2.1 Size and Coverage

We first look at the coverage of the language. Table 5.1 presents the statistics of different languages. We have considered 5 Indic scripts for our analysis and compared it with English. We have used CIIL corpus [3] for Indic scripts and have used British National Corpus (BNC) [2] for English. The BNC corpus consists of nearly 100 million words. However, to do justification in comparison (with respect to the corpus size), we have taken a subset of 5 million words from BNC to populate the data in the table. There are some interesting first level observations that we can infer from the table. The percentage of unique words in Indian languages is significantly higher than English. For eg. one in every three words in Malayalam is a new word whereas for English, it drops to one in 20 words approximately. Also note that the data shown in this table can be considered as an approximation as using a huge corpus of say 100 million can change the number. For eg. if we use the entire 100 million BNC corpus, we get a unique word count of 667,165, which is around 0.67% of the corpus. One way to gather raw text data in the present Internet age would be to crawl through the Internet and build a larger corpus. However, most of these languages do not have enough content available on Internet to generate a corpus which is even 10% of what we have for English. The entire Wikipedia dump of Malayalam is around a Million words in size. The case of other languages are also not very different. This table argues that the task of dictionary building is still an open task for Indic scripts as we need a much larger vocabulary to cover a considerable percentage of words.

Language	Total Words	Unique words	Average word length
Hindi	4,626,594	296,656 (6.42%)	3.71
Malayalam	3,057,972	912,109 (29.83%)	7.02
Kannada	2,766,191	654,799 (23.67%)	6.45
Tamil	3,763,587	775,182 (20.60%)	6.41
Telugu	4,365,122	1,117,972 (25.62%)	6.36
English	5,031,284	247,873 (4.93%)	4.66

Table 5.1 Dataset details for different languages. The percentage of unique words for Indic scripts is much larger when comparing with English.

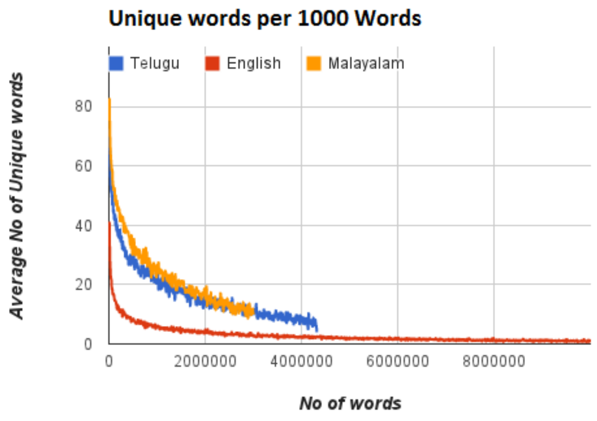


Figure 5.2 Unique word coverage between Malayalam, Telugu and English. For a language to get saturated, we would require a large dataset.(better seen in colour)

Another factor to take into consideration is the word coverage for different languages. Word coverage tell us how many unique words are needed to cover a certain percent of a language. This will help us in deciding the size of dictionary that will cover a considerable percentage of the language. There have been previous work showing such statistics [23]. We re-created the statistics from that work and the same is shown in Table 5.2. The table shows the comparison between different languages for coverage. To cover around 80% of the words, English requires around 8K words. The same for Telugu and Malayalam are closer to 300K. A linguist may argue that this explosion is primarily due to the morphological variation, and does not really relate to the richness in the language. However, at the level of error detection, this number critically affects, until we have morphological analyzers which can work on erroneous text.

The coverage statistics for Indian scripts are actually a poor estimate of the real quantity. Figure 5.2, presents the variation of the unseen words per 1000 words in a language as the corpus size increases. The graph shows that while the number of unique words becomes close to zero really fast for English, for languages like Telugu and Malayalam, the convergence has still not yet happened with the available corpus. If we try to estimate the size of the corpus required to stabilize the unique word fluctuation for other languages, it would come to approximately 10M. We got the value by considering the English graph as $Ke^{-\alpha x}$ and trying to find the area under the graph to estimate for other languages. It clearly shows that while it is easy to create a dictionary for English with extremely high coverage, similar dictionaries for Malayalam or Telugu are not immediately possible.

Corpus %	Malayalam	Tamil	Kannada	Telugu	Hindi	English
10	71	95	53	103	7	8
20	491	479	347	556	23	38
30	1969	1541	1273	2023	58	100
40	6061	4037	3593	5748	159	223
50	16,555	9680	8974	14,912	392	449
60	43,279	22,641	21,599	38,314	963	998
70	114,121	54,373	53,868	10,1110	2395	2573
80	300,515	140,164	144,424	271,474	6616	8711

Table 5.2 Word coverage statistics for different languages. As shown the number of unique words required to cover 80pc. of language varies from one language to another.

5.2.2 OCRs are Error Prone

Multiple OCRs exist for English that can provide character level accuracies in excess of 99% [5, 1]. However even the state of the art OCR systems for many Indic scripts have sub

90% accuracy [15] at character level. This increases the possibility of having multiple errors in a word, which makes the correction tougher. Also, from Table 5.1, we can see that the average word length is higher for Malayalam and Telugu. This means that, with a specific character classification accuracy, the word accuracies could be much lower for Malayalam and Telugu. The nature of the scripts and the complexities of the script, make the accuracies much smaller than what one could expect for English. With higher character and word error rates, the error detection and correction gets further compounded. We argue this with the help of hamming distance below.

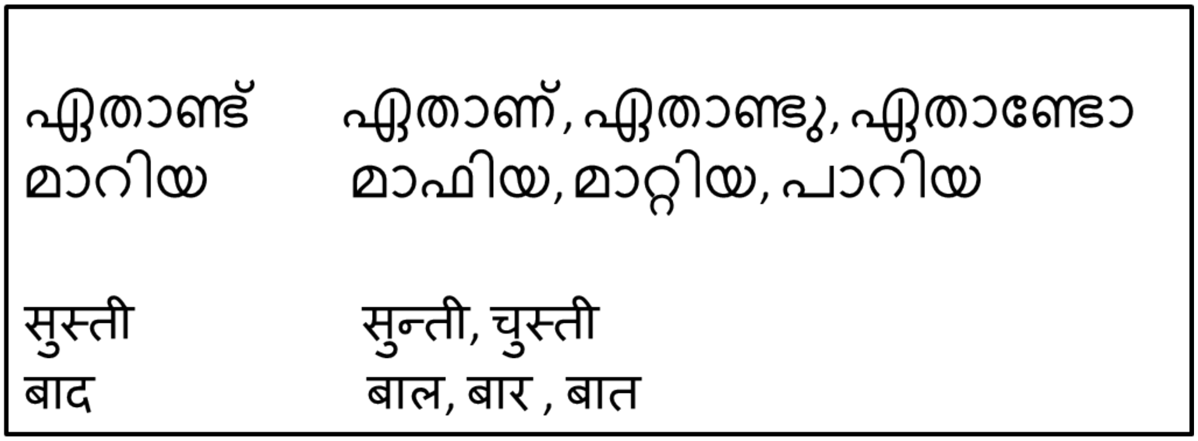
5.2.3 Error Detection Probability

Most error detection mechanisms discuss about detecting two types of errors: “Non-Word” and “Real-Word” errors. Non-word error words are those which cannot be considered as a valid word. eg: ”Ball” getting recognized as “8a11” or “l” getting confused with “!” etc. Such errors are easy to detect as the probability of a word containing letters and digits are extremely low. The second type of errors are tougher to detect. In this category, the resultant error word is another valid word. eg: “Cat” getting recognized as “Cab”. Such errors cannot be detected by considering words in isolation as detection requires availability of larger contextual information. Usage of word bigrams or trigrams have been proven to be successful in detecting such errors for English [68].

The possibility of detecting such errors in a language requires us to know what percentage of words have the above mentioned pattern. ie. an *akshara* getting replaced by another *akshara* results in generation of another valid word. Any language which has a large percentage of words exhibiting such a pattern would present a considerable challenge in error detection as every error word can also be considered as another valid word.

We investigated the Indian languages on these lines. The results were compared against English to observe if there exists any similarity between these languages. For a given hamming distance, we decided to find out what percentage of words can generate “real-word” errors. Hamming distance measure was chosen instead of edit distance as we felt that the classifier is more likely to make a substitution error rather than insertion/deletion errors.

For a given word (W) of word length (L), we computed Hamming distance (h) such that $h \leq L/2$. This was done as any word W_i of length L can be converted to W_j, which is also of length L, with a hamming distance L. eg. any 4 letter (akshara) word can be changed into another 4 letter (akshara) word using maximum 4 substitutions. As shown in the Fig. 5.4, Indian scripts exhibit a very peculiar behaviour where more than 1/3rd or even 1/2 of the words



ഏതാണ്ട്	ഏതാണ്, ഏതാണ്ടു, ഏതാണ്ടോ
മാറിയ	മാഫിയ, മാറ്റിയ, പാറിയ
सुस्ती	सुन्ती, चुस्ती
बाद	बाल, बार , बात

Figure 5.3 The above figure shows some words that can be converted to another valid character with a Hamming distance of 1.

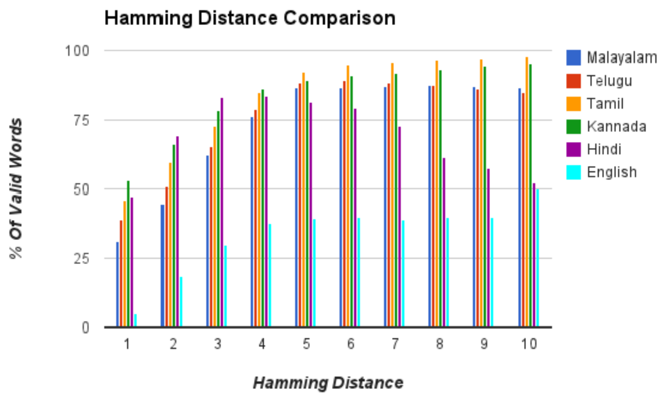


Figure 5.4 The above figure shows for a given hamming distance, what percentage of words gets converted to another valid word in a language.(better seen in colour)

can be converted to another valid word with a hamming distance of 1. While the percentage of words for English stands around 5%.

There could be multiple reasons for such characteristics to exist for Indian languages. The vocabulary size could be one major reason. While English character set can be limited to 52 characters, most Indian scripts have more than 200 symbols at their disposal. Also, the presence of considerable number of "*matras*" or vowel modifiers complicate the problem. Many such *matras* can be replaced with another *matra* to get another valid word. Figure 5.3 shows some examples from different languages where one word can be changed to another valid word with hamming distance one.

5.2.4 Error Correction Probability

The process of correcting a detected error is usually done by replacing the erroneous word/segment with another segment which has got the maximum probability of becoming right. The most common approach is by picking the "nearest" word with highest frequency of occurrence. To state an example, consider there is a wrong word W which has 5 words with hamming distance one in the dictionary. From these, we pick the one which has occurred the most in the language. Now the probability that the word with highest frequency of occurrence is actually the "right" word is 0.2 in this case. In fact, as the list grows larger, the difference in frequencies between the words will also start dropping. And at certain point, the significance of frequency will cease to exist. We will have a better chance of picking up the right candidate by choosing a sample randomly from the list. The odds will improve considerably when we tweak the method to include the individual nGram probability to the entire architecture. A majority voting mechanism can be used to improve the odds by selecting the word which is supported by both word and nGram dictionary. Further aid of language rules like a vowel cannot be replaced by matra can be used to improve the chances. However, this would require us to provide a set of possible matras and vowels to the algorithm.

5.3 Error Detection Experiments

After all the discussion about the challenges which we face in detecting errors, we look into possible methods in detecting errors. In this section, we propose multiple error detection methods and the evaluation parameter which we will be using for their evaluation.

We take Malayalam and Telugu as two working examples and demonstrate the error detection and correction methods. These languages were chosen as they appear most complex with respect to the number of unique words, average word length and word coverage.

5.3.1 Error Models

We believe that there are different types of errors that could arise and each of them needs to be handled differently. Below we give details of different error models and how it should be handled.

5.3.1.1 Natural erroneous data and Artificial data

These are the non-word and real-word errors mentioned in previous section. Artificial errors are the non-word errors which could be formed due to multiple reasons like degradations, noise etc. We feel that correction of such errors could be very tough using language models as the language information from such words is insufficient for any meaningful purposes. Such words would be tagged as ”Wrong words” and would be left as it is. One possible approach in correcting such errors could be re-classification of such words using better features/pre-processing etc.

5.3.1.2 Error at akshara level

Among the real-word errors, the errors which violate the basic word building principles are easy to detect. eg: Every *akshara* would be formed by following the pattern C^*V. i.e: zero or more consonant followed by a vowel. This will help us in detecting errors like cases where a vowel occurs in between two consonants etc.

5.3.1.3 Errors due to confusion among aksharas

Most of the languages have a set of aksharas which can get confused with each other. Such confusion can be taken into account while correcting errors. When presented with the decision to select one replacement from a list, higher weight can be given to such high confusion pairs ahead of other choices.

5.3.2 Methodology

We decided to look into possible solutions using both word level as well as sub-word(*akshara*) level methods.

5.3.2.1 Dictionary based model

The simplest of our proposed error detection models involves using a binary dictionary. Every word W_i from our test set is either considered as an in-vocabulary or as an out-of-vocabulary word. The success of such a method depends on the size of dictionary as a dictionary with larger size would infer larger word coverage.

5.3.2.2 nGram model

The above mentioned method was improved so that for any given word, we get the list of all possible *akshara* nGram combinations to compare it with dictionary. i.e: For a word of length 4, we get the list of all *akshara* 4 grams, 3 grams and 2 grams to be compared against the corresponding dictionary. This method possesses significant advantage against simple word dictionary based method as the possibility of detecting an error was much higher. In this case, the nGram probability of a word is shown as:

$$P\left(W_i\right) = \prod_{j=1}^{L} P\left(w_j | w_{j-n+1}^{j-1}\right) \geq \theta, \forall n : 2 \leq n \leq L$$

Here, θ is the threshold which decides if a word is valid or not. The model is flexible in a way such that we can have strict enforcement by saying criteria should be met $\forall n$ or it can be relaxed by saying if atleast one 'n' satisfies the criteria, word is considered valid.

5.3.3 Detection through learning

A formal method towards detection of errors can also be achieved using popular machine learning approaches. We use linear classification methods to classify a word as valid or invalid. We consider the nGram probabilities of a word as its features and send them to a linear classifier for learning. One constraint with such a method is that we require a set of ”false” labels so that the classification can happen. For this, we use a subset of our OCR output and train a binary classifier. This classifier is used to detect errors in later stages.

5.3.4 Evaluation Metrics

Error detection performance is an important factor as improper detection can significantly reduce the accuracy while error correction. eg. tagging correct words as wrong will result in "correcting" the right word to some other word, which will decrease the overall word accuracy. We use True Positive(TP), False Positive(FP), True Negative(TN) and False Negative(FN) for evaluation purposes.

A word is considered TP if our model detects it as invalid and the result is seconded by the annotation. Similarly, a word is considered as TN when we tag it as a valid word and it is in-fact a valid word. We show these numbers in percentages where TP shows percentage of invalid words which we detected correctly among all possible invalid words present. Since our final target is to correct the words which we tag as error, TP has higher significance than FP. Also, the presence of TN will indicate the level of corruptness among the list of words which our model tagged as wrong. This is significant as higher level corruptness would indicate that an error correction module would try to "correct" a word which is already correct and thus, can make it wrong. Hence, any advantage that we gained by correcting the wrong words would be gone. We also compute Precision, Recall and F-Score based on the above values for our evaluation.

5.3.5 Experiments and Results

We test our methods on the OCR outputs of state-of-the art OCR for Malayalam and Telugu [15]. Malayalam OCR has a character accuracy of around 95% whereas for Telugu, accuracy is around 75%. Both the OCRs where used to convert around 5000 pages of printed document into unicode text. The pages are of reasonable quality, taken from multiple books. We have used a dictionary covering the entire dataset and contains around 670K words for Malayalam and 700K words for Telugu. The dictionary was generated using the CIIL corpus [3] from which all unique Malayalam/Telugu words were extracted.

We tested the approaches mentioned in previous section and evaluated them on the evaluation parameters. The results are shown in Table 5.3 and Table 5.4.

Table 5.3 shows the percentage of TP, FP, TN and FN. One observation is that the FP percentage is highest for simple dictionary based method. False positive is the percentage of words that we tagged as invalid words but are actually valid words. This is because any Out-of-Vocabulary (OOV) words are tagged as invalid and since the coverage of dictionary is limited, more words get tagged as invalid. This value is decreasing as we keep using nGrams or SVMs. We are able to achieve better coverage by using such methods. Percentage of false negatives shows the amount of real-word errors existing in the data. These are those words whih has

Table 5.3 True Positive, False Positive, True Negative and false negative percentage for different methods

Method	Malayalam				Telugu			
	TP	FP	TN	FN	TP	FP	TN	FN
Word Dict.	72.36	22.88	77.12	27.63	94.32	92.13	7.87	5.67
nGram	72.85	22.17	77.83	27.15	62.12	6.37	93.63	37.88
Dict+nGram	67.97	14.95	85.04	32.02	65.01	2.2	97.8	34.99
Dict+SVM	62.87	9.73	90.27	37.13	68.48	3.24	96.76	31.52

Table 5.4 Precision, Recall and F-Score values for different methods

Method	Malayalam			Telugu		
	Precision	Recall	**F-Score**	Precision	Recall	**F-Score**
Word Dict.	0.52	0.72	**0.60**	0.51	0.94	**0.68**
nGram	0.53	0.73	**0.61**	0.91	0.62	**0.73**
Dict+nGram	0.61	0.68	**0.64**	0.94	0.64	**0.76**
Dict+SVM	0.69	0.63	**0.66**	0.95	0.67	**0.78**

been tagged as valid but is infact invalid. Such errors cannot be detected using such word/sub word based methods. Another observation is with respect to precision in Table 5.4. Precision provides the percentage of correctly detected invalid words from all words tagged as invalid. The significance of this measure is when we try to correct the detected errors. Low precision implies that the percentage of FP is high and the possibility of overall accuracy decreasing after error correction. This is because while correcting, we will be trying to 'correct' an already correct word which would eventually make that word invalid. Ideally, we would like to have 0 FP so that even in worst case scenario, the accuracy does not drop after error correction.

Using nGram dictionary along with word dictionary have helped in bringing down the FP ratio, especially for Telugu. However, deciding on the right threshold value determines the accuracy of this method. Dictionary coupled with SVMs have proven to be the best pick among the tested methods. Although SVMs are using the nGram features for classification, better formalization have helped in improving the results. Error detection score of 0.78 which we obtained for Telugu is comparable to detection scores for Hindi, which has detection score of 0.8.

5.3.6 Integration with an OCR

The field run of our method using an OCR for error correction is mentioned in this section. We use Malayalam OCR from [15] to obtain word outputs of 5000 pages. The OCR was having an initial accuracy of around 73% at word level. We integrated our method with OCR to detect the errors generated by it and proceeded to correct them as required. The error words where corrected by selecting the replacements from a set of possible nGrams and dictionary words. We were able to achieve a reduction of 10% in word error rate using our method.

One main limitation of our error correction mechanism is that it assumes perfect character segmentation i.e: no cuts or merges in characters. We understand this assumption is not practical. While we are able to detect such errors using our method, correction of such errors would be tough using a word replacement method. This is because in cases where there are two or more characters joined together, we have to rely on edit distance and not hamming distance to find the replacement. Such a method will increase the possible number of replacement words exponentially. A better method needs to be identified for such corrections. Note that the primary objective of this work is in exposing the challenges associated with the error detection of highly inflectional languages.

5.4 Summary

In this work, we have identified the linguistic challenges for Indian languages from a statistical point of view. We have identified the word distribution and other characteristics for inflectional languages and compared them against English. We have identified the major challenges in developing a language model for such languages in this work. We proposed different methods to overcome the challenges and validated our method on a large testing corpora.

Chapter 6

Conclusions

We propose the problem of OCR to be reformulated as that of transcribing feature vector set to output labels in this work. Challenges in word-symbol segmentation and conversion of latent symbol to Unicode were some of the reasons which made us in thinking in this direction. Our method presents a uniform architecture which will perform recognition for most of Indic scripts with much higher accuracy than other state-of-the-art systems. We used a neural network known as BLSTM, a variant of RNN, to achieve this. We also consider UNICODE to be the basic recognition unit thereby introducing a universally accepted standard for labelling. Such a step helped us in using same architecture for transcription of multiple languages. We show the results on 7 Indian languages and report considerable high accuracies at both character level and word level. The method was tested on approximately 5000 pages for these languages.

In future, we would like to explore the possibility of using a strong language model to improve the performance of our system. Adaptation towards complex scripts like Urdu, Arabic etc. is another direction which we hope to travel. It would be also interesting to experiment on language specific features to improve the accuracy.

In this work, we also discussed why error detection is difficult for inflectional languages, specially by comparing with languages like English. Issues related to size, coverage, hamming distance and classifier accuracies were discussed. The complexity of Indic script only add to the problems. We analysed the problem and came up with several error models and ways to detect them. Our error detection mechanisms where able to achieve an F-score of 0.66 for Malayalam and 0.78 for Telugu. Simple dictionary-frequency based error correction was performed on Malayalam and we were able to obtain word error rate reduction of 10%.

We would like to extend this work by looking into more efficient and effective methods of detecting and correcting errors. We would like to investigate the significance of specific nGrams in language that can aid us in detecting/correcting errors. A more formal method of correcting errors using machine learning techniques will also be an interesting direction to look into.

Publications

1. Naveen Sankaran and C. V. Jawahar: **Recognition of printed Devanagari text using BLSTM Neural Network**, International Conference on Pattern Recognition *(ICPR)* 2012

2. Naveen Sankaran, Aman Neelappa and C. V. Jawahar: **Devanagari Text Recognition: A Transcription Based Formulation**, International Conference on Document Analysis and Recognition *(ICDAR)* 2013

3. Naveen Sankaran and C. V. Jawahar: **Error Detection in Highly Inflectional Languages**, International Conference on Document Analysis and Recognition *(ICDAR)* 2013

4. Praveen Krishnan, Naveen Sankaran, Ajeet Kumar Singh and C. V. Jawahar: **Towards a Robust OCR System for Indic Scripts**, Document Analysis Systems *(DAS)* 2014

Other Related Publications

1. Udit Roy, Naveen Sankaran, K. Pramod Sankar and C. V. Jawahar: **Character N-Gram Spotting on Handwritten Documents Using Weakly-Supervised Segmentation**, International Conference on Document Analysis and Recognition *ICDAR* 2013

2. Shrey Dutta, Naveen Sankaran, K. Pramod Sankar and C. V. Jawahar: **Robust Recognition of Degraded Documents Using Character N-Grams** Document Analysis Systems, *DAS* 2012

3. Devendra Sachan, Shrey Dutta, Naveen T S and C.V. Jawahar: **Segmentation of Degraded Malayalam Words:Methods and Evaluation**, Proceedings of 3rd National Conference on Computer Vision, Pattern Recognition, Image Processing and Graphics *NCVPRIPG* 2011

4. Anand Mishra, Naveen Sankaran, Viresh Ranjan and C. V. Jawahar: **Automatic localization and correction of line segmentation errors**, Proceeding of the workshop on Document Analysis and Recognition, 2012

Bibliography

[1] ABBYY FineReader.

[2] British National Corpus (BNC).

[3] Central Institute Of Indian Languages (CIIL) Corpus.

[4] Official Census by Government of India.

[5] Tesseract Optical Character Recognition Engine.

[6] The OCRopus(tm) open source document analysis and OCR system.

[7] A Amin and S Singh. Recognition of Hand-Printed Chinese Characters using Decision Trees/Machine Learning C4.5 System . In *Pattern Analysis and Applications*, 1998.

[8] Adnan Ul-Hasan and Saad Bin Ahmed and Sheikh Faisal Rashid and Faisal Shafait and Thomas M. Breuel. Offline Printed Urdu Nastaleeq Script Recognition with Bidirectional LSTM Networks. In *ICDAR*, 2013.

[9] Alex Graves and Jürgen Schmidhuber. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks*, 2005.

[10] Alex Graves and Jürgen Schmidhuber. Offline Handwriting Recognition with Multidimensional Recurrent Neural Networks. In *NIPS*, 2008.

[11] Alex Graves and Santiago Fernández and Faustino J. Gomez and Jürgen Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *ICML*, 2006.

[12] Alex Graves and Santiago Fernández and Jürgen Schmidhuber. Bidirectional LSTM Networks for Improved Phoneme Classification and Recognition. In *ICANN (2)*, 2005.

[13] Andrew J. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 1967.

[14] K. Aparna and A. Ramakrishnan. A complete Tamil Optical Character Recognition system. In *Document Analysis System*, 2002.

[15] D. Arya, T. Patnaik, S. Chaudhury, C. V. Jawahar, B.B.Chaudhuri, A.G.Ramakrishna, C. Bhagvati, and G. S. Lehal. Experiences of Integration and Performance Testing of Multilingual OCR for Printed Indian Scripts. In *J-MOCR Workshop,ICDAR*, 2011.

[16] T. Ashwin and P. S. Sastry. A font and size independent OCR system for printed kannada documents using support machines. 2002.

[17] Atul Negi, Chakravarthy Bhagavati and B Krishna. An OCR system for Telugu. In *ICDAR*, 2001.

[18] B. B. Chaudhuri and U. Pal. An OCR System to Read Two Indian Language Scripts: Bangla and Devnagari (Hindi). In *ICDAR*, 1997.

[19] B. B. Chaudhuri and U. Pal. A complete printed Bangla OCR system. *Pattern Recognition*, 1998.

[20] B. Vijay Kumar and A. G. Ramakrishnan. Machine Recognition of Printed Kannada Text. In *Document Analysis Systems V*, 2002.

[21] V. Bansal and R. M. K. Sinha. A Devanagari OCR and a brief overview of OCR research for Indian scripts. In *Proc. of the Symposium on Translation Support Systems*, 2000.

[22] V. Bansal and R. M. K. Sinha. A Complete OCR for Printed Hindi Text in Devanagari Script. In *ICDAR*, 2001.

[23] A. Bharati, P. Rao, R. Sangal, and S. M. Bendre. Basic Statistical Analaysis of Corpus and Cross Comparision. In *Proceedings of ICON-2002: International Conference on Natural Language Processing, Mumbai*, 2002.

[24] Brown, Peter F. and Cocke, John and Pietra, Stephen A. Della and Pietra, Vincent J. Della and Jelinek, Fredrick and Lafferty, John D. and Mercer, Robert L. and Roossin, Paul S. A Statistical approach to Machine Translation. *Comput. Linguist.*, 1990.

[25] C V Jawahar and Anand Kumar. Content-level Annotation of Large Collection of Printed Document Images. In *ICDAR*, 2007.

[26] C. V. Jawahar, MNSSK Pavan Kumar and S. S. Ravikiran. A Bilingual OCR system for Hindi-Telugu Documents and its Applications. In *International Conference on Document Analysis and Recognition(ICDAR)*, 2003.

[27] B. Chaudhuri, U. Pal, and P. Kundu. Non-word error detection and correction of an inflectional Indian Language. In *Symposium on Machine Aids for Translation and Communication(SMATAC-96)*, 1996.

[28] D. C. Ciresan and Jürgen Schmidhuber. Multi-Column Deep Neural Networks for Offline Handwritten Chinese Character Classification. In *ICDAR*, 2013.

[29] D. C. Ciresan and U. Meier and Jürgen Schmidhuber. Multi-column Deep Neural Networks for Image Classification. In *IEEE Conf. on Computer Vision and Pattern Recognition CVPR*, 2012.

[30] F El-Khaly and M.A Sid-Ahmed. Machine recognition of optically captured machine printed Arabic text. In *Pattern Recognition*, 1990.

[31] V. Frinken, A. Fischer, and H. Bunke. A Novel Word Spotting Algorithm Using Bidirectional Long Short-Term Memory Neural Networks. In *ANNPR*, 2010.

[32] V. Frinken, A. Fischer, R. Manmatha, and H. Bunke. A Novel Word Spotting Method Based on Recurrent Neural Networks. *IEEE Trans. Pattern Anal. Mach. Intell.*, 2012.

[33] G Nagy. At the frontiers of OCR. In *Proceedings of IEEE-1992*, 1992.

[34] G Prathap. Indian Language Document Analysis and Understanding. In *Special Issue of Sadhana*, 2002.

[35] G S Lehal and C Singh. A Gurumukhi Script Recognition System. In *Inernational Conference on Pattern Recognition (ICPR)*, 2000.

[36] Ghosh, Subhankar and Bora, P. K. and Das, Sanjib and Chaudhuri, B. B. Development of an Assamese OCR Using Bangla OCR. In *Proceeding of the Workshop on Document Analysis and Recognition*, DAR '12, 2012.

[37] R. Golding and Y. Schabes. Combining trigram-based and feature-based methods for context-sensitive spelling correction. In *Proceedings off the 34th Annual Meeting of the ACL*, 1996.

[38] V. Govindaraju and S. Setlur. *Guide to OCR for Indic Scripts*. Springer, 2009.

[39] A. Graves, M. Liwicki, S. Fernandez, R. Bertolami, H. Bunke, and J. Schmidhuber. A Novel Connectionist System for Unconstrained Handwriting Recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 2009.

[40] K.H Aparna and V. S. Chakravarthy. A complete OCR system development of Tamil magazine documents. In *Tamil Internet, Chennai*, 2003.

[41] Koby Crammer and Yoram Singer. On the Algorithmic Implementation of Multiclass Kernel-based Vector Machines. *Journal of Machine Learning Research*, 2001.

[42] K. Kukich. Techniques for automatically correcting words in text. *ACM Comput. Surv.*, 24(4), 1992.

[43] L E Baum and T Petrie. Statistical inference for probabilistic functions of finite state markov chain. *The Annals of Mathematical Statistics*, 1966.

[44] Lawrence R Rabiner. A tutorial on Hidden Markov Models and selected aplications in speech recognition. In *Proceeding of the IEEE*, 1989.

[45] Lehal, Gurpreet Singh. Optical Character Recognition of Gurmukhi Script Using Multiple Classifiers. In *Proceedings of the International Workshop on Multilingual OCR*, MOCR '09, 2009.

[46] Marcus Liwicki and Horst Bunke. Feature Selection for HMM and BLSTM Based Handwriting Recognition of Whiteboard Notes. *IJPRAI*, 2009.

[47] Matusov, Evgeny and Kanthak, Stephan and Ney, Hermann. On the Integration of Speech Recognition and Statistical Machine Translation. In *Proceedings of Interspeech, Lisbon, Portugal*, 2005.

[48] Mike Schuster and Kuldip K. Paliwal. Bidirectional Recurrent Neural Networks. *IEEE Transactions on Signal Processing*, 1997.

[49] Neeba N.V. and C.V. Jawahar. Emperical evaluation of Character Classification schemes. In *International Conference on Advances in Pattern Recognition*, 2009.

[50] P. Rao and T. Ajitha. Telugu script recognition - A feature based approach. In *ICDAR*, 1995.

[51] T. L. Packer. Performing information extraction to improve ocr error detection in semi-structured historical documents. In *Proceedings of the 2011 Workshop on Historical Document Imaging and Processing*, 2011.

[52] U. Pal and B. Chaudhuri. Printed Devanagiri Script OCR System. In *Vivek*, 1997.

[53] U. Pal and B. B. Chaudhuri. Indian Script Character Recognition: A Survey . In *Pattern Recognition*, 2004.

[54] U. Pal, P. K. Kundu, and B. B. Chaudhuri. OCR error correction of an inflectional indian language using morphological parsing. *Journal Of Information Science and Engineering*, 16, 2000.

[55] J. Perez-Cortes, J. Amengual, J. Arlandis, and R. Llobet. Stochastic error-correcting parsing for OCR post-processing. In *Pattern Recognition, 2000. Proceedings. 15th International Conference on*, 2000.

[56] Pinar Duygulu and Kobus Barnard and João F. G. de Freitas and David A. Forsyth. Object Recognition as Machine Translation: Learning a Lexicon for a Fixed Image Vocabulary. In *ECCV (4)*, 2002.

[57] Premkumar S. Natarajan and Ehry MacRostie and Michael Decerbo. The BBN Byblos Hindi OCR system. In *DRR*, 2005.

[58] R. J. Williams and D. Zipser. Gradient-based learning algorithms for recurrent networks and their computational complexity. In *Back-propagation: Theory, Architectures and Applications*, 1995.

[59] Raman Jain and Volkmar Frinken and C. V. Jawahar and Raghavan Manmatha. BLSTM Neural Network Based Word Retrieval for Hindi Documents. In *ICDAR*, 2011.

[60] S. Antanani and L. Agnihotri. Gujarati Character Recognition. In *Inernational Conference on Document Analysis and Recognition (ICDAR)*, 1999.

[61] S Mori and C.Y Suen and K Yamamoto. Historical review of OCR research and development. In *Proceedings of IEEE-1992*, 1992.

[62] N. Sankaran and C. Jawahar. Recognition of Printed Devanagari text using BLSTM neural network. In *ICPR*, 2012.

[63] T. Sari and M. Sellami. MOrpho-LEXical Analysis for Correcting OCR-generated arabic words (MOLEX). *Ninth International Workshop on Frontiers in Handwriting Recognition*, 2002.

[64] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 1997.

[65] S. Shalev-Shwartz, Y. Singer, N. Srebro, and A. Cotter. Pegasos: Primal Estimated Sub-Gradient solver for SVM. In *Math. Program*, 2011.

[66] D. V. Sharma, G. S. Lehal, and S. Mehta. Shape encoded post processing of Gurmukhi OCR. In *ICDAR*, 2009.

[67] Shrey Dutta and Naveen Sankaran and K. Pramod Sankar and C. V. Jawahar. Robust Recognition of Degraded Documents Using Character N-Grams. In *Document Analysis Systems*, 2012.

[68] R. Smith. Limits on the Application of Frequency-Based Language Models to OCR. In *ICDAR*, 2011.

[69] T. V. Ashwin and P. S. Sastry. A Font and Size Independent OCR system for Printed Kannada Documents using Support Vector Machines. In *Special Issue of Sadhana*, 2002.

[70] Thomas M. Breuel and Adnan Ul-Hasan and Mayce Ibrahim Ali Al Azawi and Faisal Shafait. High-Performance OCR for Printed English and Fraktur Using LSTM Networks. In *ICDAR*, 2013.

[71] Toni M. Rath and R. Manmatha. Features for Word Spotting in Historical Manuscripts. In *ICDAR*, 2003.

[72] U. Pal and A. Sarkar. Recognition of printed Urdu script. In *ICDAR*, 2003.

[73] Utpal Garain and B. B. Chaudhuri. Segmentation of Touching Characters in Printed Devnagari and Bangla Scripts Using Fuzzy Multifactorial Analysis. In *ICDAR*, 2001.

[74] Veena Bansal and R. M. K. Sinha. Segmentation of touching and fused Devanagari characters. *Pattern Recognition*, 2002.

[75] Venkat Rasagna, Jinesh K.J. and C.V. Jawahar. On Multifond Character Classification in Telugu. In *International Conference on Information Systems for Indian Languages (ICISIL)*, 2011.

[76] M. Wick, M. Ross, and E. Learned-Miller. Context-sensitive Error Correction: Using Topic Models to Improve OCR. In *Document Analysis and Recognition, 2007. ICDAR 2007. Ninth International Conference on*, 2007.