Robust Multilingual OCR: from Ancient Indic Texts to Modern Indian Street Signs

Submitted in partial fulfillment of the requirements

of the degree of

Doctor of Philosophy

of the
Indian Institute of Technology Bombay, India
and
Monash University, Australia

by

Rohit Saluja

Supervisors:

Ganesh Ramakrishnan (IIT Bombay)

Parag Chaudhuri (IIT Bombay)

Mark Carman (Monash University)

The course of study for this award was developed jointly by
Monash University, Australia and the Indian Institute of Technology Bombay, India
and was given academic recognition by each of them.
The programme was administrated by The IITB-Monash Research Academy

(Year 2020)

Declaration

I declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I declare that I have properly and accurately acknowledged all sources used in the production of this report. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be a cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Student Name: Rohit Saluja

Abstract

It has been an integral part of the human journey to try and create machines that mimic us to ease our jobs, such as translation, reading, typing, and proofreading. Optical Character Recognition (OCR), the process of converting document or scene-text images to editable electronic format, is one of the outcomes of such human endeavor. The basic OCR steps include the pre-processing of image, text recognition, and post-processing. The pre-processing may include noise-removal, binarization, and/or segmentation to improve the quality of reading by the OCR systems. The text recognition, which is analogous to reading and typing by humans, is conventionally performed using a character classifier or a word recognizer, which is prone to errors. The post-processing of the erroneous OCR text is analogous to the proofreading by humans. It involves automatic corrections using an auxiliary source such as language dictionaries or language models. The quality of correction depends upon the type of auxiliary source, which is analogous to the expertise of a proofreader. It is also possible to improve the quality of corrections with human-in-the-loop.

To first comprehend the problem of text recognition in ancient Indic documents, we observe that a large proportion of printed ancient documents (referred to as monolithic document collections) exhibit nearly uniform font and language characteristics. Further investigations demonstrate that texts in Indic languages contain a large proportion of out-of-vocabulary words due to regular fusion and agglutination using conjoining rules. Moreover, using Open Source and Commercial systems, we observe Word Error Rates (WER) of around $20 - 50\%$ on nearly $100k$ OCR words from the printed documents of different Indic languages. We further assert that any OCR system, even with accuracies as high as 90% is not sufficiently useful, and requires a tremendous manual effort for corrections unless complemented by a partially automated correction mechanism. On monolithic document collections, it is desirable that the error correction system continuously improves itself by incorporating user feedback. We thus present OpenOCRCorrect (Saluja *et al.*, 2017b; Adiga *et al.*, 2018): an interactive framework for assisting word-level corrections in

Indic OCR documents. The framework leverages generic word dictionaries and a domain-specific vocabulary grown incrementally based on user corrections. It also learns OCR-specific confusions on-the-fly. We have incorporated word conjoining rules to parse OCR words and discover their potentially correct sub-strings. Furthermore, we present a dual-engine environment to cross-verify potential errors and corrections. We also present a plug-in classification approach to further improve the error detection results by tuning the probability threshold for classification. We show that such an interactive approach for word-level corrections applies to Indian languages with varying degrees of inflections. Given the role of user interaction in OpenOCRCorrect, we have carefully designed the UI to reduce the overall cognitive load by use of transliteration schemes, suitable color-coding, and learning on-the-fly from interactions. We then adopt a character level Long Short Term Memory (LSTM) model with a *fixed delay* for jointly addressing the problems of error detection and correction in Indic OCR (Saluja *et al.*, 2017a). Such a model learns the language as well as OCR-specific confusions. We work on the task of error correction in four different Indian languages with varying complexities. We further augment the input to LSTM based models with different encodings to capture the sub-word frequency values on a corpus (Saluja *et al.*, 2019b). Such models perform well when the correction dataset is in order of $100k$ words. However, better models work when the dataset is in order of $1000k$ words. We present that a complex encoder and decoder model consisting of a separate LSTM for each, aided by attention mechanism, is successful in learning the correction mechanism similar to the basic LSTM model. Using such a model, our team "CLAM" (Character Level Attention Model) secures 2^{nd} position in the ICDAR, 2019 PostOCR Competition on 10 languages. We also achieve the highest correction score for Finnish.

Our investigations demonstrate that modern Indian street signs and license plates can pose an even tougher machine reading challenge. They often appear in a variety of languages, fonts, sizes, and orientations. To solve the problem of reading street text in videos, we first leverage state-of-the-art text spotters (generally trained on distinct text images only) to generate a large amount of noisy labelled training data (Saluja *et al.*, 2019a). If specific domain knowledge is available, the noisy data is filtered using a pattern derived from such knowledge (e.g., in license plates; the text has to follow a set pattern). We also augment the data with interpolated boxes and annotations that make the training and testing robust for reading text in videos. Further use of synthetic data increases the coverage of the training process. The baselines include black-box detectors such as Convolution Neural Networks

(CNN) and human annotators, followed by the Recurrent Neural Network (RNN) based recognizer. Next, we build in the capability of training the model in an end-to-end fashion on scenes containing multi-lingual text by incorporating i) an inception-based CNN encoder and ii) a location-sensitive attention mechanism in the decoder. We present the first results of using multi-headed attention models on text recognition in videos and illustrate the advantages of using multiple heads over a single head. To ease the correction process in Indian traffic videos via interactivity, we present StreetOCRCorrect, which uses available detectors and trackers to break down the multi-vehicle videos into multiple clips, each containing a single vehicle from the video (Singh *et al.*, 2019). We then incorporate a multi-frame consensus (on the OCR output of each clip) for generating suggestions. The high-quality annotations obtained from such a framework can be helpful to continuously update an extensive database for surveillance as well as improving deep models on video data. On the one hand, attention-based methods have shown promise for scene-text OCR, while on the other hand, the attention masks tend to wander in the scene, making the process less effective. Our efforts also include associating semantics with attention masks and then thoroughly supervising those masks to improve scene OCR in videos.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

The digitization of images can empower machines to index historical literature that exists only in printed form. Moreover, digitization is important in both academia as well as industry for storage, reproducibility, and summarization. Due to the need for digitization, Optical Character Recognition (OCR) has became an active area of research in document understanding and computer vision. Nevertheless, even as the quality of OCR systems has improved, OCR has remained a challenging problem in various contexts like the digitisation of ancient texts and reading the haphazard text present in modern street scenes. Furthermore, language specific challenges make the overall process cumbersome. This enabled the language specific research of reading images that contain ancient and modern texts.

In this thesis, we present our investigations in the field of automatically reading ancient Indic texts and modern Indian street signs. We begin this chapter with a discussion on OCR and its various forms in Section 1.1. We then describe the basic OCR steps in Section 1.2. In Section 1.4, we present various interesting details about Indian languages to understand the challenges involved in Indic OCR. We then briefly discuss document collections and prominent OCR systems for Indian Languages in Sections 1.5 and 1.6. The chapter continues with the motivation for the necessity of interactive OCR corrections and improved scene-text models in Section 1.7. We then conclude the chapter with publications, awards, and key contributions in Sections 1.8 and 1.9.

1.1 Optical Character Recognition

Optical Character Recognition is the process of converting images of documents (or outdoor/indoor scenes) to editable text format (Cheriet *et al.*, 2007). This enables

many applications, like the automatic processing of documents/forms/routing of envelopes based on zip code, and the reading aloud of photographed text for the visually impaired (Mor and Wolf, 2018). OCR can be broadly divided into two categories: document OCR and photo OCR (or scene-text recognition), which we discuss in Sections 1.1.1 and 1.1.2 respectively.

1.1.1 Document OCR

Document OCR is the process of text recognition in images like scanned books, magazines and newspapers. As noted by La Manna *et al.* (1999), document processing is a complex task, consisting of several steps and employing different techniques according to its specific purpose. Variations such as domain specific texts, fonts (in ancient books), languages, skew, scanner noise, layouts and tables present in such images create challenges for any OCR system to produce reliable outputs. Document OCR has many applications like data compression, enabling search or edit options in the images, reading tables, form parsing (Davis *et al.*, 2019), and can help in creating the database for resources/applications like language models, domain specific dictionaries and machine translation.

1.1.2 Photo OCR

Photo Optical Character Recognition (photo OCR) aims to read scene-text in natural images. It is an essential step for a wide variety of computer vision tasks, and has enjoyed significant success in several commercial applications (Lee and Osindero, 2016). Scene-text in the real world is generally unstructured, appearing in a variety of languages, fonts, sizes and orientations. Additionally, movement of the capturing camera makes photo OCR an even more challenging task. Various applications including machine translation, autonomous driving and text to speech rely on recognition of scene-text in images or videos.

1.2 Building blocks of the OCR process

As shown in Figure 1.1, the basic OCR steps conventionally include the pre-processing of an image, text recognition and post-processing of the OCR text.

1.2.1 Pre-processing

The Pre-processing stage may include noise-removal, skew-correction, binarization, text-localization and/or segmentation. The OCR process typically includes

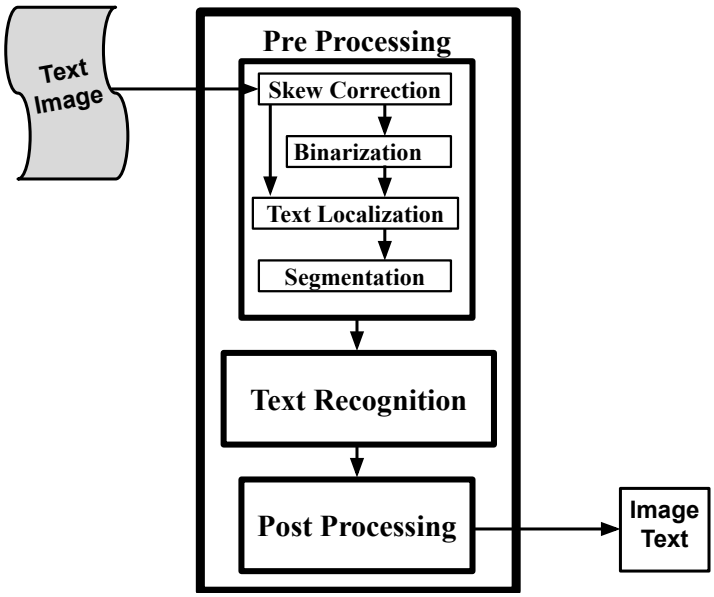


Figure 1.1: Basic OCR steps

segmentation of an image or document into higher levels of granularity such as pages or columns or paragraphs to lower level of granularity such as lines or words or even characters (Smith, 2007, 2009). For photo OCR, work specific to the localization task by Gupta *et al.* (2016), has also been extended to real-time text detection (Liao *et al.*, 2017; Minghui Liao and Bai, 2018). For multi-lingual text images, this process may also include language identification for the selection of an appropriate recognizer (Mathew *et al.*, 2016).

1.2.2 Text recognition

In order to extract and read text, image classifiers (a.k.a. recognizers) are used to read the individual characters (or sometimes entire words, lines or even paragraphs) within an image (Smith, 2007; Smith *et al.*, 2016; Arya *et al.*, 2011; Sankaran and Jawahar, 2013; Wojna *et al.*, 2017). These classifiers can be simple Gaussian Mixture Models (GMM) (Smith, 2007), Hidden Markov Models (HMM) (Toselli and Vidal, 2013) or deep neural networks (Arya *et al.*, 2011; Sankaran and Jawahar, 2013).

1.2.3 Post-processing

The processing of images for recognition at character or word level often suffers from segmentation errors (Kameshiro *et al.*, 1999; Breuel *et al.*, 2013; Smith, 2011). Moreover, various degradations in the images (as can be seen in Figure 1.5) also

lead to errors in the OCR output. Thus the post-processing of OCR text forms an important part of the OCR process. Post-processing may be performed by following a context free approach using a language dictionary, or a context based approach using a language model (Smith, 2007; Saluja *et al.*, 2017a). Smith (2011) discusses the limitations of such approaches since they do not consider OCR specific errors or confusion patterns, and concludes that the noisy-channel models that closely model the underlying classifier and segmentation errors are required for the post-processing of OCR text.

1.3 Human-machine Interactions in Digitization

Three scenarios are possible for human-machine interaction during the digitization process:

1. Pre-processing by machine, recognition by humans.
2. Recognition by humans, post-processing by machines.
3. Recognition by OCR systems, post-processing by humans.

The first scenario wherein machines improve the image quality and human reads and types the text in images is not much of a help since the visual system of the human is a masterpiece and can recognize the text in document images despite various types of noises. The pre-processing techniques are not as much of use to humans as they are to the OCR systems themselves. Moreover, recognition by humans generally requires additional proofreading step.

The second scenario, where human types the text in images and machine acts as a proofreader, is analogous to spell-checkers. Here, despite using language resources, human-in-the-loop is essential. Additionally, the first two scenarios are expensive in terms of the typing efforts required by humans.

It can take a few years to correct just a thousand sets of books if a human performs recognition. However, an OCR system can recognize such large volumes in just a single day. Thus, the third scenario of text recognition by OCR systems and proofreading by humans is exciting and can additionally benefit from the language resources, the specific error patterns of the OCR systems as well as human-in-the-loop. We will elaborate upon it in the subsequent sections.

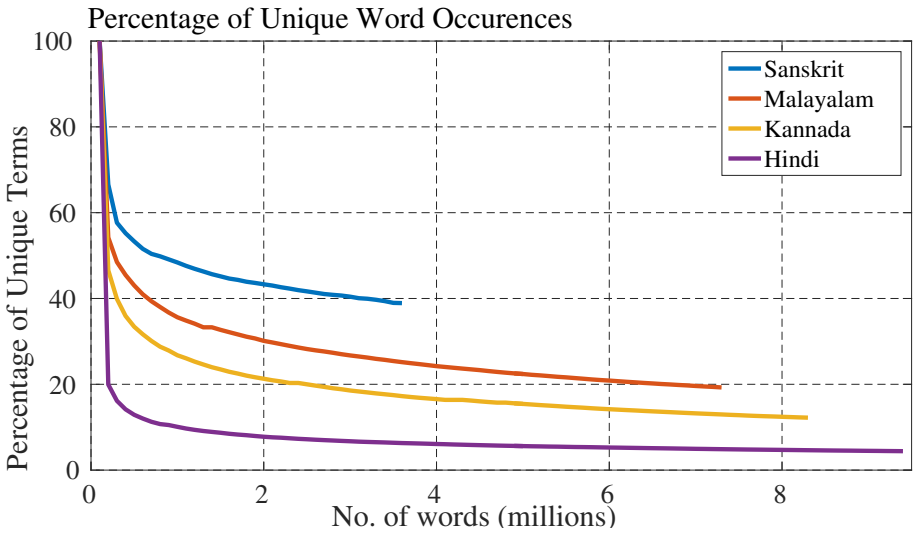


Figure 1.2: Unique word coverage between Sanskrit, Malayalam, Kannada and Hindi

1.4 Interesting details about Indian languages

Certain vocabulary characteristics, such as dynamism and size, can be analyzed through the graph of unique words versus corpus length. This has been studied for Malayalam and Telugu by Sankaran and Jawahar, 2013. In Figure 1.2, we present a similar analysis for Sanskrit, Malayalam, Kannada and Hindi. As shown, the vocabulary is most dynamic/incomplete in Sanskrit, followed by Malayalam, Kannada, and Hindi. This happens because Indic texts contain a large proportion of out-of-vocabulary (OOV) words due to frequent fusion using conjoining rules (Dīkṣita *et al.*, 2006), of which there are around $4k$ in Sanskrit. For Indic Languages, the conjoining rules are of two types, viz., agglutinative and fusional. The simple rules involve the agglutination of valid word forms, just as the words "can" and "not" form the word "cannot" in English. The complex rules involve the fusion of joined words, analogous to the formation of "couldn't" by joining the words "could" and "not" in English. The same is illustrated with an example in English and Sanskrit each. Valid lexicons are marked in alternate blue and green colors, and fused n-grams in orange.

- Agglutinative Rules:
  - Every + one = Everyone.
  - अनन्तशयन + संस्कृत + ग्रन्थावलिः = अनन्तशयनसंस्कृतग्रन्थावलिः
- Fusional Rules:
  - Would + not = Wouldn't.

– नित्य + संबद्धयोः + उपमानत्वेन + उपादानात् = नित्यसंबद्धयोरुपमानत्वेनोपादानात्

In Indic languages, often more than two words are conjoined using the same agglutinative and fusional rules, as can be seen from the above examples. For example, even if the word "wouldn't" was an out-of-vocabulary (OOV) word, the errors in the OCR output text corresponding to such a word could be corrected based on the sub-word units derived from another vocabulary word ("couldn't"). This language phenomenon forms the basis of our work.

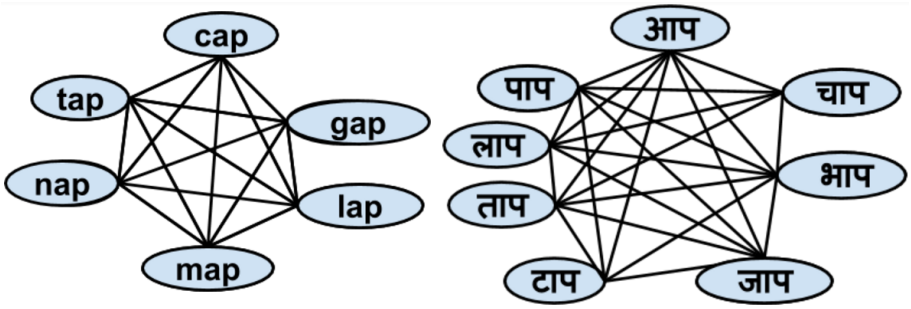


Figure 1.3: Sample words at unit distance away from each other in English & Hindi

Difficulties in developing spell-checkers for Hindi, Bengali & English involving *real-word errors* (RWE) and *non-word errors* (NWE) are discussed by Choudhury *et al.*, 2007. An RWE occurs when a valid language word is misrecognized as another valid language word, e.g. "their" misrecognized as "there". An NWE occurs when a valid language word is misrecognized as an invalid word, e.g. "team" misrecognized as "tcam". Intuitively, Indic languages with a high degree of conjoining rules result in a high percentage of non-word errors in the OCR texts. Now we discuss how even the Indic languages with a low degree of conjoining rules are likely to be associated with a higher fraction of real-word errors, as compared to English, in the OCR texts. As reported by Choudhury *et al.* (2007), it is more complicated to create a spell checker for Hindi and Bengali as compared to English due to a higher average node degree within the spell-net graph for in the former languages with respect to the latter. Spell-net is a weighted graph where words of the language are at the nodes and the weight of each edge is edit distance between the word pair it connects. Sample words at a unit distance away from each other in English and Hindi are shown in Figure 1.3 for intuition. The higher number of word ambiguities occur in Indian languages because they exhibit a larger number of prefixes and suffixes or root word forms as compared to English.

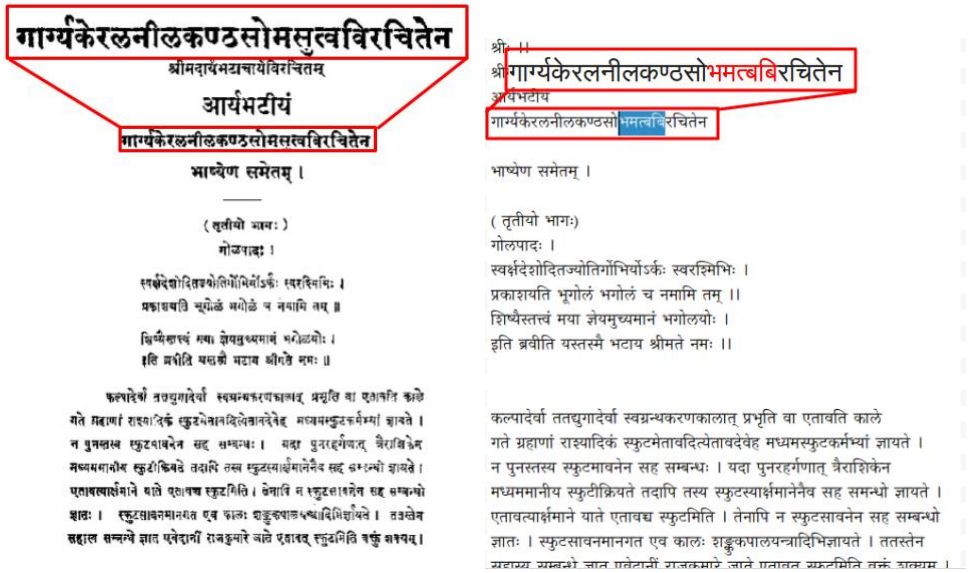


Figure 1.4: An image (left) and its OCR output (right)

1.5 Monolithic document collections

A document image in Sanskrit and its OCR output from *ind.senz* is shown in Figure 1.4. Here one of the word text is highlighted on the right side of the image to emphasize the presence of errors (shown in red) that occur in the OCR texts. As discussed in the previous section, such errors are most prominent in Sanskrit as compared to other Indic Languages.

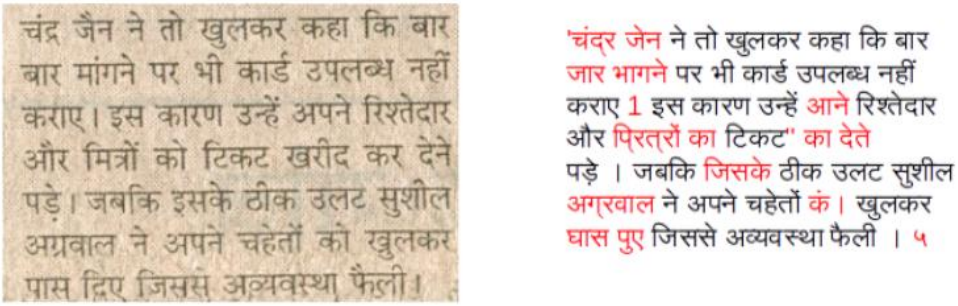


Figure 1.5: Hindi newspaper crop (left) and its OCR output (right)

An example of a Hindi newspaper crop and its output from *Tesseract (2020)* is shown in Figure 1.5. While the accuracy of OCR engines has improved for popular languages like Hindi, nevertheless the degradation like that in newspapers is still difficult to handle by the modern OCR systems.

As it can be observed from the above example, the errors in Indic OCR texts appear due to language characteristics as well as degradation in the document images. Moreover, the OCR pipeline discussed in Section 1.1 has many stages that

can benefit from interactive intervention, especially for large monolithic document collections. By "monolithic" we mean documents typeset uniformly with the same font and layout. We observe that a large proportion of printed historical documents individually exhibit nearly uniform font and language characteristics. Thus, we refer to them as large monolithic document collections. On such documents, an interactive framework that learns from human corrections and adapts from the repetitive patterns in the OCR texts (like OCR and document specific confusions and dynamic language resources) is desirable.

S.No.	Project/Work	Total Work
1.	Sandhi project at HSS Dept. - IIT Bombay (Sanskrit)	3405 pages
2.	NVLI Govt. project (English + Hindi)	4500 pages
3.	NVLI Govt. project (Marathi)	2500 pages
4.	Jain manuscripts (Hindi)	1000 pages
5.	Hindi edition of of Dharampal books[1]	1360 pages

Table 1.1: Different Projects where in OpenOCRCorrect is used

We work on five different digitization projects in different Indian Languages (and English) as shown in Table 1.1. The table also includes the number of pages we have digitized using OpenOCRCorrect discussed in Sections 1.7 and 4.1. As shown, over $12k$ document images have been corrected in different languages (Sanskrit, English, Marathi, and Hindi) using the system.

1.6 Prominent OCRs for Indian languages

In this section, we investigate three prominent OCR systems for the documents in different Indian languages. We also discuss the existing works on reading the texts in Indian street signs.

1.6.1 OCR systems for Indic texts

In this Section, we discuss 3 different systems for document OCR: *ind.senz (2020)*, *Google (2020) free OCR* and *Tesseract (2020)*.

[1] `https://archive.org/details/DharampalCollectedWritingsIn5Volumes`

Book Name with Publication Year	Publisher Details	# OCR Pages	ind.senz WER	Google WER	Tess. WER
Raghuvaṃśam Sanjīvinīsametam 1929	Nirṇaya Sāgara Press, Mumbai	200	19	35	44
Nṛsiṃhapūrvottar--atāpanīyopaniṣat 1929	Ānandāśrama, Pune	160	34	41	52
Siddhānta Śekhara-1 1932	Calcutta Univ. Press	390	40	66	71
Siddhānta Śekhara-2 1947	Calcutta Univ. Press	241	61	53	81
GaṇakaTaraṅgiṇī 1933	Jyotish Prakash Press, Varanasi	150	34	46	65
Siddhānta Śiromaṇi 1981	Sampurananda Univ., Varanasi	596	18	29	51
Āryabhaṭīyabhāṣya of Nīlkaṇṭha III, Golapāda 1957	Anantaśayana Saṃskṛta Granthāvaliḥ	177	26	45	77

Table 1.2: Word Error Rates (WER) on 7 Sanskrit books for 3 different OCR systems: *ind.senz*, *Google OCR* and *Tesseract*

Ind.senz

Ind.senz is a commercial OCR system available for accurate and fast digitization of Hindi, Marathi, Gujarati, Tamil, and Sanskrit (ind.senz, 2020). We obtain the commercial version of Sasnkrit OCR and tested it for seven different books for which the WERs are shown in Table 1.2. As shown, *ind.senz* performs better than *Google free OCR* on 6/7 books, which in turn performs better than the *Tesseract*. Since we start the research in around 2017, we work on the versions of these three systems available during that time.

Language	No. of Word Images	μ, σ of Word Length	WER for Google OCR
Sanskrit	86 k	10.22, 7.98	51.20
Malayalam	107 k	9.32, 4.93	38.32
Kannada	118 k	8.42, 3.86	47.44
Hindi[1]	67 k	5.29, 2.53	46.80

Table 1.3: Word Error Rates (WER) of *Google OCR* for different Indian languages

[1]We obtain Hindi dataset from Vinitha and Jawahar, 2016. Its predictions might not have come from *Google free OCR*. We still present its statistics here to compare the mean and standard deviation of word length with other languages.

Google free OCR

Google free OCR-service works for over 245+ languages (Google, 2020). We test it for images containing $86k$ Sanskrit, $81k$ Malayalam and $118k$ Kannada words from different documents. The word errors are given in table 1.3. Moreover, it is essential to note that the mean and standard deviation of word length is highest for Sanskrit and lowest for Kannada, which is in accordance with the vocabulary characteristics of these languages, as discussed in Section 1.4. Word Error Rates for *Google free OCR* on 7 different Sanskrit books is given in Table 1.2.

Language	**No. of chars (millions)**	**No. of words (millions)**	**Char error rate (%)**	**Word error rate(%)**
English	271	44	0.47	6.4
Italian	59	10	0.54	5.41
Russian	23	3.5	0.67	5.57
Simplified Chinese	0.25	0.17	2.52	6.29
Hebrew	0.16	0.03	3.2	10.58
Japanese	10	4.1	4.26	18.72
Vietnamese	0.41	0.09	5.06	19.39
Hindi	**2.1**	**0.41**	**6.43**	**28.62**
Thai	0.19	0.01	21.31	80.53

Table 1.4: *Tesseract* Word Error Rates (WER) reported by Smith, 2013

Tesseract

Tesseract (2020) is an open source OCR system that works for over 100 languages. The first three versions of *Tesseract OCR* are based on classical machine learning techniques. In these, the features based on the vectors obtained from character boundaries are extracted. A Gausian Mixture Model (GMM) based classifier is then used for identifying characters. The 4^{th} version of *Tesseract* adds an LSTM model, which directly works on line recognition. Word Error Rates for *Tesseract* on 7 different Sanskrit books are given in Table 1.2. The high word error rates occur due to dataset scarcity and a large percentage of OOV words in Sanskrit (as depicted by Figure 1.2). In Table 1.4, we present the accuracy of *Tesseract OCR* on Hindi, and several other non-Indic languages, as reported by Smith, 2013. Here, Smith (2013) claims that for Russian and most of the Latin-based languages, the parallel scans of books and PDF text layers were used to create the datasets. For other language styles, the datasets were created by typing the text from 10 consecutive pages chosen randomly from books scanned under the Google books project.

1.6.2 OCR systems for Indian street signs

We discussed the Word error rates for document OCR in the previous section. In this section, we discuss the (high) error rates for different models and (scarcity of) real world datasets available in the field of Indian scene-text recognition.

E2E-MLT

E2E-MLT is an unconstrained End-to-End method for Multi-Language scene Text recognition (Bušta *et al.*, 2018). The method is based on text localization, script identification, and text recognition. The model is trained and tested on 11 languages a.k.a Bengali, Latin, Arabic, Hangul, Chinese, Japanese, Korean, Hiragana, and Katakana, using datasets available from the ICDAR Multi-lingual Scene Text Detection and Script Identification-Robust Reading Competition (Nayef *et al.*, 2017). Bušta *et al.*, 2018 reports the word-error rate (WER) of 65.80% for Bengali (the only Indian Language in the dataset). In contrast, the WERs are below 35% for Latin (text as well as digits) and Hangul, and are above 53% for the remaining languages. The dataset contains $18k$ scene images, with $2k$ images per language (Nayef *et al.*, 2017).

CNN-RNN

The conventional encoder decoder based approach where CNN encodes the features from word images and RNN decodes them to produce text was trained on synthetic data and tested on around 1k real images in Devanagari, Telugu and Malayalam by Mathew *et al.*, 2017. Here the connectionist temporal classification (CTC) layer is used to align the RNN's output to labels during training. The WERs of 57.1%, 42.8%, and 26.6% are reported for Devanagari, Telugu, and Malayalam, respectively. The models were trained only on synthetic data, and for testing the text-boxes were annotated manually. Moreover, it is important to note that for end-to-end systems, such CNN-RNN based approaches additionally require a detector to localize word boxes, which adds segmentation errors as discussed in Section 1.2.3. This further increases the WER for such systems. Moreover, Mathew *et al.*, 2017 uses around 1k real word images in Devanagari, Telugu, and Malayalam for testing the recognition models, which again shows that there was a scarcity of real datasets for Indian scene-text recognition.

OCR Word	Correct Word
ज्योतिःशास्त्रीवायकग्रन्थेषु	ज्योतिःशास्त्रीयविषयकग्रन्थेषु
ठिपका-कबडसा	ठिपका-कवडसा
वगक्तिर	वर्गाकार
Niruki«	Nirukta

Figure 1.6: Examples of OCR words corrected by our framework

1.7 Motivation

Now that we have discussed the datasets and error rates for reading texts in Indian documents and scenes, we motivate the necessity of i) an interactive system for OCR corrections in Sections 1.7.1 and 1.7.2, and ii) improving OCR for Indian Street Signs in Section 1.7.3.

1.7.1 Introducing OpenOCRCorrect

As discussed in Section 1.1, OCR is the process of converting document images to editable electronic format. The problem is that OCR text has errors as presented in Section 1.6. To make the OCR output text usable we need to correct such errors. Manual correction of OCR documents is very cumbersome as the user has to go through the complete text, detect errors and correct them. While investigating the process of OCR correction, we came across many examples of Indic OCR text which were too hard for various online spell-checkers to verify and/or correct. The suggestions generated were very far from the truth for most of the words. The major reasons for this were:

1. The vocabulary was highly incomplete and limited, as discussed earlier in Section 1.4.

2. Limited implementation of language-specific rules for verifying correct words.

3. The design for such spell-checkers is based on typing errors, and not on frequent OCR n-gram confusions.

4. Ambiguities in suggestions from dictionary due to large number of basic word forms as discussed in Section 1.4.

Some of the complex Sanskrit OCR words corrected by our framework are shown in Table 1.5. In Figure 1.6, we show examples of OCR words in Sanskrit,

OCR Word	Correct Word
विशीआआआआर्‌नि	विशीर्णानि
षष्ठ्े ।पनिषत्ि९	षष्ठोपनिषदि
भर्तुर्मुनिराआस्थ्ूएतविष्टरः	भर्तुर्मुनिरास्थितविष्टरः
मङ्गलस्तनिस्वनाः	मङ्गलतूर्यनिस्वनाः
म ातपक्रं	महाचक्रं
नै९वाप्ति०इषच्यते	हैवाभिषिच्यते
सूऊउगव्ू ान्य्ुऋउScऊ	भगवान्यश्च

Table 1.5: Complex Sanskrit OCR words corrected by our framework

Marathi, Hindi and English (top to bottom), which have no suggested correction from popular engines/spell-checkers. Errors are marked in red and corrections in green. We are able to provide correction suggestions for such complex words through the system that we briefly discuss here and then elaborate upon it in Chapter 4. Correcting the errors is even more of a problem for Indian languages since the average typing speed of professional Indian typists is much lower than the average typing speed of a professional typist in English, which is 75 words per minute (WPM) at a word error rate of roughly 0.5% (Vorbeck *et al.*, 2000). The reason for the difference in typing speeds is that the keyboard is designed for a total of 26 characters whereas there are over 50 characters in Sanskrit. Moreover, the average length of a word in Indian languages (as can be observed in Table 1.3 and Figure 1.7) is much longer than English (which has the average word length of around five characters) due to conjoining words which make typing a more difficult task for curating errors.

Thus, any OCR system with accuracy below 90% is not sufficiently useful unless complemented by a partially automated mechanism for error detection and correction. Error detection can be considered a very important step in post-processing OCR words. On large monolithic documents, it may be further desirable that the error detection and correction system be able to continuously improve itself by incorporating user feedback.

We present an interactive framework for OCR corrections, which we call OpenOCRCorrect, for building up more OCR specific datasets in Indian documents. A screenshot of OpenOCRCorrect is shown in Figure 1.7. As shown, the framework exploits different kinds of i) static auxiliary sources like multi-OCR texts, language dictionary, and ii) dynamic auxiliary sources like domain dictionary and OCR confusions which are updated on-the-fly with user corrections. The details about

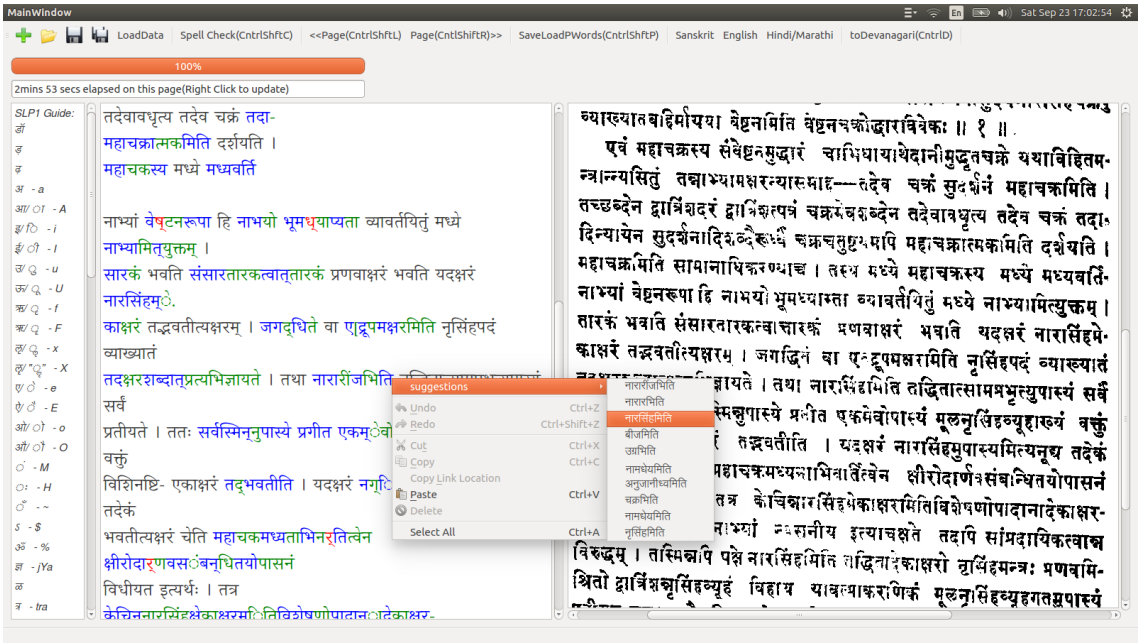


Figure 1.7: A screen shot of OpenOCRCOrrect

auxiliary sources, correction mechanism and the color coding used in the framework are given in Chapter 4. The source code of OpenOCRCorrect is available at `http://tinyurl.com/y9lms89u`.

Peformance measures for error detection and correction

The measures used for evaluating the task of error detection are defined below:

1. True Positives (TP) or Typing/Suggestion Efforts: Percentage of incorrect OCR words that are marked as incorrect by the error correction system. Such words would need typing corrections or suggestion selection (if suggestions are available).

2. True Negatives (TN) or Verification Gain: Percentage of correct OCR words that are marked as correct by the system.

3. False Positives (FP) or Verification Efforts by the error correction system.

4. False Negatives (FN) or Unavoidable Accuracy Loss: Since the user tends to ignore such incorrect words, that are marked as correct, they lead to unavoidable accuracy loss.

5. Precision: TP/(TP+FP), Recall: TP/(TP+FN) and F-Score: Harmonic mean of Precision and Recall.

There is always a trade-off between true positives and true negatives, though its degree might depend on the model. Use of conventional dictionary based methods increases the TP but lowers the TN. The reason is that a system that marks most incorrect words as incorrect also tends to mark several correct words as incorrect. Similarly, a conjoining rule-based method for increasing TN lowers TP. This trade-off can be captured through the F-Score. Maximizing the F-Score tends to yield models that are balanced in both these measures. The task of error correction is evaluated in terms of word error rates (WER) and/or Character error rates (CER). For the datasets containing words between $10k$ and $50k$, our framework leverages sub-word level information and multi-OCR consensus as a baseline for error corrections. For slightly better results, we recommend an adaptive plug-in classifier for identifying errors in Chapter 4. For datasets containing around $100k$ OCR words, we use the LSTM model with a *fixed sequential delay* in Chapter 5. In this section, we further improve the correction results by augmenting the LSTM's input with different sub-word encodings. We then recommend LSTM based encoder-decoder models with an attention mechanism for the datasets containing around $1000k$ OCR words at the end of Chapter 5. Such large datasets are available in the post-OCR competitions about which we discuss in the next section.

1.7.2 Post-OCR competitions in non-Indic languages

OCR accuracy affects the indexing and usability of digitized documents. In recent times, consistent improvement of OCR systems has resulted in high accuracy on modern document images. In contrast, many historical documents sitting on the racks of libraries still result in poor OCR accuracy. As a matter of fact, ancient publications with complex layouts and different types of preservation such as old newspapers still hold out against state-of-the-art OCR technology. Thus, the process of post OCR corrections on old digitized texts or on the latest complex documents could greatly help the community.

The importance of post-OCR text correction has been emphasized in literature Kissos and Dershowitz, 2016; Evershed and Fitch, 2014 and is further highlighted by the introduction of the competitions for comparing systems for such corrections in English, French, German, Finish, Spanish, Dutch, Czech, Bulgarian, Slovak and Polish documents ICDAR, 2017, 2019. Such competitions invite research scholars for the correction of over 12 million OCR characters. Two independent tasks are proposed.

- Error Detection: This includes the detection of error position and length in the raw OCR texts.
- Error Correction: This task involves generating a suggested correction (or ranked list of such corrections), for each incorrect word in given OCR texts.

Corrections scores for individual languages are calculated to enable language based techniques. Our team "Character Level Attention Model" (CLAM) participated in both post-OCR competitions (ICDAR 2017 and 2019) and achieved overall second position in both of them. This is discussed with details in Section 5.3.

Figure 1.8: On the left is a simple image[2] in Bengali (and English), on the right is E2E-MLT output

1.7.3 Reading modern Indian street scenes

We present a simple street sign image[1] containing a mixture of Bengali and English, and its OCR output using End-to-End method for Multi-Language scene-text recognition (E2E-MLT) by Bušta *et al.* (2018) in Figure 1.8. The image has uniform and large-sized (readable) characters, nearly orthogonal projection, empty background, and readable text. We arrange the OCR text in natural reading order, *i.e.*, from top to bottom, followed by a left to right. We mark the errors in red. As shown, there are many segmentation errors (in all the lines) in the OCR text in addition to missing characters (in Bengali), which leads to extra noise characters (in English).

Reading the text in modern street signs generally involves detecting the boxes around each word in the street signs and then recognizing the text in each box. Reading street signs is challenging because they often appear in a variety of scripts, font styles, and orientations. Reading the end-to-end text in scenes has an advantage of utilizing the global context in street signs or multi-line license plates, which

[2]Image courtesy: `flickr.com/photos/rizwanoola/279104335`.

enhances the learning of patterns. One of the important factors that separates a character level OCR system from an end-to-end OCR system is reading order. Attention is thus needed to i) locate the initial characters, read them and ii) keep the track of the correct reading order in form of change in characters, words, lines, paragraphs or columns (in multi-column texts). This observation forms the motivation for our work in this area.

Obtaining large scale multi-frame video annotations is a challenging problem due to unreliable OCR systems as well as expensive human efforts. The predictions obtained on videos by most OCR systems are fluctuating, as can be observed in `https://youtu.be/VcNSQGO0j7s` for a text-spotter proposed by Bušta *et al.*, 2017. We also present fluctuating prediction accuracies for three OCR systems in Figure 7.2). The fluctuations in the accuracy of the extracted text may also be due to various external factors such as partial occlusions, motion blur, complex font types, distant text in the videos. Thus, such OCR outputs are not reliable for downstream applications such as surveillance, traffic law enforcement and cross-border security system. We present a framework for correcting complex license plate patterns in street videos, that we call as StreetOCRCorrect. The high-quality output obtained from such a framework can be used to prepare a large database. Such a database can enable new applications like reliable text based search systems, analytic dashboards, traffic flow monitoring, etc. Such a database can also be used to continuously improve the OCR models.

1.8 Publications and Awards

We have the following publications for the multi-lingual OCR work:

1. Saluja, Rohit, Ayush Maheshwari, Ganesh Ramakrishnan, Parag Chaudhuri, and Mark Carman. “OCR On-the-Go: Robust End-to-end Systems for Reading License Plates and Street Signs.”, In International Conference on Document Analysis and Recognition (ICDAR), pp. 160-165. IEEE, 2019.

2. Saluja, Rohit, Mayur Punjabi, Mark Carman, Ganesh Ramakrishnan, and Parag Chaudhuri. “Sub-word Embeddings for OCR Corrections in Highly Fusional Indic Languages.” In International Conference on Document Analysis and Recognition (ICDAR), pp. 160-165. IEEE, 2019.

3. Singh, Pankaj, Bhavya Patwa, Rohit Saluja, Ganesh Ramakrishnan, and Parag Chaudhuri. “StreetOCRCorrect: An Interactive Framework for OCR

Corrections in Chaotic Indian Street Videos." In 2nd ICDAR Workshop on Open Services and Tools for Document Analysis (ICDAR-OST), ICDARW, vol. 2, pp. 36-40. IEEE, 2019.

4. Adiga, Devaraj, Rohit Saluja, Vaibhav Agrawal, Ganesh Ramakrishnan, Parag Chaudhuri, K. Ramasubramanian, and Malhar Kulkarni. "Improving the learnability of classifiers for Sanskrit OCR corrections." In The 17th World Sanskrit Conference (WSC), Vancouver, Canada. IASS. 2018.

5. Saluja, Rohit, Devaraj Adiga, Parag Chaudhuri, Ganesh Ramakrishnan, and Mark Carman. "Error Detection and Corrections in Indic OCR using LSTMs." In 14th IAPR International Conference on Document Analysis and Recognition (ICDAR), vol. 1, pp. 17-22. IEEE, 2017.

6. Saluja, Rohit, Devaraj Adiga, Ganesh Ramakrishnan, Parag Chaudhuri, and Mark Carman. "A Framework for Document Specific Error Detection and Corrections in Indic OCR." In 1st ICDAR Workshop on Open Services and Tools for Document Analysis (ICDAR-OST), ICDAR, vol. 4, pp. 25-30. IEEE, 2017.

We have also received the following awards for the work:-

1. Our team, Character Level Attention Models (CLAM) achieved the 2^{nd} highest percentage of corrections in the ICDAR 2019 post-OCR Competition on 10 languages as reported in Rigaud *et al.*, 2019.

    - We achieved the highest correction accuracy of 44% in Finnish (similar to Sanskrit in inflections as advocated by Sommer, 2016), which is significantly higher than the overall winner (8% in Finnish).
    - CLAM also achieved the 2^{nd} position in ICDAR 2017 post-OCR Competition on English and French documents as per Chiron *et al.*, 2017a.

2. Ganesh Ramakrishnan and Parag Chaudhuri. "Development of an adaptive framework for end-to-end corrections in Indic OCR" IIT Bombay Impactful Research Award, 2017 (received in 2018).

3. Rohit Saluja, Devaraj Adiga, Parag Chaudhuri, Ganesh Ramakrishnan and Mark Carman. "A Framework for Error Detection and Corrections in Sanskrit", Research and Innovation Symposium in Computing (RISC) 2017 (Most Admired Poster Presentation Award), IIT-Bombay, India.

1.9 Contribution

The key contributions of this thesis area as follows:

1. We present OpenOCRCorrect (Saluja *et al.*, 2017b; Adiga *et al.*, 2018), an interactive framework for end-to-end corrections in Indic OCR. Such a framework exploits multiple static and dynamic auxiliary sources to correct OCR errors. We then present an adaptive plug-in classifier as a better error detector. We show that sub-word level information, derived from background knowledge or training data, can help to identify OCR errors in Indic languages. Combining partial word forms, while taking care of OCR specific confusions and conjoining rules, improves the accuracy of suggested corrections.

2. We propose a new LSTM-based error correction model that jointly learns the language as well as the OCR patterns (Saluja *et al.*, 2017a). This approach tackles the problems of error detection and corrections at the same time for Indic OCR. We demonstrate that augmenting the input of LSTM models with the frequencies of OCR sub-words in the language data performs as well as using fastText embeddings (trained on the same data) for correcting Indic OCR (Saluja *et al.* (2019b)). Here, we propose a new procedure for training fastText on sub-word units present in the constant length words (refer Section 5.2.1). This involves the transformation of language data in such a way that it not only includes all the substrings within the language but also retains the character level context of substrings in the language. This method is shown to improve the accuracy and speed (see Fig. 5.9) for error correction in Indic OCR as compared to the state-of-the-art.

3. We present a character level attention model (CLAM), using which we achieve the highest performance for Finnish at the ICDAR 2019 post-OCR competition by Rigaud *et al.*, 2019.

4. For scene-text recognition, we present the first results for multi-head attention models on the task, and demonstrate that each head has unique coverage over the scenes (Saluja *et al.*, 2019a). We design a new methodology for labelling a large amount of training data, that allows us to work with over 1 million real scene images containing license plates. We also release a new multi-lingual scene-text data-set of $1k$ videos (with text in Hindi, Marathi, and English), each covering an Indian street sign from different orientations. The multi-lingual data-set and supplementary material for this work can be requested

from `https://www.cse.iitb.ac.in/~rohitsaluja/project`. The source code is available at `https://github.com/rohitsaluja22/OCR-On-the-go`. We present StreetOCRCorrect (Singh *et al.*, 2019): a novel framework for OCR corrections in chaotic street videos. The modular framework is available at `https://github.com/rohitsaluja22/StreetOCRCorrect`. In this framework, we simplify the task of correcting multiple predictions of a vehicle in videos.

5. Finally, we present a CATALIST[3] model that 'tames' the attention (heads). We provide supervision to the attention masks at multiple levels, *i.e.*, line, word and character levels while training the multi-head attention model. We demonstrate that such supervision improves training performance and testing accuracy. To train CATALIST and its attention masks, we also present a synthetic data generator ALCHEMIST[4] that enables the synthetic creation of large scene-text video datasets, along with mask information at character, word, and line levels. We release real scene-text dataset of $2k$ videos, CATALIST_d with videos of real scenes that contain scene-text potentially in a combination of three different languages, namely, English, Hindi, and Marathi. We record these videos using 5 types of camera transformations - (i) *translation*, (ii) *roll*, (iii) *tilt*, (iv) *pan*, and (v) *zoom* to create transformed videos.

In this chapter, we have explained the basic process for Optical Character Recognition with respect to Indian texts and street signs. We have motivated the work via a need for i) the framework to correct Indic OCR document text and ii) improving the models for reading Indian street signs. We have further reported the publications and awards we have received for the work. The chapter ends with our key contributions in the field. We examine the generic as well as Indic literature related to document and photo OCR in the next chapter.

[3]CATALIST stands for **CA**mera **T**r**A**nsformations for multi-**LI**ngual **S**cene **T**ext recognition.

[4]ALCHEMIST stands for synthetic video generation in order to tame **A**ttention for **L**anguage (line, word, character, *etc.*) and other camera-**CH**ang**E**s and co**M**binat**I**ons for **S**cene **T**ext.

Chapter 2

Literature Survey

We now turn attention towards the general trends in the field of the document and photo OCR. We begin with a discussion on the history of document OCR in general in Section 2.1. Then we elaborate upon the trends in Indic OCR and works on Indic OCR corrections in Section 2.1.3. We finally discuss about the literature on photo OCR and license plate recognition in Section 2.2. We then conclude the chapter with a short summary in Section 2.3.

2.1 History of document OCR

In this section, we discuss the history of document OCR from early hardware-based to recent software-based techniques, and then investigate the trends in Indic OCR.

2.1.1 Hardware-based techniques

As noted by Mori *et al.* (1992); Sharma and Chaudhary (2013), the idea of Optical Character Recognition (OCR) appeared even before electronic computers. Gustav (1938) obtained a patent from Germany in 1929 and later from the US in 1935. This work employed a photoelectric device to read digit images. Handel (1933) also used a photoelectric device to match templates such as light through stencil digits with digit images. Vertical projections of the ink pixels within the segmented slit of characters were used by Glauberman (1956) for the reduction in complexity of such initial template matching techniques. As reported by Sharma and Chaudhary (2013), in 1949 RCA engineers operated on a computer-type OCR to assist the visually impaired for magazine subscription at the US Veterans Administration. Here,

the typewritten documents were converted into punched cards for input into a computer, which helped in processing around 15 – 20 million books a year. In 1962, RCA (W. J. Hannan and Wemer, 1962) released its first multi-font (English and Russian fonts) reading machine with 91 channels. Extensive research based on using autocorrelation to recognize the image patterns was carried out by Horwitz and Shelton (1961) from IBM in 1961. Under statistical decision methods, the number of ink pixels that (a fixed set of) six different lines cross in a character image was used to obtain a binary pattern for character recognition (Duda *et al.*, 1973). Goodrich *et al.* (1979) presented an efficient OCR system with an improved reading speed from about 150 to 250 words per minute and also added a new voice system to help visually impaired people. From 1984 and 1994, *Tesseract* was developed by Smith (2007) at HP Labs, Bristol, as hardware-assisted software till 1988 and then onwards as PC software till 1994.

2.1.2 Software-based techniques

Tesseract outperformed several commercial systems at UNLV 1995 Annual Test of OCR Accuracy (Rice *et al.*, 1995). Since 2005, the source code of *Tesseract* is freely available[1]. Meanwhile, Jenkins *et al.* (1993) established the baselines on data of ideal images with the three different point sizes (10, 12, and 14) and fonts (Courier, Helvetica, and Times-Roman). This work employed eight commercial devices (with concealed OCR algorithms) and established the best character accuracy ranging from 99.21% to 99, 95%. Avi-Itzhak *et al.* (1995) used a simple neural network with centroid dithering to recognize multi-size and multi-font character images with extreme accuracy. Here a Support Vector Machine(SVM) is proposed for classifying the character images. Lopresti (2009) presented an interesting analysis of different types of OCR errors. The character level recognition methods discussed until now, however, suffer from segmentation errors (Kameshiro *et al.*, 1999). To avoid such errors, some of the techniques rely on recognizing word images directly. Frinken *et al.* (2010b) used Bidirectional Long Short Term Memory (BLSTM) with projection-based features obtained from each word image to read its characters. Frinken *et al.* (2010a) also used BLSTM for spotting words in handwritten documents. Several other works related to the field of document OCR are Layout Recognition or Analysis (Esposito *et al.*, 1990; Watanabe *et al.*, 1995; Shotton *et al.*, 2009; Zhong *et al.*, 2019), Structure Extraction (Doucet *et al.*, 2011, 2013, 2017), Table Structure Recognition (Kieninger, 1998; Hu *et al.*, 2000; Raja *et al.*, 2020), and Text Detection (Nikitin

[1]now at `http://code.google.com/p/tesseract-ocr`

et al., 2019; Melnikov and Zagaynov, 2020), which are not under the scope of this thesis.

As explained in the Sections 1.2 and 1.6, OCR output text contains a variety of errors owing to image degradation, misclassification, and segmentation at various levels. To make OCR output useful, such errors need to be either corrected by humans or automatically. In the field of text correction in general, there have been several attempts at using dictionaries to fix the errors (Kukich, 1992; Bassil and Alwani, 2012; Carlson and Fette, 2007). The methods are not reliable for languages with agglutinations and fusions as their performance suffers due to the OOV problem. Approaches that perform corrections based on context n-grams are more effective. The work by Wilcox-O'Hearn *et al.* (2008) used a trigram-based noisy-channel model. Golding and Schabes (1996) explored Bayesian methods based on part-of-speech trigrams for corrections based on context. The work by Smith (2011) further concluded that noisy-channel models that closely model the underlying classifier and segmentation errors are required by OCR systems to improve performance.

The recent learning techniques by Afli *et al.* (2016); Mei *et al.* (2016); Kissos and Dershowitz (2016) are proven to be effective in generating suggestions for the OCR errors.a major online platform: The work by Chiron *et al.* (2017b) introduces the OCR error model to predict the relative risk of mismatch targeted terms for around $80M$ search queries, which are made every year over a National Library of France. Nguyen *et al.* (2019) give various tips for practical post-OCR approaches, by presenting some exciting insights on four English public datasets by Evershed and Fitch (2014); Chiron *et al.* (2017a). Recently, a publication by Van Strien *et al.* (2020) studies the impact of OCR errors on various NLP tasks like sentence segmentation, dependency parsing, information retrieval, *etc.*

2.1.3 Trends in Indic OCR

In the last thirty years, there have been efforts to improve OCR systems for printed Indic scripts as presented by Govindaraju and Setlur, 2009. Handcrafted features were part of earlier works (Govindan and Shivaprasad, 1990; Chaudhuri and Pal, 1997). Initial efforts by Pal and Chaudhuri (2004) relied on character level segmentation and recognition. Subsequently, there were significant data accumulation and annotation attempts (Bhaskarabhatla *et al.*, 2004; Kumar and Jawahar, 2007; Govindaraju and Setlur, 2009; Krishnan *et al.*, 2014). Arya *et al.* (2011) improved the quality of the OCR system for multiple Indian languages by using Support Vec-

tor Machine (SVM) classifiers and script grammar. The research focus then moved towards word-level approaches (Natarajan *et al.*, 2005; Sankar K *et al.*, 2010; Dutta *et al.*, 2012) in order to avoid the character level segmentation issues discussed in Section 1.2.3. The recognition of Devanagari word images based on a neural network has been explored by Jain *et al.* (2011) and Sankaran and Jawahar (2012). Sankaran and Jawahar (2013) used the Devanagari specific segmentation free approach at (Unicode) text level to improve performance. Singh and Jawahar (2015) demonstrated performance gains by training the neural networks directly on line-level images. Recently Biswas *et al.* (2018); Paul and Chaudhuri (2019) also used the line-level approach for improving the Bengali OCR systems. A recent work by Das and Jawahar (2020) applies limited supervision for adapting OCR to reduce the domain gap between training data and real books.

The conventional spell-checkers by Whitelaw *et al.* (2009); Hanov (2013); Norvig (2011) make use of proximity-based matches, especially edit distance (by Damerau, 1964 and Levenshtein, 1966) to words from a known vocabulary (possibly gathered from the web), followed by a language model for auto-corrections. The various difficulties involved in developing a high-performing spellchecker for Hindi, Bengali and English are discussed in Choudhury *et al.*, 2007. Here, an example is given that "fun" being misspelt as "gun" is a real-word error (RWE) and "fun" being misspelt as "vun" is a non-word error (NWE). An observation is made by Choudhury *et al.* (2007), that "hardness of NWE correction is highest for Hindi, followed by Bengali and English". Intuitively, the larger the number of basic word forms that exist in a language, the more candidates there are for replacing each erroneous word and the harder it is to build a functioning spell-checker. Choudhury *et al.* (2007) also present the complexities involved in creating effective spell-checkers for Hindi, Bangla and English in terms of spell-nets. A spell-net is a graph with words as nodes and edit-distance as the weights of its edges. The spell-net for Hindi and Bengali has a higher average degree as compared to English. This leads to a higher fraction of inter-word ambiguities in Indic languages. Moreover, in the spell-nets for Indic languages, the correlation between degrees of neighbouring words (nodes) is higher than English. This makes it more difficult to rank the candidate suggestions for an incorrect word whenever the word has a high degree in the spell-net. Recent work on neural language correction by Xie *et al.* (2016) has shown the benefits of using Recurrent Neural Networks (RNN) for the purpose of correcting the text. We use a Long Short Term Memory (LSTM) model in one of the work (Saluja *et al.*, 2017a) for post-OCR corrections. LSTM models can remember longer term context

of the input sequence and might, therefore, be quite successful in correcting OCR induced errors. As discussed by Sankaran and Jawahar (2013), error detection for Indic languages presents singular challenges such as large unique word lists, lack of linguistic resources and lack of reliable language models. There are many examples of work in the literature that focus on post-OCR corrections for specific Indian languages. The techniques used by Pal *et al.* (2000) include morphological parsing for Bangla; shape-based statistics were utilized for Gurmukhi by Lehal *et al.* (2001), and a multi-stage graph module with a sub-character model for Malayalam was used by Nair and Jawahar, 2010.

By using a Support Vector Machine (SVM, proposed by Bennett and Demiriz, 1999) classifier, Sankaran and Jawahar (2013) outperform the conventional lookup technique for detecting Indic OCR errors. Vinitha and Jawahar (2016) further improve the results by using the gaussian mixture models (GMMs) and RNNs. We set new benchmarks for the tasks of error detection and corrections in Indic OCR using LSTMs (Saluja *et al.*, 2017a). A recent work presents a copying mechanism for post-OCR corrections in romanised Sanskrit (Krishna *et al.*, 2018). Recently (Saluja *et al.*, 2019b), we improve the performance of previous state-of-the-art model (Saluja *et al.*, 2017a) by the use of sub-word embeddings (refer to Section 5.2). Further, the post-OCR competitions by ICDAR (2017, 2019) highlight the importance of such works. We use attention-based character level encoder-decoder models to achieve the overall 2^{nd} highest corrections in both these competitions. The winners of these competitions also use similar models (with slight variations that we discuss in Section 5.3). Collectively correcting the OCR text of a cluster of similar images is proposed to reduce 70% of human efforts in a recent work by Das *et al.* (2019) on a large number of English and Hindi documents.

2.2 History of photo OCR

Applications of Optical Character Recognition (OCR) are not limited to document images, but include also photos of street signs. We now introduce the approaches to tackle various issues in the field of photo OCR. Spotting text in scene images is typically performed in two steps, *viz.*, i) localization of the text within the image (*i.e.*, detection of word-level boxes), and ii) recognition of the text (*i.e.*, extracting the character sequence). Works specific to text localization are proposed by Gupta *et al.*, 2016. Liao *et al.* (2017); Minghui Liao and Bai (2018) augment such work to real-time detections in the end-to-end scenes. Better solutions in terms of accuracy and speed

are presented (Karatzas *et al.*, 2015; Bušta *et al.*, 2017). The problem of scene-text spotting, however, remains complicated owing to variations in illumination, capturing methods and weather conditions. For instance; state-of-the-art recall, precision, and F-measure scores on the COCO-Text dataset as reported by Veit *et al.*, 2016 are as low as 28.33, 68.42 and 40.07 respectively. Moreover, the movement of the camera (or objects containing text) and motion blur in videos can make it harder to recognize the scene-text correctly. There has been a rising interest in end-to-end scene-text recognition in images over the last decade (Bartz *et al.*, 2017; Shi *et al.*, 2017; Karatzas *et al.*, 2013, 2015; Bušta *et al.*, 2017). Recent text-spotters by Bušta *et al.*, 2017, 2018 include deep models that are trained end-to-end but with supervision at the level of text as well as at the level of words and text-boxes. The two recent breakthroughs in this direction, which work directly on complete scene images without supervision at the level of text boxes, are:

1. STN-OCR: A single neural network for text detection and text recognition by Bartz *et al.*, 2017. The model contains a spatial transformation network that encodes the input scene image. It then applies a recurrent model over the encoded image features to output a sequence of grids. Combining the grids and the input image returns the series of word images present in the scene. Another spatial transformer network process the word images for recognition. This work does not need supervision at the level of detection.

2. Attention-OCR by Wojna *et al.* (2017): This work employs an inception network (proposed by Szegedy *et al.*, 2016) as an encoder and an LSTM with attention as a decoder. The work is interesting because it does not involve any cropping of word images but works on the principle of soft segmentation through attention. The attention-OCR model performs character-level recognition directly on the complete scene image thus utilizing the global context while reading the scene. This model is an open-source TensorFlow (a popular library for deep learning by Abadi *et al.*, 2016) implementation.

These works both experiments on a French Street Name Signs (FSNS, an example is shown in Figure 6.3 (top)) dataset, on which Attention-OCR performs the best. The Attention-OCR model also outperforms another line-level segmentation based method (refer to work by Smith *et al.*, 2016) on the FSNS dataset. In Chapter 6, we will introduce a model that outperforms these models on the FSNS dataset using multi-head attention mechanism (Saluja *et al.*, 2019a). In this work, we also set new

benchmarks for the task of reading Indian street signs and license plates in a large number of video frames.

2.2.1 Trends in Automatic License Plate Recognition

We aim to address a more specific scene-text recognition problem in the Indian context, viz., correcting the license plates text in traffic videos. Automatic License Plate Recognition (ALPR) is an essential component of surveillance systems. The problem is especially challenging in the Indian context owing to conditions such as chaotic traffic, multi-line license plates with variable-sized characters, cursive/handwritten fonts, and non-rectangular as well as old-dusty license plates. It's even harder to recognize plates in the videos due to motion blur, varying scene/camera orientations and low-resolution cameras.

License plate recognition is generally performed in two steps: i) license plate detection, and ii) license plate recognition. Du *et al.* (2013); Jain *et al.* (2016); Li and Shen (2016) use features such as edge, texture, colour, *etc.* for license plate detection. However, such approaches suffer from the problem of an excess of false positives. Early works on character-level plate recognition utilize methods such as connectivity, projections, template matching, *etc.* (Yoon *et al.*, 2011; Nomura *et al.*, 2005; Zhang *et al.*, 2013; Rasheed *et al.*, 2012). These methods suffer from the problem of segmentation errors due to overlapping characters and confusions owing to visually similar n-grams. There have been efforts by Jiao *et al.* (2009) towards developing robust methods based on learning to overcome such errors.

The literature has moved gradually from character-based to sliding window-based methods and then finally to word-level recognition via neural networks. For example, Li and Shen (2016) recently use sliding window-based CNN classifiers and RNNs to recognize characters in the license plates. However, methods based on sliding windows are computationally expensive since they require the rejection of a large fraction of non-character images. Recent work on Indian license plate recognition by Jain *et al.* (2016) utilizes edge-based features. A specific CNN then discards the false positives in this work. The license plates are then cropped from the scenes and passed to another CNN for recognition. Further, a CNN-based character level recognizer and a spatial transformer layer to overcome variations in brightness and tilt are also employed. A recent tool by Lenc *et al.* (2019) for annotating a large amount of scene-text images addresses the rareness of such tools. We also develop a tool for annotating a large number of video frames (refer Singh *et al.*, 2019 and Section 6.9).

2.3 Summary

In this chapter, we investigated the existing literature in the field of document OCR and as well as Indic OCR. We then discussed the various works in the field of photo OCR. We motivated the move towards end-to-end systems for reading text in the field of photo OCR. We further discussed trends for the specific problems of reading license plates. In the next chapter, we will reflect upon different research questions related to the multi-lingual OCR systems for Indic texts and Indian street signs.

Chapter 3

Research Questions

Now that we have examined the process of Optical Character Recognition (OCR) in documents as well as scenes and various trends in these fields, we investigate different research questions associated with our work in this chapter. We begin by discussing questions related to generic OCR in Section 3.1, followed by the questions related to OCR on Indic texts and Indian street signs in Sections 3.2 and 3.3 respectively.

3.1 Generic OCR

In this section, we address the questions related to generic OCR systems.

Is it possible to improve the performance of generic OCR systems using sequence modelling or deep learning techniques?

Although reading image characters appears to be a simple object classification problem, the diversity in fonts and languages, cameras and/or scanners, adaptive language texts, *etc.* make the overall process complicated. Therefore, we need to investigate why classical machine learning techniques are not useful in developing robust reading systems? Are the independence assumptions in sequential modelling techniques such as Hidden Markov Models (HMM) (Rabiner and Juang, 1986), accurate for real OCR systems? Are Long Short Term Memory (LSTM) models (Sundermeyer *et al.*, 2012), that avoid such assumptions, more effective in sequentially modelling the tasks of correcting OCR texts and reading text images? We answer some of these questions in Sections 5.1, 5.2, 6.5 and 6.6. Some state-of-the-art deep text-spotters like the one proposed by Bušta *et al.* (2017, 2018) use convolution based networks (Krizhevsky *et al.*, 2012) for detecting and recognizing scene-texts to reduce the inference time. However, the sequence modelling techniques based on

LSTM (Wojna *et al.*, 2017) also solve the problem of photo OCR. The convolution-based text-spotters, in addition to some early works on sequence modelling, need an additional Connectionist Temporal Classification (CTC, refer Graves *et al.*, 2006) layer for aligning the text output with the sequence. The recent works by Wojna *et al.* (2017); Ly *et al.* (2019) avoid the need for a CTC layer by using an attention mechanism in their models. This leads to the questions: Are attention-based deep models more effective in reading the image text? Is it possible to read the characters in paragraph-level images directly to utilize the global context and avoid the segmentation errors that we discussed in Section 1.2.3? For answer to these questions, refer Sections 6.5, 6.6 and 7.2.

How to correct a large volume of OCR texts with minimal human efforts?

Any OCR system, even in an industrial setting is never 100% accurate on real-life images. As proposed by Lopresti and Zhou, 1997 and Abdulkader and Casey, 2009, multi-model consensus is thus employed to achieve high accuracies. Tasks such as reading text in medical images and re-publishing library books expect no compromise on text correctness. Thus we need to determine the best practices to efficiently correct the OCR text with minimal human efforts. How can we leverage uniform font and language properties in monolithic document collections for such OCR correction tasks? We elaborate upon this in Section 4.1.

3.2 Indic OCR texts

We now discuss the questions related to Indic OCR texts in this section.

What are the most effective OCR systems for reading Indic texts? Do we need to improve such OCR systems for making their outputs usable?

Different OCR systems train on different kind of datasets. Open-source systems such as *Tesseract* and *Google free OCR* generally work well for English documents. Such systems, however, might not work as effectively for various Indian languages. Another commercial system that we investigate is *ind.senz* (described in Section 1.6). Do we need to improve the quality of existing OCR systems for reading Indic texts? Can we directly improve the quality of Indic OCR texts by exploiting the OCR-specific or language-specific patterns? We answer these questions in Sections 4.2, 5.1, 5.1.4, 5.2, 5.2.6 and 5.3.

How to handle out-of-vocabulary problems in the inflectional Indian languages?

Owing to the complex conjoining rules, any off-the-shelf vocabulary in several Indian languages is generally incomplete as affirmed by Sankaran and Jawahar, 2013; Vinitha and Jawahar, 2016. How should one approach this problem in the context of resolving OCR errors? Is it possible to exploit such rules for correcting the OCR texts? For detailed answers to these questions, refer Sections 4.2.1, 4.3.2, 4.4.1.a, 4.4.1.c, 4.4.2, 5.2 and 5.2.6.

What are the various auxiliary sources that one can leverage to correct OCR texts?

As discussed in the previous section and Section 1.4, texts in Indian languages suffer from the problem of having a significant fraction of out-of-vocabulary words. Is it possible to tackle this problem in the context of OCR errors by exploiting as many auxiliary sources as available? We try to solve this problem in Section 4.2.

Is it possible to exploit LSTMs for resolving the Indic OCR errors?

As discussed in Section 3.1, Long Short Term Memory (LSTM) models avoid neighbourhood independence assumptions. Such models also are successful language models (Sundermeyer *et al.*, 2012). Therefore, is it vital to explore the feasibility of employing LSTMs or bidirectional LSTMs (BLSTMs) for Indic OCR corrections? We use such models in Sections 5.1 and 5.2.

Is it possible to use the sub-word embeddings for improving the OCR quality for Indian Languages?

An intuitive approach to tackle the problems related to out-of-vocabulary words is to use sub-word embeddings. We need to determine that which type of sub-word embeddings is best for Indian Languages. Can we further modify such embeddings or train them differently to better suit the task of OCR corrections? We explore the possible approaches in Section 5.2.

What are the different kinds of deep models that are most suitable for datasets of order i) $100k$ *OCR words, ii)* $1000k$ *OCR words?*

Is it always true that larger the depth of the deep networks, larger is the dataset required to train them? Is it true even when the networks being compared are different, e.g. when we compare Convolutional Neural Networks (CNNs, refer Krizhevsky *et al.*, 2012) with Recurrent Neural Networks (RNNs, refer Mikolov *et al.*, 2010)?

What are the best networks available for the OCR systems? Which one out of them is best for resource-constrained languages? Moreover, which of them is worth investigating for the languages with a large amount of training data? We investigate such questions and explore the answers in Chapter 5.

3.3 Reading Indian street signs

This section covers the questions specific to the OCR systems for reading Indian street signs.

Is it possible to avoid segmentation of text images at different levels of granularity for reading modern Indian street signs?

As discussed in Section 1.1, the OCR process typically involves the segmentation of the image into paragraphs, lines, words, and sometimes even characters. Is it possible to avoid the segmentation of these types or perform soft segmentation as humans do? We could thus bypass the segmentation errors if deep models with such capabilities are enabled. We discuss this in Chapter 6.

OCR systems could recognise entire words rather than characters, so why do existing photo OCR systems still read individual characters in the images?

Tasks like machine translation, scene captioning, *etc.* often use word embeddings for predictions. Still, for the task of reading scene-text images, the final output of all the state-of-the-art models is a sequence of characters. It is important and interesting to have the outputs of different applications in the same form to mimic multi-tasking humans. So, Why do we train the scene-text recognition models to predict the sequence of characters? Is the distinction between predicting the sequence of characters or directly predicting the series of words relevant in the context of Indian street signs because the Indian languages suffer from the out-of-vocabulary problem? To gain an insight into the answer to this question, refer Chapter 6.

As it will become clear from the subsequent chapters, one of the intriguing features of our work is that the models which we develop are robust to the different languages and in turn the wildness of the languages. Moreover, the improvement gains with respect to the state-of-the-art works present that our work is vigorous against different kinds of degradations discussed in each chapter. After posing various research questions related to Indic OCR texts and reading Indian street signs,

we now conclude this chapter. We will explore the answers to these questions in subsequent chapters. We begin with a detail discussion on interactive OCR corrections in the next chapter.

Chapter 4

Classical ML Techniques for OCR Corrections

As discussed in Section 1.6, OCR systems exhibit errors in the text they output. In this chapter, we answer some of the research questions discussed in the first two sections of the previous chapter. Specifically, we try to respond to the issues related to (i) minimal human efforts required to correct OCR texts, (ii) out-of-vocabulary problems in Indic OCR, and (iii) using various auxiliary sources while correcting Indic OCR texts.

We begin this chapter by presenting a framework for assisting word-level corrections in Indic OCR documents in Section 4.1. The framework learns globally as a user corrects errors locally. It helps to correct the OCR text by incorporating the ability to identify, segment, and combine partially correct word forms (using global auxiliary sources, which we describe in Section 4.2). We use such features to train a plug-in classifier (proposed by Narasimhan *et al.*, 2014), as explained in Section 4.2.2, for achieving better F-scores as compared to lookup approaches when detecting erroneous OCR words. We summarize initial error detection results in Section 4.3.1. Furthermore, such erroneous words are corrected using suggestions with human-in-the-loop to keep the confidence level high. We summarize initial results for suggestions generation in Section 4.3.2. In Section 4.3.3, we provide page-wise correction analysis for four different books. In Section 4.4, we improve the results of the plug-in classifier for Sanskrit with language-specific auxiliary sources. Some of these auxiliary sources exploit conjoining rules to synthesize valid word forms. We also present improvements due to the adaptive auxiliary sources used in our framework. We finally conclude the chapter in Section 4.5.

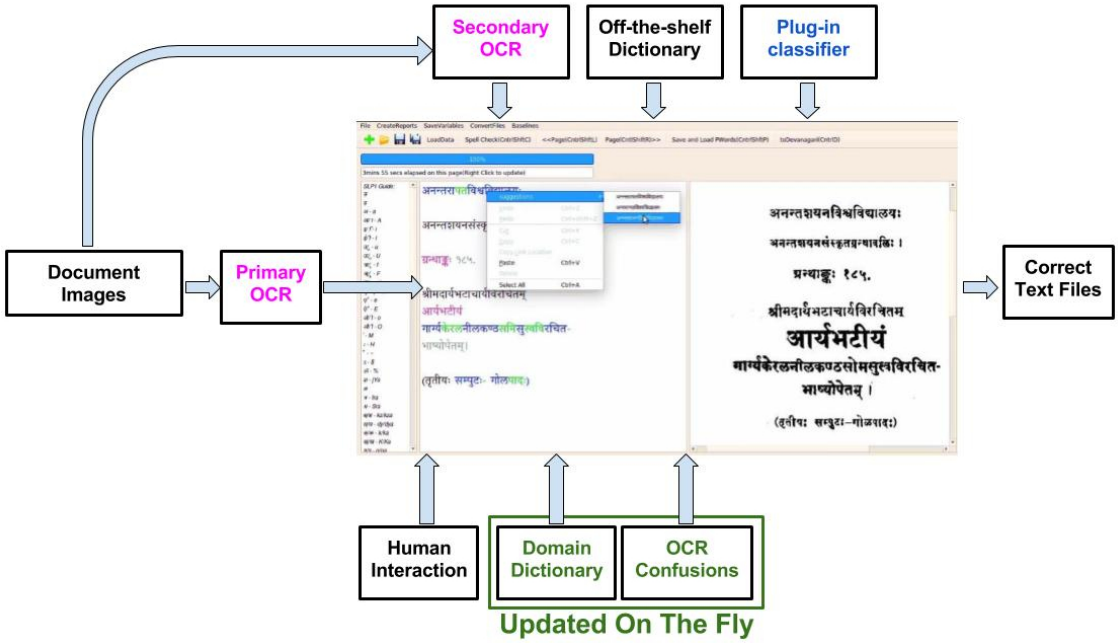


Figure 4.1: OpenOCRCorrect: Learn Globally Correct Locally

4.1 Interactive framework for OCR corrections

As discussed in Section 1.6, Word Error Rates (WER) of OCR systems on ancient Indic documents are not reliable. We, therefore, motivated the importance of an interactive system for OCR corrections in Section 1.7.1, which we now discuss in detail. The following observations drive research in this section:

1. As the number of auxiliary sources and methods for predicting corrections increase, the probability P of suggesting the correct word increase.

2. Suggestions that are not able to completely correct a word may still be helpful for partial word correction.

3. Each conjoined word (definition in Section 1.4) is a combination of the valid word forms from the language dictionary. The correct split of each conjoined word comprises a lesser number of (dictionary) word forms as compared to its incorrect splits (or splits of the corresponding wrong OCR word).

The problems of reading and typing large-length conjoined words is also a crucial factor while correcting Indic OCR errors. To (i) overcome these challenges, and (ii) generate more OCR-specific datasets in Indic languages, we introduce OpenOCR-Correct, an interactive framework for end-to-end corrections in Indic OCR. The

error detection and correction mechanisms in the framework use the partial word forms obtained from different auxiliary sources. We also provide a user-friendly color-coding scheme in the framework for the partial dictionary string matches to improve the readability of combined words. Figure 4.1 shows the block diagram of our framework. The framework exploits various static and dynamic auxiliary sources for OCR correction. In the user interface, we show a document image on the right and its OCR output on the left. The framework prints the likely correct word forms in black. The words are marked in grey if the user has verified them as correct (previously at a different position in the document). The purple words are ones that have been auto-corrected by the system. The user is required to right-click to generate suggestions. We use multiple colors to improve the readability of erroneous words. Each multi-colored (green and blue) word is a conjoined word consisting of substrings, which are valid words in either the word dictionary or the domain vocabulary (which is updated on-the-fly with corrections). The colors (green and blue) differentiate the adjacent valid sub-strings of the conjoined word.[1] An error is more likely to be present in the places where the green/blue substring is short (of length 2/3 chars). We now discuss various auxiliary sources (see Figure 4.1) used in the framework.

4.2 Auxiliary sources

We found various auxiliary sources to be helpful in the detection and correction of the incorrect OCR words. We now discuss the same in this section.

4.2.1 Static auxiliary sources

The sources such as off-the-shelf dictionary and some of the OCR texts remain static throughout the correction process. We discuss such auxiliary sources in this sub-section.

a) Off-the-shelf dictionary

Though the vocabulary is generally incomplete in Indian languages due to the rich inflections, still the frequent words can be corrected via a fixed word dictionary.

[1]In the system, we also provide the facility to type in SLP1 (an ASCII transliteration scheme) format since typing in ASCII (or English) is much faster (as explained in Section 1.7.1) once the user gets well conversant with typing in the SLP1. If the user presses "Ctrl+D" after typing in SLP1 format, our framework automatically converts the word under the cursor to Devanagari.

b) Sub-strings from OCR words conforming to word conjoining rules

Another important auxiliary source which helps generate the appropriate suggestions is the collection of sub-strings from the OCR words that are corrected, verified as correct, or conform to the rules for conjoining word forms. The sub-word forms for conjoined words are searched in the general word dictionary (defined in Section 4.2.1.c) and the updated domain-specific vocabulary (defined in Section 4.2.2.a).

c) Texts from different OCR systems

In experiments, we observed that previous pages from the same OCR document are one of the most powerful auxiliary sources in correcting the erroneous OCR text since they contain the domain information. Such an auxiliary source is helpful when the OCR document has reasonable word-level accuracy. Hence frequently occurring words can be used to generate the suggested correction for the incorrect OCR words in subsequent pages. Specifically, the words which are incorrect due to location-specific imaging errors can benefit from this source. Another relevant auxiliary source in the dual-engine environment is the text from the secondary OCR system. Since different OCR systems use different models, they are likely to make different kinds of errors. Hence, they are likely to be correct for the OCR words upon which they agree. This observation is especially leveraged by Abdulkader and Casey, 2009. The consensus from multiple OCR systems is helpful in both error detection and correction, as we explain in Section 4.3.

4.2.2 Dynamic auxiliary sources

We now discuss the dynamic auxiliary sources that update on-the-fly with user interactions. We update the sources both during suggestion clicks and the typing corrections (which are updated after the user corrects the complete page).

a) Domain-specific vocabulary

One of the relevant auxiliary sources for OCR error corrections is a domain-specific vocabulary. In this work, we aim to digitize ancient Indian books. Initially, no domain-specific is available for such books. As the user corrects words in the document, we update a domain-specific vocabulary to review the remaining corrections. We subsequently use this vocabulary for the correction of other books written in the same domain.

b) Document and OCR specific n-gram confusions

We observe that the error confusions in words from the primary OCR engine are generally different from the error confusions in the corresponding OCR words from the secondary OCR engine. It happens because two different OCR systems use different pre-processing techniques and different classifier models, as pointed by Abdulkader and Casey, 2009. The n-gram confusions of the primary OCR (refer Figure 4.1) can help in the word corrections as we illustrate through examples next. For example: if "net" and "pet" are closest dictionary words (based on edit distance) to the incorrect word "iiet", the tie can be broken by taking into account that there exists a common OCR character confusion "ii→n". Consequently, the word "net" can be given preference over "pet". Another interesting example would be correcting the output word "iiitermet" to "internet" rather than "interpret", where knowledge of the common OCR confusion "ii→n" is again useful, as is the knowledge that "m→n" is more likely than "m→pr". We also take care of n-gram confusions involving more than two characters at a time. For example, in English "iii→m" and "iii→in" are common OCR confusions. With each correction made by the user, we update the document-specific OCR confusions to correct the remaining words in the same document and words in remaining documents with a similar domain. We use dynamic programming (Bellman, 1966) for word alignment and longest-common-subsequence algorithm (Hirschberg, 1977) for confusion extraction.

c) Plug-in classifier

We now rephrase the primary problem of error detection as that of continuously evolving a classifier that labels each OCR output word as correct or incorrect. During training, the classifier should optimize a performance measure, such as the standard likelihood function or the sum of squared error. An example performance measure that is coherent with our needs of maximizing recall (coverage) in identifying erroneous words while also being precise in this detection is the $F-$score. Such a measure, unfortunately, does not decompose over the training examples and can, therefore, be hard to optimize. We adopt a plug-in approach (Narasimhan *et al.*, 2014) to train a binary classifier over such non-decomposable objectives while also being efficient for incremental re-training.

Consider a simple binary classification problem where the task is to assign every data point $\mathbf{x} \in \mathcal{X}$, a binary label $y \in \pm 1$. Plug-in classifiers achieve this by first learning to predict *Class Probability Estimate* (CPE) scores. A function

$g : \mathcal{X} \rightarrow [0, 1]$ is learned such that $g(\mathbf{x}) \approx \Pr(y = 1)$. Various tools such as logistic regression may be used to learn this CPE model g. The final classifier is of the form $sign(g(\mathbf{x}) - \eta)$ where η is a threshold that is tuned to maximize the performance measure being considered, e.g. F-measure, G-mean etc.

We use *ind.senz (2020)*, *Google (2020) free OCR* and *Tesseract (2020)* OCR for experiments. The confidence measures from the *ind.senz* and *Google free OCR* are not available. We, therefore, use the features such as (i) The edit distance between corresponding OCR words, (ii) the number of dictionary word components (both obtained by applying agglutination rules, and by applying fusion rules) and (iii) all possible products of these three features. Also, we divide each of the first three features with primary OCR word length and use all of their possible products as another set of features. We additionally use two binary features: i) word is common to both OCRs, and ii) word is form dictionary. We use all of the document words for training the classifier with a 48/12/40 training/validation/test split. We train a log-linear classifier with L2-regularized logistic regression[2] and the likelihood objective for three Indian languages with varying inflections.

4.3 Experiments and Results

In this section, we report the results for the different experiments we perform to detect OCR errors and generate correction suggestions. We experiment on three monolithic documents in Sanskrit, Marathi, and Hindi, respectively. We refer the reader to Section 1.7.1 for the details regarding the various performance measures we use.

4.3.1 Error detection

We analyze various methods for detecting errors in the OCR text. In the beginning, we observe that the commonly used dictionary lookup approach gives a high percentage of True Positives (actual errors predicted as errors) and a low percentage of True Negatives (accurate words predicted as correct). We experimented with marking all words that can be formed by applying conjoining rules to words from the off-the-shelf dictionary as correct and observed increases in the True Negatives but reductions in the True Positives. Hence, we do not use this approach. We observe through data analysis that the task of achieving high F-Score depends

[2]https://github.com/cjlin1/liblinear

Lang.	TP	FP	TN	FN	Prec.	Rec.	F-Score
Sanskrit							
LB	87.45	39.02	60.98	12.55	30.36	87.45	45.08
UB	91.62	0	100	8.38	100	91.62	95.63
Dual eng.	82.35	17.64	82.29	17.70	48.04	82.29	60.66
Plug-in classifier	85.13	17.84	82.16	14.87	48.62	85.13	61.89
Marathi							
LB	33.80	25.02	74.98	66.20	36.10	33.80	34.91
UB	15.20	0.03	99.97	84.80	99.49	15.20	26.37
Dual eng.	29.15	12.87	87.13	70.85	48.64	29.15	36.46
Plug-in classifier	76.93	3.83	96.17	23.07	78.77	76.93	77.84
Hindi							
LB	53.15	19.21	80.79	46.85	49.83	53.15	51.43
UB	44.72	1.55	98.45	55.28	91.18	44.72	60.01
Dual eng.	61.76	18.74	81.26	38.23	54.19	61.76	57.73
Plug-in classifier	64.34	15.25	84.75	35.66	55.97	64.33	59.86

Table 4.1: Error detection results in Sanskrit, Marathi and Hindi

upon the complexity of the data and the dictionary used for error detection. The difficulty can be analyzed using the two baselines as follows:-

- Lower Baseline (LB): Dictionary lookup based detection with an off-the-shelf dictionary.
- Upper Baseline (UB): Dictionary lookup based detection with dictionary set to contain all the words in the ground truth. The upper baseline is therefore idealized as out-of-vocabulary ground truth words are never available before corrections.

For the dual-engine environment in the framework, we mark words common to the output of two OCR systems as correct since it is highly unlikely for two OCR systems to output an identical erroneous word (as advocated by Abdulkader and Casey, 2009). Words, for which the OCR systems disagree, are marked as incorrect in such an environment. The results for different Indian languages are summarized in Table 4.1. As shown, the F-Score for the dual OCR system is better than the the Lower Baseline (LB) and below (or in one case even above) the idealised Upper Baseline

(UB). We observe further gains in F-scores after training the plug-in classifier with features described in Section 4.2.2.c. As shown in Table 4.1, we achieve the F-scores of **61.89**, **77.84**, and **59.86** in Sanskrit, Marathi, and Hindi using such ML-based techniques.

Sugg. Rank	%age of "correct & uniquely correct" suggestions in		
	Sanskrit	Marathi	Hindi
1	29.07, 29.07	15.73, 15.73	14.24, 14.24
2	10.76, 4.45	13.23, 5.11	13.05, 0.01
3	23.42, 4.20	14.09, 3.93	3.47, 0.36
4	15.99, 2.86	3.34, 0.60	3.83, 0.72
5	6.84, 2.06	15.20, 11.99	10.97, 8.11
Total (uniq.) suggs.	**42.64**	**37.63**	**23.44**

Table 4.2: Percentage of erroneous words correctly suggested by OpenOCRCorrect

4.3.2 Suggestions generation

At the start of this chapter, we have discussed various auxiliary sources which are useful to generate suggestions in our interactive framework. We use efficient dynamic programming (Bellman, 1966) to produce different correction suggestions. As proposed by Smith, 2011, we avoid using frequency-based language models for OCR corrections as they can do more harm than good in the OCR setting. Moreover, any good OCR system uses a language model in the post-processing stage, and hence its OCR output is likely to exhibit a lower percentage of contextual errors. We, therefore, produce different suggestions by utilizing different auxiliary sources. We present correction results in Table 4.2. Here, each value represents the percentage of errors for which the framework obtains the correct ground truth word as the suggestion. The top two suggestions are the closest suggestions to the secondary and primary OCR documents, respectively. We produce the third suggestion by applying the nearest sub-string search from the secondary OCR document (refer to section 4.2.1.a). For producing the fourth suggestion, we partially correct the word from primary OCR by comparing it with the corresponding secondary OCR word, as explained in Section 4.2.2.b. As explained earlier, we here use the OCR confusions of primary OCR that are updated on-the-fly to break the ties. We produce the fifth suggestion by applying the OCR confusions on the n-grams of (primary) OCR word

to reach a valid word form which follows the conjoining rules. Simultaneously for this suggestion, we use various frequent conjoining rules to split the OCR word until the framework reaches a valid conjoined word form. We look for the word forms of such conjoined terms in the off-the-shelf as well as domain vocabulary that updates on-the-fly. Overall we correct 42.64%, 37.63%, and 23.44% errors in Sanskrit, Marathi, and Hindi using human-in-the-loop, as depicted in Table 4.2.

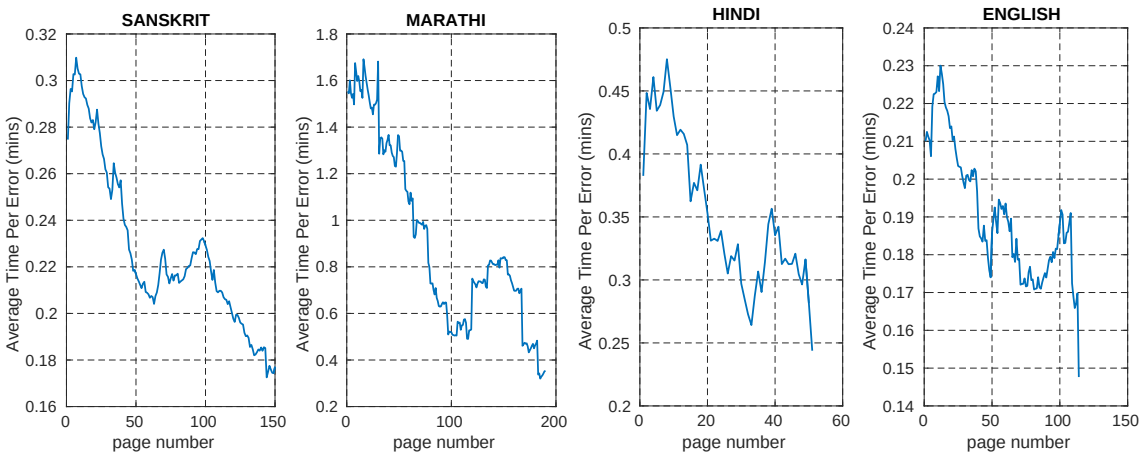


Figure 4.2: System analysis of documents in different languages

4.3.3 System analysis

As mentioned in Section 1.5, we have corrected over 12000 document images in Sanskrit, Marathi, Hindi, and English using OpenOCRCorrect.

In Figure 4.2, we present the real-time analysis for a book in each language. Each book is corrected by a single user so that the efficiency of the framework can be accurately analyzed. After the correction of each page by the user, we save the time taken to correct the page. The number of erroneous words on each page, calculated using dynamic programming (by comparing the original OCR page and corrected page), is also saved and used for calculating the time spent fixing per error (referred to as the time-per-error) on each page. As shown in Figure 4.2, for all the four books, the average time-per-error reduces with page number throughout the significant part of the correction process. Ideally, the time-per-error should be dropping throughout the documents, but fatigue and other factors cause the user to slow down at times. Thus, the system is effective in reducing human effort for corrections in Indic OCR.

Although the system is designed specifically for Indian languages, we nevertheless use it also with English documents. For the particular book in English (see

Figure 4.2, right), the average correction time-per-error drops similarly to the Indic books.

Incorrect OCR Word	Partially Correct Suggestion by System	Corresponding Correct Word
रषविषयकाः	स्घविषयकाः	स्वविषयकाः
बडीलधाऱ्या	बडीलधाऱ्या	वडीलधाऱ्या
य्अकाजोंहाइड्अॅट	काबोहाइड्रेट	कार्बोहाइड्रेट
flJese	These	these

Figure 4.3: Examples of partially correct suggestions

Some errors in OCR output for which the framework produces one of the suggestions as correct can be seen in Figure 1.6 (shown in Section 1.7.1). Some examples of erroneous OCR words, for which the framework suggests partially correct words, are shown in Figure. 4.3.

Correct Word	Word as shown in Framework (Easier to read in colored parts)
प्राणिनिवाससद्भावस्य	प्राणिनिवाससद्भावस्य
असलेल्याबरोबरचे	असलेल्याबरोबरचे
विषमपोषण	विषमपोषण
chloroplasm	chloroplasm

Figure 4.4: Examples of correct out-of-vocabulary words

In Figure 4.4, we show examples of the correct out-of-vocabulary words in the OCR output, which the framework marks as errors (colored). However, the user can easily understand that such words are accurate because of the improved readability. This is possible because we use adequate color-coding for the dictionary strings in a conjoined word. Such coding is also helpful in identifying the segmentation errors such as "Therehegoes" where the OCR system fails to recognize the whitespace characters.

Improved Readability in Errors (Easy to locate errors near red or frequently changing color)	Corresponding Correct Word
ज्योति◌ःशास्त्रीवायकग्रन्थेषु	ज्योतिःशास्त्रविष्यकग्रन्थेषु
नवीकरण्ज्ैय	नवीकरणीय
वायुमंड़ल	वायुमंडल
mountainccring	mountaineering

Figure 4.5: Examples of incorrect OCR words with improved readability

Some examples of incorrect OCR words for which error locations are easily identifiable due to adequate color-coding are shown in Figure 4.5.

Incorrect Word without Correct or Partially Correct Suggestion	Corresponding Correct Word
अर्धाशंखक	अर्धाशं-क
क्योश्मीमक्ख्ी	प्रयोगासाठी
अस्थान–प्रश्अन	आदान-प्रदान
commy	country

Figure 4.6: Examples of complex OCR errors not corrected by our framework

In certain complex cases, the framework is not able to suggest the correct word to the user. Such examples are shown in Figure 4.6. We now improve the Sanskrit plug-in classifier in the next section.

4.4 Improving the Sanskrit Classifiers

In this section, we improve the results of plug-in classifier for Sanskrit OCR texts by introducing language-specific as well as domain-specific auxiliary sources. For some of them, we synthesize valid word forms using the precise context-specific conjoining rules for all possible variants of the nouns available in Sanskrit dictionaries. We also

use one of the existing sources which contain variants of some of the common verbs in Sanskrit.

4.4.1 Language-specific Auxiliary Sources

a) Words from Subanta generator

Since any off-the-shelf dictionary is majorly incomplete for correcting Sanskrit texts due to richness in inflections, we develop a Subanta (or noun-specific declension) generator for synthesizing noun variants. Among the different declension generators, an open-source generator by Patel and Katuri (2015a) works for Sanskrit. We build a new generator for the following reasons:

- For ease of integration into the OCR correction framework;
- For overcoming the errors produced by existing generators. Examples from Patel and Katuri (2015b) include:
  - Some of the variants for nouns ending with ऋ.
  - Some variants for many of the pronouns.
  - Declensions for words ending with वसु affix, which are wrong in some of the cases.
- To have the provision for future enhancements of rules.

We code the rules corresponding to declension generator (and the conjoining rules they require) as per the rules and their explanations by Dīkṣita *et al.*, 2006. We resolve some of the meaningful dependencies of many rules by collecting the context information (refer Adiga *et al.* (2018) for details).

We process the XML file of the Monier-Williams Sanskrit dictionary available in the Cologne Digital Sanskrit Dictionary collections, Institute for Indology and Tamilistics, University of Cologne[3]. We extract more than $1800k$ words with the gender information from the XML file. We produce all possible variants for these words using our generator, which results in a total of 3.2 million unique words. We additionally use the verbs listed in the क्रियारूपनिष्पादिका (verb-forms-generator) of ILTP-DC, which are around $300k$ unique words. These 3.5 million words are used as the off-the-shelf dictionary in the framework for the OCR corrections.

[3] `http://www.sanskrit-lexicon.uni-koeln.de/download.html`

b) Initialized Domain-specific vocabulary

In Sanskrit, the set of commonly used words changes from one document to another. So the domain-specific dictionary is one of the most substantial auxiliary resources as it involves the words that are absent in any off-the-shelf dictionary. For the experiments in Section 4.4.2, we initialize the domain-specific vocabulary for each document by extracting unique strings from the various books available in the Göttingen register of electronic texts in Indian languages (GRETIL, 2001). This auxiliary source is also dynamically updated as discussed earlier in Section 4.2.2.a.

c) Word conjoining rules

Due to conjoining rules, words can change dynamically in Sanskrit documents. We apply fundamental conjoining rules to identify the necessary word forms of a conjoined word. We search for the basic word forms in the off-the-shelf dictionary. We use a greedy approach for this splitting the cojoined word which includes a minimum set of words (of maximum length) and smallest edit distance as the criteria. For example, the framework splits the OCR word **जागरितावस्थायाभेवावस्थात्रयमुक्तं** into **जागरित**, **अवस्थायाभ्**, **एव**, **अवस्थात्रयम्** and **उक्तं**. The word **अवस्थायाभ्** in this example is closer to a valid word **अवस्थायाम्**. This kind of split helps in detecting the erroneous out-of-vocabulary words and generating the correction suggestions for them. Moreover, we use the OCR-specific n-gram confusions to break the ties. For example, while changing the erroneous word **निवन्धः**, if the dictionary lookup suggests **निबन्धः** and **निरन्धः** as nearest possible words, having higher n-gram confusion to **व->ब** biases the selection towards **निबन्धः**.

4.4.2 Error Detection Methods and Results

#	Approach	TP	FP	TN	FN	Prec.	Rec.	F-Score
1.	General Dict. Lookup	89.18	40.12	59.87	10.82	29.75	89.18	44.61
2.	Conjoining Rules	54.34	13.23	86.77	45.66	43.89	54.34	48.56
3.	Secondary OCR Lookup	90.68	23.59	76.40	9.31	42.79	90.68	58.14

Table 4.3: Error detection results with lookup based methods

We apply various methods for detecting errors in the OCR text. To start with, we use the book named "Āryabhaṭīyabhāṣya of Nīlakaṇṭha III Golapāda (AnantaśayanaSaṃskṛtaGranthāvaliḥ, 1957)" for which we have the OCR texts

from *ind.senz (2020)*, *Google (2020)* and the ground truth data available. We use *ind.senz* as the primary OCR engine and *Google free OCR* as the secondary for all the experiments (we have discussed the word error rates for these OCR systems in Section 1.6).

Lookup approaches

A commonly used dictionary lookup based approach gives poor F-Scores, as it marks a large number of correct words as errors. Therefore it leads to lower True Negatives (TN) as shown in Table 4.3. Another interesting methodology is marking the words formed by applying the conjoining rules on dictionary lexicons as correct. As shown in Table 4.3 (row 2), implementing this approach improves the detection of accurate words (True Negatives for error detection). However, it also lowers the detection of errors (True Positives) in comparison to the basic dictionary lookup approach. For this book, comparing the OCR output of another engine (*Google free OCR*) for the same document images improves the F-Score (see Table 4.3, row 3).

Single-engine Environment

#	Approach	TP	FP	TN	FN	Prec.	Rec.	F-Score
1.	Classifier with ngrams frequency + word lookup in General Dict. as features	73.38	22.86	77.13	26.61	38.88	73.38	50.83
2.	Classifier with ngrams frequency + word lookup in Synthesised Dict.(superset of gen. dict.) as features	74.06	21.02	78.98	25.94	41.14	74.06	52.89
3.	Classifier with ngrams frequency + word lookup in Synthesised Dict. as well as Domain Dict. as features	66.37	13.08	86.92	33.63	50.38	66.37	57.28
4.	Classifier with features in row 3 + number of conjoined components in OCR word as features	68.50	13.53	86.47	31.50	50.10	68.50	57.87

Table 4.4: Error detection results in single-engine environment

For single-engine environment, we learn the plug-in classifier on features explained in Section 4.2.2. We split the data with train:validation:test ratio as

48 : 12 : 40. We are able to improve the row 1 results in Table 4.3 by including the frequency of OCR word's n-grams (up to 8 characters) in a general dictionary as features. We also include a binary feature based on lookup in the general dictionary. The results are shown in the first row of Table 4.4.

We further include more words in the general dictionary by synthesizing nouns and collecting the verbs as explained in Section 4.4.1.a. This approach helps us to achieve the improved results shown in row 2 of Table 4.4.

Adding frequencies of OCR word's n-grams in the initialized domain dictionary (Jyotiṣa for this book from the source given in Section 4.4.1.b) to the features obtained from the synthesized dictionary further improve the results as shown in row 3 of Table 4.4.

For improving the results further, we use three splitting based features: i) Split the OCR words using commonly used conjoining rules and use the number of lexicon components obtained from the general dictionary as features. ii) We also use number of lexicon components obtained by splitting the OCR word as lexicons of domain dictionary (for Jyotiṣa) as a feature. Herein, we also add the number of characters from unknown sub-strings in the OCR word to the feature space. iii) We also use the product of features obtained in (i) and (ii) as the features. We normalize all these features using the mean and standard deviation of complete training data. The results are shown in row 4 of Table 4.4. It is interesting to note that here in the single-engine environment, we reach performance that is closer to the results of the dual-engine based secondary OCR lookup approach given in row 3 of Table 4.3.

Multi-engine Environment

We further include the dual-engine OCR agreement as a feature in addition to the features we use in the previous section. The results are presented in Table 4.5. We improve the results by using the feature of dual OCR agreement between *ind.senz (2020)* and *Tesseract (2020)* in addition to previous features to obtain the results shown in row 4 of Table 4.5.

We present the results of the plug-in classifier trained and tested on the dataset of books with different domains in Table 4.6. These results confirm the generality of our approach for different domains of Sanskrit literature. Row 1 in this table shows the baseline for the book 'Nṛsiṃhapūrvottaratāpanīyopaniṣat' (ĀnandāśramaSaṃskṛtaGranthāvaliḥ, 1929) and row 2 shows the results achieved using all the features (obtained using triple-engine environment, off-the-shelf dictionary, domain vocabulary and n-gram frequency from general, synthesized and domain vocabular-

#	Approach	TP	FP	TN	FN	Prec.	Rec.	F-Score
1.	Classifier with features in table 4 row 2 along with dual-engine agreement (Table 4.1 results)	85.13	17.84	82.16	14.87	48.62	85.13	61.89
2.	Classifier with features in table 4 row 3 along with dual-engine agreement	78.04	13.67	86.33	21.96	53.11	78.04	63.20
3.	Classifier with features in table 4 row 4 along with dual-engine agreement	83.49	15.26	84.74	16.51	52.25	83.49	64.28
4.	Classifier with features in table 4 row 4 along with triple-engine agreements	83.43	14.95	85.04	16.56	52.74	83.43	64.63

Table 4.5: Error detection results in multi-engine environment

#	Approach	TP	FP	TN	FN	Prec.	Rec.	F-Score
1.	Vedānta gen. dict. lookup baseline	85.52	34.35	65.65	14.48	49.71	85.52	62.87
2.	Vedānta plug-in classifier	79.95	9.80	90.20	20.05	77.95	79.95	78.94
3.	Sāhitya gen. dict. lookup baseline	64.24	35.36	64.64	35.76	32.86	64.24	43.49
4.	Sāhitya plug-in classifier	87.88	13.37	86.62	12.12	66.52	87.88	75.72

Table 4.6: Error detection results for other domains

ies). It is important to note that the TP (actual errors detected as errors) percentage is high for the baseline in this case as compared to the TP baselines in other domains. However, TN (correct words detected as correct) for the dictionary lookup baselines are close to each other for all domains as shown in row 1 of Table 4.3 and row 1 and row 3 of Table 4.6. The reason for high TN could be less ambiguity (as compared to other domains) in incorrect words since TP (unlike TN) does not depend on the presence of correct OCR words in the dictionary. Hence we are getting F-scores as high as 62.87 for the baseline in this case. We also evaluate the system for Sāhitya domain. For this we use the book 'Raghuvaṃśam Sanjīvinīsametam' (Nirṇaya Sāgara Press, 1929, 1-9 Sarga) and row 3 in table 4.6 shows the baseline, whereas row 4 shows the results obtained using the framework.

Sources included	Percentage of errors for which the obtained suggestion is correct
Domain words with dual OCR agreement + Synthesized Confusions	36.26
Prev. + adapting Domain Words/Page	36.38
Prev. + adapting Confusions/Page	37.14
Prev. - Synthesized + Real Confusions	39.40

Table 4.7: Suggestions improvements for a Sanskrit book[4] due to adaptation

Suggestions Generation

The results for various methods of exploiting auxiliary sources, to generate appropriate suggestions, were given in Section 4.3.2) for "Āryabhaṭīyabhāṣya of Nīlakaṇṭha III Golapāda (AnantaśayanaSaṃskṛtaGranthāvaliḥ, 1957)".

Here, in Table 4.7, we show the improvements due to regular adaptation of domain dictionary and OCR Confusions in the framework for "Āryabhaṭīyabhāṣya of Nīlakaṇṭha III Kālakriyāpāda, 1931". For this experiment, we synthetically generate word images for the words in Sanskrit dictionaries. We then OCR the obtained images using *ind.senz (2020)* and extract around 0.5 million erroneous-correct word pairs. We use the longest-common-subsequence algorithm proposed by Hirschberg (1977) to produce about 0.78 million OCR character confusions. The row 1 of Table 4.7 shows the total percentage of incorrect OCR words for which we get the correct ground truth word as the suggestion obtained using various auxiliary sources. Such sources use i) words common to dual OCR systems as domain vocabulary throughout the document and ii) obtained synthesized confusions. As shown in row 2, we further improve the quality of suggestions by uploading the corrected domain words on-the-fly after the user corrects each page. As shown in row3, adapting the confusions on-the-fly page by page further improves the results. Using real confusions obtained from the primary OCR text and ground truth, of other books, helps in improving the results further as presented in row 4 of Table 4.7. In total, we achieve 3% of gains in suggestions for the document due to adaptive auxiliary sources we use in the framework.

[4]"Āryabhaṭīyabhāṣya of Nīlakaṇṭha III Kālakriyāpāda, AnantaśayanaSaṃskṛtaGranthāvaliḥ, 1931"

4.5 Conclusion

In this chapter, we have presented OpenOCRCorrect, an interactive system for word-level corrections in Indic OCR. The system works for multiple Indian languages with varying degrees of inflections. It is easy to adapt the system to any other language just by changing the ASCII transliteration scheme, which it uses to store and process the data. The framework leverages generic word dictionaries and a domain-specific vocabulary grown incrementally based on user corrections from the current on the OCR document. It also learns OCR-specific confusions on-the-fly. We have incorporated word conjoining rules to parse OCR words and discover their potentially correct sub-strings. Furthermore, we have presented a dual-engine environment to cross-verify potential errors and corrections. We empirically verify that the dual-engine environment, in conjunction with the previously mentioned resources, yields error detection performance close to the idealized baseline. The dual-engine environment is additionally helpful in providing accurate suggestions. We also presented a plug-in classification approach to further improve error detection by tuning the probability threshold for classification. Given the role of user interaction in the framework, we have carefully designed the UI to reduce the overall cognitive load by use of transliteration schemes, suitable color-coding, and learning on-the-fly from interactions. We also demonstrate different ML approaches for Sanskrit OCR corrections. We have presented a multi-engine environment which is useful in detecting potential errors. We have also shown the effectiveness of our framework. The average error correction time reduces as the user progresses through the pages of a book. We also demonstrated various examples with a user-friendly color-coding to reduce the cognitive load of the user. Using the features from additional Sanskrit-specific auxiliary sources, for the plug-in classifier, we succeed in improving overall F-Scores. The work presented in this section was published at ICDAR'17 and WSC'18 in our papers (Saluja *et al.*, 2017b; Adiga *et al.*, 2018).

Chapter 5

Deep Learning Techniques for OCR Corrections

In the previous chapter, we have understood how classical machine learning techniques can assist with detecting errors in Indic OCR texts and how limited these techniques are when it comes to fully-automated correction. We turn attention towards deep learning techniques in this chapter. In this chapter, we answer the research questions related to i) sequential modeling techniques, and ii) Indic OCR texts, which we raised in Chapter 3. In Section 5.1, we briefly introduce the Recurrent Neural Network (RNN) and one of its variants called Long-short Term Memory (LSTM) model. We then customize the LSTM for Indic OCR corrections. We further discuss the advantage of using sub-word embeddings to improve such models in Section 5.2. We conclude each of these sections with various experiments and results on the datasets that we have explained in Section 1.4. In Section 5.3, we introduce the attention-based encoder-decoder model using which we achieved the 2^{nd} place in two ICDAR post-OCR competitions (Chiron *et al.*, 2017a; Rigaud *et al.*, 2019). We finally conclude the chapter in Section 5.4.

5.1 Indic OCR Corrections using LSTMs

As was noted in the previous chapter, learning n-gram confusions (or error patterns) of the OCR system, as well as the partial word forms present in the language, can help correct the out-of-vocabulary words in OCR documents. In this section, we adopt an LSTM based character-level language model for jointly addressing the problems of error detection and corrections in Indic OCR. For words that need no correction in the OCR output, the model abstains from suggesting any change.

OCR Word	LSTM output/ Correct OOV Word
एवमसक्तात्करणेऽवनिघ्नीः	एवमसकृत्करणेऽवनिसूनोः
കല്ലാന്റെടുത്തുകൊണ്ടിരിക്കുന്ന	ക്ലാസ്സെടുത്തുകൊണ്ടിരിക്കുന്ന
ಗಜಸೆನಿವನು	ಗಜಸೈನ್ಯವನ್ನು
जसदपाऊ	जसदयाल

Figure 5.1: Examples of OCR words corrected by LSTM in four Indic languages

In Figure 5.1, we present the examples of errors detected and corrected by our model. Here, the correct words are all out-of-vocabulary words and mistakes are marked in red. In the process of manually correcting OCR documents, we frequently observe that the knowledge of error patterns is helpful in error detection and correction. The OCR system tends to get confused between letters with similar images. To correct such errors, we don't however, need to refer back to the original word image, since the error patterns and the context in the OCR output itself can help infer the confusion information. Our results support this observation.

We begin with a brief understanding of RNNs and LSTMs in Section 5.1.1. We then present the problem scope and error analysis in Section 5.1.2, wherein we also describe our datasets. Subsequently, in Section 5.1.3, we present the method of using an LSTM with a *fixed delay* for OCR correction in Indian languages. In Section 5.1.4, we outline the experiments, including various methods by which we exploit the LSTM with a *fixed delay* in different contexts. In Section 5.1.5, we present extensive results to validate our model's performance on four Indian languages with varying inflectional complexities. We achieve F-Scores above 92.4% and reductions in Word Error Rates (WER) of at least 26.7% across four Indian languages.

5.1.1 RNNs and LSTMs

In work by Sutskever *et al.* (2011), Recurrent Neural Networks (RNNs) are shown to be useful in learning character-level language models. Such models make no local independence assumptions on the language text, unlike Hidden Markov Models (HMM, proposed by Rabiner and Juang, 1986). RNN-based models are also used effectively in Machine Translation (Sutskever *et al.*, 2014). Character-based attention RNNs have also shown improvements in Neural Language Correction (Xie *et al.*, 2016). In particular, a specific type of RNNs called Long Short Term Memory Networks (LSTMs), learn the more extended contexts and are therefore best suited for languages with a high fraction of out-of-vocabulary words. We delay the output

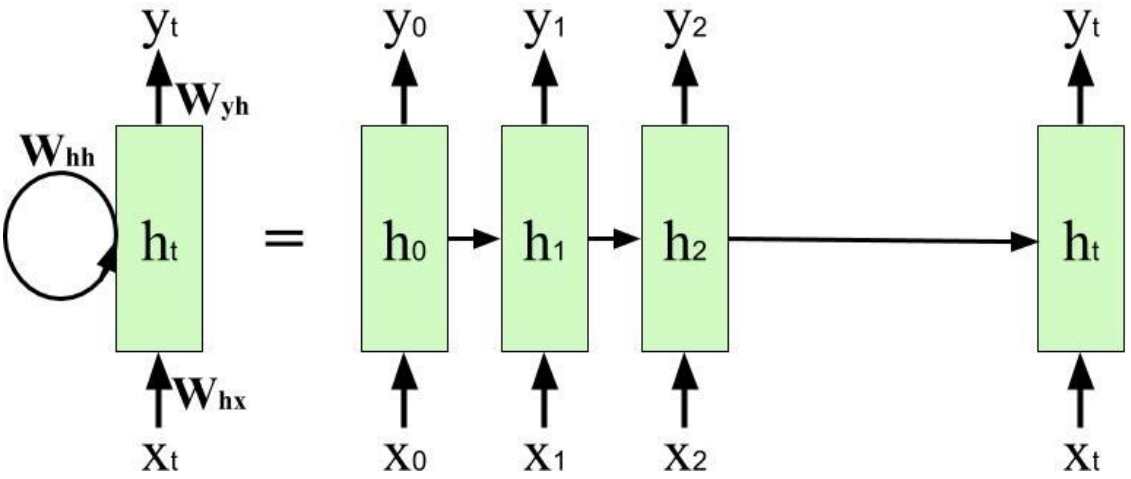


Figure 5.2: A Recurrent Neural Network unrolled for t-time units

in the LSTM model to take care of n-gram character confusions and succeeding contexts.

A basic RNN (Recurrent Neural Network) can be represented by Equations 5.1 and 5.2.

$$h_t = g(W_{hh}h_{t-1} + W_{hx}x_t + b_h) \tag{5.1}$$

$$y_t = W_{yh}h_t \tag{5.2}$$

Where g is an activation function, such as sigmoid ($\sigma(x_i) = \frac{exp(x_i)}{\sum_j[exp(x_j)]}$), tanh ($tanh(x) = 2\sigma(2x) - 1$) or Rectified Linear Unit (ReLU) ($f(x) = \max(0, x)$) (Talathi and Vartak, 2015). The matrices W_{hx} and W_{yh} connect the input x_t to the hidden layer h_t and the hidden layer h_t to the output y_t respectively. These matrices contain the parameters, which are shared across each instance in the input sequence $x_{1:T}$. The matrix W_{hh} is the feedback from preceding hidden layers (or inputs) and is responsible for remembering and forgetting the sequence history based on the context.

Equation 5.2 at each time t can be unfolded back in time, to time $t = 1$ for the 1^{st} character of the word sequence, using Equation 5.1 and the network can be trained using Back Propagation Through Time (BPTT) Schuster and Paliwal, 1997.

We ensure equal byte length per letter by using the ASCII transliteration scheme that we explain in Section 5.1.2. For the loss, we use negative log-likelihood of Log SoftMax (multi-class) function. The Log SoftMax function is shown below, where y_{t_i} is the value at i^{th} index of output vector y_t.

$$f(y_{t_i}) = log(\frac{exp(y_{t_i})}{\sum_j[exp(y_{t_j})]}) \tag{5.3}$$

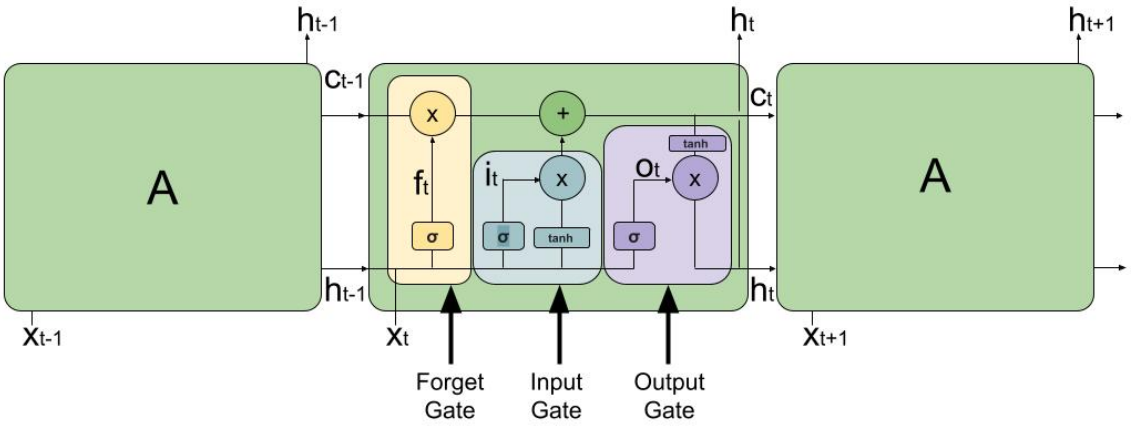


Figure 5.3: LSTM gates

The equations for the LSTM are similar to that of the RNN. Except that instead of a neuron, each unit of the LSTM is a memory unit. Such a memory unit remembers, forgets, and transfers cell state to the output (or next state) based on input history. The cell state at time t is given by Equation 5.4 where forget gate f_t and the input gate i_t fire according to Equations 5.5 and 5.6 respectively.

$$c_t = f_t \otimes c_{t-1} + i_t \otimes g_1(W_{hc}h_{t-1} + W_{xc}x_t + b_c) \tag{5.4}$$

$$f_t = g_2(W_{xf}x_t + W_{hf}h_{t-1} + b_f) \tag{5.5}$$

$$i_t = g_2(W_{xi}x_t + W_{hi}h_{t-1} + b_i) \tag{5.6}$$

Here $\otimes$ and $+$ operations are elementwise product and sum, respectively. The data is selectively transferred from the cell to the hidden state h_t according to Equation 5.7.

$$h_t = o_t \otimes g_1(c_t) \tag{5.7}$$

Here, the selection is performed by the firing of output gate o_t as per Equation 5.8 forms the basis of selection.

$$o_t = g_2(W_{xo}x_t + W_{ho}h_{t-1} + b_o) \tag{5.8}$$

Here g_1 is usually implemented using a tanh function and g_2 is generally sigmoid.

5.1.2 Problem scope, Data description and Analysis

As discussed in Section 1.4, the vocabulary is the most dynamic/incomplete in Sanskrit, followed by Malayalam, Kannada, and Hindi. A glimpse of OCR errors can

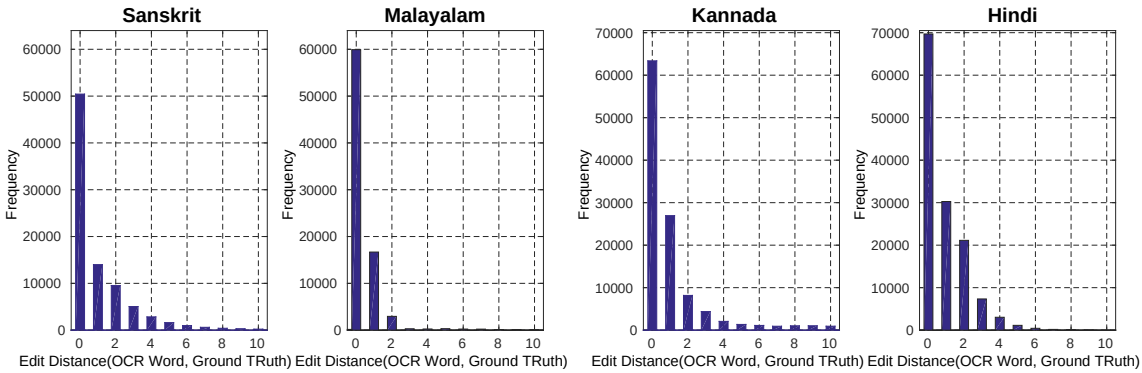


Figure 5.4: Histogram of edit distance between OCR and ground truth in word pairs

be obtained from the histogram of Levenshtein-Damerau edit distance (by Damerau, 1964 and Levenshtein, 1966) between pairs of OCR words and corresponding ground truth words. We prefer to work using edit distance instead of hamming distance since a significant fraction of OCR errors consists of confusions between letters that look similar.

We use the *Google (2020) free OCR* to scan $86k$ Sanskrit, $81k$ Malayalam and $118k$ Kannada words from different documents. We carefully corrected the OCR words to form the ground truth. For Hindi, we obtain $67k$ word pairs from Vinitha and Jawahar, 2016. Since more than 96% of the words were incorrect in the original dataset for Hindi, we balanced the dataset by also including (Ground Truth word, Ground Truth word) pairs in addition to the (OCR word, Ground Truth word) pairs. This balancing increased the number of word pairs in Hindi to $134k$. The dataset is summarized in Table 5.7. We aligned the word pairs using the recursive Text Alignment Tool (RETA) by Yalniz and Manmatha, 2011. To tackle the problem of variable byte length per character in Indic scripts, we used ASCII transliteration schemes such as SLP1 (Sanskrit Library Phonetic Basic encoding scheme) for all the experiments.

In Figure 5.4, we present the histograms for Sanskrit, Malayalam, Kannada, and Hindi word pairs. The frequency at 0 edit distance represents the number of words correctly detected by the OCR system. It is important to note that the number of erroneous OCR words, *I .e.*, words with edit distance ≥ 1, is maximum for the words that are a unit distance away from the ground truth. This number decreases exponentially with distance irrespective of the language and the OCR system. A good OCR model would tend to make fewer mistakes at higher edit distances, and thus such a histogram would decay faster as compared to a poor quality OCR model. Interestingly, such a histogram provides intuition regarding the

Devanagari Letter	Corresponding SLP1 char.	One Hot Vector
ऽ	$	10000000
◌ा	A	01000000
◌ि	i	00100000
त्	t	00001000
प्	p	00000010
ष्	z	00000001

Table 5.1: One Hot Vector and corresponding SLP1 character for letters in Fig. 5.5

appropriate amount of delay in the LSTM model (7 for Sanskrit & Kannada and 5 for Malayalam & Hindi). Words corrected by the OCR correction system should have fewer errors than the original OCR words, and hence mass in the histogram should shift left, toward lower edit distances. This is indeed the case for our model, as can be seen in Figure 5.6 for the test data.

5.1.3 LSTM with a fixed delay

An LSTM can be used to predict characters that appear in a word based on a preceding sequence of characters. Character-based approaches have not (yet) attained state-of-the-art performance on language modelling tasks (Xie *et al.*, 2016). Such an approach, however, can be useful for correcting OCR errors, since the OCR output is partially correct and most erroneous n-grams follow some known confusion patterns based on images of characters (or sequence of characters) that look similar. An erroneous n-gram in a word can be more robustly detected and corrected if we look at the sequence of characters that appear before and after the n-gram. This is modeled by an LSTM with a *fixed delay*. In this model, the delay allows the succeeding sequence of characters to also be used for learning (unlike a simple LSTM where only the preceding sequence is considered). The length of the succeeding sequence used is equal to the delay. Another reason for including the delay is to allow for character contractions, whereby multiple characters replace a single character.

We use one-hot-encoded characters (see the input layer of Figure 5.5) from a word in the OCR document as input. One-hot-encoded characters from the corresponding aligned information in the ground truth documents form the model's output for training and testing. Results show that an LSTM model, with a *fixed de-*

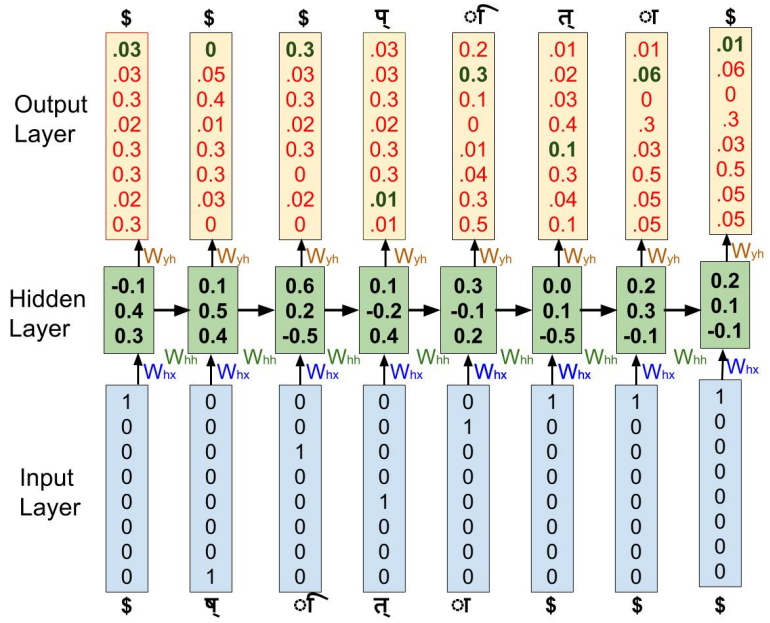


Figure 5.5: An LSTM model with 2 units of delay (appears as character $), having 1 hidden layer of 3 units, unfolded for 8/time units

lay at the output, trained in this manner is capable of word-level error detection and correction. Our results show that an LSTM model with a *fixed delay* at the output trained in this manner is capable of word-level error detection and correction.

In Figure 5.5, we illustrate an LSTM with 2 units of delay, with one hidden layer of 3 units, unfolded for 8 units of time. We represent the delay by making use of null-character ($) symbols. We also pre-condition the OCR word at the input with a buffer of 1 unit and the corresponding ground truth word at the output by 3 units of time to account for 2 units of net delay. We illustrate an example encoding for the Devanagari letters of Figure 5.5 in Table 5.1. We assume that the character vocabulary size and the maximum word length are both 8. Training the model should produce output vectors close to the one-hot vector for each letter. In the illustration, the element of the output vector that we show in green should maximize during training.

In practice, we pre-condition the input with a buffer of 15 null-characters. This is necessary to allow the recurrent network to learn a valid starting state. We delay the output with a buffer of 15 + d null-character symbols. Here d represents the sequence delay which we tuned empirically, guided by the insight in Section 5.1.2 based on Figure 5.4. We also right pad each word with a sequence of null characters to form the fixed-length (= maximum word length in the language + preceding delays) inputs and labels.

5.1.4 Experiments

Our model contains 2 hidden layers of 512 units each. We train the model for 150 epochs. The percentage of erroneous words in the validation set corrected by the model increases and then hits a maximum at a particular training epoch and starts to decrease. We use the model which corrects the maximum number of erroneous words and employ the same model on test data. Interestingly, the model corresponding to this epoch also gives the maximum F-Score over the validation dataset across all epochs. For training, we use the gradient descent algorithm with the learning rate of 0.002 and the decay of 0.97 per epoch after 10 epochs of training. We train models for 100 epochs (since we achieve maximum accuracy for all our models before 100^{th} epoch).

For LSTM models, while increasing the delay between the input and output sequence results in enhanced context, we found that increasing the delay beyond a certain point also increases the error in the output of the network. Intuitively, this could be because a larger delay makes it difficult for the model to predict the corrections and/or the model overfits on higher level n-grams. As stated earlier, we found the sequence delay of 5 to 7 (character) units between the input word and output word to work reliably in practice for the Indian Languages we work with.

We used the dataset introduced in Section 5.1.2. As noted earlier, we balanced the dataset for Hindi word pairs. This also ensures that we make a fair comparison with a previous error detection results by Vinitha and Jawahar, 2016. We use a train-val-test split ratio of 64-16-20 in experiments.

Error detection experiment

For an input OCR word, if the trained LSTM model outputs a word different from it, we mark it as incorrect else correct. This is how we detect errors using our model trained for error correction. The error detection results for the LSTM model are better than the results of the previous state-of-the-art system (see Section 5.1.5). Various performance measures for error detection are discussed in Section 1.7.1.

Error corrections experiment

We evaluate the performance of our model for error correction against two baselines that we create by combining the ideas of standard dictionary-based error correction with the tie-breaking through the n-gram character confusions of the OCR system. For each OCR word o, we find the set of nearest words W from the dictionary V

(*I .e.*, the vocabulary of familiar words). We compute the posterior distribution (described by Equation 5.9) on w to rank the replacement words in W to determine the desired word w^*.

$$w^* = \arg\max_{w \in W} P(w|o) = \arg\max_{w \in W} P(o|w)P(w) \tag{5.9}$$

For $P(w)$ we use word frequencies from training and validation datasets, while $P(o|w)$ is estimated based on character confusion probabilities as $\prod_{(c_o,c_w)\in C_{ow}} P(c_o|c_w)$. Here C_{ow} is the set of n-gram character confusions[1], (c_o, c_w), required to convert o into w. For $P(c_o|c_w)$ we consider the frequency of confusions in the union of training and validation datasets. We use Laplace smoothening for the unseen confusions.

In the first baseline, we consider V to be the set of ground truth words from training and validation datasets. The test dataset is different from V (exactly as in the LSTM model). We call this the lower baseline. In the second baseline, we assume that all the ground truth words for the test dataset are also available in V. Hence, we call it the upper baseline: an idealized, best possible baseline for word-level corrections (based on dictionary lookup).

Suggestions Generation Experiment

We observe that four different contexts help correct the characters of an OCR word;

1. PC: Preceding characters from the OCR word itself,
2. SC: Succeeding characters from the OCR word[2],
3. PCPW: Preceding characters from the OCR word and its preceding word neighbors, and
4. PCSW: Preceding characters from the OCR word and its succeeding word neighbors.

For Sanskrit, which exhibits the highest proportion of out-of-vocabulary words (see Section 5.1.2), we train the LSTM network with these 4 contexts and obtain a model from each. For PC, we train the forward character model explained in Section 5.1.3. In contrast to this, for SC, we train another word-level model with the order of characters reversed in both the input and the output words (to consider more succeeding

[1]Computed using dynamic programming (Bellman, 1966) and longest-common-subsequence algorithm (Hirschberg, 1977).

[2]Certain mistakes in Indian language scripts are more sensitive to succeeding characters than preceding ones.

contexts because LSTM considers preceding contexts by default). For PCPW, we train an LSTM model that takes characters from 5 preceding OCR words as well as 1 present OCR word in the input. In all, we consider 6 corresponding correct words in the output. On test data, 6 OCR words are provided as input, while we are concerned only with the corrections of the last. Similarly, for PCSW, we train a model that considers the characters from 6 words in the input, 5 of which succeed the present word. To achieve this, we reverse the order of words used to train the LSTM. For the last 2 models, we used the delay of 20 characters.

5.1.5 Results

Here, we present results for the settings described in Section 5.1.4.

Language	TP	TN	FP	FN	Prec.	Recall	F-Score
Sanskrit	92.63	94.54	5.45	7.36	94.84	92.64	93.72
Malayalam*	87.56	94.23	5.77	12.44	93.82	87.56	90.58
Malayalam	92.62	96.02	3.98	7.38	93.26	92.63	92.94
Kannada	98.51	97.28	2.71	1.48	96.92	98.41	97.66
Hindi*	72.30	90.90	9.10	27.70	89.30	77.22	82.82
Hindi	91.96	93.86	6.14	8.04	92.94	91.95	92.44

Table 5.2: Error detection results in Indic OCR. *Vinitha and Jawahar, 2016

Error Detection Results

We present the basic error detection results in Table 5.2. Here, we note that the word-level error detection on OCR output obtained using the basic forward character level LSTM model outperforms the state-of-the-art results by Vinitha and Jawahar, 2016 (shown as Lang.*) in Malayalam and Hindi. Further, it is important to note that the results for Sanskrit and Kannada are better than the results for Malayalam and Hindi respectively, although the former languages have the higher percentage of out-of-vocabulary words (see Figure 1.2).

Error Correction Results

In Table 5.3 we compare our method against the two baseline models described in Section 5.1.4. As shown, we achieve a reduction in overall WER by at least 26.7%, and our model corrects at least 63.3% of word errors for all the languages. The

Language	Word Error Rate (WER)				%age words corrected by		
	OCR	Baseline		LSTM	Baseline		LSTM
		Lower	Upper		Lower	Upper	
Sanskrit	51.20	58.60	20.01	21.41	9.62	66.12	63.34
Malayalam	37.28	48.43	10.83	10.59	9.09	58.20	78.30
Kannada	47.44	48.13	27.77	15.73	18.31	54.57	69.66
Hindi	46.80	45.43	34.17	16.71	20.94	27.46	72.47

Table 5.3: Decrease in WER and percentage of erroneous words corrected by LSTM

Language	TP	TN	FP	FN	Prec.	Recall	F-Score
Gujarati	95.55	85.94	14.06	4.45	87.17	95.54	91.16
Telugu	96.60	85.64	14.36	3.40	87.05	96.59	91.57

Table 5.4: Error detection for smaller datasets in Gujarati and Telugu

LSTM-based model outperforms both the baseline models. Even though the upper baseline model contains the ground truth word for the test data in its dictionary, it is unable to correct all errors. This happens because a word search in the Indic vocabulary invariably results in a large set of neighbours. From such a large set, it is ambiguous to pick the correct word even using the knowledge of OCR-specific n-gram confusions. Only in the case of Sanskrit, the upper baseline model is marginally better because the language has many long words. Therefore, usually, a word search in the idealized Sanskrit dictionary results in only one of the words as the neighbour. It is important to note that the LSTM has no access to the ground truth of the test data, as is the case for the upper-bound baseline model. The poor performance of the lower baseline attributes to the fact that the vocabulary of ground truth words in the test data is significantly different from the training and validation sets.

Our LSTM-based model with a *fixed delay* also works reliably on smaller datasets of $20k$ and $28k$ word pairs in Gujarati and Telugu respectively, which we obtained from Vinitha and Jawahar, 2016. We were able to correct 75.67% words in Gujarati and 73.36% words in Telugu (as shown in Table 5.4), with F-Scores of 91.16 & 91.57 respectively. In both these languages, our models perform better, than the upper baselines which correct 58.65% & 49.22% words in Gujarati and Telugu respectively.

Language	Word Error Rate (WER)				%age words corrected by		
	OCR	Baseline		LSTM	Baseline		LSTM
		Lower	Upper		Lower	Upper	
Gujarati	50.07	42.96	20.64	18.25	53.53	58.65	75.67
Telugu	50.00	44.59	25.38	20.49	46.15	49.22	73.36

Table 5.5: Error correction for smaller datasets in Gujarati and Telugu

We test for statistical significance of the performance of our model over the upper baseline using a Wilcoxon Signed-Rank test. The null hypothesis is that the performance of our model is not better than the upper baseline. On the percentage of word errors corrected by both the methods, we obtain a significance of 3.8% for the 6 Indian languages mentioned above. This clearly rejects the null hypothesis and supports the claim.

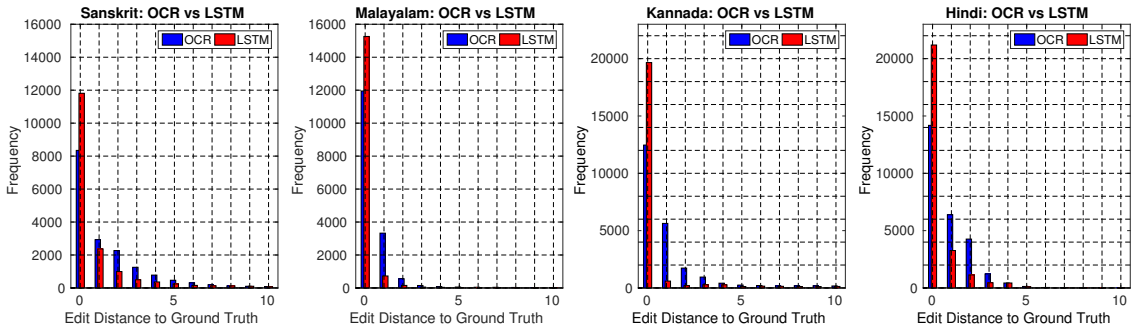


Figure 5.6: Histogram of edit distance between OCR & ground truth word pairs (in blue), LSTM output & ground truth word pairs (in red)

To analyze the partial corrections in words or character level improvements, in Figure 5.6, we show the histograms of edit distance of OCR words from ground truth words and the LSTM output from the ground truth. Note that we present these histograms only on the test data. Our model reduces character-level errors for most words. This is apparent from the shift in the histogram for OCR words with higher edit distances from the ground truth to lower values of edit distances.

Examples of words completely corrected by the system can be seen in Figure 5.1. In Figure 5.7, we show examples of words that are partially corrected by our model.

If the OCR system incorrectly map words from the document image to other correct words in the language, our model is unable to detect such Real Word Errors

OCR Word	LSTM output	Correct Word
गौघ्रपरिधिभागा	शीघ्रपरिधिभागा	शीघ्रपरिधिभागाः
തേങ്ങാക്ക്യഷി	തേങ്ങാക്കൃഷി	മാങ്ങാക്കൃഷി
ದಂದದಲಿ	ದ್ವಂದ್ವದಲ್ಲಿ	ದ್ವಂದ್ವದಲ್ಲಿ
स्म?ल्याआं	समस्याओं	समस्याओं

Figure 5.7: Examples of OCR words partially corrected by LSTM

(RWE) and does not correct these words. Examples are shown in Figure 5.8 (top). In a few rare cases, the LSTM does introduce new errors in words that are correct in

OCR Word/LSTM output	Correct Word
शीघ्रकेन्द्रभुक्ति	शीघ्रकेन्द्रभुक्तिः
വേണമെന്നു	വേണമെന്നു്
ಗಾಯಾಳುವನು	ಗಾಯಾಳುವನ್ನು
उसने	उसके

OCR Word/Correct Word	LSTM output
समशङ्क	समशङ्कः
ശല്യം	സത്യം
ಮೊರೆಯಿಡಲಾರಂಭಿಸಿದವು	ಮೊರೆಯಿಡಲಾರಂಭಿಸಿದ್ದ
मुनीम	सुनीम

Figure 5.8: Examples of words not corrected, corrupted (top, bottom) by LSTM

the OCR output. This seems to happen in the words where the model replaces less frequent n-grams by more frequent n-grams. More training data containing the less frequent n-grams should be able to correct these errors. We present some examples of these in Figure 5.8 (bottom).

Suggestion Results

As explained in Section 5.1.4, we train 4 additional models in Sanskrit to generate different suggestions for the incorrect OCR words. In Table 5.6, we summarize the results, comparing the quality of the suggestion generated by each of these models. The last column in each row is for the percentage of OCR words for which we get the

Suggestion Index	Context for training model	%age of correct suggestions	%age of unique suggestions
1.	PCPW	63.98	63.98
2.	PC	63.34	8.45
3.	SC	55.02	4.48
4.	PCSW	57.34	0.57

Table 5.6: Errors correction by LSTMs trained with different contexts in Sanskrit

correct ground truth word as the suggestion obtained by the corresponding model, that could not be corrected by the models in the rows above them. We achieve overall corrections for around 77 percent of erroneous words using the strategy of obtaining different suggestions for the same OCR word based on different contexts. Using this user-in-the-loop system, we outperform the upper baseline model for Sanskrit as well.

In this section, we discussed the applicability of using LSTM with a *fixed delay* for Indic OCR corrections. The models, however, correct the errors locally. In the next section, we improve these models by providing global language information with each training example with the help of sub-word embeddings.

5.2 Sub-word Embeddings for OCR Corrections

As discussed in Section 4.1, partial word forms (or sub-words) from different global auxiliary sources can help in local OCR corrections. With every character in the OCR text, it is natural to use the information of its context obtained from global language sources to correct it. In this section, we highlight the importance of using sub-word embeddings in the context of each OCR character to augment the input of the basic LSTM model discussed in Section 5.1.3.

An example of a complex conjoined word corrected by the sub-word embeddings based model we propose in this section is illustrated in Figure 5.9. Word embeddings are the vectors which store the information of words along with their context in the language. Some word embeddings also include sub-word units, such as in fastText by Bojanowski *et al.*, 2017, ELMO by Peters *et al.*, 2018, and BERT by Devlin *et al.*, 2018. Of these, we find fastText to be most naturally suited for the purpose of pre-training based on reconstruction of each sub-word using its context in fixed-length sub-strings (refer Figures 5.10, 5.5). Such an embedding helps in representing the

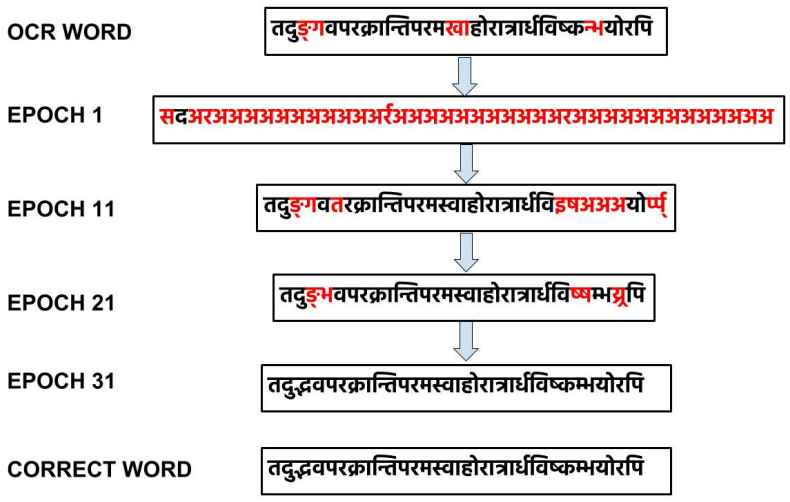


Figure 5.9: Correction flow of our model for a complex word in Sanskrit

information related to the frequency and context of sub-words in the language and especially benefits out-of-vocabulary words at test time. This makes fastText a good fit for the segmentation of the complex conjoined words into generally used words for Sanskrit (Reddy *et al.*, 2018). Sub-word embeddings have also been used in Neural Machine Translation (NMT) and sentence pair modelling by Sennrich *et al.*, 2016; Lan and Xu, 2018, and have proven helpful for producing the vectors for out-of-vocabulary words in that context. The importance of sub-word embeddings has also been highlighted recently by some of the latest works in reading comprehension (Zhang *et al.*, 2018).

Sub-word units, such as n-grams and their (possibly noisy) contexts, can be extracted from the OCR output text, and compared with their frequency statistics over a general text corpus. Text embeddings such as fastText can help compute different statistical functions for the sub-word units. FastText embeddings typically learn such combined function of n-grams and text dependency from the language data. For the OCR correction task, we will consider the frequencies of sub-words in the ground truth of training data as an initial approach (referred to as frequency-based approach) and compare it with modified fastText embeddings as an alternative approach (referred to as fastText approach).

OCR output text is mostly correct (*i.e.*, many more characters are right than incorrect), and the large fraction of errors in the OCR text follow the confusion patterns which arise due to similarly shaped glyphs in the language font. Therefore, for OCR corrections, character-based approaches are generally used (ICDAR, 2017,

2019). We also follow the character-based approaches in our experiments, which we explain in the next section.

5.2.1 Approaches

We now discuss the two approaches we use to augment the input of the LSTM with sub-word level information.

a) Frequency-based approach: Sub-word unit based learning

For each input character x_t in the training sample $\{x_{1:T_x}, y_{1:T_y}\}$, we extract the bag of sub-words with lengths varying from 2 to $l+1$ from a context window of length $2l+1$ around x_t. We only consider the sub-word units that contain the character x_t. Thus the sub-word units that we consider for the t^{th} character in the OCR word $x_{1:T_x}$ are: 2-grams ($\{x_{t-1:t}\}$, $\{x_{t:t+1}\}$), 3-grams ($\{x_{t-2:t}\}$, $\{x_{t-1:t+1}\}$, $\{x_{t:t+2}\}$), *etc.*, up to (l+1)-grams ($\{x_{t-l:t}\}$, $\{x_{t-l+1:t+1}\}$, ... , $\{x_{t:t+l}\}$).

We find the normalized frequency of each sub-word unit in the ground truth of the training data Y, and augment them to the LSTM model, as we discuss in Section 5.2.2.

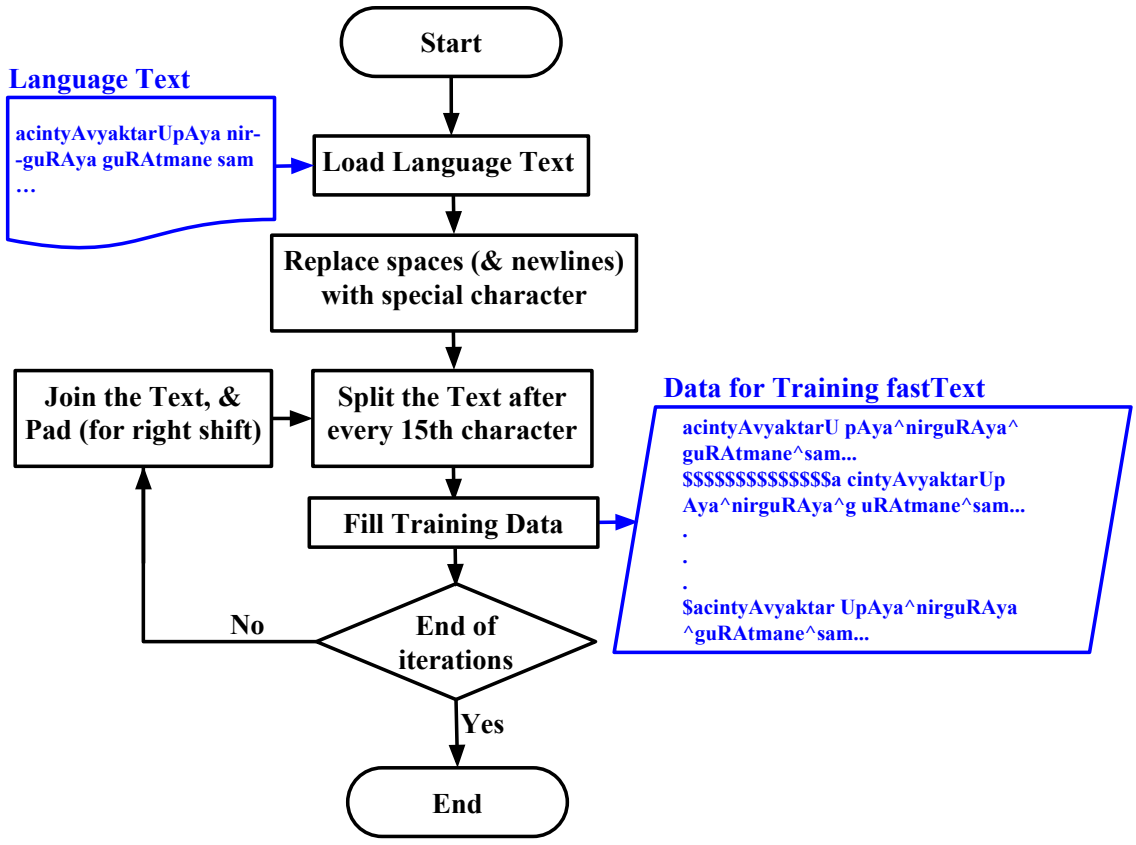


Figure 5.10: Flowchart for transformation of language data for training fastText

b) FastText approach: A new method for training fastText on sub-word units

As a meaningful contribution to the task of Indic OCR corrections, we provide a new procedure for training embeddings such as fastText by Bojanowski *et al.*, 2017 for the work. The training procedure is driven by and based on the observations described in the previous sub-section. While training the fastText embeddings, we consider the substrings of length $2l+1$ in the language. We split the language text at every $2l+1^{th}$ character (including space characters) to form substrings of length $2l+1$, before learning the desired embedding over the entire string. It is important to emphasize that the fastText implementation involves the bag of smaller sub-words (of length 2 to $l+1$ for our case) within the substring (of length $2l+1$) that we obtain. Therefore, this is similar to the learning described in the previous sub-section. In Figure 5.10 we illustrate the flowchart for this process along with an example of language data in the SLP1 (Sanskrit Library Phonetic Basic encoding scheme) format (blue, top left) and corresponding training data obtained using fixed-length substrings of size $2l+1 = 15$ (blue, bottom right). As shown, we first replace each space (and newline) character in the language text by a special character. We then split the data every 15 characters to form substrings of length 15 (adequately padding the end of language text). We then iteratively i) pad the language text in such a way that it considers the substrings starting from the subsequent characters, and ii) repeat the above splitting, 14 more times to include every possible sub-word of length 15 in the language. This transformation also retains adequate context for each substring in the original language text.

We focus on the task of correcting the (possibly incorrect) character x_t in the OCR word $x_{1:T_x}$ to the correct character y_t. Owing to confusions involving multiple characters in the source and/or target, some of the x_t and y_t could be blanks. The sub-word context $x_{t-l:t+l}$, (with l determining the window size) can be further utilized to predict the correct character y_t. The input space of the LSTM model can explode if we provide the sub-word units directly in the form of one-hot-encodings (OHE). We therefore instead provide the information about the context sub-words in the form of normalized frequencies for the frequency-based model and fastText embedding vectors trained with a new procedure that we describe in the next section.

5.2.2 Model

We train the OCR correction models using the global information from the

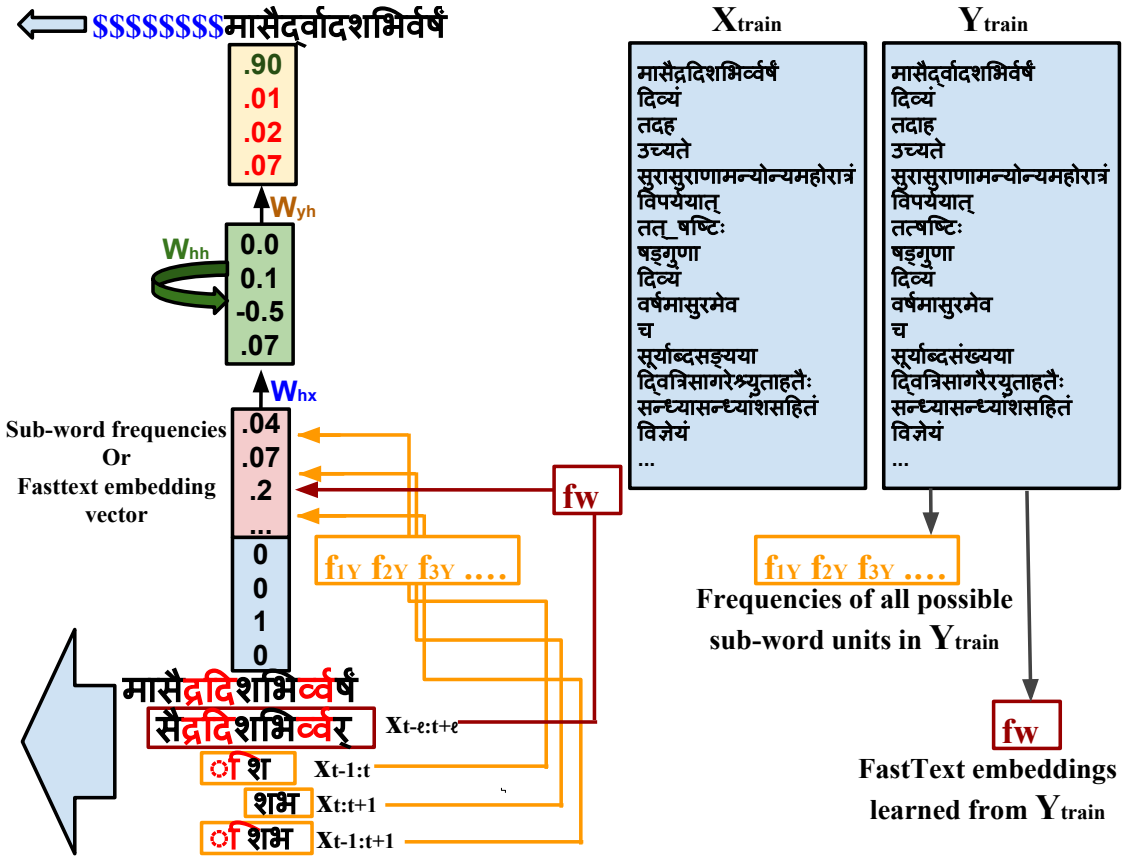


Figure 5.11: LSTM model with 7 units of delay for two types of encodings

training data, in the form of sub-word frequency values as well as fastText embedding vectors derived from the ground truth of training data as shown in Figure 5.11.

To predict the t^{th} character y_t of the correct sequence $y_{1:T_y}$ of length T_y, based on the OCR character sequence $x_{1:t+d}$, an LSTM with *fixed sequential delay* is used in Section 5.1.3. Here d is the context ahead of the location t in the input sequence $x_{1:T_x}$ that is required to resolve an error at that location. For the frequency-based model, a database D is filled with the mapping of all possible sub-words with their frequency in ground truth (i.e. corrected output) from the of training data as shown in Figure 5.11 (orange in the bottom right). Then, we append the one-hot encoding (OHE) of each character x_t at the input with the frequencies (which are derived from the mappings in the database D) of sub-words $x_{t-a:t+b}$ s.t. $0 \leq a, b \leq l$ within a context $x_{t-l:t+l}$ around it (orange in the bottom left). For illustration, only three sub-word units within the context $x_{t-l:t+l}$ with l = 5 is shown in Figure 5.11. In comparison, for using fastText sub-word embeddings in our model, we first train fastText on (the sub-words present in) the constant length $(2l + 1)$ substrings as explained in Section 5.2.1 (shown in the dark red in Figure 5.11). We then concatenate the one-

hot encoding of each input character with the embedding vector of the substring (of length $2l + 1$) present in the context window $t - l : t + l$.

Language	# Word Correction Pairs	μ, σ of Word Length	OOV percentage in test set	Word Error Rate for OCR (OOVs)
Sanskrit	86 k	10.22, 7.98	44.77	51.20 (72.16)
Malayalam	107 k	9.32, 4.93	49.32	38.32 (45.86)
Kannada	118 k	8.42, 3.86	26.59	47.44 (60.14)
Hindi	134 k	5.29, 2.53	23.70	46.80 (48.47)

Table 5.7: Datasets used for our experiments

5.2.3 Datasets used for the Experiments

We work on four Indian languages with varying complexities for the task of document OCR corrections. Table 5.7 summarizes the details of the dataset that we explained in Section 5.1.2. For Malayalam, we obtain $81k$ pairs of OCRed words and their corresponding corrected versions (hereafter referred to as *correction pairs*). We also include the $26k$ correction pairs obtained from a work by Vinitha and Jawahar, 2016. Putting these together, we get a total of $107k$ correction pairs for Malayalam. As shown in Table 5.7, the dataset consists of the order of $100k$ pairs of "OCR word, Ground Truth word" in each of the four languages. The mean and standard deviations of word lengths in Table 5.7 for the different languages are consistent with the fact that Sanskrit has the highest number of word conjoining rules, followed by Malayalam, Kannada and Hindi. We further note that although the out-of-vocabulary words in the Malayalam test set (with respect to training and validation set) are higher in percentage as compared to Sanskrit, still the Word Error Rates (WER) for general OCR words as well as out-of-vocabulary words in Sanskrit are higher than that of Malayalam.

5.2.4 Experiments

The datasets explained in the previous section were split as per the ratio 64:16:20 for training, validation and testing respectively. Following the methodology in Section 5.1.4, we evaluate our models on two tasks, *viz.*, error detection and error correction. We again use the sequence delay of 7 for Sanskrit and Kannada, and 5 for Malayalam and Hindi. We use the context of $l = 7$ characters on each

side of the OCR character x_t to derive the sub-word units, based on the n-gram level features used for training log-linear classifiers in Sections 4.4.2 and 4.3.1. Thus, we effectively use the sub-words of length 2 to 8 for the frequency-based model. For the fastText based models, we use the embedding size of 100 while training the fastText with (sub-word units present in) constant length substrings $x_{t-l:t+l}$ (as described in Section 5.2.1.b). We train the fastText models for 100 epochs on the ground truth of each language dataset. As discussed earlier, we switched off the word level n-grams and use all the sub-words of length 2 to 8 within context $x_{t-l:t+l}$. We use 2×512 sized hidden layer LSTM for all experiments. For training, we use the gradient descent algorithm with the learning rate of 0.002 and the decay of 0.97 per epoch after 10 epochs of training to learn the model parameters. We train models with cross-entropy loss for 200 epochs.

We use the disagreements between the input and output of the model for the error detection task. It is important to note, that this error-detection methodology also allows us to color code the errors at the granularity of characters as shown in Figure 5.11. We use F-Scores as the evaluation measure for error detection. Our models naturally learn to correct errors as we train them on the pairs of OCR word and its ground truth. For error correction, we use measures such as i) Word Error Rates (WER) of the model's output and ii) the percentage of word errors corrected by model.

5.2.5 Error Detection Results

Here, we first discuss the effect of training our model with different fastText embeddings. We then discuss the results on two different encodings described in Section 5.2.1 across various languages.

Effect of training LSTM with different fastText embeddings

It is important to note that the average word length and standard deviation in word length are highest for the Sanskrit dataset, as shown in Table 5.8. Thus, for Sanskrit, we perform experiments with fastText embeddings trained on i) a large generalized corpus, ii) our data, and iii) our data with the new training procedure explained in Section 5.2.1. The results are shown in Table 5.8. As shown in rows 1 and 2, the F-Score of the model with fastText embeddings pre-trained on the ground truth of data is better than the model with fastText embeddings pre-trained on a large amount of general data. This happens probably because there is sharing of sub-words (or domain) or OCR confusions among the training and the test datasets. However,

Method	TP	TN	FP	FN	Precision	Recall	F-Score
Normal training (Wikipedia & web)	93.74	94.19	5.80	6.25	94.57	93.74	94.16
Normal training (our data)	94.90	94.61	5.39	5.10	95.01	94.90	94.95
New training procedure (our data)	**95.11**	**95.39**	**4.61**	**4.89**	**95.71**	**95.11**	**95.41**

Table 5.8: Effect of pre-training fastText with different datasets and proposed procedure in Sanskrit

when we pre-train the fastText embeddings on the ground truth of training data with the new training procedure (described in Section 5.2.1) and use them with our model, the results outperform the other methods. This is shown in the 3^{rd} row of Table 5.8. This supports the claim of contributing a novel, useful training methodology using sub-word embeddings.

Language	TP	TN	FP	FN	Precision	Recall	F-Score
Sanskrit							
Basic LSTM model*	92.63	94.54	5.45	7.36	94.84	92.64	93.72
Frequency-based model	94.49	95.20	4.79	5.51	95.52	94.49	95.02
FastText model	**95.11**	**95.39**	**4.61**	**4.89**	**95.71**	**95.11**	**95.41**
Malayalam							
Basic LSTM model	91.40	96.39	3.61	8.60	94.02	91.40	92.69
Frequency-based model	91.62	**96.42**	**3.58**	8.37	**94.08**	91.62	92.84
FastText model	**94.70**	95.77	4.23	**5.30**	93.29	**94.70**	**93.99**
Kannada							
Basic LSTM model*	98.40	97.18	2.82	1.60	96.92	98.41	97.66
Frequency-based model	**98.64**	96.66	3.34	**1.36**	96.38	**98.64**	97.50
FastText model	98.36	**97.53**	**2.47**	1.64	**97.29**	98.36	**97.82**
Hindi							
Basic LSTM model*	91.96	93.86	6.14	8.04	92.94	91.95	92.44
Frequency-based model	93.68	94.36	5.64	6.32	93.60	93.68	93.64
FastText model	**96.92**	**95.68**	**4.32**	**3.07**	**95.18**	**96.92**	**96.04**

Table 5.9: Error detection results in Indic OCR, *Results in Section 5.1.5

[2]We refer to this model as fastText model for the remaining sections in this chapter.

Results on different languages

In this section, we present results for different Indian languages. We perform experiments for the four languages with the frequency-based model, and the model with fastText embeddings, both trained on the ground truth from the training data. It is important to note that the F-Scores in Table 5.2 were already above 92%. As shown in Table 5.9 the frequency-based model, as well as the fastText model, outperform the Table 5.2 results (referred to as basic LSTM model) for all the experiments (except for Kannada where the results of the frequency-based model are slightly lower than the basic LSTM model). The improvements result from the fact that we provide the context information with each OCR character in the form of sub-word frequencies, or sub-word embeddings, as explained in Section 5.2.1. As shown in the 2^{rd} row of Table 5.9, frequency-based model, that works on the principle of frequencies derived from sub-words in the ground truth of training data, performs as well as the fastText embeddings trained on the same data (as depicted in row 2 of Table 5.8) for Sanskrit. The experiments show that there are gains of 1.38% and 1.80% in F-Score using the frequency-based model and the fastText model, respectively, over the results in Section 5.1.5 for Sanskrit. Furthermore, the percentage increase in F-Scores for Malayalam, Kannada, and Hindi are 0.43%, −0.16%, and 1.30% respectively when we use the frequency-based model, and 1.62%, 0.16%, and 3.69% when using fastText embeddings pre-trained with the proposed procedure. We further note that all our models converge on an F-Score of more than 90% (on the validation set) within the first 20 epochs of training. Thus we gain both higher performance and faster convergence with the pre-trained (or pre-calculated) encodings.

5.2.6 Error Correction Results

In Table 5.10 we show that for Sanskrit, the frequency-based model reduces the word level errors to 17.85%[3], which is 3.56% better as compared to the results of the basic LSTM model (refer Section 5.1.5). The increase in the percentage of word errors corrected by the frequency-based model is 6.71% as compared to the basic LSTM model. The gains increase further with the model based on pre-trained fastText embedding (with procedure proposed in Section 5.2.1). The corresponding

[3]The word error rates are still very high because the original word error rates for the OCR system (shown in the 2^{nd} column of the Table) are very high (51.20 for Sanskrit). The improvements of text-based correction systems rely on the quality of original OCR texts. The similar observations can be made in the ICDAR (2017, 2019) post-OCR competitions.

Language	Word Error Rate (WER)		%age words
	OCR (OOVs)	LSTM (OOVs)	corrected by LSTM
Sanskrit			
Basic LSTM model*		21.41 (32.67)	63.34
Frequency-based model	51.20 (72.16)	17.85 (**28.03**)	70.05
FastText model		**17.72** (28.71)	**70.13**
Malayalam			
Basic LSTM model		11.83 (**16.07**)	75.22
Frequency-based model	38.32 (45.86)	11.55 (16.30)	75.60
FastText model		**10.95** (17.28)	**78.24**
Kannada			
Basic LSTM model*		15.73 (25.71)	69.66
Frequency-based model	47.44 (60.14)	15.53 (27.08)	70.30
FastText model		**15.38** (**25.32**)	**70.90**
Hindi			
Basic LSTM model*		16.71 (29.23)	72.47
Frequency-based model	46.80 (48.47)	14.42 (26.29)	75.59
FastText model		**9.58** (**20.26**)	**84.42**

Table 5.10: Error corrections by our model, *Results in Section 5.1.5

improvements with the frequency-based models in Malayalam, Kannada, and Hindi are 0.28%, 0.30%, and 2.29% in terms of reduction in WER. The percentage of erroneous words corrected by the models increase by 0.28%, 0.64%, and 2.82%. For the fastText model (trained with the proposed procedure), the corresponding reductions in WER are 0.88%, 0.35%, and 7.13% respectively, and the gains in word correction are 3.02%, 1.24%, and 11.95%, respectively. Moreover, it is essential to note that all the models consistently reduce the errors in out-of-vocabulary words. As shown in Table 5.7, for Sanskrit, Hindi, and Kannada, the higher context in the form of sub-word information helps in reducing word error rates in out-of-vocabulary words as compared to basic LSTM model. Interestingly, for Malayalam, we observe somewhat higher word error rates for the out-of-vocabulary words as compared to the basic LSTM model. We conjecture that this is due to differences in the statistics for out-of-vocabulary words between the training and test set, due to the addition of data from a different source, *i.e.*, from Vinitha and Jawahar, 2016 (explained in Section 5.2.1) to the test set. We also note that the correction pairs from this dataset form the 35% of test set, which leads to a high percentage of out-of-vocabulary words in Malayalam with respect to Sanskrit, as shown in Table 5.7.

OCR WORD	PREVIOUS WORK	OUR MODEL (CORRECT WORD)
ब्रह्मगुर्मीप्तामजीवा	ब्रह्मगुप्त_क्त्मामज्जावा	ब्रह्मगुप्तोक्तम्जीवा
_അരികിലേക്കാളരികിലാണത്ഭുതം	അഈഅരികിലേക്കാളരികിലാണത്ഭുതം	നീഅരികിലേക്കാളരികിലാണത്ഭുതം
പദിവ്യൂഹവസ്തു	പദ്മവ്യൂ$ൂവസ്തു	പദ്മവ്യൂഹവസ്തു
सहनशाक्ति	सहनभक्ति	सहनशक्ति

Figure 5.12: Sample correction examples of agglutination (blue-purple) & fusion (dark red) with respect to (previous) basic LSTM model

Figure 5.12 depicts sample errors corrected by our model in different languages. The first column shows incorrect OCR words. The corrections performed by the previous basic LSTM model and the fastText embeddings based model are present in the second and the third column respectively. Here, we show the correct word forms in the language in blue and purple colors, where there is a change of color from blue to purple (or vice-versa) when two words are agglutinated. The fusions are shown in dark red color. As shown, our model corrects the highly complex words that involve agglutinations and/or fusions in different Indian languages.

OCR Confusions	Frequency OCR output	Frequency Basic LSTM	Frequency FastText Model
◌ാ → ◌ീ	756	24	9
क्ष → च	382	14	9
स्व → ख	319	5	4
ല് → ൽ	1087	52	28
ര് → ർ	966	57	22
ന് → ൻ	714	36	19
ತ್ → ರ್	184	12	5
ಐ → ಎ	155	10	5
ಯ → ಇ	126	4	1
◌ೆ → ◌ಂ	707	1	0
थ → य	600	19	17
◌ै → ◌ं	373	3	3

Table 5.11: Top 3 confusions (Correct→OCR) in Sanskrit, Malayalam, Kannada and Hindi

Analysis

We now substantiate how our model improves the detection/correction for top character confusions, and also improves over basic LSTM model (discussed in Section 5.1.3), in the OCR output as motivated earlier in Section 5.2. As shown in

Table 5.11, the basic LSTM model is able to reduce the confusions to a large extent in test data, and our model reduces them further. It is important to note that these confusions are corrected by the models based on their context in different OCR sequences.

We discussed the improvements over the basic LSTM model by augmenting its input with two different types of sub-word embeddings in this section. We worked with the datasets containing around $100k$ OCR words in four different Indian languages. In the next section, we discuss the applicability of character level attention-based models for the task of OCR corrections. We show that such models are highly beneficial for the more extensive datasets in the two ICDAR, 2017, 2019 post-OCR competitions.

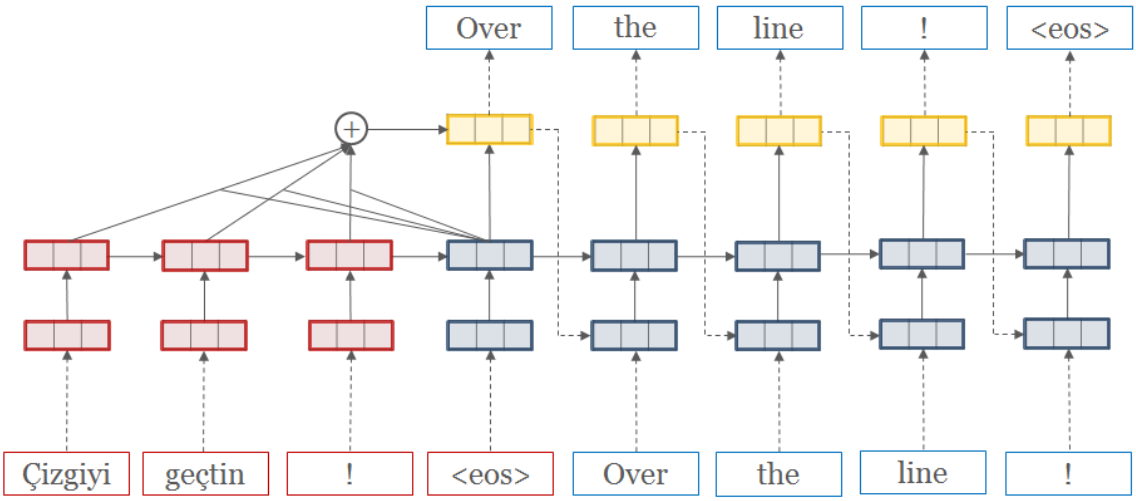


Figure 5.13: A simple word level attention model by Klein *et al.*, 2017

5.3 Attention-based models

The LSTM model that we described in previous sections can be considered simple encoder-decoder model, which encode the characters from the input OCR sequence and decode (or predict) the correct character sequence in output. In such a simple encoder-decoder model, there is no ability to emphasize or focus on the essential characters from the input sequence while decoding the correct character sequence. It is, however, possible to add such a mechanism, by making use of the attention-based models.

Figure 5.13 depicts a simple word-level attention model by Klein *et al.*, 2017. The example we present here is for the supervised machine translation task. The

Language	Source	Type	Dates	E.R.	Char.
English	BL Euro NP	serials	1744 - 1894	4%	1.8 M
	BL Monog	monog.	1858 - 1891	1%	1.2 M
	GT BnF Eng	monog.	1802 - 1911	2%	3.0 M
French	Europeana NP	serials	1814 - 1944	4%	1.0 M
	IMPACT	monog.	1821 - 1864	1%	0.4 M
	GT BnF Fr	mixed	1686 - 1943	1%	2.0 M
	Digit. BnF	mixed	1654 - 2000	3%	0.2 M
	News other	serials	1897 - 1934	4%	0.6 M
	Monog other	monog.	1689 - 1883	3%	1.8 M

Table 5.12: Datasets used in ICDAR (2017) post-OCR competetion

word sequence from the source language serves as input into a 2 layered RNN encoder (shown in red). The decoder RNN (shown in blue) starts decoding (or translating) the input sequence when it comes across the $< eos >$ symbol in the source language. At each decoding step, the mechanism applies attention over all the outputs of the encoder to weight the important input words. The result combines with the current hidden state of the decoder to predict the next word in the target sequence. Thus the attention layers, which are applied to the encoder's output help it to learn to give attention to different context information around the input word being decoded. The predicted output word is then used as input to the next step of to the decoder. The symbol "+" in the circle of Figure 5.13 represents the weighted average of all the vectors. The attention mechanism is said to choose the important words because the weights are generally sparse and are estimated based on the inputs to attention mechanism. Naturally, only the important words are given higher weights by such a mechanism. In the next two sections, we discuss how we use such attention models for character level OCR corrections in two post-OCR competetions ICDAR (2017, 2019).

5.3.1 ICDAR'17 Post-OCR Competition

For the last 30 years, OCR has been an extensive research study. The performance, however, is still not up to scratch specifically for ancient texts and newspaper images. Therefore a research competition was proposed for the error detection and corrections in the English and French OCR texts.

As shown in Table 5.12, a total of $12M$ OCR characters were obtained primarily from newspapers and monographs at the National Library of France (BnF) and the British Library (BL), was used in the competition. The participants used 80% of the data, which consisted of OCR text aligned with Ground Truth text for training and validation. The remaining 20% of data of the data, which contained only OCR text granted at the end of the competition, was used for testing. Eleven teams submitted results, and around six (as shown in Table 5.14) were able to denoise the OCR text.

Our Method for ICDAR'17 post-OCR competition

The method we proposed relies on the Character Level Attention Model (CLAM), which is an attention-based LSTM encoder-decoder model, with beam search used at the decoder's output. We used the open-source implementation by Klein *et al.* (2017) for the model proposed by Luong *et al.* (2015). Here we used one-hot encoded vectors for characters rather than words. The attention model makes use of a 2-layer BLSTM of size 1000 as encoder and a 2-layer LSTM of size 1000 as the decoder.

Training: To deal with both real-word errors as well as non-word errors, we used the characters from input OCR word o_t with space as the character delimiter at the network's input. We also used the space-separated characters from l OCR words on the left of o_t (*i.e.* $o_{t-l:t-1}$) and r OCR words on its right (*i.e.* $o_{t+1:t+r}$). We used \$ as the word delimiter. As the target labels for the network, we used the characters from the ground truth word corresponding to the input OCR word with space as the delimiter. The best results were given by $l = 4$, $r = 1$ for the first 3 datasets, and $l = 6$,$r = 1$ for the last.

Testing: We expect the model to jointly learn the language as well as error patterns in the OCR output. Since our model avoids suggesting changes to the correct words, we considered the words that the model modify as incorrect and remaining correct. We experiment with different contexts that may help identify and correct the erroneous OCR output words and thereby identify the context which gives the best F-score and corrections.

Results

The results for ICDAR'17 post-OCR competition are summarised in Tables 5.13 and 5.10. As can be observed from Table 5.13, the team "WFST-PostOCR" obtained the best results in detecting errors. They proposed Weighted Finite-State (Edit) Transducers (WFSTs, refer Roche and Schabes, 1997), which they composed by estimating the stochastic error models from the aligned training corpus. To address

	Task 1 (F-mesure)			
Corpus part	ENG-mono.	ENG-period.	FR-mono.	FR-period.
NbTokens (E.R.)	63371 (10%)	33176 (15%)	32274 (5%)	48356 (7%)
5gram-KN-LV	0.05	0.51	0.25	0.35
LSTM Monochar	-	-	0.17	-
Seq2Seq	0.45	0.39	-	-
BiLSTM	0.09	0.06	0.05	0.05
2-pass-RNN	0.66	0.66	0.43	0.60
Anavec	-	-	0.24	0.42
WFST-PostOCR	**0.73**	**0.68**	**0.55**	**0.69**
CLAM	0.67	-	0.36	0.54
Char-SMT/NMT	0.67	0.64	0.31	0.50
EFP	0.69	0.54	0.40	0.54
MMDT	0.66	0.44	0.36	0.41

Table 5.13: F-scores for error detection in ICDAR (2017) POCR competition

the segmentation errors, the team applied the error model (with at most one edition) to the words, word splits, and concatenated words. Moreover, they used Google Books n-gram corpus to generate the dictionary and 2-gram language models, which they composed with the error (or substitution) models in the WFSTs. The best path, when applied over the composed model, determined the output sequence. The team preserved the case of the output text as per the source OCR words. The symbol "-" in all the tables, in the present and the next sections, represents that results were not exploitable, *i.e.*, they were not submitted in the proper format or were incomplete.

Our team "CLAM" achieved the third or fourth highest F-scores for the error detection task on three datasets as shown in Table 5.13.

The correction results are summarized in Table 5.14. Here, the values on the left side of symbol "/" represent the percentage of words auto-corrected by the team mentioned in the first column. The value on the right of "/" represents the semi-automatic corrections based on the ranked suggestion list, along with weights for each suggestion, submitted by the team. The symbol "=", in all the tables from the present and the next section, represents no global gains (= / =) or same results (for both automatic and semi-automatic approaches) respectively. The winning team "Char-SMT/NMT" used multiple character-based statistical and neural machine translation models similar to ours for each language and type (with 10% validation set), the details of which are as follows:-

	Task 2 (%Improvement) Auto (top1) / Semi (weighted mean on top5)			
Corpus part >	ENG-mono.	ENG-period.	FR-mono.	FR-period.
NbTokens (E.R.) >	63371 (10%)	33176 (15%)	32274 (5%)	48356 (7%)
5gram-KN-LV	-	-	-	-
LSTM Monochar-	=/=	-	=/=	-
Seq2Seq	=/=	=/=	-	-
BiLSTM	=/=	-	-	-
2-pass-RNN	-	=/=	-	-
Anavec	5%/-	=/=	=/=	=/=
WFST-PostOCR	28%/=	=/=	=/=	=/=
CLAM	29%/=	22%/=	1%/=	5%/=
Char-SMT/NMT	**43%/=**	**37%/=**	**44%/=**	**29%/=**
EFP	13%/11%	=/=	23%/=	5%/4%
MMDT	20%/=	=/=	3%/=	2%/=

Table 5.14: Auto-corrections/Suggestions by each team in ICDAR (2017) POCR competition

1. The basic models corrected each word separately.
2. The models corrected the words based on a context window of 2 previous words and 1 subsequent word (similar to our models where we take up to 6 previous and 1 subsequent word in the context).
3. To embed time[4], with groups of 50 years, factored-neural models (Alexandrescu and Kirchhoff, 2006) were also used.

An OCR word is marked as an error if:-

1. There is a change in word based on lowest edit distance in the validation set, or
2. There is a dissimilarity in the predictions from the five best systems, or
3. Lookup in the correct training set fails, or

[4]Some of the word forms are prominent for a specific period, and the usage changes over time. E.g., as per the statistics of Google's "English (2012)" corpus given in `https://tinyurl.com/yan5zo58`, the word "Henceforth" was used more often than the term "Subsequently" from the year 1800 to 1824, and then onwards the usage of the word "Subsequently" is more frequent. So embedding the time information to the input can be useful for the model to modify or correct such words.

Lang.	Source	#file	#character	μ CER	σ CER
BG 1	IMPACT	200	399 636	14.96	12.49
CZ 1	IMPACT	200	274 130	5.79	12.07
DE 1	IMPRESSO	102	575 416	13.54	14.45
DE 2	IMPACT	200	494 328	39.67	16.09
DE 3	Dta19	7 623	10 018 258	24.22	3.26
DE 4	EML	321	509 757	23.95	3.94
DE 5	KA	654	818 711	24.19	3.64
DE 6	ENHG	773	935 014	30.47	3.00
DE 7	RIDGES	415	527 845	24.20	3.63
EN 1	IMPACT	200	243 107	21.28	20.25
ES 1	IMPACT	200	517 723	27.51	17.96
FI 1	NFL open	393	1 960 345	5.67	3.94
FR 1	HIMANIS	1 1722	792 067	7.14	10.09
FR 2	IMPACT	200	227 039	15.48	13.94
FR 3	RECEIPT	1 968	742 574	9.27	10.91
NL 1	IMPACT	200	764 648	26.84	23.42
PL 1	IMPACT	200	307 144	38.16	18.09
SL 1	IMPACT	200	261 060	10.16	15.83
10	**18**	**15 221**	**22 368 802**	**20.14**	**11.50**

Table 5.15: Datasets used in ICDAR (2019) post-OCR competetion

4. The word and its any neighbour (corrected/OCR word itself) do not occur in the training set, but their concatenation does.

Suggestions from the best model formed the entry for the auto-correction task. The correction mechanism also leveraged the predictions (from different models) with the lowest edit-distance on the validation set. The suggestion list for semi-automatic corrections was generated based on prediction frequency.

Our team "CLAM" achieved the second-highest results for the OCR corrections for three datasets as shown in Table 5.14. In the next section, we discuss the approach we used in the ICDAR (2019) post-OCR competition.

5.3.2 ICDAR'19 Post-OCR Competition

The ICDAR (2019) post-OCR competition also proposed the tasks of error detection and corrections similar to the ICDAR (2017) competition, but over a larger dataset of $22M$ OCR-ed symbols (754025 words) from 10 European languages. The original dataset consisted of around $22M$ OCR-ed symbols. The participants used 80% of the dataset for training and remaining 20% (without ground truth) for testing. Thirty four teams registered for the competition, out of which only 5 teams submitted their results.

Newspapers, ancient documents, ancient manuscripts, and purchasing receipts covering Bulgarian, Czech, Dutch, English, Finish, French, German, Polish, Spanish and Slovak were used as shown in Table 5.15. As per ICDAR (2019), the digitized documents were from different collections in national libraries and universities. The corresponding ground truth texts were obtained from the initiatives such as HIMANIS[5], IMPACT[6], IMPRESSO[7], Open data of National Library of Finland[8], GT4HistOCR by Springmann *et al.*, 2018 and RECEIPT by Artaud *et al.*, 2018.

Our Method for ICDAR'19 post-OCR competition

We again use the open-source system by Klein *et al.*, 2017, and the model used in Section 5.3.1. Some training and testing details differ from the previous approach, which we describe below.

Training: To take care of real-word errors as well as non-word errors, at the network's input we used the characters (with space as character delimiter) from input OCR word o_t. We also use the characters from l and r OCR words (with \$ as the word delimiter) on its left: $o_{t-l:t-1}$ and right: $o_{t+1:t+r}$. We also appended each input with a language flag and trained a single model jointly on all languages. At the network's output, we used the characters (with space as the character delimiter) from the ground truth word g_t corresponding to the input OCR word o_t. We analyzed the complete dataset and observed that there is a maximum of 10 space related errors where OCR systems introduced non-existent spaces. To successfully remove such errors, we choose $l = 10$ and $r = 10$ *i.e.* 10 words on the left as well as the right of OCR word o_t as context.

Testing: We trained our model to jointly learn the languages as well as the error patterns in the OCR output. We used the methodology followed in Section 5.3.1 for error detection, *i.e.*, we considered the word modified by the model to be incorrect and correct otherwise. We applied the edit-distance algorithm between the input and output of the model to find the length of the erroneous tokens. We considered the model's outputs as the suggestions for the error correction task.

Language	BG	CZ	DE	EN	ES	FI	FR	NL	PL	SL
CCC	0.77	0.70	0.95	0.67	0.69	0.84	0.67	0.71	0.82	0.69
CLAM	0.68	0.41	0.93	0.45	0.56	0.51	0.45	0.61	0.72	0.54
CSIITJ	-	-	-	0.45	-	-	0.42	-	-	-
RAE1	-	-	0.90	0.53	0.62	0.44	0.42	-	-	-
RAE2	-	-	0.89	0.57	0.60	0.46	0.45	-	-	-
UVA	-	-	-	0.47	-	-	-	-	-	-

Table 5.16: F-scores for Error Detection in ICDAR, 2019 POCR Competition

Language	BG	CZ	DE	EN	ES	FI	FR	NL	PL	SL
CCC	9/8	6/=	24/=	11/=	11/6	8/=	5/=	12/10	17/16	14/12
CLAM	-2/-3	-1/=	-7/=	0.4/=	-1/-5	**44/=**	4/=	-3/=	-2/=	0/-1
CSIITJ	-	-	-	2/1	-	-	-	-	-	-
RAE1	-	-	15/=	9/=	7/=	7/=	26/=	-	-	-
RAE2	-	-	14/=	6/=	7/=	6/=	20/=	-	-	-
UVA	-	-	-	0/=	-	-	-	-	-	-

Table 5.17: Percentage of Corrections/Suggestions by each Team in ICDAR, 2019

Results

As can be concluded from Tables 5.16 and 5.17, our team "CLAM" (Character Level Attention Model) secured the second position in the competition. The overall winner (referred to as team "CCC (Context-based Character Correction)") used the character level sequence to sequence model with attention mechanism similar to ours. Their detection method, however, exploited the pre-trained multi-lingual Bidirectional Encoder Representations from Transformers (BERT) by Devlin *et al.*, 2018. The BERT output of all the sub-words from the OCR word served as an input for a sub-word level neural network classifier with convolution and fully-connected layers. If the model assigned two or more subwords as erroneous, they marked the complete word as incorrect. For error correction; the characters from the OCR word, along with context from the BERT (fine-tuned at the detection task stage), served as an input to an attention model. The encoder for correction model used by "CCC", was a BLSTM (similar to our model) and additionally, both encoder and decoder shared the same character embedding. Finally, they also used the beam search for auto-corrections and suggestions.

[5] `www.himanis.org`

[6] `www.digitisation.eu`

[7] `https://impresso-project.ch`

[8] `https://digi.kansalliskirjasto.fi/opendata`

As shown in Table 5.17, "CLAM" secured the highest corrections of 44% in Finnish, which is significantly higher than the overall winner (who achieved 8% corrections in Finnish). This happened because our approach is motivated by the works on languages rich in inflections, especially Sanskrit and because Finnish is similar to Sanskrit in inflections (as proposed by Sommer, 2016). Moreover, we were the one out of the two participants who proposed an efficient model on all 10 languages for both the tasks.

5.4 Conclusion

In this chapter, we have demonstrated the use of an LSTM with *fixed sequential delay*, for jointly learning error patterns and language models. We have shown that these models are robust at fixing errors in OCR output, where the dataset has around $100k$ word pairs, and perform reliably in correcting them. We have also demonstrated the usefulness of our model by performing several experiments on the OCR texts of multiple Indian languages with varying scripts and complexities. The model sets new benchmarks for error detection and correction (both automatic and semi-automatic) tasks in Indic OCR. We further show that augmenting the input of the LSTM model with sub-word embeddings further improve the performance measures. We have presented the works on LSTM and sub-word embeddings in the two ICDAR publications; Saluja *et al.* (2017a, 2019b). For the more extensive datasets with around $1000k$ word pairs, we show the effectiveness of attention-based LSTM models we have used in the ICDAR, 2017 and ICDAR, 2019 post-OCR competitions. In the next chapter, we will use similar attention-based models for reading modern Indian (as well as French) street signs and license plates.

Chapter 6

Reading Indian Street Signs

Scene-text spotting or photo OCR has many applications such as helping the visually impaired, allowing travellers to translate texts on signboards, and also robotics to to read scene-texts for autonomous cars and in indoor/ outdoor environments. As discussed in Section 1.6.2, photo OCR systems for Indian scenes suffer from high word error rates due to the scarcity of language-specific datasets. We discuss the problem of low accuracy of existing systems for Indian street signs with some examples in Section 6.1. In order to tackle the problem of data scarcity, it is important to understand that obtaining a large amount of labelled training data is an expensive and human-intensive process. Many state-of-the-art systems collect large-scale noisy labels by combining a large amount of unlabelled data, with a modest amount of labelled data (for example in an image search for landmarks by Kennedy and Naaman, 2008). Another way to deal with the problem of data scarcity is training with noisy labels, which was extensively studied by Hedderich and Klakow, 2018. In the present work, we use a state-of-the-art text-spotter to obtain noisy labels for a large number of video frames with Indian traffic and then clean the data using a pattern grammar wherever the domain knowledge is available (refer Section 6.2). We augment the training data with synthetic images (and corresponding labels) obtained using SynthText (Gupta *et al.*, 2016), thus increasing the coverage of training. We describe this process in Section 6.2.1. We then train a CNN-RNN model (described in Section 6.5) for recognition at the word-level. The end-to-end model that we describe in Section 6.6 utilizes an inception-based CNN followed by an attention-based RNN. We present the details of the architecture in Section 6.7. To further improve the locating and recognizing power of the model, we further incorporate location-aware multi-headed attention. We present the results in Section 6.8. The work (Saluja *et al.*, 2019a) sets new benchmarks for three different datasets with

varying complexities. We also present StreetOCRCorrect, in Section 6.9, an interactive framework to correct errors in street text videos. We then conclude the chapter in Section 6.10.

6.1 Ineffective systems for Indian street signs

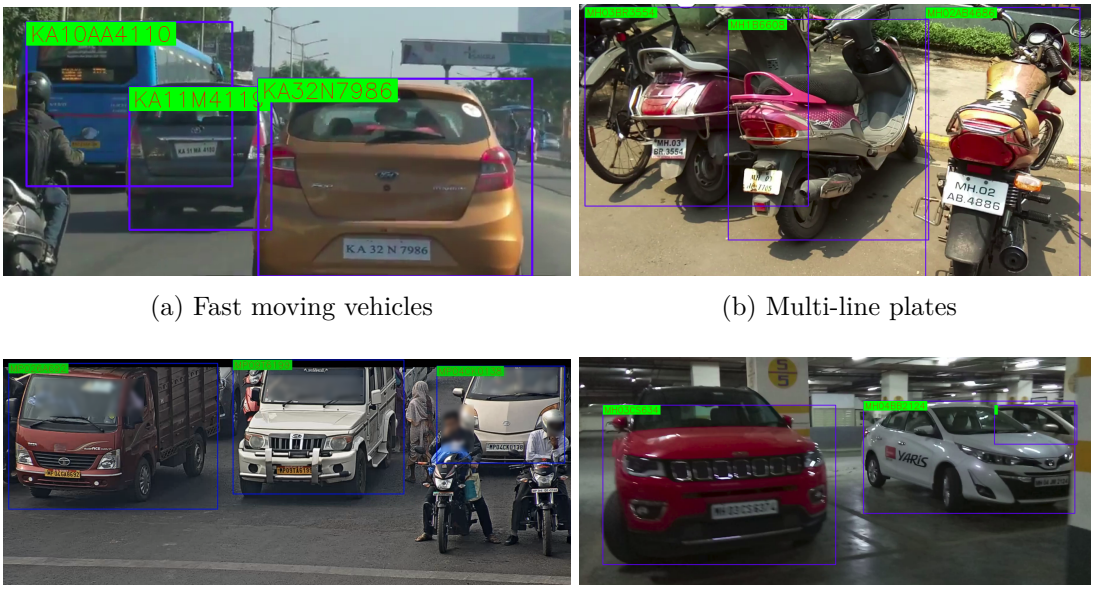
In Section 1.6.2, we summarized the state-of-the-art systems for photo OCR in Indian languages. We observed that such systems have high word error rates (65.80% on Bengali scene images for End-to-End method for Multi-Language scene Text recognition (E2E-MLT) by Bušta *et al.*, 2018 and 57.1% on Devanagari word images by Mathew *et al.*, 2017). It happens due to scarcity of real-world datasets (2k Bengali scene images by Nayef *et al.*, 2017 and 1k Devanagari, Telugu and Malayalam word images by Mathew *et al.*, 2017) in Indian languages. We have also presented a simple street sign image containing a mixture of Bengali and English, and its OCR output using E2E-MLT in Section 1.7.3. As discussed, extra noise characters owing to segmentation errors, as well as missing characters, appear in the photo OCR text.

1 গভণ্মন্ট টপ্রসে সডি ডিত্)
تلاصع دم
GOVT. T.PLACE CECNOI NORTHD
Courtesys THEKOLKATA TA MUN M AMUNICIPALC ALCOR LCORPORATION
و

Figure 6.1: On the top is the complex image[1] in Bengali (and English), on the bottom is E2E-MLT output

Running the E2E-MLT system on a complicated image containing a mixture of Bengali and English with a different zoom level and varying sized characters is shown to have more complicated OCR errors (see Figure 6.1) as compared to the

[1]Image courtesy: `https://upload.wikimedia.org/wikipedia/commons/6/6a/Government_Place_North_Signage_-_Kolkata_2011-12-18_0108.JPG`.



(a) Fast moving vehicles

(b) Multi-line plates

(c) Red Light (distant camera)

(d) Basement parking (dark scene)

Figure 6.2: Sample chaotic scenes with predictions of our model

previous example. Moreover, unusual words with redundant OCR characters arise in E2E-MLT. All this makes the OCR correction process in Indian scene-text images cumbersome. We thus collect more datasets for photo OCR in Indian languages and develop the OCR systems for reading Indian street signs. This approach differs from the efforts on document OCR for which we developed an interactive system for reliable OCR corrections in Section 4.1. We now investigate the methods to generate large scene-text datasets for training deep models in the next two sections. Determining the correct reading order over multiple text segments occurring in the same scene is another crucial problem that has received relatively little attention in the scene-text literature. With the success of end-to-end models (Wojna *et al.*, 2017; Bartz *et al.*, 2017) that can train without any supervision at the level of individual text-boxes, a natural next step is to investigate if this success can extend to determining the correct reading order. Thus we analyze results for determining the natural reading order over scenes with varying complexities.

6.2 License plate recognition

The particular problem of spotting license plates in scenes is useful in surveillance, toll collection, parking systems, developing smart cities, *etc.* The problem involves several challenges, such as the ability to handle a diversity of fonts and patterns, robustness to geometrical transformations, composite backgrounds, and

camera positions, as shown in Figure 6.2. The end-to-end model that we describe in Section 6.6, can successfully jump from one part of multi-line license plates (as shown in Figure 6.6 bottom) to another in the correct reading order.

6.2.1 Dataset Generation

To generate labels for the dataset, we first obtain noisy predictions on every frame of each video (for different conditions given in Table 6.1) using the state-of-the-art text-spotter, *viz.*, DeepTextSpotter by Bušta *et al.*, 2017. We then filter out instances in the dataset that do not follow the license plate grammar. For analysis, we calculate the accuracy (at the word-level) of the noisy predictions on an annotated set of $1k$ samples. We observe that 73% of predictions that follow the license plate grammar are indeed entirely correct. For data augmentation, we further apply the filtered predictions across the video to correct other frames in the video as follows:

- If two successive (but not consecutive) filtered frames, f_i and f_k (where $i < k-1$), have identical predictions: $p_i = p_k$;
- and there exist intermediate frames $\{f_j\}$ $(i < j < k)$ in the original video, for which no prediction fits the grammar (or the DeepTextSpotter makes no prediction)
- then we assign the prediction $p_j = p_i (= p_k)$ to all the intermediate frames $\{f_j\}$.
- To obtain text-boxes for the intermediate frames $\{f_j\}$, we apply linear interpolation on text-boxes from the previous frame f_i and from the subsequent frame f_k.

6.3 Reading street signs

Apart from the challenges discussed in Section 6.2 for reading license plates, the problem of reading street signs also involves variations such as varying backgrounds, multiple languages, and texts in paragraphs. We perform experiments with a multi-headed attention mechanism on the French Street Name Signs (FSNS) dataset and a new Indian street signs dataset. We release a new multi-lingual dataset of $1k$ videos (with text in Hindi, Marathi, and English, as shown in the sample images in Figure 6.3, bottom). The videos in this dataset have a frame-rate of 25 fps and an average duration of 3 seconds. Inspired by the noisy annotation process in the

Figure 6.3: Top: FSNS sample, Bottom: Indian street sign samples

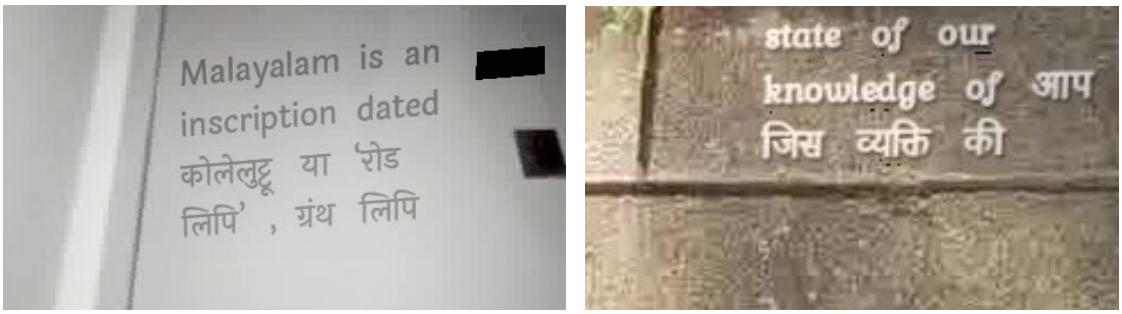


Figure 6.4: Sample synthetic scenes with Devanagari & Latin scripts.

previous section, we record each video in such a way that it covers an Indian street sign from different orientations[2]. We also augment the training dataset with $700k$ synthetic scenes obtained from SynthText (Gupta *et al.*, 2016) modified to include large multi-script sequences, some of the samples for which are shown in Figure 6.4[3]. We summarize our datasets in the next section.

6.4 Datasets used for our experiments

We employ datasets of varying complexities, as shown in Table 6.1, working with various illumination and weather conditions for the problem of license plate recognition in chaotic street scenes. The conditions and duration of the videos, which we use to create the datasets, are described in Table 6.1 (top). Our work can be useful for any scene-text recognition problem with labels that generally follow a particular

[2]The dataset can be requested from `https://www.cse.iitb.ac.in/~rohitsaluja/project`

[3]The source code for generating synthetic data is available at `https://github.com/rohitsaluja22/OCR-On-the-go`

Type of Dataset	Duration/Quantity	Variations	Max seq. len.	μ, σ (I_{avg})
Day time traffic video Night time traffic video Night time with rain	36 hr 30 mins 15 hr 56 mins 2 hr 28 mins	motion blur, chaotic, illumination, single/dual line	10 (per plate)	86.45, 12.94
FSNS Dataset	966k train, 39k validation, & 43k test data-points	motion blur, multi-line,& multiple views	37 (per signboard)	95.71, 26.01
Indian Street Signs	1000 videos each covering a sign from multiple orientations	motion blur, varying backgrounds, multiple languages & multi-line	180 (per signboard)	111.64, 32.55

Table 6.1: Datasets used for our experiments. I_{avg} stands for Average Image Intensity.

pattern or grammar. For the FSNS dataset, each data-point consists of four images of a street sign (shown in Figure 6.3, top) and a corresponding label. In Table 6.1, we provide the quantitative description of the FSNS dataset and a new Indian street signs dataset that we have released. For Indian street signs, each of the videos covers a single street sign from different orientations. Samples from these datasets are shown in Figure 6.3, bottom. We find that in addition to higher sequence length, the street-sign images also have a higher mean and standard deviation of average image intensity I_{img}, as compared to the license plate images. We now describe our models in the next two sections.

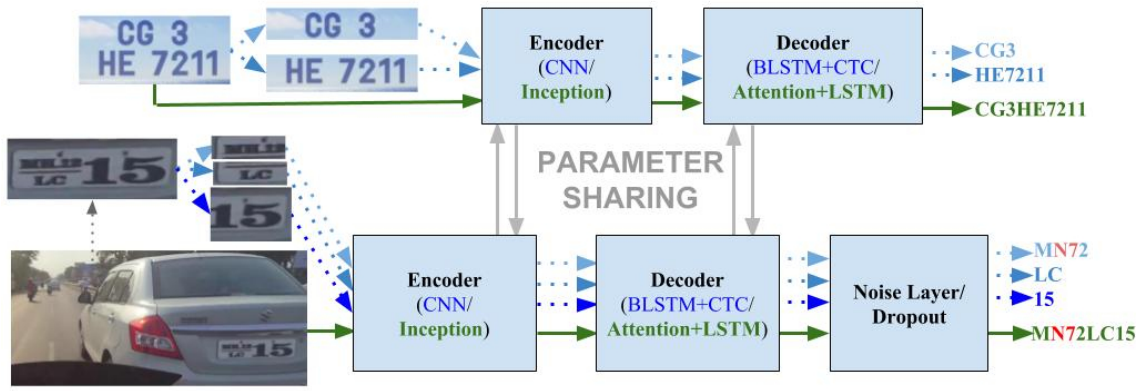


Figure 6.5: Training models on synthetic data (top) and real noisy labelled data (bottom)

6.5 Baseline model

As a baseline, we use the seven-layered convolutional neural network (CNN) to extract the features from license plate images, followed by a two-layer bi-directional long short-term memory (BLSTM) for decoding the features, and a connectionist

temporal classification (CTC) layer for aligning the decoded predictions. We use the TensorFlow implementation described in Shi *et al.*, 2017. However, as we will discuss further in Section 6.6, the problem of multi-scale variation can be handled by using the inception-based CNN as the encoder. Moreover, we handle the challenge of reading the characters at different locations in a scene by using an attention-based RNN decoder, thus enabling end-to-end recognition in the scene-text images. As shown in Figure 6.5 (follow dotted blue arrows), the baseline model is trained on:

1. Synthetic clean data: $\{X_s,Y_s\}$, with X_s being the synthetic license plate image (or sub-plate image in the case of a multi-line license plate), and Y_s being the corresponding clean labels.

2. Real noisy data :$\{X_r,Y_r\}$, with X_r being the real license plate image (or sub-plate image) and Y_r being the corresponding noisy labels produced using the state-of-the-art DeepTextSpotter.

Inspired by the literature (Hedderich and Klakow, 2018) on training neural networks with noisy data, our model shares parameters between the networks that are trained on the clean and noisy datasets, respectively. We argue, however, that our case is different from previous work, owing to three distinctive properties:

1. Firstly, the set of synthetic input images $\{X_s\}$ are visually distinct from the set of real input images $\{X_r\}$ and are not as useful as the real images for the test data. We observed in experiments that our models, when consecutively trained on the two datasets, tend to overfit to the dataset that we initially use. We therefore randomly shuffle the order in which the two datasets get used in each epoch.

2. Secondly, a dropout ("keep probability" = 0.5) is applied to the last stage of the decoder, which acts to prevent overfitting even in the case of noisy labels. We argue that the additional noise correction layer (as advocated by Hedderich and Klakow, 2018) is irrelevant in this case as overfitting to noisy characters can be avoided by applying sufficient regularization through dropout (Srivastava *et al.*, 2014).

3. Moreover, we observe in the literature by Hedderich and Klakow, 2018 that the noise layer significantly helps in the case of less clean training data, and the results do not improve significantly from the addition of large amounts of clean data. Since we train our model with a large amount of clean as well as noisy labelled real data, we avoid using the noise correction layer.

We also train an end-to-end model, which we describe in the next section, on synthetic as well as real datasets as shown in Figure 6.5. In the figure, the license plate image flows in multiple splits through the *baseline model*, shown by dotted blue arrows. Note that for the *end-to-end model*, the entire image flows through the model together, indicated by green arrows. The *end-to-end model* avoids splitting of multi-line license plates and makes it easier for the model to learn patterns.

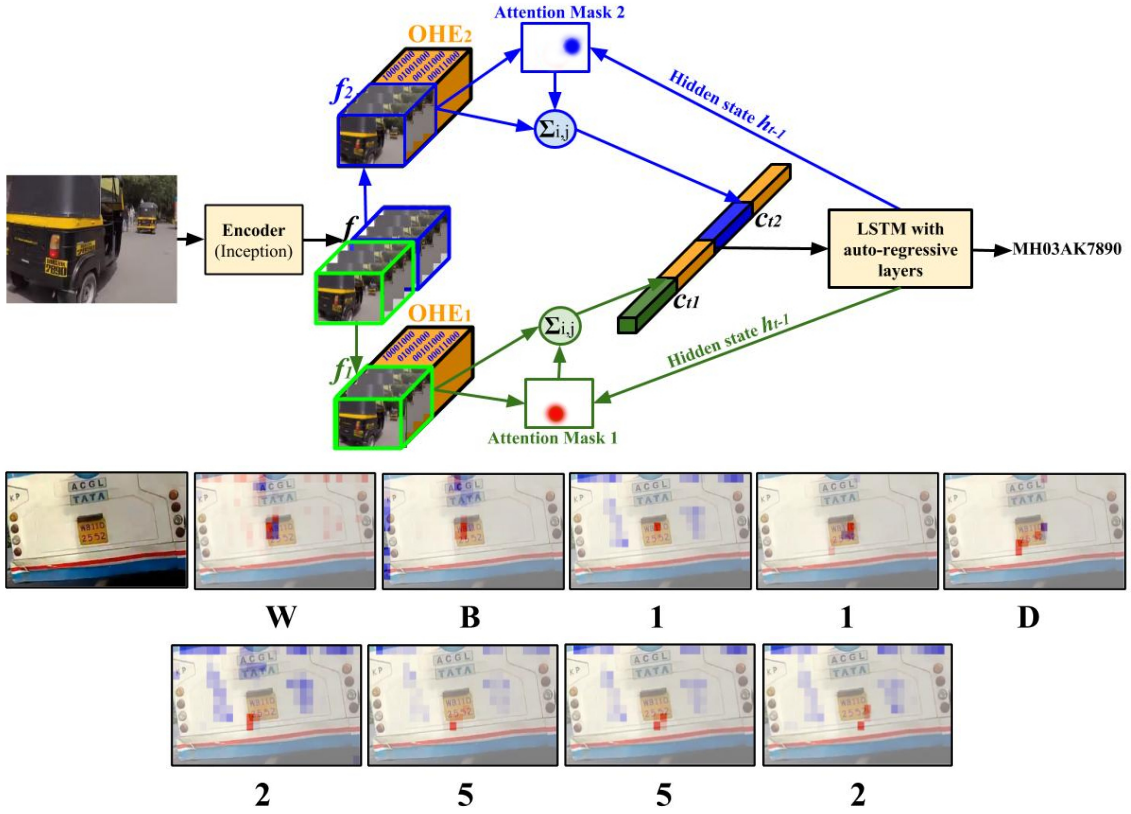


Figure 6.6: Top: Two-headed split-attention based model. Bottom: Attention masks, note that the two masks (shown in red and blue) have unique coverage.

6.6 OCR-on-the-go model

We develop an end-to-end model by extending the Tensorflow implementation of attention_ocr by Wojna *et al.*, 2017. It is important to note that the vehicles appear at different scales in the scene, as shown in Figure 6.2 (a,b,d). Thus, a powerful encoder is needed to capture the multi-scale variation across the license plates. Furthermore, as shown in Figure 6.2 (c), similarly scaled license plates are present at varying locations in the scene. Moreover, license plates exist at varying orientations in the scenes (Figure 6.2 (b,d)). Thus attention-based models are important to locate the character images in the scene.

Our model depicted by Figure 6.6 has the following components:

1. As a powerful encoder, the inception-based CNN by Szegedy *et al.*, 2016 learns to extract the features f from the input image. One of the important parts of the inception-based network is that it has varying sized convolution layers in parallel, which helps in learning the text images at different resolutions.

2. With attention-based LSTM as the powerful decoder, we learn the attention over (i) the features from one of the middle layer of the inception-based network, (ii) one-hot-encoded (OHE) vectors e_x and e_y for both x and y coordinates of the features, and (iii) hidden state of the LSTM at the previous time step of decoding. The OHE vectors for the coordinates provide location awareness to the network as advocated by Wojna *et al.*, 2017, thus making it possible to jump from the upper right character of the multi-line plate to the lower-left character, as shown in Figures 6.2 (d) and 6.6.

3. An LSTM layer takes the context vector from the attention layer as well as the previous OHE output of the decoded sequence, thus learning the language model (or license plate grammar in our case) via auto-regression.

We also experiment with a multi-head attention mechanism, which consists of several attention layers running in parallel as described by Vaswani *et al.*, 2017. We split the encoded features f into two or more parts and learn separate attention masks over each. Moreover, we keep each attention mask location-aware by appending to their feature set one-hot encoded vectors of the x and y coordinates within the image. The splitting of features reduces the computational overhead of the multi-head attention approach, and also allows the different "attention heads" to learn different information from the respective features. It also leads to an ensemble effect. Finally, context vectors obtained by applying the attention masks to corresponding splits (concatenated with OHE position vectors), are concatenated together to form the input to the LSTM. Figure 6.6 illustrates such a model for a two-headed attention. As shown, the features f are split into two parts f_1 and f_2 (shown in green and blue colors). The context vectors c_{t1} and c_{t2} are finally concatenated to form the input of the LSTM layer. For reasons explained in Section 6.5, we train by randomly switching between the models for clean synthetic data and noisy labelled data, and incorporate the dropout (with keep probability = 0.5) in the last layer of the LSTM to avoid overfitting to noisy characters. The complete pipeline is given in Figure 6.5. We refer to our end-to-end model as *OCR-on-the-go* since we train on continuous real video frames. We discuss this in the next section.

6.7 Experiments

We experiment on three different datasets with varying complexities (as described in Section 6.4). Firstly, we work on license plate recognition in chaotic Indic scenes with noisy labelled real data as well as clean synthetic data. To obtain real data with noisy labels, we use DeepTextSpotter proposed by Bušta *et al.*, 2017. The DeepTextSpotter model is trained on the SynthText dataset, as well as the ICDAR datasets (Gupta *et al.*, 2016; Karatzas *et al.*, 2013, 2015). Since these datasets do not cover license plate images, the overall performance is not satisfactory, whereas we obtain 73% word-level accuracy on the data that follows the license plate grammar. We work on 480×260 images for end-to-end experiments, and all the license plate/sub-plate images are resized, with bilinear interpolation, to 32×100 images for input to the baseline model. To obtain clean data, we synthesize a large number of scene images with text from Indian license plates using SynthText (Gupta *et al.*, 2016). For each license plate, we select a random 280×460 crop around it (covering the other license plate images with black pixels if they exist in the crop). For the baseline, we obtain the license plate images similar to the one shown in Figure 6.5 (top-left). To obtain all the synthetic images for experiments, we use 18 freely available license plate fonts[4]. Using the method described in Section 6.2.1, we obtain $1063k$ frames with license plates and corresponding predictions for training using 55 hours of video data. We train models on $1063k$ frames with a 64:16:20 train:val:test split. Additionally, to improve generalization across various Indian states, we use $187k$ synthetic scenes (only while training). Figure 6.5 (top-left) depicts an example of a synthetic image. Such a dataset improves the generalization because it covers i) license plate images from multiple perspectives, ii) several possible license plate patterns across all Indian states, and iii) clean labels, which are not present in the real data.

For the French Street sign data (FSNS), we avoid the use of synthetic dataset and dropout layer since it is significant in quantity and contains clean annotated labels (as compared to the noisy-labelled license plate dataset) as well as multiple views for each data point (as illustrated in Figure 6.3 (top)). Therefore, each training and testing sample from the FSNS dataset utilizes the encoder four times, once for each view. We further perform experiments with a mixed-6a layer from the inception-resnet-v2 as an encoder for all our datasets. Using this encoder, we obtain the features of size $14 \times 28 \times 1088$ for license plate images and Indian street sign

[4]Available at `https://fontspace.com/category/license%20plate`

images, and $7 \times 7 \times 1088$ for each view in FSNS images. For Indian street signs, we obtain around $79k$ frames, each of size 280×460, from the videos described in Table 6.1. We use initial $50k$ frames for training, the next $12k$ for validation, and the remaining $17k$ for testing. We also augment the training dataset with $700k$ synthetic scenes obtained from SynthText (modified to include large multi-script sequences, as shown in Figure 6.4) with around 50 Unicode fonts[5].

6.8 Evaluation

We now evaluate our models on the three different datasets, with varying complexities, as discussed in Sections 6.4 and 6.7.

6.8.1 Visualization of attention masks

In Figure 6.2., we show the predictions of the *OCR-on-the-go* model for some of the complex test cases. We present results visualizing the multi-head attention masks (re-sized to image size with nearest neighbour interpolation) for text recognition in Figure 6.6 (bottom). It is important to note that, specifically for license plate scenes, one of the attention masks moves over each character due to the randomness of the chosen character at each position on the plate. In contrast, we observe that for other datasets that the attention mask often remains idle (on the edges of the image) after reading the first few characters of highly frequent words in the language (refer attention masks in the literature by Wojna *et al.*, 2017, which tend to wander over the edges after the model reads few characters from a common word). This is probably due to i) the implicit language model, ii) the large receptive fields around each feature location, or iii) the mask not finding the character due to occlusion in the image and therefore failing to work properly. We observed that 47% of attention weights are focused on the edge of 100 sample images of the FSNS dataset, whereas the fraction goes down to 20% (mainly due to borders in some of the videos) for the same number of license plate scenes. As shown in Figure 6.6 (bottom), both the masks (shown in red and blue) have unique coverage. Moreover, it is crucial to observe in Figure 6.6 (bottom) that the first attention mask (in red) moves from the first character to the last character in the correct reading order (top-to-bottom followed by left-to-right) for the multi-line license plates. The second mask (in blue) possibly explores the new lines, non-pattern text, and the background regions. Moreover, the blue mask is highly scattering at locations where it could not find the

[5] `http://indiatyping.com/index.php/download/top-50-hindi-unicode-fonts-free`

license plate patterns. We observe that the third and fourth attention masks for 4 (and more) headed attention models are more scattered as compared to 2 headed attention models. Though scattered, coverage of each mask is unique irrespective of the number of heads used in the models.

6.8.2 License plate videos

We now present the results of experiments on traffic scenes in Indian streets. The license plate dataset, we use to train and test the models, is complicated because i) we perform recognition on continuous video frames, ii) the presence of motion blur in the frames where we interpolate the annotations from neighboring frames.

Training Method/ Model	Char. Acc. with inception-v3	Seq. Acc. with inception-v3	Char. Acc. with inc.-resnet-v2	Seq. Acc. with inc.-resnet-v2
Baseline CNN-RNN	96.74%	86.12%	96.74%	86.12%
1 head attention	97.48 %	89.87%	97.94 %	91.28%
2 head attention	**97.62%**	**91.06%**	98.06 %	91.56%
4 head attention	88.40 %	69.43%	98.12%	92.05%
8 head attention	-	-	**98.43%**	**93.30%**

Table 6.2: Evaluation on license plate videos

Effect on increasing number of heads

Table 6.2 depicts the results of the experiments on license plate scenes. As shown, the baseline model achieves a character level accuracy of 96.74% and sequence accuracy of 86.12%. With the inception-v3 encoder, the *OCR-on-the-go* model with single-head attention achieves a gain in sequence level accuracy of 3.75%. The accuracy gains further to 4.94% with the two-headed attention model. Splitting the features further for four-headed attention decreases the performance with the inception-v3 encoder. This probably happens because it becomes difficult to learn the proper attention masks with a smaller number of features (288/4 = 72). It also indicates that improvement gains for the two-headed attention model are not due to an increase in the number of parameters of the model. The *residual* networks have successfully improved the performance in image classification tasks in literature (He *et al.*, 2016). We resolve the problem of a decrease in accuracy with an increase in heads by using a better encoder with a denser layer to derive features *i.e.*, the mixed-6a layer (with feature depth = 1088) of the inception-resnet-v2 network. With this encoder, we observe a gain of 5.16% using single-headed attention with

respect to the baseline, and the gain further increases to 7.18% using eight-headed attention (such that the number of features assigned to each attention head is 136).

Effect of increasing the size of the inception encoder

As shown in Table 6.2, both character-level accuracy, as well as sequence accuracy, increase with the inception-resnet-v2 encoder as compared to the inception-v3 encoder for the same number of heads.

Training Method	Sequence Accuracy on FSNS dataset	Sequence Accuracy on IIIT-ILST
Baseline Model	Smith *et al.*, 2016	CNN-RNN
Baseline Results	72.46%	42.90% (Mathew *et al.*, 2017)
E2E model w/t 1 head attention	84.20% (Wojna *et al.*, 2017)	46.27%
E2E model w/t 2 head attention	84.59%	47.63%
E2E model w/t 4 head attention	84.86%	50.36%
E2E model w/t 8 head attention	**85.30%**	**51.09%**

Table 6.3: Evaluation on FSNS dataset and IIIT-ILST Devanagari dataset

6.8.3 French and Indian street signs

We further experiment with multi-headed attention models on the French Street Name Signs (FSNS) dataset. We use the mixed-6a layer of the inception-resnet-v2 network as an encoder for the reasons mentioned in the previous subsection.

The performance on the FSNS dataset is shown in the second column of Table 6.3. The models with multi-headed attention masks perform better than the model with single-headed attention. Moreover, the performance improves with an increase in the number of heads from two to eight. Finally, we note that our models outperform state-of-the-art results by Wojna *et al.*, 2017.

We also trained the *OCR-on-the-go* model on $750k$ Indian street sign images/frames described in Section 6.7. On the standard IIIT-ILST Devanagari dataset of $1.1k$ images, each containing a single word, we obtain the results shown in the last column of Table 6.3. As shown, our models outperform the state-of-the-art results for the dataset by Mathew *et al.*, 2017. Moreover, the models with multiple masks perform better than the models with a comparatively fewer number of masks.

Our initial experiments on the end-to-end test set of $17k$ frames from the multilingual dataset resulted in a system with 35% accuracy on the character-level and 1.3% accuracy on the sequence-level. This happens because the task of end-to-end recognition in Indian street sign scenes is extremely challenging due to the i) presence

of hand-written characters, ii) multiple scripts (Devanagari and Latin), iii) multiple languages (Hindi, Marathi, and English), and iv) long sequence lengths with a max of 180 as compared to that of 35 in the FSNS dataset. We further improve these results in Chapter 7 by training attention-based models on more extensive sequential multi-lingual data. Since the Indian languages contain a significant fraction of out-of-vocabulary words, we prefer to work on character-level predictions. In the future, we would like to use word embeddings for frequent words in the languages. It is not possible to predict the out-of-vocabulary words even by making use of fastText embeddings, which we employed in the previous chapter since such embeddings allow for the mappings from words to vectors. In contrast, the mappings from vectors to words is not feasible for out-of-vocabulary terms. As discussed in Section 6.1 and Section 6.7, the accuracies of existing photo OCR systems for Indian contexts were not sufficient for designing an interactive system for OCR corrections in the field. We improve such models in this chapter. Specifically, after achieving the results presented in Table 6.2 for the street videos, we now introduce an interactive tool for text correction in the next section.

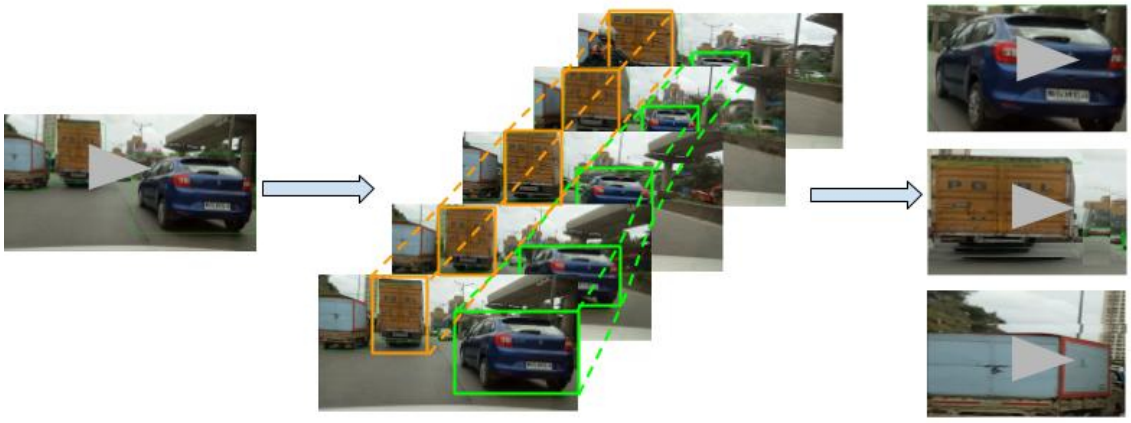


Figure 6.7: Breakdown of a video using our framework in both the spatial and temporal domain

6.9 StreetOCRCorrect

In this section, we introduce StreetOCRCorrect: a framework for interactively correcting OCR output in license plates with a human-in-the-loop. Such a framework can help collect a large amount of data for further improving the accuracy of deep learning models.

Robust OCR systems often fail in chaotic Indian scenes due to the presence of extreme variations in spatial and temporal domains like high vehicle density, multiple frames, unpredictable traffic conditions, and occlusions. We present a modular framework to handle such variations. The first module of the framework breaks down the video in a Spatio-temporal domain. We achieve the spatial breakdown by using the *You Only Look Once* (YOLO) object detector proposed by Redmon *et al.*, 2016. The framework further breaks down the video in the temporal domain using the MedianFlow video tracker (Varfolomieiev and Lysenko, 2016) and creates multiple clips. Each clip contains a single vehicle within a fixed window[6], as illustrated in Figure 6.7. Partial-occlusions (and inclusion of other vehicles in the clip) are handled by blurring the surrounding of the box that contains the vehicle under consideration, as shown in Figures 6.8and 6.9. The first module removes unnecessary complexities related to handling multiple vehicles and hence enables the user to focus on correcting the license plate text for a single-vehicle, as shown Figure 6.8.

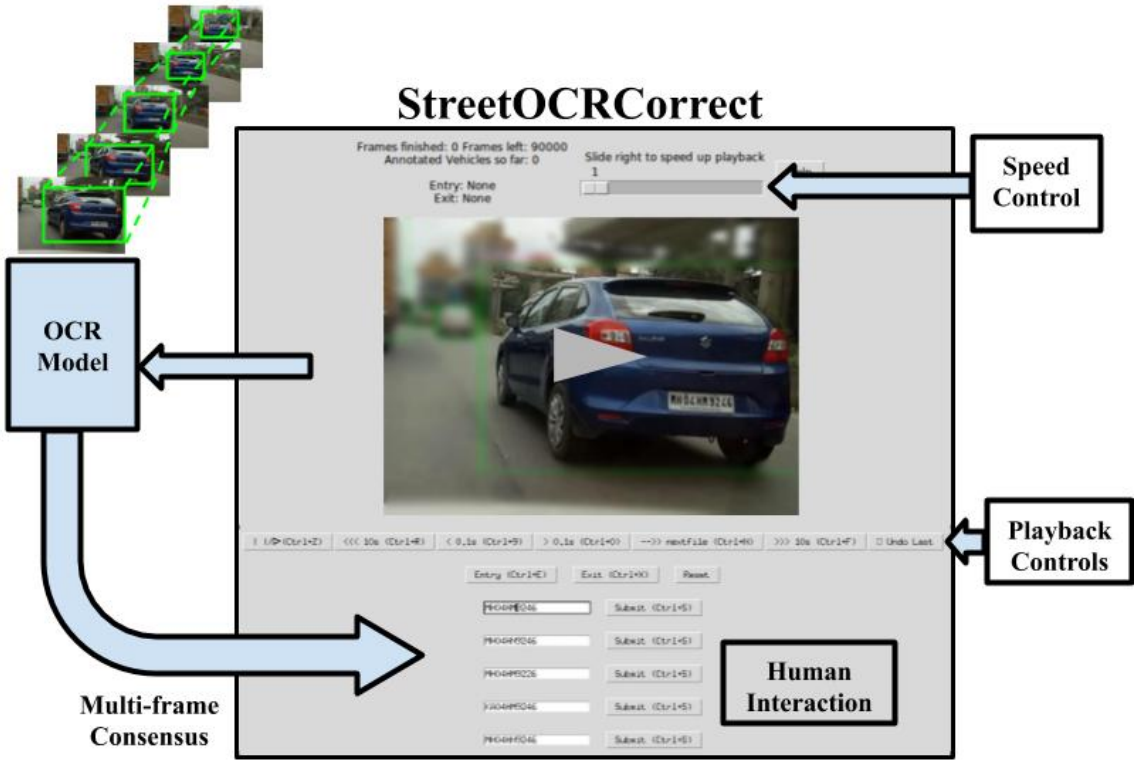


Figure 6.8: Components of our framework[7].

[6]We consider a fixed window to introduce stability in the video.

[7]Source code: `https://github.com/rohitsaluja22/StreetOCRCorrect`.

The second module of the framework enables the user to verify/correct the predictions of the extracted clips with minimal effort via interactive suggestions. We obtain these suggestions by using a multi-frame consensus on the output from an OCR model on the clips generated by the first module. We can further reduce the required human intervention by selectively presenting only clips with low confidence *i.e.* where we observe low consensus scores across the frames from the OCR system. As shown in Figure 6.8, the framework contains a speed control panel, a playback control panel, and a human interaction panel for improving the user experience. Here are the key features of our implementation:

1. When the user opens the application and loads the folder of clipped videos obtained from the first module, the video of an individual vehicle becomes visible to the user along with the suggestions from the OCR model in the "Human Interaction" area.

2. If the license plate is visible and readable throughout the video, the user can directly verify or correct one of the suggestions and submit the result.

3. If the license plate is not visible or readable at the start or the end of the video, the user selects the good frames by using entry and exit buttons, and then verifies or corrects one of the suggestions and submits the result.

4. The user then clicks on the "next video" button to upload the video of the next vehicle and repeat the above steps until completing the annotation for all the vehicles.

As a part of the framework, the third module stores the verified/corrected text with other metadata, such as timestamp and GPS location (if available). We also share the links of 22 youtube videos with chaotic Indian traffic scenes, along with the code.

GT: KA01MJ3793	GT: MH03CW4026	GT: MP04GA8692	GT: MH47A4487
KA01MJ3793 - 47	MH03CW4026 - 16	MP04GA8692 - 6	MH47A4487 - 66
KA01AJ3793 - 6	MH03CV4026 - 4	MP04GH692 - 3	MH47A44481 - 5
KA01NJ3793 - 1	MH03GVW4028 - 2	MP04GA6527 - 2	MH47A44482 - 4

Figure 6.9: Sample inputs, extracted from chaotic scenes, given to our framework

6.9.1 Video Results

The demo video on the GitHub link[8] shows the performance of our backend model on various challenging scenarios. The model performs end-to-end license plate recognition on a vehicle image. Therefore, before presenting the vehicle image to the model, we extend the vehicle box obtained from *YOLO* on its left and the right and blur the extended regions to avoid the recognition of other license plates. We include the additional area to maintain the fixed size as well as aspect ratio in frames of the clipped videos. We detect vehicles on each frame of the videos, as explained in Section 6.9. We show this in Figure 6.9. In the figure, we also present the top three predictions, along with their count, for some of the continuous sample frames for different vehicles. As shown, the predictions of our model are correct for the majority of frames on these samples. We use this system to generate a (clean) license plate dataset, which we further use to improve the recognition accuracy of the license plate recognition model. Since the deep learning techniques require a large amount of data, we collected 100 hours of single lane traffic video data from 15 different sources. With 5 annotators working for a total of 15 hours each and 3 reviewers working for 8 hours each, we generate around 2.67 million high-quality frame-level labelled dataset. Thus we obtain a high-quality dataset for 100 hours of video in a total of less than 100 working hours. This dataset helps us in improving the sequence accuracy of the model from 41% to 81% in the complicated settings mentioned above (with 15 different sources). We also observe that it takes 4 hrs to manually annotate a sample 1 hour video, whereas it takes 55 minutes to annotate the same 1 hour video using the StreetOCRCorrect. We thus demonstrate the effectiveness of the multi-frame consensus used in the framework.

6.10 Conclusion

In this chapter, we have investigated methods for generating datasets for scene-text recognition. We also presented an end-to-end trainable framework for reading text from scenes. We illustrated its application in two scenarios: (i) recognizing license plates automatically in chaotic traffic conditions, a task for which we curated our dataset, and (ii) on existing publicly available FSNS and IIIT-ILST Devanagari datasets. We perform experiments for license plate recognition on a large number of video frames. A salient point of our framework is that our models, when trained only on a combination of noisy labelled data and clean synthetic data, set new

[8]https://github.com/rohitsaluja22/StreetOCRCorrect

benchmarks for the task. We also demonstrate the importance of using multi-head attention in deep models. We are the first to observe that multi-headed attention is more effective in reading scene-text than single-headed attention. We show that such a multi-head attention mechanism helps in improving the accuracy of OCR systems due to the unique coverage learned by each attention mask. We further designed StreetOCRCorrect, an interactive framework for large scale OCR corrections in Indian street videos. The framework leverages upon state-of-the-art detectors and trackers to ease the correction process. We further use a multi-frame consensus to detect errors and reduce the cognitive load significantly via suggestions. The framework further maintains a large scale database of high-quality text, which can be used to improve the OCR models further and in other downstream applications. Work presented here was published at ICDAR'19 in our papers (Saluja *et al.*, 2019a; Singh *et al.*, 2019). For the Indian street-sign videos with considerable sequence length, initial results in Section 6.8.3 need to be improved. We improve models for the task in the next chapter by using the additional controlled video datasets and taming the attention masks of the end-to-end models.

Chapter 7

Taming the Attention Masks

In this chapter, we demonstrate that the accuracy of scene-text recognition can be improved by guiding the attention masks based on the orientations and positions of the camera. We improve ther end-to-end model described in Section 6.6 on continuous video frames by taming the attention masks in synthetic videos and on novel controlled datasets that we record for capturing possible camera movements. We begin by motivating our work in Section 7.1. We base a video scene-text recognition model (referred to as CATALIST[1]) on partly supervised attention. Like a teacher holding a lens through which a student can learn to read on a board, CATALIST exploits supervision for attention masks at multiple levels (as shown in Figure 7.3). Some of the attention masks might be interpreted as covering different orientations that occur in frames during individual camera movements (through separate masks) while others might focus on the line, word or character level reading order. We train CATALIST using synthetic data generated using a non-trivial extension of Synth-Text Gupta *et al.* (2016). The extension allows for the generation of text videos using different camera movements while also preserving character-level information. We describe the CATALIST model which 'tames' the attention (heads) in Section 7.2.1. We demonstrate that providing direct supervision to attention masks at multiple levels, (*i.e.*, line, word, and character levels) yields improvement in the recognition accuracy. To train CATALIST and its attention masks, we also present a synthetic data generator ALCHEMIST[2] that enables the synthetic creation of large scene-text video datasets, along with mask information at character, word and line levels. We describe the procedure to generate synthetic videos in Section 7.2.2. We also present a new video-based real scene-text dataset, $\text{CATALIST}_{\text{d}}$ in Section 7.2.3. We

[1]CATALIST stands for **CA**mera **T**r**A**nsformations for multi-**LI**ngual **S**cene **T**ext recognition

[2]ALCHEMIST stands for synthetic video generation in order to tame **A**ttention for **L**anguage (line, word, character, *etc.*) and other camera-**CH**ang**E**s and co**M**binat**I**ons for **S**cene **T**ext.

Figure 7.1: Sample video frames from CATALIST_d

present the sample video frames of the dataset in Figure 7.1. We create these videos using 5 types of camera transformations - (i) *translation*, (ii) *roll*, (iii) *tilt*, (iv) *pan*, and (v) *zoom*. We provide the dataset and experimental details in Section 7.3. We summarize the results in Section 7.4 and conclude the work in Section 7.6.

7.1 Motivation

We motivate our work of training the scene-text spotting models on the real (as well as synthetic) videos captured via continuous camera movements. Various end-to-end scene-text spotters, such as the ones proposed by Bušta *et al.* (2018, 2017), train on synthetic as well as augmented real data to cover different capturing perspectives/orientations. The problem, however, is that during the training phase, such models do not exploit all the continuous perspectives/orientations captured by the camera movement (or scene movement). Thus the OCR output fluctuates when tested on all/random video frames. Also, to deploy such models on real-time videos, two scenarios may occur. Firstly, the multi-frame consensus is desirable to improve OCR accuracy or interactive systems. Secondly, since it is computationally expensive to process each frame for readability, it is not possible to verify the quality of the frame to be OCR-ed. In any of these scenarios, the recognition system needs to work reasonably well on continuous video frames.

We present the frame level accuracy of E2E-MLT proposed by Bušta *et al.* (2018) on an 8 second video clip with a frame size of 480×260 in the first plot of Figure 7.2 (with sample frames shown at the bottom). Since the model does not work for Hindi, we recognize the Hindi text using *OCR-on-the-go* model described in Section 6.6. As shown, the E2E-MLT model produces the most unstable text on a simple video (from the test dataset) with the average character accuracy of 83.1% and the standard deviation of 9.20. The reason for this is that E2E-MLT, which

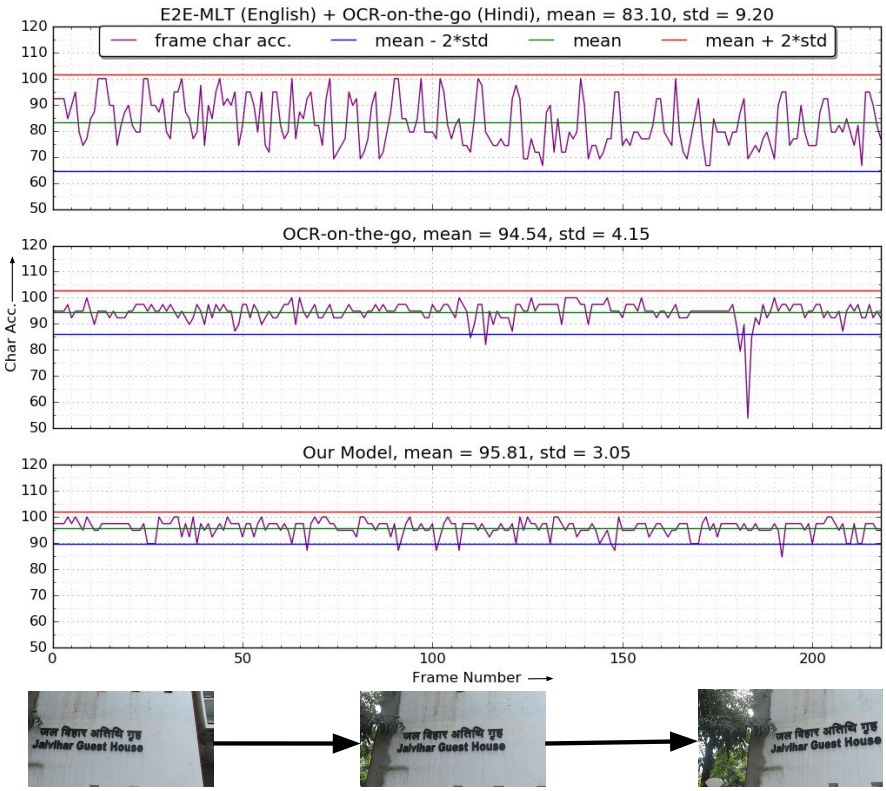


Figure 7.2: Frame wise accuracy of 3 text-spotters on a simple video exhibiting *pan*

does not train on continuous video frames, produces extra text-boxes on many of them during the detection phase. Thus extra noise characters or strings are observed during recognition. For instance, the correct text "Jalvihar Guest House" appears in 18 frames, the text "Jalvihar arG Guest House" appears in 10 frames, and the text "Jalvihar G Guest House" appears in 9 frames. The text "Jalvihar G arGu Guesth R H House" appears in one of the frame.

The instability in the video text, however, reduces when we use the *OCR-on-the-go* model (described in Section 6.6) to read these video frames. As shown in the second plot of Figure 7.2, we achieve the (higher) average character accuracy of 94.54% and (lower) standard deviation of 4.15. This model works on the principle of end-to-end recognition and soft detection via unsupervised attention. The instability further reduces, as shown in the third plot of Figure 7.2, when we train our CATALIST model on the continuous video datasets proposed in this chapter.

7.2 The CATALIST model and enabling datasets

We use end-to-end attention-based encoder-decoder model proposed by Wojna *et al.* (2017). For better inference of attention masks, and improved recognition, we use the

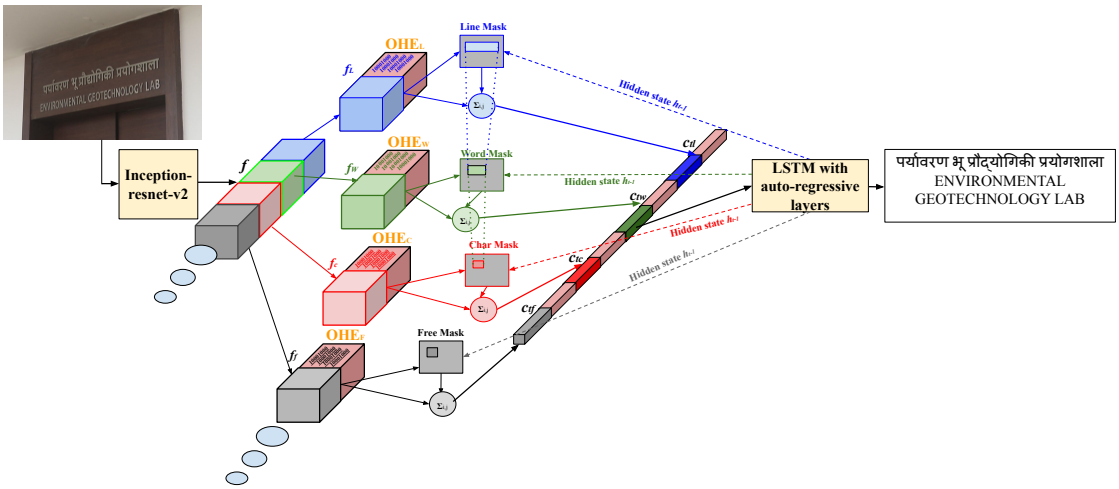


Figure 7.3: Our model (and its first four attention masks) that tames attention at multiple levels of granularity.

multi-head version of this model (discussed in Section 6.6). In Figure 7.3, we present the CATALIST model, that uses multi-task learning to update attention masks. Each mask is updated based on two loss functions. For end-to-end supervision, we use cross-entropy loss, and to guide the masks, we use dice loss[3] between the predicted masks and the segmented masks obtained using text-boxes from SynthText proposed by Gupta *et al.*, 2016. We also transform the synthetic images, along with text-boxes, to form the videos which we describe in the next section.

7.2.1 The CATALIST model

As shown in Figure 7.3, the powerful inception-based encoder proposed by Szegedy *et al.* (2016), which performs multiple convolutions in parallel, enhances the ability to read the text at multiple resolutions. We extract the features f from the input image using the inception-based encoder. Moreover, the multi-head attention mechanism in our model exploits: i) the splits of feature f into f_L, f_w, f_c, f_f[4], etc. (refer Figure 7.3), ii) one-hot-encoded (OHE) vectors (OHE_L, OHE_w, OHE_c, OHE_f, etc.[5]) for both x and y coordinates of each feature split, iii) hidden layer at the previous decoding step (h_{t-1}) of an LSTM (decoder). To learn the attention at multiple levels of granularity, we provide supervision to the first three masks in the form of the line, word, and character level segmented binary images. The

[3] dice_coe@`tensorlayer.readthedocs.io/en/latest/_modules/tensorlayer/cost.html`

[4] f_L represents the features used for producing line masks, f_w represents features used for word masks, f_c represents features used for character masks, and f_f represents features used for free attention masks

[5] for the corresponding features f_L, f_w, f_c, f_f, etc.

remaining masks are set free to assist/exploit end-to-end recognition/supervision. Thus we refer to the first three of them as line mask, word mask, and char mask in Figure 7.3, and the fourth mask (and the remaining masks) as free. We also hard-code the word mask to remain inside the line mask, and the character mask to remain inside the word mask. The context vectors (c_L, c_w, c_c, c_f, etc.), which are obtained after applying the attention mechanism, are fed into the LSTM to decode the characters in the input image. It is important to note that for each input frame, the features f and splits remain fixed, whereas the attention masks move in line with the decoded characters. Thus, we avoid using simultaneous supervision for all the character masks (or word masks or line masks) in a frame. Instead, we use a sequence of masks (in the form of segmented binary images) at each level for all the video frames. We accomplish this by keeping the word-level as well as the line-level segmented images constant and moving the character level segmented images while decoding the characters in each word. Once the decoding of all the characters in a word is complete, the word level segmented image moves to the next word in the line, and the character level image keeps moving as usual. Once the model has decoded all the characters in a line of text, the line (and word) level segmented image moves to the next line, and the character level segmented image continues to move within the word image.

7.2.2 The ALCHEMIST videos

We generate synthetic data for training the attention masks (as well as the complete model) using our data generator, which we refer to as ALCHEMIST. ALCHEMIST enables the synthetic creation of large scene-text video datasets. ALCHEMIST overlays synthetic text on videos under 12 different transformations described in the next section. By design, we preserve the information of the transformation performed, along with information of the character, word, and line positions (as shown in Figure 7.10). This information in the synthetic data provides for fairly detailed supervision on the attention masks in the CATALIST model. We build ALCHEMIST as an extension of an existing fast and scalable engine called SynthText proposed by Gupta *et al.*, 2016.

Methodology: According to pinhole camera model, a (2-d) point x (in homogeneous coordinate system) of image captured by a camera is given by equation 7.1.

$$x = K[R|t]X \tag{7.1}$$

Here K is the intrinsic camera matrix, R and t are rotation and translation matrices respectively, and X is a (3-d) point in *real world coordinates* in an homogeneous coordinate system.

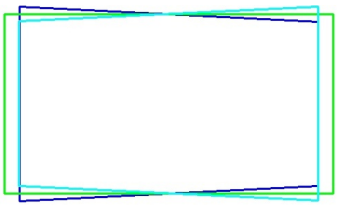


Figure 7.4: For videos with camera *pan*, we find Homography between the corners of a rectangle and 4 points equidistant from them (which form one of the blue trapeziums).

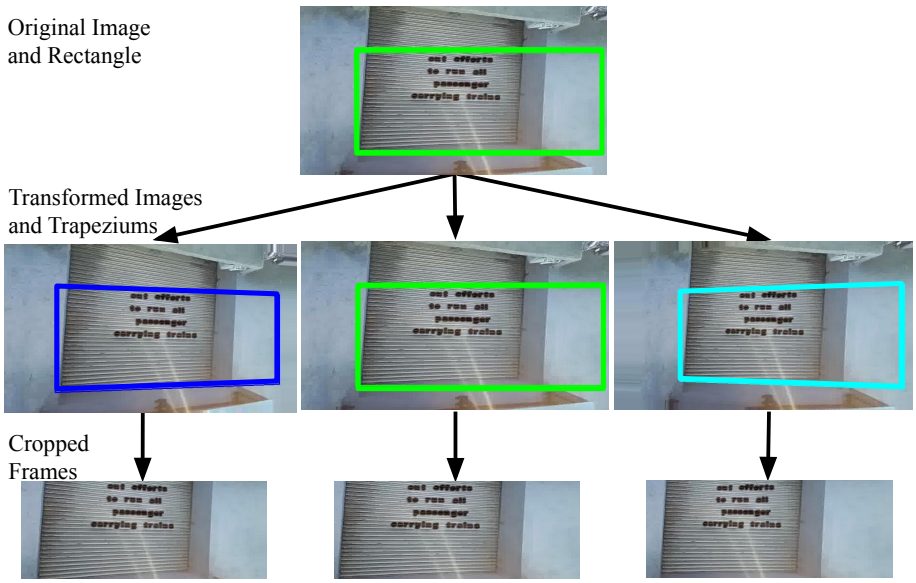


Figure 7.5: Generating video with camera *pan* (3 frames at the bottom for dark-blue, green and light-blue perspectives respectively) from an image (at the top)

For generating synthetic videos, we first select a fixed crop within the synthetic image (as denoted by the green rectangle in Figure 7.5). We then warp the corners of the crop by finding a planar homography matrix H (using algorithm given by Hartley and Zisserman, 2003) between the corner coordinates and four points equidistant from corners (direction depends on the kind of transformation as explained later). For Figure 7.4 (and Figure 7.5), we find the planar homography matrix H between corners of one of the blue trapezium and the green rectangle. Thus instead of (2-d) point x in the homogeneous coordinate system as explained earlier, we get a translated point x_{new} defined in equation 7.2:

$$x_{new} = HK[R|t]X \tag{7.2}$$

Here, H is the known homography. The above equation is simplified from the equation below:

$$x_{new} = KT[R|t]X = KTK^{-1}K[R|t]X \tag{7.3}$$

Here T is the unknown transformation matrix. We then warp the complete image using H and crop the rectangular region (refer green rectangle in Figure 7.5), to obtain the video frames. To find all the homography matrices for a video with camera *pan*, we consider the corners of the trapezium moving towards the rectangle corners. Once the homography matrix becomes the *identity matrix*, we move the corners of the trapezium away from the rectangle in the opposite direction to the initial flow (to form the mirrors of the initial trapeziums, e.g. light-blue trapezium in Figure 7.5).

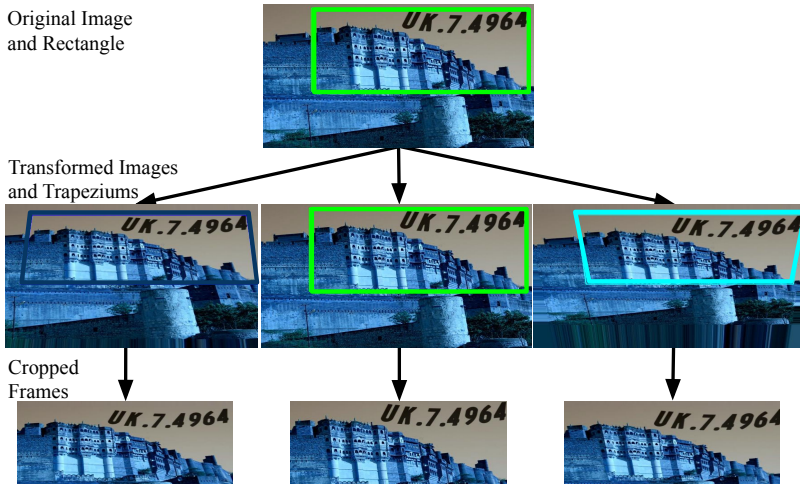


Figure 7.6: Generating video with camera *tilt* (frames at the bottom)

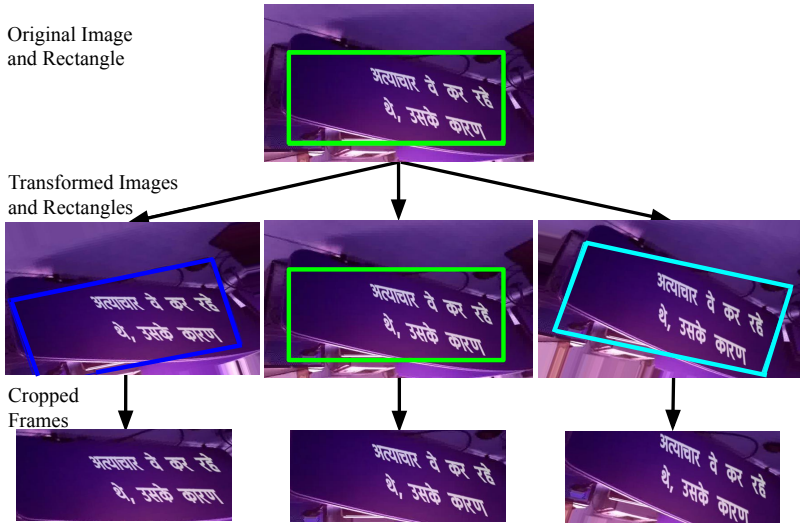


Figure 7.7: Generating video with camera *roll* (frames at the bottom)

The process for generating videos with camera *tilt* is similar to that of *pan*. The only difference is that the trapeziums in videos with camera *tilt* have vertical sides as parallel (as shown in Figure 7.6) whereas the trapeziums in videos with camera *pan* have horizontal sides as parallel. For the videos with camera *roll*, we utilize the homography matrices between the corners of the rectangles rotating around the text center and the base (horizontal) box, as shown in Figure 7.7.

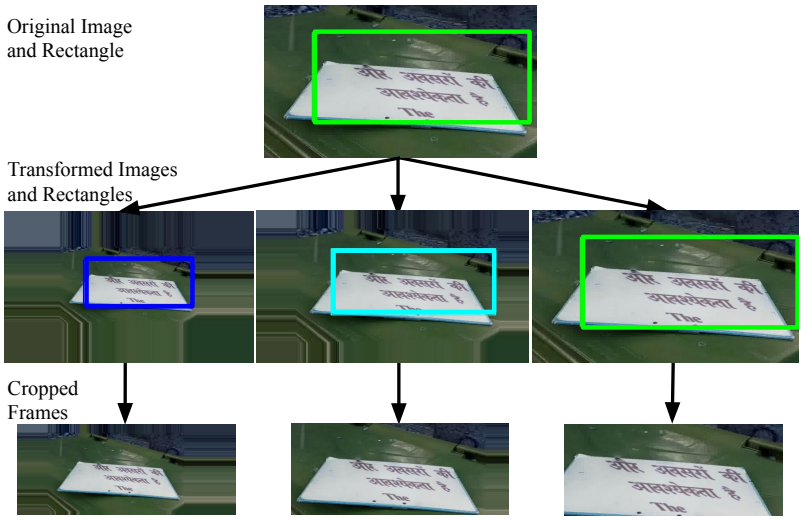


Figure 7.8: Generating video with camera zoom (frames at the bottom)

As shown in Figure 7.8, the corners of varying sized rectangles are the basis for synthesizing video with camera *zoom*.

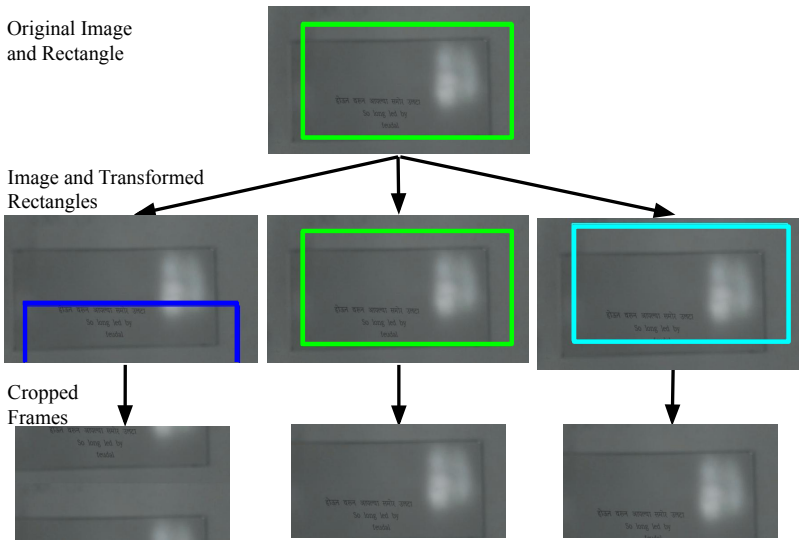


Figure 7.9: Generating video with camera *translation* (frames at the bottom)

For videos with camera *translation*, we use the regions a moving rectangle beginning from one text boundary to the other and generate the frames, as shown in Figure 7.9. We make sure that the complete text, with rare partial occlusion of boundary characters, lies within each frame of the videos.

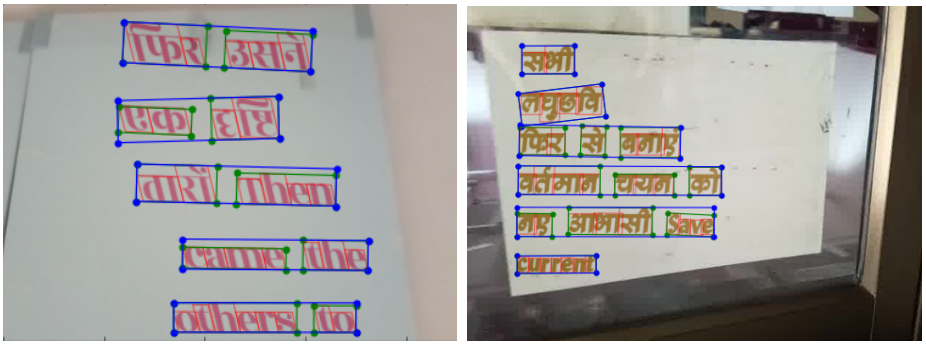


Figure 7.10: Sample frames from the synthetic videos with multi-level text-boxes

We also use the homography H to transform the multi-level text-boxes in the cropped image. Figure 7.10 depicts sample video frames with text-boxes at the line, word, and character[6] levels – shown in blue, green, and red, respectively.

7.2.3 The CATALIST$_d$ videos

We now present a new video-based scene-text dataset, which we refer to as CATALIST$_d$. Every video in CATALIST$_d$ contains scene-text, potentially in a combination of three different languages, namely, English, Hindi, and Marathi. For each such scene-text, we create 12 videos using 12 different types of camera *transformations*, broadly categorized into 5 groups:- (i) four types of *translation*, that could be left, right, up and down, (ii) two types of *roll*, including clockwise and anti-clockwise, (iii) two types of *tilt* which could be up-down or down-up motion, (iv) two types of *pan*, that is left-right and right-left , and (v) two types of *zoom* which could be in or out. We use a camera with a tripod stand to record all these videos to have a uniform control.

S.No.	Transformation Type	Number of Videos
1.	Translation	678
2.	Roll	320
3.	Tilt	360
4.	Pan	393
5.	Zoom	362

Table 7.1: Distribution of videos in the CATALIST$_d$ dataset

[6]For Devanagari (the script used for Hindi and Marathi), we carefully consider the boxes at the level of joint-glyphs instead of characters since rendering characters individually (to obtain character level text-boxes) hamper glyph substitution rules that form the joint glyphs in Devanagari.

We summarize the distribution of different types of videos in Table 7.1. It is important to note that there are four types of translations, whereas there are only two types for all other transformations. We capture all these videos at 25 fps with a resolution of 1920×1080.

7.3 Experiments

We synthesize around 12000 videos using ALCHEMIST data generator, which we use only for training the models. We use 50 Unicode fonts[7] and 18 license plate fonts[8] to render text in these videos. Here the duration and frame-rate for each video are 5 seconds and 25 fps, respectively. Moreover, we record a total of around $2k$ real videos (uniformly divided across 12 camera transformations) using a camera mounted over tripod stand for $\text{CATALIST}_{\text{d}}$ dataset. The setup allows smooth camera movements for *roll*, *tilt*, *pan* and *zoom*. We record the horizontal translation videos with the camera and tripod moving on a skateboard. Other translation videos, which exhibit top to bottom and reverse movements, have jitter because our tripod does not allow for smooth translation while recording such videos. We use a train:test split of 75:25, and carefully avoid letting any testing labels (as well as redundancy of the scenes) be present in the training data. We additionally record around $1k$ videos using handheld mobile phones and use them for training the models. Finally, we also make use of the 640 videos from Section 6.7 (that were used to train the *OCR-on-the-go* model). We refer to the complete training dataset described above as $\text{CATALIST}_{\text{ALL}}$ in the next sections.

We further add the ICDAR'15 English video dataset of 25 training videos (13,450 frames) and 24 testing videos (14,374 frames) by Karatzas *et al.* (2015) to the datasets. For each frame in the ICDAR'15 dataset, we first cluster the text-boxes into paragraphs and then sort the paragraph text-boxes from top-left to bottom-right. A sample video frame with the reading order mentioned above and the text-boxes sorted using our algorithm are shown in Figure 7.11. We visually verify that the reading order remains consistent throughout their appearance and disappearance in the videos. The reading order, changes when a new piece of text appears in the video or an old piece of text disappears from the video.

Although we record the controlled videos with a high resolution of 1920×1080, we work with the frame size of 480×260 for all videos owing to the more limited size

[7] `http://indiatyping.com/index.php/download/top-50-hindi-unicode-fonts-free`
[8] `https://fontspace.com/category/license%20plate`

Figure 7.11: A sample video frame from ICDAR'15 competition with text-boxes sorted using our algorithm

of the videos captured on mobile devices, as well as for to reduce training time on a large number of video frames. To take care of resolution as well as to remove the frames without text, we extract the 480×260 sized clips containing the mutually exclusive text regions in the videos from the ICDAR'15 dataset. Features of size $14 \times 28 \times 1088$ are extracted from the mixed-6a layer of inception-resnet-v2 (Szegedy *et al.*, 2016). The maximum sequence length of the labels is 180, so we unroll the LSTM decoder for 180 steps. We train all the models for 15 epochs.

7.4 Results

We now present the results of the CATALIST model on the different datasets described in the previous section. It is important to note that we use a single CATALIST model to jointly train on all the datasets ($\text{CATALIST}_{\text{ALL}}$) at once.

[9] 640 real videos + $700k$ synthetic images

[10] $3.7k$ real videos + $12k$ synthetic videos

S. No.	Training Model	Training Data	Test Data	Char. Acc.	Seq. Acc.
1.	OCR-on-the-go (8 free masks)	OCR-on-the-go[9]	OCR-on-the-go 200 test videos	35.00*	1.30*
2.	CATALIST model (8 free masks)	$\text{CATALIST}_{\text{ALL}}$[10]		65.50	7.76
3.	CATALIST model (3 superv., 5 free masks)	$\text{CATALIST}_{\text{ALL}}$		**68.67**	**7.91**
4.	CATALIST model (8 free masks)	$\text{CATALIST}_{\text{ALL}}$	491 $\text{CATALIST}_{\text{d}}$ videos	**73.97**	6.50
5.	CATALIST model (3 superv., 5 free masks)			73.60	**7.96**
6.	CATALIST model (8 free masks)	$\text{CATALIST}_{\text{ALL}}$	24 ICDAR'15 Competition videos by Karatzas *et al.* (2015)	34.37	**1.70**
7.	CATALIST model (3 superv., 5 free masks)			**35.48**	0.72

Table 7.2: Test Accuracy on different datasets. *results in Section 6.8.3

Results on the OCR-on-the-go dataset

In the first three rows of Table 7.2, we show the results on the test data used in Section 6.7 for the *OCR-on-the-go* model. The first row shows the results of this work. As shown in row 2, there is a dramatic improvement in character accuracy by 30.50% (from 35.0% to 65.5%) as well as sequence accuracy by 6.46% (1.30% to 7.76%), due to proposed CATALIST model as well as the ALCHEMIST and $\text{CATALIST}_{\text{d}}$ datasets we have have created. Adding the multi-level mask supervision to the CATALIST model further improves the accuracies by 3.17% (from 65.50% to 68.67%) and 0.15% (from 7.76% to 7.91%).

Results on the $CATALIST_d$ dataset

As shown in the fourth and fifth row of Table 7.2, the gain of 1.46% (6.50 to 7.96) is observed in the sequence accuracy of the CATALIST model, when we use the mask supervision. We, however, observe a slight gain of 0.37% in character level accuracy when all the masks are set free (*i.e.*, trained without any direct supervision).

Results on the ICDAR'15 competition dataset

We observe a gain of 1.11% (from 34.37% to 35.48%) in character-level accuracy on the ICDAR'15 competition dataset due to mask supervision. The end-to-end

sequence accuracy for this dataset is, however, as low as 1.70% for the model with all free masks, and further lower (by 0.98%) for the model with the first 3 masks trained using direct semantic supervision. We observe that the reason for the lower sequence accuracy for this dataset is the complex reading order in the frames.

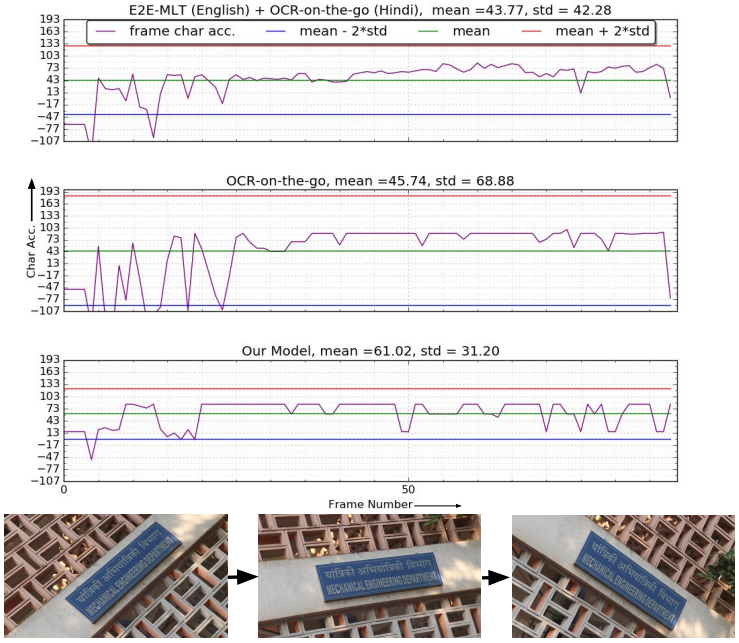


Figure 7.12: Frame-wise accuracy of 3 text-spotters on a video exhibiting *roll*

7.5 Frame-wise accuracies for all transformations

In Figure 7.2, we presented the frame-level accuracy of E2E-MLT (with the Hindi text recognized using *OCR-on-the-go* model) by Bušta *et al.* (2018), *OCR-on-the-go* model, and the present work on an 8 second video exhibiting *pan*. In this section, we present the frame-level accuracy of the above mentioned text-spotters for the other transformations: *roll*, *zoom*, *tilt*, and *translation*. The accuracy plots for a video with 88 frames (at 25 fps) exhibiting *roll* (clockwise) is shown in Figure 7.12. We use the formulae in Equation 7.4 for calculating the character accuracy taking noise characters into consideration.

$$Accuracy = 100 * \frac{length(GT) - edit\_distance(P, GT)}{length(GT)} \tag{7.4}$$

Here, GT denotes the ground truth sequence and P is the predicted sequence. For some of the frames (with large amounts of transformations), predicted sequence contains a lot of noise characters. As a result, the *edit_distance* between predicted sequence and ground truth sequence may go higher than the length of ground truth sequence. Thus we get negative accuracy for some of the frames in Figure 7.12. As shown, our model has the highest mean and lowest standard deviation for this

video as well. It demonstrates the importance of the CATALIST model trained with mask supervision on continuous video frames. Furthermore, it is essential to note that all the models perform poorly at the start of this video due to larger amounts of rotation as compared to the later parts of the video.

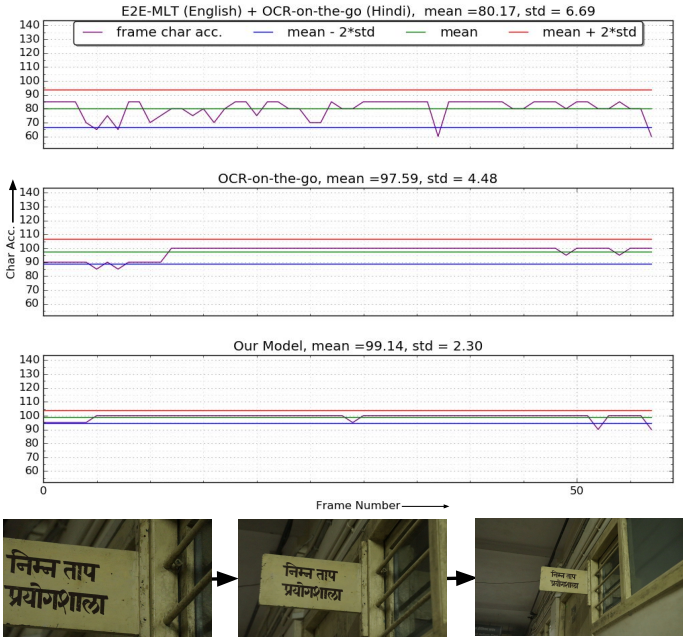


Figure 7.13: Frame-wise accuracy of 3 text-spotters on a video exhibiting *zoom*

In Figure 7.13, we present similar plots for a video with 58 frames exhibiting *zoom* (out). The signboard in the video only contains Hindi text. The E2E-MLT model, however, outputs some English characters due to script misidentification. Owing to this, the overall accuracy of the topmost plot (E2E-MLT + *OCR-on-the-go*) in Figure 7.13 is most unstable. Our model again achieves the highest mean and lowest standard deviation across all the video frames.

The plots for a video with 75 frames exhibiting *tilt* (up-down) is shown in Figure 7.14. As shown, contrary to other figures, the *OCR-on-the-go* model performs poorly on this video. The reason for this is that the model perhaps overfits to its license plates dataset. E2E-MLT generalizes well with respect to *OCR-on-the-go* model, however, our model has the highest average accuracy. In Figure 7.15, we present similar plots for a video with 121 frames exhibiting *translation* (upward). As discussed earlier in Section 7.3, the video clips recorded with the vertical camera movements in the setup possess jitter because the tripod does not allow for smooth translation while recording such videos. Our model, however, outputs the text with the highest accuracy and lowest standard deviation for the video we present in Figure 7.15.

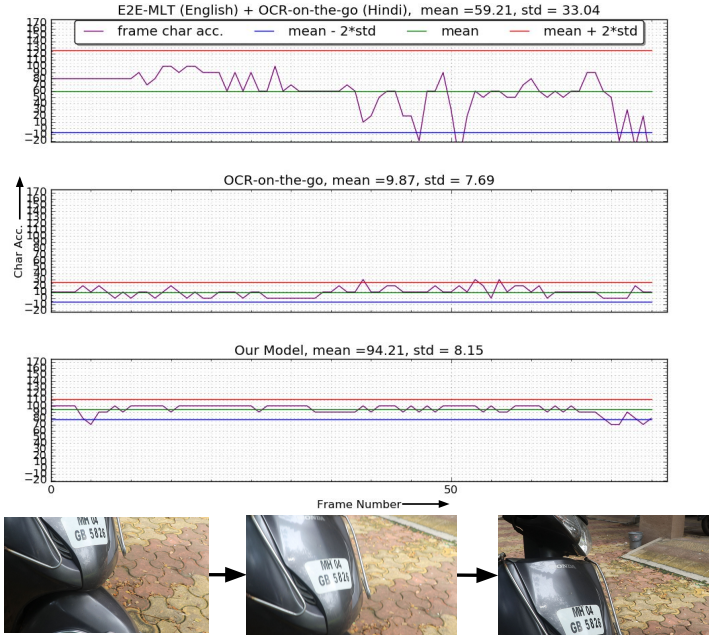


Figure 7.14: Frame-wise accuracy of 3 text-spotters on a video exhibiting *tilt*

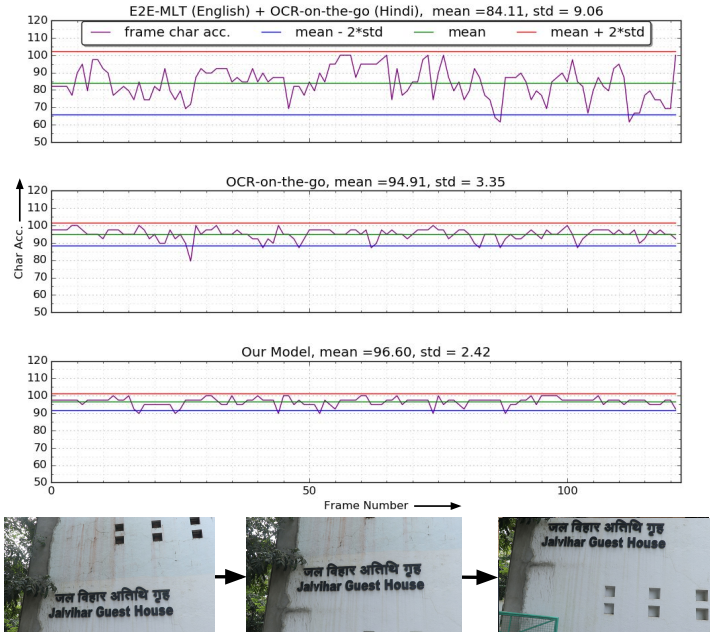


Figure 7.15: Frame wise accuracy of 3 text-spotters on a video exhibiting *translation*

7.6 Conclusion

In this chapter we presented CATALIST, a multi-task model for reading scene-text in videos and ALCHEMIST, a data generator that produces the videos from text images. Such videos mimic the behavior of videos captured via five different camera movements. We also presented the $\text{CATALIST}_{\text{d}}$ dataset of around two thousand real videos recorded with the camera movements mentioned above. By training the CATALIST model on the real and synthetic videos, we set new benchmarks for the task of reading multi-lingual scene-text in Hindi, Marathi, and English. The multi-

level mask supervision resulted in the improvement of either character or sequence (or both) accuracy on three different datasets with varying complexities.

Chapter 8

Conclusion and Future Work

Various Artificial Intelligence (AI) applications, including machine translation, autonomous driving, and text to speech, rely on Optical Character Recognition (OCR) in images or videos. Our investigations demonstrate that texts in Indic languages contain a large proportion of out-of-vocabulary words due to frequent fusion and agglutination. Using open-source and commercial OCR systems, we observe very high word error rates (WER) of approximately $20 - 50\%$ on four languages with varying complexities viz. Sanskrit, Malayalam, Kannada, and Hindi. We assert that any OCR system with an accuracy below 90% is not useful since it requires tremendous manual effort to correct its output. Therefore, an interactive system for spell checking OCR text is needed to automatically correct errors in low-accuracy OCR output. Conventional approaches to spell checking suggest corrections using proximity-based matches to a known vocabulary. For highly inflectional Indian languages, any off-the-shelf vocabulary is significantly incomplete. Moreover, in a spell checking system, the ranking of the suggested corrections is improved using language models. Owing to corpus resource scarcity, however, Indian languages lack reliable language models. Thus, learning the character confusions and error patterns of the OCR system can help correct the out-of-vocabulary words in OCR documents. We present OpenOCRCorrect: an interactive framework for assisting word-level corrections in Indic OCR documents, with the ability to identify, segment, and combine partially correct word forms. The model learns from corrections of the document and also auxiliary sources such as dictionaries and common OCR character confusions. The framework can also leverage multiple OCR systems on the same text for dynamic dictionary building. Over $12k$ document images in four different languages have been corrected using OpenOCRCorrect. We further present that a log-linear classifier, which exploits signals such as frequency values of the partial word forms

on various auxiliary sources and performs better than multiple-OCR consensus. We then adopt a Long Short-Term Memory (LSTM) based character level model with a *fixed delay* for jointly addressing the problems of error detection and correction in Indic OCR. Such a model jointly learns the language as well as OCR-specific confusions. For words that need not be corrected in the OCR output, the model simply abstains from suggesting any changes. Using this mechanism, we achieve error detection F-Scores above 92.4% and reduction in WER of at least 26.7% across the four Indian languages. We further improve the results by augmenting the input of LSTM model with two different encodings to capture the sub-word frequency values on a corpus. The first type of encoding makes direct use of sub-word unit frequency values derived from the training data. The formulation results in faster convergence and better accuracy values of the error correction models. The second type of encoding makes use of trainable sub-word embeddings. We also introduce a new procedure for training fastText on the sub-word units and further observe a large gain in F-Scores, as well as word-level accuracy values. The LSTM models perform well when the data is in order of $100k$ words. However, the performance drops when data is in order of $1000k$ words. In such situations, we show that a complex encoder and attention-based decoder model consisting of a separate LSTM for each is successful in learning the error correction mechanism similar to the basic LSTM model. Using such model our team "CLAM" (Character Level Attention Model) secured 2^{nd} position in the ICDAR, 2019 PostOCR Competition on 10 languages. Our model achieves the highest corrections of 44% in Finnish, which is significantly higher than the overall winner (8% in Finnish).

Further investigations into modern Indian street signs demonstrate that street signs are harder to read as compared to ancient texts. Scene-text in the real world is generally unstructured, appearing in a variety of languages, fonts, sizes, and orientations. Moreover, we use videos for training the deep models to i) collect and annotate a large number of video frames and ii) obtain robust training data. The movement of the capturing camera makes OCR an even more challenging task. We also observe that the existing scene OCR models are not reliable to develop the OCR correction systems and have WER of around 60% in Devanagari street sign images. We first leverage available text-spotters that are trained only on different text images, to generate a large amount of noisy dataset. We augment the data with interpolated boxes and annotations that make the training and testing robust for reading text in videos. Further use of synthetic data increases the coverage of the training process. We train two different models for reading street signs. The

baselines include black box detectors such as Convolution Neural Network (CNN) and humans, followed by the Recurrent Neural Network (RNN) based recognizer. Next, we build in the capability of training the model end-to-end on scenes containing license plates. We achieve this by incorporating inception-based CNN encoder, and location-sensitive attention mechanism in the LSTM decoder of a deep model. We also present the first scene-text recognition results using multi-headed attention models and demonstrate that each head has unique coverage over the scenes. We then present StreetOCRCorrect: a modular framework for OCR corrections in the chaotic Indian traffic videos. To ease the correction process, StreetOCRCorrect uses available detectors and trackers to break down the multi-vehicle videos into multiple clips, each containing a single vehicle from the video. We then incorporate multi-frame consensus (on the OCR output of each clip) for generating suggestions in the framework. The framework then selectively presents these extracted clips to the user to verify/correct the predictions with minimal human efforts via interactive suggestions. The high quality output obtained from such a framework can be used to continuously update a large database for surveillance as well as improving deep models on video data. We finally demonstrate that associating semantics with attention masks and then completely supervising those masks, leads to significant gains in scene OCR. The semantics include (i) camera *transformations* such as translation, pan, tilt, roll and zoom, and (ii) *granularity* of text in the scene such as character, word, and sentence. We extend an existing fast and scalable engine that overlays synthetic text on existing background images, to perform the text overlaying on videos under different transformations, naturally. The data synthesizer provides for fairly detailed supervision on the attention masks, including 5 camera *transformations* as well as 3 levels of *granularity* of the text in the scene. Subsequently, we present the architecture that uses such training data to tame the various attention masks to individually and collectively adhere to the semantics mentioned above.

8.1 Limitations and Future Work

Limitations to work on ancient Indic texts in the thesis include the use of

- **Word-level embeddings:** we use word-level embeddings, which include subword information, in Section 5.2 to improve the OCR correction models. The context learned by such embeddings is limited to a fixed number of words. The performance of OCR correction models can be enhanced if the context

boundaries can extend to lines or even paragraphs while preserving the sub-word information.

- **A large number of auxiliary sources:** We use six different sources in Section 4.2 for interactive OCR corrections. The reason for using such a large number of sources is the limited dataset in each auxiliary source and ambiguities that arise while using some of them for corrections. The number can reduce if we can make some of the non-ambiguous sources more powerful.

- **Constrained OCR systems to assist the interactive OCR corrections:** As shown in Table 1.2, the error rates for the OCR systems are different for different books. It happens because the type of variations (such as fonts, printing process, scanning process, etc.) in the datasets used to train the OCR systems are different from such variations used in many of the ancient documents.

Some of the interesting future directions for our document OCR work are as follows:

- **Analyzing the performances in dimensions like inflexions and OOVs:** Although we motivate in Figure 1.2 and Table 5.7 that the problem of Indic post-OCR involves a large fraction of out-of-vocabulary (OOV) terms, and present some of the results w.r.t. OOV terms in Tables 6.2 and 5.10. Nevertheless, it is important to mention that the OOV problem will eventually reduce as the OCR models as well as datasets in Indian languages improve. Therefore, it will become important to systematically analyze the performance in the dimensions like inflections and OOVs. Since the ground truth is available for the post-OCR datasets, it is possible to analyze the performances based on the degree of conjoining rules used in the ground truth words. For Sanskrit (or similar languages), sandhi splitters can be used to split each ground truth word and the performance can be analyzed by clustering the words based on the number of splits. Alternatively, in terms of OOV, unique OOV n-grams in an OCR and/or the corresponding ground truth word that are not present in the general language dictionary (or ground truth of training data) can be used to give the OOV score to each example.

- **Relevance of Post-OCR as the underlying models improve:** Since the printed document OCR is a simpler task than machine translation and scene captioning, many papers present over 98% accuracy on standard datasets. It shows that algorithms and losses are perfect for the OCR tasks now, given

the adequate amount of supervised dataset. So the question might be raised that will the post-OCR techniques remain relevant once the OCR systems are trained well on the large amount of supervised dataset obtained from ancient document images in multiple languages? Would the architectures or loss functions need revision in such scenarios?

It is important to note, that there are a few challenges when it comes to multi-lingual OCR in real ancient document images:

1. The variety of fonts and printing processes used in typewriters that printed such ancient documents are not available. Hence, we are not able to train the models on such fonts and diverse collections. The only solution is to collect a large amount of realistic data.
2. Even if we collect a large amount of data, the annotations' errors are another bottleneck in training such models. We observe that the human annotators tend to make character-level errors in long words, which is the unavoidable feature in languages rich in inflections. A proposed solution to this would be to have an unsupervised model that finds the character level errors in the labels given the image-label pairs.
3. To handling the degraded document images, the works on neural morphological processing (Mondal *et al.*, 2019) can be adopted.
4. Class balanced cross entropy loss, as advocated by Cui *et al.*, 2019 recently, can be used for loss functions to balance the scarcity of datasets in some of the languages.

- Automatic learning with no additional data: Techniques such as self-training, as recently used by Das and Jawahar, 2020, can be applied to automatically update the trained models to learn from pseudo labels. Additionally, using AI to compare the OCR text (or rendered image of the OCR text) with the original image, by adapting visual question answering models or siamese networks to the problem, and identifying OCR errors can help with proper subset selection of pseudo labels.

The winner of ICDAR (2019) post-OCR competition uses the sentence embeddings to assist error corrections in non-Indic OCR texts. Such embeddings might also help improve the Indic OCR correction systems. Moreover, with the scaling for error correction datasets, the interactive tasks may require a fewer number of auxiliary sources. The existing accuracies of document OCR systems depend on the type

of font used to train the models. The fonts are generally not available for ancient documents. As the extensive datasets are collected using our framework, we would like to use them to improve the Indic OCR systems as future work. The OCR correction process itself can also incrementally update the OCR systems on-the-fly with the images and corrected labels to reduce the correction load dynamically. We would like to involve the sentence level embeddings as well as image-based updates rather than just relying on text-based auxiliary sources for Indic OCR corrections as future work.

The work on reading street signs has following limitations:

- Low sequence accuracy on datasets with lengthy sentences: As shown in Section 6.7, the sequence accuracy of the *OCR-on-the-go* model on the FSNS dataset is above 85%. However, the sequence accuracy of the CATALIST is below 10% (as shown in Section 7.4) on $\text{CATALIST}_{\text{d}}$ dataset with more extended labels, although the character accuracy is above 70%. All this indicates that the overall settings are prone to errors for the lengthy sequences of text.

- The inability of models to process multiple-frames: The 2-d CNNs used in the *OCR-on-the-go* model and the CATALIST model limit them to train on individual video frames. Such a training process makes the model robust to different perspectives during test time, and subsequently, the model works equally well for any of the video frames. However, it does not allow for selectively reading high-quality regions from multiple frames. Using 3-d CNNs instead of 2-d CNNs can help overcome this limitation.

The sequence accuracy can benefit by i) training on more extensive datasets, and/ or ii) directly predicting the word-embeddings of frequent words instead of characters at the target side of end-to-end models. On the one hand, processing the multiple-frames using 3-d CNNs as the encoder might help resolve the issue of processing individual video frames. On the other hand, it would not work for the videos where the text frequently changes (like on the display board screens at the airports). Modifying the decoder to predict multiple sequential outputs (one for each frame) might help tackle the issue.

An interesting future direction for the photo-OCR work are as follows:

- **Exploiting multilingual alignment for post correction:** Translations and transliteration schemes can be utilized to post-correct the OCR output because the multilingual street signs are aligned multilingual corpora. Figure 7.15 illustrates that a balanced mix of translation and transliteration for

each word may help improve the system. However, we also observe two issues with this approach; i) in some of the samples, the translations and transliterations are wrong. It happens because certain examples include advertisements, which purposefully make mistakes to attract attention, and ii) some words are highly technical and are not available in popular bilingual dictionaries. Moreover, there is always a trade-off between word-level accuracy and character level accuracy while using dictionary-based approaches in a way that (incorrect) translations and transliterations may reduce character level accuracy while trying to improve word-level accuracy. So we avoided using such methods for post-processing in the thesis. However this space can be further explored in the future work.

As future work, we would also like to work on improving the photo OCR systems by i) prediction at multiple levels (word-level embeddings for frequent words, character-level for out-of-vocabulary words), and ii) additionally predicting the essential text features such as text-location, text-font, font-weight or text-attention, and style.

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., *et al.*, 2016, "Tensorflow: A System for Large-Scale Machine Learning," in *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pp. 265–283.

Abdulkader, A., and Casey, M. R., 2009, "Low Cost Correction of OCR Errors Using Learning in a Multi-Engine Environment," in *10th International Conference on Document Analysis and Recognition* (IEEE). pp. 576–580.

Adiga, D., Saluja, R., Agrawal, V., Ramakrishnan, G., Chaudhuri, P., Ramasubramanian, K., and Kulkarni, M., 2018, "Improving the learnability of classifiers for Sanskrit OCR corrections," in *The 17th World Sanskrit Conference, Vancouver, Canada. IASS*, pp. 143–161.

Afli, H., Barrault, L., and Schwenk, H., 2016, "OCR Error Correction Using Statistical Machine Translation.." *Int. J. Comput. Linguistics Appl.* **7**, 175–191.

Alexandrescu, A., and Kirchhoff, K., 2006, "Factored neural language models," in *Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Short Papers*, pp. 1–4.

Artaud, C., Sidère, N., Doucet, A., Ogier, J.-M., and Yooz, V. P. D., 2018, "Find it! fraud detection contest report," in *2018 24th International Conference on Pattern Recognition (ICPR)* (IEEE). pp. 13–18.

Arya, D., Jawahar, C., Bhagvati, C., Patnaik, T., Chaudhuri, B., Lehal, G., Chaudhury, S., and Ramakrishna, A., 2011, "Experiences of integration and performance testing of multilingual ocr for printed indian scripts," in *Proceedings of the 2011 joint workshop on multilingual OCR and analytics for noisy unstructured text data* (ACM). p. 9.

Avi-Itzhak, H. I., Diep, T. A., and Garland, H., 1995, "High accuracy optical character recognition using neural networks with centroid dithering," *IEEE Transactions on pattern analysis and machine intelligence* **17**, 218–224.

Bartz, C., Yang, H., and Meinel, C., 2017, "Stn-ocr: A single neural network for text detection and text recognition," *arXiv preprint arXiv:1707.08831*

Bassil, Y., and Alwani, M., 2012, "OCR Context-sensitive Error Correction Based on Google Web 1T 5-gram Data Set," *arXiv preprint arXiv:1204.0188*

Bellman, R., 1966, "Dynamic Programming," *Science* **153**, 34–37.

Bennett, K. P., and Demiriz, A., 1999, "Semi-supervised support vector machines," in *Advances in Neural Information processing systems*, pp. 368–374.

Bhaskarabhatla, A. S., Madhvanath, S., Kumar, M., Balasubramanian, A., and Jawahar, C., 2004, "Representation and Annotation of Online Handwritten Data," in *Ninth International Workshop on Frontiers in Handwriting Recognition* (IEEE). pp. 136–141.

Biswas, C., Mukherjee, P. S., Ghosh, K., Bhattacharya, U., and Parui, S. K., 2018, "A Hybrid Deep Architecture for Robust Recognition of Text Lines of Degraded Printed Documents," in *2018 24th International Conference on Pattern Recognition (ICPR)* (IEEE). pp. 3174–3179.

Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T., 2017, "Enriching word vectors with subword information," *Transactions of the Association for Computational Linguistics* **5**, 135–146.

Breuel, T. M., Ul-Hasan, A., Al-Azawi, M. A., and Shafait, F., 2013, "High-performance ocr for printed english and fraktur using lstm networks," in *2013 12th International Conference on Document Analysis and Recognition* (IEEE). pp. 683–687.

Bušta, M., Neumann, L., and Matas, J., 2017, "Deep textspotter: An end-to-end trainable scene text localization and recognition framework," *International Conference on Computer Vision*

Bušta, M., Patel, Y., and Matas, J., 2018, "E2E-MLT-an Unconstrained End-to-End Method for Multi-Language Scene Text," in *Asian Conference on Computer Vision* (Springer). pp. 127–143.

Carlson, A., and Fette, I., 2007, "Memory-based Context-sensitive Spelling Correction at Web Scale," in *International Conference on Machine learning and applications (ICMLA)*, pp. 166–171.

Chaudhuri, B., and Pal, U., 1997, "An OCR system to read two Indian language scripts: Bangla and devnagari (hindi)," in *Proceedings of the fourth international conference on document analysis and recognition*, Vol. 2 (IEEE). pp. 1011–1015.

Cheriet, M., Kharma, N., Liu, C.-L., and Suen, C., 2007, *Character recognition systems: a guide for students and practitioners* (John Wiley & Sons).

Chiron, G., Doucet, A., Coustaty, M., and Moreux, J.-P., 2017a, "ICDAR2017 Competition on Post-OCR Text Correction," in *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, Vol. 1 (IEEE). pp. 1423–1428.

Chiron, G., Doucet, A., Coustaty, M., Visani, M., and Moreux, J.-P., 2017b, "Impact of OCR Errors on the Use of Digital Libraries: Towards a Better Access to Information," in *2017 ACM/IEEE Joint Conference on Digital Libraries (JCDL)* (IEEE). pp. 1–4.

Choudhury, M., Thomas, M., Mukherjee, A., Basu, A., and Ganguly, N., 2007, "How Difficult is it to Develop a Perfect Spell-checker? A Cross-linguistic Analysis through Complex Network Approach," *arXiv preprint physics/0703198*

Cui, Y., Jia, M., Lin, T.-Y., Song, Y., and Belongie, S., 2019, "Class-Balanced Loss Based on Effective Number of Samples," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 9268–9277.

Damerau, F. J., 1964, "A technique for computer detection and correction of spelling errors," *Communications of the ACM* **7**, 171–176.

Das, D., and Jawahar, C., 2020, "Adapting OCR with Limited Supervision," in *International Workshop on Document Analysis Systems* (Springer). pp. 30–44.

Das, D., Philip, J., Mathew, M., and Jawahar, C., 2019, "A Cost Efficient Approach to Correct OCR Errors in Large Document Collections," in *2019 International Conference on Document Analysis and Recognition (ICDAR)* (IEEE). pp. 655–662.

Davis, B., Morse, B., Cohen, S., Price, B., and Tensmeyer, C., 2019, "Deep visual template-free form parsing," *arXiv preprint arXiv:1909.02576*

Devlin, J., Chang, M., Lee, K., and Toutanova, K., 2018, "BERT: pre-training of deep bidirectional transformers for language understanding," *CoRR*

Doucet, A., Coustaty, M., *et al.*, 2017, "Enhancing Table of Contents Extraction by System Aggregation," in *2017 14th IAPR international conference on document analysis and recognition (ICDAR)*, Vol. 1 (IEEE). pp. 242–247.

Doucet, A., Kazai, G., Colutto, S., and Mühlberger, G., 2013, "Icdar 2013 Competition on Book Structure Extraction," in *2013 12th International Conference on Document Analysis and Recognition* (IEEE). pp. 1438–1443.

Doucet, A., Kazai, G., and Meunier, J.-L., 2011, "Icdar 2011 Book Structure Extraction Competition," in *2011 International Conference on Document Analysis and Recognition* (IEEE). pp. 1501–1505.

Du, S., Ibrahim, M., Shehata, M., and Badawy, W., 2013, "Automatic license plate recognition (alpr): A state-of-the-art review," *IEEE Transactions on circuits and systems for video technology* **23**

Duda, R. O., Hart, P. E., *et al.*, 1973, *Pattern classification and scene analysis*, Vol. 3 (Wiley New York).

Dutta, S., Sankaran, N., Sankar, K. P., and Jawahar, C., 2012, "Robust recognition of degraded documents using character n-grams," in *2012 10th IAPR International Workshop on Document Analysis Systems* (IEEE). pp. 130–134.

Dīkṣita, B., Dīkṣita, V., and Sarasvatī, J., 2006, *Vaiyākaraṇasiddhāntakaumudī with the commentary Bālamanoramā and Tattvabodhinī* (Motilal Banarasidas).

Esposito, F., Malerba, D., Semeraro, G., Annese, E., and Scafuro, G., 1990, "An experimental page layout recognition system for office document automatic classification: An integrated approach for inductive generalization," in *[1990] Proceedings. 10th International Conference on Pattern Recognition*, Vol. 1 (IEEE). pp. 557–562.

Evershed, J., and Fitch, K., 2014, "Correcting Noisy OCR: Context Beats Confusion," in *Proceedings of the First International Conference on Digital Access to Textual Cultural Heritage*, pp. 45–51.

Frinken, V., Fischer, A., and Bunke, H., 2010a, "A novel word spotting algorithm using bidirectional long short-term memory neural networks," in *IAPR Workshop on Artificial Neural Networks in Pattern Recognition* (Springer). pp. 185–196.

Frinken, V., Fischer, A., Bunke, H., and Manmatha, R., 2010b, "Adapting blstm neural network based keyword spotting trained on modern data to historical documents," in *2010 12th International Conference on Frontiers in Handwriting Recognition* (IEEE). pp. 352–357.

Glauberman, M., 1956, "Character recognition for business machines," *Electronics* **29**, 132–136.

Golding, A. R., and Schabes, Y., 1996, "Combining Trigram-based and Feature-based Methods for Context-sensitive Spelling Correction," in *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, pp. 71–78.

Goodrich, G. L., Bennett, R., De l'Aune, W. R., Lauer, H., and Mowinski, L., 1979, "Kurzweil reading machine: A partial evaluation of its optical character recognition error rate," *J. Visual Impairment & Blindness* **73**, 389.

Google, 2020, "Convert pdf and photo files to text," *https://support.google.com/drive/answer/176692?hl=en*

Google, 2020, "Google's Optical Character Recognition (ocr) Software Works for 248+ Languages," `https://opensource.com/life/15/9/open-source-extract-text-images`. Last accessed on 21 January'20.

Govindan, V., and Shivaprasad, A., 1990, "Character recognition—A review," *Pattern recognition* **23**, 671–683.

Govindaraju, V., and Setlur, S., 2009, *Guide to OCR for Indic Scripts* (Springer).

Graves, A., Fernández, S., Gomez, F., and Schmidhuber, J., 2006, "Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks," in *Proceedings of the 23rd international conference on Machine learning*, pp. 369–376.

GRETIL, 2001, "Göttingen Register of Electronic Texts in Indian Languages," *http://gretil.sub.uni-goettingen.de/*

Gupta, A., Vedaldi, A., and Zisserman, A., 2016, "Synthetic Data for Text Localisation in Natural Images," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2315–2324.

Gustav, T., 1938 Apr. 26, "Reading machine," uS Patent 2,115,563.

Handel, P. W., 1933 Jun. 27, "Statistical machine," uS Patent 1,915,993.

Hanov, S., 2013, "Fast and easy Levenshtein distance using a Trie," *http://stevehanov.ca/blog/index.php?id=114*

Hartley, R., and Zisserman, A., 2003, *Multiple view geometry in computer vision* (Cambridge university press).

He, K., Zhang, X., Ren, S., and Sun, J., 2016, "Deep Residual Learning for Image Recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.

Hedderich, M. A., and Klakow, D., 2018, "Training a neural network in a low-resource setting on automatically annotated noisy data," *ACL 2018*

Hirschberg, D. S., 1977, "Algorithms for the longest common subsequence problem," *Journal of the ACM (JACM)* **24**, 664–675.

Horwitz, L., and Shelton, G., 1961, "Pattern recognition using autocorrelation," *Proceedings of the IRE* **49**, 175–185.

Hu, J., Kashi, R. S., Lopresti, D. P., and Wilfong, G., 2000, "Table structure recognition and its evaluation," in *Document Recognition and Retrieval VIII*, Vol. 4307 (International Society for Optics and Photonics). pp. 44–55.

ICDAR, 2017, "Competition on Post-OCR Text Correction," `https://sites.google.com/view/icdar2017-postcorrectionocr/`. Last accessed on 6 February'20,

ICDAR, 2019, "Competition on Post-OCR Text Correction," `https://sites.google.com/view/icdar2019-postcorrectionocr`. Last accessed on March 7

ind.senz, 2020, "IndicOCR," *http://www.indsenz.com/*

Jain, R., Frinken, V., Jawahar, C., and Manmatha, R., 2011, "Blstm Neural Network based Word Retrieval for Hindi Documents," in *2011 International Conference on Document Analysis and Recognition* (IEEE). pp. 83–87.

Jain, V., Sasindran, Z., Rajagopal, A., Biswas, S., Bharadwaj, H. S., and Ramakrishnan, K., 2016, "Deep automatic license plate recognition system," in *Proceedings of the Tenth Indian Conference on Computer Vision, Graphics and Image Processing*, pp. 1–8.

Jenkins, F., Kanai, J., and Nartker, T., 1993, "Using ideal images to establish a baseline of ocr performance," *Information Science Research Institute Annual Research Report*, 47–54.

Jiao, J., Ye, Q., and Huang, Q., 2009, "A configurable method for multi-style license plate recognition," *Pattern Recognition* **42**

Kameshiro, T., Hirano, T., Okada, Y., and Yoda, F., 1999, "A document image retrieval method tolerating recognition and segmentation errors of ocr using shape-feature and multiple candidates," in *Proceedings of the Fifth International Conference on Document Analysis and Recognition. ICDAR'99 (Cat. No. PR00318)* (IEEE). pp. 681–684.

Karatzas, D., Gomez-Bigorda, L., Nicolaou, A., Ghosh, S., Bagdanov, A., Iwamura, M., Matas, J., Neumann, L., Chandrasekhar, V. R., Lu, S., *et al.*, 2015, "Icdar 2015 competition on Robust Reading," in *2015 13th International Conference on Document Analysis and Recognition (ICDAR)* (IEEE). pp. 1156–1160.

Karatzas, D., Shafait, F., Uchida, S., Iwamura, M., i Bigorda, L. G., Mestre, S. R., Mas, J., Mota, D. F., Almazan, J. A., and De Las Heras, L. P., 2013, "ICDAR 2013 Robust Reading Competition," in *2013 12th International Conference on Document Analysis and Recognition* (IEEE). pp. 1484–1493.

Kennedy, L. S., and Naaman, M., 2008, "Generating diverse and representative image search results for landmarks," in *Proceedings of the 17th international conference on World Wide Web*, pp. 297–306.

Kieninger, T. G., 1998, "Table structure recognition based on robust block segmentation," in *Document Recognition V*, Vol. 3305 (International Society for Optics and Photonics). pp. 22–32.

Kissos, I., and Dershowitz, N., 2016, "OCR Error Correction Using Character Correction and Feature-based Word Classification," in *12th IAPR Workshop on Document Analysis Systems (DAS)*, pp. 198–203.

Klein, G., Kim, Y., Deng, Y., Senellart, J., and Rush, A., 2017, "OpenNMT: Open-Source Toolkit for Neural Machine Translation," in *Proceedings of ACL 2017, System Demonstrations* (Association for Computational Linguistics, Vancouver, Canada). pp. 67–72.

Krishna, A., Majumder, B. P., Bhat, R. S., and Goyal, P., 2018, "Upcycle your ocr: Reusing ocrs for post-ocr text correction in romanised sanskrit," *arXiv preprint arXiv:1809.02147*

Krishnan, P., Sankaran, N., Singh, A. K., and Jawahar, C., 2014, "Towards a Robust OCR System for Indic Scripts," in *2014 11th IAPR International Workshop on Document Analysis Systems* (IEEE). pp. 141–145.

Krizhevsky, A., Sutskever, I., and Hinton, G. E., 2012, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105.

Kukich, K., 1992, "Techniques for Automatically Correcting Words in Text," *ACM Computing Surveys (CSUR)* **24**, 377–439.

Kumar, A., and Jawahar, C., 2007, "Content-level Annotation of Large Collection of Printed Document Images," in *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, Vol. 2 (IEEE). pp. 799–803.

La Manna, S., Colia, A., and Sperduti, A., 1999, "Optical font recognition for multi-font ocr and document processing," in *Proceedings. Tenth International Workshop on Database and Expert Systems Applications. DEXA 99* (IEEE). pp. 549–553.

Lan, W., and Xu, W., 2018, "The Importance of Subword Embeddings in Sentence Pair Modeling," in *NAACL 2018*, pp. 1636–1646.

Lee, C.-Y., and Osindero, S., 2016, "Recursive recurrent nets with attention modeling for ocr in the wild," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2231–2239.

Lehal, G., Singh, C., and Lehal, R., 2001, "A Shape Based Post Processor for Gurmukhi OCR," in *International Conference on Document Analysis and Recognition (ICDAR)*, pp. 1105–1109.

Lenc, L., Martínek, J., and Král, P., 2019, "Tools for semi-automatic preparation of training data for ocr," in *IFIP International Conference on Artificial Intelligence Applications and Innovations* (Springer). pp. 351–361.

Levenshtein, V. I., 1966, "Binary codes capable of correcting deletions, insertions, and reversals," in *Soviet physics doklady*, Vol. 10, pp. 707–710.

Li, H., and Shen, C., 2016, "Reading car license plates using deep convolutional neural networks and lstms," *arXiv preprint arXiv:1601.05610*

Liao, M., Shi, B., Bai, X., Wang, X., and Liu, W., 2017, "Textboxes: A fast text detector with a single deep neural network.." in *AAAI*, pp. 4161–4167.

Lopresti, D., 2009, "Optical character recognition errors and their effects on natural language processing," *International Journal on Document Analysis and Recognition (IJDAR)* **12**, 141–151.

Lopresti, D., and Zhou, J., 1997, "Using consensus sequence voting to correct ocr errors," *Computer Vision and Image Understanding* **67**, 39–47.

Luong, M.-T., Pham, H., and Manning, C. D., 2015, "Effective approaches to attention-based neural machine translation," *arXiv preprint arXiv:1508.04025*

Ly, N. T., Nguyen, C. T., and Nakagawa, M., 2019, "An attention-based end-to-end model for multiple text lines recognition in Japanese Historical Documents," in *15th IAPR International Conference on Document Analysis and Recognition (ICDAR)* (IEEE). pp. 629–634.

Mathew, M., Jain, M., and Jawahar, C., 2017, "Benchmarking Scene Text Recognition in Devanagari, Telugu and Malayalam," in *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, Vol. 7 (IEEE). pp. 42–46.

Mathew, M., Singh, A. K., and Jawahar, C., 2016, "Multilingual ocr for indic scripts," in *Document Analysis Systems (DAS), 2016 12th IAPR Workshop on* (IEEE). pp. 186–191.

Mei, J., Islam, A., Wu, Y., Moh'd, A., and Milios, E. E., 2016, "Statistical Learning for OCR Text Correction," *arXiv preprint arXiv:1611.06950*

Melnikov, A., and Zagaynov, I., 2020, "Fast and Lightweight Text Line Detection on Historical Documents," in *International Workshop on Document Analysis Systems* (Springer). pp. 441–450.

Mikolov, T., Karafiát, M., Burget, L., Černockỳ, J., and Khudanpur, S., 2010, "Recurrent neural network based language model," in *Eleventh annual conference of the International Speech Communication Association (INTERSPEECH)*, pp. 1045–1048.

Minghui Liao, B. S., and Bai, X., 2018, "TextBoxes++: A single-shot oriented scene text detector," *CoRR* **abs/1801.02765**

Mondal, R., Chakraborty, D., and Chanda, B., 2019, "Learning 2D Morphological Network for Old Document Image Binarization," in *2019 International Conference on Document Analysis and Recognition (ICDAR)* (IEEE). pp. 65–70.

Mor, N., and Wolf, L., 2018, "Confidence prediction for lexicon-free ocr," in *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)* (IEEE). pp. 218–225.

Mori, S., Suen, C. Y., and Yamamoto, K., 1992, "Historical review of ocr research and development," *Proceedings of the IEEE* **80**, 1029–1058.

Nair, K., and Jawahar, C., 2010, "A Post-Processing Scheme for Malayalam Using Statistical Subcharacter Language Models," *Proceeding of the IAPR Workshop on Document Analysis Systems (DAS)*

Narasimhan, H., Vaish, R., and Agarwal, S., 2014, "On the statistical consistency of plug-in classifiers for non-decomposable performance measures," in *Advances in Neural Information Processing Systems*, pp. 1493–1501.

Natarajan, P. S., MacRostie, E., and Decerbo, M., 2005, "The bbn byblos hindi ocr system," in *Document Recognition and Retrieval XII*, Vol. 5676 (International Society for Optics and Photonics). pp. 10–16.

Nayef, N., Yin, F., Bizid, I., Choi, H., Feng, Y., Karatzas, D., Luo, Z., Pal, U., Rigaud, C., Chazalon, J., *et al.*, 2017, "Icdar2017 robust reading challenge on multi-lingual scene text detection and script identification-rrc-mlt," in *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, Vol. 1 (IEEE). pp. 1454–1459.

Nguyen, T., Jatowt, A., Coustaty, M., Nguyen, N., and Doucet, A., 2019, "Deep Statistical Analysis of OCR Errors for Effective Post-OCR Processing," in *2019 ACM/IEEE Joint Conference on Digital Libraries (JCDL)*, pp. 29–38.

Nikitin, F., Dokholyan, V., Zharikov, I., and Strijov, V., 2019, "U-net based architectures for document text detection and binarization," in *International Symposium on Visual Computing* (Springer). pp. 79–88.

Nomura, S., Yamanaka, K., Katai, O., Kawakami, H., and Shiose, T., 2005, "A novel adaptive morphological approach for degraded character image segmentation," *Pattern Recognition* **38**

Norvig, P., 2011, "How to write a spelling corrector?." *http://norvig.com/spell-correct.html*

Pal, U., and Chaudhuri, B., 2004, "Indian script character recognition: a survey," *pattern Recognition* **37**, 1887–1899.

Pal, U., Kundu, P. K., and Chaudhuri, B. B., 2000, "OCR Error Correction of an Inflectional Indian Language Using Morphological Parsing," *Journal of Information Science and Engg.* **16**, 903–922.

Patel, D., and Katuri, S., 2015a, "Prakriyāpradarśinī - an open source subanta generator," in *16th World Sanskrit Conference*, pp. 195–221.

Patel, D., and Katuri, S., 2015b, "Subanta generator," *http://www.sanskritworld.in/sanskrittool/SanskritVerb/subanta.html*

Paul, D., and Chaudhuri, B. B., 2019, "A BLSTM Network for Printed Bengali OCR System with High Accuracy," *arXiv preprint arXiv:1908.08674*

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L., 2018, "Deep contextualized word representations," *arXiv preprint arXiv:1802.05365*

Rabiner, L., and Juang, B., 1986, "An introduction to hidden markov models," *ieee assp magazine* **3**, 4–16.

Raja, S., Mondal, A., and Jawahar, C., 2020, "Table structure recognition using top-down and bottom-up cues," *arXiv preprint arXiv:2010.04565*

Rasheed, S., Naeem, A., and Ishaq, O., 2012, "Automated number plate recognition using hough lines and template matching," in *Proceedings of the World Congress on Engineering and Computer Science*, Vol. 1, pp. 24–26.

Reddy, V., Krishna, A., Sharma, V. D., Gupta, P., Goyal, P., *et al.*, 2018, "Building a word segmenter for sanskrit overnight," *arXiv preprint arXiv:1802.06185*

Redmon, J., Divvala, S., Girshick, R., and Farhadi, A., 2016, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788.

Rice, S. V., Jenkins, F. R., and Nartker, T. A., 1995, *The fourth annual test of OCR accuracy*, Tech. Rep. (Technical Report 95).

Rigaud, C., Doucet, A., Coustaty, M., and Moreux, J.-P., 2019, "Icdar 2019 competition on post-ocr text correction," in *2019 International Conference on Document Analysis and Recognition (ICDAR)*, pp. 1588–1593.

Roche, E., and Schabes, Y., 1997, *Finite-state language processing* (MIT press).

Saluja, R., Adiga, D., Chaudhuri, P., Ramakrishnan, G., and Carman, M., 2017a, "Error Detection and Corrections in Indic OCR using LSTMs," in *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, Vol. 1 (IEEE). pp. 17–22.

Saluja, R., Adiga, D., Ramakrishnan, G., Chaudhuri, P., and Carman, M., 2017b, "A framework for document specific error detection and corrections in indic ocr," in *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, Vol. 4 (IEEE). pp. 25–30.

Saluja, R., Maheshwari, A., Ramakrishnan, G., Chaudhuri, P., and Carman, M., 2019a, "Robust End-to-end Systems for Reading License Plates and Street Signs," in *2019 15th IAPR International Conference on Document Analysis and Recognition (ICDAR)* (IEEE). pp. 154–159.

Saluja, R., Punjabi, M., Carman, M., Ramakrishnan, G., and Chaudhuri, P., 2019b, "Sub-word embeddings for ocr corrections in highly fusional indic languages," in *2019 International Conference on Document Analysis and Recognition (ICDAR)* (IEEE). pp. 160–165.

Sankar K, P., Jawahar, C., and Manmatha, R., 2010, "Nearest neighbor based collection ocr," in *Proceedings of the 9th IAPR International Workshop on Document Analysis Systems*, pp. 207–214.

Sankaran, N., and Jawahar, C., 2012, "Recognition of Printed Devanagari Text Using BLSTM Neural Network," in *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)* (IEEE). pp. 322–325.

Sankaran, N., and Jawahar, C., 2013, "Error Detection in Highly Inflectional Languages," in *12th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, pp. 1135–1139.

Schuster, M., and Paliwal, K. K., 1997, "Bidirectional Recurrent Neural Networks," *IEEE Transactions on Signal Processing* **45**

Sennrich, R., Haddow, B., and Birch, A., 2016, "Neural machine translation of rare words with subword units," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (Association for Computational Linguistics). pp. 1715–1725.

Sharma, A., and Chaudhary, D. R., 2013, "Character recognition using neural network," *International journal of engineering Trends and Technology (IJETT)* **4**, 662–667.

Shi, B., Bai, X., and Yao, C., 2017, "An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition," *IEEE transactions on pattern analysis and machine intelligence* **39**

Shotton, J., Winn, J., Rother, C., and Criminisi, A., 2009, "Textonboost for Image Understanding: Multi-Class Object Recognition and Segmentation by Jointly Modeling Texture, Layout, and Context," *International journal of computer vision* **81**, 2–23.

Singh, A. K., and Jawahar, C., 2015, "Can RNNs reliably separate script and language at word and line level?." in *2015 13th International Conference on Document Analysis and Recognition (ICDAR)* (IEEE). pp. 976–980.

Singh, P., Patwa, B., Saluja, R., Ramakrishnan, G., and Chaudhuri, P., 2019, "Streetocrcorrect: An Interactive Framework for OCR Corrections in Chaotic Indian Street Videos," in *2019 International Conference on Document Analysis and Recognition Workshops (ICDARW)*, Vol. 2 (IEEE). pp. 36–40.

Smith, R., 2007, "An overview of the tesseract ocr engine," in *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, Vol. 2 (IEEE). pp. 629–633.

Smith, R., 2011, "Limits on the Application of Frequency-based Language Models to OCR," in *11th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, pp. 538–542.

Smith, R. W., 2013, "History of the tesseract ocr engine: what worked and what didn't," in *Document Recognition and Retrieval XX*, Vol. 8658 (International Society for Optics and Photonics). p. 865802.

Smith, R., Gu, C., Lee, D.-S., Hu, H., Unnikrishnan, R., Ibarz, J., Arnoud, S., and Lin, S., 2016, "End-to-end interpretation of the french street name signs dataset," in *European Conference on Computer Vision* (Springer). pp. 411–426.

Smith, R. W., 2009, "Hybrid page layout analysis via tab-stop detection," in *2009 10th International Conference on Document Analysis and Recognition* (IEEE). pp. 241–245.

Sommer, L., 2016, ""sanskrit has guided me to the finnish language": Herman kellgren's writings on finnish or the dilemmas of a fennoman," *Historiographia Linguistica* **43**, 145–173.

Springmann, U., Reul, C., Dipper, S., and Baiter, J., 2018, "Gt4histocr: Ground truth for training ocr engines on historical documents in german fraktur and early modern latin,"

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R., 2014, "Dropout: a simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research* **15**

Sundermeyer, M., Schlüter, R., and Ney, H., 2012, "Lstm Neural Networks for Language Modeling," in *Thirteenth annual conference of the International Speech Communication Association (INTERSPEECH)*, pp. 194–197.

Sutskever, I., Martens, J., and Hinton, G. E., 2011, "Generating Text with Recurrent Neural Networks," in *Proceedings of the 28th International Conference on Machine Learning (ICML)*, pp. 1017–1024.

Sutskever, I., Vinyals, O., and Le, Q. V., 2014, "Sequence to Sequence Learning with Neural Networks," in *Advances in neural information processing systems*, pp. 3104–3112.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z., 2016, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826.

Talathi, S. S., and Vartak, A., 2015, "Improving Performance of Recurrent Neural Network with Relu Nonlinearity," *arXiv preprint arXiv:1511.03771*

Tesseract, 2020, "Tesseract Open Source OCR," *https://github.com/tesseract-ocr/*

Toselli, A. H., and Vidal, E., 2013, "Fast hmm-filler approach for key word spotting in handwritten documents," in *2013 12th International Conference on Document Analysis and Recognition* (IEEE). pp. 501–505.

Van Strien, D., Beelen, K., Ardanuy, M. C., Hosseini, K., McGillivray, B., and Colavizza, G., 2020, "Assessing the Impact of OCR Quality on Downstream NLP Tasks.." in *ICAART (1)*, pp. 484–496.

Varfolomieiev, A., and Lysenko, O., 2016, "An improved algorithm of median flow for visual object tracking and its implementation on ARM platform," *Journal of Real-Time Image Processing* **11**, 527–534.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I., 2017, "Attention is all you need," in *Advances in neural information processing systems*, pp. 5998–6008.

Veit, A., Matera, T., Neumann, L., Matas, J., and Belongie, S., 2016, "Coco-text: Dataset and benchmark for text detection and recognition in natural images," *arXiv preprint arXiv:1601.07140*

Vinitha, V., and Jawahar, C., 2016, "Error Detection in Indic OCRs," in *2016 12th IAPR Workshop on Document Analysis Systems (DAS)* (IEEE). pp. 180–185.

Vorbeck, F., Ba-Ssalamah, A., Kettenbach, J., and Huebsch, P., 2000, "Report generation using digital speech recognition in radiology," *European Radiology* **10**, 1976–1982.

W. J. Hannan, E. M., G. L. Ficher et al., and Wemer, 1962, "R.C.A. multifont reading machine," *Optical Character Recognition*, 3–14.

Watanabe, T., Luo, Q., and Sugie, N., 1995, "Layout Recognition of Multi-Kinds of Table-Form Documents," *IEEE Transactions on Pattern Analysis and Machine Intelligence* **17**, 432–445.

Whitelaw, C., Hutchinson, B., Chung, G. Y., and Ellis, G., 2009, "Using the web for language independent spellchecking and autocorrection," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing: Volume 2* (Association for Computational Linguistics). pp. 890–899.

Wilcox-O'Hearn, A., Hirst, G., and Budanitsky, A., 2008, "Real-Word Spelling Correction with Trigrams: A Reconsideration of the Mays, Damerau, and Mercer

Model," in *International Conference on Intelligent Text Processing and Computational Linguistics*, pp. 605–616.

Wojna, Z., Gorban, A. N., Lee, D.-S., Murphy, K., Yu, Q., Li, Y., and Ibarz, J., 2017, "Attention-Based Extraction of Structured Information from Street View Imagery," in *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, Vol. 1 (IEEE). pp. 844–850.

Xie, Z., Avati, A., Arivazhagan, N., Jurafsky, D., and Ng, A. Y., 2016, "Neural Language Correction with Character-based Attention," *arXiv preprint arXiv:1603.09727*

Yalniz, I. Z., and Manmatha, R., 2011, "A Fast Alignment Scheme for Automatic OCR Evaluation of Books," in *2011 International Conference on Document Analysis and Recognition* (IEEE). pp. 754–758.

Yoon, Y., Ban, K.-D., Yoon, H., and Kim, J., 2011, "Blob extraction based character segmentation method for automatic license plate recognition system," in *2011 IEEE International Conference on Systems, Man, and Cybernetics* (IEEE). pp. 2192–2196.

Zhang, Y., Zha, Z. Q., and Bai, L. F., 2013, "A license plate character segmentation method based on character contour and template matching," in *Applied Mechanics and Materials*, Vol. 333 (Trans Tech Publ). pp. 974–979.

Zhang, Z., Huang, Y., and Zhao, H., 2018, "Subword-augmented embedding for cloze reading comprehension," *CoRR* **abs/1806.09103**

Zhong, X., Tang, J., and Yepes, A. J., 2019, "Publaynet: Largest Dataset Ever for Document Layout Analysis," in *2019 International Conference on Document Analysis and Recognition (ICDAR)* (IEEE). pp. 1015–1022.

Acknowledgements

We thank Nvidia for the GPU support. I would also like to thank Devaraj Adiga, Ayush Maheshwari, Mayur Punjabi, Pankaj Singh and Bhavya Patwa for their contributions.

Rohit Saluja
IIT Bombay
22 January 2021