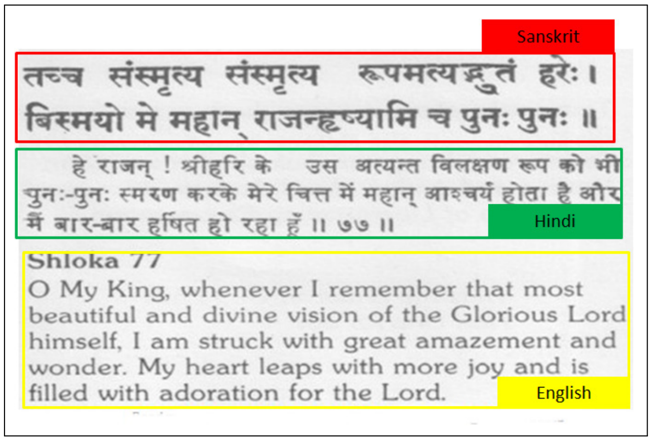
Multilingual OCR for Indic Scripts

Minesh Mathew, Ajeet Kumar Singh and C. V. Jawahar
Center for Visual Information Technology, IIIT Hyderabad, India.

***Abstract*—In Indian scenario, a document analysis system has to support multiple languages at the same time. With emerging multilingualism in urban India, often bilingual, trilingual or even more languages need to be supported. This demands development of a multilingual OCR system which can work seamlessly across Indic scripts. In our approach the script is identified at word level, prior to the recognition of the word. An end-to-end RNN based architecture which can detect the script and recognize the text in a segmentation-free manner is proposed for this purpose. We demonstrate the approach for 12 Indian languages and English. It is observed that, even with the similar architecture, performance on Indian languages are poorer compared to English. We investigate this further. Our approach is evaluated on a large corpus comprising of thousands of pages. The Hindi OCR is compared with other popular OCRs for the language, as a further testimony for the efficacy of our method.**

***Keywords*—Multilingual OCR, RNN, Indic Scripts**


Sanskrit

तच्च संस्मृत्य संस्मृत्य रूपमत्यद्भुतं हरेः ।
बिस्मयो मे महान् राजन्हृष्यामि च पुनः पुनः ॥

Hindi

हे राजन् ! श्रीहरि के उस अत्यन्त विलक्षण रूप को भी पुनः-पुनः स्मरण करके मेरे चित्त में महान् आश्चर्य होता है और मैं बार-बार हर्षित हो रहा हूँ ॥ ७७ ॥

English

Shloka 77
O My King, whenever I remember that most beautiful and divine vision of the Glorious Lord himself, I am struck with great amazement and wonder. My heart leaps with more joy and is filled with adoration for the Lord.

Fig. 1. **Example of a trilingual document**. This document has three languages — Sanskrit, Hindi and English.

I. Introduction

India uses 22 official languages [1], and many more unofficial languages for its communication, administration and documentation of her cultural heritage. Most of these languages have their own scripts, which had also gone through revisions (e.g. natural evolution, revisions to support the technology) on a regular basis.

Though many of these languages share common linguistic and grammatical structures, script remain very different except for few languages (among the 12 languages Hindi and Marathi use the same script–Devanagari and Bangla, Assamese and Manipuri use Bangla script. Others have their own unique scripts. See Figure 2.) Non-standardization of fonts and their rendering schemes, especially the ones designed prior to the emergence of Unicode, has made the development of OCRs further challenging. Though there have been many attempts in developing OCRs for Indian scripts from the 1970s to the beginning of this decade [2, 3, 4], methods that can scale across languages and yield reasonable results over a wide variety of documents are not yet devised.

Data driven methods using machine learning for the prediction are the natural choice for adapting the solution to newer scripts and styles with minimal effort. Ideas from machine learning were used for Indic scripts with the help of nearest neighbors [5], neural networks [6] and Support Vector Machine (SVM) methods [7]. Due to the mismatch between the basic units for representation and rendering, (Unicode, *Akshara* and the Glyphs), creation of examples for fully supervised machine learning methods remain very hard. Employing large number of training examples and using a classifier that can scale to these examples resulted in highly accurate character classifiers [8]. Even with powerfully trained classifiers, such methods had difficulty in robustly segmenting/detecting the character occurrences during prediction phase.

In recent years, we have explored the utility of Recurrent Neural Networks (RNN) for recognition of Indic scripts [9, 10, 11]. In [11], more than 97% accuracy was reported for some of the Indian languages on ∼1000 pages. RNN based methods are suited for Indic scripts as they are segmentation- free, needs minimal supervision and uses transition probabilities at feature unit level. In this work, we extend this direction of research further.

- The solution is extended to more Indian languages. Our results are validated on larger data sets (∼5000 pages) and results are superior to [11].
- A holistic solution to suit a natural multilingual setting is developed. We employ the ideas from [12] to design a fully RNN based scheme for this purpose.
- We further provide insight into a set of specific issues that results in lower accuracy across Indic scripts compared to English even when the recognition architectures are same and the amount of data used is similar.
- Finally we empirically compare our approach with a set of popular emerging OCR solutions for Indic scripts, including that of Google OCR. Even with out any special post processing scheme, our method outperforms these methods that could possibly be using lessons from English, very powerful algorithms and computational resources. This validation is done on Hindi[1].

[1]For data, resources and results please visit http://ocr.iiit.ac.in/

II. Multi-Lingual Indic OCR

There were attempts in the direction of Multilingual OCR (M-OCR), especially bilingual OCRs (for example, Bangla-Hindi [13] and Hindi-Telugu [8]). Some others were bilingual in the sense that English was supported in addition to an Indian language. Most of the earlier methods had script specific rules for character segmentation and input representation. Our approach to M-OCR, follows a uniform method across the languages. It is designed to recognize 12 Indian languages and English. It is essentially a set of OCR engines used along with a script identification module. In the following sections our approach is described in the context of challenges posed by Indian languages.

A. *Challenges in Input and Output Space*

First generation of Indic OCRs used rule based solutions and intuitive features for the recognition of characters. The second generation OCRs continued with the definition of characters, but moved forward with more principled features based on signal processing or statistical techniques. Traditional machine learning based methods used for Indic OCRs used multi-class classification schemes implemented with neural networks or SVMs. However, such solutions demanded two separate modules. 1. A segmentation step that could create isolated character examples. 2. A step that convert the class labels into a Unicode sequence based on the ordering of these characters.

Indian language OCRs that were developed until recently (with exceptions like [14]) used a segmentation scheme prior to recognition. However, due to the complex nature of the scripts, robust segmentation remained as the hardest block in the pipeline. From a machine learning perspective, OCR is more of a structured prediction problem where the output is a sequence of characters/symbols of arbitrary length and input is a sequence of feature vectors of varying length. For similar structured prediction problems, people have used Max-Margin Markov (M3) Networks [15], Hidden Markov Models (HMM) [14], and RNNs [9]. This direction of research was widely appreciated as segmentation-free recognition schemes.

For Indic scripts, input and output spaces are some what more complex compared to English. The output space is much larger. When all the possible characters which can be generated by composition of consonants and vowels are considered, the number of classes for Indian languages would be a prohibitively large number. If the output space is modelled as a series of k *askharas*, the output space would be all k-permutations of these classes. For this work we use the output space as a series of Unicodes. This considerably reduces the size of the output space as total number of Unicodes in a language is usually less than 200 (127 for Devanagari). Even after this the size of the output space is larger compared to English. At the input space, the feature sequence could differ only minimally to yield very different Unicode sequences. This demands precise modelling of the contextual dependencies of the input sequence. Few examples where fine character shape variations, resulted in wrong output sequences are shown in

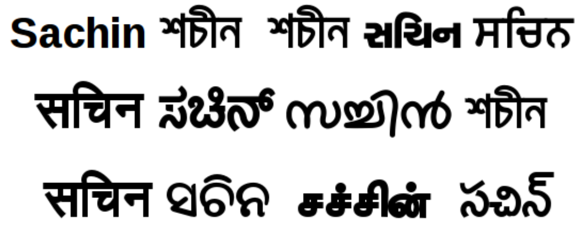


Fig. 2. **Examples of words from the popular languages used in India**. (from top left English, Assamese, Bengali, Gujarati, Gurumukhi, Hindi, Kannada, Malayalam, Manipuri, Marathi, Odia, Tamil and Telugu) Many of these languages have lot of commonalities in the language and organization of the scripts. However, the visual appearance of the scripts vary a lot, making the OCR development a different problem for each language.

section III. A RNN using Connectionist Temporal Classification (CTC) [16] algorithm can do this in a segmentation-free manner.

In the design of M-OCR for Indic scripts, we make use of RNN, for two different but related problems — script identification and segmentation-free OCR. Script identification module is a RNN trained for sequence classification. The network is trained to classify an input sequence of profile features to a target script. Once the script is identified the word is recognized on another RNN, which is trained for sequence transcription. This network transcribes an input sequence of binary pixel values to output Unicode sequence. The utility of RNN for both the tasks - sequence classification and sequence transcription, is discussed in the following two subsections.

B. *Scripts: To Separate or Not ?*

There are two different ways to build M-OCRs. i) train a single OCR for all the languages, or ii) train an isolated OCR for each language. Both these methods have their own advantages and disadvantages. In first case, RNN would require a very large corpus of data and cardinality of the output space is much larger. With larger output space, the RNN learning becomes time consuming as well as computationally expensive. However, in latter case, the data required for RNN training would be lesser and the output space would be smaller. In this case the network would converge to an optimum in lesser time. This approach defeats the purpose of the M-OCR where we want a "single" OCR to recognize multiple languages. Due to the demerits mentioned above for a lone M-OCR, we introduce a process of script separation in the M-OCR pipeline which would identify the script of the incoming word beforehand and then send the word to the corresponding script's OCR engine.

In a recent work [12], it was shown that RNNs can easily be used for script and language separation which do not require any special tuning (n-grams, textures). In this approach, the method represents word images as a sequence of feature vectors, and then employs a RNN for the script and language identification.

In Section III-A we establish the case for script separation module in M-OCR, by empirically comparing the two approaches discussed.

C. English vs Indian Languages

In this section we investigate why the inherent challenges in Indic scripts make the sequence transcription harder compared to English. We also analyze how the prolonged memory of the LSTM cells help a RNN to deal with these challenges.

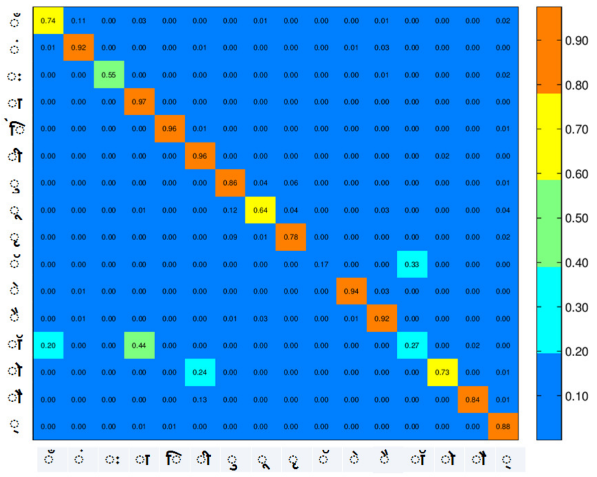


Fig. 3. **Confusion matrix for *matras* (vowel modifiers) in Devanagari script .**

Word Type	# Words	Char. Error	Word Error
All Words	1.20M	2.30	8.4
Words w/o *matra(s)*	0.25M	2.34	4.10
Words with *matra(s)*	1M	2.30	11.70
Words with easily confusable *matra(s)*	0.20M	6.30	22.6

TABLE I

EFFECT OF *matras* ON RECOGNITION IN HINDI WORDS WITH *matras* ARE MORE ERRONEOUS. *Matras* WHICH ARE BEING ATTACHED TO THE BOTTOM OF OTHER SYMBOLS(U (0941), UU (0942), R (0943), RR (0944) AND HALANT (094D)) ARE EASILY CONFUSABLE. WORD ERROR RATES ARE MUCH HIGHER IN SUCH CASES AS SHOWN IN THE LAST ROW.

Firstly we see how the similar looking glyphs in Indic scripts, especially the vowel modifiers (*matras*) make the recognition harder. Figure 3 shows confusion matrix for (*matras*) in Devanagari script, which have lot of similar looking glyphs among them. Higher rate of confusion among the *matras* made us probe if this had any effect on the performance of Hindi OCR. As evident from the Table I words with *matras* have higher recognition error rates than words which do not have *matras*. The relative difficulty in learning *matras*, especially the ones attached to the bottom parts of other symbols is reflected in the output probabilities of these classes during recognition. This is shown in Figure 4. Output activations for classes corresponding to *matras* and other easily confusable symbols are relatively lower.

The ability of RNNs to model contextual dependencies within an input sequence, is critical in transcription tasks involving Indic scripts, as the confusion among symbols is

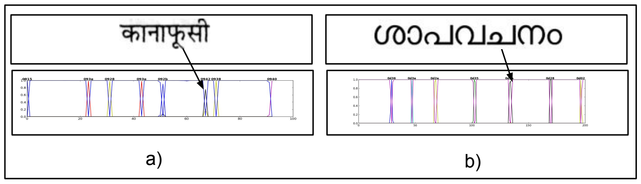


Fig. 4. **Output Activations for a Hindi and a Malayalam word.** Activation for the *matra* symbol is relatively lesser than the activations for other classes in case of the Hindi word (a). For the Malayalam word (b), output activation of 5th character is slightly lesser compared to other characters. This character is visually similar to the preceding two characters.

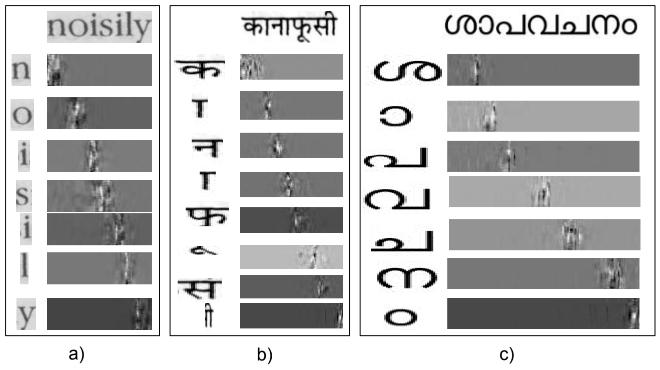


Fig. 5. **Visualizing use of context using Input Jacobians.** Input Jacobian is a means to assess the relative influence inputs on the output. Here Jacobians of all top classes, at times when they are emitted are plotted. Brighter areas in the plots correspond to inputs, which influenced the decision the most.

more. Therefore it is important to analyze how and where the network uses context during a particular input sequence. In case of RNNs we can have insights into the use of context by measuring the sensitivity of the outputs to inputs. The input Jacobian for a RNN, is defined as, $J_{ki}^{tt'} = \frac{\partial y_k^t}{\partial x_i^{t'}}$. The above equation defines the relative sensitivity of output y_k at time t to the input x_i at time t'. Since raw features are used in our implementation, if we compute derivatives of a single output unit at a particular time with respect to all inputs at all times throughout the sequence, we would a get 2D matrix of the same size as the input word image. If these matrices are visualized as 2D images, brighter pixels would be indicative of pixels in the input word image, which the output is more sensitive to. In Figure 5, input Jacobains for an (a) English, (b) Hindi and (c) Malayalam words are shown. Jacobians are plotted as 2D images as described above, for the top classes, when the classes are emitted. The magnitude of the derivatives forms an 'envelope' around time at which derivative is measured (t). The derivatives remain large for around 10-20 pixels before and after t. In case of English the 'envelope' is more vertically oriented, and in case of Malayalam it is spread in all directions, reflecting the curvy writing style of the Malayalam script. For the Hindi word, the *matra* symbol, which had relatively lower output activation (Figure 4), the 'envelope' is broader indicating use of more context in the decision. It is evident from the input Jacobian plots that, the network make use of wider context, whenever similar looking characters are encountered.

D. Implementation

The recognition system comprises of two RNNs. The first one is used for script identification and the other for recognition. Hidden layers of both the networks consist of 3 levels of 50 Long Short Term Memory LSTM [17] nodes. There are two sets of such hidden layers. One processes the input sequence from beginning to end and the other in the reverse direction. In this manner, context in both directions is made available. The input is fully connected to both the hidden layers and the output layer is fully connected to the two hidden layers.

For script identification, profile features extracted over a sliding window are used. [12]. The 12 input nodes of the RNN are fed with the 12 profile features, one timestep at a time. Output layer is a soft-max layer with as many nodes as the number of target scripts plus a class for *blank* or *null* predictions.

The second RNN has a CTC output layer. All Unicodes in the language block, Arabic numerals, basic punctuation symbols and the *null* label form the set of output classes for each language. Raw features of binarized word image are used as features in this case. We resize the word images to a height of 32 pixels, and hence a column of 32 pixels forms a feature vector. The sequence of feature vectors is fed to the network one at a time.

III. Results and Discussions

In this section results of performance evaluation of M-OCR and a discussion based on the results are presented. Recognition performance with and without script separation is studied in III-A. In section III-B quantitative and qualitative results of M-OCR evaluation on a large corpus is presented. This is is followed by a performance comparison of our system with other popular OCRs for Hindi. In III-D we analyze why a fully RNN based approach to M-OCR is well suited for Indic scripts.

A. Script separation

In Section II-B, we discuss two ways to build a M-OCR. One method is to train a single flat OCR for all languages. The other method is to have a system with script separation module and language specific OCRs in a hierarchical manner. Here, we contrast both the methods and comment on the need for a script separation module in a M-OCR.

In our first experiment, script separation is tried out on two groups of 4 scripts - groups of South Indian and North Indian scripts. We represent the word images as sequential features using the method described in [12]. Both the groups were mixed with English language. A RNN is trained for each group separately. For both the groups, 100K words are used for training and the resultant network is validated on 25K words.

Results of the script separation experiment is shown in Table II. We report an accurate script separation for both of the aforementioned groups. From the evidence, we can safely infer that RNN is able to identify between English and other Indic scripts accurately. Also, separation among Indic scripts is very high. From North group, we are getting a higher accuracy of 99.29% for Gujarati and lower accuracy of 98.40% for Hindi. From South group, we report a high accuracy of 99.57% for Malayalam and lower accuracy of 98.78% for Kannada. The lower accuracy of Hindi is probably due to its similarity with Gurumukhi and similarly the lower accuracy of Kannada can be attributed to its similarity with Telugu

Scripts	Acc(%) (word)
North Scripts	
English	99.99
Hindi	98.40
Bangla	99.16
Gurumukhi	98.63
Gujarati	99.29

Scripts	Acc (%) (word)
South Scripts	
English	99.98
Kannada	98.78
Malayalam	99.57
Tamil	99.13
Telugu	99.15

TABLE II
Script Separation Results on groups of North Indian and South Indian Scripts

	L1+L2								**L1+L2+L3**			
(Bi/Tri)-lingual	**B1**		**B2**		**B3**		**B4**		**T1**		**T2**	
OCR	*bf*	*ho*	*bf*	*ho*	*bf*	*ho*	*bf*	*ho*	*tf*	*ho*	*tf*	*ho*
Average Char. Error Rate	3.88	2.87	2.16	1.65	2.31	2.13	1.11	0.61	3.85	3.31	2.65	2.02

TABLE III
Multilingual OCRs: comparison of bilingual (*bf*) and trilingual (*tf*) OCRs with hierarchical (*ho*) OCR. Here, B1,B2,B3 and B4 are ENG+HIN, ENG+BAN, ENG+KAN, ENG+TEL OCRs respectively. T1 and T2 are ENG+HIN+KAN and ENG+KAN+TEL OCRs respectively. In both bilingual and trilingual settings, hierarchical OCRs outperform flat OCRs.

In order to establish the need for script separation in M-OCR, we perform experiments with and without script separation in two multilingual settings - bilingual and trilingual. The bilingual OCRs (*bfOCR*) are, i) English(Eng) and Hindi(Hin), ii) English and Bangla(Ban), iii) English and Kannada(Kan) and iv) English and Telugu(Tel). And, the trilingual OCRs (*tfOCR*) are, i)English, Hindi and Bangla and ii) English, Kannada and Telugu. In contrast to this a hierarchical system is implemented(*h*OCR), which is a combination of script separation module and isolated language specific OCRs. Now we compare the bilingual and trilingual flat OCRs with the proposed system– a *h*OCR. This comparison will enable us to comment on the requirement for script separation module in M-OCRs.

Table III shows the comparison of our hierarchical *h*OCR with bilingual and trilingual flat OCRs. In both settings, the hierarchical OCR comprising of a script separation module and individual OCRs outperforms, flat bilingual or trilingual OCRs. It is evident from these results that a hierarchical OCR outperform flat, lone OCRs, in a multilingual setting, involving a variety of scripts.

B. Evaluation on a Large Corpus

Language	#Books	#Pages	#Lines	#Words
Assamese	20	3.5K	93K	0.7M
Bangla	13	2.8K	96K	1.0M
Gujarati	25	5.0K	140K	1.1M
Gurmukhi	32	5.0K	140K	1.6M
Hindi	33	5.0K	112K	1.3M
Kannada	27	5.0K	129K	0.8M
Malayalam	31	5.0K	183K	1.0M
Manipuri	25	3.5K	91K	0.7M
Marathi	20	5.0K	154K	1.4M
Odia	17	5.0K	151K	1.4M
Tamil	23	5.0K	147K	0.7M
Telugu	28	5.0K	134K	0.8M
English	12	1.5K	47K	0.5M

TABLE IV
DETAILS OF THE DATASET USED IN OUR EXPERIMENTS.

Our dataset of Indian languages and English consist of scanned images of printed documents with moderate complexity. All the pages have been typed manually and is annotated at word level using a semi-supervised approach [18].

The evaluation measure used for evaluating M-OCR performance is Character Error Rate (CER). Character error rate is defined as,

$$CER = (\frac{Total \quad Char \quad Errors}{Total \quad Characters}) \times 100$$

Total Character Error is computed using Levenshtein distance method, by computing the distance between the output Unicode sequence and the desired Unicode sequence (Ground truth).

Language	# Pages Tested	Only Recognition		Segmentation + Recognition
		Word	Line	Line
Assamese	1000	1.78	1.65	2.10
Bengali	1300	2.13	2.22	2.30
Gujarati	3500	3.42	3.00	4.70
Gurmukhi	3500	1.28	1.22	2.30
Hindi	3000	2.30	2.00	3.90
Kannada	3500	4.10	4.16	5.60
Malayalam	3500	0.88	0.74	3.60
Manipuri	2000	1.30	1.21	2.30
Marathi	3500	1.29	1.10	3.80
Odia	3500	3.49	2.40	3.30
Tamil	3500	2.44	4.00	5.60
Telugu	3500	2.00	1.90	4.86
English	300	0.93	0.65	1.25

TABLE V
M-OCR EVALUATION RESULTS . CHARACTER ERROR RATES OF ALL OCRS AFTER EVALUATION ON THOUSANDS OF PAGES IN EACH LANGUAGE. 'ONLY RECOGNITION' IS THE CASE WHERE, LINE OR WORD IMAGES COULD BE USED DIRECTLY (USING ANNOTATION). THE LAST COLUMN SHOWS RESULTS OF OCR, WHEN THE PAGES WERE ANALYZED AND SEGMENTED INTO TEXT LINES AUTOMATICALLY.

Table V shows the results of OCR transcription for all languages. We tried out both word level and line level transcriptions when segmentation was available (i.e only recognition is evaluated). The last column of the table shows error rates when segmentation was automatic and transcription was at line level. In each language around 20% of the total pages were used for training, 10% for validation and the rest for testing. The results of our approach are analyzed qualitatively in Table VI. We study the success and failures under three themes - script misclassification, Recognition in presence of easily confusable symbols and effect of image degradation.

C. Comparison with other systems for Hindi

OCR	Char. Error Rate
NEW OCR [19]	67.40
i2OCR [20]	10.00
ind.senz [21]	8.57
Google [22]	7.10
Our Method	**5.85**

TABLE VII
PERFORMANCE COMPARISON OF OUR SYSTEM WITH OTHER POPULAR OCRS IN HINDI.

In the past few years there has been mounting interest in Indian language OCR, especially in Devanagari OCR. There have been efforts to train the Tesseract OCR engine for Devanagari script and few commercial OCR systems have also been released. We have compared our Hindi OCR with the popular Hindi OCRs and the results are presented in VII.A separate dataset of 100 images, with varying levels of image quality and layout complexity were used in the experiment.

D. Discussion

Amidst the inherent complexities of Indian scripts and wide variations in the input and output spaces across languages, a practical solution to OCR for Indic languages using modern machine learning tools was devised. We report accuracies of more than 95% for all the languages, even after segmentation errors are counted for (Table V). And this is further corroborated by Table VII, where our method outperformed other popular Hindi OCRs. Most of these systems use language specific rules, and often employ a post processor in the final stage to refine the recognition results. In contrast to this we follow a uniform approach for all languages, relying solely on the sequence learning capabilities of the RNNs.

A RNN with a CTC output layer could directly map from a sequence of input vectors to output Unicode sequence. This ability to transcribe did away with the need for segmentation of text lines into words, and further segmentation of words to symbols. This was critical in our case where sub word segmentation is often difficult.

Unicode representation of Indian scripts is not monotonic and requires reordering of the Unicode points at places where *matras* (vowel modifiers) are present. In cases where latent symbols are classified, explicit rules had to be written to map from the output symbols to Unicode sequence of the word. In a transcription setting, the Unicode reordering scheme is learned by the network, on its own.

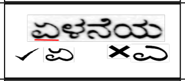
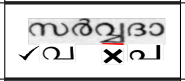
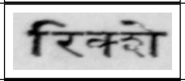
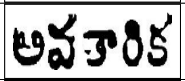
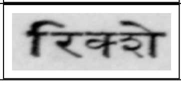
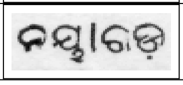
	Hindi	Telugu	Tamil	Kannada	Malayalam	Odia
A	✗ Kannada	✗ Malayalam	✗ Oriya	✗ Oriya	✗ Kannada	✗ Telugu
B						
C						
D						

TABLE VI

Qualitative results of our Approach to Script Separation and ocr. Examples of script misclassification are shown in (a). In (b) are recognition results where a character is confused with another similar looking character. Degraded images which were not correctly recognized are shown in (c). And (d) shows images which were correctly recognized, even with degradations

RNNs with LSTM cells in the hidden layers have access to unlimited range of context. And with two sets of hidden layers for processing the input sequence in both forward and backward directions, both past and future contexts were made available while emitting any output. This helped in two ways in our case. Most Indian languages are highly inflectional which induces a large vocabulary and longer average word lengths. LSTM cells' ability to keep gradients for unlimited time solved the problem of vanishing gradients and any length sequences could be transcribed with full access to the context. Secondly availability of longer contexts from both past and future, helped in accurate predictions in a setting where *matras*, and other similar looking symbols demanded perfect modelling of contextual dependencies in the input sequence.

A web interface for M-OCR is being developed and will soon be available for the general public. For details please visit http://ocr.iiit.ac.in/.

IV. Conclusion

Our approach presented in this paper, addresses the need for a multilingual OCR in Indian setting. A single recognition system, but comprising of multiple OCR engines was used. The crux of our solution lies in detecting the script prior to recognition. This enabled each word to be recognized on an OCR, trained exclusively for the script. Each individual OCR was devised using a RNN, without any language specific modelling. We have investigated 'why' and 'how' RNNs could do this, despite the complexity of the scripts and variations across scripts. With the web based system in place, we would get a chance to evaluate our sytem on wide variety of documents. We are working on methods to make the best use of the new data, in further improving the performance of our system.

References

[1] "Census 2011," http://censusindia.gov.in/.
[2] R. M. K. Sinha and H. Mahabala, "Machine recognition of devnagari script," in *IEEE Trans. on Systems, Man and Cybernetics*, 1979.
[3] B. Chaudhuri and U. Pal, "A complete printed bangla OCR system," *Pattern recognition*, 1998.
[4] D. Arya, T. Patnaik, S. Chaudhury, C. V. Jawahar, B.B.Chaudhuri, A.G.Ramakrishna, C. Bhagvati, and G. S. Lehal, " Experiences of Integration and Performance Testing of Multilingual OCR for Printed Indian Scripts," in *J-MOCR Workshop,ICDAR*, 2011.
[5] K. Aparna and A. Ramakrishnan, "A complete tamil optical character recognition system," in *DAS*, 2002.
[6] R. Sanjeev Kunte and R. Sudhaker Samuel, "A simple and efficient optical character recognition system for basic symbols in printed kannada text," *Sadhana*, vol. 32, no. 5, 2007.
[7] T. Ashwin and P. S. Sastry, "A font and size independent OCR system for printed kannada documents using support vector machines," in *Sadhana*, 2002.
[8] C. V. Jawahar, M. N. S. S. K. P. Kumar, and S. S. R. Kiran, "A bilingual OCR for hindi-telugu documents and its applications," in *ICDAR*, 2003.
[9] N. Sankaran and C. V. Jawahar, "Recognition of printed devanagari text using blstm neural network," in *ICPR*, 2012.
[10] N. Sankaran, , and C. V. Jawahar, "Devanagari Text Recognition:A Transcription Based Formulation," in *ICDAR*, 2013.
[11] P. Krishnan, N. Sankaran, A. K. Singh, and C. V. Jawahar, "Towards a robust OCR system for indic scripts," in *DAS*, 2014.
[12] A. K. Singh and C. V. Jawahar, "Can RNNs reliably separate script and language at word and line level?" in *ICDAR*, 2015.
[13] B. Chaudhuri and U. Pal, "An ocr system to read two indian language scripts: Bangla and devnagari (hindi)," in *DAS*, 1997.
[14] Premkumar S. Natarajan and Ehry MacRostie and Michael Decerbo, "The BBN Byblos Hindi OCR system," in *DRR*, 2005.
[15] B. Taskar, C. Guestrin, and D. Koller, "Max-margin markov networks," in *NIPS*, 2004.
[16] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks," in *ICML*, 2006.
[17] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, 1997.
[18] C. V. Jawahar and Anand Kumar, "Content-level Annotation of Large Collection of Printed Document Images," in *ICDAR*, 2007.
[19] "New ocr," https://www.newocr.com/.
[20] "i2ocr free hindi ocr," http://www.i2ocr.com/free-online-hindi-ocr.
[21] "ind.senz," http://www.indsenz.com/.
[22] "Google ocr," https://support.google.com/drive/answer/176692?hl=en.