Devanagari Text Recognition: A Transcription Based Formulation

Naveen Sankaran, Aman Neelappa and C. V. Jawahar
International Institute of Information Technology, Hyderabad, INDIA

***Abstract*—Optical Character Recognition (OCR) problems are often formulated as isolated character (symbol) classification task followed by a post-classification stage (which contains modules like Unicode generation, error correction etc.) to generate the textual representation, for most of the Indian scripts. Such approaches are prone to failures due to (i) difficulties in designing reliable word-to-symbol segmentation module that can robustly work in presence of degraded (cut/fused) images and (ii) converting the outputs of the classifiers to a valid sequence of Unicodes. In this paper, we propose a formulation, where the expectations on these two modules is minimized, and the harder recognition task is modelled as learning of an appropriate sequence to sequence translation scheme. We thus formulate the recognition as a direct transcription problem. Given many examples of feature sequences and their corresponding Unicode representations, our objective is to learn a mapping which can convert a word directly into a Unicode sequence. This formulation has multiple practical advantages: (i) This reduces the number of classes significantly for the Indian scripts. (ii) It removes the need for a reliable word-to-symbol segmentation. (ii) It does not require strong annotation of symbols to design the classifiers, and (iii) It directly generates a valid sequence of Unicodes. We test our method on more than 6000 pages of printed Devanagari documents from multiple sources. Our method consistently outperforms other state of the art implementations.**

I. Introduction

There have been many attempts in recognizing printed documents in Indian scripts [1], [2]. However, performance of the best available solution [3] is not yet comparable to their English counterparts. There are multiple reasons for this. The number of symbols (or segments) to be recognized is often few hundreds (eg. 800 for Devanagari [4]). Segmentation of words into symbols (basic unit for recognition) is hard due to similarity in characters, complex shapes etc. In our recent experience in design of OCRs for Indian languages [3], we have observed that the errors are primarily caused by two modules. The first one, which segments the word into symbols, and the second one which generates valid Unicode sequence form the classifier outputs. (See also Fig. 2(a) and Section II-A.)

In this paper, we argue that one can look at this problem from a different angle. We formulate the problem of OCR as an automatic transcribing task. We formulate this as that of learning a module which minimizes the transcription error (as well as the transposition error) directly. This formulation has multiple practical advantages. This method reduces the number of classes significantly for the Indian scripts. The number of labels that the recognition module need to output is limited to the space of Unicodes for a language (which is 127 for Devanagari). This is significantly lesser than the previous methods, which requires more than 800 classes [4]. However, in this case, the output is a structured space. However, this demands a different type of classifier. Our method removes the need for a word-to-symbol segmentation as popularly done in the case of recognition methods based on Hidden Markov Models(HMM) [4] and Recurrent Neural Networks (RNN) [5]. Since the annotation required for this module is only at the word level, we bypass the requirement of strong annotation of symbols to design the classifiers. Note that the definition of symbols is not yet standard. We do not explicitly use this latent script definition at any stage of our solution. Since we directly generate Unicode, our output is syntactically correct. Note that a significant percentage of errors in Indian language OCRs is visible as invalid Unicode sequences. We experimentally validate our approach on around 6000 printed documents and show that our method outperforms the other existing solutions. Fig. 1 shows a sample page from our database along with some words present in them. Our method was able to recognize many of these degraded words.

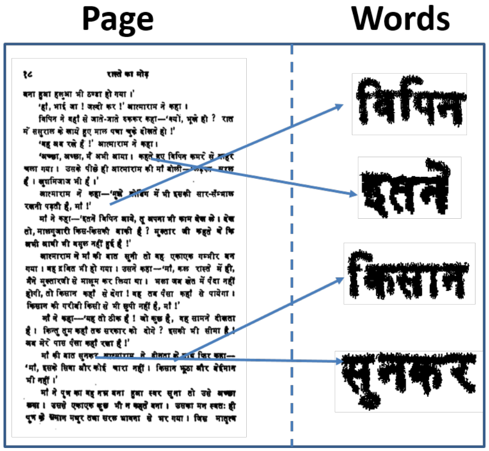

Fig. 1. A sample page from our dataset with examples of words present in it. We were able to recognize pages with similar degradation with very less error while many state-of-the-art OCR systems have high errors.

There have been many attempts in the past in recognizing Indian scripts. Initial attempts were based on modelling the shape with intuitive features. Simple classifiers like a KNN or Multilayer perceptions (MLP) were used for the recognition task. Pal and Chaudhuri [2] provides an excellent summary of this period. Bansal and Sinha [6] and Chaudhuri and Pal [7] have developed OCR systems for Hindi script based on isolated symbol recognition. More recently, Support Vector Machine (SVM) based techniques have been used in Hindi character classification [3]. The challenges present in character extraction and recognition lead to the popularity of HMMs [4] and other word recognition strategies [5] for Devanagari.

There are two popular schools in design of OCRs. A large

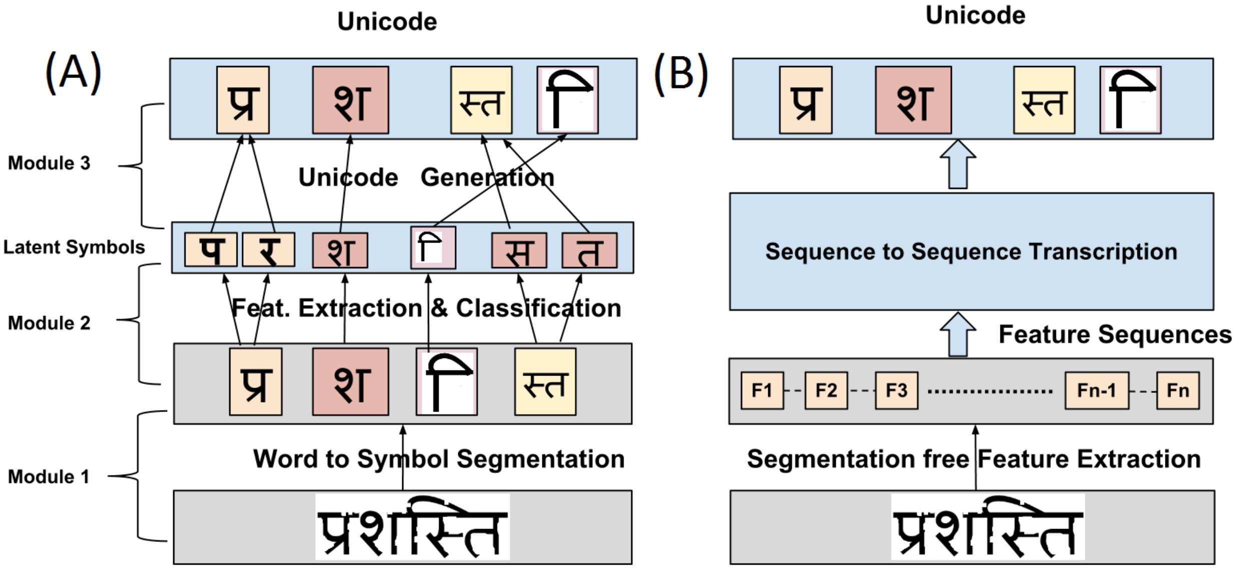

Fig. 2. Difference in the OCR solutions. (A) The architecture of a traditional OCR, which starts with symbol/character extraction and classification. (B) Our approach. We bypass the two harder modules of the traditional OCR. We directly output a Unicode sequence, given a word image.

number of methods depend on isolated character (symbol) classifiers like SVMs or MLPs. An alternative is to use segmentation free methods based on Hidden Markov Models (HMM) or Recurrent Neural Networks (RNN). Isolated character classifier based methods recognise individual symbols, which is then converted to respective Unicodes. However such methods were sensitive in dealing with issues related to word-symbol segmentation, as cut or fused symbols would make the recognition inaccurate. This demands a harder segmentation problem to solve [8], [9]. HMM based approaches explicitly tried to address this problem by defining the input as a sequence of feature vectors. HMM/RNN based solutions does not require explicit symbol segmentation for recognition [4], [5]. However, the definition of symbols and Unicode generation are still required [4], [5]. In this paper, we expand our previous work [5] by defining the problem as an end-to-end sequence transcription task.

II. OCR as Transcription

A. Recognition Challenges in Indic scripts

Fig. 2(A) depicts a schematic of a typical Devanagari OCR. Input word images are first segmented into symbols in module 1. However, this could become hard due to the definition of symbols, cuts and merges, and formatting/rendering issues in word-processors. Part of these issues (like cuts and merges) are common to almost all the OCRs. However, the similarities of sub-parts across symbols make the problem for Devanagari more challenging. Challenges in Module-3 is possibly more unique. The outputs of the isolated symbol classifiers need to be rearranged and processed to obtain a valid Unicode sequence. A single error in isolated character classification can result in multiple errors in the Unicode. Because of the specific manner in which Devanagari (and many other Indian languages) gets written, this conversion is not monotonic. i.e., later symbols could output earlier Unicodes. (Note the intersecting lines in Module 3).

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

Modelling recognition problem as translation or transcription has been attempted in the past. Object detection was modelled as machine translation [10] by annotating image regions with words from a lexicon. Translation techniques have also been used by linguists [11] and speech recognition community [12] for modelling recognition. While translation techniques focus on conversion of input to output using specific rules, transcription often involves mapping of input to a specified output, and is considered as a simpler task. We consider our problem as that of transcription where we start with a set of word image features, and obtain a corresponding Unicode text.

B. Learnable Transcription Module

The input to the transcription module is a feature sequence and output is a sequence of Unicodes. Feature extraction is one major challenge when we deal with languages with complex scripts. For our experiments, we extract 7 features from every word image. The features are extracted using vertical sliding windows with fixed width of 20px and an overlap of 75%. For every window, we scan from top to bottom and extract popular profile based features like upper profile, lower profile, vertical distance profile and ink-background transitions. More details about these features can be found in [13]. All the profiles are normalized with respect to the image height to [0,1] which we use for training the transcription module. We also make the features more representative of the given image by dividing the image horizontally into two and computing the above mentioned features for both regions. This results in generating 14 features. This increase in number of features though seems insignificant, have increased the accuracy in practice. This is because there are many symbols which look similar but appear in different areas. Splitting the word into two will help in differentiating such symbols.

C. Requirements of the Transcription Module

We model the problem of Unicode generation as transcription of features to Unicode text. We call it transcription as we try to map the input to a specified output. By this we mean that, given a sequence of feature vectors, the algorithm should be capable of generating the textual output. Input image is converted to feature vector sequence $f_1, f_2, ...f_N$; $f_i \in R^{14}$. This is then matched to the target Unicode labels. There are two important sub-problems to be solved for transcription. (i) Feature vectors need to be translated to corresponding Unicode. (ii) the transposition that has occurred in the script needs to be compensated and a correct sequence need to be learned. There are many possible approaches for solving the first task i.e., of decoding the variable length stochastic sequences. For example, Hidden Markov Models (HMMs) can do this task. This capability of HMM is exploited in the speech and handwriting recognition solutions. However, the second requirement makes the problem more complex. This demands two additional characteristics for the algorithm: (i) It should have memory so that it can remember the previous observations and output the label in a later time instance. (i.e., the reversal of ordering is possible). (ii) Solution should be capable of looking into the future (non-causal). These requires the solution to be non-Markovian.

III. Sequence to Sequence Transcription

One significant conclusion from previous section is that the sequence-sequence transcription module should be inherently capable of learning the context from the past inputs so that a proper prediction can be made during the Unicode generation. Since this requires memory based learning, recurrent neural networks were the first choice for this task. We decided to use a variant of RNN known as Bi-directional Long-Short Term Memory (BLSTM) [14]. This allows longer memory and non-causal sequence processing. A detailed introduction to the theory of this network is beyond the scope of this paper.

A. Recognition Module

We require a system that can transcribe feature sequence to Unicode sequence. It should accept features along with the corresponding Unicode text and recognize the mapping. BLSTM neural networks have been successfully used in the past for both printed and handwritten text recognition tasks [15], [16]. The distinctive feature about these networks is their ability to remember long range context over several timesteps. The use of Connectionist Temporal Classification (CTC) layer as the output layer allows the words to be presented as a sequence of unsegmented features thus doing away with the character segmentation issues in Indian languages. Further the bidirectional nature of the network makes it suitable for Unicode level learning as it is capable of handling Unicode reordering issues quite prevalent in Indian scripts. These features make BLSTM a natural choice for developing a document image translation system for Devanagari. Further details into BLSTM architecture can be found in [14]. One major advantage of using such a system is that given an input, we can directly model the output label sequence probability. This gives our method an edge over HMM based solutions [17] which are generative in nature. Also, our network models continues trajectories and can use contextual information that can span the entire input sequence, something which the HMM bases systems lack.

B. Training time reduction

Most of the RNNs are typically trained by Back Propagation Through Time (BPTT). Being a gradient based method, it is susceptible to being stuck in a local minima. One popular technique to avoid this issue is to use stochastic gradient descent [18] where the weight updates are based on only a subset of the complete training set. We found this to be extremely useful in our experiments. It was observed while dealing with large quantities of training data that it contains quite a few redundant samples. It seemed possible that a smaller subset of training samples should have the same information content in terms of training the network. Motivated by this we switched to using stochastic gradient descent by solving subsets of the training sets. After each epoch, a fixed number of examples were sampled with replacement. There is another way to view this strategy. The standard method of presenting data to the neural network consists of doing it in terms of epochs. That is, all the training samples are presented one after the other and the process is repeated till convergence. In the other extreme we can do a random sampling where we sample one example after the other with replacement. In the first case the network is presented with the complete information about the data only in quanta of the complete training set. In the latter case, the network is presented with the same information in any given k examples, for a reasonable size of k. The strategy of sampling a subset of the examples after each epoch is somewhere in the between these two. We provide 20% of training data at a time for training in this mode. This method converges to 5% training error after 42 epochs, taking totally 126 minutes while the normal method takes 10 epoch totalling 190 minutes. We had compared the total time taken for training a dataset so that the training error stabilizes and network convergences. As argued, normal approach takes very less number of epochs, but the time taken per epoch is very high. While comparing with stochastic method, it takes nearly an hour more to converge. This difference becomes significant when the training data is very large. Although this method had very little contribution towards final testing accuracy, it helped in reducing the training time considerably and thereby allowing us to perform multiple experiments and obtain better accuracies.

IV. Experiments, Results and Discussions

A. Dataset

For the purpose of evaluation, we used two annotated datasets. The first dataset consists of 5000 page corpus which

	Character Error Rate(CER)				Word Error Rate (WER)			
Dataset	**Our Method**	**BLSTM [5]**	**Char. OCR [3]**	**Tesseract [19]**	**Our Method**	**BLSTM [5]**	**Char. OCR [3]**	**Tesseract [19]**
Dataset 1	7.06	9.87	12.03	20.52	30.52	34.41	38.61	34.44
Dataset 2	12.31	18.69	53.6	38.2	40.49	51.32	83.36	57.9

TABLE I. CHARACTER AND WORD ACCURACY FOR DIFFERENT HINDI CORPUS. WE COMPARE OUR RESULTS AGAINST OTHER OCR SYSTEMS

has emerged as a common benchmark data within Indian research community [20]. These pages are from popular Hindi books printed in the last 50 years or so. This dataset is refereed as *Dataset 1*. We also use 1000 pages from the publicly available Digital Library of India. This corpus contains pages that are scanned and stored as binarized images. The pages are considerably degraded compared to Dataset 1. We refer to this dataset as *Dataset 2*. More details can be found in Table II. We also show couple of qualitative examples from the two datasets in Fig. 3. Dataset 2 is degraded more than Dataset 1. This is due to the age of the books, print/paper quality and the process of digitization.

TABLE II. DATASET DETAILS WHICH WERE USED FOR OUR EXPERIMENTS

Corpus Name	No. of Books	No. of Pages	No. of Words
Dataset 1	33	5000	1.5M
Dataset 2	11	1000	300K

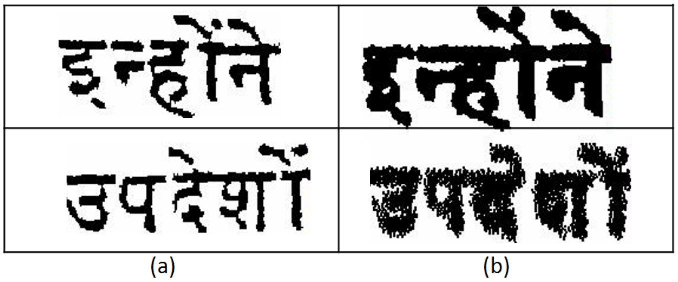

Fig. 3. Example images from the datasets which we use. We show same images from the two datasets to show the difference in quality of images. (a) Examples from Dataset 1 (b) Examples from Dataset 2.

B. Recognition Results

For our experiments, we used 20% of the data for training and tested the results on remaining 80% data. The results are shown in Table I. We show our method is performing considerably better than state-of-the-art methods. We get poorer accuracy for Dataset 2 as the image quality is much poorer when comparing Dataset 1. For Dataset 1, we report an improvement of 3% in character accuracy while comparing with [5]. The major difference between these two modules is the presence of sequence-sequence mapping for Unicode generation. Also, character level accuracy of [3] is better than [19]. However, when we take word accuracy, the positions get reversed. One possible reason could be that for [19], the errors may not be distributed across the words. For Dataset 2, [19] outperforms [3] in character and word level because of better preprocessing techniques. A considerably better post-processing scheme would have contributed to this numbers. However, our method has shown significant efficiency in recognizing words even in Dataset 2.

Our method took approximately 16 hours for training while running on a mid-level desktop PC having 16GB RAM and a 2.3GHz processor. During testing of our method, recognition of a page took 1.57 seconds on average, considering 250 words in a page. One of the first experiments that we conducted was to identify the effect of training size and number of features on final accuracy. We took a subset of 200 pages from Dataset 1 and evaluated them by varying the number of features and amount of training data. Table III shows the results. The accuracy is highest for 20% training data with 14 features. Training size of more than 20% did not improve the accuracies by much and hence we decided to use 20% for training.

TABLE III. EFFECT OF NUMBER OF FEATURES AND TRAINING SIZE ON A DATASET. THE TESTING CHARACTER ERROR RATE IS SHOWN ON TABLE BELOW.

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

C. Discussions

A significant reason for improvement in accuracies after using BLSTM network is its ability to store context, both past as well as future. The image shown in Fig. 5 is a typical example that occurs in Hindi. Fig. 5 (a) and (b) shows two different word containing similar looking segments (highlighted area in words). While the segment shown in (a) consists of two Unicodes, (b) image segment is infact representing a single Unicode. This is because, (b) segment is actually a single glyph but due to degradation or font rendering, it was split into two and could be confused with the segment in (a). Such cases can be recognized only by considering a larger context for recognition. Availability of memory is vital in such cases where past and/or future context can be referred to decide upon the present output. This is shown in Fig. 5(c) where we show a common symbol and the associated Unicode for it. While generating Unicode, we need to place them in the proper order so that the text-processors can render them correctly. Therefore, eventhough symbol 1 (in red) appears first in image, it has to be placed in the end. Such reordering requires the presence of memory so that the future information can be referred to decide the output of present sequence.

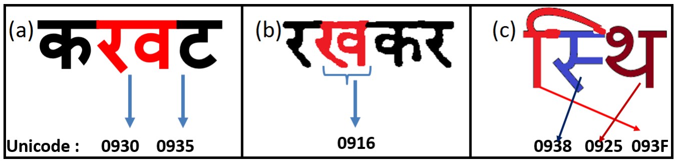

Fig. 5. The above example shows the significance of context. Two different words with same sub-image is shown with (a) representing two Unicodes and (b) representing a single Unicode. The reason being (b) sub-image should have been joined together but due to degradation, they have been separated. Such words can be detected by considering the larger context

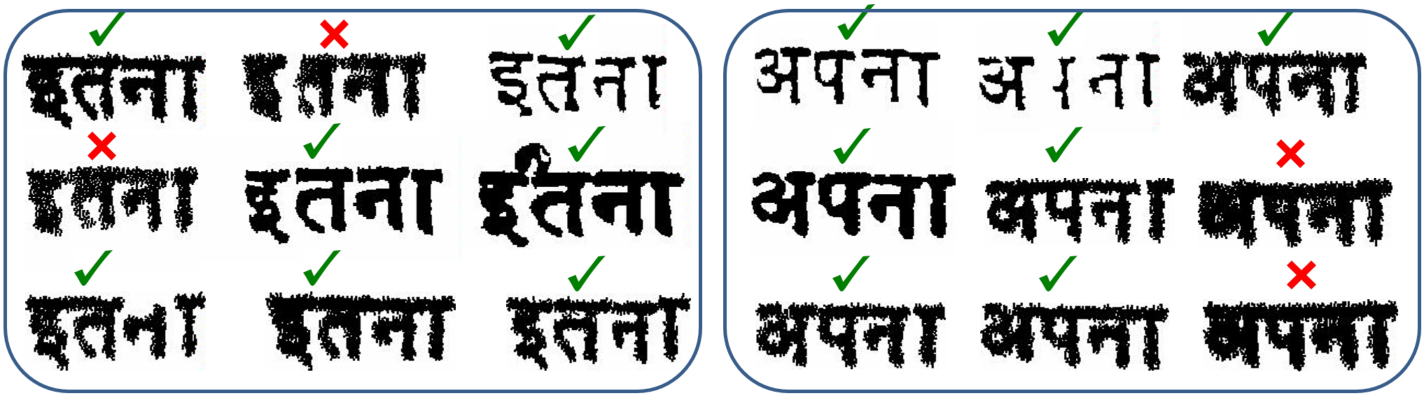

Fig. 4. Success and failure examples of our method is shown. All images present in a box is of same word. Notice the level of degradation and the variation in font present in the database. Our method has failed when the level of degradation is extreme.

Eventhough our method was able to correctly recognize a large number of degraded words, most of our failures are also due to extreme degradations. Dataset 2 consists of books which have high degree of degradations present in them. Another significant reason for errors is the inability of our method in recognizing punctuation marks and other symbols correctly. There is a significant percentage of errors due to this. This is because profile based features are not suited in capturing the information of small components like punctuations. Using better features should be helpful in solving this issue.

V. Conclusion and Future Work

In this paper, we model the problem of OCR as that of transcribing a set of feature vectors to output labels. The need to re-engineer the existing OCR architecture was felt due to the lack of formalism present in sub-word segmentation and latent symbol to Unicode conversion. We consider Unicode as the fundamental recognition unit and use a sequence-sequence transcription module to map the word features to corresponding Unicode. A variant of RNN known as BLSTM was used for this task. The experiments were conducted on two different datasets consisting of more than 6000 pages. Quantitative comparison to the previous methods is reported. In future, we would like to explore the possibility of extending the idea to other Indian languages. Initial results in this direction are very promising. We also would like to try this approach for scripts like Urdu where the script complexity makes the problem very exciting. We believe that better language models and features can further improve the accuracies. However, use of a statistical language model in this framework is not trivial.

Acknowledgment

We would like to thank R. Manmatha and Volkmar Frinken for introducing us to the BLSTM neural network. We thank Alex Graves for providing us with many technical clarifications.

References

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

[2] U. Pal and B. B. Chaudhuri, "Indian Script Character Recognition: A Survey ," in *Pattern Recognition*, 2004.

[3] 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.

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

[5] N. Sankaran and C. V. Jawahar, "Recognition of Printed Devanagari text using BLSTM neural network," in *ICPR*, 2012.

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

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

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

[9] Veena Bansal and R. M. K. Sinha, "Segmentation of touching and fused Devanagari characters," *Pattern Recognition*, vol. 35, no. 4, 2002.

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

[11] Brown, Peter F. and Cocke, John and Pietra, Stephen A. Della and Pietra, Vincent J. Della and Jelinek, Fredrick and Lafferty, John D. and Mercer, Robert L. and Roossin, Paul S., "A Statistical approach to Machine Translation," *Comput. Linguist.*, vol. 16, no. 2, 1990.

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

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

[14] A. Graves, M. Liwicki, S. Fernandez, R. Bertolami, H. Bunke, and J. Schmidhuber, "A Novel Connectionist System for Unconstrained Handwriting Recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 31, no. 5, 2009.

[15] V. Frinken, A. Fischer, R. Manmatha, and H. Bunke, "A Novel Word Spotting Method Based on Recurrent Neural Networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 2, 2012.

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

[17] P. Natarajan, E. MacRostie, and M. Decerbo, "The BBN Byblos Hindi OCR system," in *DRR*, 2005.

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

[19] Tesseract Optical Character Recognition Engine. [Online]. Available: http://code.google.com/p/tesseract-ocr/

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