Towards Deployable OCR Models for Indic Languages

Minesh Mathew[0000−0002−0809−2590], Ajoy Mondal[0000−0002−4808−8860], and C V Jawahar[0000−0001−6767−7057]

CVIT, International Institute of Information Technology, Hyderabad, India
minesh.mathew@gmail.com
{ajoy.mondal,jawahar}@iiit.ac.in

Abstract. The difficulty of reliably extracting characters had delayed the character recognition solutions (or OCRs) in Indian languages. Contemporary research in Indian language text recognition has shifted towards recognizing text in word or line images without requiring sub-word segmentation, leveraging Connectionist Temporal Classification (CTC) for modeling unsegmented sequences. The next challenge is the lack of public data for all these languages. And there is an immediate need to lower the entry barrier for startups or solution providers. With this in mind, (i) we introduce *Mozhi* dataset, a novel public dataset comprising over 1.2 million annotated word images (equivalent to approximately 120 thousand text line images) across 13 languages. (ii) We conduct a comprehensive empirical analysis of various neural network models employing CTC across 13 Indian languages. (iii) We also provide APIs for our OCR models and web-based applications that integrate these APIs to digitize Indic printed documents. We compare our model's performance with popular publicly available OCR tools for end-to-end document image recognition. Our model outperform these OCR engines on 8 out of 13 languages. The code, trained models, and dataset are available at https://cvit.iiit.ac.in/usodi/tdocrmil.php.

Keywords: Printed text · Indic OCR · Indian languages · CRNN · CTC · text recognition · APIs · web-based application.

1 Introduction

Text recognition faces challenges related to language/script, text rendering, and imaging methods. This study concentrates on recognizing printed text in Indian languages, particularly on text recognition alone, assuming cropped word or line images are provided. The 2011 official census of India [1] lists 30 Indian languages with over a million native speakers, 22 of which are recognized as official languages. These languages belong to three language families: *Indo-European*, *Dravidian*, and *Sino-Tibetan*. Our focus is on text recognition in 13 official languages: *Assamese, Bengali, Gujarati, Hindi, Kannada, Malayalam, Manipuri, Marathi, Oriya, Punjabi, Tamil, Telugu, and Urdu*. While some share linguistic similarities, their scripts are distinct, with Devanagari script used in Hindi

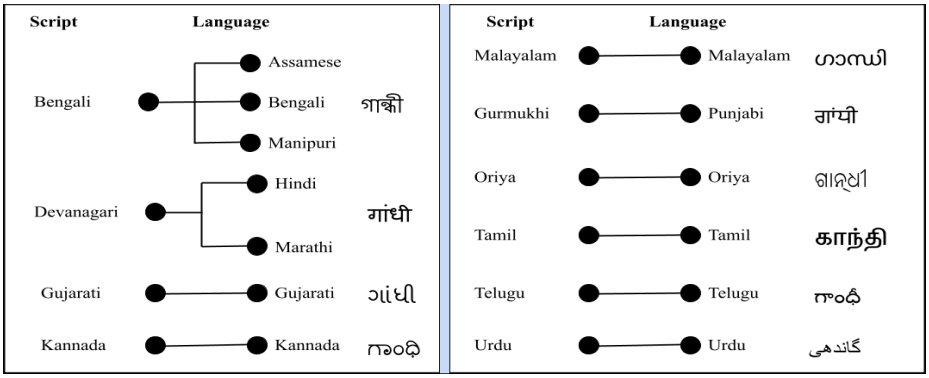

Fig. 1. We explore printed text recognition across 13 Indian languages, covering ten unique scripts. Although many languages share a common alphabet, their scripts vary, with exceptions like Hindi and Marathi. The last column shows the name "Gandhi" in all ten scripts.

Fig. 2. Shows a few sample of cropped images of each of 13 languages from our *Mozhi* dataset.

and Marathi and Bengali script in Bengali, Assamese, and Manipuri, among others. Our study explores printed text recognition across 13 Indian languages, representing ten scripts. Fig. 1 illustrates "Gandhi" written in these ten scripts.

At the same time, Fig. 2 depicts a sample of cropped images from 13 languages from our newly created *Mozhi* dataset. The APIs corresponding to our developed models are integrated into Bhashini[1] for public use. However, we are continuously working on including the remaining low-resource languages — *Bodo, Dogri, Kashmiri, Konkani, Maithili, Nepali, Sanskrit, Santali, and Sindhi* — to cover all twenty-two languages of India.

Efforts to develop OCRs for Indian scripts began in the 1970s but faced challenges in scaling across languages and achieving satisfactory results across diverse document types until recently [29,6,4]. Challenges such as script intricacies, linguistic diversity, and limited annotated data hindered progress in Indian language OCR. The adoption of Connectionist Temporal Classification (CTC), initially successful in speech transcription, revolutionized text recognition across various forms, including handwritten [11], printed [31,26], and scene text [30,28]. Popular open-source OCR tools like *Tesseract* [2], *EasyOCR* [15], and *ocropy* [20] now leverage CTC-based models, enabling recognition of word or line images without sub-word segmentation.

Segmenting words into sub-word units presents a significant challenge for Indian languages compared to English [25]. Developing Indian language recognizers is further complicated by the intricate relationships between script glyphs, language text, and machine representation. In the script, the atomic unit is an isolated symbol (glyph), while in the language, it's an *Akshara* or an orthographic syllable. Machine text representation uses *Unicode* points. An *Akshara* can comprise multiple glyphs, and a sequence of multiple *Unicode* points can represent an *Akshara*. Splitting text at *Aksharas* and mapping them to *Unicode* sequences necessitates language and script knowledge [25,19]. Therefore, adopting CTC-based sequence modeling has become the standard approach for Indian language OCR [25,3,17]. This approach directly maps features from word or line images to target *Unicode* sequences, eliminating the need for explicit alignment during training. Our study offers a comprehensive empirical analysis of various design considerations in developing a CTC-based printed text recognition model for Indian languages.

Our contributions are the following:

- We introduce a new public dataset *Mozhi* for text recognition in 13 Indian languages, comprising cropped line and word segments with corresponding ground truth for all languages except Urdu. With over 1.2 million annotated word images, this dataset is the largest for text recognition in Indian languages (refer Table 1 and Fig. 3).
- We empirically compare the performance of four types of CTC-based text recognition methods across 13 official languages of India, varying in feature extraction and sequence encoding. Additionally, we assess word level and line level recognition models.
- We develop end-to-end page level OCR systems by integrating our best text recognition models with existing line and word segmentation tools. These

[1] `https://bhashini.gov.in/`

systems outperform *Tesseract5* [2] and *Google Cloud Vision OCR* [9] for 8 out of 13 languages (refer Table 4).
– Offer APIs for our OCR models and web-based applications that seamlessly integrate these APIs to digitize Indic printed documents.

2 Related Work

Current OCRs for Indian scripts mainly rely on segmentation-free approaches, which directly produce a label sequence from word or line images. Sankaran *et al.* [26] introduced CTC-based sequence modeling for printed text recognition in Indian languages. Their method utilizes an RNN encoder and CTC transcription to map features extracted from Devanagari word images to class labels. Profile-based features [32] extracted using a 25×1 sliding window are employed. Initially, the model maps *Akshara*s to class labels and uses rule-based mapping to Unicode. In a subsequent work [25], they directly map feature sequences from word images to *Unicode* sequences, eliminating the need for rule-based *Akshara* to *Unicode* mapping.

The introduction of the CTC-based transcription method marked a significant advancement in Indic scripts, particularly by overcoming the challenge of sub-word segmentation. Directly transcribing word images into machine-readable *Unicode* sequences also eliminated the need for language-specific rules to map latent output classes to valid *Unicode* sequences. Krishnan *et al.* [17] utilized profile-based features and a CTC-based model similar to [25] for recognizing seven Indian languages. Their evaluation on a large test set per language demonstrated the effectiveness of a unified framework employing CTC transcription for multilingual text recognition, eliminating the necessity for language or script-specific modules.

Hasan *et al.* [3] proposed an RNN+CTC model for printed Urdu text recognition, directly generating Unicode sequences from text line images. Utilizing a 30×1 sliding window for raw pixel feature extraction, their method yielded promising outcomes. Similarly, our prior work[19] centered on multilingual OCR for 12 Indian languages and English, employing a two-stage system with a script identification module and a recognition module. Chavan *et al.* [7] compared RNN and multidimensional RNN (MDRNN) encoders with CTC transcription. They found the MDRNN encoder outperformed the RNN encoder, using HOG features with the former and raw pixels with the latter. Another study achieved over 99% character/symbol accuracy for Bengali script recognition [22] using an RNN+CTC model. Kundaikar and Pawar [18] explored the robustness of CTC-based Devanagari OCR to font and size variations. At the same time, Dwivedi *et al.* [8] achieved a character/symbol error rate under 3% for Sanskrit recognition using an encoder-decoder model. These findings, particularly the reliance on CTC transcription, motivate our comprehensive empirical study comparing various encoder types and features for both line and word recognition in Indian languages.

3 Mozhi Dataset

To our knowledge, no extensive public datasets are available for printed text recognition in Indian languages. Early studies often utilized datasets with cropped characters or isolated symbols for character classification [24,5]. Later research relied on either internal datasets or large-scale synthetically generated samples for word or line level annotations [26,17,19,7,16,8,18,3]. While recent efforts have introduced public datasets for Hindi and Urdu, they typically contain a limited number of samples intended solely for model evaluation [19,16]. However, due to variations in training data among these studies, comparing methods can be challenging. To address the scarcity of annotated data for training printed text recognition models in Indian languages, we introduce the *Mozhi* dataset. This public dataset encompasses both line and word level annotations for all 13 languages examined in this study. It includes cropped line images, corresponding ground truth text annotations for all languages, and word images and ground truths for all languages except Urdu. With 1.2 million word annotations (approximately 100,000 words per language), it is the largest public dataset of real word images for text recognition in Indian languages. For each language, the line level data is divided randomly into training, validation, and test splits in an 80:10:10 ratio, with words cropped from line images forming corresponding splits for training, validation, and testing. Table 1 shows statistics of *Mozhi*.

Script	**Language**	**Train**		**Validation**		**Test**	
		Lines	**Words**	**Lines**	**Words**	**Lines**	**Words**
Bengali	Assamese	9566	79959	1196	9945	1196	10146
Bengali	Bengali	7579	80113	948	9787	947	10113
Gujarati	Gujarati	8632	79910	1080	10016	1079	10090
Devanagari	Hindi	6525	79762	816	10114	816	10173
Kannada	Kannada						
Malayalam	Malayalam	15112	80146	1889	9893	1889	9980
Bengali	Manipuri	9765	79691	1221	10254	1221	10061
Devanagari	Marathi	8380	80151	1048	10005	1048	9855
Oriya	Oriya	8260	79945	1033	10089	1033	9994
Gurumukhi	Punjabi	6726	79931	841	10036	841	10038
Tamil	Tamil	16074	80022	2010	10021	2009	9974
Telugu	Telugu	12722	80337	1591	9811	1590	9876
Nastaliq	Urdu	9100	-	1138	-	1137	-

Table 1. Statistics for the new *Mozhi* dataset, a public resource for recognizing printed text in cropped words and lines, reveal over 1.2 million annotated words in total. Notably, only cropped lines are annotated for Urdu.

Fig. 3. A few sample of word level images from our *Mozhi* dataset.

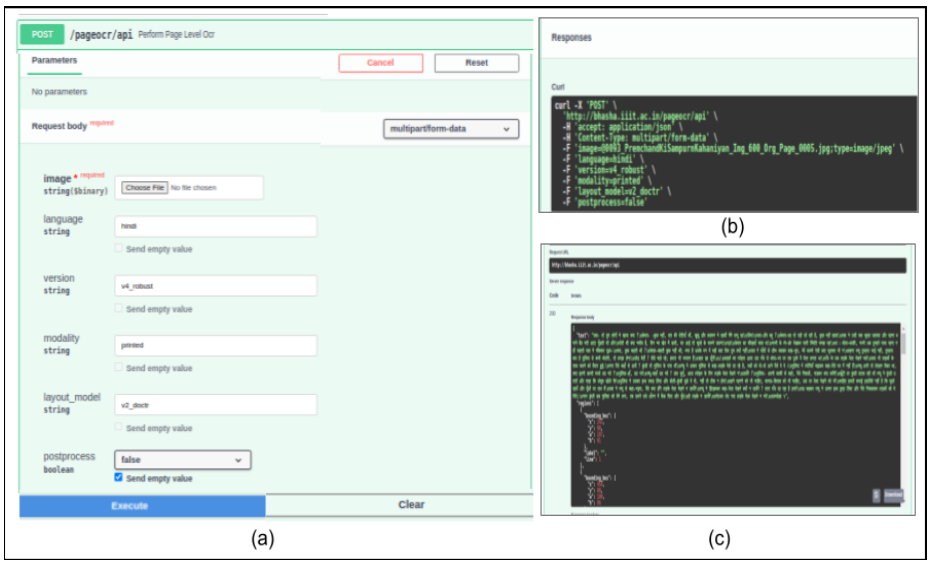

Fig. 4. Shows screen shot of our web-based APIs to digitize Indic printed documents.

4 APIs and Web-based Applications

We develop APIs for page level recognition models across 13 languages and built a web-based application available at `https://ilocr.iiit.ac.in/fastocr/` that integrates these APIs for digitizing printed documents in Indic languages. Fig. 4 illustrates the steps for utilizing our web-based APIs to digitize Indic printed doc-

uments. Users can upload a document image, select the language, OCR model version, layout version, and execute to obtain OCR output.

5 Text Recognition using CTC Transcription

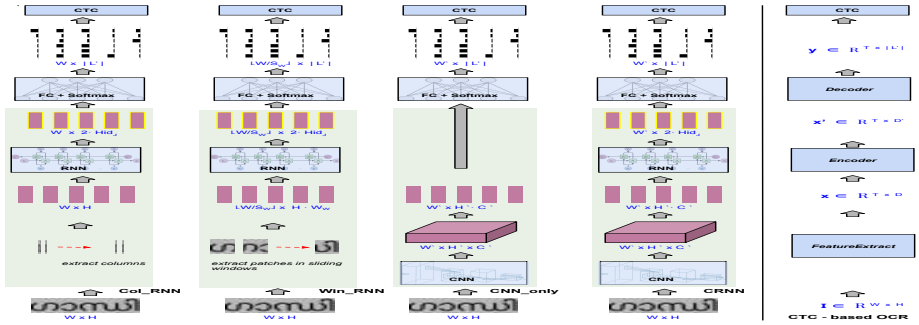

Fig. 5. We examine four CTC-based text recognition methods — Col_RNN, Win_RNN, CNN_only, and CRNN, distinguished by their feature extraction and sequence encoding. W and H represent the width and height of the input image I, respectively. $|L'|$ indicates the number of class labels, including the *blank* label. Hid_j signifies the number of hidden units in the last RNN layer. In the case of Win_RNN, W_W, and S_W denote the width and step size of the sliding window, respectively.

Given an input image I containing a word or a line, text recognition involves converting the text on the image into a machine-readable format. We frame this task as a sequence modeling problem utilizing CTC. The input comprises a sequence of features $\mathbf{x} = x_1, x_2, ..., x_T$, where $x_t \in \mathbb{R}^D$ is extracted from the image I. The output is a sequence of class labels $\mathbf{l} = l_1, l_2, ..., l_N$, where $l_n \in L$ and L represents the output alphabet, i.e., the set of unique class labels. In our scenario, L corresponds to all *Unicode* code points we aim to recognize. We adopt an encoder-decoder interpretation of the CTC framework, as described in [12].

5.1 Extracting Feature Sequence

Graves *et al.* [10] introduced CTC for speech-to-text transcription, employing a sliding window method to extract features from the time axis of the speech signal. They used a window size of 10 milliseconds (ms) and a step size of 5 ms, extracting a fixed-size feature vector termed a time-step or a frame at each instance of the sliding window. However, grey-scale images represent 2D scalar-valued spatial signals in contrast to speech signals. Thus, approaches employing CTC for text transcription from images typically extract features along the horizontal axis of the image [25,3,28]. We follow a methodology similar to that outlined in [25,3,28], where feature vectors in the input sequence $\mathbf{x}$ represent horizontal

segments of the image. Each instance of the input sequence is referred to as a time-step or a frame, consistent with the original approach [10]. The horizontal span of a frame varies depending on the feature extraction method. The feature sequence, $\mathbf{x}$, is extracted in alignment with the script direction. Specifically, for languages other than Urdu, features are extracted from left to right, whereas they are extracted in the opposite direction for Urdu. In summary, given a document image $I \in \mathbb{R}^{W \times H}$ (grey-scale), the feature sequence is obtained as follows:

$$\mathbf{x} \in \mathbb{R}^{T \times D} = FeatureExtract(I). \tag{1}$$

Encoder: The sequence encoder's task is to transform the input sequence $\mathbf{x}$ into an encoded representation $\mathbf{x}' \in \mathbb{R}^{T \times D'}$, where D' represents the encoding size — i.e., the fixed dimensional to which each feature vector is encoded.

$$\mathbf{x}' \in \mathbb{R}^{T \times D'} = Encoder(\mathbf{x}). \tag{2}$$

In this work, we explore several encoder configurations — Col_RNN, Win_RNN, CNN_only, and CRNN for feature extraction[2].

Decoder: The encoded features $\mathbf{x}'$ undergo a linear projection layer followed by Softmax normalization, aligning their size with the number of output classes. This procedure, resembling the decoding phase of CTC as interpreted in [12], extends the original output alphabet L with an extra label for blank, denoted as $\sim$. The blank label signifies instances where no label is assigned to an input. Softmax normalization at each time step yields class conditional probabilities, forming the posterior distribution over the classes. Essentially, given the sequence of encoded features,

$$\mathbf{y} \in \mathbb{R}^{T \times L'} = Decoder(\mathbf{x}'), \tag{3}$$

where each $y_t \in R^{L'}$ represent activations at time step t. Thus y_t^k is a score indicating the probability of k^{th} label at time step t.

We utilize CTC transcription[3] to determine the most likely sequence of class labels given $\mathbf{y}$.

5.2 Training

Let the training dataset be denoted as $S = I_i, \mathbf{l}_i$, where I_i represents a word or line image and $\mathbf{l}_i$ represents its corresponding ground truth labeling. The objective function for training the encoder-decoder neural network for CTC transcription is derived from Maximum Likelihood principles. The aim is to minimize this objective function to maximize the log-likelihoods of the ground truth labeling. Therefore, the objective function utilized is:

$$\mathbb{O} = -\sum_{I_i, \mathbf{l}_i \in S} \log p(\mathbf{l}_i | \mathbf{y}_i), \tag{4}$$

[2] Details of them are presented in the supplementary material.

[3] Additional information regarding CTC transcription can be found in the supplementary material.

where $\mathbf{y}_i$ is the decoder output for the i^{th} sample. The above objective function can be optimized using gradient descent and back-propagation.

5.3 Inference

During inference, the CTC-based classifier aims to output the labeling $\mathbf{l}^*$ with the highest probability, as defined in Eq. (5).

$$p(\mathbf{l}|\mathbf{x}) = \sum_{\pi \in \boldsymbol{\mathcal{B}}^{-1}(\mathbf{l})} p(\pi|\mathbf{x}). \tag{5}$$

6 Experimental Setup

6.1 Implementation Details

In all experiments, cropped word or line images are resized to a height of 32 pixels and converted to grayscale, maintaining the original aspect ratio. To establish a validation split, we randomly select 5% of pages from each book in the train split for all languages. It ensures that the validation split reflects the pages in the train split while the test split comprises pages from different sets of books. In Win_RNN, the sliding window width W_W is set to 20, and the step size W_S is set to 5. For Col_RNN, Win_RNN, and CRNN, we utilize a bi-directional LSTM with 256 hidden units per direction across two layers, resulting in an output size of 2×256 at each time step. The CNN architecture in CNN_only and CRNN follows the original CRNN paper [28]. Our models are implemented using PyTorch [21]. We utilize an existing CRNN implementation [14] for our experiments, conducting training on a single Nvidia GeForce 1080 Ti GPU. Training is set for 30 epochs. Word recognition models have a batch size of 64, while line recognition models use a batch size of 16. RMSProp [13] is employed as the optimizer. Col_RNN and Win_RNN are assigned a learning rate of $10e{-}03$, while CNN_only and CRNN variants converge faster with a lower learning rate of $10e-04$.

6.2 Evaluation

We need to assess text recognition in three scenarios: (i) word OCR: recognizing cropped word images, (ii) line OCR: recognizing cropped line images, and (iii) page OCR: end-to-end text recognition from document images. Our main evaluation metric in all cases is Character Accuracy (CA), determined by the Levenshtein distance between predicted and ground truth strings. For a formal definition of CA, let us denote the predicted text for a word/line/page as l_i and the corresponding ground truth as g_i. If there are N such samples, CA is defined as

$$CA = \frac{\sum_i len(g_i) - \sum_i LD(l_i, g_i)}{\sum_i len(g_i)} \times 100, \tag{6}$$

where len is a function that returns the length of the given string, and LD is a function that computes the Levenshtein distance between the given pair of strings. Note that Character Error Rate (CER), another commonly used metric for OCR evaluation, is essentially $100 - CA$. We also include Sequence Accuracy (SA) alongside CA for word OCR and line OCR. SA represents the percentage of samples where the prediction is entirely correct (i.e., $LD(l_i, g_i) = 0$). In the context of word recognition models, SA is equivalent to 'word accuracy' and is commonly used in scene text recognition literature.

Language	**Word Recognition**							
	Col_RNN		Win_RNN		CNN_only		CRNN	
	CA	SA	CA	SA	CA	SA	CA	SA
Assamese	98.6	95.4	97.6	92.9	98.3	96.0	**99.0**	**96.5**
Bengali	99.1	97.0	98.3	94.5	99.2	97.3	**99.4**	**97.9**
Guajrati	96.2	92.4	95.1	89.5	96.2	90.9	**96.5**	**93.9**
Hindi	97.6	95.1	96.3	92.3	97.4	94.2	**98.2**	**96.3**
Kannada	97.4	88.9	96.4	84.7	96.7	85.8	**97.7**	**90.7**
Malayalam	99.5	96.6	99.3	95.6	98.0	83.7	**99.7**	**97.7**
Manipuri	98.6	95.4	97.8	92.8	98.2	93.1	**99.0**	**96.9**
Marathi	99.0	96.2	98.5	94.2	98.9	95.0	**99.2**	**96.9**
Odia	96.8	93.5	95.7	90.8	96.9	93.7	**97.2**	**94.8**
Punjabi	99.1	97.7	98.4	96.4	99.2	97.8	**99.5**	**98.7**
Tamil	97.9	91.0	97.4	88.4	97.3	87.2	**98.0**	**91.8**
Telugu	96.3	91.4	95.3	86.8	96.4	92.0	**96.8**	**93.6**
Urdu	-	-	-	-	-	-	-	-

Table 2. Results for recognition-only tasks are presented for each language individually on validation set of *Mozhi* dataset. Each model configuration (Col_RNN, Win_RNN, CNN_only, and CRNN) is trained separately for each language. Character Accuracy (CA) and Sequence Accuracy (SA) are reported for word recognition. The highest CA and SA values among the four encoder configurations are highlighted in bold.

We employ a standard OCR evaluation toolkit for page OCR, where the input is a document image. Specifically, we utilize a modern adaptation [27] of the original ISRI Analytic Tools for OCR Evaluation [23]. Using this toolkit, we compute Character Accuracy (CA) and Word Accuracy (WA). CA is calculated following the method described in Eq. (6). Word accuracy is determined by aligning the sequences of words in the prediction l_i with those in the ground truth g_i and identifying the Longest Common Sub-sequence (LCS) between them. For a set of pages,

$$WA = \frac{\sum_i len(LCS(l_i, g_i))}{\sum_i len(g_i)} \times 100 \tag{7}$$

where len returns the number of words in a given sequence of words.

7 Experiments and Results[4]

7.1 Comparing Different Encoder Configurations

We assess the performance of four encoder configurations on the validation set of *Mozhi* dataset for word recognition. Results are presented in Table 2. Each CA and SA pair in the table corresponds to a CTC-based network trained separately for a specific combination of language, recognition unit (word), and encoder configuration (Col_RNN, Win_RNN, CNN_only, and CRNN). Across all cases except for Urdu word recognition, CRNN emerges as the top performer among the four configurations. The superior performance of CRNN over the CNN configuration highlights the necessity of capturing long-term dependencies in word or line images. Unlike fully connected networks, CNN layers have limited receptive fields, necessitating numerous layers to cover the entire input. Our seven-layer CNN lacks the depth to model extensive horizontal dependencies adequately. This deficiency is mitigated by employing a sequence encoder (bi-directional LSTM) that proficiently captures long-term dependencies in both directions.

Language	**Test**			
	Word		Line	
	CA	SA	CA	SA
Assamese	98.9	96.2	99.2	76.8
Bengali	99.0	96.9	98.1	68.4
Gujarati	98.0	94.9	97.4	63.1
Hindi	98.1	95.5	98.8	63.5
Kannada	97.1	88.7	97.5	53.9
Malayalam	99.5	97.3	99.5	87.3
Manipuri	98.4	95.9	99.2	79.4
Marathi	99.3	97.0	99.3	73.8
Oriya	97.5	94.3	98.8	73.1
Punjabi	99.2	98.2	99.3	79.7
Tamil	98.0	91.6	98.3	68.1
Telugu	99.1	95.4	98.9	71.7
Urdu	-	-	93.8	24.2

Table 3. CRNN evaluation on test set of *Mozhi* dataset. For each language, we train both word and line level CRNN models on the respective train split of the *Mozhi* dataset.

7.2 Evaluating CRNN on Test Set of *Mozhi*

Table 2 highlights that among four different models — Col_RNN, Win_RNN, CNN_only, and RCNN, RCNN obtained the best results for all languages on

[4] Additional results can be found in the supplementary material.

Language	End-to-End OCR				GT Detection+CRNN			
	Tesseract		Google		GT word		GT line	
	CA	SA	CA	SA	CA	SA	CA	SA
Assamese	92.7	91.2	90.0	86.0	99.3	97.0	**99.4**	**97.2**
Bengali	93.5	96.2	84.0	91.3	**99.1**	**97.3**	99.0	96.8
Gujarati	96.9	92.4	93.0	**95.2**	**98.0**	93.7	97.7	91.9
Hindi	95.0	93.3	95.2	**97.3**	**98.1**	96.0	98.0	95.6
Kannada	94.9	85.1	85.7	84.6	95.6	**89.2**	**95.9**	86.4
Malayalam	96.2	78.7	88.0	74.8	**99.4**	**98.0**	99.3	97.9
Manipuri	90.9	80.6	85.7	77.4	98.4	94.7	**98.7**	**94.9**
Marathi	97.9	97.4	98.3	**98.4**	**99.6**	98.2	99.5	98.0
Oriya	94.0	83.6	92.6	90.0	**98.6**	**95.4**	98.0	94.5
Punjabi	93.2	89.8	92.7	96.7	99.2	**98.3**	**99.3**	97.9
Tamil	79.3	42.4	92.5	**93.1**	96.1	85.6	*96.5*	85.4
Telugu	93.7	79.3	94.2	89.2	**99.1**	**95.1**	98.9	94.0
Urdu	68.3	26.2	92.7	**85.7**	-	-	**94.7**	81.5

Table 4. Performance of our page OCR pipelines compared to other public OCR tools. In this setting, we evaluate text recognition in an end-to-end manner on the test split of our dataset. Since the focus of this work is on text recognition, for end-to-end settings, for text detection, gold standard word/line bounding boxes are used. Under 'End-to-End OCR' we show results of *Tesseract* [2] and *Google Cloud Vision OCR* [9]. Given a document image, these tools output a transcription of the page along with the bounding boxes of the lines and words detected. Under 'GT Detection+CRNN', we show results of an end-to-end pipeline where gold standard word and line detection are used. For instance, 'GT Word' means we used ground truth (GT) word bounding boxes and the CRNN model trained for recognizing words, for that particular language. Bold value indicates the best result.

validation set of *Mozhi* dataset with respect to CA and SA metrices for word recognition task. Since RCNN, highest performing model for validation set, we evaluated these models on test set of the same dataset. Table 3 presents obtained results for word and line recognition on test set.

7.3 Page Level OCR Evaluation

In page level OCR, the goal is to transcribe the text within a document image by segmenting it into lines or words and then recognizing the text at the word or line level. Our focus lies solely on text recognition, excluding layout analysis and reading order identification. To construct an end-to-end page OCR pipeline, we combine existing text detection methods with our CRNN models for recognition. Transcriptions from individual segments are arranged in the detected reading order. We evaluate the end-to-end pipeline by using gold standard detection to establish an upper bound on our CRNN model's performance. Additionally, we compare our OCR results with two public OCR tools: *Tesseract* and *Google Cloud Vision OCR*. Results from all end-to-end evaluations are summarized in Table 4.

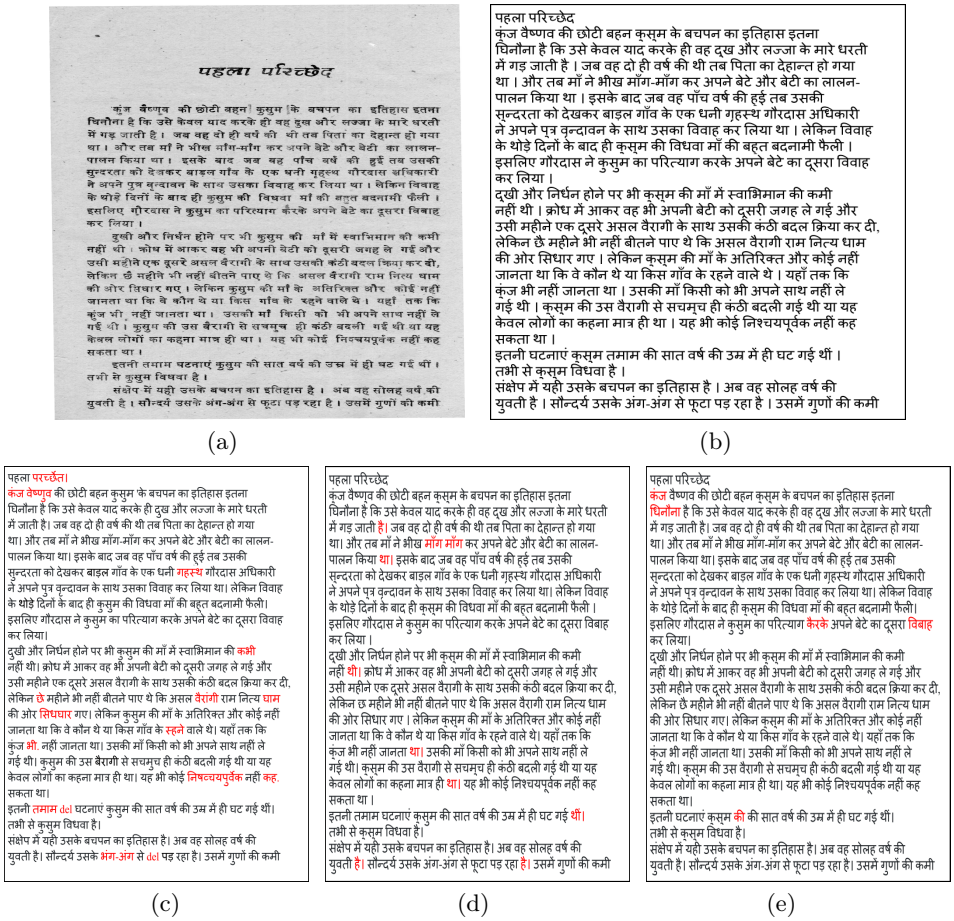

Fig. 6. Displays qualitative results at the page level using *Tesseract*, *Google OCR*, and our method on a Hindi document image. For optimal viewing, zoom in. (a) original document image, (b) ground truth textual transcription, (c) predicted text by *Tesseract*, (d) predicted text by *Google OCR*, and (e) predicted text by our approach.

In Fig. 6, visual results at the page level using *Tesseract*, *Google OCR*, and our approach are depicted. Panel (a) presents the original document image, while panels (b) to (e) display the ground truth and the predicted text by *Tesseract*, *Google OCR*, and our approach, respectively. Wrongly recognized texts are highlighted in red. This figure emphasizes that our approach outperforms existing OCR tools in producing accurate text outputs.

7.4 Use Cases

We leverage our OCR APIs for various significant applications. Notable examples include the pages of the *Punjab Vidhan Sabha*, *Loksabha records*, and *Tel-*

ugu Upanishads. These digitization efforts enable easier access, preservation, and analysis of these valuable texts. The output and effectiveness of our OCR technology in these diverse use cases are illustrated in Fig. 7. These applications showcase the versatility and reliability of our OCR APIs in handling different scripts and document types, ensuring high accuracy and efficiency.

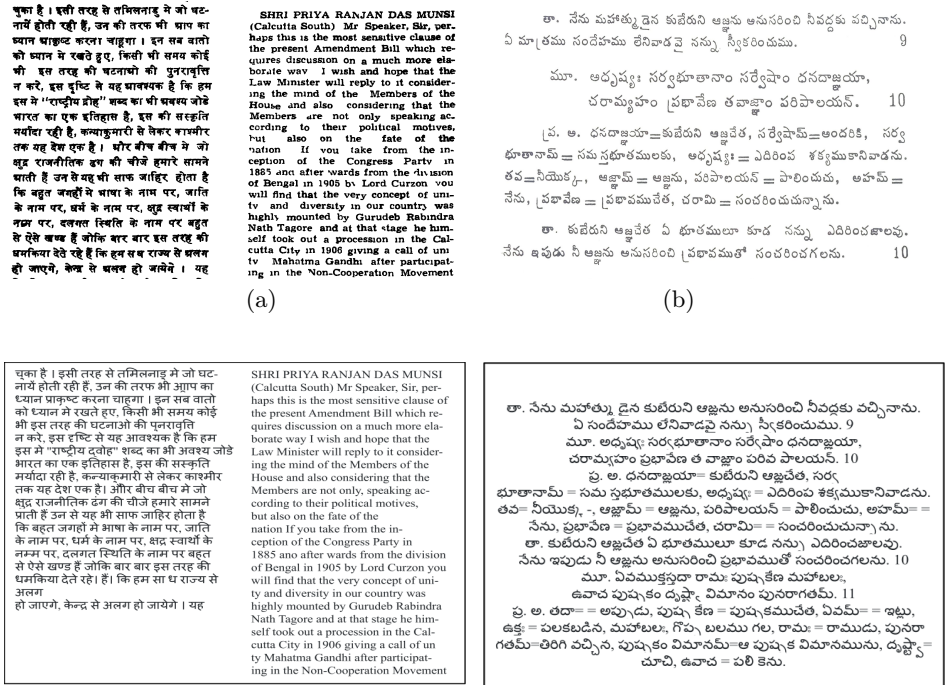

Fig. 7. Illustrates use cases for the digitization of *Loksabha records* and *Telugu Upanishad* pages. (a) and (b) display cropped regions from the original images of *Loksabha* and *Upanishad* documents, respectively. Panels (c) and (d) present the corresponding text outputs generated using our OCR APIs.

7.5 Discussion

Our method performs better in page level recognition than *Tesseract* across all 13 languages, as evidenced by the results in Table 4. Specifically, our approach surpasses *Google* for eight languages, as indicated in the same table when considering ground truth bounding boxes. However, our dataset predominantly comprises pages from books, resulting in limited font, style, layout, and distortion diversity. Nevertheless, this dataset can serve as valuable pre-training data. Moving forward, we aim to enrich the dataset by gathering diverse documents with

varying layouts, content, fonts, styles, and distortions, enhancing its comprehensiveness and utility for developing robust recognition models.

8 Conclusions

We empirically study different CTC-based word and line recognition models in 13 Indian languages. Our study concludes that CRNN, which uses a CNN for feature representation and a dedicated RNN-based sequential encoder, works best. Using existing text detection tools and our recognition models, we build page level OCR pipeline and show that our approach works better than two popular OCR tools for most of the languages. We also introduce a new public *Mozhi* dataset for cropped word/line recognition in 13 Indian languages with more than 1.2 million annotated words. Additionally, we provide APIs for our page level OCR models and web-based applications that integrate these APIs to digitize Indic printed documents. We believe our study, the *Mozhi* dataset, and available APIs will encourage research on OCR of Indian languages.

Acknowledgment

This work is supported by MeitY, Government of India, through the NLTM-Bhashini project.

References

1. Census 2011. `https://censusindia.gov.in/2011-Common/CensusData2011.Html`
2. Tesseract (2021), `https://github.com/tesseract-ocr/tesseract`, accessed on 20 November 2021
3. Adnan Ul-Hasan and Saad Bin Ahmed and Sheikh Faisal Rashid and Faisal Shafait and Thomas M. Breuel: Offline Printed Urdu Nastaleeq Script Recognition with Bidirectional LSTM Networks. In: ICDAR (2013)
4. Arya, D., Patnaik, T., Chaudhury, S., Jawahar, C.V., B.B.Chaudhuri, A.G.Ramakrishna, Bhagvati, C., Lehal, G.S.: Experiences of Integration and Performance Testing of Multilingual OCR for Printed Indian Scripts. In: J-MOCR Workshop,ICDAR (2011)
5. C. V. Jawahar, MNSSK Pavan Kumar and S. S. Ravikiran: A Bilingual OCR system for Hindi-Telugu Documents and its Applications. In: International Conference on Document Analysis and Recognition(ICDAR) (2003)
6. Chaudhuri, B.B., Pal, U.: A complete printed bangla ocr system. Pattern Recognition **31**, 531–549 (1998)
7. Chavan, V., Malage, A., Mehrotra, K., Gupta, M.K.: Printed text recognition using blstm and mdlstm for indian languages. In: ICIIP (2017)
8. Dwivedi, A., Saluja, R., Sarvadevabhatla, R.K.: An ocr for classical indic documents containing arbitrarily long words. In: CVPR Workshops (2020)
9. Google: Google Cloud Vision OCR. `https://cloud.google.com/vision/docs/ocr` (2021), accessed on 10 November 2021

10. Graves, A., Fernández, S., Gomez, F.J., Schmidhuber, J.: Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In: ICML (2006)
11. Graves, A., Liwicki, M., Fernandez, S., Bertolami, R., Bunke, H., Schmidhuber, J.: A Novel Connectionist System for Unconstrained Handwriting Recognition. IEEE Trans. Pattern Anal. Mach. Intell. (2009)
12. Hannun, A.: Sequence modeling with ctc. Distill (2017). `https://doi.org/10.23915/distill.00008`, https://distill.pub/2017/ctc
13. Hinton: Neural Networks for Machine Learning, Lecture 6. `http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf` (2012), accessed on 10 November 2021
14. Holmeyoung: crnn-pytorch. `https://github.com/Holmeyoung/crnn-pytorch` (2019), accessed on 3 February 2021
15. jaidedAI: Easyocr (2022), `https://github.com/JaidedAI/EasyOCR`
16. Jain, M., Mathew, M., Jawahar, C.V.: Unconstrained ocr for urdu using deep cnn-rnn hybrid networks. In: ACPR. p. 6 (2017)
17. Krishnan, P., Sankaran, N., Singh, A.K., Jawahar, C.V.: Towards a robust OCR system for indic scripts. In: DAS (2014)
18. Kundaikar, T., Pawar, J.D.: Multi-font devanagari text recognition using lstm neural networks. In: ICCIGST (2020)
19. Mathew, M., Singh, A.K., Jawahar, C.V.: Multilingual OCR for indic scripts. In: DAS (2016)
20. ocropus: ocropy (2022), `https://github.com/ocropus/ocropy`
21. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: Pytorch: An imperative style, high-performance deep learning library. In: NeurIPS (2019)
22. Paul, D., Chaudhuri, B.B.: A blstm network for printed bengali ocr system with high accuracy. ArXiv **abs/1908.08674** (2019)
23. Rice, S.V., Nartker, T.A.: The isri analytic tools for ocr evaluation version 5.1 (1996)
24. Sanjeev Kunte, R., Sudhaker Samuel, R.: A simple and efficient optical character recognition system for basic symbols in printed kannada text. Sadhana **32**(5) (2007)
25. Sankaran, N., Jawahar, C.V.: Devanagari Text Recognition:A Transcription Based Formulation. In: ICDAR (2013)
26. Sankaran, N., Jawahar, C.: Recognition of Printed Devanagari text using BLSTM neural network. In: ICPR (2012)
27. Santos, E.A.: OCR evaluation tools for the 21st century. In: Workshop on the Use of Computational Methods in the Study of Endangered Languages (2019)
28. Shi, B., Bai, X., Yao, C.: An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition. CoRR (2015)
29. Sinha, R.M.K., Mahabala, H.: Machine recognition of devnagari script. In: IEEE Trans. on SMC (1979)
30. Su, B., Lu, S.: Accurate scene text recognition based on recurrent neural network. In: ACCV (2014)
31. Thomas M. Breuel and Adnan Ul-Hasan and Mayce Ibrahim Ali Al Azawi and Faisal Shafait: High-Performance OCR for Printed English and Fraktur Using LSTM Networks. In: ICDAR (2013)
32. Toni M. Rath and R. Manmatha: Features for Word Spotting in Historical Manuscripts. In: ICDAR (2003)