OPTICAL CHARACTER RECOGNITION (OCR) FOR TELUGU: DATABASE, ALGORITHM AND APPLICATION

Konkimalla Chandra Prakash, Y. M. Srikar, Gayam Trishal, Souraj Mandal, Sumohana S. Channappayya

Indian Institute of Technology Hyderabad, Kandi - 502285, Telangana, India

ABSTRACT

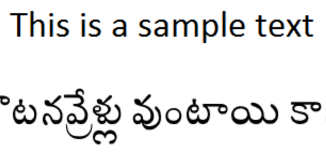
Telugu is a Dravidian language spoken by more than 80 million people worldwide. The optical character recognition (OCR) of the Telugu script has wide ranging applications including education, health-care, administration etc. The beautiful Telugu script however is very different from Germanic scripts like English and German. This makes the use of transfer learning of Germanic OCR solutions to Telugu a non-trivial task. To address the challenge of OCR for Telugu, we make three contributions in this work: (i) a database of Telugu characters, (ii) a deep learning based OCR algorithm, and (iii) a client server solution for the online deployment of the algorithm. For the benefit of the Telugu people and the research community, our code has been made freely available at this *link*.

Index Terms— OCR, Telugu, Convolutional neural network, Deep learning, Document Recognition.

1. INTRODUCTION

Telugu is the official language of the Indian states of Telangana and Andhra Pradesh. It ranks third by the number of native speakers in India, and fifteenth in the Ethnologue list of most-spoken languages worldwide [1]. There are a large number of Telugu character shapes whose components are simple and compound characters from 16 vowels (called *achus*) and 36 consonants (called *hallus*). Optical Character Recognition (OCR) is the mechanical or electronic conversion of images of typed, handwritten or printed text into machine-encoded text. The availability of huge online collections of scanned Telugu documents in conjunction with applications in e-governance and healthcare justifies the necessity for an OCR system, but the complex script and grammar make the problem a challenging one.

OCR for Indian languages is much more challenging than that of Germanic languages because of the huge number of combinations of the main characters, *vattus*, and *guninthas* (modifiers). Unlike Germanic languages, Telugu characters are round in shape, and seldom contain any horizontal or vertical lines. In the English language, character segmentation can be easily done using connected components-like algorithms, as a majority of the characters are formed by a single stroke. In the Telugu script however, parts of the character extend both above and below the main characters and are also not joined to the main character as shown in 1. This makes the use of histogram based segmentation methods (and transfer learning in general) difficult.


This is a sample text

బొటనవ్రేళ్లు వుంటాయి కాబట్టి

Fig. 1: English text vs Telugu text.

The complexity of the problem at hand is huge because of the large number of output classes possible and the inter class variability. The absence of robust deep learning based OCR systems for Telugu has motivated us to build one. An OCR system has a huge impact in real life applications along with a word processor. In the literature, other attempts on Telugu OCR have neither shown results on large datasets [2] nor have considered all possible character and *vattu* combinations which exist in the language. Here, we describe a novel end-to-end approach for Telugu OCR.

Besides data, classifier selection also has a significant impact on an OCR system. Before deep learning was used, feature learning was a critical step in the design of any classifier because feeding raw data would not lead to the targeted results. Therefore, classification is generally performed after the difficult process of appropriate feature selection that distinguishes classes. The advent of Convolutional Neural Networks (CNNs) has paved the way for automated feature learning. Also, the strong generalization capability of this multi-layered network has pushed the classification performance beyond human accuracy. Due to these reasons, we have used a CNN based classifier in our OCR system.

We have addressed these challenges in Telugu OCR and summarize our contributions as follows:

- We introduce the largest dataset for Telugu characters with 17387 categories and 560 samples per category.
- We propose a 2-CNN architecture that performs extremely well on our dataset.
- We have developed an android application for its deployment.

The rest of the paper is structured as follows. Section 2 talks about previous works on OCR of Telugu and Kannada (a similar script). Section 3 briefly describes the methodology and novelties introduced in this paper. Sub-section 3.1 talks about the proposed dataset. Sub-section 3.2 presents the architectural details of the proposed CNN framework and the overall model for classification of characters. Sub-section 3.3 gives an in-depth explanation of the prepossessing steps and segmentation algorithm. We finally present results in Section 4 and offer concluding remarks in Section 5.

2. RELATED WORK

Optical character recognition (OCR) has been one of the most studied problems in pattern recognition. Until recently, feature engineering was the dominant approach that used features like Wavelet features, Gabor features, Circular features, Skeleton features etc; [3] [4] [5] followed by a support vector machine (SVM) or boosting based classifiers. The recent and astounding success of CNNs in feature learning has motivated us to use them for Telugu character recognition.

The first reported work on OCR for Telugu can be dated back to 1977 by Rajasekharan and Deekshatulu [6] which used features that encode the curves that trace a letter, and compare this encoding with a set of predefined templates. It was able to identify 50 primitive features, and proposes a two-stage syntax-aided character recognition system. The first attempt to use neural networks was made by Sukhaswami et al., which trains multiple neural networks, pre-classifies an image based on its aspect ratio and feeds it to the corresponding network [7]. It demonstrated the robustness of a Hopfield network for the purpose of recognition of noisy Telugu characters. Later work on Telugu OCR primarily followed the featurization classification paradigm [8].

The work by Jawahar et al., [2] describes a bilingual Hindi-Telugu OCR for documents containing Hindi and Telugu text. It is based on Principal Components Analysis (PCA) followed by support vector regression. They report an overall accuracy of 96.7% over an independent test set. They perform character level segmentation offline using their data collecting tools. However, they have only considered 330 distinct classes.

The work by Achanta and Hastie [9] on Telugu OCR using convolutional neural networks is also interesting. They used 50 fonts in four styles for training data each image of size 48×48. However, they did not consider all possible outputs (only 457 classes) of CNN. The work by Kunte and Samuel [10] on Kannada OCR employs a twp-stage classification system that is similar to our approach. They have first used wavelets for feature extraction and then two-stage multi-layer perceptrons for the task of classification. They have divided the characters into seperate sub classes but have not considered all possible combinations.

Our proposed approach addresses some of the shortcomings in the literature and is described next.

3. PROPOSED METHODOLOGY

The pipeline followed here is a classic one: skew correction – word segmentation – character segmentation – recognition. Our paper introduces novelties in the dataset and classifier. Our pre-processing and segmentation techniques are minor modifications of existing techniques and are fine-tuned for the Telugu script as described in the following subsections. We describe the dataset next followed by a description of the classifier and the application.

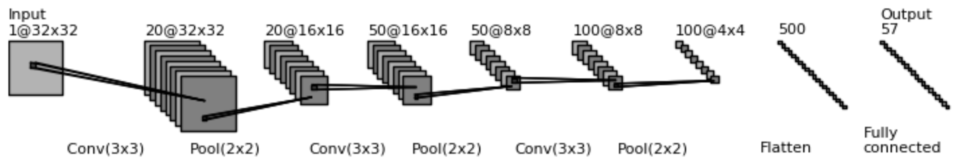
3.1. Dataset

A major issue in the field of Telugu OCR is a lack of large data repositories of Telugu characters that are needed for training deep neural networks. This could be attributed to the fact that most previous methods (as described in the previous Section) did not rely on deep learning techniques for feature extaction.

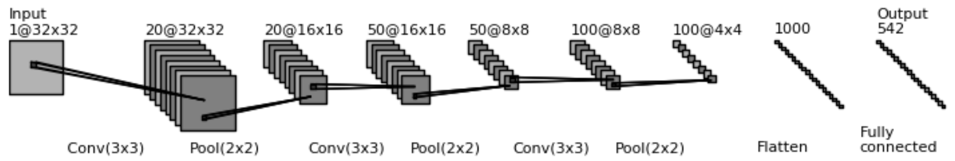
For e.g., the work by Pramod et al., [11] has 1000 words and on an average of 32 images per category. They used the most frequently occuring words in Telugu but were unable to cover all the words in the Telugu language. Later works were based on character level [3] [12] [10]. The dataset by Achanta and Hastie [9] has 460 classes and 160 samples per class which made up 76000 images. However, these works have not considered all the possible combinations of *vattu* and *guninthas*. To tackle this issue, we propose a dataset which takes into consideration all possible combinations of *vattu* and *guninthas*. This is to ensure that the classification algorithms have a good set of training samples which in turn helps improve overall performance.

Each character has been augmented with 20 different fonts downloaded from [13]. Using all the fonts for *gutintham* variants and 3 fonts for *vattu* variants, all possible *vattu* and *gunintham* forms of a character have been manually entered in Microsoft Word. We then changed the font size from 15 to 40 with a step size of 5 covering 6 different font sizes for all the variants of each character. We then took screen-shots of each page containing these characters and used our segmentation algorithm on them to get the individual characters.

We have also introduced random rotations (angle in degrees: -6, -2, 2, 6), additive noise (variance = $0.5 + \frac{J}{10} * \frac{2}{3}$, $J \in (0,5)$) and random crops to simulate realistic conditions. We have then applied elastic deformations on the characters. The dataset has 17387 categories and nearly 560 samples per class. All the images are of size 32×32. There are 6,757,044 training samples, 972,309 validation samples and 1,934,190 test samples which add upto 1 million images (10 GB). Our dataset is novel because unlike other datasets which only take into account the commonly occuring permutations of characters and *vattu*s, we have spanned the entire Telugu alphabets



(a) Architecture 1 for main character.



(b) Architecture 2 for *vattu*.

Fig. 2: Architecture of 1st CNN (main character) and 2nd CNN (*vattu* and *gunintham.*)

and their corresponding *vattu* and *guninthas*.

3.2. Classifier

The performance of an OCR system depends hugely on the performance of its classifier. Previous works [14] on Telugu OCR have done the character level segmentation based on histograms along the x and y directions. Assuming that the histogram method for segmentation would work perfectly, they have used an SVM based classifiers for character classification. However, we have observed that in real scenarios, the histogram method fails to properly segment out the *vattu* and the main character together. It also fails when the characters are rotated or if they share common region when projected on x-axis or y-axis.

Inspired by the success of deep neural networks for feature learning, we have explored CNNs to classify the characters and proposed a new architecture for the same. A CNN is a type of feed-forward neural network or a sequence of multiple layers which is inspired by biological processes. It eliminates the dependency on hand-crafted features and directly learns useful features from the training data itself. It is a combination of both a feature extractor and a classifier and mainly consists of convolutional (weight-sharing), pooling and fully connected layers.

In general, a Telugu character consists of two main components - the main character and the *vattu*/*gunintham* as shown in Figure 3. Using a single CNN would be futile because of the huge number of classes arising from various permutations of the main character, *vattu* and *gunintham*. Therefore, we have used a 2 CNN architecture for classifying the character. The first CNN is used for identifying the main character and the second CNN for identifying the *vattu* and/or *gunintam* present along with the main character. The architectures for both the CNNs is shown in Figure 2.

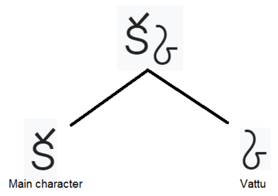


Fig. 3: Main character and *vattu*.

3.3. Pre-Processing and Segmentation

Pre-processing consists of skew correction in which tilt in the image is adjusted and binarization after which the image is binarized and segmented into individual words. This is followed by character level segmentation and classification.

Our segmentation algorithm assumes that there is no skew in the image. This makes skew correction a matter of utmost importance. We have used a straight line Hough transform based technique for correcting skew that can detect and correct skew upto 90 degrees. We used a modified version of Otsu's thresholding [15] for our binarization. We used morphological closing algorithm for noise removal. We then computed the logical OR between the denoised image and that of the Otsu's thresholding result and applied mode based threshold on it.

For application to Telugu characters, we modified the MSER method [16] to take into consideration *dheergas* and *vattu*s. In order to eliminate the possibility of *dheergam* and *vattu* being segmented separately we merged the nearby characters into one word by dilating the output of MSER.

We used the connected components algorithm for character level segmentation. After binarization of the word, we apply the algorithm to separate all the characters as components (groups of binary pixels). In this process, minor blobs are removed from the components. In some cases, *vattu*s are not connected with the main/base character. So, for connecting the base character with its *vattu*, we measured the overlapping distance in horizontal and vertical direction and grouped them together.

3.4. Mobile Application

To facilitate usability, we have developed an Android app that deploys the proposed OCR solution. The app does online image to text conversion with Industry standard (MVP Architecture) which works on any Android (4.4+) device. We used client-server based communication where the client (App user) requests the server with an image and the server responds to the app with a html file. The theme is specifically made keeping in mind that old age or low vision people can use it easily. The App uses camera or gallery for images. This App will also be made publicly available.

4. RESULTS AND DISCUSSION

We now present the experimental details and the results of our proposed algorithm. As presented earlier, a total of 6,757,044 samples were used for training the network. The performance was validated using 972,309 samples. We used a batch size of 500 because of the large training data size. Initially, we trained our network using a SGD + momentum optimizer. Even after 70-80 epochs, the accuracy was not satisfactory (80%). By using the Adam optimizer, we were able to attain much higher accuracy within 30-40 epochs. We halted the training process when there is no increase in validation accuracy for a few epochs (5). Our model was trained on GTX 1060 with 16GB RAM.

We would like to note that in addition to the CNN architectures that have been proposed in Fig. 2, standard CNN architectures defined in Cifar [17] and Lenet [18] were also trained using the same approach described above. This was done primarily for comparative analysis as described next.

Table 1: CNN accuracies for Character Classification.

Network	Architecture	Accuracy
MC Cifar	CRPC32-CRPC32-CRPC64-D360	98.60
MC Lenet	CRPL20-CRPL50-D500	98.62
TCCNN-S	*CRP25-CRP20-DD256*	*97.95*
TCCNN-L	*CRP20-CRP50-CRP100-DD500*	*98.74*

Table 2: CNN accuracies for *Vattu*.

Network	Architecture	Accuracy
MV Cifar	CRPC32-CRPC32-CRPC64-D500	95.46
MV Lenet	CRPL20-CRPL50-D500	95.59
TVCNN-S	*CRP25-CRP20-DD256*	*94.32*
TVCNN-L	*CRP20-CRP50-CRP100-DD1000*	*96.09*

4.1. Table descriptions

Tables 1 and 2 show the accuracy of various CNN architectures on our testing data after the CNN was trained on the proposed dataset. The abbreviations in the tables are explained below.

- CRP (n) - Convolution (3x3, n filters), Relu, Pool(2x2)
- CRPC (n) - Convolution (3x3, n filters), Relu, Pool(3x3)
- D (n) - Dense layer of n nodes.
- DD (n) - Dropout and Dense layer of n nodes.
- TCCNN-L/S - Telugu Character CNN Large/Small
- TVCNN-L/S - Telugu Vattu CNN Large/Small
- MC/MV Cifar - Modified Character/Vattu Cifar
- MC/MV Lenet - Modified Character/Vattu Lenet

The last layers of the Cifar [17] and Lenet [18] architectures have been modified according to the number of outputs of the main character and *vattu*. We have introduced two different architectures, each having two CNNs – one for the main character and one for the *vattu*. TCCNN-S and TVCNN-S are smaller architectures which are faster than the others but with slightly lower accuracy. TCCNN-L and TVCNN-L achieve better accuracy than both the Cifar and Lenet architectures. Even though the improvement is small, it is signficant due to the large size of our dataset. This improvement could be explained by the fact that the proposed architectures in Figs. 2 are tuned for characters and *vattu* individually. Further, the architecture does not reduce the resolution of the image patch as much as the other architectures thereby helping with classification of subtle shapes in the Telugu character set.

We have not used very deep models like VGG [19] and Resnet [20] because they are trained with input images of size 224×224. In our case however, the images are of size 32×32.. We couldn't compare with other works on Telugu OCR because there are very few which have character level segmentation and use a deep learning based approach for classification. The work by Achanta and Hastie [9] is the closest one to ours but has 48×48 images. On the other hand, our images are 32×32. CNN's structure varies with the image size, so such comparison would be futile. Further, their classes also differ. Hence, we have compared with standard CNN architectures on our dataset.

5. CONCLUSION AND FUTURE WORK

We have presented a solution for Telugu OCR that includes a database, algorithm and an application. We have spanned the entire Telugu language while creating the dataset, so there isn't any further possibility of increase in data. The segmentation algorithm can be improved so that every character is segmented together with its *vattu* and *gunintham*. Network accuracy can be further improved to make the classifier better. This proposed work can be further extended to other languages with the scope of having a common OCR system for all the languages of India.

6. REFERENCES

[1] Wikipedia contributors, "Telugu language — wikipedia, the free encyclopedia," 2018, [Online; accessed 13-February-2018].

[2] CV Jawahar, MNSSK Pavan Kumar, and SS Ravi Kiran, "A bilingual ocr for hindi-telugu documents and its applications," in *Document Analysis and Recognition, 2003. Proceedings. Seventh International Conference on*. IEEE, 2003, pp. 408–412.

[3] Arja Rajesh Babu, "Ocr for printed telugu documents," *Diss. Indian Institute of Technology Bombay Mumbai*, 2014.

[4] R Ramanathan, Arun S Nair, L Thaneshwaran, S Ponmathavan, N Valliappan, and KP Soman, "Robust feature extraction technique for optical character recognition," in *Advances in Computing, Control, & Telecommunication Technologies, 2009. ACT'09. International Conference on*. IEEE, 2009, pp. 573–575.

[5] Peifeng Hu, Yannan Zhao, Zehong Yang, and Jiaqin Wang, "Recognition of gray character using gabor filters," in *Information Fusion, 2002. Proceedings of the Fifth International Conference on*. IEEE, 2002, vol. 1, pp. 419–424.

[6] SNS Rajasekaran and BL Deekshatulu, "Recognition of printed telugu characters," *Computer graphics and image processing*, vol. 6, no. 4, pp. 335–360, 1977.

[7] PVS Rao and TM Ajitha, "Telugu script recognition-a feature based approach," in *Document Analysis and Recognition, 1995., Proceedings of the Third International Conference on*. IEEE, 1995, vol. 1, pp. 323–326.

[8] OCR An, "System for telugu," in *Proceedings of the Sixth International Conference on Document Analysis and Recognition*, 2001, p. 1110.

[9] Rakesh Achanta and Trevor Hastie, "Telugu ocr framework using deep learning," *arXiv preprint arXiv:1509.05962*, 2015.

[10] R Sanjeev Kunte and RD Sudhaker Samuel, "An ocr system for printed kannada text using two-stage multi-network classification approach employing wavelet features," in *Conference on Computational Intelligence and Multimedia Applications, 2007. International Conference on*. IEEE, 2007, vol. 2, pp. 349–353.

[11] Pramod Sankar K, CV Jawahar, and Raghavan Manmatha, "Nearest neighbor based collection ocr," in *Proceedings of the 9th IAPR International Workshop on Document Analysis Systems*. ACM, 2010, pp. 207–214.

[12] L Prasanth, V Babu, R Sharma, GV Rao, and M Dinesh, "Elastic matching of online handwritten tamil and telugu scripts using local features," in *Document Analysis and Recognition, 2007. ICDAR 2007. Ninth International Conference on*. IEEE, 2007, vol. 2, pp. 1028–1032.

[13] "http://fonts.siliconandhra.org," .

[14] Rinki Singh and Mandeep Kaur, "Ocr for telugu script using back-propagation based classifier," *International Journal of Information Technology and Knowledge Management*, vol. 2, no. 2, pp. 639–643, 2010.

[15] Nobuyuki Otsu, "A threshold selection method from gray-level histograms," *IEEE transactions on systems, man, and cybernetics*, vol. 9, no. 1, pp. 62–66, 1979.

[16] David Nistér and Henrik Stewénius, "Linear time maximally stable extremal regions," in *European Conference on Computer Vision*. Springer, 2008, pp. 183–196.

[17] Alex Krizhevsky and G Hinton, "Convolutional deep belief networks on cifar-10," *Unpublished manuscript*, vol. 40, pp. 7, 2010.

[18] Yann LeCun et al., "Lenet-5, convolutional neural networks," *URL: http://yann. lecun. com/exdb/lenet*, p. 20, 2015.

[19] Karen Simonyan and Andrew Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.