Assamese Character Recognition using Convolutional Neural Networks

Mihir Yadav[1][0000−0002−3991−0760], Divyansh Mangal[1][0000−0002−7400−1327], Srinivasan Natesan[1][0000−0001−7527−1989], Marcin Paprzycki[2][0000−0002−8069−2152], and Maria Ganzha[3][0000−0001−7714−4844]

[1] Indian Institute of Technology, Guwahati, Assam, India
m.yadav.mihir@gmail.com, natesan@iitg.ac.in
[2] Systems Research Institute Polish Academy of Sciences, Warsaw, Poland
[3] Warsaw University of Technology, Warsaw, Poland

Abstract. In this article, we study recognition of handwritten Assamese alphabet when Convolutional Neural Networks (CNN) are applied. First, we generate a dataset, consisting of the 52 characters, with over 12k images in total. To this dataset we apply LeNet-5, ResNet-50, InceptionV3, and DenseNet-201 models. Transfer learning is used for faster training. Best accuracy of over 94.62% is obtained on the test data. This is, currently, state of the art performance in Assamese character recognition.

Keywords: Character Recognition, Assamese language, Deep Learning, Convolutional Neural Networks, Transfer Learning.

1 Introduction

Optical character recognition (OCR) is the oldest subdomain of pattern recognition. It refers to the conversion of machine-printed, or handwritten, symbols into a machine-encoded text.

A lot of work has been devoted to OCR in Western scripts, and languages like Chinese, Japanese, etc. (see, [11]). Over the past two decades, significant progress has been made also in Indian scripts, like Devanagari [2], Gurumukhi [7], Bengali [1], Tamil, Telugu or Malayalam [9]. However, character recognition for Assamese, the major North-eastern language of India, spoken by over 14 million people, as their native language, is still lacking. Specifically, only a few attempts have been made towards handwritten Assamese Character recognition. Sarma et.al. [10] used template matching, and reached up to 80% accuracy. Bania et al. [3] applied a feed forward back propagation Neural Network (NN). The maximum reported accuracy was 90.34%. In the context of their work, the major limitation of NNs is their inability to capture spatial features in an image. This limitation can be overcome by using the CNN, which eliminated the need for manual feature extraction, as they learn features during training, directly from the images. However, to the best of our knowledge, CNNs haven't been applied on Assamese characters yet. Doing this became the goal of our work.

Since, there is no standard Assamese Character image dataset available on the Internet, we have collected handwritten character samples and generated a dataset containing more than 12k images, which can also facilitate further research in this domain [4]

The rest of the contribution is organized as follows. Section 2 describes dataset generation, preprocessing steps applied, architecture of the developed CNN models, and the relevant deep learning concepts. Section 3 summarizes the obtained experimental results. Section 4 outlines the future work directions.

2 Proposed Methodology

In this section, we outline how CNNs have been applied on the handwritten Assamese characters, to assess their potential in this context.

2.1 Datasets used in experiments

According to the best of of knowledge, there is no "standard" image dataset of Assamese language handwritten characters, available on the Internet. Hence, we decided to generate our own. Specifically, we have created two datasets (DATASET_1 and DATASET_2), which, when combined, contain total of 12,863 images. The combined dataset is almost balanced, as each character appears in almost equal number of images.

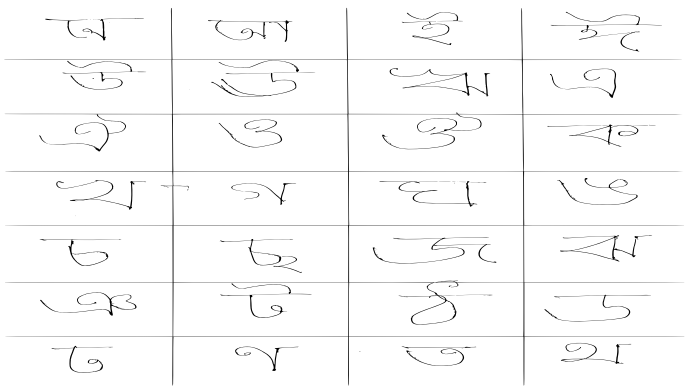

Fig. 1. Sample of a scanned page, from DATASET_1

DATASET_1 has been generated by collecting handwritten samples, each containing the 52 Assamese characters (11 vowels and 41 consonants), provided by a group of over 200 students, faculty and staffs members of IIT Guwahati. This

[4] This dataset is available from: Srinivasan Natesan at natesan@iitg.ac.in

group comprised of persons representing several age groups, education levels, and both genders. Specifically, participants were given 2 sheets, divided into boxes, and were asked to write the characters, in the order in which they appear in the Assamese alphabet. Next, these sheets were scanned (see, Figure 1) and individual characters' images cropped out, preserving the order in which they were written. As a result, a total of 10,994 images have been generated.

DATASET_2 was formed using the Tezpur University – Online Handwritten Assamese Character (TU-OHAC) dataset [5], generated by Baruah and Hazarika [4]. This dataset, collected from over 45 writers, has characters represented as writer's pen movement, on a digitizing tablet. A stroke is defined as a collection of (x, y) coordinates recorded between a single PEN_DOWN and a PEN_UP. To generate the image, all strokes have been plotted. Prior to plotting, preprocessing is performed as described in Section 2.2. In addition to the 52 characters, the TU-OHAC dataset contains extra conjunct consonants, which we ignored to unify both datasets. Hence, DATASET_2 contains 1869 images of 52 characters.

2.2 Dataset preprocessing

Otsu Binarization. Binarization is a very important noise filtering technique applied during the preprocessing step in OCR related problems. Here, all pixels, classified as background are colored white and the remaining ones are colored black. Since images in DATASET_2 are obtained by plotting graphs (not photographed or scanned) binarization is not needed there. Binarization methods can be divided into 3 categories: threshold-based, clustering-based and hybrid [13]. Threshold methods are further divided into global threshold and locally adaptive. Otsu Binarization is a global threshold method. Here, instead of arbitrarily deciding the value of the threshold for the entire image, an optimal value of threshold is calculated. Before performing Otsu Binarization, image is converted to gray-scale. Given the histogram $P(i)$ of the input image, this technique evaluates an optimal threshold t, to be used to segmented pixels into two clusters. Process iterates over possible threshold values and finds the one that minimizes the weighted sum of the intra-class variance of the two clusters $(\sigma_w^2(t))$, which is calculated as follows:

$$\sigma_w^2(t) = q_1(t)\sigma_1^2(t) + q_2(t)\sigma_2^2(t), \tag{1}$$

where

$$\begin{aligned} q_1(t) &= \sum_{i=1}^{t} P(i), & q_2(t) &= \sum_{i=t+1}^{255} P(i), \\ \mu_1(t) &= \sum_{i=1}^{t} \frac{iP(i)}{q_1(t)}, & \mu_2(t) &= \sum_{i=t+1}^{255} \frac{iP(i)}{q_2(t)} \\ \sigma_1^2(t) &= \sum_{i=1}^{t} [i - \mu_1(t)]^2 \frac{P(i)}{q_1(t)}, & \sigma_2^2(t) &= \sum_{i=t+1}^{255} [i - \mu_2(t)]^2 \frac{P(i)}{q_2(t)}. \end{aligned}$$

[5] Available from the UCI Machine Learning repository [5]

Here μ_1 and μ_2 denote the means, while σ_1^2 and σ_2^2 are the variances within the clusters. To perform Otsu Binarization, we have used the OpenCV library function cv.threshold and passed cv.THRESH_OTSU as the flag.

Outlier Elimination. While writing/drawing the characters, writers mistakenly scribbled some random dots near the corners of the boxes. To remove them, we employed a simple technique, taking motivation from the statistical interquartile range method. We evaluate the median x and y coordinates and remove the farthest 2% of the points, measured from this median. Then, we crop out the desired area, which varies from the minimum to the maximum x and y coordinates, of the leftover 98% of the points.

Normalization. Writers were asked to write the characters inside a square box. Since different people write in different orientations, the relative positions of the characters were different. Furthermore, font-size of the characters written by different persons also varied significantly. Hence, the values of x and y were normalized, to ensure that the character's plot occupies the complete area, and is not skewed in a particular direction. The latter point will be addressed in the future research, to bring more flexibility to the trained models.

For DATASET_2, the (x, y) coordinates are immediately available. For images in DATASET_1, coordinates were extracted using thresholding. Next, the maximum and minimum values of x and y coordinates were calculated and used to re-scale all coordinate values to the interval [0,M]. The following expression describes this process:

$$x(i) = \left(\frac{x(i) - x_{\min}}{x_{\max} - x_{\min}}\right) * M, \quad y(i) = \left(\frac{y(i) - y_{\min}}{y_{\max} - y_{\min}}\right) * M, \quad \forall 1 \leq i \leq N. \quad (2)$$

Here N, M are the number of points and size of final image, respectively.

Image smoothing. DATASET_1 is developed by scanning the characters written on paper. However, each image in DATASET_2 results from plotting a finite number of points (300 to 500). Here, the curve obtained by joining these points may fail to depict the actual shape, resulting in sharp edges and irregularities in the plotted curve. Therefore, we applied smoothing to images in DATASET_2. Specifically, a 3-point moving average filter method, was used to smooth individual strokes, for $1 < i < N$:

$$x(i) = \frac{x(i-1) + x(i) + x(i+1)}{3}, \quad y(i) = \frac{y(i-1) + y(i) + y(i+1)}{3} \quad (3)$$

where N is the number of points and we iterate over i in an increasing order.

2.3 Generating final image dataset

Preprocessing is done directly on the images of DATASET_1 and on the (x, y) coordinates of the text files in DATASET_2. With the help of a Python script, we shall plot these coordinates (using Matplotlib) to obtain the character images for DATASET_2. The axes and labels are removed and the figure is resized to 96×96 pixels. The process is summarized in Figure 2.

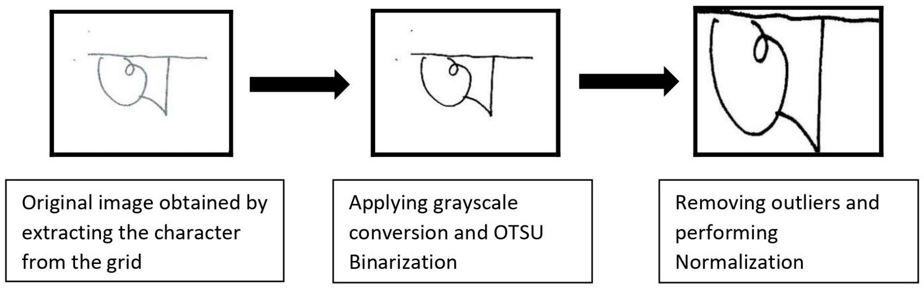
Original image obtained by extracting the character from the grid
Applying grayscale conversion and OTSU Binarization
Removing outliers and performing Normalization

Fig. 2. Preprocessing applied step-by-step on the character label 'A' (DATASET_1)

2.4 CNN Model Architecture

Here, architecture refers to the number of layers, size of the filters, and types of activation and pooling function used. In this work we have experimented with four CNN models: LeNet-5, ResNet-50, InceptionV3 and DenseNet-201, and analysed their performance on the, above described, dataset. The underlying learning rule was the same in all cases, i.e. backpropagation using gradient descent. The activation function was ReLu (Rectified Linear Unit) and the pooling strategy was MaxPooling. Let us now describe in more details each CNN architecture.

LeNet-5 Default LeNet 5 contains 2 sets of convolutional layers, followed by pooling layers [8]. We increased this count to four. Number of filters, used in each layer, was also increased. Finally, we added a dropout layer, and a fully connected layer, which uses Softmax as the activation function. It generates output between 0 and 1, the class with maximum value of the output is selected as the predicted class.

ResNet-50 The main hindrance of training very deep neural networks was the vanishing gradient problem. This may be conceptualized as the rapid diminishing of the gradient of the loss function, during back propagation, due to successive multiplication. To overcome this issue, ResNet employs skip connection, which adds the output from the earlier layers to later layers, thus allowing a shortcut path for the gradient to follow. ResNet-50 comprises of 5 stages, each with a convolution and identity block. Convolution and identity blocks have 3 convolutional layers each.

InceptionV3 Original incarnation (InceptionV1) of this architecture was called GoogleNet. The central idea behind GoogleNet was the inception module, which performs convolution on an input with multiple sizes of filters, concatenating the resulting outputs, and sending them to the next layer. This network consists of such modules "stacked upon each other", with occasional max-pooling layers with stride 2, to halve the resolution of the grid. InceptionV3 architecture has several innovations over the initial GoogleNet, like convolution factorization, auxiliary classifiers and grid size reduction (see [12] for further details).

DenseNet-201 This is a very recent architecture, introduced in 2018. It connects each layer to every other layer, in a feed forward fashion. For each layer, feature maps of all preceding layers are used as inputs. It concatenates them, instead of summing them up. They alleviate the vanishing gradient problem, and strengthen feature propagation. Through feature reuse, they are easy to train and highly parameter efficient. Their overall structure contains "dense blocks" connected by convolutional and pooling layers (see [6]).

The total number of trainable parameters of ResNet, DenseNet and InceptionV3 models is of the order of a million. The complexity of their architecture makes it difficult to make changes to the network. Scaling up may cause large parts of computational gains to be lost. Therefore, we have decided to not to tweak their internal structure, as we did for Lenet-5 and use the default implementation that Keras provides. However, this is also one of interesting research directions that we plan to explore in the future.

2.5 Transfer Learning

Training a Deep CNN, like ResNet, with random initialization of model weights, is highly computationally expensive. It may even be practically infeasible, due to the limited size of our dataset. So, in such cases, it is suggested that the concept of transfer learning may be applied. Here, one uses a pre-developed model as a starting point, for another task, to improve the performance of the second task.

Typically, transfer learning is used by two approaches, develop model approach and pre-trained approach [2]. Pre-trained approach has two main methods, namely, fine tuning and fixed feature extractor. In the latter, all the layers of the pre-trained model are frozen except the last fully connected layer. In this context, we have decided to use fine tuning pre-trained approach, wherein instead of random initialization, weights are initialized with a network pre-trained on the ImageNet dataset, comprising of 1.2 million images belonging to over 1000 different categories. Thus, some fundamental features are already learned and weights are adjusted. We now feed our dataset to the entire model (no layer is freezed here) and start training it further. Note, however, that this is also one of points worthy further exploration.

3 Experimental results and analysis

The models were implemented using Python frameworks Tensorflow and Keras. They were trained on 70% of data, validated on 20%, and then tested on the remaining 10% of the dataset. The split was done in a balanced way, ensuring that the number of instances of each character image is proportionally the same in each part. The number of epochs was set to 100. The checkpoint callback has been used while training, which after each epoch, checks the validation accuracy of the model, and saves the weights corresponding to the model that gives the highest validation accuracy, into an external hdf5 file. This highest validation accuracy is displayed in Table 1. After the training is complete, the model is saved in the external hdf5 file, and is run on the test data and the test accuracy is evaluated by applying it to the selected 10% of the original dataset.

All models except LeNet used transfer learning for weight initialization. The models have been run on several combinations of hyperparameters, namely, learning rate and batch size and, the best performing combination was selected for each model. Observed in all cases stability of results, with the increasing number of epochs shows that models have reached saturation and can be reasonably expected that they are at their "peak performance".

Table 1 summarizes the results obtained by various architectures. As noted, these results are the best results obtained when running experiments. Figure 3 shows the obtained validation accuracy curves. Overall, DenseNet shows the best performance in terms of accuracy. This was expected, as DenseNet is the most recent architecture that incorporates all the beneficial aspects of ResNet and GoogleNet, while introducing some additional useful concepts, like feature reuse. Based on analysis of literature we can state that the 94.62% accuracy can be seen as state of the art in Assamese character recognition.

Table 1. Validation and test accuracies

Architecture	**Learning Rate**	**Batch Size**	**Val acc(%)**	**Test acc(%)**
LeNet 5	0.0008	48	88.82	86.25
ResNet 50	0.0008	128	94.88	93.55
InceptionV3	0.001	48	94.73	94.09
DenseNet 201	0.001	48	95.37	**94.62**

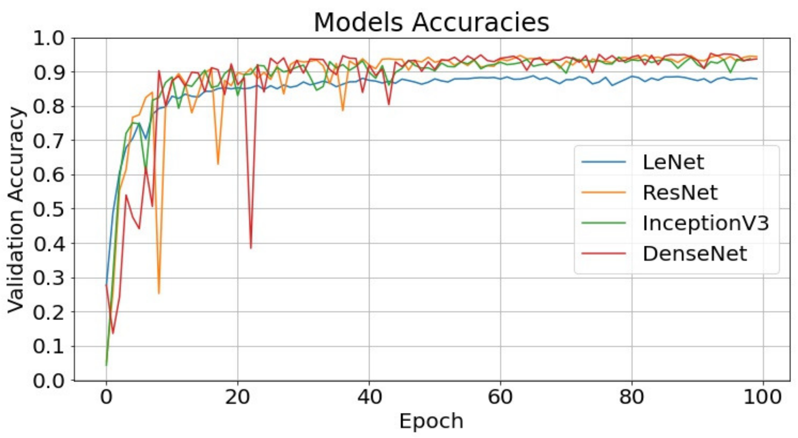

Fig. 3. Validation Accuracy plot

4 Concluding remarks

The aim of this contribution was to present preliminary results of our research into recognition of handwritten Assamese characters. Specifically, based on literature survey, we have applied four different CNNs to the problem. Before doing so, we had to create a dataset to train our models, as no Assamese character dataset could be found online. The best obtained result, ∼94% accuracy, can be seen as the benchmark result for the future improvements. We have also indicated research areas that we plan to pursue in the future.

References

1. M. Al Rabbani Alif, S. Ahmed and M. A. Hasan, ”Isolated Bangla handwritten character recognition with convolutional neural network,” International Conference on Computer and Information Technology, Dhaka, Bangladesh,pp. 1-6, 2017; DOI: 10.1109/ICCITECHN.2017.8281823.
2. N. Aneja and S. Aneja, ”Transfer Learning using CNN for Handwritten Devanagari Character Recognition,” International Conference on Advances in Information Technology, Chikmagalur, India, pp. 293-296, 2019
3. R. K. Bania and R. Khan. ”Handwritten Assamese Character Recognition using Texture and Diagonal Orientation features with Artificial Neural Network”. International Journal of Applied Engineering Research. 13, 2018; DOI: 10.1007/978-981-15-0339-9_11.
4. U. Baruah and S. M. Hazarika, ”A Dataset of Online Handwritten Assamese Characters,” Journal of Information Processing Systems, vol. 11, no., pp.325-341, 2015; DOI: 10.3745/JIPS.02.0008.
5. D. Dua and C. Graff, UCI Machine Learning Repository Irvine, CA: University of California, School of Information and Computer Science, 2019.
6. G. Huang, Z. Liu, L. Van Der Maaten and K. Q. Weinberger, ”Densely Connected Convolutional Networks,” CVPR , Honolulu, HI, USA, pp. 2261-2269, 2017
7. U. Jindal, S. Gupta, V. Jain and M. Paprzycki ”Offline Handwritten Gurumukhi Character Recognition System Using Deep Learning”,Advances in Intelligent Systems and Computing, vol 1064., 2020.
8. Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, ”Gradient-based learning applied to document recognition,” in Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324, Nov. 1998; DOI: 10.1109/5.726791.
9. P. P. Nair, A. James and C. Saravanan, ”Malayalam handwritten character recognition using convolutional neural network,” International Conference on. Inventive Communication and Computational Technologies, Coimbatore, 2017.
10. P. Sarma, C. K. Chourasia and M. Barman, ”Handwritten Assamese Character Recognition,” IEEE Conference for Intelligent Technologies, Bombay, India, 2019; DOI: 10.1109/ICAIT47043.2019.8987286.
11. H. Singh, R.K. Sharma and V.P Singh, ”Online handwriting recognition systems for Indic and non-Indic scripts: a review” Artif Intell Rev 54, 1525–1579, 2021.
12. C. Szegedy, V. Vanhoucke, Sergey Ioffe, J. Shlens and Z. Wojna, ”Rethinking the Inception Architecture for Computer Vision”, Conference on Computer Vision and Pattern Recognition (CVPR) 2016.
13. J. Wen, S. Li, J. Sun, ”A new binarization method for non-uniform illuminated document images, Pattern Recognition” 10.1016/j.patcog.2012.11.027, 2013, Volume 46, Issue 6, pp. 1670-1690,