AKHCRNet: Bengali Handwritten Character Recognition Using Deep Learning

Akash Roy

July 28, 2020

AKHCRNet: Bengali handwritten character recognition using deep learning

Roy, Akash

Government College of Engineering and Ceramic Technology
royakashappleid@icloud.com

Abstract. I propose a state of the art deep neural architectural solution for handwritten character recognition for Bengali alphabets, compound alphabets as well as numerical digits that achieves state-of-the-art accuracy 96.8% in just 11 epochs. Similar work has been done before by Chatterjee, Dutta, et al. 2019[1] but they achieved 96.12% accuracy in about 47 epochs. The deep neural architecture used in that paper was fairly large considering the inclusion of the weights of the ResNet 50 model which is a 50 layer Residual Network. This proposed model achieves higher accuracy as compared to any previous work & in a little number of epochs. ResNet50 is a good model trained on the ImageNet dataset, but I propose an HCR network that is trained from the scratch on Bengali characters without the "Ensemble Learning" that can outperform previous architectures.

Keywords: Deep Learning · Image Recognition · Convolutional Networks · Handwritten character recognition

1 Introduction

1.1 Motivation

Bengali is one of the official languages of the Republic of India and the official language of Bangladesh. About 230 million people speak and write Bengali as a native language around the world. So recognition of handwritten Bengali character is an important problem needed to be solved. Although there is a substantial amount of work that has been done on this topic as detailed in the previously done section improvements still can be achieved as the benchmark has not yet reached human-level accuracy or surpassed that, using recent advancements both in the usage of convolution kernels in Deep Learning and also in model training procedures (like hyper-parameter tuning, data augmentation, transfer learning, etc.). Although scientists who most recently worked on this topic used the transfer learning from the ResNet50 model achieved 96.12% accuracy, I used a sophisticated architecture and trained from scratch, the BanglaLekha-Isolated Dataset as described in datasets section was used for evaluation mainly because it has a large sample size and large variance and also because the output classes are balanced. The dataset has a total of 84 characters, which has 10 numerals, 50 basic characters, and 24 most common conjunct-consonants.

1.2 Previous works

Several other people laid the foundation for Bengali handwritten character recognition, in those work Ray and Chatterjee did the first work. After that, many more researchers did other methods, improving the performance of Bangla Handwritten Character Recognition (HCR) like Hasant et al.[2] proposed an HCR capable of classifying both printed and handwritten characters by using Discrete Cosine Transform method over the input image and Hidden Markov Model (HMM) for character classification. Wen et al.[3] proposed a Bangla numerals recognition method using Principal Component Analysis and Support Vector Machines, but those are using traditional machine learning methods. In Hassan and Khan's work, they used the KNN algorithm where features were extracted using local binary patterns. Das et al.[4] proposed a feature set representation for Bangla handwritten character recognition. It was done by hand engineering 8 distance features, 24 shadow features, 84 quad-tree based longest run features, and 16 centroid features. Their accuracy was about 85.40% on a 50 character class dataset as hand engineering was difficult for the Conjunct-Consonants in Bengali characters. Rumman Rashid Chowdhury used CNN based classifier[10] and gained over 98% accuracy on 10 neumericals and 91.12% accuracy on 50 basic Bengali character. In the mentioned methods, the main concern is that they used many handcrafted features extracted for the small dataset, or used didn't achieved human level accuracy. So these methods are in no means achieved the human level accuracy in handwritten character recognition. As using Convolutional Nets eliminated the need for feature extraction. It automated the process. So I'll be using that to make the model architecture.

2 Dataset

The dataset I'm gonna be using for this problem is the BanglaLekha Isolated Dataset[11], made with the help of male and female native Bengali speakers of different states of Bangladesh. Their age ranging from 4-27. To remove the bias, this dataset also introduces handwritten data from the people of special ability.

So the dataset is not biased to certain age groups or gender or people of special abilities. The total data available for the training is the following.

But the data has some blank images or images that are not taken properly or mislabeled data due to human error or other that I don't about. So I have to manually go through every directory of 84 classes and remove from the dataset. This process is much important as I am using an algorithm to make a validation dataset (Data the neural network hasn't seen yet) taken at random from the directory. If any of those mislabeled or blank images get into the validation set the neural network will never know and this will generate an error that can be avoided. But for the training dataset, if 1-2 images get blank or mislabeled error, it doesn't matter much. But due to sheer amount of data and unavailability of man power I only managed to remove mislabeled categories of first 20 categories.

Dataset preparation goes like this: Out of 166K (after cleaning mislabeled and blank images) I've used 28% of data randomly chosen from the 84 classes, so no human bias for selection of validation data is present and the rest of the data is used for training.

This is how the dataset is considered:

Dataset	Total Images Used	Percentage of total data
Training	118394	72%
Validation	45983	28%

So the validation data is selected by computer at random before the training and the neural network doesn't see the validation data during training, that way we can safely measure the accuracy of the network. The accuracy measuring algorithm used here is described below in the architecture section. In the next section it's shown how data is processed before going into the network.

3 Image data pre-processing

The very recent work that has been done by Kinshuk Chatterjee et. al 2019[1] achieved the best result yet for a training of 47 epochs at 96.12% and the 97% accuracy achieved by Saha and Saha et. al used ensemble CNN architecture run for 500 epochs. Individually, their accuracy was 95.67% and 92.43%.

Let's look deeper at what K Chatterjee et. al did. They used ResNet50 model, which is pre-trained on the ImageNet dataset. For a hand-written classification task I do think that they used transfer learning from a model that is specifically NOT trained to classify handwritten characters. I think that ResNet 50 model's CNN learnt to detect several features that is not quite useful for this task of classifying hand written characters. If I'd to use a model trained previously on MNIST dataset[7] for example, I think it'd converge much faster.

Secondly they used images of size sometimes height of 200 px, width of 200 px, color depth of 3 to comply with ResNet, other times 128 by 128, but in general these are huge. I got rid of the unnecessary color channels. I used grayscale image as an input shape because fundamentally you wouldn't care whether your bengali handwritten character is colorful or not in order to classify them. This made less information but the 'same' useful information going into the network, so the training will converge sooner than that of K Chatterjee et.al's training duration that is, 47 epochs.

Next step of pre-processing is to decrease the amount of information coming in neural network. I propose instead of using 200 pixels for width and height that is 40,000 parameters we should be using 32 pixels for width and height that is 1024 parameters. Let's have a look in the data and if those image resizing actually hurt the data and make them unrecognizable? Each pixel value of the images is also normalized between 0-1, and this makes the neural network easier to train.

I'm using the idea of bi-linear interpolation technique when resizing the input images from the original size, that goes like this:

When we are given an image suppose that we want to find the value of the unknown function f at a specific point in image (x, y). It is assumed that we know the value of function f at the four points being

$$Q_{11} = (x_1, y_1),$$

$$Q_{12} = (x_1, y_2),$$

$$Q_{21} = (x_2, y_1),$$

$$Q_{22} = (x_2, y_2).$$

We first do linear interpolation in the x-direction. This gives us the following:

$$f(x, y_1) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{21}) \tag{1}$$

$$f(x, y_2) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{12}) + \frac{x - x_1}{x_2 - x_1} f(Q_{22}) \tag{2}$$

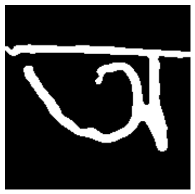
Then to the y direction:

$$f(x, y) \approx \frac{y_2 - y}{y_2 - y_1} f(x, y_1) + \frac{y - y_1}{y_2 - y_1} f(x, y_2) \tag{3}$$

$$= \frac{y_2 - y}{y_2 - y_1} \left(\frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{21}) \right) + \frac{y - y_1}{y_2 - y_1} \left(\frac{x_2 - x}{x_2 - x_1} f(Q_{12}) + \frac{x - x_1}{x_2 - x_1} f(Q_{22}) \right)$$

$$= \frac{1}{(x_2 - x_1)(y_2 - y_1)} \begin{bmatrix} x_2 - x & x - x_1 \end{bmatrix} \begin{bmatrix} f(Q_{11}) & f(Q_{12}) \\ f(Q_{21}) & f(Q_{22}) \end{bmatrix} \begin{bmatrix} y_2 - y \\ y - y_1 \end{bmatrix}$$

Using this bi-linear interpolation technique for image resizing you can see the result here:


(a) Image Size (171, 170)

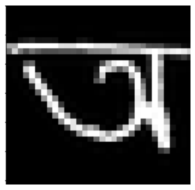

(b) Image Size (32, 32)

Fig. 1: Class 1 Image before resize and after

So we can clearly see that our images are still recognizable and so we can get away with 1024 pixels, which will not going to hurt the deep learning model for less data in an image. Let's see more images resized to (32, 32, 1) and how those looks:

(a) Image Size (245, 227) (b) Image Size (32, 32)

Fig. 2: Class 80 image resized before & after

4 Architecture

The data is coming into the network as (32, 32) gray-scale image, so I have added a (32, 32, 1) layer as the input layer. For the next two layers I used the (5, 5) convolutions and 32 filters of those, this combination worked great against a single (3, 3) convolution. These convolution layers have "same" padding as we have very little information on the images and we don't want to lose those during convolutions. Throughout the entire deep neural architecture, the ReLU activation is used. Where

$$ReLU(z) = max(0, z)$$

Next, a Batch-Normalization is used, this makes our model's weights of deeper layers more robust to changes than to the first few layers. Next, I've added a Max Pooling Layer.

I've added a block of (1, 1) convolution with 128 filters followed by one (3, 3), one (5, 5) convolution with 128 filters separately and one (1, 1) convolution and one "same" padded max-pooling layer followed by a (3, 3) convolution. All convolution layers mentioned here have "same" padding. Without the inclusion of this block the architecture converged to a 96.38% accuracy in 20 epochs, which still has the better validation accuracy than that of previous state of the art architecture, but with the inclusion of this block validation accuracy peaked 96.80% in just 11 epochs.

The total of 4 outputs from the previous block has been concatenated into a (16, 16, 448) layer and a ReLU activation is applied on top of it. Following this there is another two sets of convolution-maxpooling-batch norm-maxpooling blocks. In these two blocks the convolutions has 256 and 512 filters respectively. Same padding and ReLU activation is used throughout and the data is flattened out in the next layer. Following this I've added fully connected layer which is

4 layer deep and each layer has 1024, 512, 256, 128 number of hidden units respectively and the output layer has 84 units.

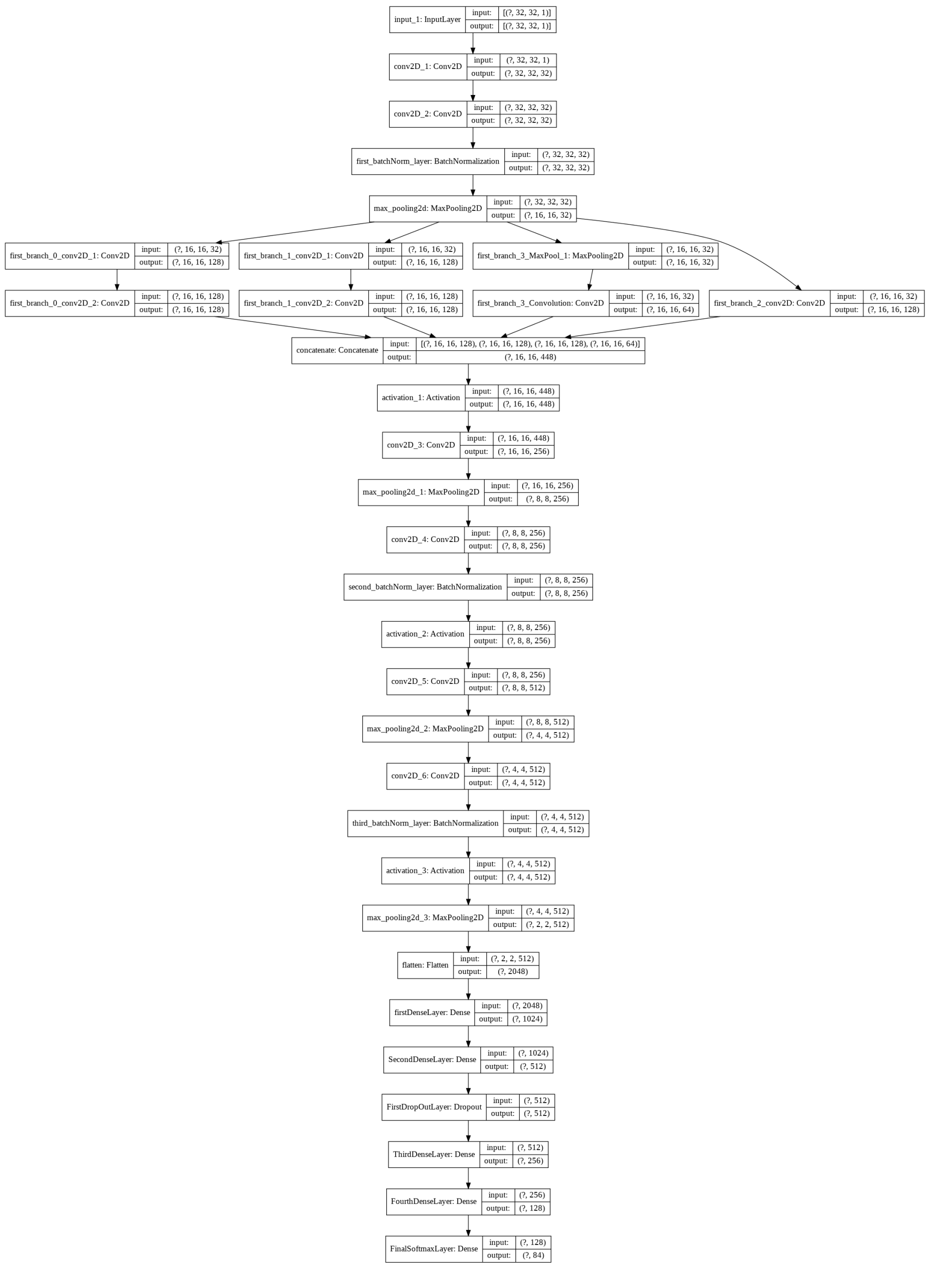

input_1: InputLayer | input: [(?, 32, 32, 1)] | output: [(?, 32, 32, 1)]
conv2D_1: Conv2D | input: (?, 32, 32, 1) | output: (?, 32, 32, 32)
conv2D_2: Conv2D | input: (?, 32, 32, 32) | output: (?, 32, 32, 32)
first_batchNorm_layer: BatchNormalization | input: (?, 32, 32, 32) | output: (?, 32, 32, 32)
max_pooling2d: MaxPooling2D | input: (?, 32, 32, 32) | output: (?, 16, 16, 32)
first_branch_0_conv2D_1: Conv2D | input: (?, 16, 16, 32) | output: (?, 16, 16, 128)
first_branch_1_conv2D_1: Conv2D | input: (?, 16, 16, 32) | output: (?, 16, 16, 128)
first_branch_3_MaxPool_1: MaxPooling2D | input: (?, 16, 16, 32) | output: (?, 16, 16, 32)
first_branch_0_conv2D_2: Conv2D | input: (?, 16, 16, 128) | output: (?, 16, 16, 128)
first_branch_1_conv2D_2: Conv2D | input: (?, 16, 16, 128) | output: (?, 16, 16, 128)
first_branch_3_Convolution: Conv2D | input: (?, 16, 16, 32) | output: (?, 16, 16, 64)
first_branch_2_conv2D: Conv2D | input: (?, 16, 16, 32) | output: (?, 16, 16, 128)
concatenate: Concatenate | input: [(?, 16, 16, 128), (?, 16, 16, 128), (?, 16, 16, 128), (?, 16, 16, 64)] | output: (?, 16, 16, 448)
activation_1: Activation | input: (?, 16, 16, 448) | output: (?, 16, 16, 448)
conv2D_3: Conv2D | input: (?, 16, 16, 448) | output: (?, 16, 16, 256)
max_pooling2d_1: MaxPooling2D | input: (?, 16, 16, 256) | output: (?, 8, 8, 256)
conv2D_4: Conv2D | input: (?, 8, 8, 256) | output: (?, 8, 8, 256)
second_batchNorm_layer: BatchNormalization | input: (?, 8, 8, 256) | output: (?, 8, 8, 256)
activation_2: Activation | input: (?, 8, 8, 256) | output: (?, 8, 8, 256)
conv2D_5: Conv2D | input: (?, 8, 8, 256) | output: (?, 8, 8, 512)
max_pooling2d_2: MaxPooling2D | input: (?, 8, 8, 512) | output: (?, 4, 4, 512)
conv2D_6: Conv2D | input: (?, 4, 4, 512) | output: (?, 4, 4, 512)
third_batchNorm_layer: BatchNormalization | input: (?, 4, 4, 512) | output: (?, 4, 4, 512)
activation_3: Activation | input: (?, 4, 4, 512) | output: (?, 4, 4, 512)
max_pooling2d_3: MaxPooling2D | input: (?, 4, 4, 512) | output: (?, 2, 2, 512)
flatten: Flatten | input: (?, 2, 2, 512) | output: (?, 2048)
firstDenseLayer: Dense | input: (?, 2048) | output: (?, 1024)
SecondDenseLayer: Dense | input: (?, 1024) | output: (?, 512)
FirstDropOutLayer: Dropout | input: (?, 512) | output: (?, 512)
ThirdDenseLayer: Dense | input: (?, 512) | output: (?, 256)
FourthDenseLayer: Dense | input: (?, 256) | output: (?, 128)
FinalSoftmaxLayer: Dense | input: (?, 128) | output: (?, 84)

Fig. 3: Layer by layer image data flow in AKHCRNet Model

The last layer has 84 hidden units with Softmax activation applied to it to comply with the 84 predictable classes in the Bengali dataset. The last few hidden layers are penalized with L2 Regularization method to decrease over-fitting, Corinna Cortes et. al [5]. The regularizers used in the last few hidden layers are kernel regularizer, as weights in the network is a very high dimensional parameter-vector, but the bias is just a single real number. so almost all the parameters are in weights rather than in the bias. So I think adding a bias regularizer won't make that much of a difference. So it's useful to only use the kernel regularizer in this problem.

For a L layer deep network the loss function J is the following:

$$J(w^{[1]}, b^{[2]}, ..., w^{[L]}, b^{[L]})$$

Where W[i], b[i] is the i-th layers weight vector and bias respectively. The function J is a softmax cross-entropy loss also known as categorical cross-entropy loss shown below. Softmax function:

$$f(s)_i = \frac{e^{s_i}}{\sum_j^C e^{s_j}} \tag{4}$$

Categorical cross-entropy loss:

$$CCE = -\sum_i^C t_i \log(f(s)_i) \tag{5}$$

In this specific case of Multi-Class classification the labels are one-hot, so only the positive class Cp keeps its position in the loss. There is only one element of the target vector t which is not zero ti = tp. So discarding the elements of the summation which are zero due to target labels, we can say:

$$CCE = -\log\left(\frac{e^{s_p}}{\sum_j^C e^{s_j}}\right)$$

Here ti and si are the ground truth and the CNN prediction score for each class i in all classes C, An softmax activation is applied.

To make understanding clear from now on the categorical loss will be represented as

$$L(\hat{y}, y)$$

where y hat is the predicted label from softmax output unit for a given ground truth y. After adding the L2 regularization the loss function J has become:

$$J(w^{[1]}, b^{[2]}, ..., w^{[L]}, b^{[L]}) = \frac{1}{m}\sum_1^n L(\hat{y}, y) + \frac{\lambda}{2m}\sum_1^L \left\|W^{[L]}\right\|^2$$

The extra term

$$\frac{\lambda}{2m}\sum_1^L \left\|W^{[L]}\right\|^2$$

is the L2 weight regularizer. W is a weight vector of dimensions (n[L-1], n[L]) where n[L-1] and n[L] is the number of hidden unit in the L-1 th and Lth layer of the deep network. So the weight decay is now applied so that overfitting doesn't occur. The reason behind this deep network is to ensure not to under-fit the network. I used a 7-10 layer deep network, it was getting 90% to 92% accurate in 20 epochs. So adding more layers has made the architecture more prone to learning intricate features that it hasn't learnt before.

As this is turning out to be a very deep network I also added a dropout with the rate of 50% in the second hidden layer. Experiments are made and it is seen that between 20-80% dropout rate of 50% worked the best for this problem.

5 Training, Learning rate scheduling, Optimization

The Adam Optimizer algorithm is used for this architecture, with no change to the default hyper-parameters

$$\alpha = 1e - 3$$

$$\beta_1 = 0.9$$

$$\beta_2 = 0.999$$

$$\epsilon = 1e - 8$$

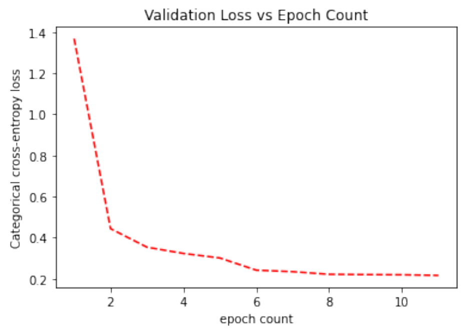
and instead of using a automated learning rate scheduler I've manually tweaked the learning rate epoch by epoch. Here is how that goes:

Epochs	Set Learning rate	Accuracy achieved	Cross Entropy Loss
First 5 Epochs	0.001	95.11%	0.30081
Next 3 Epochs	0.0001	96.79%	0.22147
Last 3 Epochs	0.00004	96.80%	0.21612

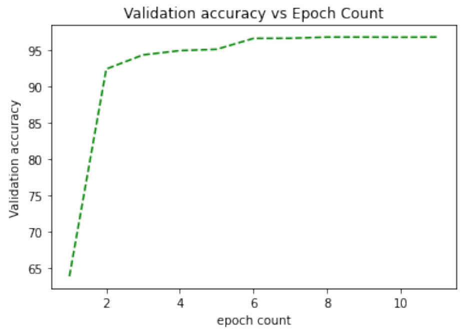
The result after 11 epochs is shown in the Result Section.

6 Result

In the following figure Figure-4 epoch by epoch validation loss and accuracy is shown. Clearly no sign of overfitting can be seen, the overall accuracy is 96.80% in 11 epochs which is a monumental jump from previous 500 epoch CNN ensemble training by Saha and Saha et al. and 47 epoch training by K Chatterjee et al.



(a) Validation loss over time



(b) Validation accuracy over time

Fig. 4: validation accuracy and loss with each passing iteration.

Below is the confusion matrix of 84 classes of Bengali character predictions. You can also see all the precision, recall, f1-score for each category in the following link. `https://bit.ly/akhcrnetreport-csv`

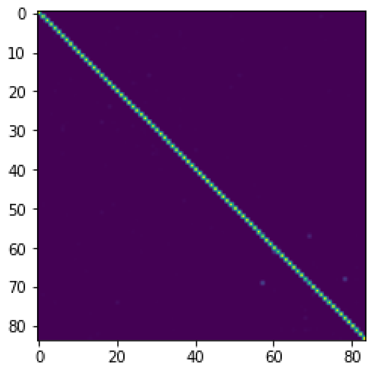


Fig. 5: Confusion Matrix

Here Precision is the ratio of correctly predicted positive observations in Bengali characters to the total positive predictions. The question that this metric answer is of all characters that labeled as some label, how many of them is actually that label? The high precision always relates to the low false positive rate. We have got about 0.97 (avg over 84 class examples) precision which is pretty good for a hand character recognizer. The report for each class can be seen following the link `https://bit.ly/akhcrnetreport-csv`.

$$precision = \frac{TP}{(TP + FP)}$$

Recall (Sensitivity) - Recall is the ratio of correctly predicted positive observations to the all observations in actual class - yes. The question recall answers is: Of all the characters that truly that label, how many did we label? We have got recall of 0.97 which is good for this model.

$$recall = \frac{TP}{(TP + FN)}$$

F1 score - F1 Score is calculated by the weighted average of precision and recall. Therefore, this score takes into account both false positives and false negatives. Intuitively it is not as easy to understand as accuracy, but f1-score can be more useful than accuracy, as we have an some-what uneven class distribution. Accuracy works best when the false positives and false negatives have the similar cost. If the cost of false positives and false negatives are very different, it's better to look at both Precision and Recall. In our case, F1 score is 0.97 (avg). The f1-score for each class can be seen following the link
`https://bit.ly/akhcrnetreport-csv`.

$$f1 - score = \frac{2 * (Recall * Precision)}{(Recall + Precision)}$$

6.1 Discussion

I call this architecture AKHCRNet which is a fancy way of saying Hand Character recognizer Network by by Akash. This architecture out-performs every other proposed Bengali classifier algorithms. Let's see how other researchers works did back then:

Researchers and their work	Total Classes	Architecture	Accuracy	Epochs
Purkayastha et al.[6]	50	Vanilla CNN	89.01%	
Rahmen et al.[8]	50	CNN	85.96%	
Alif et al.[9]	84	Modified ResNet-18 with dropout layer of 20% rate	95.10%	
Rumman Rashid Chowdhury et al.[10]	50 Characters + 10 Numbers	Custom CNN	91.13 % (50 Char) + 98.42% (Numbers)	92
Majid and Smith[12]	50	Non CNN Architecture	96.80%	
Saha and Saha[13]	84	Ensemble CNN	97.21% (95.67% & 96.12%)	500
Chatterjee, Dutta et al.[1]	84	Transfer Learning on ResNet-50	96.12%	47
Proposed Work	84	Custom 32-Layer Deep Network, AKHCRNet	96.80%	11

The Saha and Saha paper did the ensemble of CNN, with two CNN architecture, each of which had the accuracy 95.67% and 92.43% respectively, but AKHCRNet model was able to achieve the accuracy of 96.80%. The code and the weight file is available for download here:
`https://www.github.com/theroyakash/AKHCRNet`.

7 The Conclusion

AKHCRNet model was able to achieve the accuracy of 96.80% on the BanglaLekha-Isolated Dataset of 166K images, Without ensembling, without transfer learning and with a minimal number of epochs our proposed solution achieves state of the art result on Bengali Handwritten character recognition.

References

1. arXiv:1902.11133v1 [cs.CV] 25 Feb 2019 Bengali Handwritten Character Classification using Transfer Learning on Deep Convolutional Neural Network
2. M. A. Hasnat, S. M. Habib, and M. Khan, "A high performance domain specific OCR for Bangla script," in Novel Algorithms and Techniques In Telecommunications, Automation and Industrial Electronics, Springer, 2008, pp. 174–178.
3. Y. Wen, Y. Lu, and P. Shi, "Handwritten Bangla Numeral Recognition System and Its Application to Postal Automation," Pattern Recogn., vol. 40, no. 1, pp. 99–107, Jan. 2007
4. N. Das, S. Basu, R. Sarkar, M. Kundu, M. Nasipuri, and D. kumar Basu, "An Improved Feature Descriptor for Recognition of Handwritten Bangla Alphabet," arXiv:1501.05497 [cs], Jan. 2015.
5. Corinna Cortes et. al. L2 Regularization for Learning Kernels from the people at Google Research Team, https://arxiv.org/ftp/arxiv/papers/1205/1205.2653.pdf
6. B. Purkaystha, T. Datta, and M. S. Islam, "Bengali handwritten character recognition using deep convolutional neural network," in 2017 20th International Conference of Computer and Information Technology (ICCIT). IEEE, 2017, pp. 1–5
7. LeCun, Y., Cortes, C.: MNIST handwritten digit database. (2010).
8. M. M. Rahman, M. Akhand, S. Islam, P. C. Shill, M. Rahman et al., "Bangla handwritten character recognition using convolutional neural network," International Journal of Image, Graphics and Signal Pro- cessing (IJIGSP), vol. 7, no. 8, pp. 42–49, 2015
9. M. A. R. Alif, S. Ahmed, and M. A. Hasan, "Isolated Bangla handwritten character recognition with convolutional neural network," in 2017 20th International Conference of Computer and Information Technology (ICCIT), 2017, pp. 1–6.
10. Chowdhury, Rumman and Hossain, Mohammad and Islam, Raihan and Andersson, Karl and Hossain, Sazzad. Bangla Handwritten Character Recognition using Convolutional Neural Network with Data Augmentation. 10.1109/ICIEV.2019.8858545.
11. M. Biswas et al., "BanglaLekha-Isolated: A multi-purpose comprehensive dataset of Handwritten Bangla Isolated characters," Data in Brief, vol. 12, pp. 103–107, Jun. 2017
12. N. Majid and E. H. B. Smith, "Introducing the Boise State Bangla Handwriting Dataset and an Efficient Offline Recognizer of Isolated Bangla Characters," in 2018 16th International Conference on Frontiers in Handwriting Recognition (ICFHR), 2018, pp. 380–385.

13. S. Saha and N. Saha, “A Lightning fast approach to classify Bangla Handwritten Characters and Numerals using newly structured Deep Neural Network,” Procedia Computer Science, vol. 132, pp. 1760–1770, 2018.