Recognition of Bengali Handwritten Digits Using Convolutional Neural Network Architectures

Md Mahmudul Hasan
Department of CSE
BUET
Dhaka, Bangladesh
youngladesh@gmail.com

Md Rafid Ul Islam
Department of CSE
BUET
Dhaka, Bangladesh
rfd.009@gmail.com

Md Tareq Mahmood
Department of CSE
BUET
Dhaka, Bangladesh
tareq.mahmood2@gmail.com

***Abstract*—Handwritten digit recognition has been the "hello world" of deep learning. Yet, there are no significant work on Bengali handwritten digits due to a lack of benchmark dataset. NumtaDB is the largest dataset on Bengali handwritten digits and currently we have the best accuracy of 99.3359% on it. We used popular CNN architectures namely, ResNet34 and Resnet50. We preprocessed the data, used data augmentation, and trained our models with augmented data. We tested our models on both the raw test data and cleaned test data. We found that slightly underfitted models work better on the test data. And finally ensembled our six best models to get our final predictions. In this paper we describe some methods and techniques that performs well in NumtaDB dataset.**

***Keywords*—Handwriting recognition, Convolutional Neural networks, Image Recognition**

I. Introduction

Handwritten digit recognition (HWDR) is one of the stepping stone for many computer vision problems. The famous MNIST handwritten digit dataset [1] is used to benchmark new Machine Learning methods and architectures. HWDR is also fundamental to applications, such as finding postal codes [1], license plate recognition[2], automated bank check reading[3], etc. Although Bengali is one of the most spoken language in the world, historically, research in Bengali handwritten digits has been very limited. The latest papers in Bengali handwritten digit recognition are decades old [4]. And it is almost untouched by the recent advancement in Machine Learning and Deep Learning [5]. The main bottleneck for this drawback has been unavailability of publicly available large datasets that are free from age, gender, geographical biases. NumtaDB [6] is the largest dataset of Bengali handwritten digit dataset that was created to ameliorate this situation. The NumtaDB dataset consists of more than 85000 handwritten digits, collected from over 2700 contributors from different age group, gender and geography. In a recently organized Bengali handwritten digit recognition competition, around 57 teams participated to compete against each other, where our model achieved the highest accuracy, which is 99.3359%. In the subsequent sections, we would describe the techniques we have used to achieve the high accuracy.

We have found that for adequately large Convolutional Neural Network architectures, such as ResNet34 or ResNet50 [5], contrary to common intuition, preprocessed test image gives worse result than unprocessed images. We observed that when the training data is heavily augmented to match the augmented test data, the model gives better result if it is slightly underfitted. We also used a weighted voting method with 6 of our best performing models to generate the final prediction. We observed that in this ensembling method, even number of models produce a better result than odd number of model. In this paper we describe the methods and techniques we have found to be best for the particular task of Bengali handwritten digit recognition in the NumtaDB dataset.

In section II we briefly describe the NumtaDB dataset. And the evaluation method that was used to find our accuracy.

Section III describes two data preprocessing methods, one is data augmentation before training, and another is preprocessing the test data to eliminate unnecessary noises from data.

Section IV demonstrates the Neural Network architectures that was used in our model prediction, and the hyperparameters and other tweaks we have used to create our final model.

Section V discusses our results, and section VI concludes the paper.

II. Data Description

NumtaDB is a collection of Bengali Handwritten Digit data [6]. The dataset contains more than 85,000 digits from over 2,700 contributors. The dataset was rigorously checked for mislabeling, noise and labeling biases. The dataset was collected from 6 different sources[6]. Data from 5 of the sources were split 85% - 15% for training and testing. And one of the source was used completely as testing data. Splitting was done such that set of people who contributed to the training data does not intersect with set of people who contributed to the test set. Number of images per digit kept approximately equal. Most of the data were collected from students of public universities in Dhaka. As students at these universities come from all around Bangladesh, it is implicitly made sure that the dataset contains representation from diverse regions of Bangladesh. The dataset also contains data from both children and adults. So it is supposed to be unbiased in terms of geographic location and age. The test dataset contained 6 subsets, codenamed a, b, c, d, e, f. Additionally two augmented data sets were produced from test set a and c. The final

978-1-5386-8207-4/18/$31.00

evaluation accuracy was unweighted average of accuracy in these 8 subsets. [7]

$$Acc = \frac{1}{8}\sum_{i=1}^{8} A_i$$

III. Data Preprocessing

A. Data Augmentation

As mentioned earlier, NumtaDB contains augmented images in its test set. Types of augmentation that are applied on dataset 'a' and 'c' are mentioned in NumtaDB's data overview page[1]. We apply these augmentation along with some other augmentations on train data in order to mimic test data. They are as follows.

- Rotation: Rotating image with respect to center. One key thing to note about this operation is that image dimensions may not be preserved after rotation. We limit rotation up to $45\,\mathrm{deg}$ in either direction (clockwise and anti-clockwise). And, to keep output image same dimension, we squeeze rotated image in front of constant black image as done in 'auga' and 'augc' test dataset.
- Translation: Translation is the shifting of digits location along row or column. However, this creates a blank area on the opposite direction of the translation. We fill the blank area repeating border values of original image.
- Shear: We displace each pixel of am image in the vertical direction with proportional to its distance from left edge of that image. We displace in either direction (left or right) randomly by choosing random proportional constant.
- Salt-pepper noise: This noise is produced by sharp and sudden disturbances of signal. It looks like sparingly occurring white and black dots.
- Blurring: This is a smoothing operation. We apply a linear filter on initial image.
- Zoom in: This augmentation zooms the initial image in. We randomly zoom image in by not more than 2x.
- HSV shift: In other words, Hue Saturation Value shift. This is an alternative representation of an RGB image. We randomly shift hue and saturation value of image.
- Coarse dropout: We randomly black out certain rectangular areas of image.
- Superimpose: This is done to replicate the effect of text being written on the back of an already written page. For each image, we find another image. We vertically flip the other image and take a weighted sum of two image.

For each image, we randomly pick 1-3 augmentations and apply them simultaneously. Also, the parameters for each augmentations are selected randomly within a fixed range. With this procedure, we create 5 to 7 new images for each image in training dataset.

[1]https://www.kaggle.com/BengaliAI/numta/home

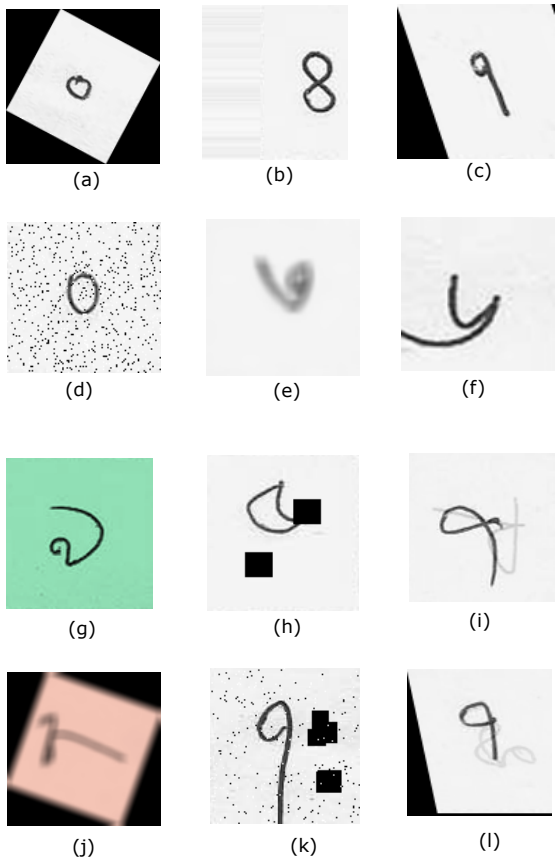

Fig. 1. (a) Rotation, (b) Translation, (c) Shear, (d) Salt-pepper noise, (e) Blur, (f) Zoom in, (g) HSV shift, (h) Coarse dropout, (i) Superimpose, and (j), (k), (l) are sample combined augmentations.

B. Test Data Cleaning

The testing dataset had 8 subsections,

1) "testing-a"
2) "testing-b"
3) "testing-c"
4) "testing-d"
5) "testing-e"
6) "testing-f"
7) "testing-auga"
8) "testing-augc"

The augmented datasets, "testing-auga", "testing-augc" and "testing-f" were particularly troublesome. They were heavily augmented and transformed to make it hard to recognize. Some of the augmentations includes,

- Affine transformation
- Coarse dropout
- Addition of noise
- Superimpose
- Inverted image

One or more of these transformations were applied to augmented image sets to make them non-recognizable. "testing-f" data was poorly lit, unusually cropped, and then also added different type of noises and transformations.

We used various type of data cleaning process to each of these test sets to make it more recognizable. Each data cleaning process was applied to all images of a particular set. We did not pick and choose a transformation for particular images of a subsection. But we looked at the condition of a subsection as a whole, and applied a transformation to the entire subsection. Here we describe the data cleaning process and transformations we have used to the subsections of test sets.

One of the difficult type of augmentation to remove was Coarse dropout and affine transformation. In Coarse dropout, a parallelogram size part of the image was deleted right on top of the digit edge (Figure 4). In affine transformation, the images were translated, scaled, rotated, shear mapped, and sometimes multiple of these effects combined (Figure 2). Each of these transformations applied, or Coarse dropout added, a part of the image becomes dark. But upon observation, we found that the the dark portion of the images created with affine transformation or Coarse dropout, was almost homogeneously complete black, that means gray scale value = 0. In these images, the actual digit portion of the image was almost never complete black. So, in order to remove this dark portion, we set all complete black pixels to the median grayscale value of the image. In this process, there were still the edge of the black portion of the augmented image, where a thin dark ramp remained. So, we again applied a median blur effect on the image to remove the ramp. The final result was visually very close to normal images (Figure 3 Figure 5).

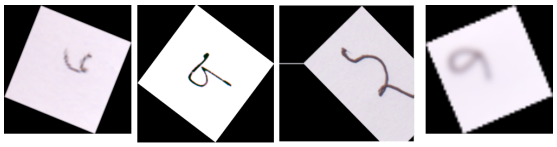

Fig. 2. Affine Transformation augmented images

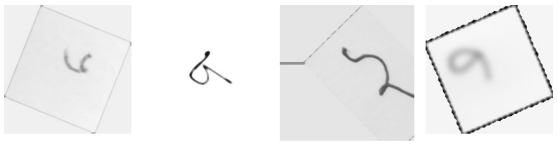

Fig. 3. Affine Transformation removed

Fig. 4. Parallelogram Cropped augmented images

One thing to mention is that, this effort not necessarily improved our accuracy. We have come to the conclusion that, even if we replaced the dark pixels with median value, there still remained an edge that the convolutional layers of our network would pick up. So, this is an important realization, that if an image is visually okay, it still might not be a good data for a Convolutional Neural Network.

Fig. 5. Parallelogram Cropped

To remove the added noise and superimposed image in ”testing-auga”, ”testing-augc” and ”testing-f” we used a median blur of filter size 9x9. This method successfully removed Guassinan noise completely, and also other type of noises and sharp edges (Figure 6, Figure 7). But some test data in ”testing-f” became victim of the blurring, and the digit was blurred to a point that was unrecognizable. So we had to withdraw this operation from ”testing-f”.

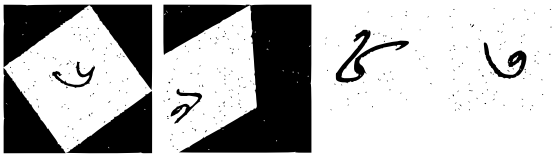

Fig. 6. Noise example

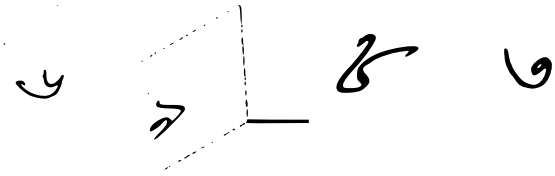

Fig. 7. Noise removal example

”testing-e” subset of test images had all the images inverted. That means the digits were written in white pixels, where the background was dark. The digits were binarized and they covered the entire image. We inverted these images and added a padding to make it more consistent with our training image (Figure 8, Figure 9). The inversion didn't seem to have any significant advantage, as our network seemed to identify white digit in dark background just as fine. We think it is because the filters used in the convolutional layers work on edges and the network easily identified the edges regardless of the background or foreground color.

Fig. 8. White digit in dark background

After applying these transformations and operations in testing set “testing-auga”, ”testing-augc” and ”testing-f”, we

Fig. 9. Image inverted to normal

created a new testing set, and we named it ”clean_test” (Figure 10). At this point of the competition the submission window was 2 submission a day. After submitting the test score of ”clean_test”, we found that the overall accuracy dropped by around 1%. Due to the small submission window, we were unable to verify which of these individual data preprocessing operations worked and which did not.

Fig. 10. Cleaned Data sample

By applying Otsu thresholding method to this ”clean_test” dataset, we created another dataset ”binarized_clean_test” (Figure 11). But after testing, the accuracy of ”binarized_clean_test” was even worse than ”clean_test” dataset.

Fig. 11. Binarized clean data sample

We understand that as the smallest neural network we have used in our final prediction is ResNet34, which has 34 layers, our network was complex enough for identifying subtle nuances of the images to make a final prediction. But by cleaning or binarizing the data, we remove information from the image that the network find useful to make a prediction. That’s why applying image processing to test data resulted in worse result. Another reason that can be attributed to the failure of image processing is, the same transformations were not used in our training data. So preprocessing only the test data results in inconsistency in training and testing.

One operation that almost always improved the accuracy is padding unusually shaped images. A large portion of the ”testing-f” dataset had unusual shape. I.e. the width of the image was much smaller than the height. In this type of situation our network was failing because our network was made to take only square images as input. So when the input images were rectangular in shape, but not square, our architecture crops a square portion of the image from the center, and feeds it to the convolutional layers. To avoid this problem, we identified all the test images that has a width/height ratio of less than 70%, and padded them on left and right to make it square. The color of the padding was chosen to be the median of the image. This operation allowed one of our models to first reach 99% accuracy, and 5 of the 6 ensembled models in our final submission applied this operation to test images.

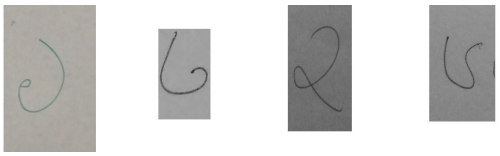

Fig. 12. Unusually shaped data

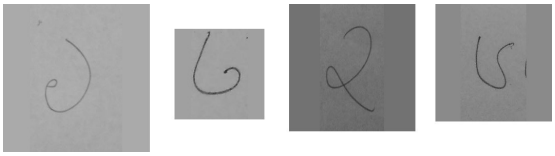

Fig. 13. Median value padded to reshape the data

IV. Methodology

We first chose the suitable Convolutional Neural Network architectures. Then, we tuned the hyperparameters to train our models. Finally, we ensembled our best six models and got our final results. Figure 14 shows the whole pipeline.

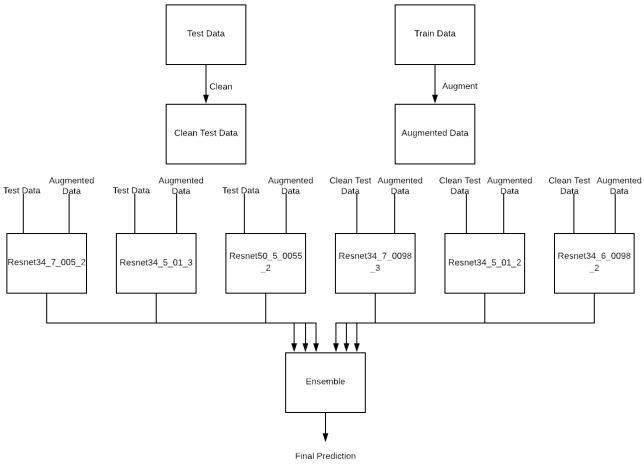

Fig. 14. Conceptual Diagram of the Solution Pipeline

A. Architectures

After AlexNet [8] the ILSVRC2012-ImageNet contest, deep Residual Network [9] is next significant work in deep learning based computer vision. ResNet is able to train deep networks with hundreds or even thousands of layers and accomplishes fascinating performance with the idea of ‘identity shortcut connection’. We use two variants of ResNet ResNet34 and ResNet50.

ResNet50 is a 50 layers Residual Network. Similarly ResNet34 is of 34 layers. We train six different models, five of them are ResNet34 and one is ResNet50.

B. Hyperparameter Tuning

We consider 3 things as hyperparameter in this work.

1) No. of augmentations per image
2) Learning rate
3) No. of epochs to train

C. Ensembling

In our work, we ensembled six different models to achieve our highest accuracy 99.3359%. We used weighted voting from the individual model predictions to get the final prediction. For weights of a model, we used the normalized form of their validation accuracy. We used even number of models instead of odd numbers because even number ensembles are less likely to predict wrong on a particular data due to wrong predicted models having higher weights for a particular data. Suppose we have five models A, B, C, D, E, with individual validation accuracy 99.008%, 98.986%, 98.74%, 98.65%, and 98.55%. When we ensemble with these five models, models C, D, and E are more likely to predict the same wrong digit. When we take a weighted vote of these five models, the three wrong prediction wins over the two correct predictions no matter how high the accuracy of the loosing predictions are. So instead if we ensemble with model A, B, C, D and exclude E. Now if A and B make a prediction, and C and D oppose, C and D are going to loose because of their low weight. If low accuracy predictions wants to overpower high accuracy predictions, by sheer number, three of them have to agree on the incorrect prediction. Which is less likely. Table I shows the summary of the ensembled models.

D. Underfitting

When we are heavily augmenting, augmented and non augmented datasets has a ratio of $3:1$, $5:1$ or $7:1$. The network was seeing more distorted images than normal images. The training became biased towards the distorted features. It started expecting the augmented features from test images. But because the normal features were the ones that were common among all test sets, the network failed to give more emphasis on those features. The more we trained on augmented data, the more the network got biased towards the augmented features. So less trained or underfitted model gave better results on the test sets. We chose the uderfitted model by determining the epoch where training and validation accuracy became almost equal and chose the epoch previous to it.

V. Evaluation and Results

We trained our models by splitting the dataset into 80% training and 20% validation set. We found that underfitted models performed better as discussed in Section IV-D. So we took the model from the epoch previous to the epoch where train and validation accuracy became almost same for submitting our predictions. Then we found the accuracy on the test set from the Bengali Handwritten Digit Recognition competition in Kaggle[2]. The accuracy on the test set for our selected models is given in Table II.

TABLE I
Ensembled Models Summary

Architecture	Hyperparameter	Value	Weight
Resnet34	No of Augmentation per sample	7	0.1669829
	Learning rate	0.0098	
	No of Epoch	3	
	Image resized	True	
Resnet34	No of Augmentation per sample	5	0.16680934
	Learning rate	0.01	
	No of Epoch	2	
	Image resized	True	
Resnet34	No of Augmentation per sample	6	0.16674025
	Learning rate	0.0098	
	No of Epoch	2	
	Image resized	True	
Resnet34	No of Augmentation per sample	7	0.166588595
	Learning rate	0.005	
	No of Epoch	2	
	Image resized	False	
Resnet34	No of Augmentation per sample	5	0.16629538
	Learning rate	0.01	
	No of Epoch	3	
	Image resized	False	
Resnet50	No of Augmentation per sample	7	0.16658354
	Learning rate	0.0055	
	No of Epoch	2	
	Image resized	False	

TABLE II
Test Score from the Competition

Model (Arch_Aug_Lr_Epoch)	Kaggle Score (test accuracy)
Resnet34_7_0098_3	99.094%
Resnet34_5_01_2	98.991%
Resnet34_6_0098_2	98.95%
Resnet34_7_005_2	98.86%
Resnet34_5_01_3	98.686%
Resnet50_7_0055_2	98.857%

VI. Conclusion

Due to our limitation of computing resources, we could not test large deep neural network architectures such as, Inceptionv3, ResNext101, etc. In future we might look into the performance of neural network models using these architectures. Even after rigorous checking from the creators, some of the data in NumtaDB was mislabeled. And some images were so distorted that it was impossible for humans to recognize

[2]https://www.kaggle.com/c/numta

them. Handling these type of images were particularly challenging for us. We are hoping NumtaDB creators will release a new version of dataset with minimum number of mislabeled images.

As mentioned earlier, because of the submission limitation during the competition, we could not justify how each of the data cleaning methods we have applied contribute to higher accuracy. In future we want to find out which type of data cleaning procedure produced the best results.

Also we wish to extend our current work beyond the scope of digit recognition and apply it to broader problems such as license plate recognition, or handwritten character recognition. The performance of the methods we have described in this paper is yet to be seen in those applications.

ACKNOWLEDGMENT

We would like to thank the Bengali.Ai [10] community to give us the opportunity to participate in the "Bengali Hand Written Digit Recognition" competition and be able to set a benchmarking score on the NumtaDB dataset. We would also like to offer our thanks to Google Colaboratory [11] to provide researchers like us with free GPU support which we used extensively throughout the competition [7].

REFERENCES

[1] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel, "Handwritten digit recognition with a back-propagation network," in *Advances in Neural Information Processing Systems 2*, D. S. Touretzky, Ed. Morgan-Kaufmann, 1990, pp. 396–404. [Online]. Available: http://papers.nips.cc/paper/293-handwritten-digit-recognition-with-a-back-propagation-network.pdf

[2] S.-L. Chang, L.-S. Chen, Y.-C. Chung, and S.-W. Chen, "Automatic license plate recognition," March 2004, vol. 5, no. 1, pp. 42–53.

[3] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," vol. 86, no. 11, Nov 1998, pp. 2278–2324.

[4] S. Basu, R. Sarkar, N. Das, M. Kundu, M. Nasipuri, and D. K. Basu, "Handwritten bangla digit recognition using classifier combination through ds technique," 2005.

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

[6] S. Alam, T. Reasat, R. M. Doha, and A. I. Humayun, "Numtadb - assembled bengali handwritten digits," 2018.

[7] "Bengali handwritten digit recognition — kaggle," https://www.kaggle.com/c/numta, accessed on September 20, 2018.

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[9] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[10] "Bengali.ai — bengali.ai is a community that works to open-source datasets for research." https://bengali.ai/, accessed on September 20, 2018.

[11] "Google colaboratory — google," http://colab.research.google.com/, accessed on September 20, 2018.