DEEP LEARNING: BENGALI COMPETITION

Austin Bell & Ziyin Wang
alb2307 & zw2605
Columbia University
{alb2307, zw2605}@columbia.edu

ABSTRACT

Our final submission, currently ranked 260 with a score of .9752, was the result of ensembling six models. Three models that we built and three high performing models shared on Kaggle. We leveraged a series of data augmentation (centering, shifting, adding noise, blocking parts of images), tested a variety of pretrained models (DenseNet and SeResNext), and leveraged a variety of other tools and techniques (class balancing in multi-label classification, various loss functions, and multi-task learning). Overall, we are quite satisfied with the results of our model since this is our first time working with image data.

1 INTRODUCTION

At the time of writing, our current submission to the Kaggle Bengali AI Handwriting Recognition competition scored 0.9752, which places us at rank 260. We tested and developed a variety of models with various augmentations. In this paper, we provide an overview of the final models that were employed to achieve our final score. We next discuss the issue of memory management, which we saw as a primary roadblock in this competition. Finally, this paper concludes with a discussion of what steps we would have liked to take with more time and money.

Overall, our final submission consisted of three models that our team developed. These are described in detail in the methodology section

- A baseline DenseNet model
- A multi-task learning model
- A SeResNext Model with a Focal Loss cost function

The goal of the competition was to predict the components of a Grapheme (handwritten Bengali character). A Grapheme consists of three components: the grapheme root, vowel diacritic, and consonant diacritic.

2 METHODOLOGY

Our methodology consisted of three core steps: 1) data augmentation, 2) modelling, and 3) ensembling.

2.1 DATA AUGMENTATION

Data augmentation is a key component for increasing the amount of data seen by the model, but also increasing the variety of data that the model sees. It is obvious why increasing the amount of data that is seen by the model would improve our score, but we also found that variety made a significant difference during ensembling. By utilizing different data augmentation techniques for each ensembled model, each model had different strengths leading to larger efficacy gains during ensembling.

The first augmentation that we employed was to resize, center, and rescale our images. We created a bounding box around the handwritten character then filled in the sides with our neutral color. This

was resized to 128 x 128 to minimize memory consumption. This augmentation required a decent number of iterations to get right due to Kaggle's memory constraints.

Next, we utilized the Albumentations package for a variety of standard data augmentations. While we want to increase the variety of our augmentations, it is important to keep them within the realm of possibility otherwise they are useless. For example, flipping an image upside down would provide zero benefit as we made the assumption that our test data would not include any upside down handwritten characters. We did however, include the following augmentations with varying probabilities: different blurs, randomly adding noise, randomly distorting, shifting, scaling, and rotating the images.

Additionally, we utilized an implementation of the Gridmask augmentation (Chen et al. (2020)). This augmentation involves randomly placing a grid of square blocks over the image. This has the effect of blocking out random parts of the handwritten character to ensure that the model can predict with only partial characters.

Finally, we had some severely imbalanced classes across our three labels. Balancing classes in multi-label prediction is not a solved problem yet and there are many ways to do it. We solved it by re-weighting our samples to be included in each batch by re-initializing our weight sampling data loader at the beginning of each epoch. Essentially, epoch one balanced the classes in one label, epoch two balanced the classes in another label, so on and so forth. This enabled us to better predict rare classes.

2.2 MODELS

First, from our research, we found that DenseNet worked pretty well on Bangla character recognition. So, we defined our own model leveraging DenseNet121 (Huang et al. (2016)) as the base model. Our DenseNet was composed of one convolutional layer, one DenseNet121 which was pretrained on ImageNet, and two linear blocks. Each linear block consisted of a fully connected linear layer, optional batch normalization, optional dropout, an activation layer. The final layer output the number of classes equal to the sum of our three components. Essentially, our labels had hard parameter sharing for the entire network.

We then defined our classifier class, which wrapped our base model class. In this class, we split the final layer into three components relative to each label's size. Then, we calculated the individual loss value for each component and summed them to a single loss value for gradient descent. Our initial loss function was cross entropy across all three components.

Our second model implemented an additional form of multi-task learning. While we were already sharing weights across our three labels, we implemented a fourth label prediction. In this case, we predicted the entire Grapheme and then concatenated this grapheme prediction with the input to the final layer of our grapheme components prediction. The loss across all four labels were then used for optimization. Our belief here was that having knowledge of the whole grapheme would improve the prediction of its components.

After we had our baseline model, we wanted to try some different pretrained models. And we believed that SeResNext (Hu et al. (2017)) would perform better based upon the results of the shared Kaggle Notebooks. While we experienced a performance boost from the SeResNext model, it was not substantial.

Finally, we tested and applied a variety of additional changes. A few of the ones included in our final model are outlined below.

First, we added Early Stopping to our training process, which could ignore the result of the current epoch if our validation loss did not improve. Additionally, it ensured that we saved only the best performing model. This allowed us to maximize generalization performance and reduce the chance of overfitting. After some testing, we found that sometimes validation recall increased as validation loss did not decrease. Given that the competition's, metric was based on model recall, we changed the metric of Early Stopping from validation loss to validation recall.

Next, we tried to do some hyperparameter tuning, including changing dropout rate, activating / deactivating batch normalization, adjusting batch size, epoch times, layers, and layer hyperparameters.

Moreover, we added a learning rate scheduler to our training process, in case a gradually decreasing learning rate improved gradient descent's ability to reach a minimum.

Because of the aforementioned class imbalance of our data, we replaced our initial cross entropy loss function with a new loss function called 'Focal Loss' (Lin et al. (2017)). Focal Loss penalizes easily classified examples more providing an additional tool to help tackle class imbalance.

2.3 ENSEMBLING

The key component that improved our models was to use ensembling. We ensembled our three models plus three or four of the top performing models shared as notebooks during the competition. We used a simple averaging across all models, but weighted higher performing models slightly more than others. The key challenges here were memory consumption and the Kaggle time limit (submissions cannot exceed six hours). Overall though, ensembling methods improved our models by approximately 1%.

3 AN EXERCISE IN MEMORY MANAGEMENT

This competition required that we submitted Kaggle notebooks and did not allow us to ever see the actual test data. While this is totally fine, the Kaggle notebooks have very strict memory requirements during submission which cannot be measured or monitored (the only way of finding out that it exceed memory is to submit and wait and see if an error occurs). Our inexperience with this led to our baseline model taking significantly more time than anticipated as our first few models exceeded this memory limit.

Therefore, there was a week or two where we solely explored how to overcome these memory limitations to find out how we could submit the models. In the end, this was overcome through a series of changes including significantly reducing batch size, reducing image size, streaming our test data and predictions, utilizing smaller models, and deleting all unused data.

This also prevented any pseudo or active labelling strategies, which we believe would have improved our final results as well.

4 DISCUSSION AND NEXT STEPS

Given that we were both busy students and also learning how to implement computer vision models along the way, we did not have the time (or money) to implement everything that we wanted to. In this section, I outline two strategies that we were not able to pursue.

4.1 FINE-TUNING PRETRAINED MODELS

All of our pretrained models were the basic versions and all of them were pretrained on Imagenet. Imagenet is a great starting point, but the images in that repository are very different than images used in handwriting recognition. However, there are dozens of datasets dedicated to handwriting recognition – outside of our Bengali competition dataset. Combining all of these datasets would likely include millions of images. Fine-tuning our pretrained models on this combined dataset would have significantly improved the pretrained models' transfer learning capabilities, as it would be trained on a task much more similar to that of the competition. Our belief is that just this would have significantly improved our results and pushed us into high bronze or silver medal territory. However, given the size of the dataset, this would have drained our credits.

4.2 LEVERAGING BEAM SEARCH AND WHOLE GRAPHEME PREDICTION

Our overall goal was predict the grapheme components: grapheme root with 168 classes, vowel diacritic with 11 classes, and consonant diacritic with 7 classes. Which means there are around 13,000 combinations of the three components. However, some combinations may be more likely than others. Furthermore, if we knew the actual grapheme (i.e., sum of the components) then we would be able to predict its parts.

To accomplish this, we would have wanted to develop a separate model that predicted the entire grapheme. This model would serve as input prior to prediction of our components. We would implement a beam search technique that allowed us to identify the most likely combination of components given our prediction probabilities for both the components and the whole grapheme.

5 CONCLUSION

Overall, we feel very satisfied with our results, particularly since this was both of ours' first time working with image data or doing any sort of image classification. We think that with our next Kaggle competition we could quite easily receive a medal.

REFERENCES

Pengguang Chen, Shu Liu, Hengshuang Zhao, and Jiaya Jia. Gridmask data augmentation. *ArXiv*, abs/2001.04086, 2020.

Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7132–7141, 2017.

Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. Densely connected convolutional networks. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2261–2269, 2016.

Tsung-Yi Lin, Priya Goyal, Ross B. Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 2999–3007, 2017.