Bengali.AI Handwritten Grapheme Classification - MLiP Challenge 1 Report

Nolan Cardozo (s1034065) — Ajinkya Indulkar (s1034517) — Rodrigo Perez Victoria (s1034694)

March 2020

1 Introduction

In this report, we discuss the process of building a deep neural network that classifies three constituents (grapheme root, vowel diacritics, and consonant diacritics) from a handwritten grapheme in an image. While doing this, our primary goal is to build a moderately deep network (i.e., has moderate number of parameters) but still achieves an outstanding classification accuracy. This might be of particular interest because lighter models are easier and more efficient to deploy in real-time and use lesser computational power. Moreover, we would also like to understand whether the three components: grapheme root, vowel diacritics, and consonant diacritics, are tasks that are dependent on each other or not.

2 Approach

We decided to employ the power of transfer learning instead of training from scratch since the base models can already identify essential features, and it is efficient to train. We deploy two architectures, namely Xception [1] (moderately deep) and DenseNet101 [3] (very deep), and compare and contrast both.

For preparing the training data, we decided to crop and resize the images to a 64 x 64 resolution, which gets rid of the unnecessary padding in the image, and besides that, saves the model a lot of training time without compromising on the classification accuracy. We removed the final layer of the base network. We used three fully connected layers instead, since it is a multiple label multiple class task in which each parallel layer classifies each of the three components.

At first, we trained the network as described and noticed that the smaller components: vowel (11 classes) and consonant (7 classes) reached a very high accuracy instantly as compared to the grapheme(168 classes) and caused overfitting. To understand if the three components were dependent on each other and to solve the problem of overfitting, we employed the strategy of training only the root grapheme class layer while freezing the other 2 output layers. This results in the backpropagation of the loss of the other classes as noise, trying to achieve some kind of additive noise [2]. This technique forced the hardest task first and the results were comparable which confirms that the components are highly dependent on each other. Though unfortunately this did not solve the problem of overfitting.

Deep Neural networks are known to be data-hungry. To improve the performance of the model and to avoid the problem of overfitting, we used data augmentation consisting of rotation (around ten degrees), zoom (up to 15% of the total size), width shift and height shift (both randomly up to 15% of the width and height respectively). This also adds variation in the dataset and helps the network generalize well on the data.

3 Results

The Source Code of our Best Working Model can be viewed as a Kaggle Kernel **here**.

Our primary objective was to employ a moderately deep network, which would still comparatively give us good results. In Table 1, we see the comparison of the moderately deep Xception network to the very deep DenseNet101 architecture. As we can see, we managed to achieve 95.29% accuracy using the Xception network, which is comparable with the DenseNet101 network but takes half the amount of time to train.

Table 1: Comparison between XceptionNet and DenseNet101 Architectures

Model Architecture	Test Accuracy(Kaggle)	Training Time
XceptionNet	95.29%	150 mins
DenseNet101	95.14%	352 mins

In Table 2 we demonstrate the performance improvement of the Xception model, first when training all the final component layers and second when training only the hardest task: Grapheme layer and freezing the other 2 (with and without image augmentation).

In Figure 2, we see the training and validation accuracies and loss per epochs. The dataset was provided in 4 parquet files and it was stated that all the files are from the same underlying distribution. Hence, we improved the training efficiency by lazy loading and training each parquet file iteratively instead of loading all files in memory at once. This is why in Figure

Table 2: Performance Improvement of XceptionNet

Task	Test Accuracy(Kaggle)	Training Time
Train all final layers (without Augmentation)	92.84%	152 mins
Train one layer and freezing other 2 (without Augmentation)	92.82%	150 mins
Train one layer and freezing other 2 (with Augmentation)	95.29%	150 mins

1 we see the model statistics for each parquet file separately. We show in the figure, the loss and accuracy over the first 30 epochs, for each training set (0,1,2,3).

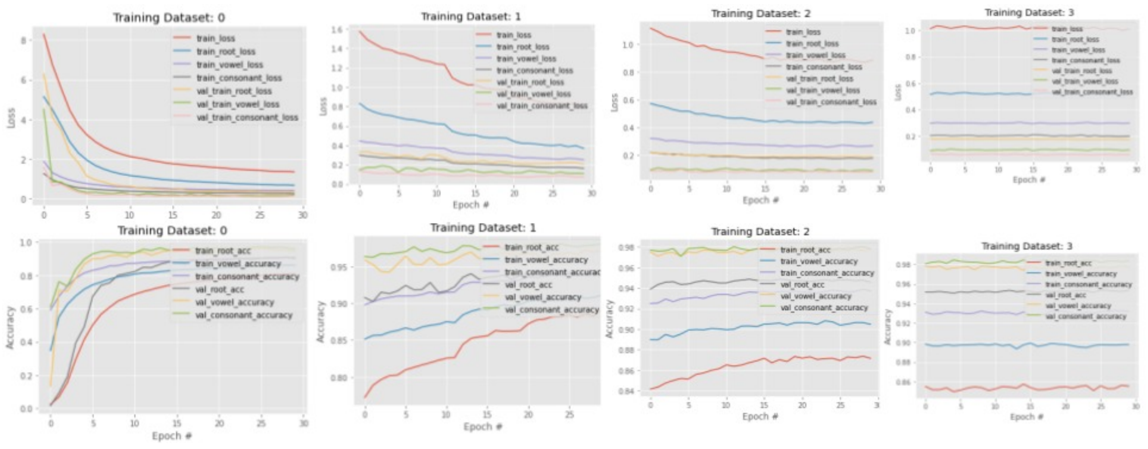

Figure 1: Training History of XceptionNet Architecture

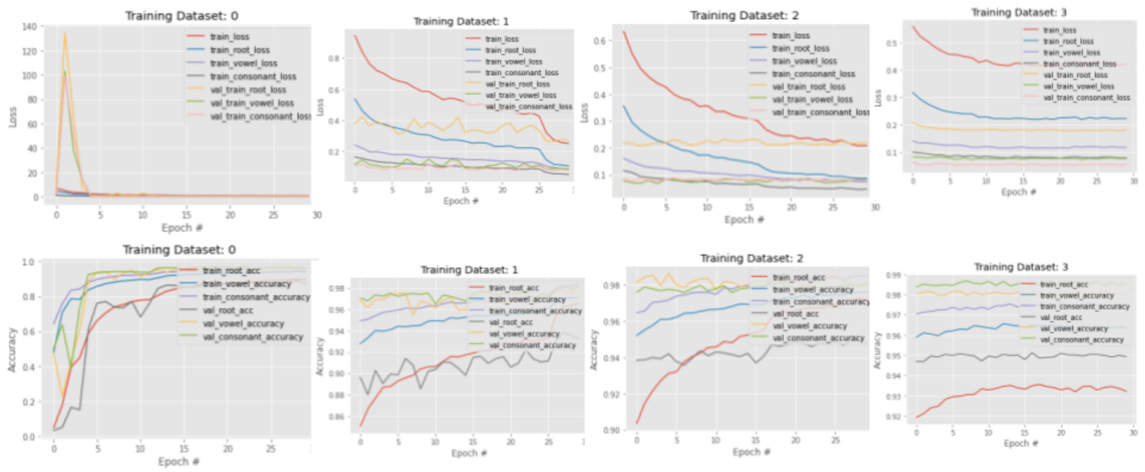

Figure 2: Training History of DenseNet101 Architecture

By comparing with figure 2 we learned from the training of these architectures, that the largest one has a highly unstable value for the validation of the root during training.

Another key observation about the dataset was that not all the combinations of constituents appear in the training set because not all possible combinations need to exist in a language. We assumed that if there is a combination present in training, then it is part of the language.

4 Author contributions

The report acknowledges the following contributions:

- **Nolan Cardozo**: Contributed in training other deep architectures such as ResNet50 and DenseNet161. Proposed research ideas for this task and communicated with the coach during meetings. Helped create content for the flash talk.
- **Ajinkya Indulkar**: Contributed in training models with XceptionNet architecture with no image augmentation for performance improvement analysis. Helped create content for the flash talk and presented the flash talk.

- **Rodrigo Perez Victoria**: Team captain. Set up team meetings and communicated with the coach during meetings. Contributed in training models of XceptionNet and DenseNet101 architecture for model comparison analysis. Helped create content for the flash talk.

5 Evaluation of the Process and Supervision

The public Kaggle Kernels were a good starting point as provided us with an in-depth Exploratory Data Analysis and a variety of Model Architectures to choose from. The provided dataset was large and required minimal pre-processing steps. Thus, there was a good learning curve while working with this dataset.

The Coach Meetings helped us in exploring tasks such as freezing of layers in a model architecture to understand its effect on test accuracy and tackle this challenge from a research-oriented perspective. Also, the flash talks helped us discuss and brainstorm on what essentially is the most crucial aspect of this challenge and kept us focused.

We mainly worked with Kaggle kernels as we were provided with a 30 hours per week limit on GPU usage. Working Kaggle kernels was also a good learning experience as the working pipeline isn't trivial and required some time to understand. Learning this will help us for the next challenge as well as future Kaggle Competitions.

References

[1] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258, 2017.

[2] Lasse Holmstrom and Petri Koistinen. Using additive noise in back-propagation training. *IEEE transactions on neural networks*, 3(1):24–38, 1992.

[3] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.