Machine Learning in Practice - Bengali.AI Handwritten Grapheme Classification

Jan Ondruch s1045948 | Koen Baarda s4613147 | David Roefs s4666623

Group 17 - MLiP B-Team

1 Introduction

During the last six weeks, our team took the first steps to explore and understand the field of machine learning. This was done primarily by taking part in the Bengali.AI Handwritten Grapheme Classification competition[1], which was hosted on the Kaggle platform[2]. This report describes how we educated ourselves in the domain by studying current state-of-art solutions, step by step achieving our final submission of a ResNet-18 architecture with a final private score 0.8784.

2 Approach

As some of our team members have never encountered machine learning before, we first had to acquire knowledge in the field. YouTube playlists such as Machine Learning & Deep Learning Fundamentals[3] have proven to be very helpful to understand general concepts. Studying Kaggle notebooks, especially those directly related into the competition[4] gave us a direct insight to possible solutions and served as the most useful source of inspiration for our own entry.

Slowly after we started working on the project, we realised that neural networks have many parts contributing to their performance, such as data normalisation, data augmentation, models etc. With the knowledge we had gained and some assumptions, we could focus on these individual bits, experiment with them and tweak them accordingly. The following sections describe which different solutions we attempted, why we tried them and what results they yielded.

2.1 Library choice

Our team decided to use Keras, which is a library running on top of TensorFlow. This decision was guided by the information that Keras is fairly simple to use and for us beginners, this seemed to be the best option.

2.2 Normalization

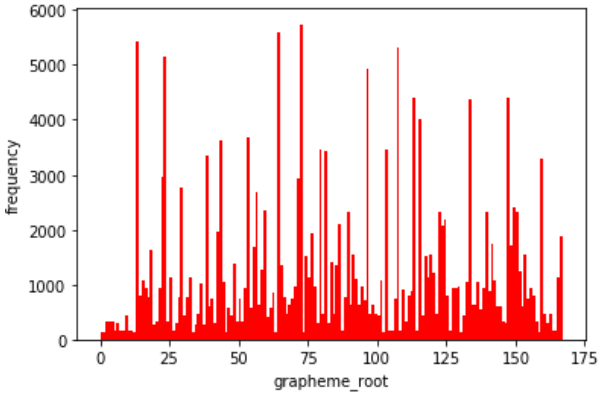
We tried to normalize the data using two different methods. The first one was a simple resize function, which performed relatively well compared to the second method, which we eventually used. The difference was about 2% improvement, averaging all of the three language constituents. The latter solution utilised inter-area interpolation with an image cropping combined.

2.3 Balancing data sets

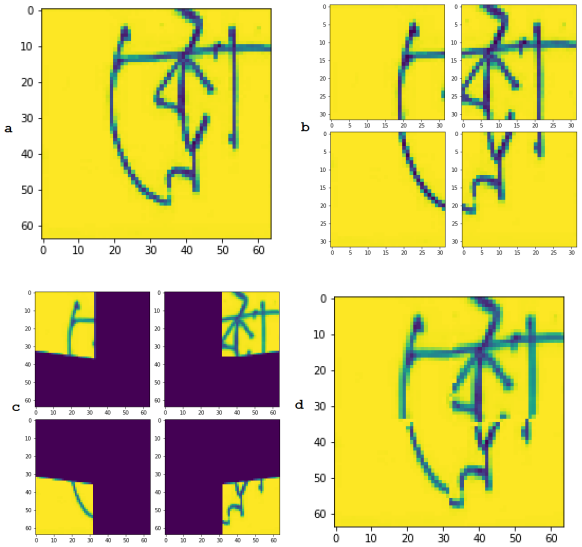
As was shown in the flashtalks and figure 1a, the dataset provided for this competition is imbalanced. It is possible to address this by creating more images of classes that are underrepresented in the data. Adding class weights or removing over represented images is also a possibility, but as overfitting was already a problem in this competition when using low amounts of images, we choose to augment the data with Elastic grid transformation in order to solve the problem of imbalance.

Elastic grid transformation [3] is a data augmentation technique that is especially relevant for character recognition. As we could not find an existing python implementation, we implemented a simplified version of

[1] https://www.kaggle.com/c/bengaliai-cv19/overview
[2] https://www.kaggle.com/
[3] https://www.youtube.com/watch?v=gZmobeGL0Yg&list=PLZbbT5o_s2xq7LwI2y8_QtvuXZedL6tQU
[4] https://www.kaggle.com/c/bengaliai-cv19/notebooks



(a) The unbalance in the grapheme_root class.



(b) Elastic grid transformation

Figure 1: Data imbalance and augmentation

n.o. unique input images	Accuracy - No augmentation	Accuracy - Elastic grid transformation
1000	4%	57%
2000	8%	-
25000	66%	82%

Table 1: Accuracy of a simple model with and without elastic grid transformation

elastic grid transformation. The original image is split into four equal parts, which are stretched and then put back together. Due to the way the stretch function works, some pixels are lost, causing small parts of the character to disappear. The process is visualised in figure 1b.

When testing this implementation in a simple model, it is clear that the transformation drastically reduces overfitting. The results of this test are presented in table 1. Elastic grid transformation was implemented in a kernel that was not compatible with the kernel that we submitted, so unfortunately this technique did not contribute to our final result.

2.4 Model

U-Net[1] is a convolutional network mainly used for image segmentation. U-Net was one of the first models that was implemented due to its recent popularity and the fact that coincidentally, one of our group members had recently worked with a U-Net which made the implementation simple. However, the realization came quickly that the Bengali problem is a classification problem and not a segmentation problem. This means that the output of the network is not the same size as the input so upsampling, which is half of the network, would not be necessary. Therefore, U-Net did not seem to be a logical solution to the Bengali problem, however it gave us an idea how a network can be optimised.

This knowledge was then used during implementation of two other models - ResNet and DenseNet, which have proven to be one of the most effective solutions for character recognition [2]. More specifically, we tried out ResNet-18 and DenseNet-121. Both yielded accuracy of predicted score of more than 0.9. Eventually we decided to stick to ResNet-18 and optimize it because of its slightly better performance.

2.5 Training

Training was done on each of the four datasets in order. This order does not need to be shuffled given the even class distributions amongst the training sets. To save time during development, proposed versions of networks were first trained on only one of the four data sets to quickly detect and correct networks that would not converge.

Number of epochs, image size, batch size, callbacks and training split were among others means to improve the network. These variables were changed gradually - adjust one parameter, train the model and in case of

improvement, keep tweaking the parameter until either the score is lower, or limits of the network, such as available RAM or GPU training time, are reached.

Number of epochs was set to 30, which proved to be sufficient for our solutions. Introduction of callbacks (ReduceLearningOnPlateau, EarlyStopping, ModelCheckpoint) prevented the network from overfitting, helped adjusting the learning rate and consequently, terminated the training of a dataset before reaching the limit of 30 epochs. Introduction of these callbacks decreased our training times and boosted our network's accuracy by approximately 2%.

We started with an image size of 32x32, getting a root accuracy of 66% after training one dataset. After we changed image size to 64x64, the root accuracy sky-rocketed to 88%. We explained this dramatic change to ourselves in the following way, that some spatial features got lost by resizing the input images too much. 92x92 performed even better, however, due to the competition's hardware resources limitation, we would constantly run out of memory, even with a batch size of 32. Eventually we settled for 64x64 images with a batch size of 32, which yielded higher accuracy and lower loss than batch sizes of 64, 128 or even 256.

The last parameter we played around with was training split size. A training split of 0.08 seemed to perform best for our solution. For instance, a split of 0.15 was deteriorating our network. Smaller splits were not tested as we assumed too small validation data sample would worsen the accuracy.

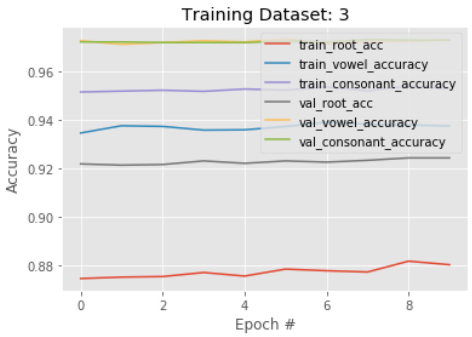
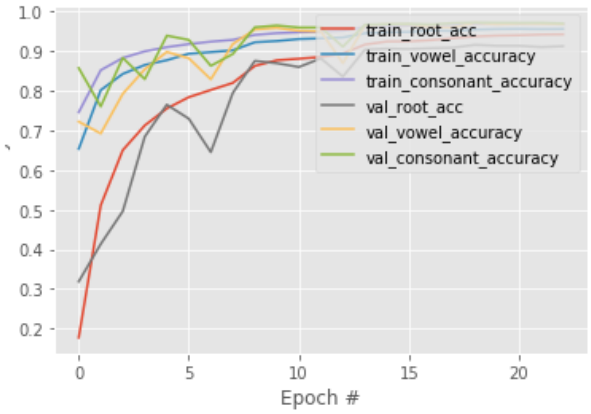


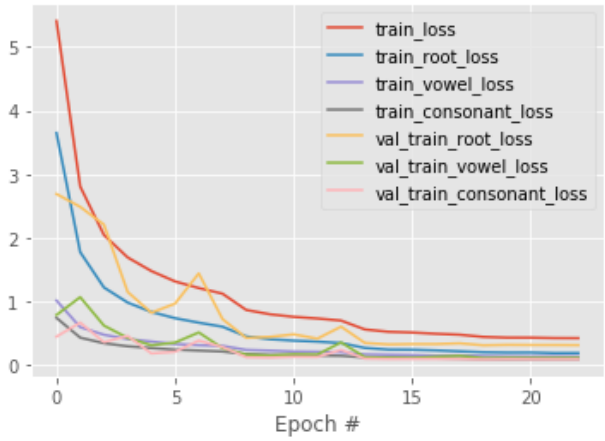
Figure 2: Accuracy of our final model on the last of four datasets

3 Results

Since only the final score is available for all data (the private score), to evaluate the results we use the public score. The public score is based on only 46% of the test data. The private score for our best solution is 0.8784, and the public score is 0.9295. From the three target classes, the grapheme root was the most difficult to classify. This class is predicted with 92% accuracy by our model, as can be seen in figure 2. The vowel and consonant are both predicted with an accuracy of 97%. The training accuracy and loss over the first file of the dataset are plotted in figure 3.



(a) The accuracy on the first file of the dataset



(b) The loss on the first file of the dataset

Figure 3: Data imbalance and augmentation

4 Evaluation of the process

This first competition was a large project that required a good
working method and working process. This was even more important as the team ran into trouble at the start of the project, causing a delay that had to be compensated for later. The delay was caused by the difficulty to install Keras locally and David getting locked out of his Kaggle account, which was resolved after intervention of David van Leeuwen. In this section, author contributions for this project as well as the working method are discussed, and a short evaluation of the supervision is given.

4.1 Author contributions

Jan integrated parts of the network to a working result, played around with the U-Net and then based on the learnings tweaked the ResNet-18 model. In total he trained around 20 different solutions. David worked on balancing the dataset and creating a kernel from scratch. Koen implemented a U-Net and a DenseNet-121 with transfer learning.

4.2 Working method

Our team had regular meetings every Tuesday to discuss progress and problems. We also tried to tackle hard problems together during the meeting, while at other moments of the week work was done on individual tasks. There were two general approaches: top down, using the existing notebooks on Kaggle to explore techniques and bottom up, building a model from scratch. In the last two weeks, the latter method was discontinued as learning the basics became less important. Besides, the models created by combining the existing notebooks were much more interesting and more complete. The models were mostly trained on Kaggle and Google Cloud Platform due to the powerful GPUs. Local setups were mainly used for learning purposes, preprocessing and minor network tweaks.

4.3 Evaluation of supervision

The meetings with our team coach (Chris) were useful to get a good start and get more information on different aspects of the project. This also holds for the meeting with the teachers, which resulted in a better overview of what we had accomplished and could focus on in the future.

References

[1] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[2] Friedhelm Schwenker, Md Zahangir Alom, Paheding Sidike, Mahmudul Hasan, Tarek M. Taha, and Vijayan K. Asari. Handwritten bangla character recognition using the state-of-the-art deep convolutional neural networks. https://doi.org/10.1155/2018/6747098, 2018.

[3] C. Wigington, S. Stewart, B. Davis, B. Barrett, B. Price, and S. Cohen. Data augmentation for recognition of handwritten words and lines using a cnn-lstm network, Nov 2017.