BACKPROP GROUP REPORT

ON

APPLICATION OF NEURAL NETWORKS

TO

KUZUSHIJI MNIST DATASET

TEAM MEMBERS:

HUGO COUSSENS, KEVIN FUNG, FALOLA YUSUF, NITCHAKUL PIPITVEJ

SUBMITTED TO: PROFESSOR OLIVIER DUBRULE AND DR LUCAS MOSSER

MAY 2019

REPORT OUTLINE

1. Abstract

This work is based on the application of Convolutional Neural Networks for prediction on the Kuzushiji-MNIST data set. There have been previous attempts to classify this dataset, with the highest accuracy recorded so far being 98.90%, which involved using ResNet18 + VGG ensemble. Three networks were created from LeNet and AlexNet to classify the dataset. One of the modified AlexNet was identified as our best network and was used for the final prediction. We introduced dropout to control overfitting and performed 2D batch normalisation to all convolution layers. Data augmentation and pre-processing were performed to create more input dataset to train and make the models more robust. Also, random-grid search was done to select the best hyperparameters for validation using Adams optimiser. We recorded a final accuracy of 97.914% on the test set, surpassing the GTA baseline of 96.228%. For future research, other networks like ResNet, VGG along with ensembling could be considered for better accuracy.

2. Introduction

Handwritten digit recognition is an important problem in optical character recognition, and it can be used as a test case for theories of pattern recognition and machine learning algorithms. To promote research of machine learning and pattern recognition, several standard databases have emerged. The handwritten digits are preprocessed, including segmentation and normalization, so that researchers can compare recognition results of their techniques on a common basis as well as reduce the workload [1].

This work is based on application of Neural Networks for prediction on the Kuzushiji-MNIST data set. Cursive Kuzushiji is a Japanese script that has been used for over 1000 years, without common standards, and sometimes included dozens of styles and formats for the same word. In the 19th century, Japan reformed its official language and writing system and standardized it, and over time Kuzushiji became extinct, causing millions of documents of Japanese culture and history to be inaccessible for most people [2]. The Kuzushiji dataset is created by the National Institute of Japanese Literature (NIJL) and is curated by the Center for Open Data in the Humanities (CODH). In 2014, NIJL and other institutes begun a national project to digitize about 300,000 old Japanese books, transcribing some of them, and sharing them as open data for promoting international collaboration [3]. Kuzushiji MNIST dataset contains 70,000 (28x28) grayscale images, labelled into 10 classes.

There have been previous attempts to classify this dataset. Prior machine learning techniques used include; Nearest Neighbour, Support Vector Machine, Simple Convolution Neural Network, ResNet18. The highest accuracy recorded so far is 98.90%, which involved using ResNet18 + VGG ensemble [4]. In the work, we had 60,000 image samples, along with corresponding labels, provided as training dataset, from which we had to create a validation set. We were only provided the image samples of the test set and no access to labels. Hence, we made our decision on the performance of our models on the validation set. The Network Architecture used in the work was a modified AlexNet. The performance of this network was enhanced with robust data augmentation/pre-processing and efficient random-grid search to get optimal hyperparameters.

3. Data Pre-processing and Data Augmentation:

A first glance at the data reveals the large variation of the data in each class. In the image below, the first column represents the 'standard font' for each of the classes. Looking along each row the variation in the representation of each classes is shockingly different. Its is clear this will cause some difficulties in achieving accurate predictions for K-mnist images.

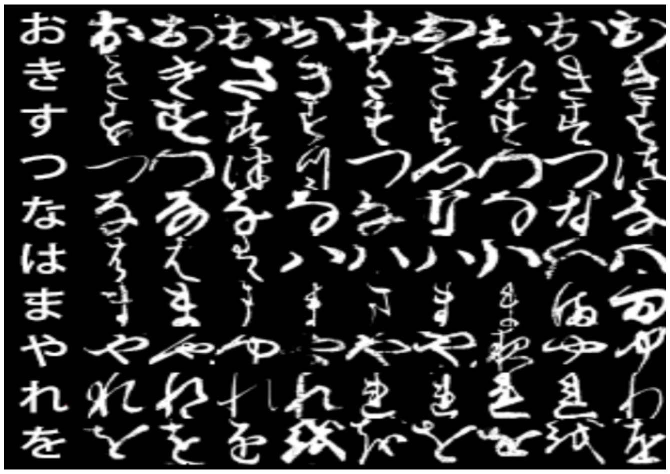


Figure 1: K-MNIST letters

3.1 Augmentations

The method of data augmentation used was based on the CustomImageTensorDataset as used in the workshop classes. Within this class, we implemented 3 different types of image transform as shown in the figure below. Each of these transforms were applied independently with a set probability. Every time an image was called for training it had a single transform applied. The probabilities shown to give the best result are as follows: Rotation (10deg) 20%, Random-resize-crop (0.7-1x crop size) 30%, Random affine (stretching to random degree +/- 10 degrees) 40%, no transform 10%. After each transform is applied the images are normalised based on the calculated mean/std of the training dataset.

Horizontal and vertical flips were not considered as they would remove the relevant spatial relationships of the data. Brightening was tested and found to bring no increase in performance of the model. This is probably because this has the effect of removing resolution of data after normalisation of the image.

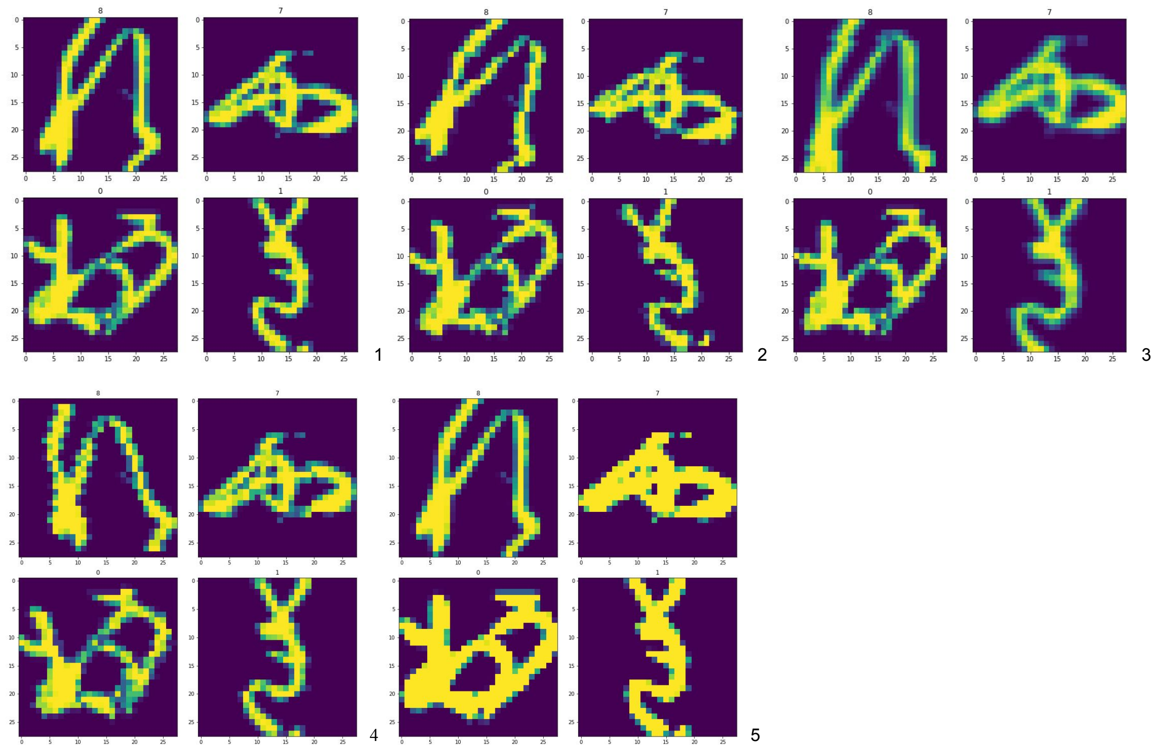


Figure 2: Letters from data augmentation

3.2 Class distributions:

[1] Normal
[2] Rotated
[3] Random-Resize-Crop
[4] Random Shear
[5] Brightening (NOT USED)

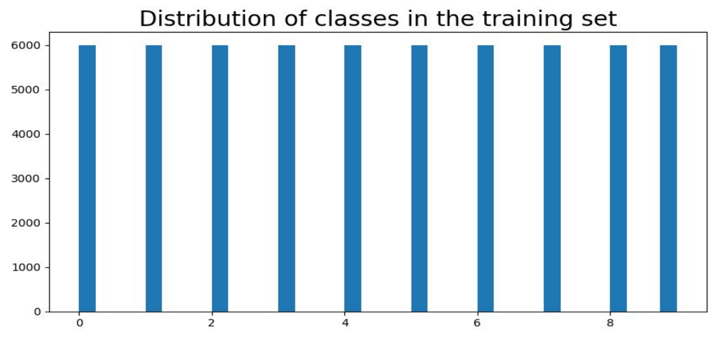


Figure 3: Distribution of classes in training set

A confirmatory initial check revealed that all the data are successfully evenly stratified into the training set. Next step is to train a few of our models to see which class is most commonly mis-classified. After seeing that two classes (2 and 4) stood out as particularly poorly classified on the validation. We tried an approach to increasing the proportion of these in the training data. We increased the number of class 4 three-fold and class 2, two-fold. We observed that the misclassifications of class 2 and 4 both decreased and the training accuracy increased. This was however at the sacrifice of other classes prediction performance. Overall, we opted to not utilise this method in the final prediction model.

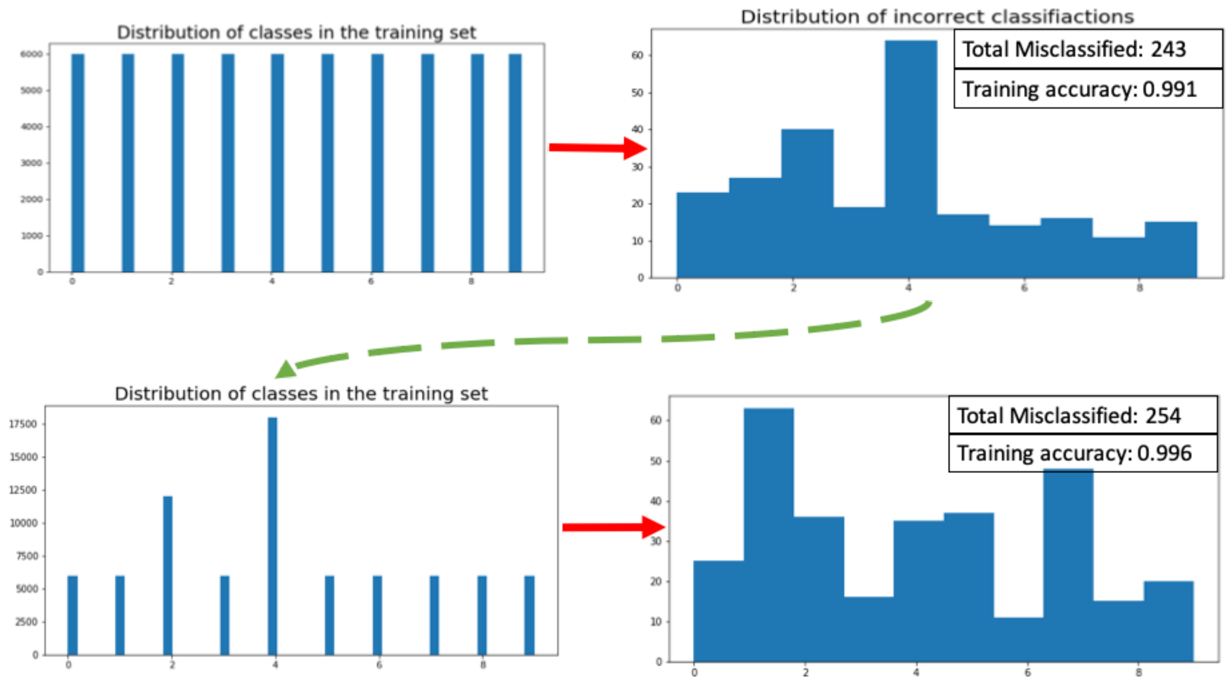


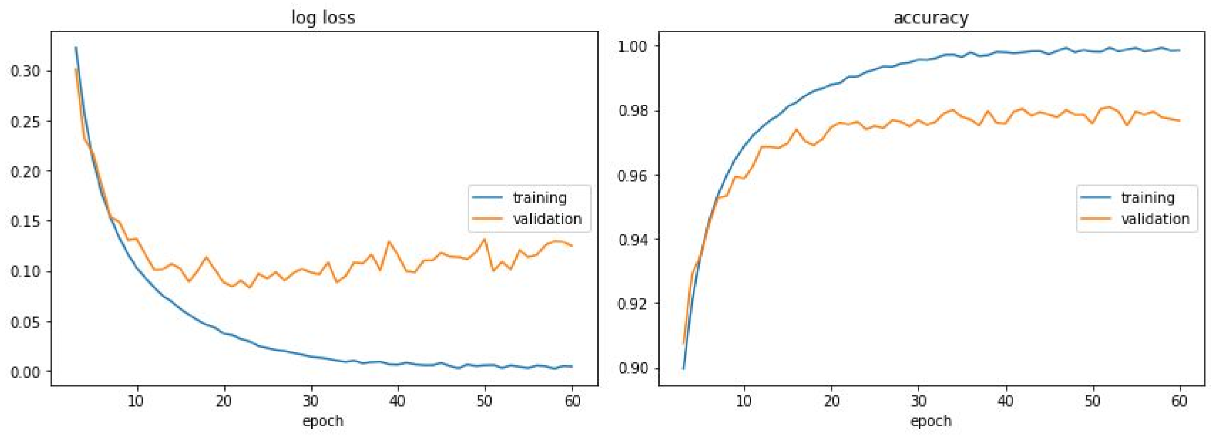
Figure 4: Distribution of misclassified classes in validation set

4. Network architecture

4.1 AlexNet Layers Exploration

Having found that Lenet5 models were inferior to AlexNet, we decide to investigate further in the architecture itself. The original AlexNet model was designed for 224x224 pixel images. In terms of finding the right balance between speed and prediction accuracy, using 60 million parameters was clearly unsuitable for 24x24 pixel K-MNIST images.

Our first approach in setting a baseline model were to assume the same original architecture but with a scaling down of the number of channels and parameters in our layers. We opted to use a scaling factor of 1/8. Cross validating with the unaugmented dataset gives the following:



```
log loss:
training   (min:    0.002, max:    1.009, cur:    0.004)
validation (min:    0.083, max:    0.577, cur:    0.125)

accuracy:
training   (min:    0.660, max:    0.999, cur:    0.999)
validation (min:    0.815, max:    0.981, cur:    0.977)
```

Figure 5: AlexNet BaseLine Model

Figure 5 above shows a clear overfit to the training set with a train accuracy of 0.999 and validation accuracy of 0.977 with the difference being 0.022. With the overall aim to achieve the best prediction in mind, we have several potential modifications to reduce overfitting and also increase the predictive power of our model:

1. Increasing the number of convolutional layers
2. Increasing the number of fully connected (FC) layers
3. Increasing the number of filters
4. Increasing the number of parameters in FC layers
5. Choosing sensible filter parameters during convolution

We ran multiple independent experiments in changing the AlexNet model to test the effects of each potential modification (See appendix for graphical results). Our first experiment (Appendix 1) of adding another convolutional layer to AlexNet can be seen to help alleviate some overfitting with the difference between training and validation accuracies dropping to 0.020. For the second experiment (Appendix 2) by adding another fully connected layer we achieve more removal of overfitting with a train and validation accuracy difference of 0.019. On the other hand, both experiments showed an overall decrease in predictive modelling power.

For the third experiment (Appendix 3), we double the number of filters in our convolutional layers and double the fully connected parameters used in our baseline AlexNet. The outcome shows underfitting instead with a final train and validation difference of 0.001. This shows that validation accuracy has increased significantly, and thus the improves predictive power of the model.

From this analysis, we experimented using the combination of the modifications (Appendix 4). Our first combined modification uses a scaling factor of ¼, with an additional convolutional layer and an additional 4 classification layers. The max validation accuracy of the model increases to 0.988 with no reduction in max train accuracy (0.999) compared to our baseline AlexNet. As we haven't dropped our training accuracy, it makes sense to utilise more techniques to regularise our model which includes data augmentation.

In addition to adding our layers and parameters, we had to consider the effect in which the filter sizes and padding would affect the separation of the image's features. It was found that starting with a large filter size at the start of the convolutional layers and smoothly transitioning into smaller sizes would help filter out the features in the images.

4.2 Final Architecture

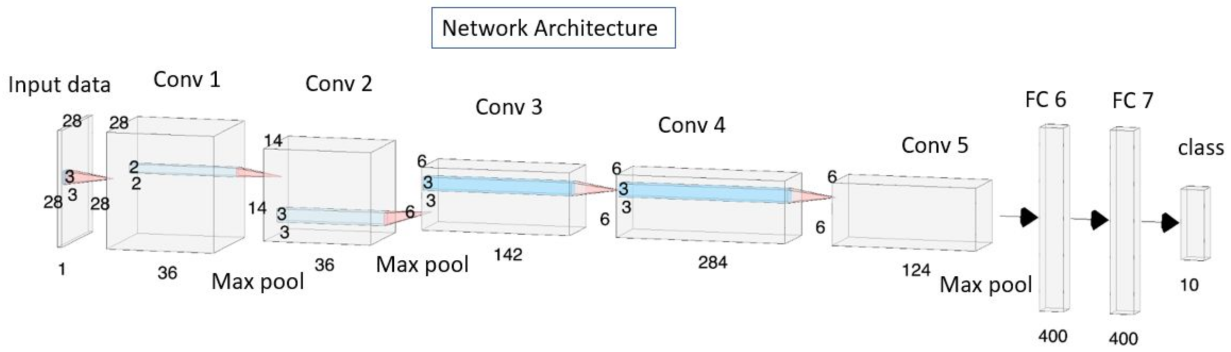
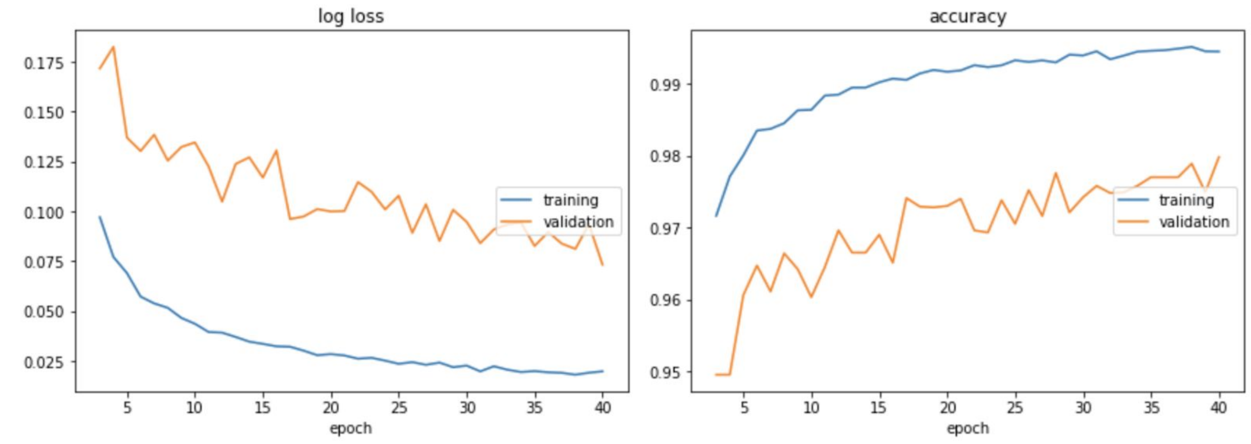


Figure 6: Final Network Architecture

Based on the AlexNet, the final architecture that was used is shown in Figure 6. The hyperparameter used has the learning rate of 7.e-05, batch size of 64, test batch size of 1000, 40 epochs, and used Adam optimizer algorithm. The difference between the two final submitted models is the dropout rate and the weight decay, one with 50 percent and no weight decay, and another with 70 percent with weight decay of 7.e-04. The models are uploaded on the GitHub repository. Below is the log loss and accuracy graph obtained from the model.



```
log loss:
training   (min:    0.018, max:    0.614, cur:    0.020)
validation (min:    0.073, max:    0.291, cur:    0.073)

accuracy:
training   (min:    0.808, max:    0.995, cur:    0.995)
validation (min:    0.913, max:    0.980, cur:    0.980)
```

Figure 7: log-loss and accuracy plot from final model

5. Training Approach

There are many strategies that were looked at in order to overcome underfitting and overfitting problem of the models. Apart from the architecture modifications mentioned previously, the other hyperparameters were altered to find the best combination for the selected model.

The approach we use to train and improve our models uses an iterative optimizer to repeatedly forward and backpropagate our model for a given number of epochs. From a large range of optimizers, we chose and compared between Stochastic gradient descent and ADAM optimizer. It was observed that the ADAM optimizer provides a better result and thus used in the final code.

In order to make sure that each model training process was consistent, the seed was fixed for every run. The learning rate and the weight decay was chosen from the random-grid search method, which chose the optimum hyperparameters. This will be further discussed in the Hyperparameters Selection section. The test batch size used was 1000, and batch size was 64 samples.

6. Validation Strategy and Hyperparameters Selection

6.1 Validation Strategy

There are two methods of validation in the final program, K-fold cross validation, and Hold-out validation. The two methods are used in different process in order to ensure that the best performance can be met.

For the Hold-out validation method, while it takes considerably less time to compute, it may not represent an accurate accuracy of the model. This method is used during the model fine-tuning phase to quickly generate each model's accuracy result before being sent forward to evaluate with K-fold cross validation.

As for the K-fold cross validation, the evaluated accuracy is more robust and lowers the variance from using only one training set and validating set. The resulting accuracy will represent a more accurate result and as it would not be based entirely on just one set of training dataset. The downside of this is that it will take more time to compute as it will be computed as many time as the number of folds presented. With this in mind, the K-fold cross validation is only performed once we have found the model that performs well enough. The model with high K-fold cross validation accuracy result will then be submitted on the Kaggle competition page.

6.2 Hyperparameter Selection

After deciding to choose the popularly used ADAM optimizer for our training during exploration. We initially decide to set the L2 regularisation weight decays to 0.0 and to choose a learning rate of 1e-4 (a commonly used value).

In order to achieve further improvements in the model, we conduct an optimal weight decay and learning rate search for our ADAM optimizer. We considered two methods, Grid search and Random search. Given the limited time on the project, performing a grid search of every combination of the weight decay ranging from (1e-3 to 1e-7 with alternating 1s and 5s) and learning rate ranging from (1e-3 to 1e-6 with alternating 1s and 5s also) would take up to 80 individual searches. Hence we use a mixed search strategy to find the optimal parameters in relatively less time with considerably better hyperparameters to before.

The strategy involves running a random search parameter of up to 15 iterations and then explore the domain around the best random search parameters using grid search. With this we go from a naive grid search of 80 to 20 searches depending on the type of grid search performed. The found hyperparameters for our best model were a learning rate of 7e-05 and a weight decay of 7e-04 with a validation prediction of 0.9941.

7. Discussion of results

Having created three different networks, we wrote an ensembling method to evaluate its performance on the evaluation set. Classes 4 and 2 were noted to be poorly predicted in the validation of models. Redistribution of these in the training data with the aim of increasing the importance of these in model parameters did not lead to a better score on the prediction set.The result of increasing the training accuracy but decreasing validation means this method did nothing but overfit the training data. The results of combining the networks created didn’t yield a higher accuracy than our final network on the validation set, the reason for this is not fully understood why and further investigation would be aimed at trying the ensembling with a wider array of more varied model architectures.

Furthermore, using our selected modified network model, we obtained obtained an accuracy of 97.914% on the private leaderboard, and 98.266% on the public leaderboard.

8. Conclusion

Three networks were created from LeNet and AlexNet to classify the Kuzushiji MNIST dataset. One of the modified AlexNet was identified as our best network and was used for the final prediction. The basic AlexNet was modified by altering the input and output channels of the layers. We introduced dropout to control overfitting and performed 2D batch normalisation to all convolution layers. Data augmentation and pre-processing were done to create more input dataset to train and make the models more robust. Also, random-grid search was done to select the best hyperparameters for validation using both Adams and SGD optimisers. We opted for Adams as it proved more accurate than SGD for our final network.

Lastly, based on our selected network, we recorded a final accuracy of 97.914% on the test set, surpassing the GTA baseline of 96.228%. For future purpose, we would consider spending more time on adding more layers to the networks and ensembling to get a better result. Other network, like ResNet, not treated in class could also be explored.

9. References

1. Zhu, W. (2019). Classification of MNIST Handwritten Digit Database using Neural Network. *Research School of Computer Science, Australian National University*.

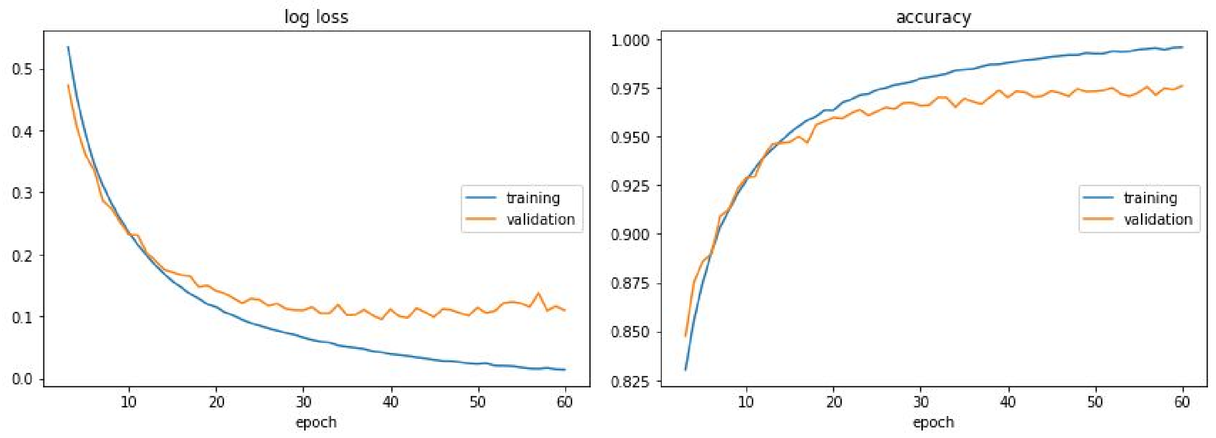
2. Horev, R. (2018). *Kuzushiji-MNIST - Japanese Literature Alternative Dataset for Deep Learning Tasks*. [online] towards data science. Available at: https://towardsdatascience.com/kuzushiji-mnist-japanese-literature-alternative-dataset-for-deep-learning-tasks-d48ae3f5395b [Accessed 22 May 2019].

3. Wu, H. (2018). CNN-Based Recognition of Handwritten Digits in MNIST Database. *Research School of Computer Science, The Australia National University, Canberra*.

4. GitHub. (2019). [online] Available at: 4. https://github.com/rois-codh/kmnist [Accessed 24 May 2019].

10. Appendices

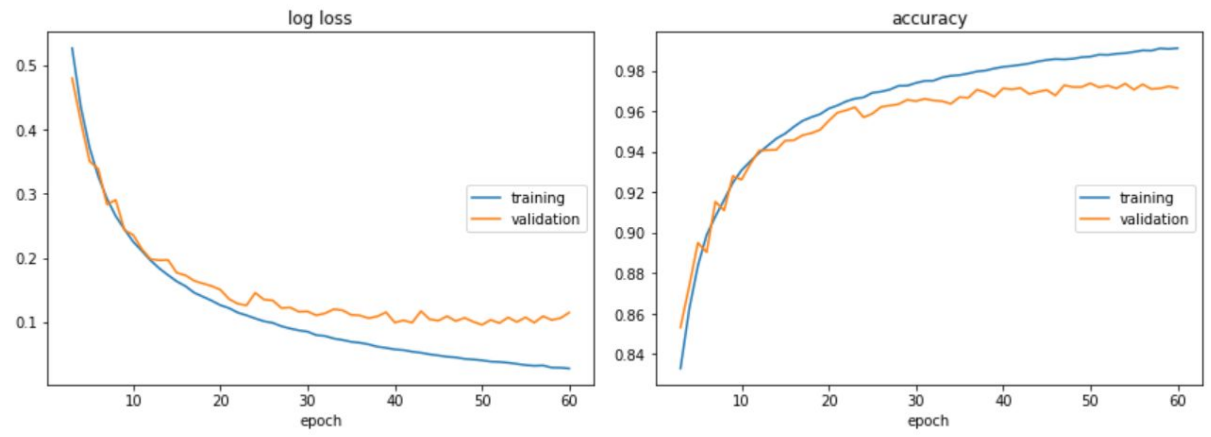
Appendix 1 - Baseline AlexNet with Additional Convolutional Layer



```
log loss:
training   (min:    0.014, max:    1.335, cur:    0.014)
validation (min:    0.095, max:    0.840, cur:    0.110)

accuracy:
training   (min:    0.542, max:    0.996, cur:    0.996)
validation (min:    0.724, max:    0.976, cur:    0.976)
```

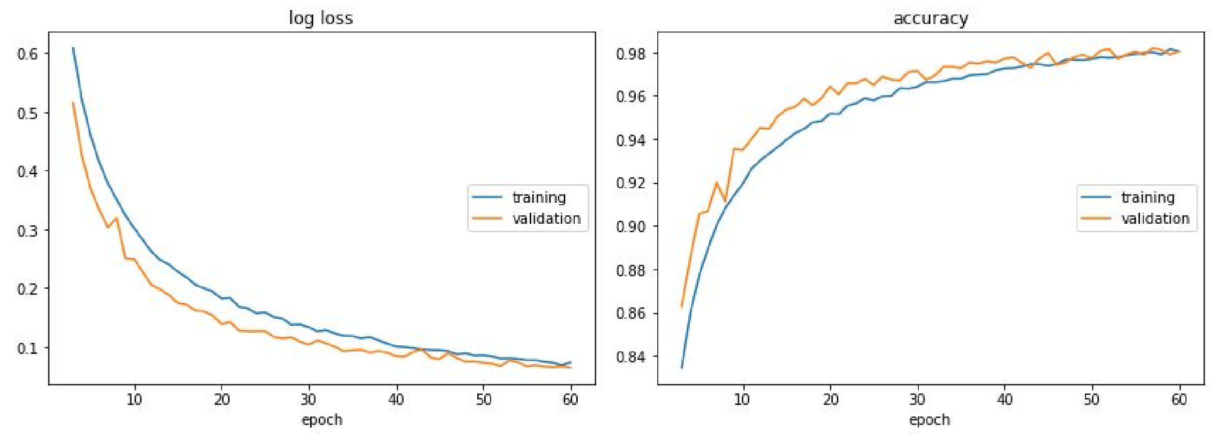
Appendix 2 - Baseline AlexNet with Additional Class Layer



```
log loss:
training   (min:    0.028, max:    1.255, cur:    0.028)
validation (min:    0.096, max:    0.802, cur:    0.115)

accuracy:
training   (min:    0.563, max:    0.991, cur:    0.991)
validation (min:    0.744, max:    0.974, cur:    0.972)
```

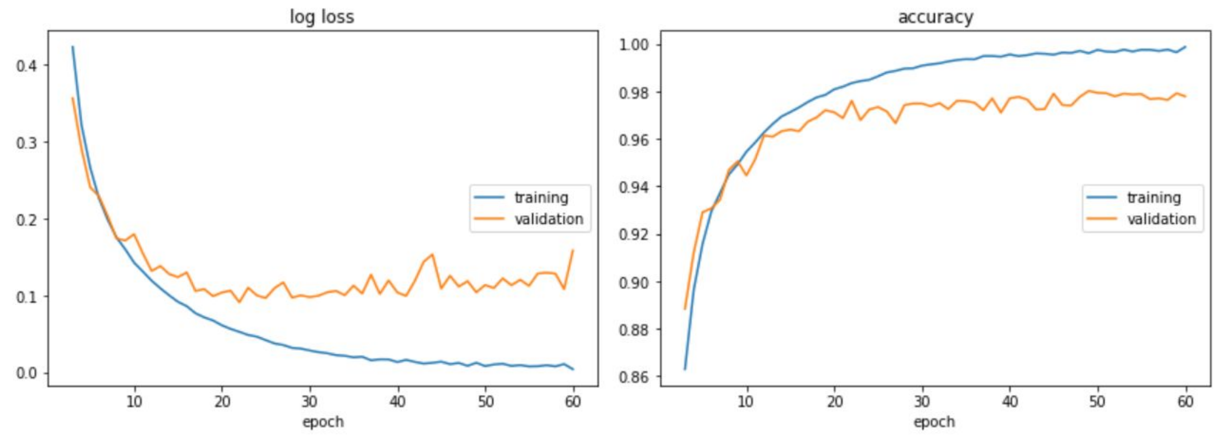
Appendix 3 - Baseline AlexNet with Scale Factor of 2



```
log loss:
training   (min:    0.069, max:    1.324, cur:    0.073)
validation (min:    0.065, max:    0.851, cur:    0.065)

accuracy:
training   (min:    0.592, max:    0.982, cur:    0.980)
validation (min:    0.764, max:    0.982, cur:    0.981)
```

Appendix 4 - Baseline AlexNet with Combined Modifications



```
log loss:
training   (min:    0.004, max:    1.372, cur:    0.004)
validation (min:    0.091, max:    0.864, cur:    0.159)

accuracy:
training   (min:    0.504, max:    0.999, cur:    0.999)
validation (min:    0.709, max:    0.980, cur:    0.978)
```