KUZUSHIJI-MNIST CLASSIFIER - SOFTMAX REPORT

ACSE-4

Richard Boyne
MSc Applied Computational Science and Engineering
Imperial College London
`rmb115@ic.ac.uk`

Mattia Guerri
MSc Applied Computational Science and Engineering
Imperial College London
`mg1618@ic.ac.uk`

Mao Jiaye
MSc Applied Computational Science and Engineering
Imperial College London
`jm2518@ic.ac.uk`

May 24, 2019

ABSTRACT

Our goal is to develop a neural network model to classify images from the Kuzushiji-MNIST dataset. We tested various convolutional architectures characterized by different numbers of parameters. Using the LeNet5 architectures we are able to obtain a validation set accuracy of 98%. In order to imrove on this, we turned to deepr structures, inspired by the AlexNet architectures. Given the high numebr of parametrs, these networks suffer from sever overfitting. In order to counter that, we use data augmentation techniques and neurons dropout. In addition we explore the effects of tuning hyper-parameters, as learning rate and weight decay. We also test the effect of batch normalization and different optimizers. We obtained few models perfoming equally well on the validation set and showing similar confusion matrices. We combined them in two ensembles and submit the predictions on Kaggle to be compared against the test set, reaching a top score of 0.989 and 0.988 and the public and private leaderboard, respectively.

***K*eywords** deep learning · CNNs · Kuzushiji-MNIST

1 Introduction

Kuzushiji-MNIST is an ensemble of Japanese characters images subdivided in 10 classes in the same manner as other MNIST sets in that images are monochromatic and 28×28 pixels. Kuzushiji-MNIST has been shown to be more troublesome to classify though, with current state of the art networks[1] performing at 98.9% test accuracy. Our goal is to produce a neural network algorithm able to achieve a test accuracy of at least 98.6%.
When dealing with images, convolutional neural networks (CNNs) have proven to be a powerful method, in particular for their 2d geometry preserving the features depicted in the image. Classical examples of CNNs architectures which we focus on here are the LeNet5 (1998)[2], trained on MNIST) and the AlexNet(2012)[3], trained on ImageNet LSVRC-2010. LeNet5 is a relatively simple model with 3 convolutional and 2 fully connected layers. AlexNet is a deeper network with 5 convolutional layers and two fully connect layers.
We tested the LeNet5 architecture and variations of it. We then modified the AlexNet architecture obtaining various models that differ for number of layers, layers size and depth. In order to increase the performance of our models we tested various techniques, specifically various kinds and intensity of data augmentation, neurons dropout, batch normalization and k-fold validation. In addition, we explored the effect of changing changing hyper parameters like learning rate and weight decay, and we tested the effect of different optimizers. We obtained various models with similar performance on the validation set and we combined them to produce predictions to be submitted on Kaggle for the comparison with the test set.

2 Data Pre-Processing & Augmentation

Images are originally a single pixel value, integer between 0 and 255. Since it has been shown[4] that often neural networks perform better on data that has been centered to a mean of zero and normalised by the standard deviation, we

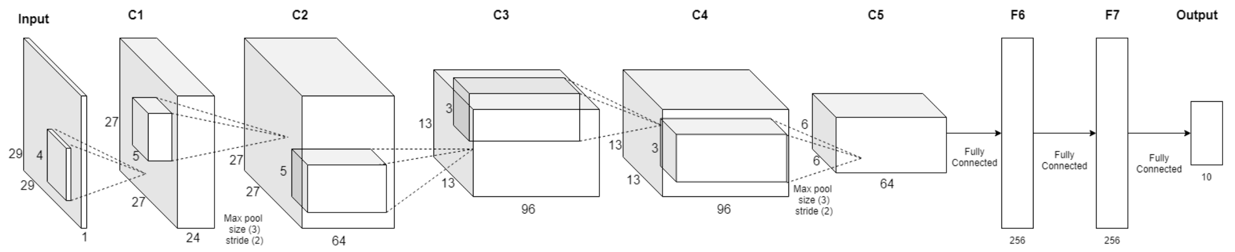

Figure 1: AlexNet 5 Structure, very similar to the original[3] with three convolutional and two fully connected layers. This network has $\sim 960,000$ parameters

normalise the images computing the mean and standard deviation for the set on which we train the model, and applying these values to the validation set.
The number of images in training data is limited to 60000. The LeNet5 network, which is a relatively simle one, already has 62,000 parameters, making this an under-defined problem. Deeper models, like those in the style of AlexNet, have from millions to tens of millions of parameters, giving an even more sever overfitting problem. To counter the issue, a relatively simple and yet effective method is represented by data augmentation.
For this data set a crop and rotation seem reasonable due to the different angles, sizes as brush strokes of the images. For this Pytorch transforms RandomCrop and RandomRotation were implemented at training time as they are cheap to perform and prevent the need for augmented storage. This increases our effective dataset by an order of 10^4, but compromised the IID assumption (independent and identically distributed)[5], so caution is needed. For the higher capacity AlexNet models heavier augmentation was required to reduce over fitting, for which altering brightness, contrast, shift and shear were implemented too.

3 Network Architecture

Throughout this project we used the ReLU activation function as this has been repeatedly shown to improve both the learning rates and performances of most neural networks[6]. Though several specific architectures and variations were tested there are three significant ones which performed well.

3.1 LeNet5

Practically unchanged from the original LeNet5 used on MNIST network. Though this network is significantly smaller than most used to analyse this data set, it is still a valid choice as both a baseline here and as a starting point to experiment with different techniques (such as different augmentations or adding additional layers).

3.2 AlexNet5

To extract more complex patterns from the images, deeper models were required, in particular, we start from the AlexNet network. The modifications of the original model follow two rationales:

- AlexNet was meant to work with $3 \times 224 \times 224$ images, subdivided in 1000 classes. Our dataset has lower resolution and monochromatic images subdivided in ten classes. We therefore changed the size and depth on the input convolutional layers, and decreased the number of neurons in the later fully connected layers.
- To reduce the computational run-time to train the model (AlexNet takes over a week to train) we reduced the number of channels and fully connected neurons so as to keep overall structure but reduce the number of parameters we refer to our modified AlexNet structure as reduced AlexNet.

The resultant network is shown in figure 1.

3.3 AlexNet7

It became apparent that even deeper networks would perform better in identifying the more complex symbols previous networks could not correctly distinguish. Here a balance needed to be struck between increasing network depth without increasing the number of parameters to a training time that would be impractical. Hence we experimented with adding a few extra convolutional layers and neurons, specifically:

- two convolutional layers between C1 and C2 in figure 1.
- increasing the first fully connected layer to 512 neuron.

This resulted in a network with ~ 1.9 million parameters.

4 Training Approach

The training took around $15min$ for LeNet5 and $1 - 3hr$ on the reduced AlexNet, though also varied significantly depending on the specific hyper parameter choice. To assist with testing different models many were trained simultaneously on different GPUs, though more than one GPU was never used to train a single model.

4.1 Dropout & Batch Normalization

The first fully connected layer are responsible for most of the parameters count, and hence, neurons develop co-dependency amongst each other during training which curbs the individual power of each neuron leading to over-fitting of training data. In order to reduce over-fitting, we decided to use dropout in the last two linear layers (in the same manner as[3]). This has the affect of reducing over training by reducing the number of parameters optimised during each back propagation. It also forces the CNN to learn how to classify certain features separately, essentially forming an ensemble of different architectures within a single network which are averaged across to give a more reliable classification. During training, dropout roughly doubles the number of iterations required to converge.
Just as networks perform best on centered and normalized input, there performance can be improved by doing the same process between convolutional layers. This reduces the amount by what the hidden unit values shift around (covariance shift) and forces each layer of a network to learn its own patterns, less relying on the features learned by other layers. We normalize each batch during the forward pass (after the activation function). Combining dropout and batch normalization, we found an increase in validation accuracy of 0.6% on AlexNet5 with light data augmentation (only random rotations and cropping).

4.2 Loss Function & Final Activation

Throughout this project we used the cross entropy loss function and the softmax function for the output neurons as these are the common choices for multi-classification networks.

4.3 Optimizer

Many variations on the gradient descent method are available to optimize the loss function. We tested three different optimizers, SGD (Stochastic Gradient Descent), RMSprop and Adam. We usually obtain a faster convergence with Adam, though SGD was the common choice to due to stability in the validation accuracy.

5 Validation Approach

We divided the training set in an actual training set and a validation set, with a ratio of 9:1. Furthermore, in order to obtain a more robust evaluation of the model performance, we applied k-fold validation. Formally this consists of subdividing the training dataset in a k number of partitions, and train the model on a stack of k-1 of them, using the last partition as validation set. The training is repeated for each possible training validation set. This has the advantage of ensuring the validation accuracy is not specific to any one training validation split, increasing the confidence in the model. However, as training time is significantly large (considering this a one week project), hence only two of the ten possible k folds were trained, which is enough to gain some confidence in our models.
The use of different optimizers had a surprising effect on the model predictions. With all other things kept constant, prediction disagreements between SGD and Adam training was between $0.6\% - 1.6\%$, in spite of having very similar (0.1 different) or the same validation accuracies. Having no particular justification to choose the predictions obtained with one optimizer over the other one, we combined them in an ensemble by averaging their output pdfs. For submission ensembles of more than two models a democratic combination of the predictions was used with ties being settled randomly.

6 Hyper-parameters Tuning

As explained above, we explored different architectures in terms of number of layers, layers size and depth. For what concern the hyper parameters, a line search method was used to test different values of learning rate, momentum and weight decay, with the SGD optimiser. To be able to do this in reasonable time the AlexNet5 network was used for the comparison. This later proved to be an issue as the optimal hyper-parameters are very sensitive to the exact network architecture, hence these optimal values were only useful when training AlexNet5. Furthermore, the optimiser used is also sensitive to these (where cross applicable), so again only SGD were valid for these hyper parameters. For the specific case of AlexNet5 coupled with the SGD optimizer, we found a learning rate of 5^{-3}, momentum of 0.8 and weight decay of 10^{-4} gave an overall validation increase of 0.2% which was not particularly significant compared to data augmentation, network and dropout hyper parameters. When using the Adam optimizer, we chose a learning rate value of 10^{-4}. This values is commonly used when employing such an optimizer.
We concluded that generally the choice of optimal hyper parameters would need further research and likely have small impacts compared to other factors, like network depth and data augmentation. The impact on training time was not compared as the performance of GPUs available varied highly, however it is likely that we would have improved our training time with a more in depth analysis of the issue.
Several other parameters, having different degrees of influence on the models performance, can be tuned. These are for example, which layers are subjected to neurons dropout and with what probability, which layers are subjected to batch normalization, kind and intensity of data augmentation. We have chosen these parameters essentially adopting a trial and error approach to give the networks we present here. In case of the intensity of data augmentation, we tuned this together with the layers of parameters in the networks, in particular, when increasing the augmentation intensity, for example the angle of random rotation, we also increased the total number of parameters in the networks, in order to avoid bias and extract the maximum possible amount of information from the augmented data. The affect of changing dropout probabilities and batch normalisation layers was not thoroughly investigated due to restriction in time, though these are likely significant.

7 Results

We obtain the best performance on the test set using two ensembles (indicated by the private score in figure 2). These are composed of four models; two AlexNet5 architecture and two AlexNet7 architectures with different optimisers each (Adam and SGD, see above). Since the training sets were more augmented in the AlexNet7 models they were run for more epochs, 175 and 200 for the SGD and Adam optimizer respectively, to ensure that all obtainable information from the training set were fully extracted. Although the only difference between the two models is represented by the optimizer, their predictions present significant differences, with a disagreement of approximately 1.2%. Due to time restraints the AlexNet7 models were not trained on the full training set, where as the AlexNet5 models were trained on the full training sets for 100 epochs each. These four individual models showed similar performance on the validation set of 99.7% and also similar confusion matrices, while slightly disagreeing among each other hence the ensemble was expected to be more reliable.
The first ensemble in figure 2 is a voting combination of all these networks, which all reached a validation set accuracy of 99.7%. The ensemble reached a test set score of 98.7%. The second ensemble is an averaging of two AlexNet7 PDFs, which achieved the same validation accuracy as the other models but a higher test performance of 98.8%. This suggests an improvement of AlexNet7 over AlexNet5, meaning that in this case a deeper network is better provided the data augmentation is increased to prevent over fitting. However the improvements are marginal (less than 10 images) whilst the parameters have increase two fold and the training time comparatively. It is possible that using an increased drop out chance rather than increasing data augmentation could have a similar or better affect on the reduction in over fitting needed for more parameters and is something we would recommend exploring further.

8 Conclusion

We developed several neural networks to classify images from the Kuzushiji-MNIST dataset. Our models are based on two well-known convolutional structures, LeNet5 and AlexNet. We modify the architectures adding more layers, changing their size and, in case of the convolutional layers, their depth. We adopted various techniques to improve the performance of our models, for example, data augmentation to counter overfitting, neurons dropout and batch normalization. We validate our models on validation sets, also adopting k-fold validation. The top performance we achieved on the validation set is roughly 99.7 %. This was reached by various methods. We combined their predictions and submitted to Kaggle for evaluation on the test set, reaching a top score of 0.989 and 0.988 and the public and private leaderboard, respectively.

Model	**Training Acc.** (%)	**Validation Acc.** (%)	**Private Score** (%)	**Agreement** (w.r.t Ensemble)
Ensemble (AlexNet5 and AlexNet7)	99.5	99.7	0.98685	1.00
Ensemble (AlexNet7)	99.6	99.7	0.98800	0.992785
AlexNet5 (Dropout, Data Aug, Batch Normalization, Full Train)	~99	99.6	0.97900	0.988737
AlexNet5 (Dropout, Data Aug, Batch Normalization)	98.8	99.7	0.98557	0.989688
AlexNet5 (Data Aug)	99.4	99.0	0.97785	-
LeNet5 (Data Aug)	-	98.8	0.96028	-
LeNet5	-	98.0	0.93985	-

Figure 2: Table comparing the accuracies of different models and there agreement with each other in their test predictions. Private Score is the test accuracy on 70% of the test dataset (7,000 images). Both AlexNet5 and AlexNet7 were trained with two different optimisers. The Ensemble all 4 of these gives the upper ensemble and just the 2 AlexNet7 give the lower one. Both the ensembles were our final submissions with the AlexNet7 ensemble giving the best private score accuracy.

References

[1] T. Clanuwat, M. Bober-Irizar, A. Kitamoto, A. Lamb, K. Yamamoto, and D. Ha, "Deep learning for classical japanese literature," *CoRR*, vol. abs/1812.01718, 2018.

[2] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, pp. 2278–2324, Nov 1998.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, (USA), pp. 1097–1105, Curran Associates Inc., 2012.

[4] J. Bjorck, C. P. Gomes, and B. Selman, "Understanding batch normalization," in *NeurIPS*, 2018.

[5] T. M. Cover and J. A. Thomas, *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. New York, NY, USA: Wiley-Interscience, 2006.

[6] D. Pedamonti, "Comparison of non-linear activation functions for deep neural networks on mnist classification task," *CoRR*, vol. abs/1804.02763, 2018.