Kannada MNIST Report

Oraz Ospanov Valeriya Rudikova Yegor Polikarpov

***Abstract*—The goal of this report is to use various deep learning algorithms to classify a new handwritten digits-dataset, Kannada-MNIST, which is a simple extension to the classic MNIST competition. For this dataset, we used Pytorch framework to build convolutional neural network (CNN) and deep neural network (DNN) and observed variations in performances of two models. The performance is evaluated on the categorization accuracy of predictions, by changing various parameters of two networks. The results show better performance of the convolutional neural network (CNN) approach with: 80% accuracy for the CNN, while DNN accuracy is 60%**

***Index Terms*—Epochs, Hidden Layers, Stochastic Gradient Descent,Adam optimization, Backpropagation.**

I. Introduction

Kannada is the official language of the state of Karnataka in India with approximately 40 million native speakers [1]. This competition is of the same format as the standard digit recognition MNIST competition in terms of how the data is structured, though with diffence in how it approaches runs of Kernels.

Libraries used: PyTorch, MatPlotLib, NumPy

II. Dataset

The main dataset for training consists of 60000 28x28 gray images of digits in Kannada script, Figure 1 shows a sample image of a Kannada digit.

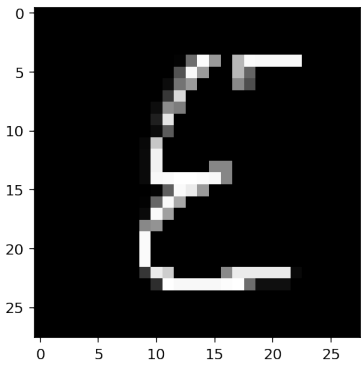

Fig. 1. Sample image from Kannada-MNIST dataset

The additional Dig-MNIST dataset, which was used for testing our network as the out-of-domain dataset, consists of 10240 28x28 gray scale images. This dataset was purposely collected from non-native speakers writings, with smaller resolution and scanned with different scanner compared to the main ones, therefore comprising a more challenging set to analyze. As the author of the competition suggests, the training accuracy achieved on main dataset is approximately 98%, whereas for the Dig-MNIST the value achieved is 76.1%1001[2].

III. Deep Neural Network Approach

We created the model with two hidden layers and for every layer the Rectified Linear unit activation function was used. The reason of using the RELu function is that it produced better results in MNIST datasets. For obtaining probabilities of classes we applied LogSoftMax function instead of Softmax, because in practice LogSoftMax implementation is more efficient; and in order to convert from logarithms to actual probabilities, produced by LogSoftMax, exponent funtionn exp() was applied. For training our model we used pytorch's optim package, specifically we used Adam optimization in DNN.

A. Changing network parameters

For improving the accuracy of DNN we changed several parameters of our network, such as activation function, optimization parameters, number of hidden layers and so on. We experimented with several activation functions such as sigmoid and RELu functions. On average the use of Relu function showed 5% increase in accuracy of our network. By checking with other participants in Kannada-MNIST community we confirmed that RELu function was more favourable.

We used two types of optimization in training our model.The choice of SGD optimization was due to that it just showed better results in accuracy of DNN. The other optimization was Adam optimization. Provided below are the graphs of loss functions during training of DNN, where figure 2 shows loss for SGD optimization, and figure 3 shows loss for Adam optimization, where x-axis is the number of epochs and y-axis loss function. Generally SGD is ann iterative method of optimizing function, and it is regarded as stochastic since it replaces the actual gradient calculated from the entire data set by an estimate calculated from a randomly selected subset of the data [3]

One of the main problems was the choice of hidden layers on our network. We just didn't have the starting point for choosing the parameters of hidden layers: number of layers, number of nodes in intermediate hidden layers. In general a rule of thumb is to choose 2 hidden layers for MNIST datasets, as was done across many other participants in Kannada-MNIST.

Also one of the main roles in improving the results of our network was the number of epochs used. For training DNN

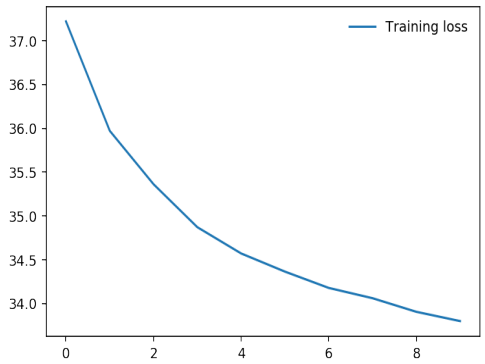

Fig. 2. Loss functions of DNN with SGD optimization

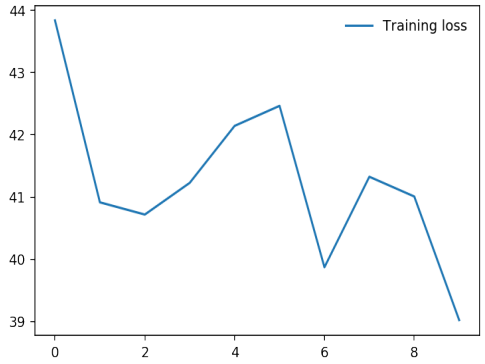

Fig. 3. Loss functions of DNN with Adam optimization

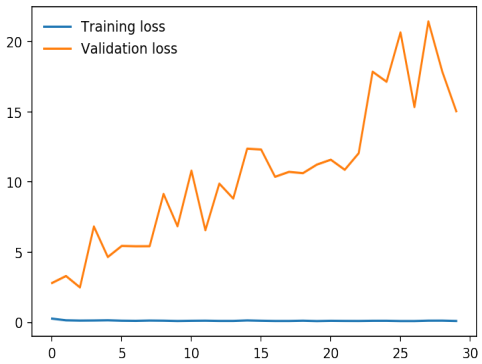

Fig. 4. Loss functions of DNN without Dropout

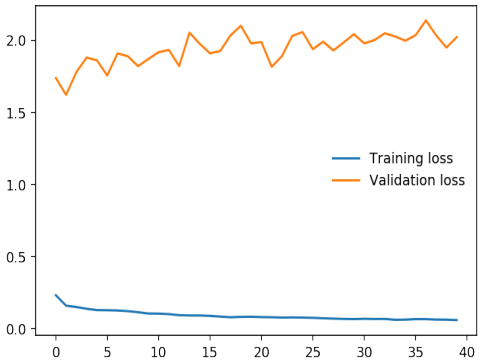

Fig. 5. Improved Loss functions of DNN with Dropout

30 epochs were used, which resulted in overall 60% accuracy of DNN.

B. Testing and Optimization

Figure 4 shows loss functions for training and testing datasets. We can see as the number epochs grows the loss for testing increases, which shows the sign of overfitting. We applied the method of dropout to reduce the overfitting, where we randomly drop some of the nodes in hidden layers. Through the use of nn.Dropout module provided by pytorch package. As an input to dropout module we used 20% probability that some node will be dropped. This resulted in the following graph of losses given in figure 5. Here we can see that the trend doesn't increase so rapidly in testing loss graph as in the previous graph, nonetheless the testing loss is still bad.

IV. Convolutional Neural Network Approach

Convolutional Neural Network (CNN for short) that consists of several convolutional layers, in which a series of filters (kernel) is slid over the input data. The filter is a matrix with weights, this filter is multiplied with the corresponding value of input pixels of the respective dimensions, then added up to comprise a single grid cell of the output. Our model consists of 5 hidden layers of convolutional layers, with MaxPooling of kernel size 2 applied after the 2nd, 4th and the output convolutional layers. At the output, the number of the output channels reaches 90 ($1 \rightarrow 30, 30 \rightarrow 30, 30 \rightarrow 60, 60 \rightarrow 60, 60 \rightarrow 90, 90 \rightarrow 90$). The convolutional layers are followed by 3 fully connected layers with the respective dropout of 0.5 and 0.3 after the 1st and the 2nd layer in order to prevent overfitting. The number of dimensions is given by the following scheme: $90 * 3 * 3 \rightarrow 400, 400 \rightarrow 300, 300 \rightarrow 10$. Such number of layers was also based on initial guesses based on the commonly used models of other Kannada-MNIST participants and was proven practically effective for our model. Here, as before, the RELu activation function was used due to proven efficiency on the given dataset, same goes for LogSoftMax.

A. Optimization

The optimization algorithm used in this implementation is AdamW, a modification of Adam with weight decay. Adam optimization algorithm is an extension of the classical stohastic gradient descent, and a combination of **Adaptive Gradient Algoritm** and **Root Mean Square Propogation**.Taking the best from its predecessors, Adam borrows from the former the ability to deal with problems with sparse gradients; and from the latter the the ability to deal with problems with noisy data.

First presented in 2014 by Diedirk Kingma from OpenAI and Jimmy Ba in the paper called "Adam: A Method for Stohastic Optimization" [4], it gathered an number of

supporters in the folllowing years. The benefits of Adam are the following: easy to implement, efficient in computation and memory use, well-adapted for problems with large number of parameters and big amount of data, deals efficiently with very noisy gradients. One of the main differences between Adam and SGD is that the latter maintains a steady learning rate independent of weight; whereas the former finds the individual adaptive learning rates for different parameters based on both the average first moments(the mean) and the second moments of the gradients(uncentered variance). Figure 6 shows the comparison of Adam to other Optimization algorithms for a multilayer perceptron in terms of speed of training. [5] . The Adam optimization procedure has a

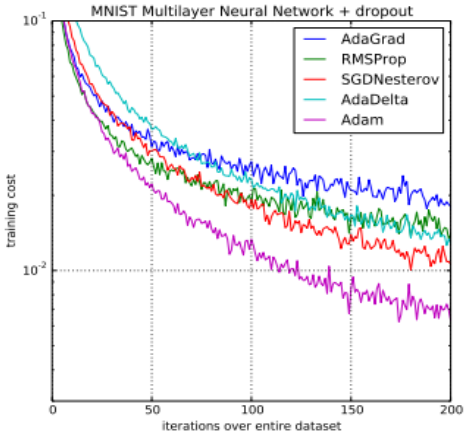

Fig. 6. Comparison of Adam to other Optimization algorithms for a multilayer perceptron. Source: [4]

flaw, since it does not always converge to steady solution at later stages. A cure to that was introducing Weight Decay Regularization into the weight update rule [6]. Figure 7 depicts the relation between learning rate and regularization method. In this implementation we practically compared the AdamW version of model to the SGD, and obtained a consistent increase in accuracy of about 4% for the former.

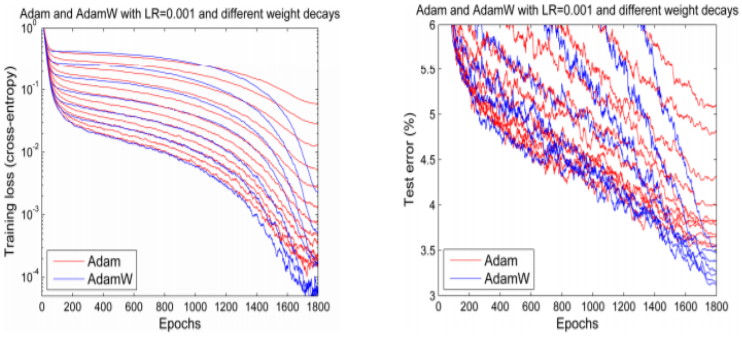

Fig. 7. Relation between learning rate and regularization method. Source: [6]

B. Accuracy

The following results were achieved for the number of epochs equal to 5:

- **Training total loss:**8.2681
- **Training accuracy:** 0.9961
- **Test dataset total loss:**1.05657
- **Test dataset accuracy:** 0.9950
- **Dig-MNIST dataset total loss:**77.6468
- **Dig-MNIST dataset accuracy:** 0.8068

It is apparent, that the accuracy for the Dig-MNIST dataset drops significantly, however, it is to be expected due to the reasons described earlier. Dig-MNIST comprises a more challenging data-set, since it was gather and processed under different conditions. The accuracy stated by the author of the challenge is shown to be on main dataset is approximately 98%, and for the Dig-MNIST the value ais $76.1\%1001$[2]; whereas here the values are 99.50% and 80.7% respectively. At the Figure 8 the training and validation losses are presented. At the Figure 9 the training and validation accuracies are shown.

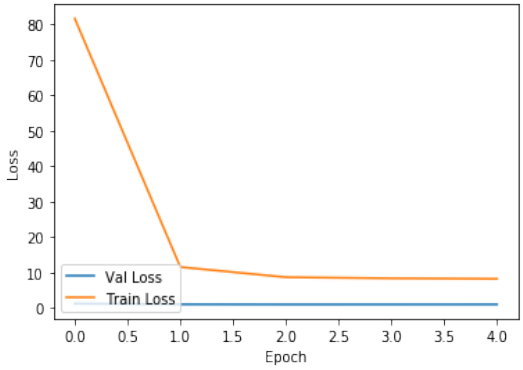

Fig. 8. Training and validation losses of CNN

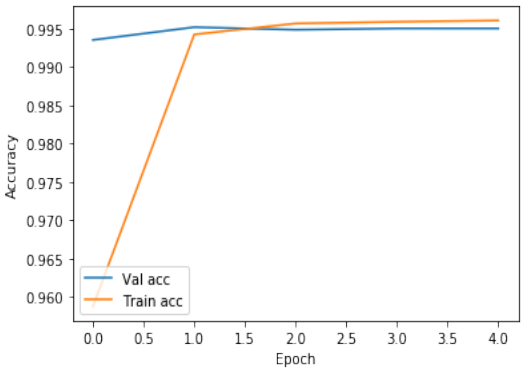

Fig. 9. Training and validation accuracy of CNN

As we can see, the validation accuracy slightly drops as the number of epochs increases, whilst the training accuracy increases. However, this result is acceptable, since a more radical change in training accuracy would imply overfitting,

which is not the case. The steadiness of the training and validation losses also implies that the model is already stable and has converged, the latter aspect could be the problem, if Adam instead of AdamW was implemented.

V. Conclusion

Two deep learning algorithms were implemented to classify Kannada script digits: deep neural networks and convolutional neural networks. As far as performance evaluation comes the convolutional neural network produced a better result in terms of predicting test images. Nevertheless, the main goal of this project was to learn and apply our skills of machine learning concepts by competing at Kannada-MNIST competition. In terms of results of competition - in order to produce a desirable result it takes time to optimize and train networks, but we got the main point of how to train and optimize the network, how to produce a dataset and so on. Also, the CNN was submitted to the competition, and scored 98% for categorization accuracy and currently is at 535th position out of 1005 [7].

References

[1] C. Chandramouli and R. General, "Census of india 2011," *Provisional Population Totals. New Delhi: Government of India*, 2011.

[2] V. U. Prabhu, "Kannada-mnist: A new handwritten digits dataset for the kannada language," *arXiv preprint arXiv:1908.01242*, 2019.

[3] S. Sra, S. Nowozin, and S. J. Wright, *Optimization for machine learning*. Mit Press, 2012.

[4] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[5] Y. S. Abu-Mostafa, M. Magdon-Ismail, and H.-T. Lin, *Learning from data*. AMLBook New York, NY, USA:, 2012, vol. 4.

[6] I. Loshchilov and F. Hutter, "Fixing weight decay regularization in adam," *CoRR*, vol. abs/1711.05101, 2017. [Online]. Available: http://arxiv.org/abs/1711.05101

[7] "Kannada mnist kaggle competition: Public leaderboard," https://www.kaggle.com/c/Kannada-MNIST/leaderboard, accessed: 2019-12-06.