Handwritten Letter Classification Using Small Convolutional Neural Network

Joseph Melville and Mert Canatan

***Abstract*— The goal of this paper is to create a model for handwritten letter classification task. We use "Class Dataset" which contain approximately 6,400 binary letter images of letters 'a', 'b', 'c', 'd', 'h', 'i', 'j', 'k'. We hypothesize that the use of sufficiently small networks will avoid over training of the network and provide higher accuracy results. We determine what is "sufficiently" simple by starting with a small network and adding layers with increasing regularization to avoid over training until performance starts to decline. We evaluate four separate methods to avoid over training, namely, dropout, inter layer normalization, affine transformations of the input data, and extension of the training data using a larger handwritten letter data set called EMNIST. The final method also assists in classifying a ninth class of "unknown" characters by extending the original "Class Dataset" with "unknown" characters from EMNIST. We then use the best performing network and successively pretrain it using the following steps: EMNIST with affine transformations, EMNIST, extended data set with affine transformations, extended data set, and finally the "Class Dataset". Using this method we achieve a final classification accuracy of ___ on the extended class data set.**

***Index Terms*— handwritten character recognition, convolutional neural network**

I. INTRODUCTION

IN this project we trained various small convolutional neural networks [1] to be used for classifying handwritten letters. There are two classification tasks to be accomplished. These are: The "easy" set which is the classification of handwritten letters 'a' and 'b' and the "hard" set which is the classification of handwritten letters 'a', 'b', 'c', 'd', 'h', 'i', 'j', 'k' and 'unknown'. Unknown class include the rest of the English alphabet. The dataset is used for training the model is the class dataset which have approximately 6,400 binary images containing letters 'a', 'b', 'c', 'd', 'h', 'i', 'j', 'k'. Challenges with this dataset include noisy training data (some images were rotated, some images did not include information about the letter, and some of the images were all black), limited number of data, and not having a training set for an unknown class. Since handwritten letter recognition is a well sourced field we have referenced datasets used for this task. These datasets include MNIST Database [2], EMNIST Dataset [3]. In the following sections we have explained our implementation and experiments respectively. In experiments section we also provided our results. Lastly we discussed what can be done to increase our model's validation accuracy in conclusion part.

II. IMPLEMENTATION

There are four parts in our implementation. These are:
1) Extending "Class Dataset" with EMNIST dataset,
2) Neural Network Structures,
3) Normalization, Dropout, and Affine Transformations, and
Firstly, we extend the "Class Dataset" EMNIST dataset by taking 1200 samples from the beginning of EMNIST dataset, which includes unknown characters, and concatenate to the "Class Dataset". Secondly we have introduced different neural network architectures. In these network architectures; all convolutional layers use a 3 by 3 kernel and a 2 by 2 max pooling kernel with stride 1, unless otherwise stated. All layers use rectified linear unit activation functions unless otherwise stated. Network classification error is calculated using the Pytorch CrossEntropyLoss function. Backpropogation is performed using the Pytorch Adam function [4] with a 1e-5 weight decay.

SimpleNet00:

- Four convolutions layers (3 to 6 to 12 to 24 to 48 channels)
- Four fully connected layers (1728 to 800 to 200 to 64 to 9 nodes). The final layer had no activation function.

SimpleNet00_norm:

- Three convolutions layers with normalization (3 to 6 to 12 to 24 to 48 channels)
- Three fully connected layers (1728 to 800 to 200 to 64 to 9 nodes). The final layer had no activation function.

SimpleNet0:

- Three convolutions layers (3 to 6 to 12 to 24 channels). The first convolutional layer max pool used a 4 by 4 kernel.
- Three fully connected layers (432 to 300 to 64 to 9 nodes). The final layer had no activation function.

SimpleNet1:

- Two convolutions layers (3 to 6 to 12 channels). The first convolutional layer max pool used a 4 by 4 kernel.

- Two fully connected layers (432 to 64 to 9 nodes). The final layer had no activation function.

SimpleNet1_drop:
- Two convolutions layers (3 to 6 to 12 channels). The first convolutional layer max pool used a 4 by 4 kernel.
- Two fully connected layers (432 to 64 to 9 nodes). Dropout used on first fully connected layer. The final layer had no activation function.

SimpleNet1_norm:
- Two convolutions layers with normalization (3 to 6 to 12 channels). The first convolutional layer max pool used a 4 by 4 kernel.
- Two fully connected layers (432 to 64 to 9 nodes). The final layer had no activation function.

SimpleNet2:
- One convolutions layer (3 to 6 channels). Max pool used a 5 by 5 kernel. A 5 by 5 convolutional kernel was used.
- Two fully connected layers (600 to 64 to 9 nodes). The final layer had no activation function.

SimpleNet2_norm:
- One convolutions layer with normalization (3 to 6 channels). Max pool used a 5 by 5 kernel. A 5 by 5 convolutional kernel was used.
- Two fully connected layers (600 to 64 to 9 nodes). The final layer had no activation function.

SimpleNet3:
- One convolutions layer (3 to 6 channels). Max pool used a 10 by 10 kernel. A 5 by 5 convolutional kernel was used.
- Two fully connected layers (600 to 9 nodes). No activation function used.

Lastly, to produce random affine transformations, we used the PyTorch RandomAffine function with a 90 degree random rotation, 10% vertical and horizontal random shifts, and 30% to 100% random scale. All dropout was accomplished using the PyTorch Dropout function, where 20% dropout was used. All inter-layer normalization was accomplished using the PyTorch GroupNorm function, when the entire layer was normalized in a single group.

III. EXPERIMENTS

Before performing the experiments we loaded the pre-processed data which include Class data, Class data labels, EMNIST data, and EMNIST data labels. We also have introduced extended data which is the concatenation of Class data and labels, EMNIST data and labels. We extended the labels to include the 'unknown' labels. Then we shuffled all the data and labels concurrently. We have split the dataset into 80% for training set and 20% for validation set. We performed four experiments to be able to obtain the best validation accuracy. These experiments came from the questions:
1) Which simple network provide best validation accuracy in data?
2) Does normalization or dropout help to reduce over-training and increase validation accuracy?
3) Will random affine transformations help to stop over-training and continue learning by expanding our training set?
4) Will using unknown data from the EMNIST dataset help to accurately classify unknown letters?

From Fig.1 to Fig. 11 you can see all the Neural Network structure training and validation results as well as their accuracy on the letter prediction. Table 1 shows the detailed results of our different neural network structures.[Table 1]

	1	2	3	4	5	Mean	STD
SimpleNet00_norm	94,4	94,67	93,75	95,52	95,01	94,67	0,66
SimpleNet00	94,47	94,14	94,54	94,73	94,01	94,38	0,30
SimpleNet0	94,21	93,75	94,67	94,73	94,21	94,31	0,40
SimpleNet1	91,77	91,77	93,22	92,03	93,09	92,38	0,72
SimpleNet2	91,05	91,31	92,5	92,43	91,57	91,77	0,66
SimpleNet3	87,89	87,82	89,01	87,56	89,01	88,26	0,70
SimpleNet1_drop	91,05	90,72	90,92	91,44	90,98	91,02	0,26
SimpleNet1_norm	91,84	92,43	93,81	93,68	93,29	93,01	0,85
SimpleNet2_norm	90,19	92,69	92,76	93,35	92,56	92,31	1,22
SimpleNet1 with Trans	86,11	86,24	86,57	85,19	88,22	86,47	1,11
SimpleNet1 with EMNIST	91,11	92,17	92,96	91,64	91,57	91,89	0,71

Table 1. Cross Validation Results of our Model Accuracies

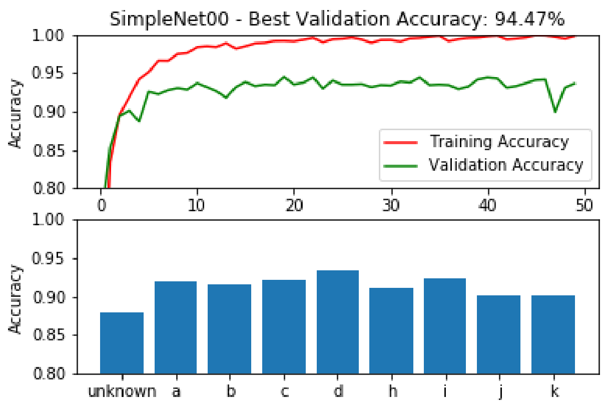

Fig. 1. SimpleNet00 Training and Validation Accuracy

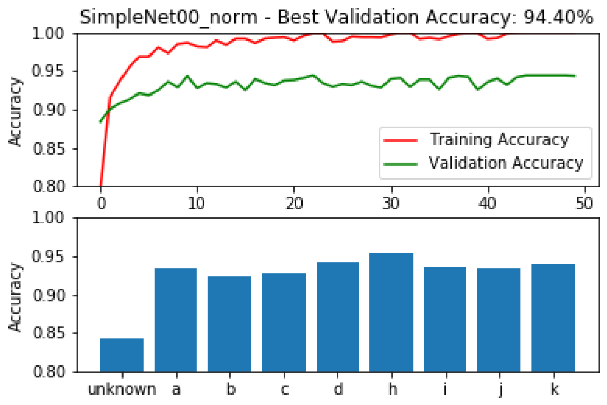

Fig. 2. SimpleNet00 with Normalization Training and Validation Accuracy

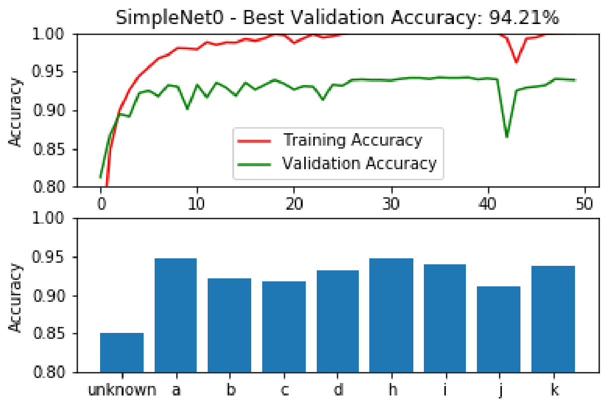

Fig. 3. SimpleNet0 with Transformations Training and Validation Accuracy

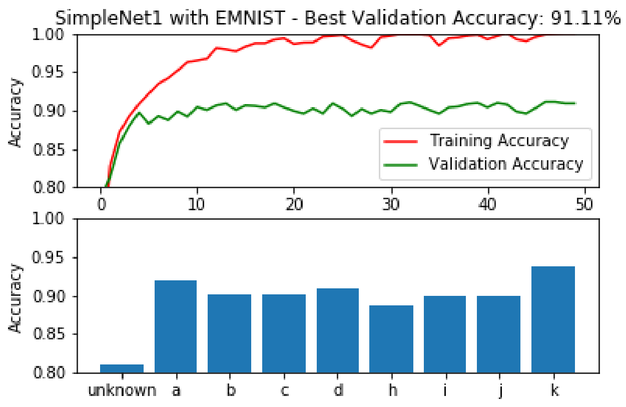

Fig. 4. SimpleNet1 with EMNIST Training and Validation Accuracy

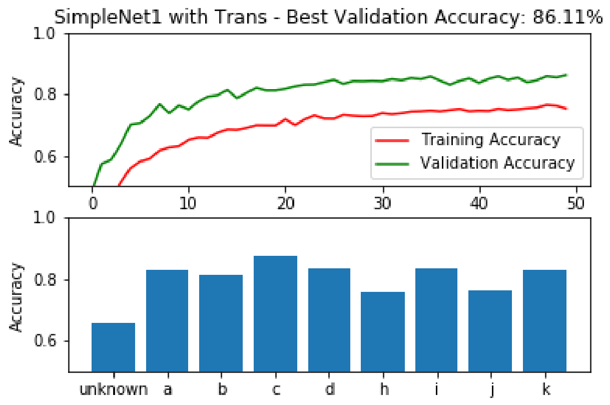

Fig. 5. SimpleNet1 with Transformations Training and Validation Accuracy

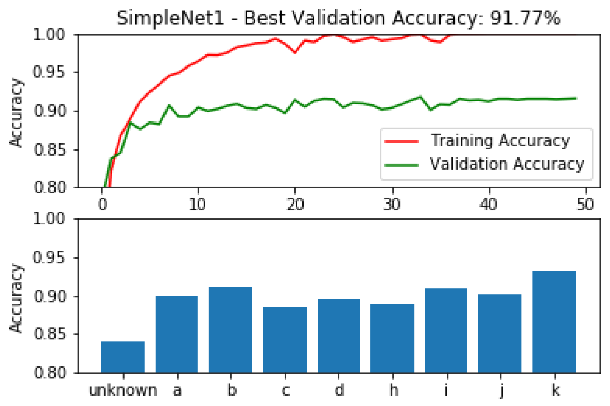

Fig. 6. SimpleNet1 Training and Validation Accuracy

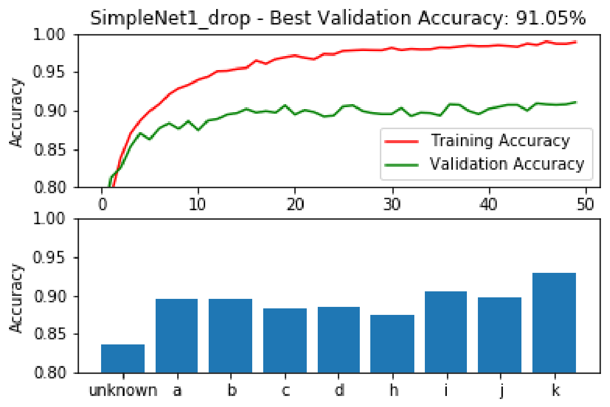

Fig. 7. SimpleNet1 with Dropout Training and Validation Accuracy

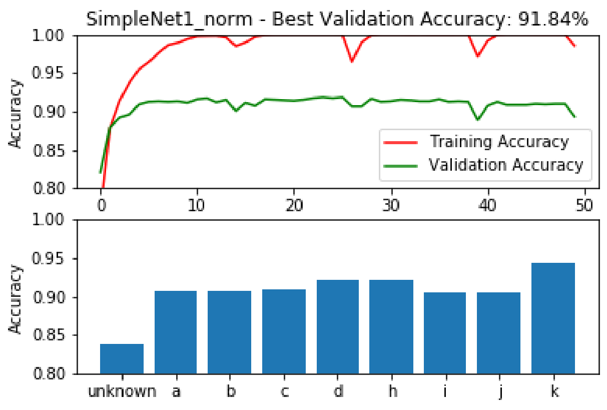

Fig. 8. SimpleNet1 with Normalization Training and Validation Accuracy

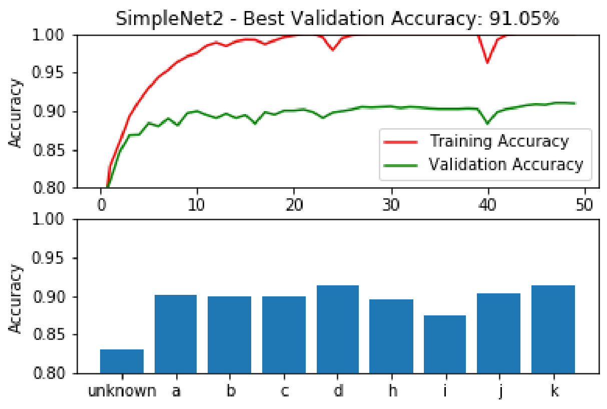

Fig. 9. SimpleNet2 Training and Validation Accuracy

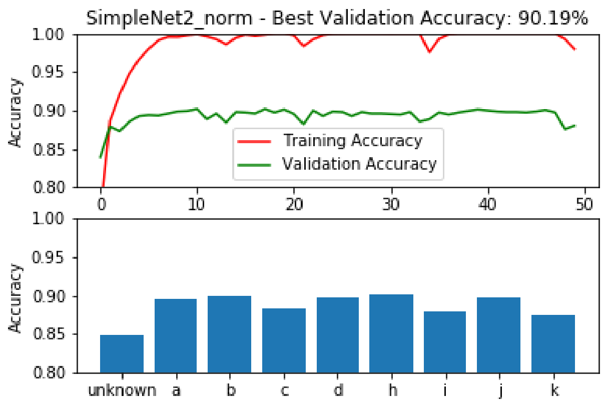

Fig. 10. SimpleNet2 with Normalization Training and Validation Accuracy

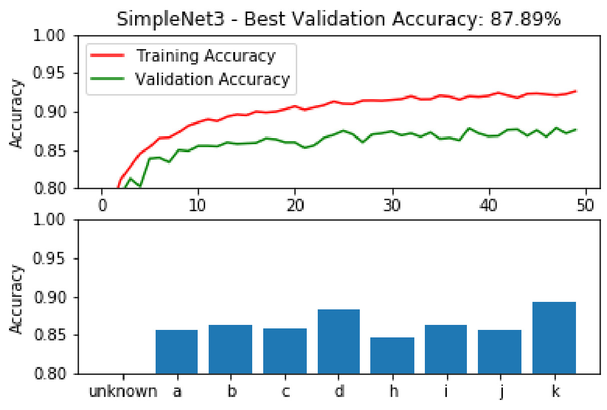

Fig. 11. SimpleNet3 Training and Validation Accuracy

IV. CONCLUSIONS

The main problem of using neural network for classifying handwritten characters is over-training which means memorizing every feature in the training set and not have a good accuracy on different test sets. In this project we showed that we can overcome this problem using small neural networks. Additionally, we implemented some modifications to these neural network structures to increase our model's validation accuracy. For classifying the unknown characters we have concatenated our dataset with EMNIST dataset since EMNIST dataset has all the English letters. As a result training our model with concatenated data gave us a promising result for classifying unknown characters. To increase validation accuracy of a model with neural network structure following things can be done:

1) Applying different affine transformations to neural network structure.
2) Using a different normalization method.
3) Using a different dropout technique.

REFERENCES

[1] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," Neural Computation, vol. 1, no. 4, pp. 541-551, Winter 1989.

[2] LeCun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-based learning applied to document recognition. Proc. IEEE 1998, 86, 2278–2324.

[3] Cohen, G.; Afshar, S.; Tapson, J.; van Schaik, A. EMNIST: An extension of MNIST to handwritten letters. arXiv 2017, arXiv:1702.05373.

[4]] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," in Int. Conf. Learn. Represent, 2015, pp. 1–13.