Implementation of a Convolution Neural Network for MNIST Classification

Mohammad Sareeb Hakak
Depaerment of ECE
North Carolina State University
Raleigh, USA
mhakak@ncsu.edu

***Abstract*— This report discusses the architecture of a Convolution Neural Network (CNN) used for classification on the MNIST Dataset. The CNN was built using Keras and the hyper parameters of the model were tuned to get an optimal performance using a Grid Search technique. The various learning curves for each of the hyper parameters are presented and the performance of the CNN for the hyper parameters are analyzed.**

Keywords—Convolution Neural Networks, Random Search, Hyper parameter tuning

I. Introduction

The Convolution Neural Networks are the most commonly used deep neural networks for classification tasks because of their ability to learn certain specific features of the data through creating feature maps. The CNN architecture consists of various Convolution layers which are mapped using 'Kernels' through an operation called 'Convolution'. This gives a very sparse network where each neuron is connected only to few other neurons in the next layer. Such a sparse network makes CNNs extremely efficient at learning specific features of the data. A convolution layer is also followed by a Pooling layer which reduces the network size further while preserving the feature data.

II. Architecture of the Network

The CNN used for the present classification task on the MNIST dataset consists of two Convolution layers, each followed by a Max Pooling Layer. These two Convolution layers are followed are followed by three fully-connected layers.

A. Details of the Architecture

The details of the complete architecture of the network:

Convolution Layer 1	**Pooling Layer 1**
Input Size: 28*28*1	Input Size:28*28*64
Output Size: 28*28*64	Output Size: 14*14*64
Filters: 64	Filters: 1
Filter Size:5*5	Filter Size: 2*2
Stride:1	Stride: 1
Activation: ReLU	Activation: ReLU
Zero Padding: 'Same'	Zero Padding: 'Same'

Convolution Layer 2	**Pooling Layer 2**
Input Size: 14*14*64	Input Size:14*14*128
Output Size: 14*14*128	Output Size: 7*7*128
Filters: 128	Filters: 1
Filter Size:3*3	Filter Size: 2*2
Stride:1	Stride: 1
Activation: ReLU	Activation: ReLU
Zero Padding: 'Same'	Zero Padding: 'Same'
Flatten	
Input Size: 7*7*128 Output Size: 6321	
Fully Connected Layer 1	
Input Size: 6321 Output Size: 256	
Fully Connected Layer 2	
Input Size: 256 Output Size: 128	
Fully Connected Layer 3	
Input Size: 128 Output Size: 10	

III. Hyper-Parameter tuning

Deep networks such as CNNs have a lot of hyper parameters to tune. However, due to time and computational resource constraints, the following hyper parameters were chosen to be optimized :

- Learning Rate : [1e-5, 1e-2]
- Batch Size : [32, 512]
- Dropout Rate : [0.2, 0.8]

A random search technique with a two-step approach was implemented for the hyper parameter tuning. In the first step, a random number generator was used to create a combination of the selected hyper parameters within the specified range. The model was then trained on these set of hyper parameters for 25 epochs each. Training was done for 4 such random models initially and the hyper parameters set which produced the best results was chosen. For the second step, a narrower search range was defined around this optimal point from the first step in the hyper parameter space and 4 more random models were generated. Of these 4 models, a narrower range was chosen as the following:

- Learning Rate : [0.005, 0.008]
- Batch Size : [170, 210]
- Dropout Rate : [0.4, 0.510]

The performance of the different models was evaluated based on their mean validation errors after convergence and the number of epochs it took to converge. It was observed from the random search that the model with the following hyper parameters produced the best accuracy and performance:

- Learning Rate : 0.0005
- Batch Size : 180
- Dropout Rate : 0.440

The dropout was applied only to the fully connected layers.

Step 1 – Random Search	1	2	3	4
Learning Rate	9.038901e-05	5.14429400e-03	7.73391887e-03	8.70557258e-03
Batch Size	93	170	188	240
Dropout	0.33275297	0.39097065	0.50787003	0.52351996
Step 2 – Random Search	1	2	3	4
Learning Rate	0.00050241	0.00065418	0.00073195	0.00076113
Batch Size	180	186	196	198
Dropout	0.44005317	0.45884454	0.45929139	0.47495672

IV. Model Performance vs. Hyper-Parameters

Once the best model was found, the dependence of the model performance on the hyper parameters about the selected point in the hyper parameter space was studied. This was done by varying each of the hyper parameters in a specified range around the best point in the hyper parameter space while keeping the other hyper parameters constant. The variation of the model performance are plotted about the best point in the hyper parameter space. It was observed that there was not much statistically significant dependence of the model performance on the hyper parameters around the current optimal point. However, the performance seems to drop only slightly around the optimal hyper parameters point indicating a stable optima in the hyper parameter space.

The various plots which demonstrate the change in accuracy vs. the hyper parameters are as follows:

Random Search 1:

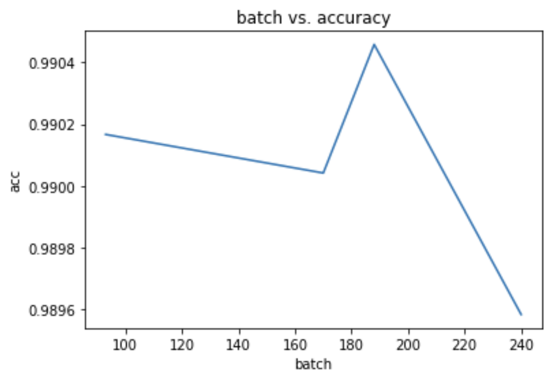

Fig.1: Batch vs. Accuracy

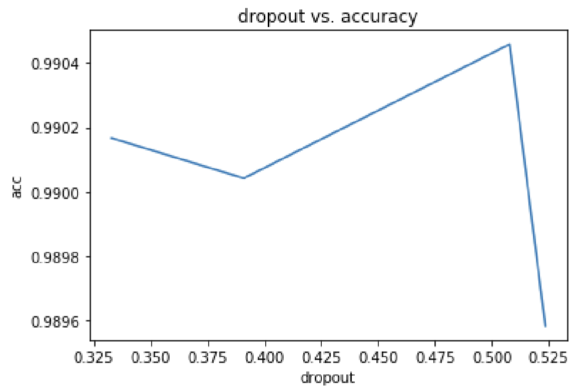

Fig.2: Dropout vs. Accuracy

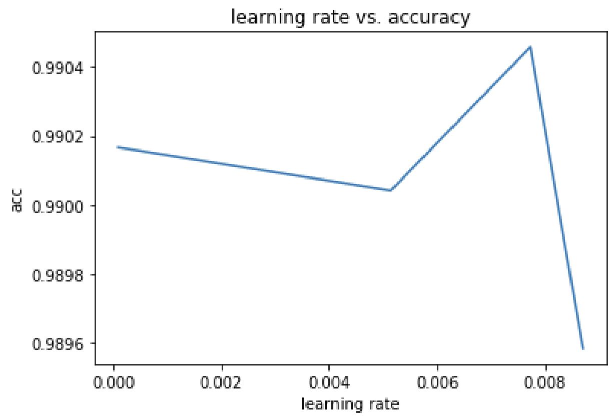

Fig.3: Learning Rate vs. Accuracy

Random Search 2:

Fig.1: Batch vs. Accuracy

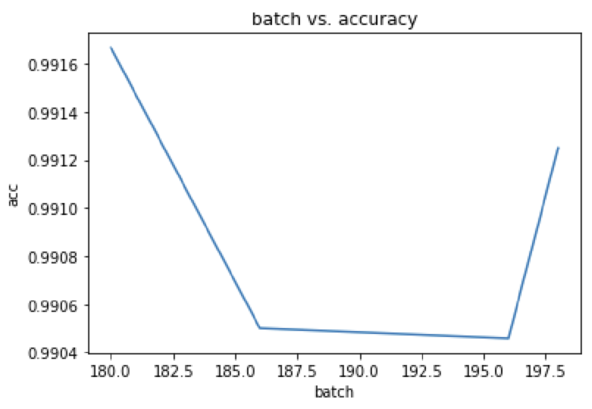

Fig.4: Batch vs. Accuracy

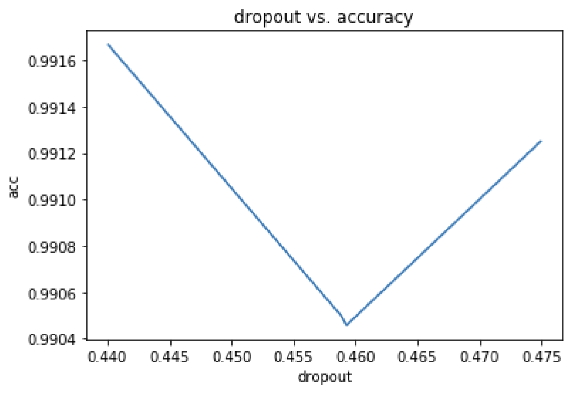

Fig.5: Dropout vs. Accuracy

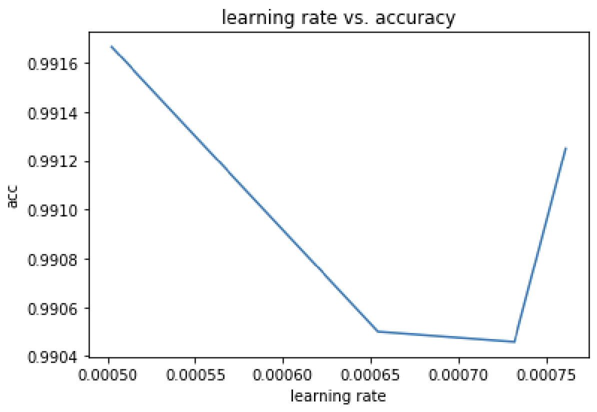

Fig.6: Learning Rate vs. Accuracy

From studying these graphs, I was able to choose a set of hyper parameters with which I proceeded to pick the activations.

V. Model Performance vs. Unit activations

With the best model hyper parameters, the model performance was tested by changing the activation functions of all the units except the units in the last layer. The last layer activation was used as Softmax. It was observed that ReLU activation gave the best performance with a test accuracy of 99.36%.

Activation Function	Test Accuracy
ReLU	99.36
Sigmoid	99.13
Tanh	98.89

The learning curves for both the training and validation sets for these different activation functions are:

1. ReLU

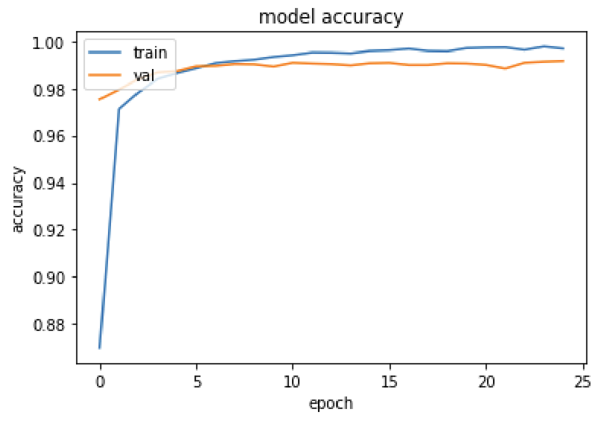

Fig.7 : Accuracy for ReLU

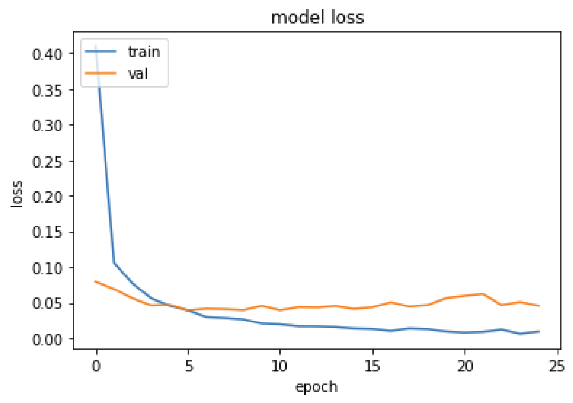

Fig.8 : Loss for ReLU

2. Sigmoid

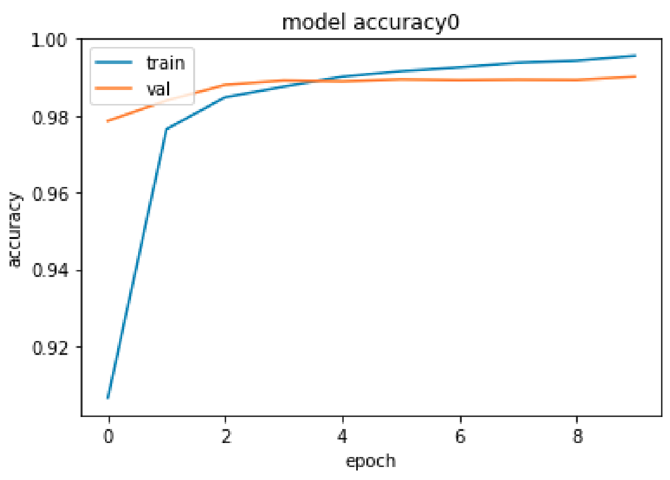

Fig.9 : Accuracy for Sigmoid

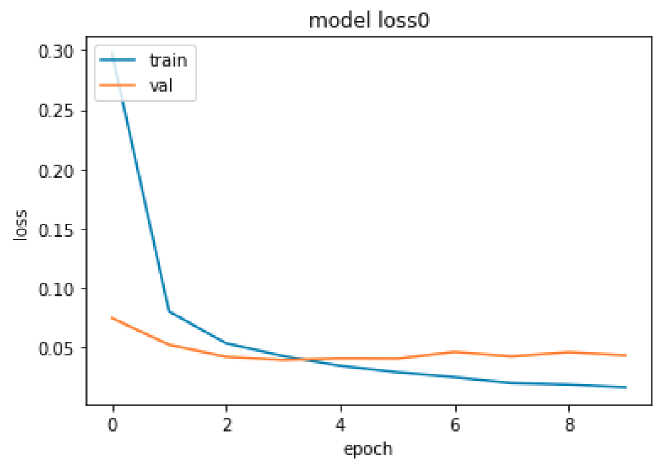

Fig.10 : Loss for Sigmoid

3. Tanh

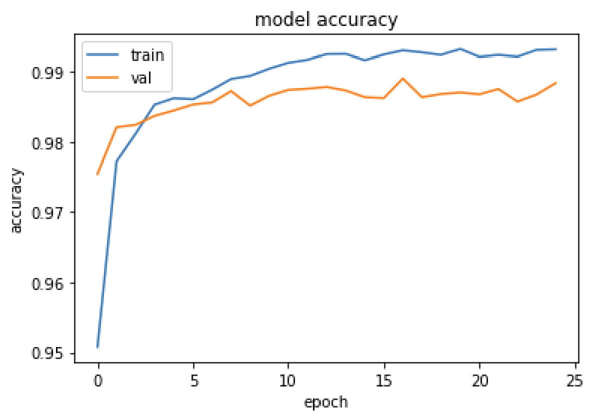

Fig.11 : Accuracy for Tanh

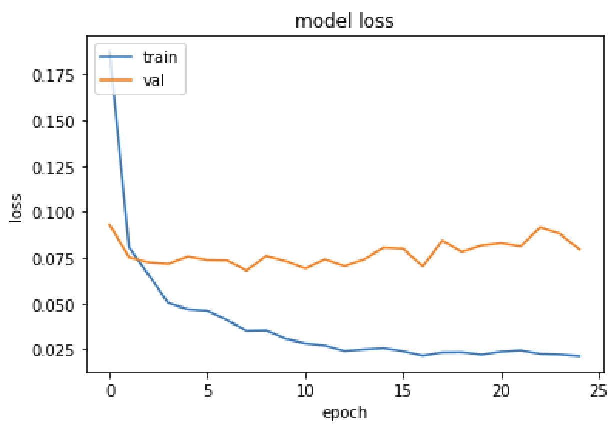

Fig.12 : Loss for Tanh

VI. Conclusion

The Convolution Network was successfully implemented using Keras for MNIST data classification. The hyper parameters - learning rate, batch size and dropout rate - were tuned using a random-search technique and the optimal model was generated. The hyper parameter dependence of this model was studied. Also the model performance was tested on the testing data using the different activation functions - ReLU, Sigmoid and Tanh. It was observed that ReLU activation produced best performance on the test data with a test accuracy of 99.36%.