Convolutional Neural Network for MNIST classification

Abhisu Mishra
Dept. of Electrical and Computer Engg.,
North Carolina State University
Raleigh, NC, USA
amishra6@ncsu.edu

Eshaan V Kirpal
Dept. of Electrical and Computer Engg.,
North Carolina State University
Raleigh, NC, USA
evkirpal@ncsu.edu

Sreeraj Rajendran
Dept. of Electrical and Computer Engg.,
North Carolina State University
Raleigh, NC, USA
srajend2a@ncsu.edu

***Abstract*—The following report details our design and implementation of a convolutional neural network (CNN) for MNIST digit classification. The designed network predicts the digit contained in the input image fed to the network with very high accuracy. The network architecture involves multiple layers including convolution, maxpooling, and fully connected layers. Optimal values for hyperparameters are critical in obtaining good results. Hyperparameter tuning is discussed in detail and associated results have been visualized through various plots. The results using the final model gives accuracy of over 99 percentage on the test dataset which is presented in the later section.**

***Index Terms*—Convolutional Neural Network, hyperparameter, activation functions, trained model, test set**

I. Introduction

The goal of this project was to implement a CNN architecture using a deep learning framework. A convolutional neural network for MNIST classification was designed and implemented using keras with tensorflow as backend. Hyperparameter tuning was then performed to decide upon the optimal number of layers, learning rate, and batch size. Babysitting and grid search approaches were tried out and the optimal method and corresponding results are discussed in detail. Further, once the best set of hyperparameters were decided, different activation functions like Relu, sigmoid and tanh were applied to compare their performances. Corresponding results and learning curves are provided as well. The optimally trained model was then used to predict the testing data-set and the results are presented in detail in later sections.

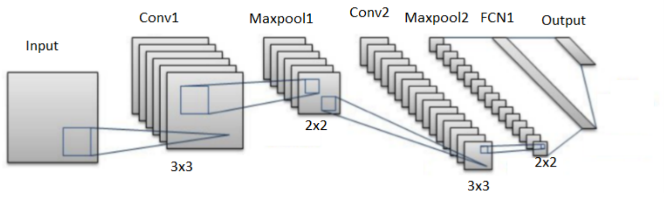
II. Convolutional Neural Network Architecture

Convolutional Neural Networks are a special kind of multi-layer neural networks. Like almost every other neural networks they are trained with a version of the back-propagation algorithm. Convolutional Neural Networks are designed to recognize visual patterns directly from pixel images with minimal preprocessing. They can recognize patterns with extreme variability, and with robustness to distortions and simple geometric transformations.

Although fully connected feedforward neural networks can be used to learn features as well as classify data, it is not practical to apply this architecture to images. A very high number of neurons would be necessary, even in a shallow architecture, due to the very large input sizes associated with images, where each pixel is a relevant variable. Convolutional layers apply a convolution operation to the input. The convolution operation brings a solution to this problem as it reduces the number of parameters, allowing the network to be deeper with fewer parameters. The primary purpose of convolution is to extract features from the input image. Convolution preserves the spatial relationship between pixels by learning image features. Each filter in the convolution layer with its own unique set of weights acts as a specific type of feature detector. Initial layer learns low level features such as edges and lines. Subsequent layers learn higher level patterns. The architecture we designed for image classification application consists of the 2 convolutional layers with max-pooling followed by a fully connected layer with 64 hidden units and a final output layer with 10 hidden units to predict the result. The layers in sequence are described below:

Input: The input data is a grayscale image of size 28x28x1, consisting of raw pixels of image containing digit data.
Conv2D Layer: Consists of 32 filters/kernals of size 3x3 convolving over the input image. Relu activation is used to introduce non-linearity.
Maxpooling2D Layer: 2x2 max-pooling is used to reduce the dimension of the output of first convolution layer. With a stride of zero, every 2x2 cell acted upon will be replaced by the maximum value amongst the cells.
Conv2DLayer: An additional convolutional layer consisting of 64 filters of size 3x3 is introduced followed by a relu activation function.
Maxpooling2D Layer: Another max-pooling layer is introduced to further reduce the number of neurons.
Fully Connected Layer 1: This layer consists of 64 hidden units and is connected to every neuron in the output of previous max-pooling layer. This layer works similar to a regular neural network with many parameters to be learned. But the convolution operations performed earlier helps significantly in learning with minimal parameters allowing number of units in this layer to be minimal. Relu activation is used at the end of fully connected layer.

Fully Connected Layer 2: This layer consists of 10 hidden units and a softmax activation is applied to predict the digit in the input image. The softmax activation outputs the model's predictions in the form of probabilities for each of the 10 labels ranging from 0 to 9.



CNN architecture

III. Implementation

A. Dataset

The dataset used is the MNIST dataset. It is database of handwritten digits with each image a gray scale image of dimension 28*28. We used 48000 images in the training set, 12000 images in the validation set and 10000 images in the testing set.

B. Training

The training dataset is split into training and validation sets. The training is done on the training set and the hyperparameters are tuned on the training and validation losses calculated during cross validation. Grid search is performed to obtain ideal values of learning rate and batch size. And finally, the optimum number of epochs is decided based on the plots generated during cross-validation.

C. Hyperparameter Optimization

Models with one, two, and three convolution layers along with fully connected layers were tried. Keeping the epochs constant at 10, we observed higher accuracy by increasing the number of layers. But, it was noticed that even with a single convolution layer of 64 filters and an output dense layer of 10 units, training accuracies in the range of 98 percentage is easily obtained , demonstrating the power of convolutional networks over regular neural network for image classification. An architecture with two convolution layers as described in the previous section seemed optimal for our application and thus further hyperparameter tuning was performed on this network.

As earlier mentioned, the hyperparameter optimization is done using a grid search on the values of learning rate and batch size. Grid search is done because when there are three or fewer hyperparameters, the common practice is to perform grid search. For each hyperparameter, the user selects a small nite set of values to explore. The grid search algorithm then trains a model for every joint specication of hyperparameter values in the cartesian product of the set of values for each individual hyperparameter. The experiment that yields the best validation set error is then chosen as having found the best hyperparameters. The hyperparameters that were first tuned are learning rate and batch size.

Learning Rate	Batch Size	Validation Loss	Validation Accuracy
0.01	64	0.22735	0.9855
0.01	128	0.24399	0.98391
0.01	256	0.27787	0.982
0.01	512	0.11672	0.99066
0.001	64	0.091982	0.99108
0.001	128	0.86343	0.99016
0.001	256	0.70916	0.99083
0.001	512	0.63076	0.99008
0.0001	64	0.47751	0.98841
0.0001	128	0.0426	0.98891
0.0001	256	0.4992	0.98491
0.0001	512	0.05017	0.98591

TABLE I
Comparison of Validation loss and accuracy for different Learning Rate and Batch Size.

1) Learning Rate: We performed a grid search for tuning the appropriate learning rate while training the neural network. The learning rates selected for the grid search were 0.01, 0.001 and 0.0001. The learning rate will be selected which gives the least possible validation loss.

2) Batch Size: We performed a grid search for tuning the appropriate batch size while training the neural network. The batch sizes selected for the grid search were 64, 128, 256 and 512.

The learning rate and batch size which gave the least possible validation loss, were selected for the next phase of the hyperparameter tuning. Finally, the ideal number of epochs were decided based on the accuracy and loss plots obtained during this process.

Table I sums up our observations for the grid search showing us how the validation loss and accuracy vary with the different values of learning rate and batch size.

IV. Results

A. Determination of optimal learning rate and batch size

As discussed in the above section the learning rate and batch size are varied to find the optimal validation loss. Also special care is taken to avoid overfitting when selecting these parameters. The validation loss and accuracy was plotted against the number of epochs to give us a visualization of the cross validation being performed. These visualizations are shown in images 2-25. As it can be seen from these images the value of learning rate and batch size for which validation loss is minimum and there is minimum overfitting is validation loss = 0.001 and batch size = 64, which is figure number 10 and 11. From these two figures, we can also infer that the model performs best (least loss) at around epoch 10 and thereafter tends to overfit. Hence, number of epochs we eventually trained and tested our model on was 10.

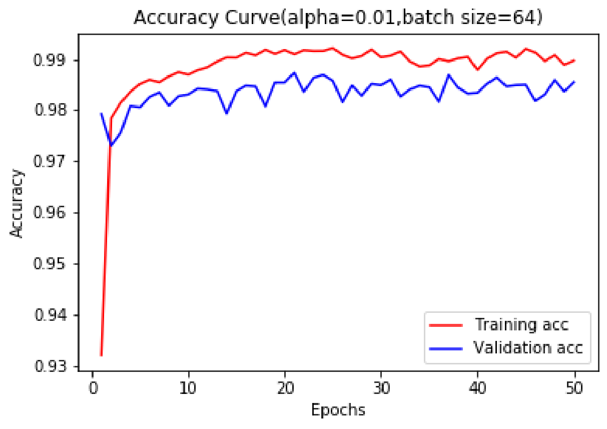


Fig 2: Learning Rate = 0.01 and Batch Size = 64

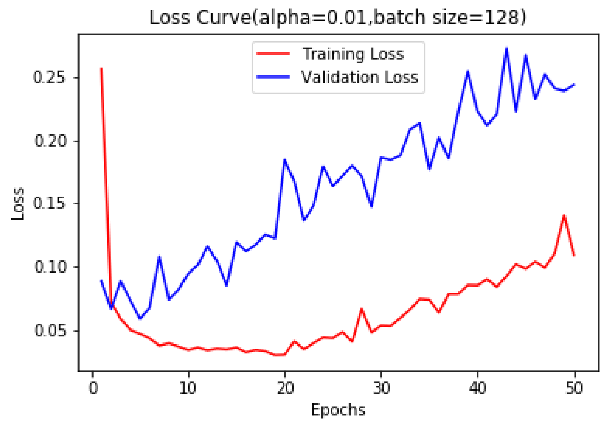


Fig 5: Learning Rate = 0.01 and Batch Size = 128

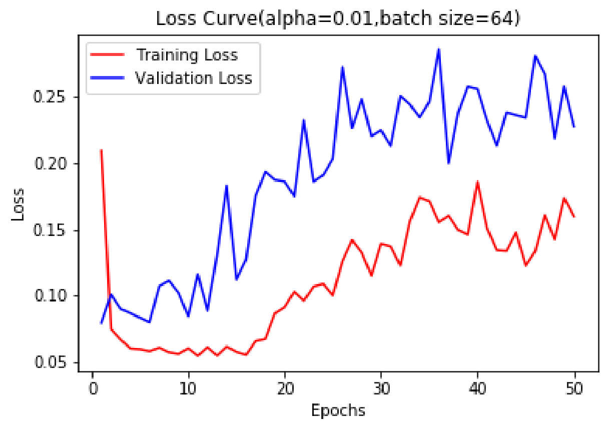


Fig 3: Learning Rate = 0.01 and Batch Size = 64

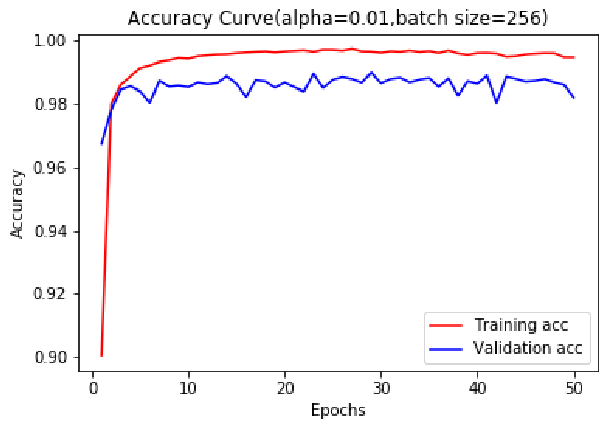


Fig 6: Learning Rate = 0.01 and Batch Size = 256

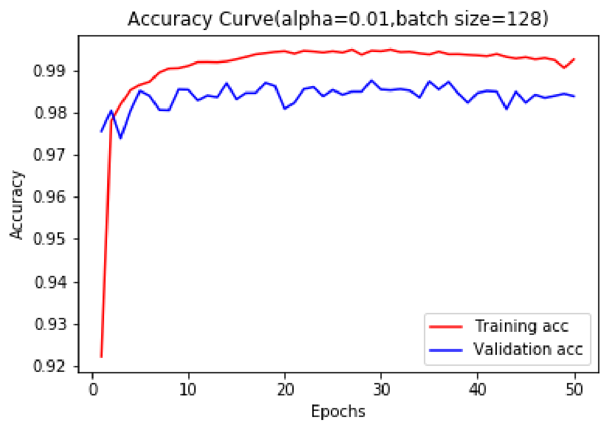


Fig 4: Learning Rate = 0.01 and Batch Size = 128

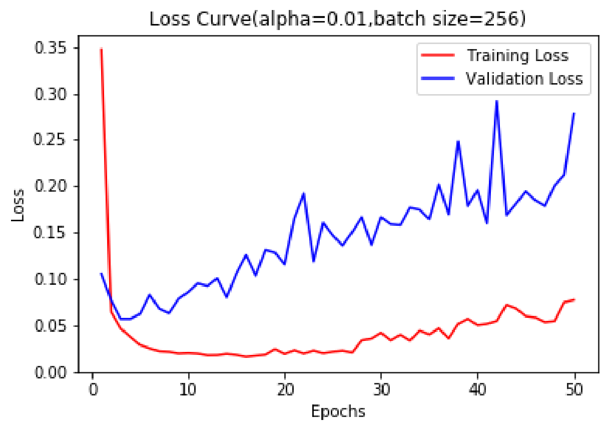


Fig 7: Learning Rate = 0.01 and Batch Size = 256

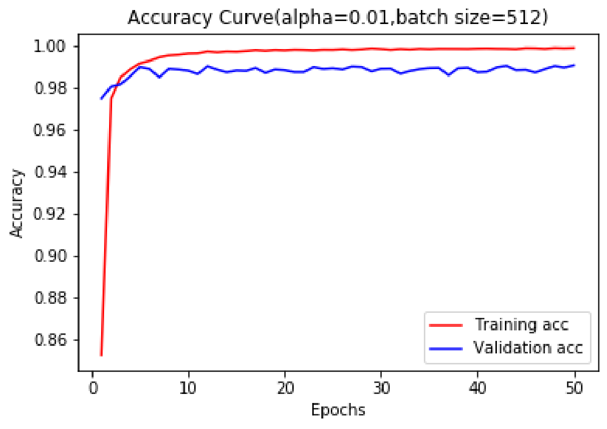


Fig 8: Learning Rate = 0.01 and Batch Size = 512

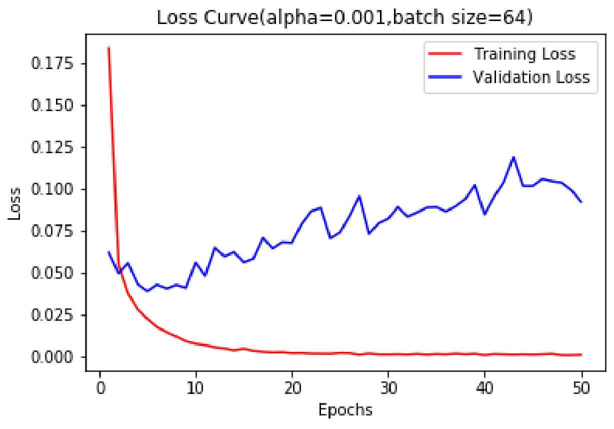


Fig 11: Learning Rate = 0.001 and Batch Size = 64

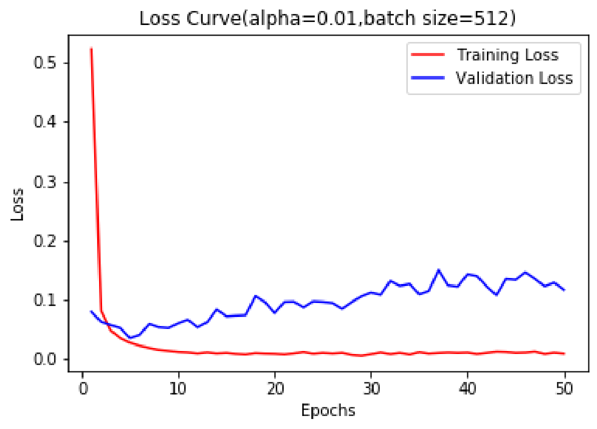


Fig 9: Learning Rate = 0.01 and Batch Size = 512

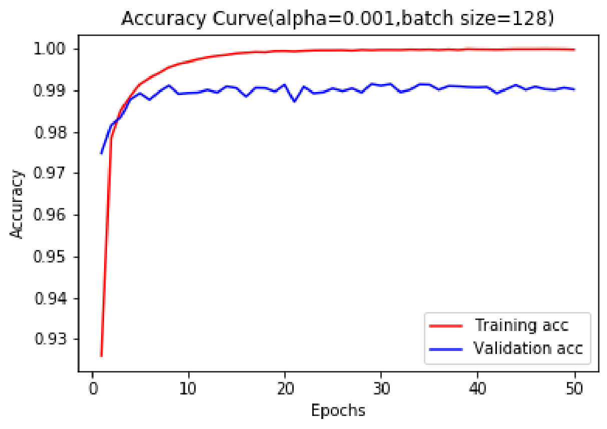


Fig 12: Learning Rate = 0.001 and Batch Size = 128

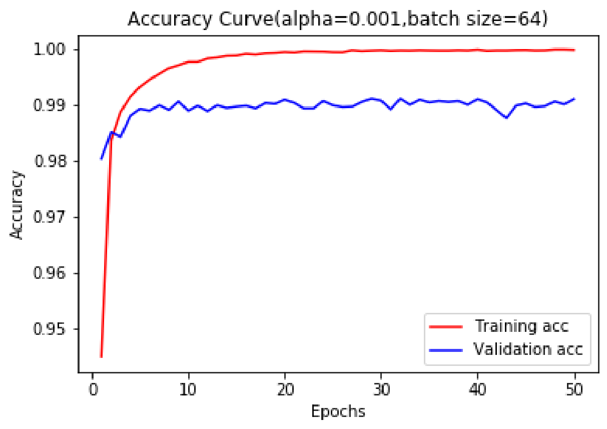


Fig 10: Learning Rate = 0.001 and Batch Size = 64

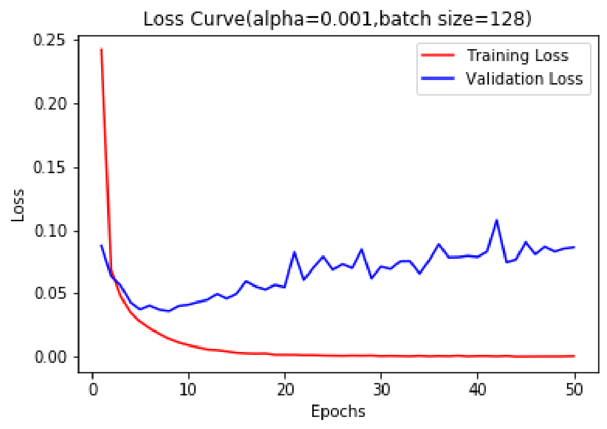


Fig 13: Learning Rate = 0.001 and Batch Size = 128

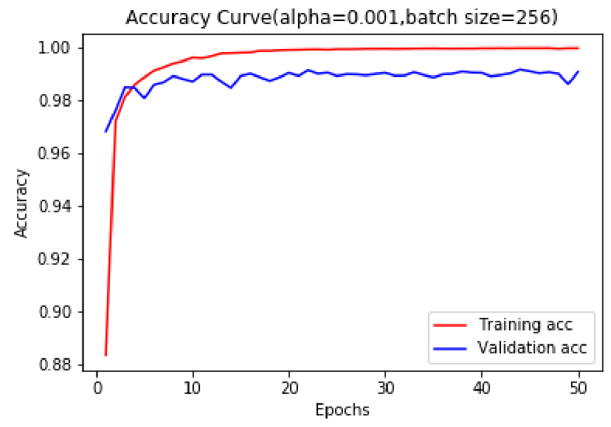


Fig 14: Learning Rate = 0.001 and Batch Size = 256

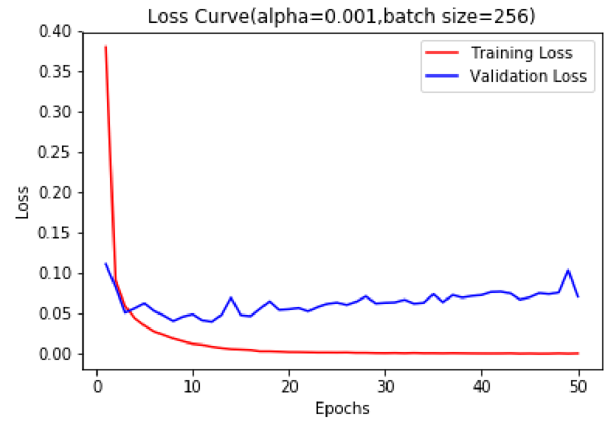


Fig 15: Learning Rate = 0.001 and Batch Size = 256

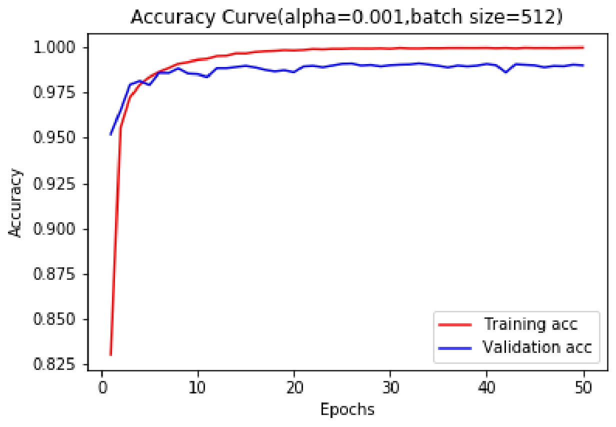


Fig 16: Learning Rate = 0.001 and Batch Size = 512

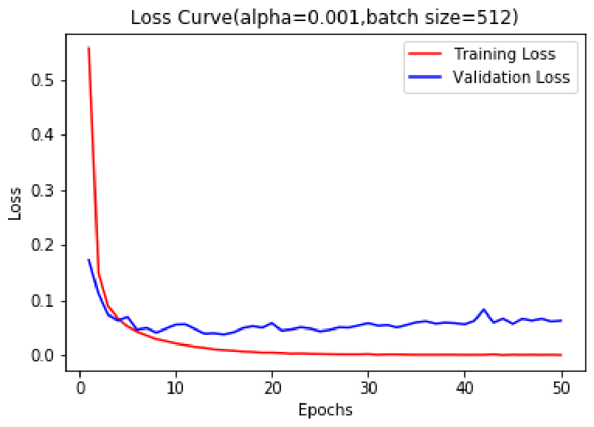


Fig 17: Learning Rate = 0.001 and Batch Size = 512

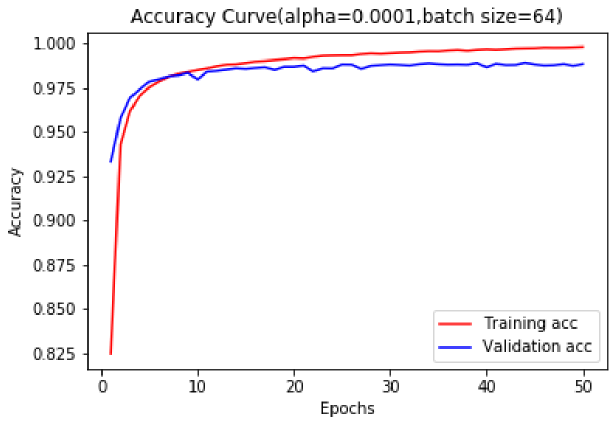


Fig 18: Learning Rate = 0.0001 and Batch Size = 64

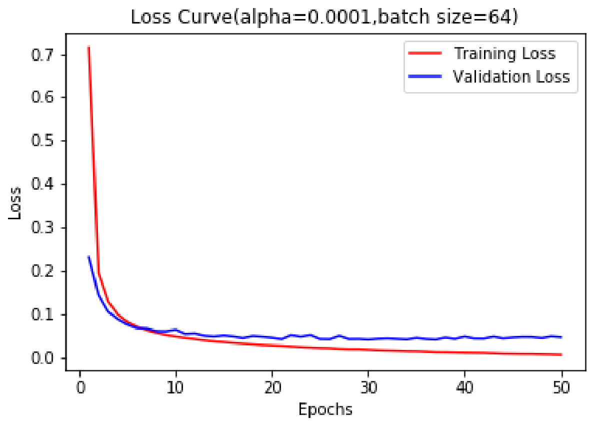


Fig 19: Learning Rate = 0.0001 and Batch Size = 64

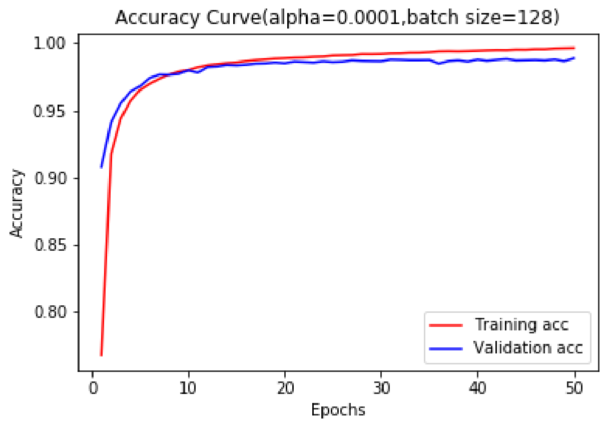


Fig 20: Learning Rate = 0.0001 and Batch Size = 128

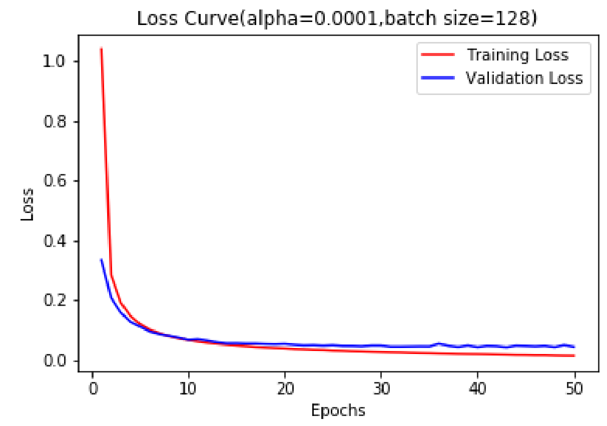


Fig 21: Learning Rate = 0.0001 and Batch Size = 128

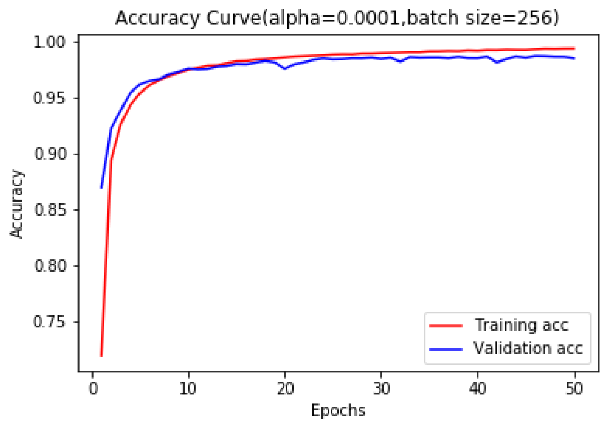


Fig 22: Learning Rate = 0.0001 and Batch Size = 256

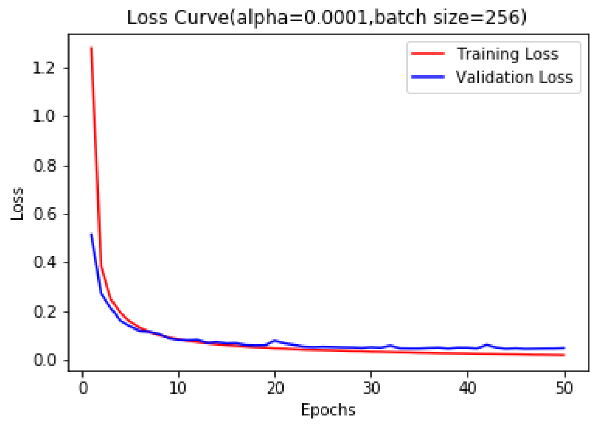


Fig 23: Learning Rate = 0.0001 and Batch Size = 256

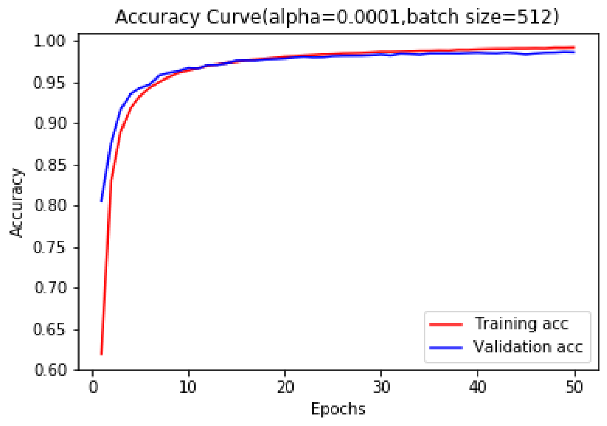


Fig 24: Learning Rate = 0.0001 and Batch Size = 512

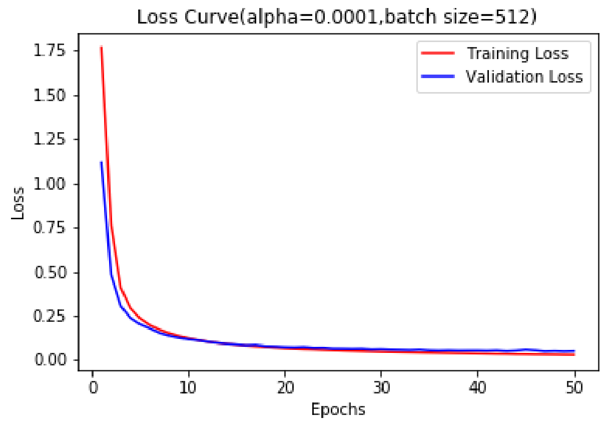


Fig 25: Learning Rate = 0.0001 and Batch Size = 512

B. Applying Different Activation Functions

This was the second stage of the hyperparameter tuning process where we applied different activation functions while keeping the number of epochs equal to 50, learning rate and the batch size were kept at the value we inferred from the previous stage.

1) ReLU: By having a ReLU activation function with Learning Rate=0.001 and Batch Size=64 we get a cross-validation accuracy of 99.108 and cross-validation loss of

0.091982. Figures 26 and 27 below shows the variation of cross-validation accuracy and Loss as number epochs increases.

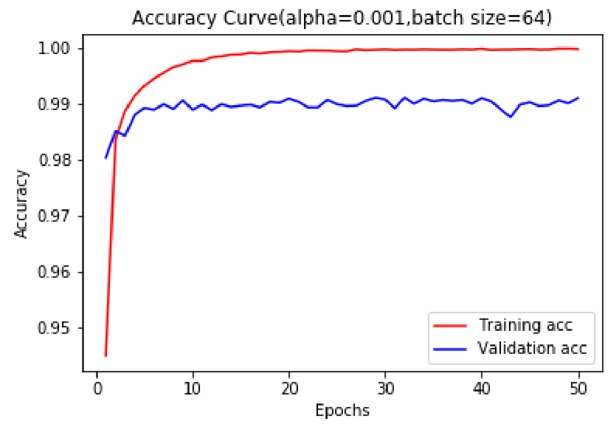


Fig 26: Learning Rate = 0.001 and Batch Size = 64

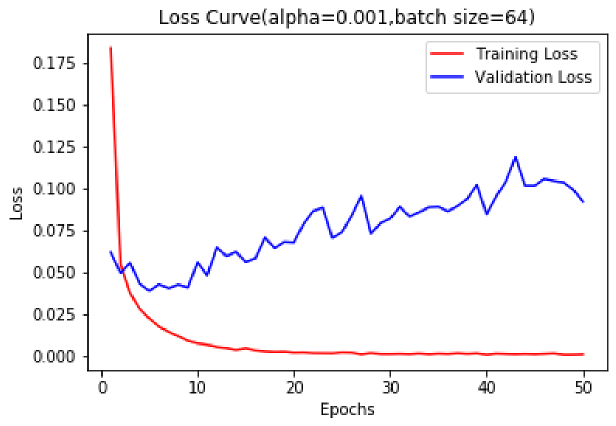


Fig 27: Learning Rate = 0.001 and Batch Size = 64

2) tanh: By having a tanh activation function with Learning Rate=0.001 and Batch Size=64 we get a cross-validation accuracy of 99.08 and cross-validation loss of 0.0716. Figures 28 and 29 below shows the variation of cross-validation accuracy and Loss as number epochs increases.

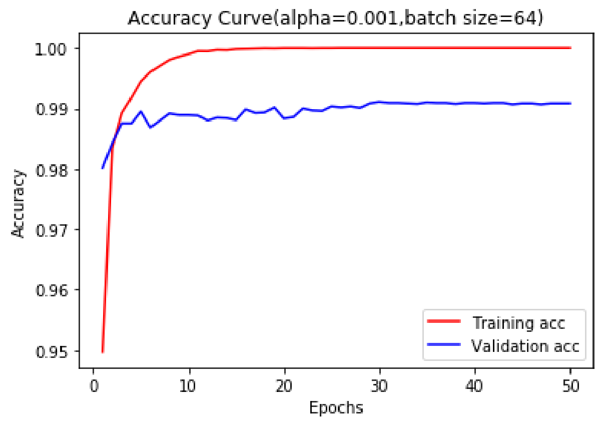


Fig 28: Learning Rate = 0.001 and Batch Size = 64

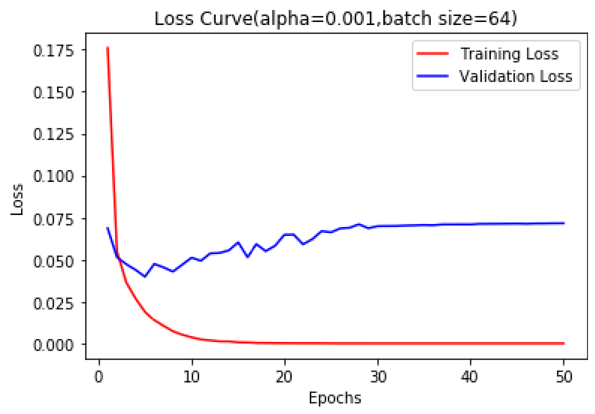


Fig 29: Learning Rate = 0.001 and Batch Size = 64

3) Sigmoid: By having a Sigmoid activation function with Learning Rate=0.001 and Batch Size=64 we get a cross-validation accuracy of 98.76 and cross-validation loss of 0.0643. Figures 30 and 31 below shows the variation of cross-validation accuracy and Loss as number epochs increases.

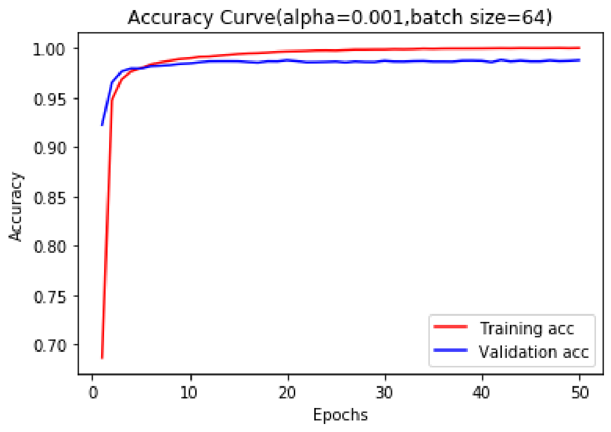


Fig 30: Learning Rate = 0.001 and Batch Size = 64

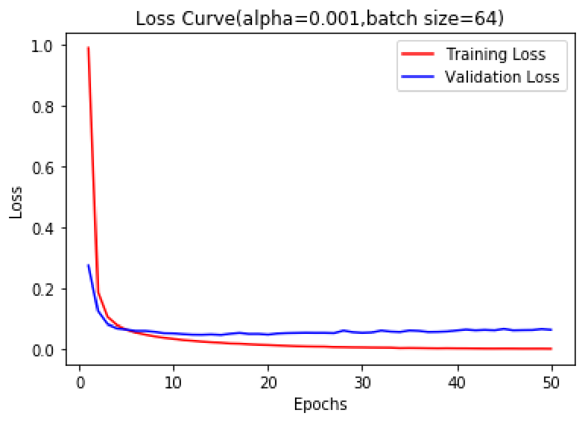


Fig 31: Learning Rate = 0.001 and Batch Size = 64

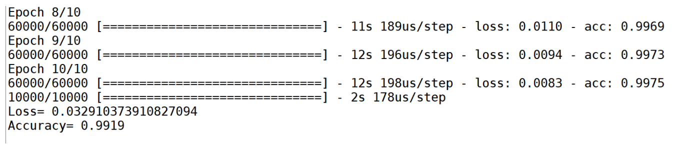
Activation Function	Validation Loss	Validation Accuracy
ReLU	0.091982	0.99108
tanh	0.0716	0.9908
Sigmoid	0.0643	0.9876

Table 2: Comparison of Validation loss and accuracy for different Activation Functions.

Upon comparing figures 26-31 at around epoch 10, we find that the ReLU activation function has the lowest validation loss while preventing overfitting. Also it can be seen from the table and the figures, ReLu function has the best accuracy amongst all. Hence ReLU activation function was selected for our architecture.

C. Performance on Test Data

The final model was trained on the entire training dataset which includes the validation set as well using the chosen hyper-parameters,i.e. epochs=10, learning rate=0.001, batch size=64, and activation= ReLu. The model was then evaluated on the test data giving an accuracy percentage of 99.19 with a loss value of 0.03291 (as shown in the below figure).



```
Epoch 8/10
60000/60000 [==============================] - 11s 189us/step - loss: 0.0110 - acc: 0.9969
Epoch 9/10
60000/60000 [==============================] - 12s 196us/step - loss: 0.0094 - acc: 0.9973
Epoch 10/10
60000/60000 [==============================] - 12s 198us/step - loss: 0.0083 - acc: 0.9975
10000/10000 [==============================] - 2s 178us/step
Loss= 0.032910373910827094
Accuracy= 0.9919
```

Fig 32: Accuracy and Loss Results on Test Set

References

[1] Practical Methodology — Deep learning, chapter 11, DeepLearning. [Online]. Available: https://www.deeplearningbook.org. [Accessed: 12-Oct-2018].

[2] GitHub. [Online]. Available: https://github.ncsu.edu/qge2/ece542-2018fall. [Accessed: 12-Oct-2018].