Convolutional Neural Network for MNIST classification

1st John McDonald
Electrical and Computer Engineering Dept
North Carolina State University
Raleigh, North Carolina
jamcdon3@ncsu.edu

***Abstract*—This is an assignment for the ECE 542 course to become familiar with Convolutional Neural Networks (CNN) and deep learning processes. The goal is to design and implement a CNN for MNIST classification. Alternatively, the goal is to become familiar with Tensorflow and in this case Keras Python libraries, for the use of CNN architecture. This paper includes the details of the delivered neural network structure and additionally performance analysis, using a high level API library. The performance of the CNN is tested against multiple independent variables.**

***Index Terms*—Convolutional Neural Networks, Machine Learning, Keras, Tensorflow, Relu, Sigmoid, Tanh**

I. Structure of Network

The network is built in Python using a high-level Keras library that sits on top of Tensorflow. Keras caters to neural networks and has a user-friendly interface. The CNN is built by adding layers to a model using Sequential() functionality [3]. I chose to build the CNN using the architecture shown below:

TABLE I
Model Architecture

Layer	Type	Details
1	Convo2D	Relu with kernal (3,3)
2	Convo2D	Relu with kernal (3,3)
3	MaxPooling2D	Pool size (2,2)
4	Dropout	Rate (.25)
5	Dense	FC using Relu
6	Dropout	Rate (.5)
7	Dense	FC using Softmax

The model is then compiled using a cross entropy loss function as provided by the Keras framework. Cross entropy loss is best suited for multi-class classification such as the MNIST data set, as opposed to binary classification as performed in Homework 3. The optimizer is chosen as AdaDelta, which has a per-dimension learning rate and performs the gradient descent [3]. Details regarding training and evaluation of the model are included in the following sections.

II. Hyper-Parameter Tuning

The purpose of the hyper-parameter tuning is to select parameters that facilitate highest accuracy for the CNN model training. The two easiest hyper-parameters to adjust in the high level build of Keras CNN are the batch size and number of Epochs for model training.

In this paper I compare the model accuracy and loss for two experiments; First, I change the number of epochs from 1 to 15 and compare the results with validation error; Second, I keep the number of epochs consistent, while adjusting the batch size between 64, 128, and 256.

III. Visualization of Cross-Validation for Hyper Parameters

Epoch is the first hyper parameter chosen for tuning. It defines how many cycles are run through the full data training set. Traditionally higher number of Epoch's leads to higher training accuracy. However, too many epochs can lead to overfitting, which creates bias in accuracy of the model test results. Traditionally, overfitting is indicated by an increase in the validation error after a large number of epochs.

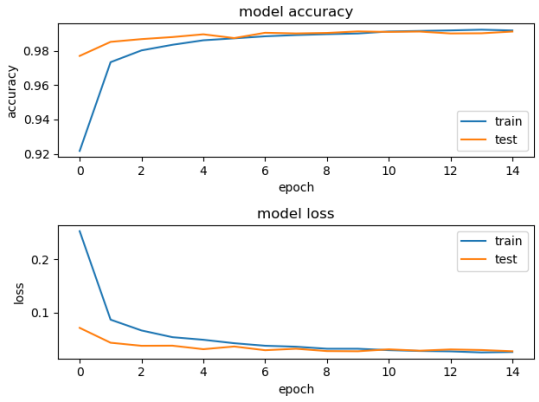

Fig. 1. Model loss and Model accuracy of training and validation sets when compared with Hyper-Parameter number of Epochs, ranging from one to fifteen.

As shown above, the training and validation errors meet at around the 8 epochs mark. However, the validation error does not begin to increase. Theoretically the model can be trained

more, but seeing as how the validation error is very slowly increasing, I have decided to choose 10 as the optimal number of epochs due to the computational stress on my computer.

Batch size is the second hyper parameter chosen for tuning. The results are shown below for testing accuracy and loss.

TABLE II
BATCH SIZE RESULTS

Batch Size	Test Loss	Test Accuracy
64	0.03352211109676864	0.989300012588501
128	0.03449841103209183	0.988099992275238
248	0.04233065822778735	0.9855999946594238

From the table above we see that the lowest batch size yields the highest accuracy and lowest test loss. These batch sizes were measured while keeping the epoch variable consistent at 3.

IV. LEARNING CURVE OF ACTIVATION FUNCTIONS

Different activation functions can be used for the convolution layers when building the architecture of the CNN. In this section I compare the results of using Relu, Sigmoid, and Tanh as activation functions for the convolutional layers, and the first fully connected 'dense' layer. Note that the last 'dense' layer was kept consistent as a Softmax activation function to intuitively output the propabilities of each classification.

A. ReLu Activation

First we test relu as the activation function. Relu is the most popular for CNN architecture of convolutions, as it provides consistently good results. It is half rectified from the bottom, and linear for positive numbers [2].

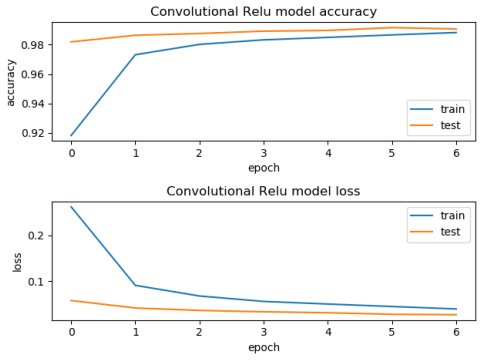

Fig. 2. Model loss and Model accuracy of training and validation sets for the Relu Activation function.

B. Sigmoid Activation

Second we test Sigmoid as the activation function. Sigmoid's key characteristic is that the output is between (0,1), so it is often used for predictions and propability. It is half rectified from the bottom, and linear for positive numbers. The logistic sigmoid function can CNN's to get stuck when training; this most likely explains the first epoch not returning positive results [2].

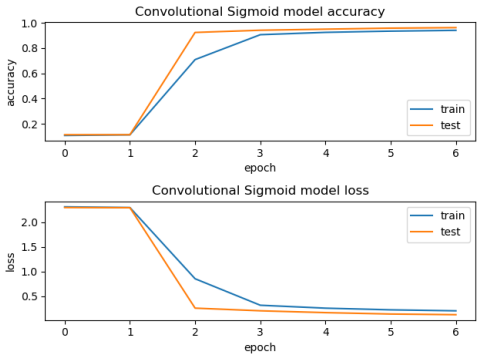

Fig. 3. Model loss and Model accuracy of training and validation sets for the Sigmoid Activation function.

C. Tanh Activation

Third we test the Tanh activation function. Tanh is similar to sigmoid but generally performs better. It has a sigmoidal shape, with a range from (0,1). It is often used for binary classification. Like the sigmoid, it is monotonic and differentiable [2].

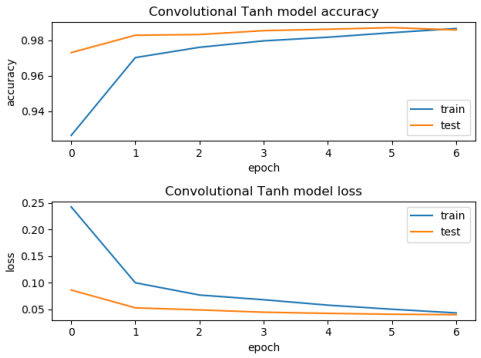

Fig. 4. Model loss and Model accuracy of training and validation sets for the Tanh Activation function.

Relu performed best out of the three activation functions tested. Tanh also performed well, with a significant drop off in accuracy for Sigmoid. More details regarding accuracy and loss metrics are included in Section V.

V. ACCURACY ON TESTING SET

For hyper parameter tuning, the first test visualized the results on training and validation sets for 1-15 epoch's. The testing results from that experiment, after being trained with 15 epochs, are as follows:

TABLE III
HYPER-PARAMETER (EPOCH) TEST RESULTS

Test Loss	0.027105548594243828
Test accuracy	0.9912999868392944

As discussed, this testing set may be subject to overfitting with a high epoch parameter. So for optimization and simplicity in computing, I have chosen 8 epochs for my optimal design.

For the second hyper parameter, a batch size of 64 was chosen for the optimal design. This is based off training results from section III.

The results for Test Loss and Test Accuracy for tested activation functions is shown below. Relu performed best with the highest accuracy and lowest loss. These results make sense as ReLu is the most used activation function in the world right now, and is used in almost all CNN and deep learning networks. (SOURCE)

TABLE IV
ACTIVATION FUNCTION METRICS

Activation Function	Test Loss	Test Accuracy
Relu	0.028480343098720187	0.9894000291824341
Sigmoid	0.12745234538540245	0.961899995803833
Tanh	0.039550631829642226	0.9858999848365784

VI. ANALYSIS AND DISCUSSION

The results of this paper are synonymous with a lot of research as shown online. ReLu is commonly known as the best activation function for convolutions, which similarly proved to have the best results in this paper [2]. Hyper parameter tuning was similar to what online research showed, as commonly larger batch create a significant degradation in the quality of model accuracy, as measured by the ability to generalize results. This is because larger batch sizes tend to converge to sharp minimizers of the training set data [1]. Regarding number of epochs, the epoch can theoretically be increased until the validation error starts to increase. However, due to computation stress of the CNN training, I have decided to choose 15 epochs as the optimal number for training my model. This is beyond the point where the validation error starts to converge on a single number. Unfortunately more epochs takes a very long time to train the model.

For the optimal design I have settled on the following independent variables, as shown in the below table.

TABLE V
OPTIMAL DESIGN

Independent Variable	Chosen Implementation
Number of Epochs	15
Batch Size	128
Activation Function	ReLu

For my optimal design, the test accuracy is 0.9912999868392944 and the test loss is 0.027105548594243828. The results for this prediction set are attached in a separate mnist.csv file, and the source code is attached.

REFERENCES

[1] N. Ketkar, "Training Deep Learning Models," Deep Learning with Python, pp. 215–222, Feb. 2017.

[2] S. Sharma, "Activation Functions in Neural Networks," Medium, 14-Feb-2019. [Online]. Available: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6. [Accessed: 15-Oct-2019]."Why use Keras?,"

[3] Why use Keras - Keras Documentation. [Online]. Available: https://keras.io/why-use-keras/. [Accessed: 15-Oct-2019].