Report on CNN architecture to classify the MNIST handwritten dataset

Istiaq Md Jamil
American International University-Bangladesh (AIUB)
Dept. of CSE
jamilistiaq@gmail.com

***Abstract*— In this report we are going to achieve excellent accuracy over 98% using CNN architecture on MNIST dataset. Different optimizers and hidden layer and filter used in the CNN architecture. In the project 'Adam', 'SGD', 'RMSProp' used as optimizer. The train model is designed with 'Relu', 'Softmax' as activation function and 'Maxpool' for pooling.**

Keywords—CNN, Neural Network, MNIST dataset, Adam, SGD, RMSProp, Tenorflow.

I. Introduction

CNN or ConvNet stands for Convolutional Neural Networks. It is identically parallel to neural networks made up with neurons, learnable weights and biases. Neuron receives some inputs and executes a dot product of matrices and pass it to another neuron [1]. A mathematical operation on two function which is produced a third function that is shows how the shape of one is modified by another is call convolution [2].

Next thing is hidden layer, which is 'n' number of layers in between input and output layers. Each layer's input is the previous layer's output. Filter is a square shaped object which is scan and detect different type of patters in an image by changing the position of an image pixels sweeps left to right over each row [3]. Then build a sequential model. In activation function it decides a neuron should activated or not by calculating the weighted sun and bias. Pooling is the function of reduce the size representation to reduce the number of parameters and computing in the network [4].

Flattening is translating the data into a 1-dimensional array for inputting it to the following layer. Deeply connected layers are called dense layer. Algorithms or methods which is used to change the attribute of a neural network like weights, learning rate to reduce the loss and help to get result faster. Calculating loss there are some functions which calculated the loss. Batch size is the number of samples which will pass through the network at a time.
For data preprocessing there is a normalization. After that data is compile in the project.

Experiment time validation split is 0.2% from the train dataset. Batch size and epoch is set respectively 64 and 10
In the model there are three hidden, one input and one dense layer used to get best result. To compile the model the optimizer 'Adam', 'SGD' and 'RMSProp' is used

II. Result

After defining model, fit model into different optimizer and got excellent result on the MNIST datasets. Here is some graph of different optimizer using normalize data and without normalize data.

ADAM:

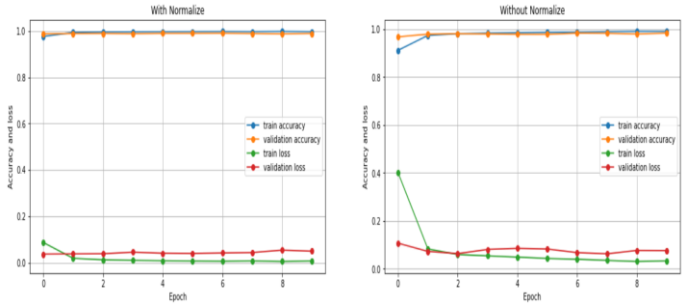

Fig 1: ADAM Optimize

SGD:

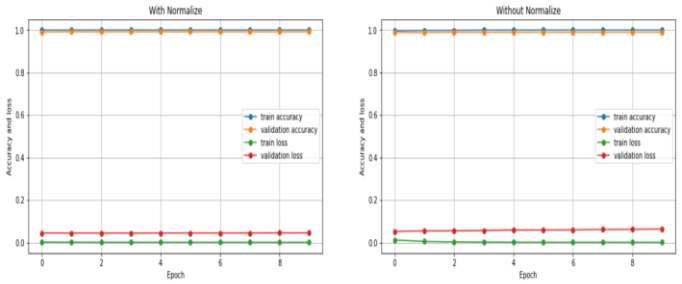

Fig 2: SGD Optimizer

RMSProp:

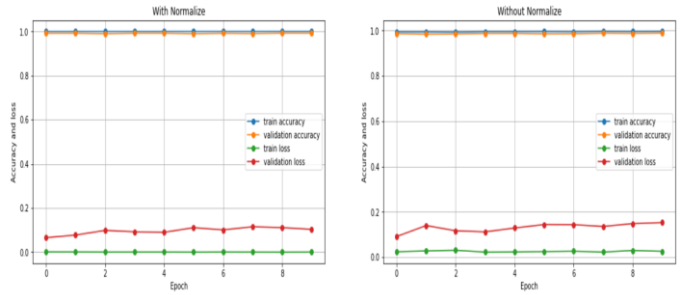

Fig 3: RMSPROP Optimizer

ADAM vs SGD vs RMSPROP with normalization:

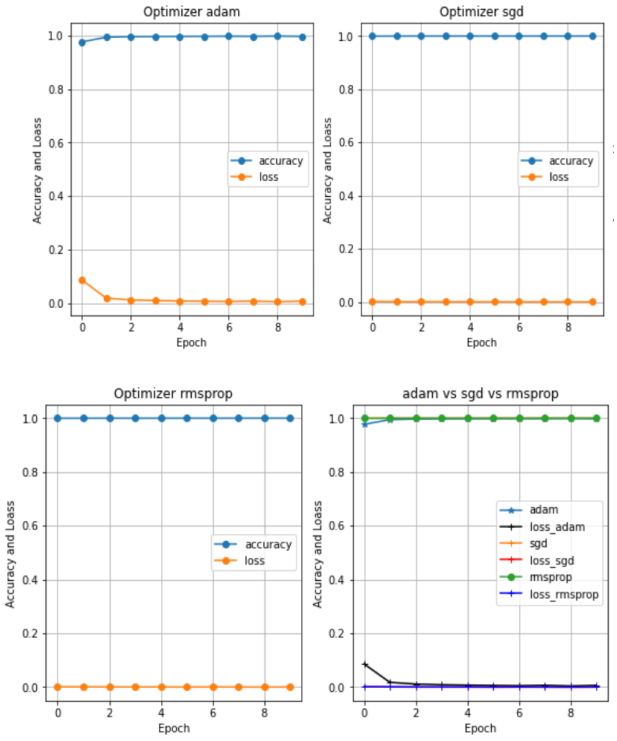

Fig 4: ADAM vs SGD vs RMSPROP Optimizer

ADAM vs SGD vs RMSPROP without normalization:

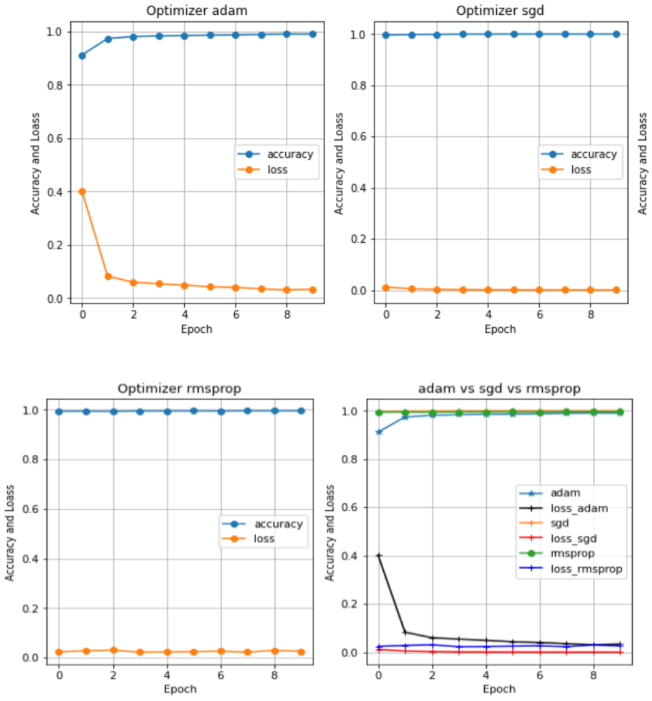

Fig 5: ADAM vs SGD vs RMSPROP Optimizer

SUMMARY OF THE MODELS ON DIFFERENT OPTIMIZER

Optimizer		Normalize Data		Without Normalize Data		Remarks
		Train	Test	Train	Test	
Adam	Epoch No	1st	Accuracy: **99.09%** Loss: 0.033%	1st	Accuracy: **98.51%** Loss: 0.654%	**Adam** with normalize data give better accuracy than without normalize data.
	Accuracy	97.70%		91.18%		
	Val. Accuracy	98.90%		96.79%		
	Epoch No	10th (last)		10th (last)		
	Accuracy	99.77%		99.02%		
	Val. Accuracy	99.01%		98.35%		
SGD	Epoch No	1st	**Accuracy: 99.37%** **Loss: 0.028%**	1st	Accuracy: **99.21%** Loss: 0.050%	**SGD** with normalize data give better accuracy than without normalize data.
	Accuracy	99.96%		99.68%		
	Val. Accuracy	99.11%		98.89%		
	Epoch No	10th (last)		10th (last)		
	Accuracy	100%		100%		
	Val. Accuracy	99.15%		98.98%		
RMSProp	Epoch No	1st	Accuracy: **99.30%** Loss: 0.079%	1st	Accuracy: **98.83%** Loss: 0.143%	**RMSProp** with normalize data give better accuracy than without normalize data.
	Accuracy	99.96%		99.44%		
	Val. Accuracy	99.16%		98.64%		
	Epoch No	10th (last)		10th (last)		
	Accuracy	99.99%		99.59%		
	Val. Accuracy	99.19%		98.85%		

From the table the best optimizer is **'SGD'** the higher accuracy and lowest loss. After that we predict a random picture from test data and the model predict all the number correctly. Here the picture and the highest predicted label is showing the model.

Prediction Visualization using the train model with SGD optimized:

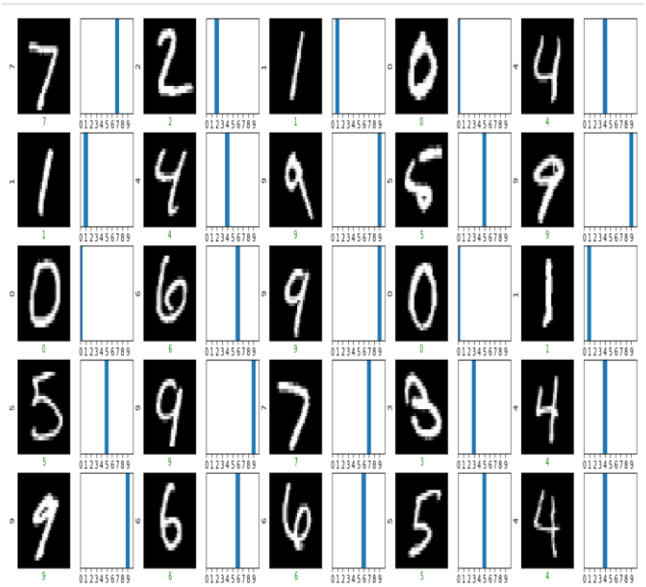

Fig-6: Predicted Photo with Label

III. Discussion

For building project import requires things like tensorflow, matplotlib, numpy, keras and the MNIST dataset.

After importing, load the dataset into two tuples. One is holding test data, test data label and another is train data and train data label. The project architecture is CNN. So that reshape the dataset add a color channel on it but the original size is intake. After plotting some photo, the model is built with 3 hidden layer 'relu' activation function, 'maxpool' for pool, dense layer. The pre-processed and non-pre-processed data is passed into the model. In the compilation of the model for pre-processed and not preprocessed data there is two types of accuracy and loss we have got in the project. The best and highest accuracy is found in pre-processed data.

The different optimizers gave very close accuracy to each other. From the above graphs and table, it is clear that the accuracy of the three optimizer is very close to each other. The 'Adam' has 98.51 to 99.09 percent accuracy depending on the data normalization. Same for 'SGD' it's accuracy is 99.21 to 99.37 percent and the 'RMSProp' has 98.83 to 99.30 percent accuracy.

After analyzing the graph and table data it clear that for CNN on MNIST dataset 'SGD' optimizer giver the best accuracy and lowest loss.

V. References

[1] https://cs231n.github.io/convolutional-networks/

[2] https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-cnn/.

[3] https://kharshit.github.io/blog/2018/12/14/filters-in-convolutional-neural-networks

[4] https://medium.com/technologymadeeasy/the-best-explanation-of-convolutional-neural-networks-on-the-internet-fbb8b1ad5df8