IMPLEMENTATION OF CNN ARCHITECTURE TO CLASSIFY THE MNIST HANDWRITTEN DATASET USING DIFFERENT OPTIMIZERS

Rifat, Md. Ishtiyak Ali
18-37927-2
Dept. of Computer Science and Engineering
American International University Bangladesh
rifatishtiyak@gmail.com

***Abstract*— This paper is focused on the implementation of an MNIST handwritten digit classifier. It was aiming to develop and train a basic convolutional Neural Network (CNN) for classifying handwritten digits from a popular dataset. The digit was taken from 0 to 9. The dataset was cleaned, scaled, and well-shaped. TensorFlow was used to develop a CNN model, which was then trained on the training dataset. With the help of multiple convolutions, relu and pooling layers, an efficient model was created to see the needed accuracy. Different optimizers like Adam, SGD, RMSprop were used to differentiate the accuracy and result. Also, higher accuracies were achieved by tweaking the model hyperparameters a bit in this implementation. The targeted accuracy for this project is above 98%.**

***Index Terms*—CNN, dataset, Optimizer, Adam, SGD, RMSprop.**

I. INTRODUCTION

The MNIST handwritten digit classification problem is a standard dataset used in computer vision and deep learning**.** The full form of MNIST dataset is Modified National Institute of Standards and Technology dataset. This dataset can be used to learn and practice how to build, analyze, and apply convolutional deep learning neural networks for image classification from the scratch. This dataset is the collection of 60000 28×28-pixel grayscale images of the 10 digits.

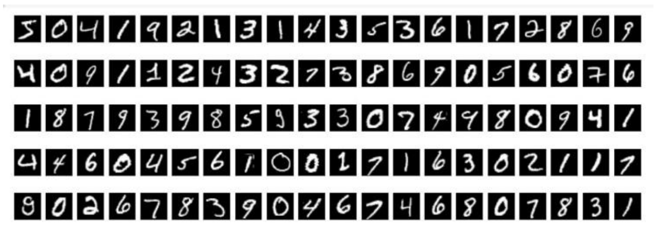
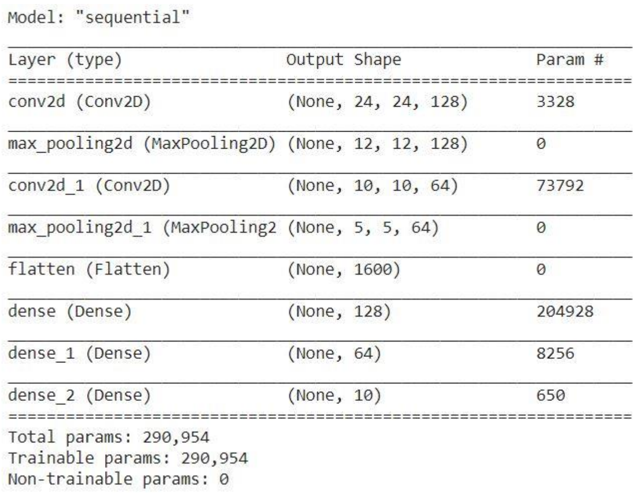


Figure 1: MNIST handwritten dataset

The term CNN stands for Convolution Neural Network. CNN is a deep learning technique to classify the input automatically. It has gained popularity by detecting the image data. The convolution layer extracts information from nearby pixels to down sample the picture into features, which are then predicted target values by prediction layers. In this implementation, multiple convolution filters were used to run over the image in order to compute the dot product. Each filter shows the different information from the image. The filter is taken to perform an element wise multiplication between the image pixel values that match the size of kernel. The data set was divided into a train and validation dataset. In this project, there is an input layer followed by two hidden layers and three dense layers where the last layer is output layer. A flatten layer is also used in the project. Activation functions of a model are responsible for making decision. In a deep neural network like CNN, there are many neurons, and based on activation functions, neurons fire up and the network moves forward. Here is the model below:



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 24, 24, 128)       3328
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 12, 12, 128)       0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 10, 10, 64)        73792
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 1600)              0
_________________________________________________________________
dense (Dense)                (None, 128)               204928
_________________________________________________________________
dense_1 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_2 (Dense)              (None, 10)                650
=================================================================
Total params: 290,954
Trainable params: 290,954
Non-trainable params: 0
```

Figure 2: sequential model structure.

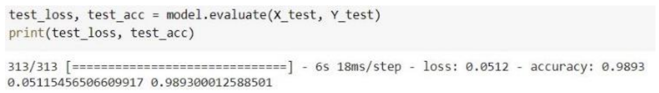
Once the model has been developed, it is time to compile and fit it. During the fitting process, the model will go over the dataset and understand the connections. It will learn as many times as the epoch has been defined. In this project epoch is defined 10. This CNN model will learn automatically and also make wrong prediction. This wrong prediction is represented as the loss value. The shorter loss will generate high accuracy.
Higher accuracy of model can be found at the end of the last epoch.

II. RESULTS

In this project, same model is used for different optimizers like Adam, SGD, RSMprop to measure the difference of accuracy and loss. Also, matplotlib is used to show the graphical representation of the data accuracy and loss. The epoch rate is 10 for every optimizer with validation split 0.2. The summary of the result is shown below:

Optimizer	Accuracy (%)	Loss (%)
Adam	98.93	5.12
SGD	99.36	3.3
RMSprop	99.39	10.4

While compiling the model in Adam optimizer, test accuracy is 98.93% where the loss is about 5.12%.



```
test_loss, test_acc = model.evaluate(X_test, Y_test)
print(test_loss, test_acc)

313/313 [==============================] - 6s 18ms/step - loss: 0.0512 - accuracy: 0.9893
0.05115456506609917 0.989300012588501
```

Figure 3: Accuracy and loss of Adam optimizer

And the graphical representation is given below:

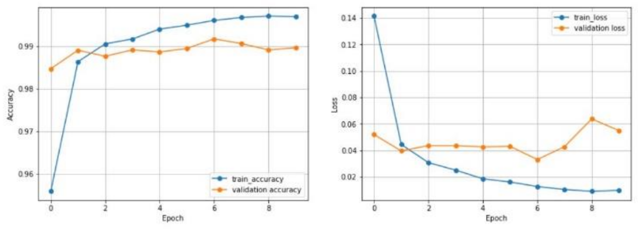
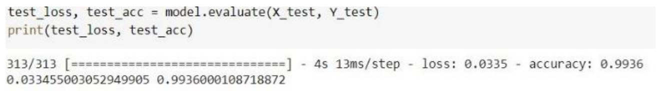


Figure 4: Accuracy & loss graph of Adam optimizer

In SGD optimizer, accuracy is 99.36% where the loss is 3.3%.



```
test_loss, test_acc = model.evaluate(X_test, Y_test)
print(test_loss, test_acc)

313/313 [==============================] - 4s 13ms/step - loss: 0.0335 - accuracy: 0.9936
0.033455003052949905 0.9936000108718872
```

Figure 5: Accuracy & loss of SGD optimizer

The graphical representation is given below:

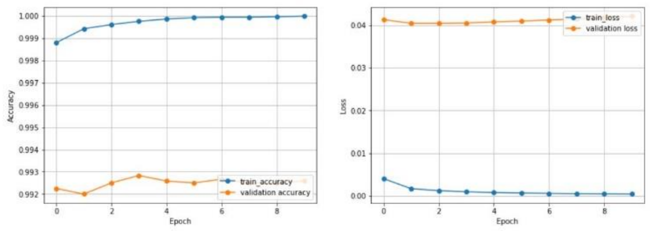
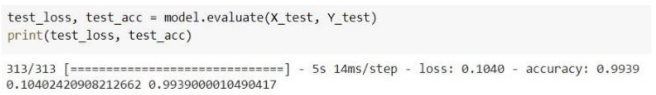


Figure 6: Accuracy & loss graph of SGD optimizer

In RMSprop, accuracy is 99.39% and the loss is about 10.4%.



```
test_loss, test_acc = model.evaluate(X_test, Y_test)
print(test_loss, test_acc)

313/313 [==============================] - 5s 14ms/step - loss: 0.1040 - accuracy: 0.9939
0.10402420908212662 0.9939000010490417
```

Figure 7: Accuracy & loss of RMSprop optimizer

The line graph of RMSprop is given below:

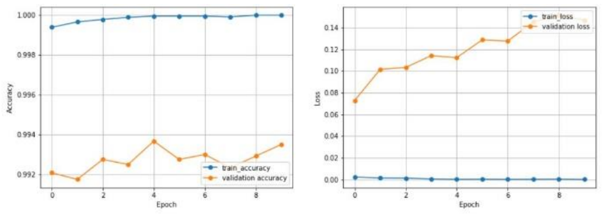


Figure 8: Accuracy and loss graph of RMSprop

By comparing these three optimizers, it can be said that the accuracy is much higher in RMSprop optimizer and we get less accuracy in Adam. But the loss is much higher in RMSprop in compare to other optimizers.

III. DISCUSSION

As we get a result by using different optimizers that doesn't mean the result will not change. The design of the model can be changed in order to get more accurate result. Also, it depends on tuning the layers and parameters. By analyzing all the result, if SGD optimizer is more efficient than the other two optimizers. It has a higher accuracy rate with 3.3% loss. Also, it has almost same accuracy as RMSprop optimizer. The objective of this project is to achieve 98% accuracy which is gained successfully with the designed model also various results of the optimizers has been shown.