Computer Vision and Pattern Recognition

Mid Project Report
Section-B

MD Shakhwat Hossain

ID: 17-35332-2

Title: Implementation of CNN architecture to classify the MNIST handwritten dataset.

Abstract:

The CNN (Convolutional Neural Network) models are inescapable in the image data space. This work phenomenally well on image classification, object detection, image recognition. In this project CNN architecture is implemented by classifying MNIST (Modified National Institute of Standards and Technology) handwritten digit dataset to get accuracy over 98% using different optimizers.

Introduction:

A Convolutional neural network (CNN) is a Deep Learning algorithm which can take an input image, assign importance (learnable weights and biases) to various aspects in the image and able to differentiate one from the other. The pre-processing required in a Convolutional neural network (CNN) is much lower as compared to other classification algorithms. Convolutional neural network (CNN) is a neural network that has more than one convolutional layer. These are used mainly for image processing, classification, segmentation and also for other auto correlated data. The layers of a CNN consist of an input layer, a hidden layer and an output layer that includes multiple convolutional layers, pooling layers, fully connected layers and normalization layers. The removal of limitations and increase in efficiency for image processing results in a system that is far more effective, simpler to trains limited for image processing and natural language processing.

There are various datasets that can be achieved the purpose for applying convolutional neural networks. MNIST is one of them. Handwritten digits on MNIST Dataset are classified certainly using CNN for best accuracy result. MNIST is composed of images that are handwritten digits (0-9**),** split into a training set of 60,000 images and a test set of 10,000 where each image is of 28 x 28 pixels in width and height. To build image classification model using a neural network need to flatten the input image dimensions to 1D (width pixels x height pixels). Then normalize the image pixel values to get gray scale dimension. After that building a Sequential model architecture with Dense layers. To get result have to training the model and making predictions.

Model Summary:

```
Model: "sequential_28"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_24 (Conv2D)           (None, 26, 26, 64)        640       
_________________________________________________________________
max_pooling2d_24 (MaxPooling (None, 13, 13, 64)        0         
_________________________________________________________________
conv2d_25 (Conv2D)           (None, 11, 11, 32)        18464     
_________________________________________________________________
max_pooling2d_25 (MaxPooling (None, 5, 5, 32)          0         
_________________________________________________________________
flatten_12 (Flatten)         (None, 800)               0         
_________________________________________________________________
dense_72 (Dense)             (None, 64)                51264     
_________________________________________________________________
dense_73 (Dense)             (None, 10)                650       
=================================================================
Total params: 71,018
Trainable params: 71,018
Non-trainable params: 0

_________________________________________________________________
```

With the Dense layers there are Conv2D, MaxPooling and Flatten layers Sequential Model has been built. These layers have been used in respective segments.

```
input layer
    keras.Input(shape=(28,28,1)),
hidden layer
    layers.Conv2D(filters=64, kernel_size=(3,3), activation='relu'),
    layers.MaxPooling2D(pool_size=(2,2)),

    layers.Conv2D(filters=32, kernel_size=(3,3), activation='relu'),
    layers.MaxPooling2D(pool_size=(2,2)),
       layers.Flatten(),
       layers.Dense(64, activation='relu'),
output layer
     keras.layers.Dense(units=10, activation='softmax')
```

Three different optimizers have been used to get best accuracy result. These are Adam, SGD and RMSProp.

Results:

Using three optimizers the result has been found –

Optimizer	Train Accuracy	Train Loss	Validation Accuracy	Validation Loss	Test Accuracy
Adam	99.25%	2.4%	98.58%	6.96%	98.54%
SGD	100%	6.88*10^-3%	98.90%	9.05%	99.08%
RMSprop	99.89%	0.81%	98.97%	21.11%	98.87%

Using Adam Optimizer, the train, validation accuracy and loss are demonstrating in the graph below-

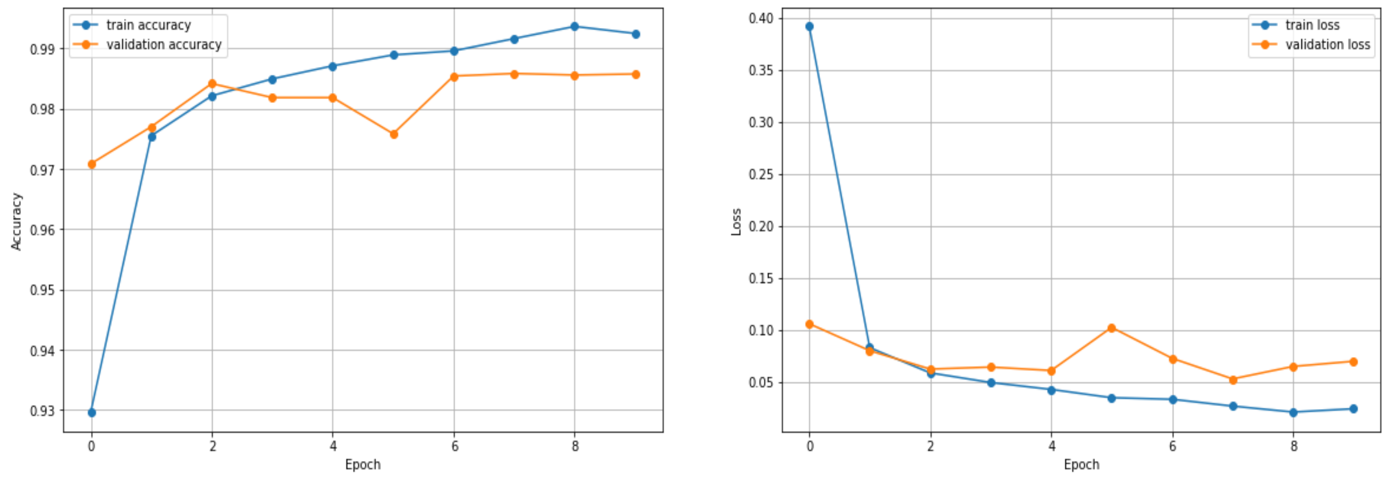

Figure: Graph of Adam optimizer

Using SGD Optimizer, the train, validation accuracy and loss are demonstrating in the graph below-

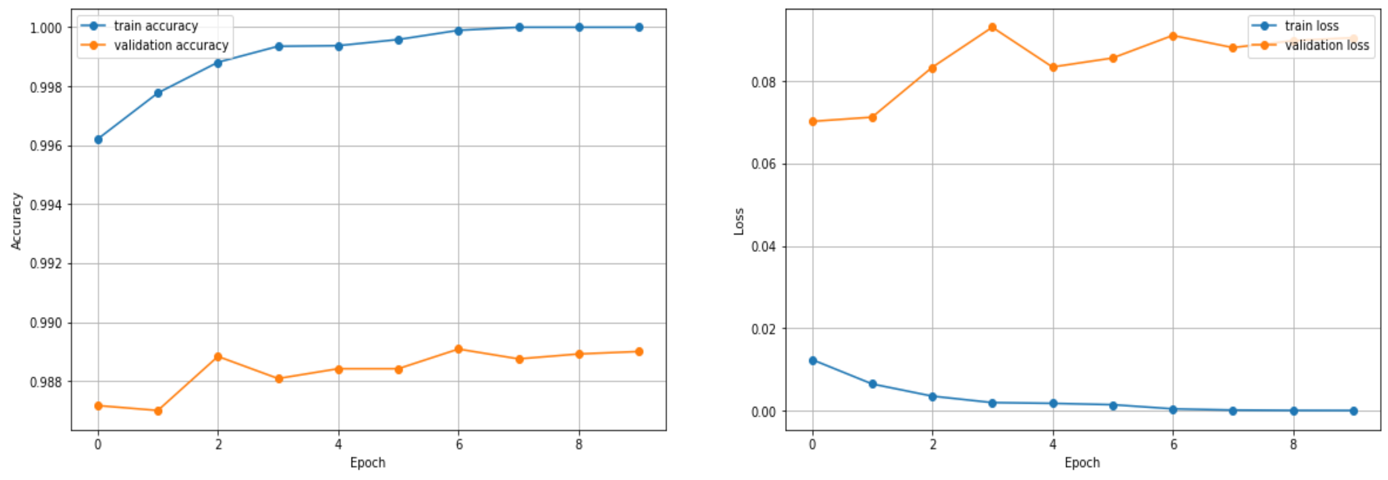

Figure: Graph of SGD optimizer

Using RMSprop Optimizer, the train, validation accuracy and loss are demonstrating in the graph below-

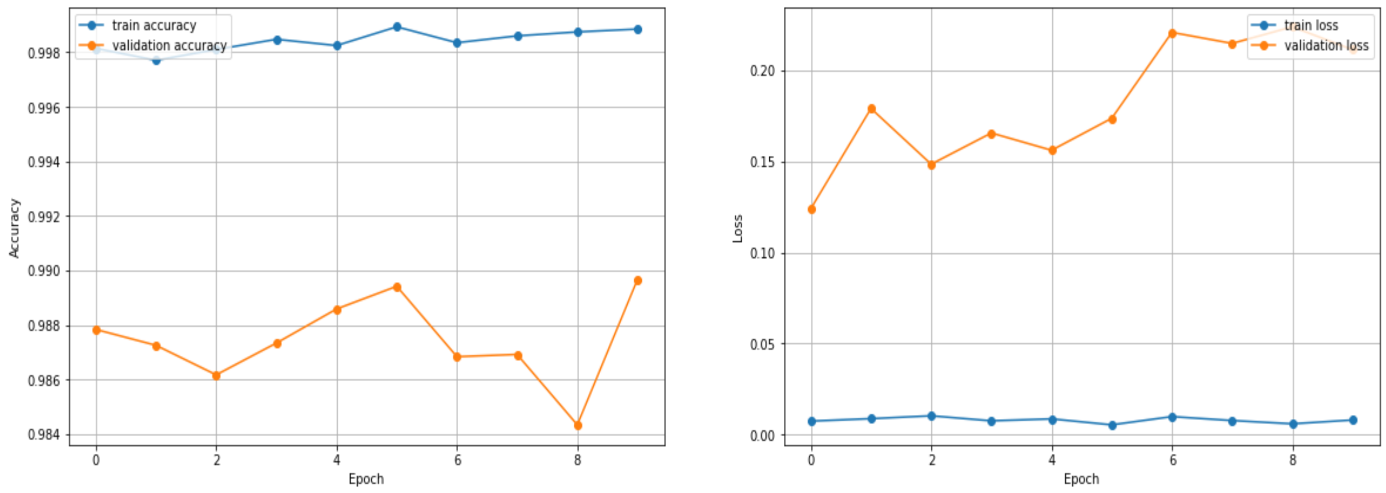

Figure: Graph of RMSprop optimizer

Discussion:

Implementing the three optimizers we have found expected accuracy over 98%. 98.54% of accuracy is on Adam optimizer where the training and validation accuracy is respectively 99.25% & 98.58%.

In contrast, 99.08% of accuracy is on SGD optimizer where the training and validation accuracy is respectively 100% & 98.90%.

On the contrary, RMSprop gave of accuracy 98.87%, while the training and validation accuracy is respectively 99.89% & 98.97%

Analyzing the result of the three optimizers we can see that; SGD has got the highest accuracy of 99.08% whereas other two are slightly lower than it. However, SGD has more training and validation loss than Adam. Moreover, RMSprop has more accuracy but its validation loss is the highest of these three. Thus we can say that Adam gave the best accuracy of over 98% considering its all segments.