ABSTRACT

The main objective of this project is to familiar with Convolutional Neural Network (CNN) which is a Deep Learning Algorithm used for image classification. In this project, TensorFlow has been used to build the Model with Convolution layer. The “MNIST Handwritten Digit” dataset has been used to train and test the Model. KERAS from TensorFlow is used to import and load the “MNIST Handwritten Digit” dataset. The Model is created with two Convolution layer. Three optimizers have been used to compile the sequential Model separately such as, “Adam”, “SGD”, "RMSprop”.

INTRODUCTION

The Convolutional Neural Network (CNN) is a Deep Learning Algorithm which is a class of Deep Neural Network used to classify or analyze the images. It takes images as input then the image passes some hidden layers. The CNN algorithm is used in those hidden layers which is known as Convolution Layer and then we get the successful classified images as output. “TensorFlow” is used in this project to build the classification model with Convolution layer. TensorFlow is an open-source software library which is used to develop and train Machine Learning Models. From TensorFlow, KERAS is used to load the dataset (MNIST Handwritten Digit dataset) and develop the model. KERAS is a software library which is an interface provided by the TensorFlow library. In this project the “MNIST Handwritten Digit” dataset is used from KERAS. This dataset has 60,000 train data (images) and 10,000 test data (images). The pixel size of every image is same which is (28 x 28). The main objective of this project is to develop a machine learning image classification model with CNN algorithm (Convolution Layer) using TensorFlow and KERAS library.

RESULTS

The Model is created with two Convolution layer. Three optimizers have been used to compile the model such as Adam, SGD, RMSprop.

The Model has been trained with the Train data (60,000 train data (images)) of MNIST Handwritten Digit dataset with 12 epochs. And the Test Data (10, 000 test data (images)) of MNIST Handwritten Digit dataset is used to evaluate/test the trained model. The results are given below:

#Using Optimizer “SGD” in Model compilation, we get:

Train Accuracy = 100%

Validation Accuracy (Train) = 99.7%

Test Accuracy = 99.28 %

Test Loss = 4.69 %

```
Epoch 8/12
1500/1500 [==============================] - 7s 5ms/step - loss: 6.7330e-05 - accuracy: 1.0000 - val_loss: 0.0106 - val_accuracy: 0.9975
Epoch 9/12
1500/1500 [==============================] - 7s 5ms/step - loss: 6.2378e-05 - accuracy: 1.0000 - val_loss: 0.0105 - val_accuracy: 0.9976
Epoch 10/12
1500/1500 [==============================] - 7s 5ms/step - loss: 5.8215e-05 - accuracy: 1.0000 - val_loss: 0.0105 - val_accuracy: 0.9976
Epoch 11/12
1500/1500 [==============================] - 7s 5ms/step - loss: 5.4626e-05 - accuracy: 1.0000 - val_loss: 0.0105 - val_accuracy: 0.9976
Epoch 12/12
1500/1500 [==============================] - 7s 5ms/step - loss: 5.1513e-05 - accuracy: 1.0000 - val_loss: 0.0105 - val_accuracy: 0.9976
<keras.callbacks.History at 0x7fa0c07378d0>
```

```
x_test = x_test.reshape(-1, 28, 28, 1)
test_loss, test_accuracy = model.evaluate(x_test, y_test)
print('\nTest Accuracy:', test_accuracy)
print('\nTest Loss:', test_loss)
```

```
313/313 [==============================] - 1s 3ms/step - loss: 0.0469 - accuracy: 0.9928

Test Accuracy: 0.9927999973297119

Test Loss: 0.04690428823232651
```

Figure 1: Training and Test Result using Optimizer “SGD” in Model compilation

#Using Optimizer “Adam” in Model compilation, we get:

Train Accuracy = 99.75 %

Validation Accuracy (Train) = 99.08 %

Test Accuracy = 99.2 %

Test Loss = 3.48 %

```
Epoch 9/12
1500/1500 [==============================] - 7s 5ms/step - loss: 0.0099 - accuracy: 0.9965 - val_loss: 0.0471 - val_accuracy: 0.9893
Epoch 10/12
1500/1500 [==============================] - 7s 5ms/step - loss: 0.0073 - accuracy: 0.9977 - val_loss: 0.0441 - val_accuracy: 0.9910
Epoch 11/12
1500/1500 [==============================] - 8s 5ms/step - loss: 0.0070 - accuracy: 0.9979 - val_loss: 0.0476 - val_accuracy: 0.9897
Epoch 12/12
1500/1500 [==============================] - 8s 5ms/step - loss: 0.0081 - accuracy: 0.9975 - val_loss: 0.0472 - val_accuracy: 0.9908
<keras.callbacks.History at 0x7f4e0c2f3650>
```

```
x_test = x_test.reshape(-1, 28, 28, 1)
test_loss, test_accuracy = model.evaluate(x_test, y_test)
print('\nTest Accuracy:', test_accuracy)
print('\nTest Loss:', test_loss)
```

```
313/313 [==============================] - 1s 4ms/step - loss: 0.0348 - accuracy: 0.9920

Test Accuracy: 0.9919999837875366

Test Loss: 0.03481189161539078
```

Figure 2: Training and Test Result using Optimizer “Adam” in Model compilation

#Using Optimizer "RMSprop" in Model compilation, we get:

Train Accuracy = 99.76 %

Validation Accuracy (Train) = 98.87 %

Test Accuracy = 99.1 %

Test Loss = 4.43 %

```
1500/1500 [==============================] - 8s 5ms/step - loss: 0.0121 - accuracy: 0.9964 - val_loss: 0.0478 - val_accuracy: 0.9908
[ ] Epoch 10/12
1500/1500 [==============================] - 9s 6ms/step - loss: 0.0100 - accuracy: 0.9969 - val_loss: 0.0740 - val_accuracy: 0.9871
Epoch 11/12
1500/1500 [==============================] - 9s 6ms/step - loss: 0.0086 - accuracy: 0.9974 - val_loss: 0.0602 - val_accuracy: 0.9891
Epoch 12/12
1500/1500 [==============================] - 8s 5ms/step - loss: 0.0086 - accuracy: 0.9976 - val_loss: 0.0714 - val_accuracy: 0.9887
<keras.callbacks.History at 0x7f78400a0d90>

[ ] x_test = x_test.reshape(-1, 28, 28, 1)
test_loss, test_accuracy = model.evaluate(x_test, y_test)
print('\nTest Accuracy:', test_accuracy)
print('\nTest Loss:', test_loss)

313/313 [==============================] - 1s 4ms/step - loss: 0.0443 - accuracy: 0.9910

Test Accuracy: 0.9909999966621399

Test Loss: 0.04427473992109299
```

Figure 3: Training and Test Result using Optimizer "RMSprop" in Model compilation

DISCUSSION

The MNIST Handwritten Digit dataset is used for this project. This dataset has 60,000 train data (images) and 10,000 (images) test data. Both train and test data were preprocessed before training with the model by dividing all data of the dataset by 255 in order to decrease the pixel range (0-255) to (0-1).

```
[ ] x_train = x_train.astype('float32') / 255
    x_test = x_test.astype('float32') / 255
```

Figure 4: Preprocessing the data

Sequential model is used to build the classification model form KERAS, TensorFlow. Convolution layer is used form KERAS, TensorFlow (Conv2D ()) as hidden layer since the objective is to use the CNN algorithm. Input shape is (28,28,1) where (28,28) indicates the pixel size of the image and (,1) indicates the Grayscale image. For Max Polling, pool size is given (2x2) matrix and

Activation Function “RELU” is used for the model except Output layer. In output layer Activation Function “SOFTMAX” is used. In output layer, Dense layer is used with 10 Units because the dataset we used it has (0-9) digit total 10 digit to classify. Before using the Dense layer, a Flatten layer is used to maintain the consistency between Convolution layer and Dense layer.

In [ ]:
```
model = tf.keras.Sequential([
     tf.keras.Input(shape= (28, 28, 1)),
     tf.keras.layers.Conv2D(filters=28, kernel_size=(5,5), activation='relu'),
     tf.keras.layers.MaxPool2D(pool_size=(2,2)),
     tf.keras.layers.Conv2D(filters=56, kernel_size=(3,3), activation='relu'),
     tf.keras.layers.MaxPool2D(pool_size=(2,2)),


     tf.keras.layers.Flatten(),
     tf.keras.layers.Dense(64, activation='relu'),
     tf.keras.layers.Dense(10, activation='softmax')
])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 24, 24, 28)        728
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 12, 12, 28)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 10, 10, 56)        14168
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 56)          0
_________________________________________________________________
flatten (Flatten)            (None, 1400)              0
_________________________________________________________________
dense (Dense)                (None, 64)                89664
_________________________________________________________________
dense_1 (Dense)              (None, 10)                650
=================================================================
Total params: 105,210
Trainable params: 105,210
Non-trainable params: 0
```

Figure 5: Sequential Model developed with Convolution Layer (CNN algorithm)

The model is compiled with 3 types of optimizers such as “Adam”, “SGD”, “RMSprop” separately.

Some difficulties were found while training the data set with the model. The “fit ()” method is used on the training data to train the model with 12 epochs. And every time the code ended up with some sort of dimension error such as, **“ValueError: Input 0 of layer sequential is incompatible with the layer: : expected min_ndim=4, found ndim=3. Full shape received: (32, 28, 28)”.**

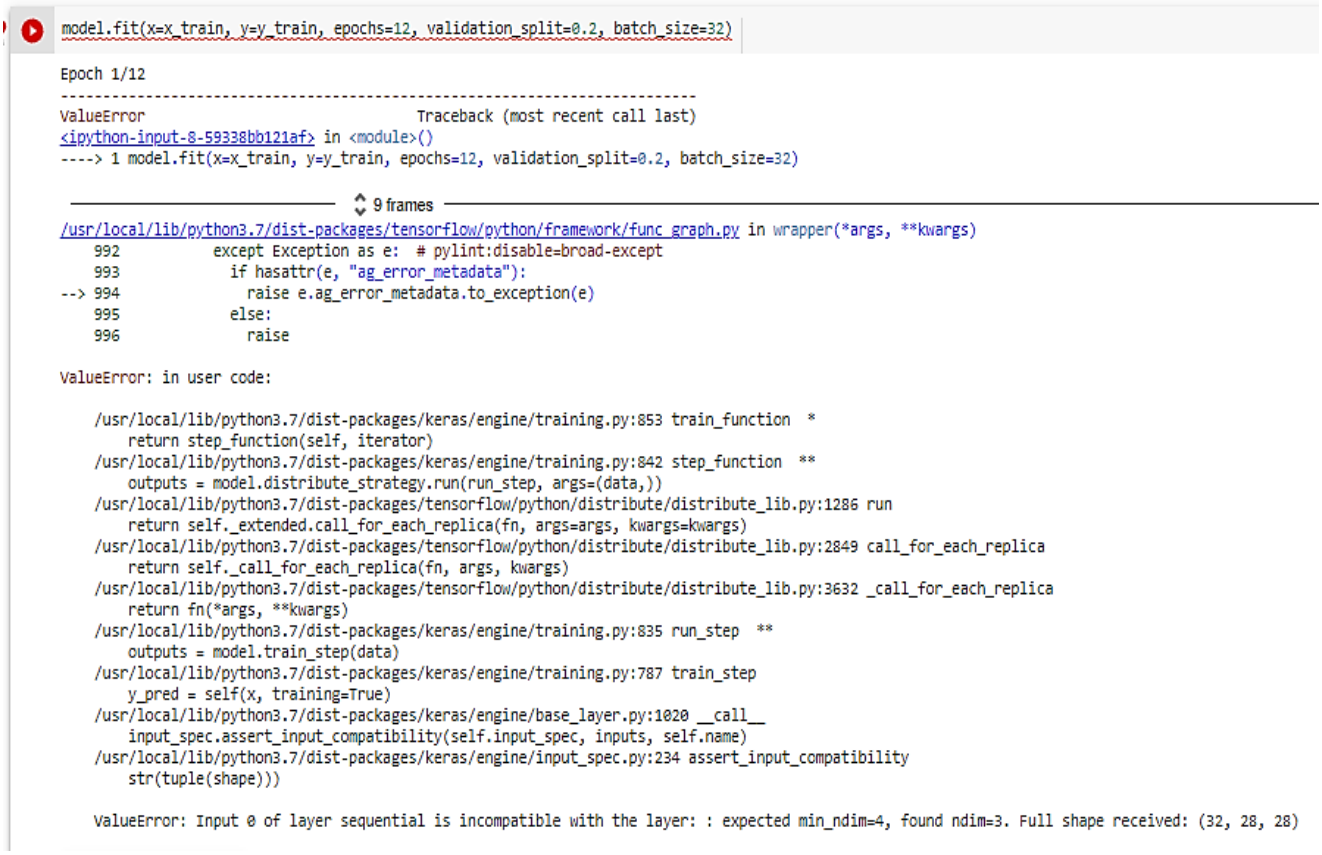

Figure 6: Dimension Error while training the model with training data.

Expected minimum dimension of the data was 4 but the model received is 3. To fix this error, I had to reshape the training data using “reshape ()” function to make the dimension of the data 3 to 4.

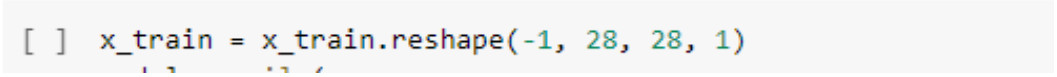

Figure 7: Reshaping the training data.

After reshaping the data, the training of the model worked properly with the dataset and got the training accuracy of 100% using “SGD” optimizer, 99.75% using “Adam” optimizer and 99.76% using “RMSprop”. [All results are shown in **RESULTS** section].

Before test the model the test data also had to reshaped with the “reshape ()” function.

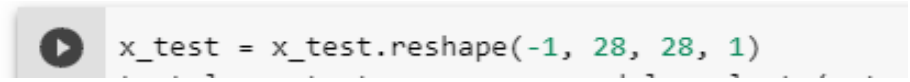

Figure 8: Reshaping the test data.

To test the model the “evaluate ()” method is used with the test data and got the test accuracy of 99.28% using “SGD” optimizer, 99.2% using “Adam” optimizer and 99.1% using “RMSprop”. [All results are shown in **RESULTS** section].