Team Members Names & IDs:

Team Member Name	Team Member ID	
محمد عبد الرحيم ابراهيم محمد	201900698	
محمد خميس احمد عبدالجيد	201900666	
محمد مبارك حسين احمد	201900719	
مصطفى عصام عبدالفتاح ابوشامه	201900824	
احمد مصطفى اسماعيل علام	201900103	
مارك فايز وديع قسطنطين	201900597	

Introduction:

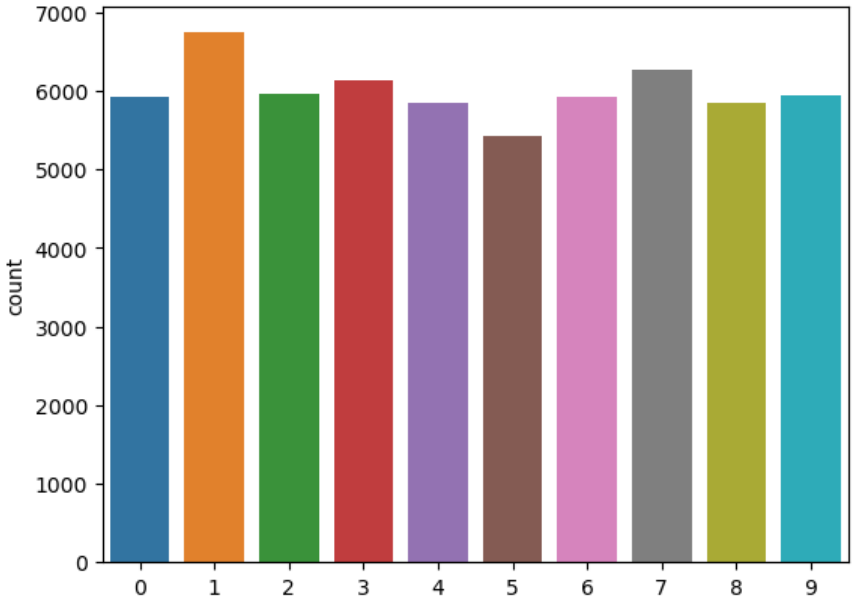
Handwritten digit recognition is a classic problem in the field of image classification. The MNIST dataset, consisting of 60,000 training and 10,000 testing grayscale images of handwritten digits.

In this document, we will explore the use of RNNs for MNIST digit classification by using the popular Long Short-Term Memory (LSTM).

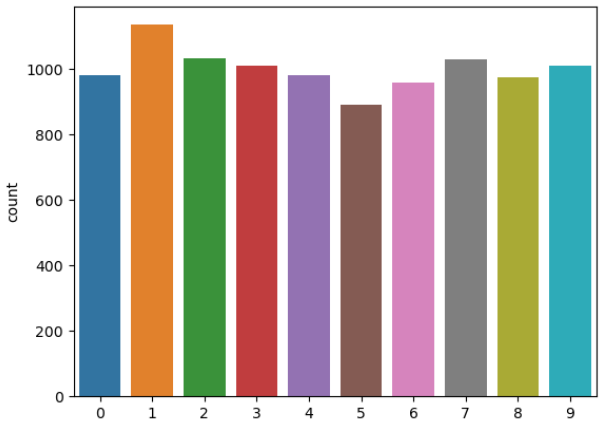
We have changed the hyperparameters in model many times to get the best results.

Analysis & Distribution Data

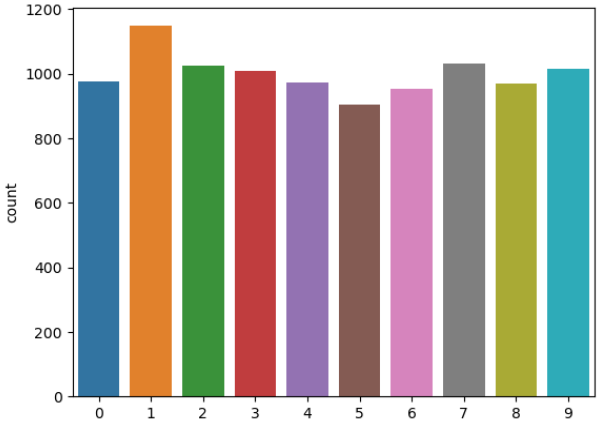
Training Data



Testing Data



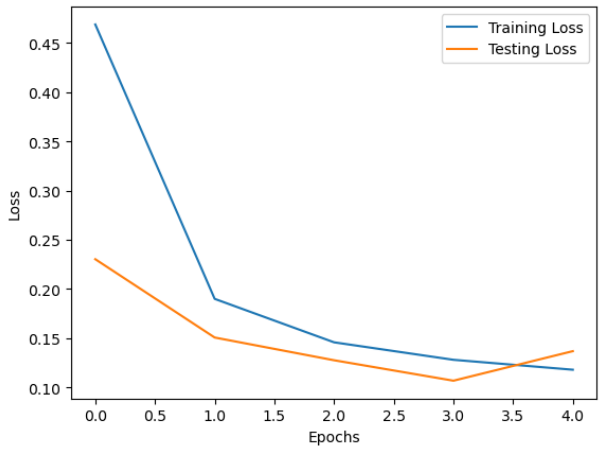
Predicting Data

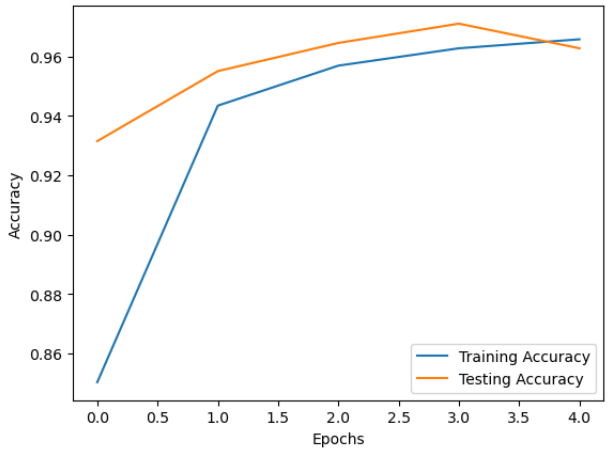


Charts & Insights

```
In [5]:
model = Sequential()
model.add(BatchNormalization())

model.add(SimpleRNN(128, activation = 'relu', input_shape = (28, 28, 1)))

model.add(Dense(64, activation = 'relu'))
model.add(Dense(10, activation = 'softmax'))

model.compile(optimizer = 'adam', loss = 'sparse_categorical_crossentropy', metrics = ['accuracy'])
history = model.fit(X_train, y_train, validation_data = (X_test, y_test), epochs = 5, batch_size = 64)

y_pred = model.predict(X_test)

model.summary()
```

```
Epoch 1/5
938/938 [==============================] - 11s 10ms/step - loss: 0.4687 - accuracy: 0.8502 - val_loss: 0.2302 - val_accuracy: 0.9315
Epoch 2/5
938/938 [==============================] - 10s 11ms/step - loss: 0.1900 - accuracy: 0.9435 - val_loss: 0.1507 - val_accuracy: 0.9551
Epoch 3/5
938/938 [==============================] - 10s 10ms/step - loss: 0.1458 - accuracy: 0.9570 - val_loss: 0.1276 - val_accuracy: 0.9646
Epoch 4/5
938/938 [==============================] - 10s 11ms/step - loss: 0.1281 - accuracy: 0.9628 - val_loss: 0.1068 - val_accuracy: 0.9711
Epoch 5/5
938/938 [==============================] - 10s 11ms/step - loss: 0.1180 - accuracy: 0.9658 - val_loss: 0.1368 - val_accuracy: 0.9628
313/313 [==============================] - 1s 3ms/step
Model: "sequential_1"
```





When we use SimpleRNN technique and change unites to 128, activation = ‘relu ’, and we use 2 layers of dense, the first one has 64 units and activation = ‘relu ’ the second one have 10 units (number of Digits 0-9) and activation = ‘softmax ’. we use optimizer ‘adam’, loss ‘sparse_categorical_crossentropy’, batch size = 64.

We found that Training Accuracy = 96.58%

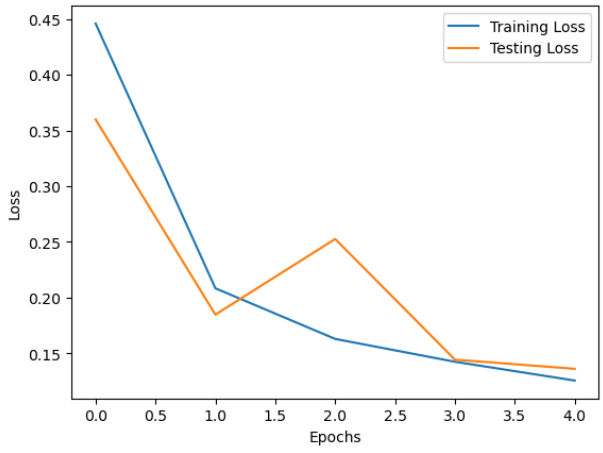
We found that Validation Accuracy = 96.28%

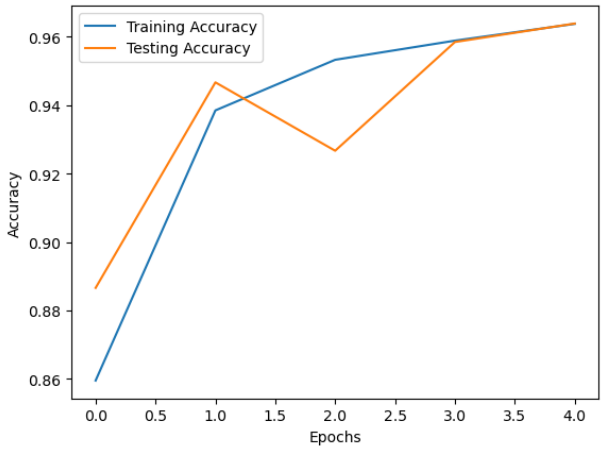
……

In [11]:

```
model = Sequential()
model.add(BatchNormalization())

model.add(SimpleRNN(128, activation = 'tanh', input_shape = (28, 28, 1)))

model.add(Dense(64, activation = 'tanh'))
model.add(Dense(10, activation = 'softmax'))

model.compile(optimizer = 'rmsProp', loss = 'sparse_categorical_crossentropy', metrics = ['accuracy'])
history = model.fit(X_train, y_train, validation_data = (X_test, y_test), epochs = 5, batch_size = 64)

y_pred = model.predict(X_test)

model.summary()
```

```
Epoch 1/5
938/938 [==============================] - 11s 10ms/step - loss: 0.4460 - accuracy: 0.8595 - val_loss: 0.3598 - val_accuracy: 0.8866
Epoch 2/5
938/938 [==============================] - 10s 10ms/step - loss: 0.2082 - accuracy: 0.9385 - val_loss: 0.1846 - val_accuracy: 0.9467
Epoch 3/5
938/938 [==============================] - 10s 10ms/step - loss: 0.1628 - accuracy: 0.9533 - val_loss: 0.2525 - val_accuracy: 0.9267
Epoch 4/5
938/938 [==============================] - 10s 10ms/step - loss: 0.1423 - accuracy: 0.9589 - val_loss: 0.1442 - val_accuracy: 0.9585
Epoch 5/5
938/938 [==============================] - 10s 10ms/step - loss: 0.1254 - accuracy: 0.9638 - val_loss: 0.1359 - val_accuracy: 0.9639
313/313 [==============================] - 1s 3ms/step
Model: "sequential_3"
```





When we use SimpleRNN technique and change unites to 128, activation = ‘tanh ’, and we use 2 layers of dense, the first one has 64 units and activation = ‘tanh ’ the second one have 10 units (number of Digits 0-9) and activation = ‘softmax ’. we use optimizer ‘rmsProp, loss ‘sparse_categorical_crossentropy’, batch size = 64.

We found that Training Accuracy = 96.38%

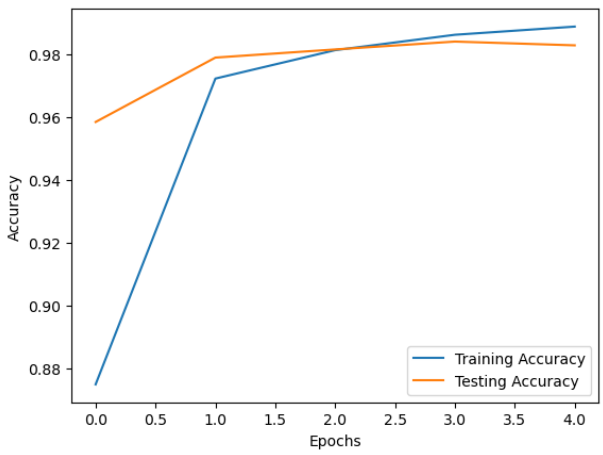
We found that Validation Accuracy = 96.39%

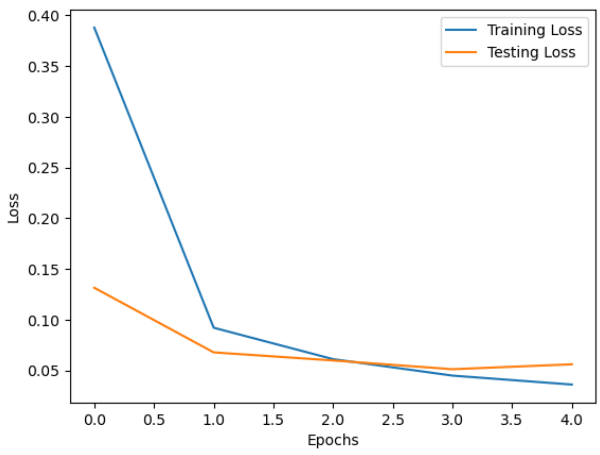
……

In [22]:

```
model = Sequential()
model.add(BatchNormalization())

model.add(LSTM(128, activation = 'tanh', input_shape = (28, 28, 1)))

model.add(Dense(64, activation = 'tanh'))
model.add(Dense(10, activation = 'softmax'))

model.compile(optimizer = 'adam', loss = 'sparse_categorical_crossentropy', metrics = ['accuracy'])
history = model.fit(X_train, y_train, validation_data = (X_test, y_test), epochs = 5, batch_size = 128)

y_pred = model.predict(X_test)

model.summary()
```

```
Epoch 1/5
469/469 [==============================] - 17s 30ms/step - loss: 0.3878 - accuracy: 0.8749 - val_loss: 0.1314 - val_accuracy: 0.9585
Epoch 2/5
469/469 [==============================] - 14s 30ms/step - loss: 0.0920 - accuracy: 0.9723 - val_loss: 0.0678 - val_accuracy: 0.9790
Epoch 3/5
469/469 [==============================] - 14s 30ms/step - loss: 0.0611 - accuracy: 0.9814 - val_loss: 0.0598 - val_accuracy: 0.9816
Epoch 4/5
469/469 [==============================] - 14s 31ms/step - loss: 0.0449 - accuracy: 0.9863 - val_loss: 0.0512 - val_accuracy: 0.9841
Epoch 5/5
469/469 [==============================] - 15s 32ms/step - loss: 0.0361 - accuracy: 0.9889 - val_loss: 0.0560 - val_accuracy: 0.9829
313/313 [==============================] - 4s 8ms/step
Model: "sequential_6"
```





When we use LSTM technique and change unites to 128, activation = ‘tanh ’, and we use 2 layers of dense, the first one has 64 units and activation = ‘tanh ’ the second one have 10 units (number of Digits 0-9) and activation = ‘softmax ’. we use optimizer ‘adam’, loss ‘sparse_categorical_crossentropy’, batch size = 128.

We found that Training Accuracy = 98.89%

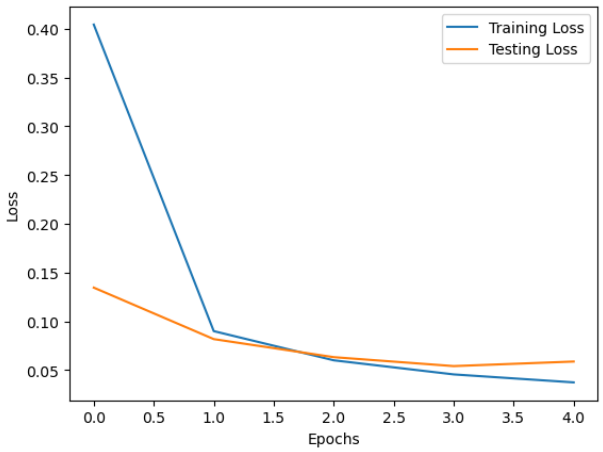
We found that Validation Accuracy = 98.29%

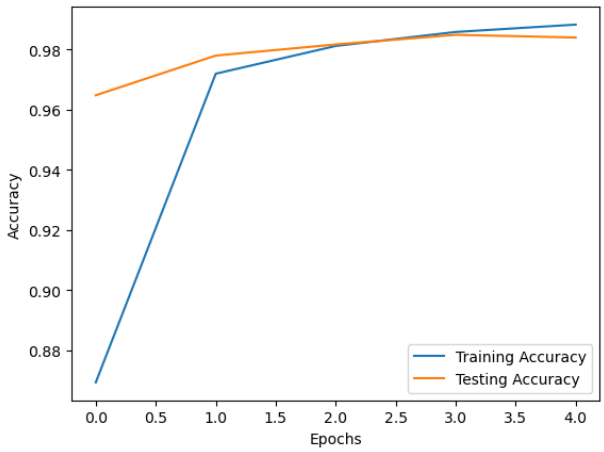
……

In [3]:

```
model = Sequential()
model.add(BatchNormalization())

model.add(LSTM(256, activation = 'relu', input_shape = (28, 28, 1)))

# model.add(Dense(64, activation = 'relu'))
model.add(Dense(10, activation = 'softmax'))

model.compile(optimizer = 'adam', loss = 'sparse_categorical_crossentropy', metrics = ['accuracy'])
history = model.fit(X_train, y_train, validation_data = (X_test, y_test), epochs = 5, batch_size = 128)

y_pred = model.predict(X_test)

model.summary()
```

```
Epoch 1/5
469/469 [==============================] - 35s 72ms/step - loss: 0.4042 - accuracy: 0.8693 - val_loss: 0.1346 - val_accuracy: 0.9648
Epoch 2/5
469/469 [==============================] - 34s 73ms/step - loss: 0.0900 - accuracy: 0.9720 - val_loss: 0.0818 - val_accuracy: 0.9780
Epoch 3/5
469/469 [==============================] - 35s 74ms/step - loss: 0.0601 - accuracy: 0.9812 - val_loss: 0.0633 - val_accuracy: 0.9817
Epoch 4/5
469/469 [==============================] - 34s 72ms/step - loss: 0.0456 - accuracy: 0.9858 - val_loss: 0.0542 - val_accuracy: 0.9849
Epoch 5/5
469/469 [==============================] - 35s 74ms/step - loss: 0.0375 - accuracy: 0.9883 - val_loss: 0.0588 - val_accuracy: 0.9840
313/313 [==============================] - 4s 11ms/step
Model: "sequential"
```



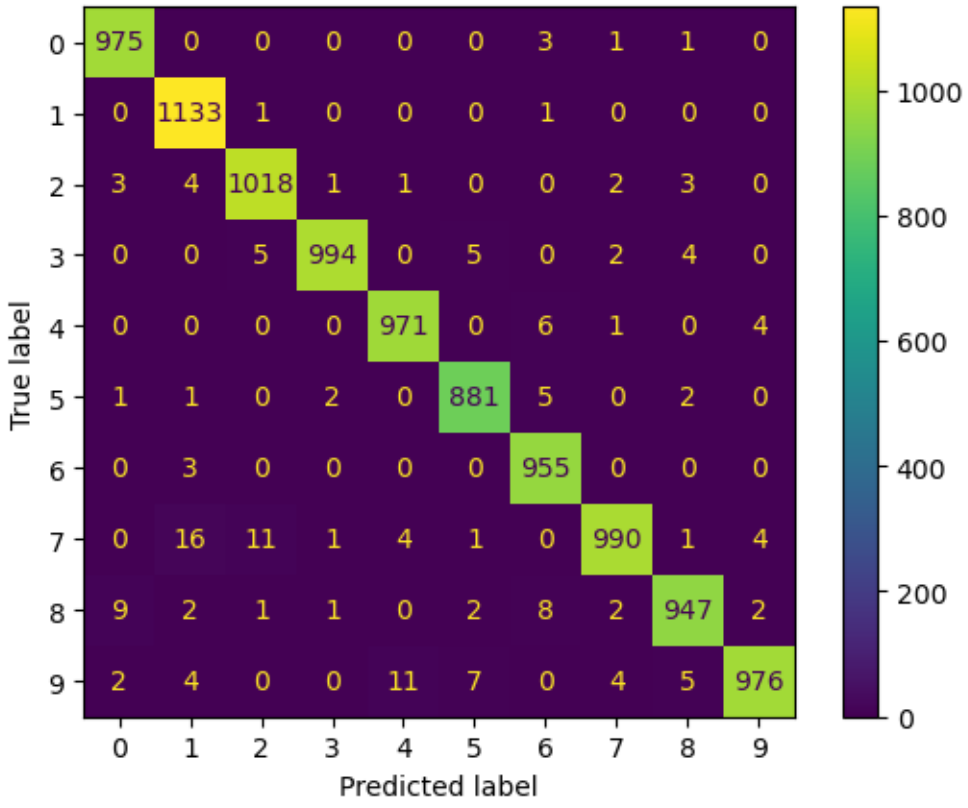


When we use LSTM technique and change unites to 256, activation = ‘relu ’, and we use one layers of dense, it is the output layer have 10 units (number of Digits 0-9) and activation = ‘softmax ’. we use optimizer ‘adam’, loss ‘sparse_categorical_crossentropy’, batch size = 128.

We found that Training Accuracy = 98.83%

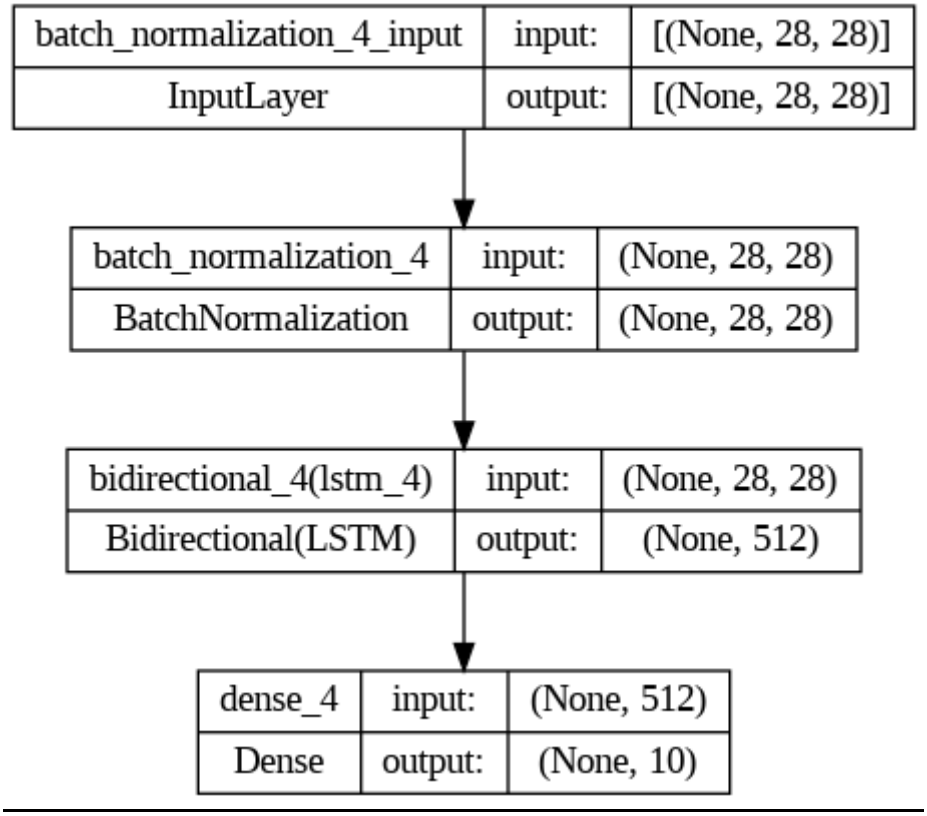
We found that Validation Accuracy = 98.40%

<u>Confusion Matrix:</u>



……

Plot Model:


batch_normalization_4_input
input:
[(None, 28, 28)]
InputLayer
output:
[(None, 28, 28)]
batch_normalization_4
input:
(None, 28, 28)
BatchNormalization
output:
(None, 28, 28)
bidirectional_4(lstm_4)
input:
(None, 28, 28)
Bidirectional(LSTM)
output:
(None, 512)
dense_4
input:
(None, 512)
Dense
output:
(None, 10)