Name: Adham Ali
ID: 5703031

CAP4630 Assignment 5 Report

Note: I will give you three jupyter notebooks, one for 8 neurons, 128 neurons and a model with a 99% F1 score or higher.

There are minor differences in every notebook, the difference will be in the category "**Build the Neural Network Model**" and **"Train the Neural Network (Only for a model with a 99% F1 score or higher)"**

Part 1:

Finished, I explained the code and the concept of it in detail.

Part 2:

→The code used:

```
model = Sequential([
    # Input layer for flattened 28x28 pixel images (784 total inputs)
    Input(shape=(784,)),

    # Hidden layer with 8 neurons.
    # ReLU activation introduces non-linearity and helps the model learn complex patterns in the data.
    Dense(8, activation='relu'),

    # Output layer with 10 neurons (one for each digit class from 0 to 9).
    # Softmax activation converts the output to probability values summing to 1.
    Dense(10, activation='softmax')
])
```

```
# Train the model using training data and evaluate on test data after each epoch.
# validation_data: used to calculate val_accuracy and val_loss after each epoch.
# epochs: number of complete passes through the training dataset.
# batch_size: number of samples used per gradient update.
# callbacks: includes the checkpoint defined above to save the best model.
history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10, batch_size=32, callbacks=[checkpoint])
```

→Output of the Codes:

Load and Preprocess the MNIST Dataset

```
(60000, 28, 28)
Training data shape: (60000, 784)
Test data shape: (10000, 784)
```

Build the Neural Network Model

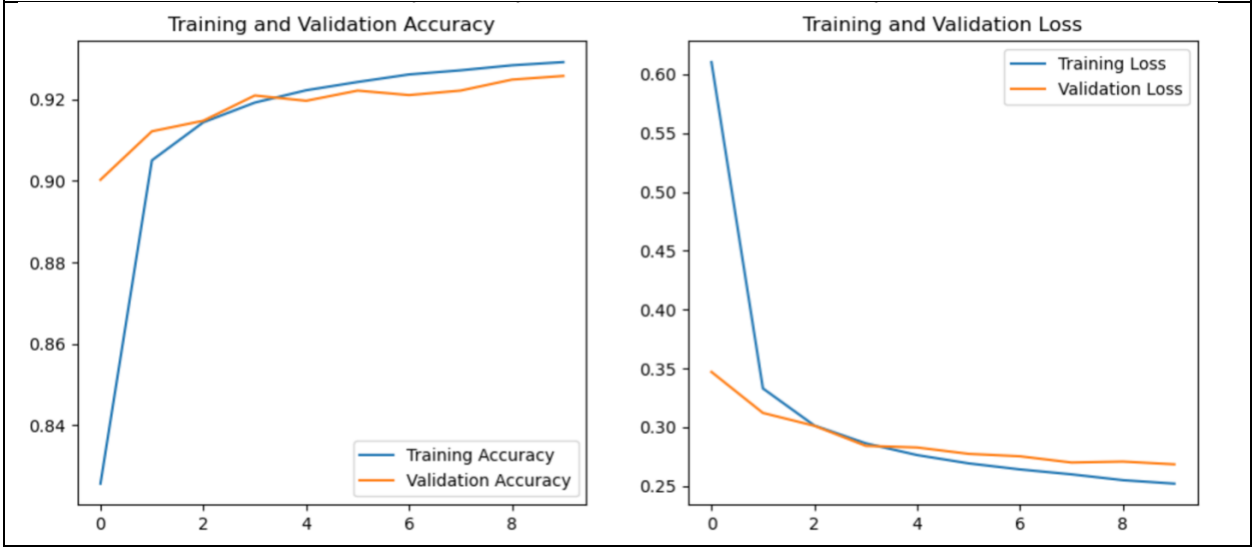
Model: "sequential_3"

Layer (type)	Output Shape	Param #
dense_6 (Dense)	(None, 8)	6,280
dense_7 (Dense)	(None, 10)	90

Total params: 6,370 (24.88 KB)
Trainable params: 6,370 (24.88 KB)
Non-trainable params: 0 (0.00 B)

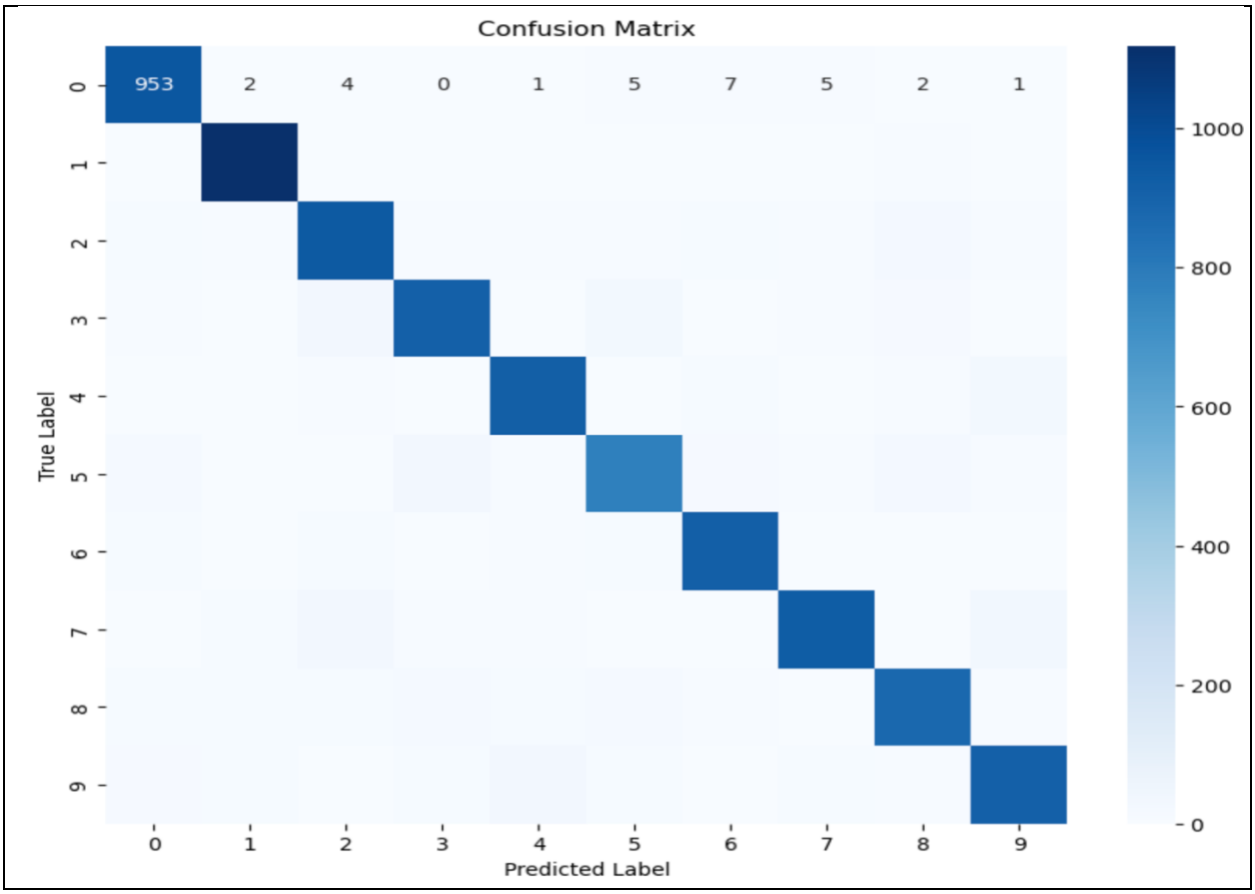
Train the Neural Network

```
Epoch 1/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.7005 - loss: 0.9916 - val_accuracy: 0.9003 - val_loss: 0.3469
Epoch 2/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9012 - loss: 0.3410 - val_accuracy: 0.9122 - val_loss: 0.3120
Epoch 3/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9125 - loss: 0.3051 - val_accuracy: 0.9148 - val_loss: 0.3011
Epoch 4/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9181 - loss: 0.2827 - val_accuracy: 0.9210 - val_loss: 0.2840
Epoch 5/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 1ms/step - accuracy: 0.9211 - loss: 0.2763 - val_accuracy: 0.9197 - val_loss: 0.2826
Epoch 6/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9221 - loss: 0.2738 - val_accuracy: 0.9222 - val_loss: 0.2772
Epoch 7/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9267 - loss: 0.2604 - val_accuracy: 0.9211 - val_loss: 0.2751
Epoch 8/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9286 - loss: 0.2541 - val_accuracy: 0.9222 - val_loss: 0.2698
Epoch 9/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9280 - loss: 0.2532 - val_accuracy: 0.9249 - val_loss: 0.2706
Epoch 10/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9300 - loss: 0.2479 - val_accuracy: 0.9258 - val_loss: 0.2683
```

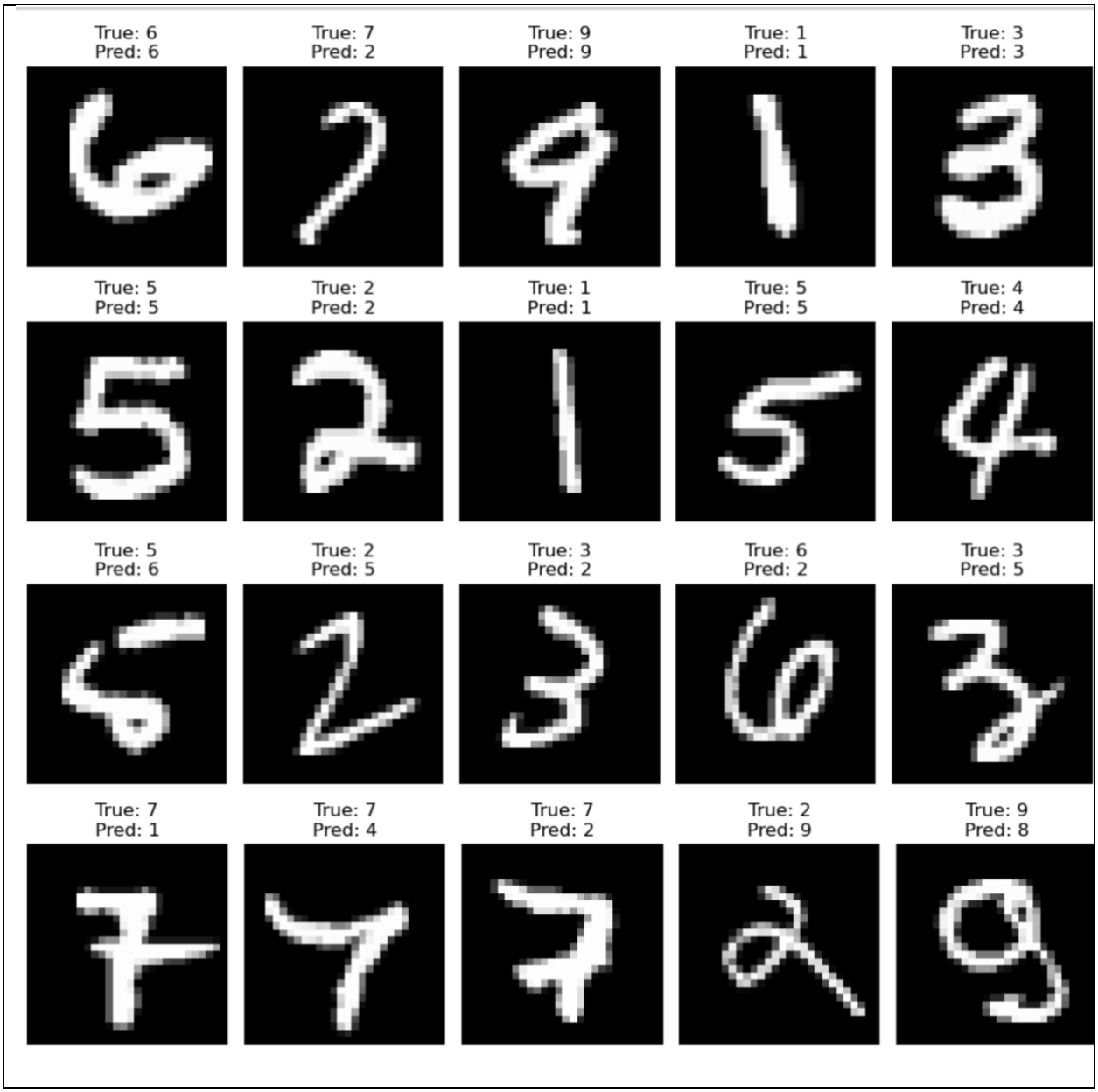


Evaluate the Model

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9129 - loss: 0.3095
Test Accuracy: 0.9258
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
              precision    recall  f1-score   support

           0       0.93      0.97      0.95       980
           1       0.96      0.99      0.97      1135
           2       0.90      0.91      0.91      1032
           3       0.92      0.90      0.91      1010
           4       0.92      0.94      0.93       982
           5       0.90      0.87      0.89       892
           6       0.94      0.96      0.95       958
           7       0.95      0.90      0.93      1028
           8       0.91      0.90      0.90       974
           9       0.92      0.90      0.91      1009

    accuracy                           0.93     10000
   macro avg       0.93      0.92      0.92     10000
weighted avg       0.93      0.93      0.93     10000
```



Visualize Predictions



Part 3:

→The code used:

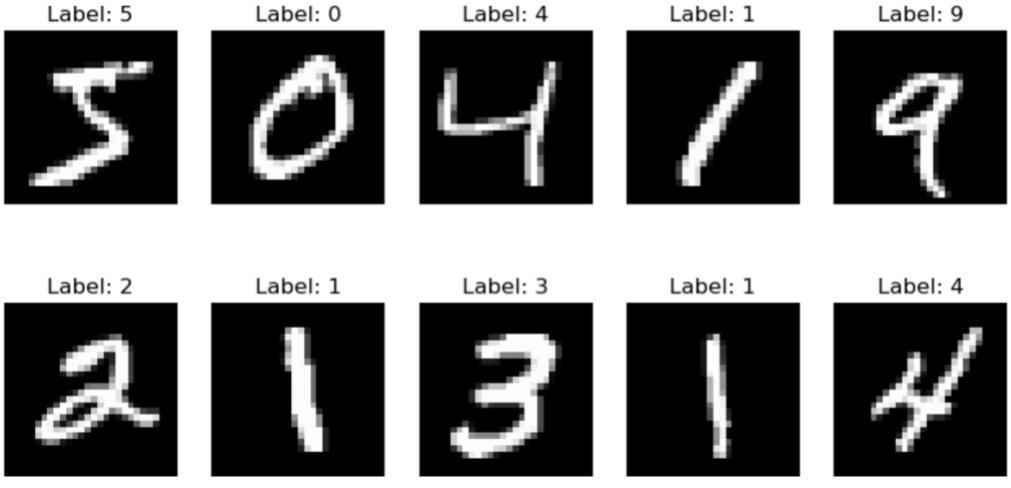
```
model = Sequential([
    # Input layer for flattened 28x28 pixel images (784 total inputs)
    Input(shape=(784,)),

    # Hidden layer with 8 neurons.
    # ReLU activation introduces non-linearity and helps the model learn complex patterns in the data.
    Dense(8, activation='relu'),

    # Output layer with 10 neurons (one for each digit class from 0 to 9).
    # Softmax activation converts the output to probability values summing to 1.
    Dense(10, activation='softmax')
])
```

```
# Train the model using training data and evaluate on test data after each epoch.
# validation_data: used to calculate val_accuracy and val_loss after each epoch.
# epochs: number of complete passes through the training dataset.
# batch_size: number of samples used per gradient update.
# callbacks: includes the checkpoint defined above to save the best model.
history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10, batch_size=32, callbacks=[checkpoint])
```

→Output of the Codes:

Load and Preprocess the MNIST Dataset

```
(60000, 28, 28)
Training data shape: (60000, 784)
Test data shape: (10000, 784)
```



Build the Neural Network Model

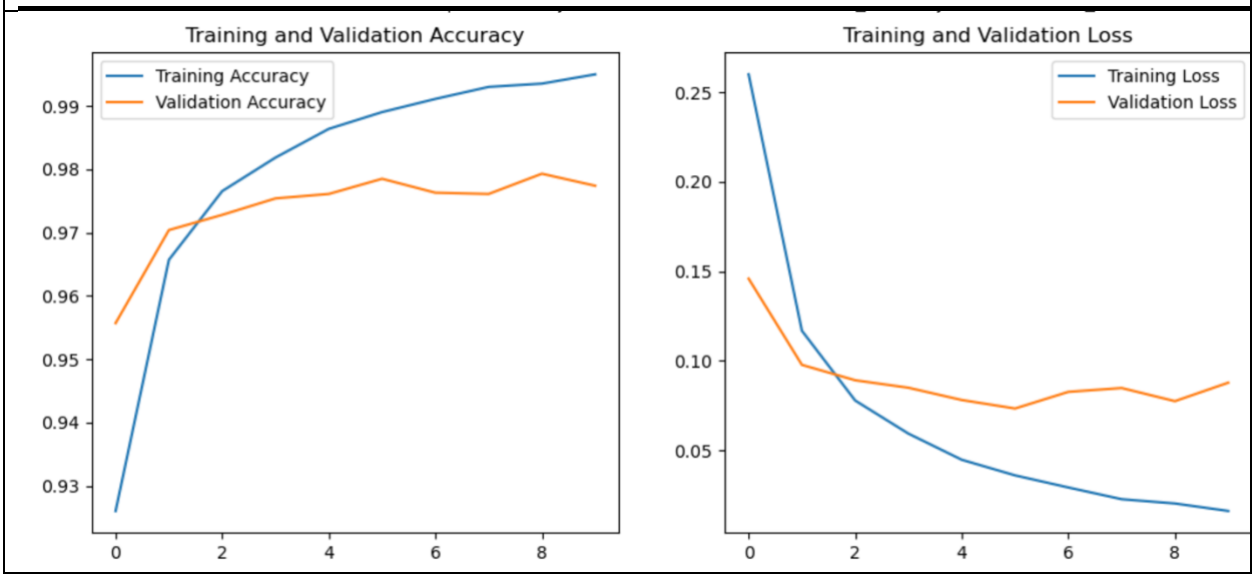
Model: "sequential_1"

Layer (type)	Output Shape	Param #
dense_2 (Dense)	(None, 128)	100,480
dense_3 (Dense)	(None, 10)	1,290

Total params: 101,770 (397.54 KB)
Trainable params: 101,770 (397.54 KB)
Non-trainable params: 0 (0.00 B)

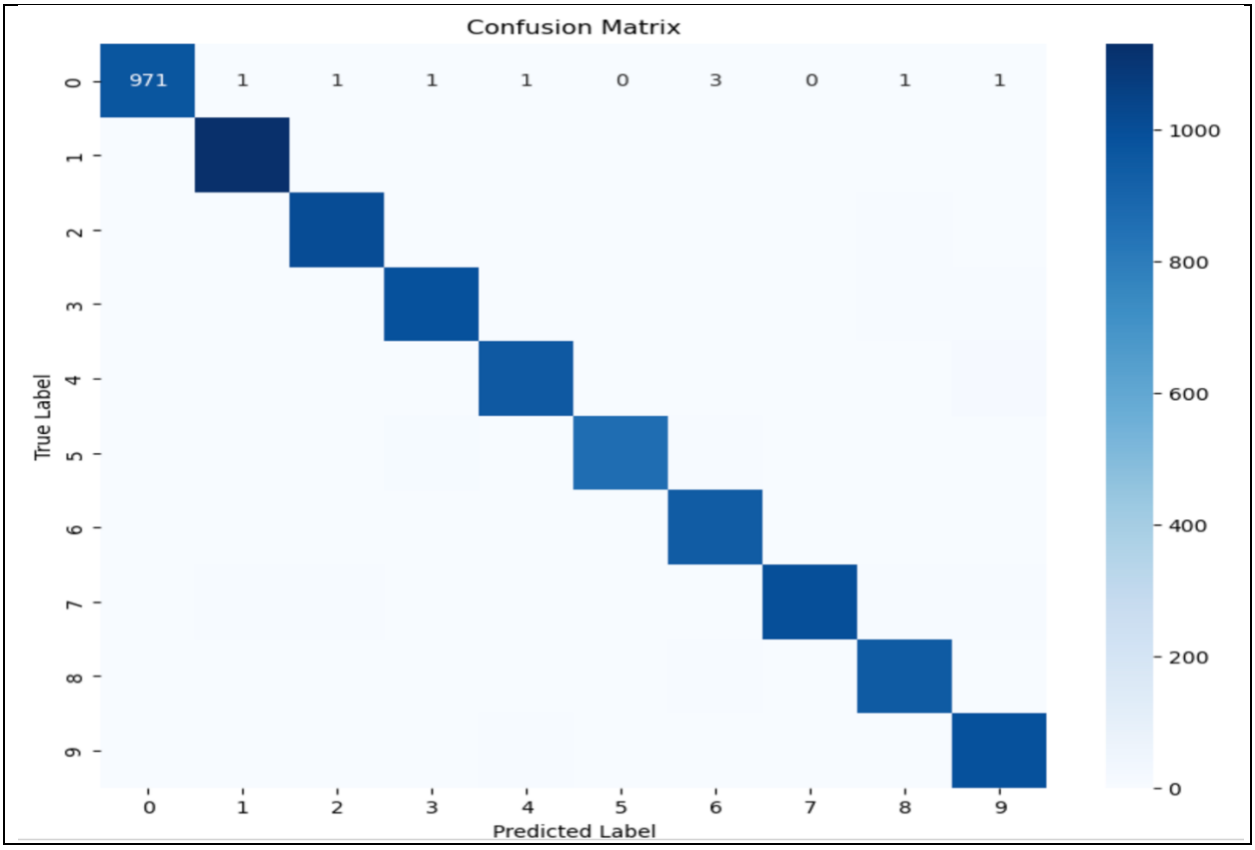
Train the Neural Network

```
Epoch 1/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.8774 - loss: 0.4299 - val_accuracy: 0.9557 - val_loss: 0.1458
Epoch 2/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9639 - loss: 0.1241 - val_accuracy: 0.9704 - val_loss: 0.0976
Epoch 3/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9775 - loss: 0.0763 - val_accuracy: 0.9728 - val_loss: 0.0890
Epoch 4/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.9828 - loss: 0.0589 - val_accuracy: 0.9754 - val_loss: 0.0849
Epoch 5/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9876 - loss: 0.0419 - val_accuracy: 0.9761 - val_loss: 0.0780
Epoch 6/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9898 - loss: 0.0328 - val_accuracy: 0.9785 - val_loss: 0.0732
Epoch 7/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9916 - loss: 0.0274 - val_accuracy: 0.9763 - val_loss: 0.0826
Epoch 8/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9940 - loss: 0.0206 - val_accuracy: 0.9761 - val_loss: 0.0847
Epoch 9/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9947 - loss: 0.0173 - val_accuracy: 0.9793 - val_loss: 0.0774
Epoch 10/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9955 - loss: 0.0148 - val_accuracy: 0.9774 - val_loss: 0.0877
```

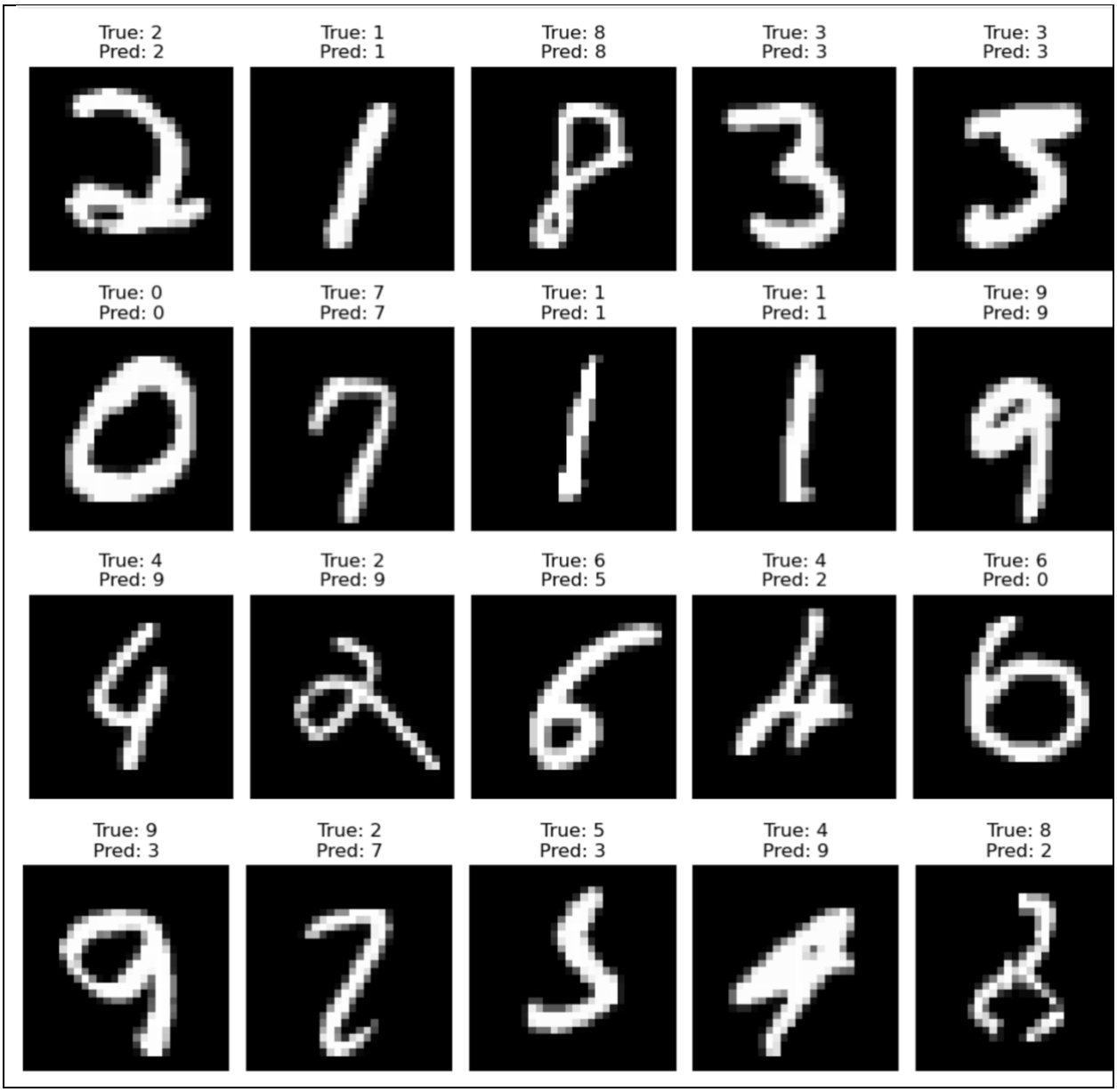


Evaluate the Model

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.9763 - loss: 0.0865
Test Accuracy: 0.9793
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       980
           1       0.99      1.00      0.99      1135
           2       0.98      0.98      0.98      1032
           3       0.97      0.98      0.98      1010
           4       0.98      0.97      0.98       982
           5       0.99      0.97      0.98       892
           6       0.98      0.98      0.98       958
           7       0.99      0.97      0.98      1028
           8       0.97      0.97      0.97       974
           9       0.96      0.98      0.97      1009

    accuracy                           0.98     10000
   macro avg       0.98      0.98      0.98     10000
weighted avg       0.98      0.98      0.98     10000
```



Visualize Predictions



Part 4:

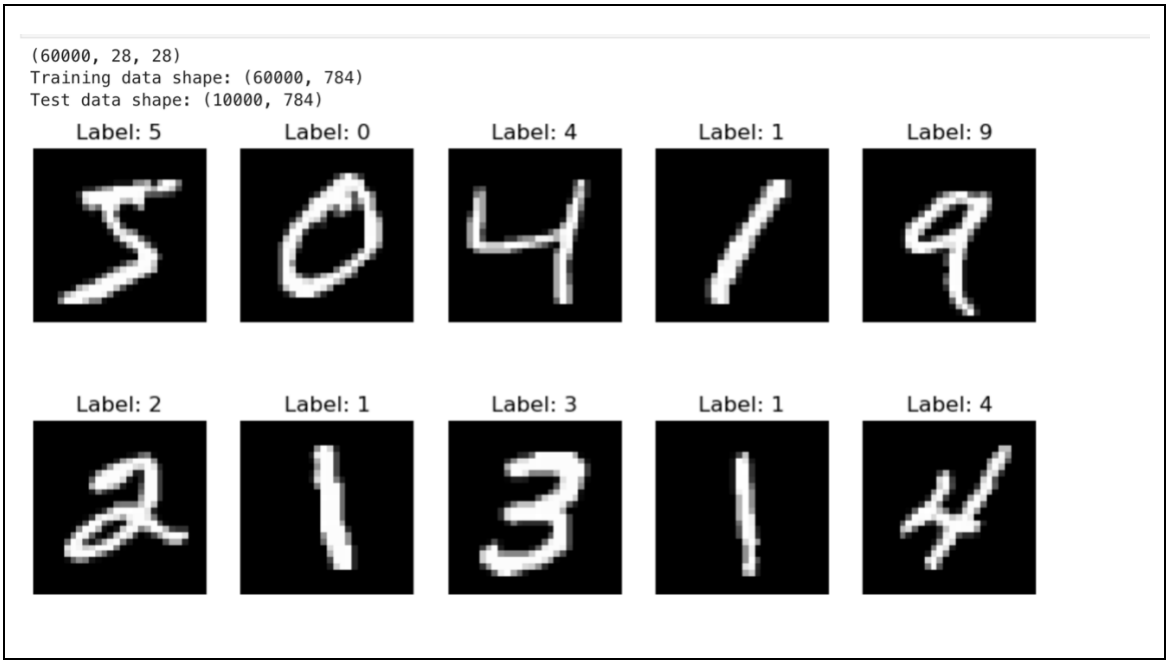
→The code used:

```
model = Sequential([
    # Input layer for flattened 28x28 pixel images (784 total inputs)
    Input(shape=(784,)),

    # A wide 512-neuron layer captures rich, high-level patterns in the data. ReLU adds non-linearity.
    Dense(512, activation='relu'),
    # BatchNorm keeps activations well-scaled, which speeds up convergence and adds a bit of regularisation.
    BatchNormalization(),
    # Dropout randomly deactivates 20 % of the neurons during training to restrict over-fitting.
    Dropout(0.20),

    # Halve the width to 256 neurons, pushing the model to learn a more compressed representation.
    Dense(256, activation='relu'),
    BatchNormalization(),
    Dropout(0.20),

    # Another reduction (128 neurons) for deeper feature abstraction.
    Dense(128, activation='relu'),
    BatchNormalization(),
    Dropout(0.20),

    # Final dense layer (64 neurons) refines features just before classification.
    Dense(64, activation='relu'),
    BatchNormalization(),
    Dropout(0.20),

    # Output layer with 10 neurons (one for each digit class from 0 to 9).
    # Softmax activation converts the output to probability values summing to 1.
    Dense(10, activation='softmax')
])
```

```
# Train the model using training data and evaluate on test data after each epoch.
# validation_data: used to calculate val_accuracy and val_loss after each epoch.
# epochs: number of complete passes through the training dataset.
# batch_size: number of samples used per gradient update.
# callbacks: includes the checkpoint defined above to save the best model.
history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=40, batch_size=128, callbacks=[checkpoint])
```

→Output of the Codes:

Load and Preprocess the MNIST Dataset



Build the Neural Network Model

Model: "sequential_1"

Layer (type)	Output Shape	Param #
dense_5 (Dense)	(None, 512)	401,920
batch_normalization_4 (BatchNormalization)	(None, 512)	2,048
dropout_4 (Dropout)	(None, 512)	0
dense_6 (Dense)	(None, 256)	131,328
batch_normalization_5 (BatchNormalization)	(None, 256)	1,024
dropout_5 (Dropout)	(None, 256)	0
dense_7 (Dense)	(None, 128)	32,896
batch_normalization_6 (BatchNormalization)	(None, 128)	512
dropout_6 (Dropout)	(None, 128)	0
dense_8 (Dense)	(None, 64)	8,256
batch_normalization_7 (BatchNormalization)	(None, 64)	256
dropout_7 (Dropout)	(None, 64)	0
dense_9 (Dense)	(None, 10)	650

Total params: 578,890 (2.21 MB)

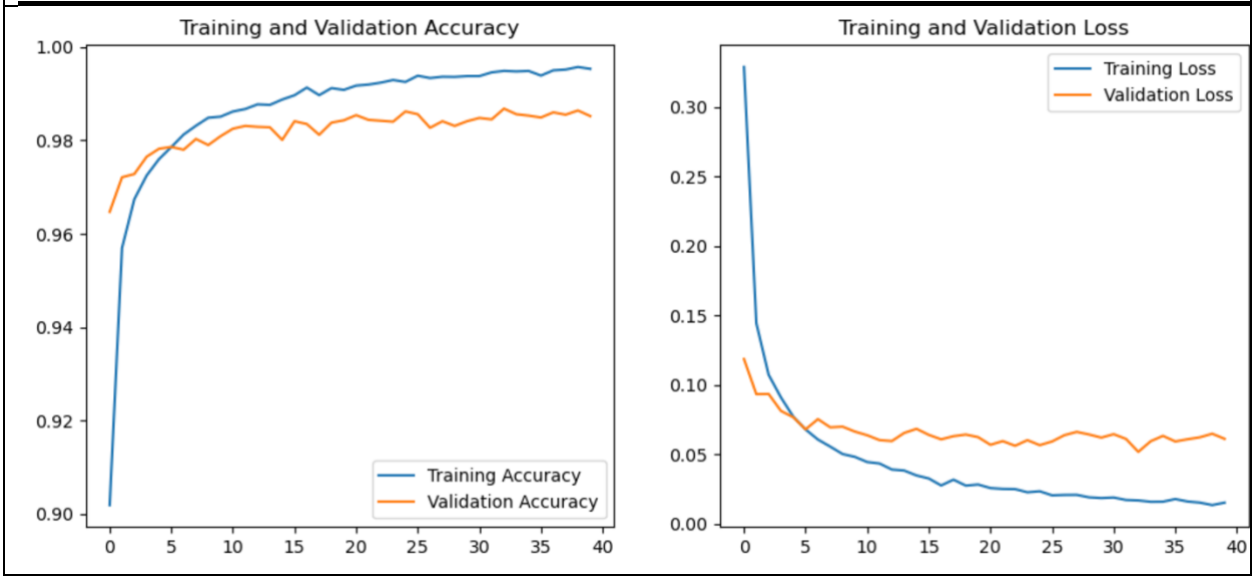
Trainable params: 576,970 (2.20 MB)

Non-trainable params: 1,920 (7.50 KB)

Train the Neural Network

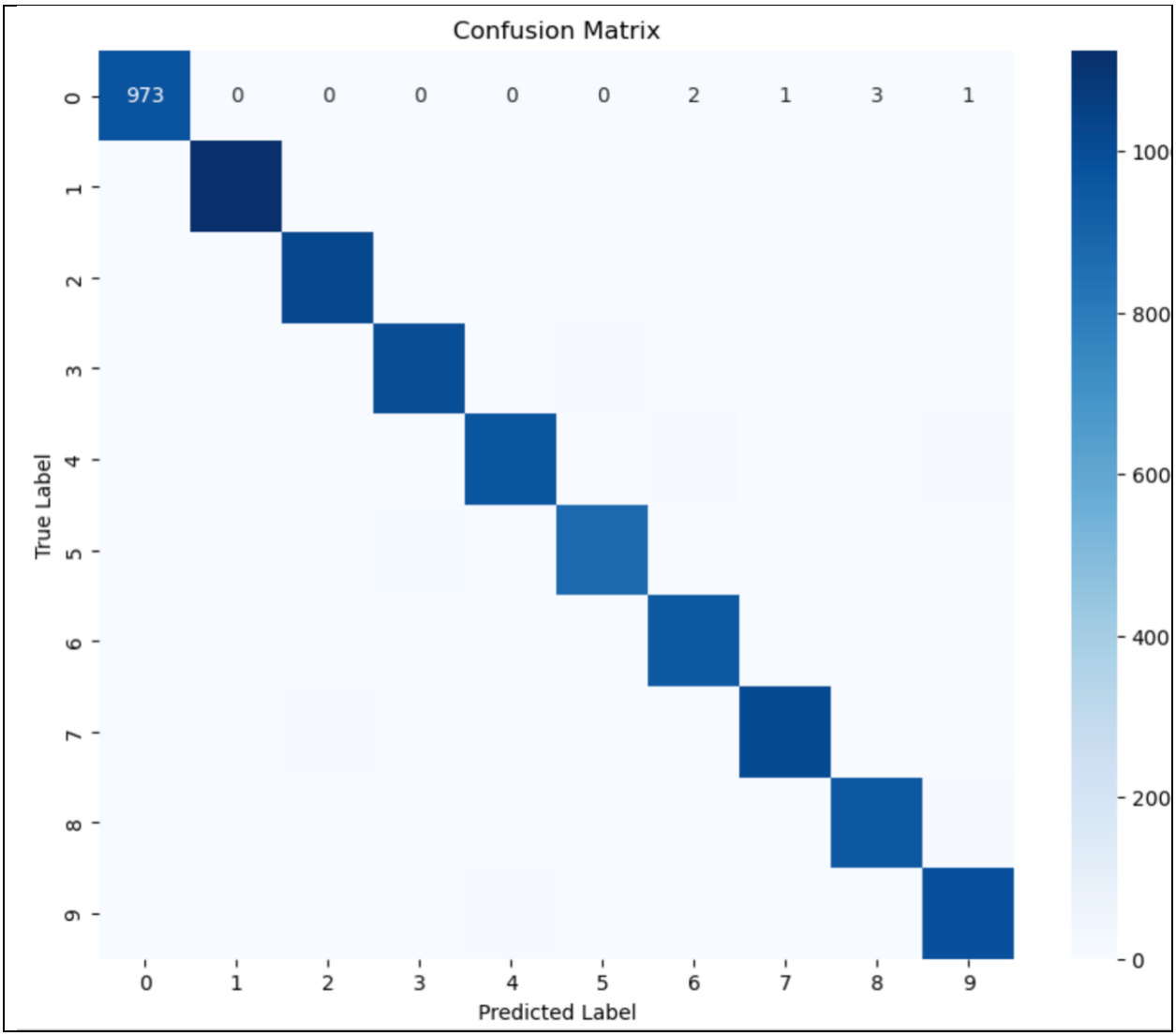
```
Epoch 1/40
2025-04-18 20:55:27.227181: W external/local_xla/xla/tsl/framework/cpu_allocator_impl.cc:83] Allocation of 188160000 ex
ds 10% of free system memory.
469/469 ━━━━━━━━━━━━━━━━━━━━ 7s 9ms/step - accuracy: 0.8214 - loss: 0.5887 - val_accuracy: 0.9647 - val_loss: 0.1186
Epoch 2/40
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 9ms/step - accuracy: 0.9553 - loss: 0.1502 - val_accuracy: 0.9721 - val_loss: 0.0934
Epoch 3/40
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 9ms/step - accuracy: 0.9676 - loss: 0.1068 - val_accuracy: 0.9728 - val_loss: 0.0935
Epoch 4/40
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 9ms/step - accuracy: 0.9736 - loss: 0.0890 - val_accuracy: 0.9765 - val_loss: 0.0813
Epoch 5/40
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 9ms/step - accuracy: 0.9766 - loss: 0.0751 - val_accuracy: 0.9782 - val_loss: 0.0767
Epoch 6/40
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 9ms/step - accuracy: 0.9783 - loss: 0.0680 - val_accuracy: 0.9786 - val_loss: 0.0680
Epoch 7/40
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 9ms/step - accuracy: 0.9813 - loss: 0.0591 - val_accuracy: 0.9780 - val_loss: 0.0754
Epoch 8/40
469/469 ━━━━━━━━━━━━━━━━━━━━ 8s 16ms/step - accuracy: 0.9846 - loss: 0.0503 - val_accuracy: 0.9803 - val_loss: 0.0695
Epoch 9/40
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 9ms/step - accuracy: 0.9851 - loss: 0.0491 - val_accuracy: 0.9790 - val_loss: 0.0700
Epoch 10/40
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 9ms/step - accuracy: 0.9869 - loss: 0.0422 - val_accuracy: 0.9809 - val_loss: 0.0664
Epoch 11/40
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 9ms/step - accuracy: 0.9864 - loss: 0.0440 - val_accuracy: 0.9825 - val_loss: 0.0638
Epoch 12/40
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 9ms/step - accuracy: 0.9877 - loss: 0.0392 - val_accuracy: 0.9831 - val_loss: 0.0603
Epoch 13/40
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 8ms/step - accuracy: 0.9881 - loss: 0.0372 - val_accuracy: 0.9829 - val_loss: 0.0596
Epoch 14/40
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 9ms/step - accuracy: 0.9876 - loss: 0.0388 - val_accuracy: 0.9828 - val_loss: 0.0654
Epoch 15/40
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 9ms/step - accuracy: 0.9892 - loss: 0.0333 - val_accuracy: 0.9801 - val_loss: 0.0684
Epoch 16/40
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 9ms/step - accuracy: 0.9908 - loss: 0.0301 - val_accuracy: 0.9841 - val_loss: 0.0640
Epoch 17/40
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 9ms/step - accuracy: 0.9914 - loss: 0.0267 - val_accuracy: 0.9835 - val_loss: 0.0608
Epoch 18/40
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 8ms/step - accuracy: 0.9903 - loss: 0.0294 - val_accuracy: 0.9812 - val_loss: 0.0631
Epoch 19/40
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 8ms/step - accuracy: 0.9913 - loss: 0.0271 - val_accuracy: 0.9838 - val_loss: 0.0643
Epoch 20/40
```

```
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 9ms/step - accuracy: 0.9915 - loss: 0.0268 - val_accuracy: 0.9843 - val_loss: 0.0624
Epoch 21/40
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 9ms/step - accuracy: 0.9921 - loss: 0.0255 - val_accuracy: 0.9854 - val_loss: 0.0569
Epoch 22/40
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 10ms/step - accuracy: 0.9920 - loss: 0.0251 - val_accuracy: 0.9844 - val_loss: 0.059
Epoch 23/40
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 9ms/step - accuracy: 0.9922 - loss: 0.0252 - val_accuracy: 0.9842 - val_loss: 0.0562
Epoch 24/40
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 8ms/step - accuracy: 0.9932 - loss: 0.0223 - val_accuracy: 0.9840 - val_loss: 0.0602
Epoch 25/40
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 9ms/step - accuracy: 0.9923 - loss: 0.0235 - val_accuracy: 0.9862 - val_loss: 0.0566
Epoch 26/40
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 9ms/step - accuracy: 0.9944 - loss: 0.0192 - val_accuracy: 0.9856 - val_loss: 0.0593
Epoch 27/40
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 9ms/step - accuracy: 0.9929 - loss: 0.0217 - val_accuracy: 0.9827 - val_loss: 0.0638
Epoch 28/40
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 10ms/step - accuracy: 0.9935 - loss: 0.0222 - val_accuracy: 0.9841 - val_loss: 0.066
Epoch 29/40
469/469 ━━━━━━━━━━━━━━━━━━━━ 6s 12ms/step - accuracy: 0.9940 - loss: 0.0172 - val_accuracy: 0.9831 - val_loss: 0.064
Epoch 30/40
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 11ms/step - accuracy: 0.9938 - loss: 0.0182 - val_accuracy: 0.9841 - val_loss: 0.062
Epoch 31/40
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 9ms/step - accuracy: 0.9943 - loss: 0.0168 - val_accuracy: 0.9848 - val_loss: 0.0645
Epoch 32/40
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 9ms/step - accuracy: 0.9946 - loss: 0.0161 - val_accuracy: 0.9845 - val_loss: 0.0611
Epoch 33/40
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 9ms/step - accuracy: 0.9950 - loss: 0.0164 - val_accuracy: 0.9868 - val_loss: 0.0518
Epoch 34/40
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 8ms/step - accuracy: 0.9953 - loss: 0.0140 - val_accuracy: 0.9856 - val_loss: 0.0594
Epoch 35/40
469/469 ━━━━━━━━━━━━━━━━━━━━ 7s 15ms/step - accuracy: 0.9953 - loss: 0.0145 - val_accuracy: 0.9853 - val_loss: 0.063
Epoch 36/40
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 10ms/step - accuracy: 0.9947 - loss: 0.0151 - val_accuracy: 0.9849 - val_loss: 0.059
Epoch 37/40
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 8ms/step - accuracy: 0.9945 - loss: 0.0168 - val_accuracy: 0.9860 - val_loss: 0.0609
Epoch 38/40
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 9ms/step - accuracy: 0.9954 - loss: 0.0151 - val_accuracy: 0.9855 - val_loss: 0.0623
Epoch 39/40
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 9ms/step - accuracy: 0.9961 - loss: 0.0127 - val_accuracy: 0.9864 - val_loss: 0.0649
Epoch 40/40
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 9ms/step - accuracy: 0.9954 - loss: 0.0148 - val_accuracy: 0.9852 - val_loss: 0.0612
```

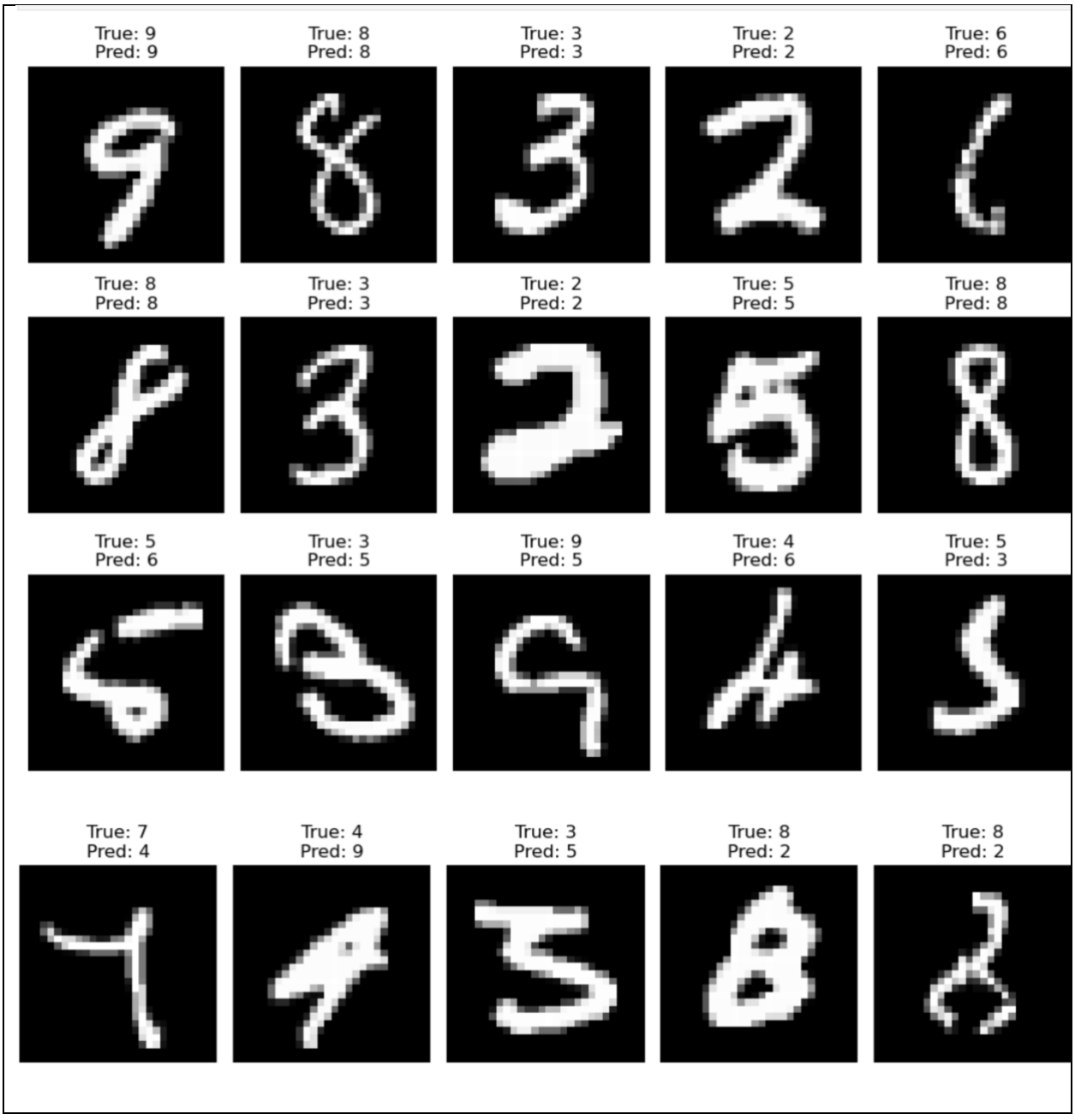


Evaluate the Model

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9821 - loss: 0.0661
Test Accuracy: 0.9868
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       980
           1       0.99      0.99      0.99      1135
           2       0.99      0.99      0.99      1032
           3       0.98      0.99      0.98      1010
           4       0.99      0.99      0.99       982
           5       0.99      0.98      0.98       892
           6       0.98      0.99      0.99       958
           7       0.99      0.98      0.99      1028
           8       0.98      0.98      0.98       974
           9       0.99      0.98      0.98      1009

    accuracy                           0.99     10000
   macro avg       0.99      0.99      0.99     10000
weighted avg       0.99      0.99      0.99     10000
```


Confusion Matrix
973 0 0 0 0 0 2 1 3 1
True Label
Predicted Label
0 1 2 3 4 5 6 7 8 9
100
800
600
400
200
0

Visualize Predictions



Part 5 – Discussion Questions:

1. **Epochs**

- **What is an epoch?**
  An epoch is one complete pass through the entire training dataset by the neural network.

- **Why are epochs used?**
  Epochs allow the model to gradually adjust its weights and learn from the data through repeated training cycles.
- **What happens when the number of epochs changes?**
  Increasing epochs can improve learning but may cause overfitting. Decreasing epochs can result in underfitting if the model doesn't learn enough.

2. **Batch Size**

- **What is batch size and its role?**
  Batch size is the number of samples the model processes before updating the weights. It determines how many examples are passed through the network at once during training.
- **Effects of changing batch size:**
  A larger batch size trains faster but may generalize poorly. A smaller batch size can improve performance but increases training time and noise in updates.

3. **Dropout**

- **What is Dropout?**
  Dropout is a regularization technique where some neurons are randomly turned off during training to prevent over-reliance on specific neurons.
- **How does it help?**
  It reduces overfitting and improves generalization by forcing the model to learn more distributed and robust representations. Typical dropout rates are 0.2–0.5; rates above 0.5 often under-fit the data.