```
from sklearn.preprocessing import LabelBinarizer
from sklearn.metrics import classification_report
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.datasets import mnist


import matplotlib.pyplot as plt
import numpy as np



print("[INFO] accessing MNIST...")
```

```
[INFO] accessing MNIST...
```

```
((trainX, trainY), (testX, testY)) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
```

```
trainX = trainX.reshape((trainX.shape[0], 28 * 28 * 1))
testX = testX.reshape((testX.shape[0], 28 * 28 * 1))


trainX = trainX.astype("float32") / 255.0
testX = testX.astype("float32") / 255.0


lb = LabelBinarizer()
trainY = lb.fit_transform(trainY)
testY = lb.transform(testY)


model = Sequential()
model.add(Dense(256, input_shape=(784,), activation="relu"))
model.add(Dense(128, activation="relu"))
model.add(Dense(64, activation="relu"))
model.add(Dense(10, activation="softmax"))


print("[INFO] training network...")
Adm = Adam(0.01)
model.compile(loss="categorical_crossentropy", optimizer=Adm,
metrics=["accuracy"])
H = model.fit(trainX, trainY, validation_data=(testX, testY),
epochs=100, batch_size=128)
```

```
Epoch 90/100
469/469 [==============================] - 4s 8ms/step - loss: 0.0367 - accuracy: 0.9944 - val_loss: 0.2526 - val_accuracy: 0.975
Epoch 91/100
469/469 [==============================] - 4s 8ms/step - loss: 0.0179 - accuracy: 0.9960 - val_loss: 0.2878 - val_accuracy: 0.977
Epoch 92/100
469/469 [==============================] - 5s 10ms/step - loss: 0.0367 - accuracy: 0.9942 - val_loss: 0.3848 - val_accuracy: 0.97
Epoch 93/100
469/469 [==============================] - 4s 8ms/step - loss: 0.0277 - accuracy: 0.9949 - val_loss: 0.3882 - val_accuracy: 0.975
Epoch 94/100
469/469 [==============================] - 4s 8ms/step - loss: 0.0274 - accuracy: 0.9964 - val_loss: 0.3445 - val_accuracy: 0.977
Epoch 95/100
469/469 [==============================] - 4s 9ms/step - loss: 0.0179 - accuracy: 0.9970 - val_loss: 0.4133 - val_accuracy: 0.976
Epoch 96/100
469/469 [==============================] - 4s 8ms/step - loss: 0.0353 - accuracy: 0.9936 - val_loss: 0.3252 - val_accuracy: 0.976
Epoch 97/100
469/469 [==============================] - 4s 8ms/step - loss: 0.0183 - accuracy: 0.9966 - val_loss: 0.3795 - val_accuracy: 0.974
Epoch 98/100
469/469 [==============================] - 4s 10ms/step - loss: 0.0194 - accuracy: 0.9964 - val_loss: 0.3829 - val_accuracy: 0.97
Epoch 99/100
469/469 [==============================] - 4s 8ms/step - loss: 0.0172 - accuracy: 0.9965 - val_loss: 0.4113 - val_accuracy: 0.972
Epoch 100/100
```

```
print("[INFO] evaluating network...")
predictions = model.predict(testX, batch_size=128)
print(classification_report(testY.argmax(axis=1),
  predictions.argmax(axis=1),
  target_names=[str(x) for x in lb.classes_]))
```

```
[INFO] evaluating network...
79/79 [==============================] - 0s 4ms/step
              precision    recall  f1-score   support

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

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

```
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, 100), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, 100), H.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, 100), H.history["accuracy"], label="train_acc")
plt.plot(np.arange(0, 100), H.history["val_accuracy"],
label="val_acc")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7dd41427e350>
```