Practical 1

Aim: Implement Feed-forward Neural Network and train the network with different optimizers and compare the results.

Training with SGD optimizer:

```
# import the necessary packages
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 SGD, Adam, RMSprop
from tensorflow.keras.datasets import mnist
import matplotlib.pyplot as plt
import numpy as np

# grab the MNIST dataset
print("[INFO] accessing MNIST...")
((trainX, trainY), (testX, testY)) = mnist.load_data()

# flatten the images to 28*28=784 pixels
trainX = trainX.reshape((trainX.shape[0], 28 * 28 * 1))
testX = testX.reshape((testX.shape[0], 28 * 28 * 1))

# scale data to the range of [0, 1]
trainX = trainX.astype("float32") / 255.0
testX = testX.astype("float32") / 255.0

# convert the labels from integers to vectors
lb = LabelBinarizer()
trainY = lb.fit_transform(trainY)
testY = lb.transform(testY)

# define the model architecture
def build_model():
    model = Sequential()
    model.add(Dense(256, input_shape=(784,), activation="sigmoid"))
    model.add(Dense(128, activation="sigmoid"))
    model.add(Dense(10, activation="softmax"))
    return model

# list of optimizers to train the model with
optimizers = {
```

```python
    'SGD': SGD(0.01),
    'Adam': Adam(0.001),
    'RMSprop': RMSprop(0.001)
}
epochs = 50
results = {}
for opt_name, opt in optimizers.items():
    print(f"[INFO] training network with {opt_name} optimizer...")
    model = build_model()
    model.compile(loss="categorical_crossentropy", optimizer=opt,
metrics=["accuracy"])
    H = model.fit(trainX, trainY, validation_data=(testX, testY),
epochs=epochs, batch_size=128, verbose=0)

    # evaluate the network
    print(f"[INFO] evaluating network with {opt_name} optimizer...")
    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_]))

    # store the results
    results[opt_name] = H

# plot the results
plt.style.use("ggplot")

for opt_name, H in results.items():
    plt.figure(figsize=(12, 6))

    # plot loss
    plt.subplot(1, 2, 1)
    plt.plot(np.arange(0, epochs), H.history["loss"],
label="train_loss")
    plt.plot(np.arange(0, epochs), H.history["val_loss"],
label="val_loss")
    plt.title(f"{opt_name} - Loss")
    plt.xlabel("Epoch #")
    plt.ylabel("Loss")
    plt.legend()

    # plot accuracy
    plt.subplot(1, 2, 2)
```

```
    plt.plot(np.arange(0, epochs), H.history["accuracy"],
label="train_acc")
    plt.plot(np.arange(0, epochs), H.history["val_accuracy"],
label="val_acc")
    plt.title(f"{opt_name} - Accuracy")
    plt.xlabel("Epoch #")
    plt.ylabel("Accuracy")
    plt.legend()
    plt.suptitle(f"Training Loss and Accuracy with {opt_name}
Optimizer")
    plt.show()
```

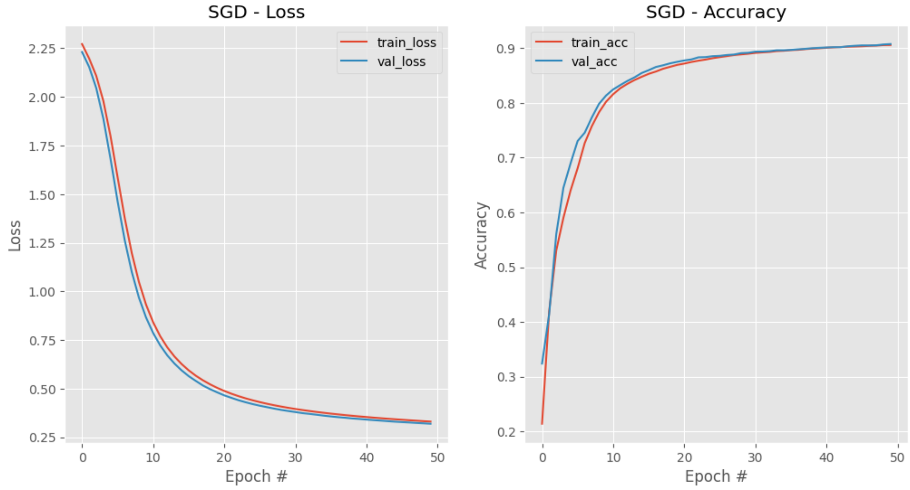
Output:

```
[INFO] training network with Adam optimizer...
[INFO] evaluating network with Adam optimizer...
79/79 [==============================] - 0s 2ms/step
              precision    recall  f1-score   support

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

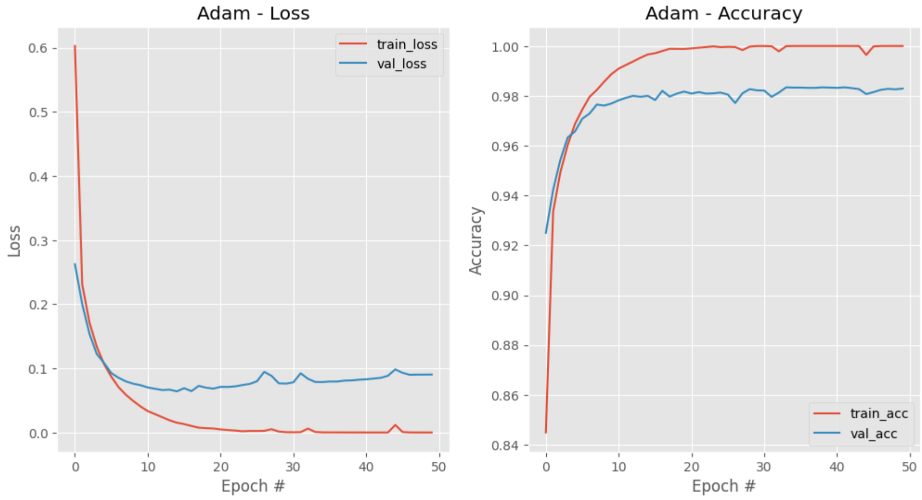
```
[INFO] accessing MNIST...
[INFO] training network with SGD optimizer...
[INFO] evaluating network with SGD optimizer...
79/79 [==============================] - 0s 2ms/step
              precision    recall  f1-score   support

           0       0.92      0.98      0.95       980
           1       0.96      0.97      0.97      1135
           2       0.90      0.88      0.89      1032
           3       0.89      0.89      0.89      1010
           4       0.90      0.93      0.91       982
           5       0.88      0.83      0.85       892
           6       0.92      0.93      0.93       958
           7       0.92      0.92      0.92      1028
           8       0.87      0.85      0.86       974
           9       0.89      0.88      0.89      1009
    accuracy                           0.91     10000
   macro avg       0.91      0.91      0.91     10000
weighted avg       0.91      0.91      0.91     10000
```

```
[INFO] training network with RMSprop optimizer...
[INFO] evaluating network with RMSprop optimizer...
79/79 [==============================] - 0s 2ms/step
              precision    recall  f1-score   support

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

Training Loss and Accuracy with SGD Optimizer



Training Loss and Accuracy with Adam Optimizer



Training Loss and Accuracy with RMSprop Optimizer