```python
# Implementing feedforward neural networks with Keras and TensorFlow
# import the necessary packages
import numpy as np
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import matplotlib.pyplot as plt


# grab the MNIST dataset (if this is your first time using this
# dataset then the 11MB download may take a minute)

print("[INFO] accessing MNIST...")
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train.reshape((x_train.shape[0], -1)).astype('float32') / 255
x_test = x_test.reshape((x_test.shape[0], -1)).astype('float32') / 255
```

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

```python
# One-hot encode the labels
y_train = tf.keras.utils.to_categorical(y_train, num_classes=10)
y_test = tf.keras.utils.to_categorical(y_test, num_classes=10)


# Each data point in the MNIST dataset has an integer label in the range [0,
9], one for each of the possible ten digits in the MNIST dataset.
# A label with a value of 0 indicates that the corresponding image contains a
zero digit. Similarly, a label with a value of 8 indicates
# that the corresponding image contains the number eight.

# However, we first need to transform these integer labels into vector
labels,
#where the index in the vector for label is set to 1 and 0 otherwise (this
process is called one-hot encoding).
```

one-hot encoding representations for each digit, 0–9, in the listing below: 0: [1, 0, 0, 0, 0, 0, 0, 0, 0, 0] 1: [0, 1, 0, 0, 0, 0, 0, 0, 0, 0] 2: [0, 0, 1, 0, 0, 0, 0, 0, 0, 0] 3: [0, 0, 0, 1, 0, 0, 0, 0, 0, 0] 4: [0, 0, 0, 0, 1, 0, 0, 0, 0, 0] 5: [0, 0, 0, 0, 0, 1, 0, 0, 0, 0] 6: [0, 0, 0, 0, 0, 0, 1, 0, 0, 0] 7: [0, 0, 0, 0, 0, 0, 0, 1, 0, 0] 8: [0, 0, 0, 0, 0, 0, 0, 0, 1, 0] 9: [0, 0, 0, 0, 0, 0, 0, 0, 0, 1]

```python
# Step 3: Define the network architecture
model = Sequential([
    Dense(64, activation='relu', input_shape=(784,)),
    Dense(64, activation='relu'),
    Dense(64, activation='relu'),
    Dense(10, activation='softmax')
])
```

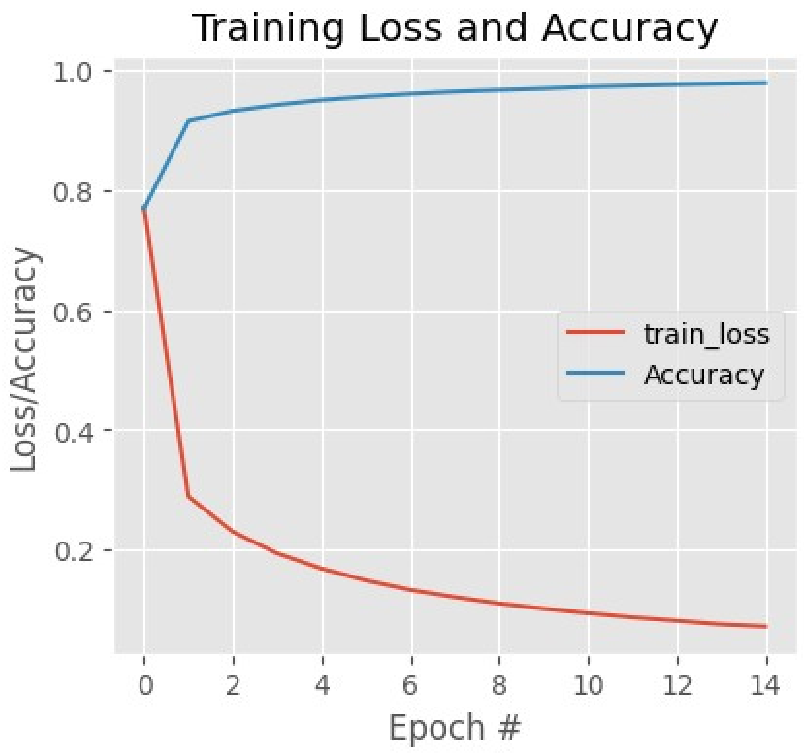
```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer.
When using Sequential models, prefer using an `Input(shape)` object as the
first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
# Step 4: Compile the model
model.compile(optimizer='sgd', loss='categorical_crossentropy',
metrics=['accuracy'])

# Step 5: Train the model
H = model.fit(x_train, y_train, epochs=15, batch_size=32,
validation_data=(x_test, y_test))

Epoch 1/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 11s 5ms/step - accuracy: 0.5761 - loss: 1.3276
- val_accuracy: 0.9113 - val_loss: 0.3130
Epoch 2/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9084 - loss: 0.3089
- val_accuracy: 0.9315 - val_loss: 0.2393
Epoch 3/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9306 - loss: 0.2326
- val_accuracy: 0.9420 - val_loss: 0.2036
Epoch 4/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 3ms/step - accuracy: 0.9425 - loss: 0.1965
- val_accuracy: 0.9446 - val_loss: 0.1831
Epoch 5/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 3ms/step - accuracy: 0.9496 - loss: 0.1711
- val_accuracy: 0.9520 - val_loss: 0.1617
Epoch 6/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.9560 - loss: 0.1479
- val_accuracy: 0.9574 - val_loss: 0.1467
Epoch 7/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 4ms/step - accuracy: 0.9607 - loss: 0.1363
- val_accuracy: 0.9617 - val_loss: 0.1352
Epoch 8/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.9638 - loss: 0.1231
- val_accuracy: 0.9612 - val_loss: 0.1261
Epoch 9/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9675 - loss: 0.1101
- val_accuracy: 0.9618 - val_loss: 0.1214
Epoch 10/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9681 - loss: 0.1066
- val_accuracy: 0.9651 - val_loss: 0.1174
Epoch 11/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - accuracy: 0.9730 - loss: 0.0913
- val_accuracy: 0.9671 - val_loss: 0.1104
Epoch 12/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 3ms/step - accuracy: 0.9741 - loss: 0.0870
- val_accuracy: 0.9658 - val_loss: 0.1125
Epoch 13/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - accuracy: 0.9762 - loss: 0.0814
- val_accuracy: 0.9694 - val_loss: 0.1022
Epoch 14/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9780 - loss: 0.0756
- val_accuracy: 0.9674 - val_loss: 0.1073
Epoch 15/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 11s 6ms/step - accuracy: 0.9786 - loss: 0.0726
- val_accuracy: 0.9656 - val_loss: 0.1155

# Step 6: Evaluate the network
test_loss, test_accuracy = model.evaluate(x_test, y_test)
print(f'Test accuracy: {test_accuracy*100:.2f}%')
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9608 - loss: 0.1332
Test accuracy: 96.56%
```

```
# Step 7: Plot the training loss and accuracy
plt.style.use("ggplot")
plt.figure(figsize=(10, 4))
plt.subplot(1, 2, 1)
plt.plot(H.history['loss'], label="train_loss")
plt.plot(H.history['accuracy'], label="Accuracy")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7cc7584e28c0>
```


Training Loss and Accuracy
Loss/Accuracy
Epoch #
train_loss
Accuracy