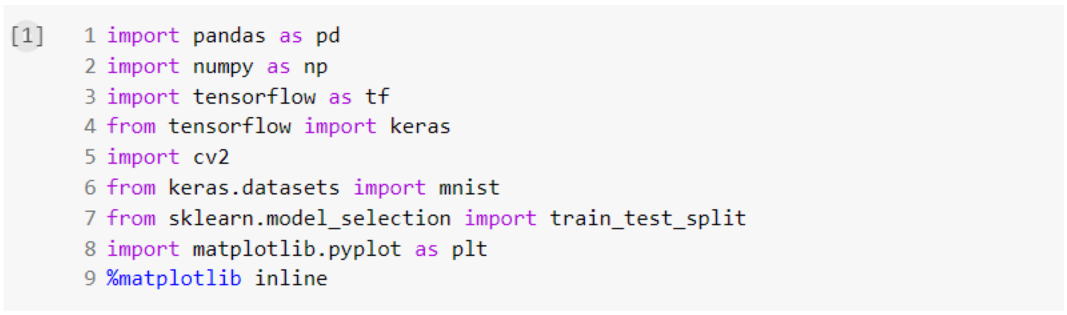
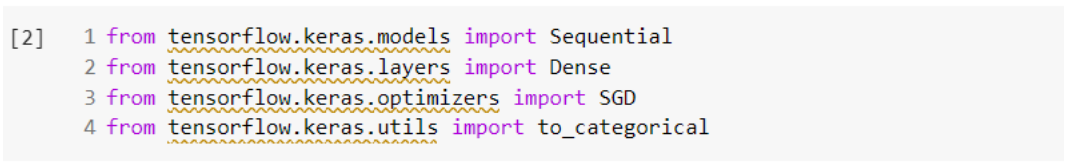
Application of Keras to build, compile, and train a neural network as a multi-class classifier for MNIST dataset (0 vs. 1 vs. 2 vs. 3):

a) Use mnist function in keras.datasets to load MNIST dataset and split it into training and testing sets. Then, randomly select 20% of the training images along with their corresponding labels to be the validation data

- Importing the required datasets:

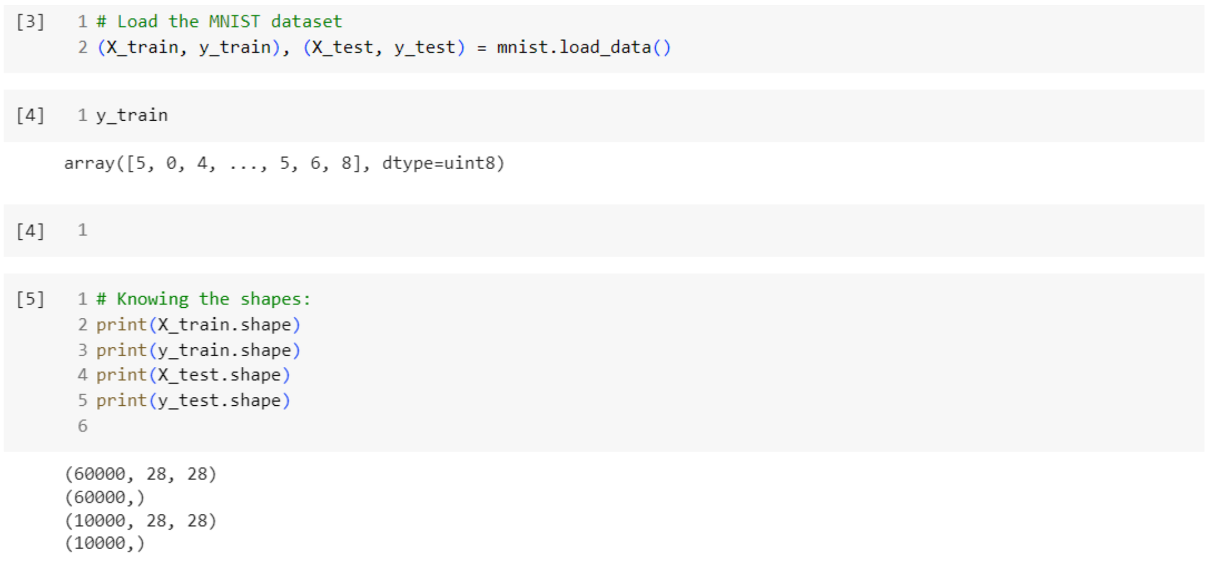


```
[1]
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow import keras
import cv2
from keras.datasets import mnist
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
%matplotlib inline
```



```
[2]
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.utils import to_categorical
```

Loading the dataset:



```
[3]
# Load the MNIST dataset
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

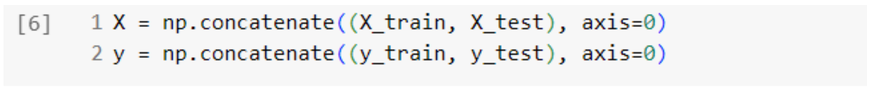
```
[4]
y_train
```

```
array([5, 0, 4, ..., 5, 6, 8], dtype=uint8)
```

```
[4]
```

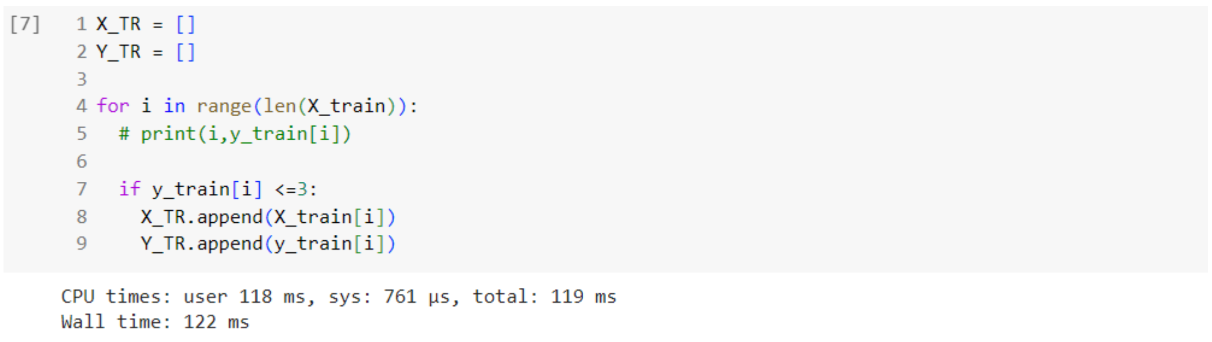
```
[5]
# Knowing the shapes:
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(60000, 28, 28)
(60000,)
(10000, 28, 28)
(10000,)
```



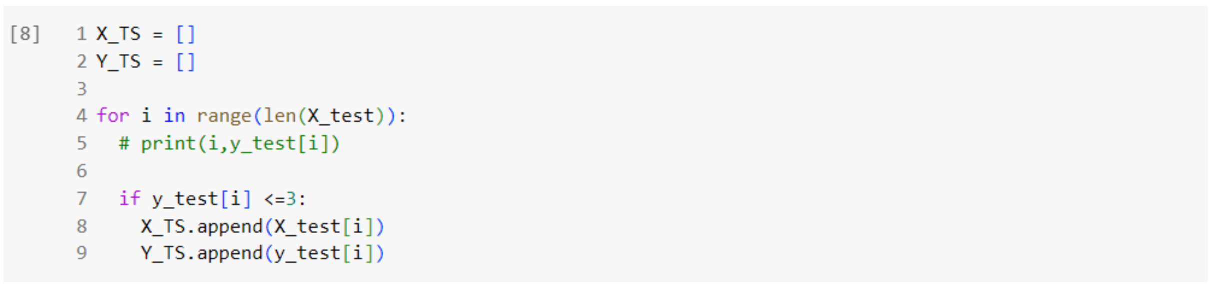
```
[6]
X = np.concatenate((X_train, X_test), axis=0)
y = np.concatenate((y_train, y_test), axis=0)
```

- Filtering the data (0 vs. 1 vs. 2 vs. 3):

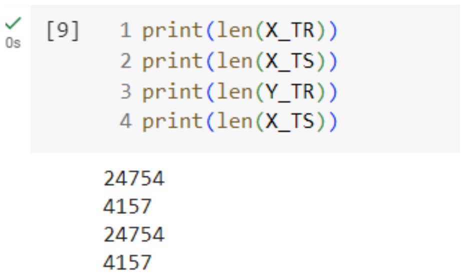


```
X_TR = []
Y_TR = []

for i in range(len(X_train)):
  # print(i,y_train[i])

  if y_train[i] <=3:
    X_TR.append(X_train[i])
    Y_TR.append(y_train[i])
```

```
CPU times: user 118 ms, sys: 761 µs, total: 119 ms
Wall time: 122 ms
```



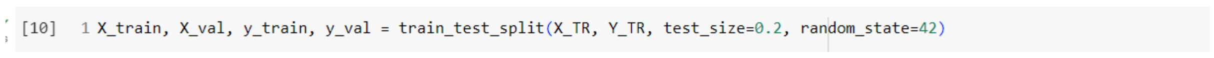
```
X_TS = []
Y_TS = []

for i in range(len(X_test)):
  # print(i,y_test[i])

  if y_test[i] <=3:
    X_TS.append(X_test[i])
    Y_TS.append(y_test[i])
```



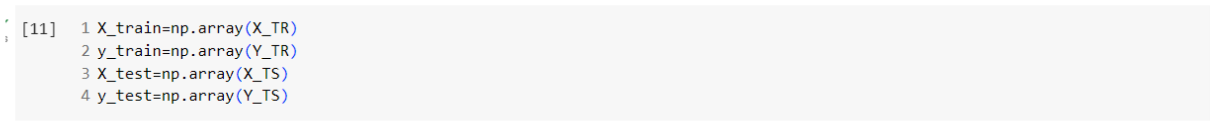
```
print(len(X_TR))
print(len(X_TS))
print(len(Y_TR))
print(len(X_TS))
```

```
24754
4157
24754
4157
```

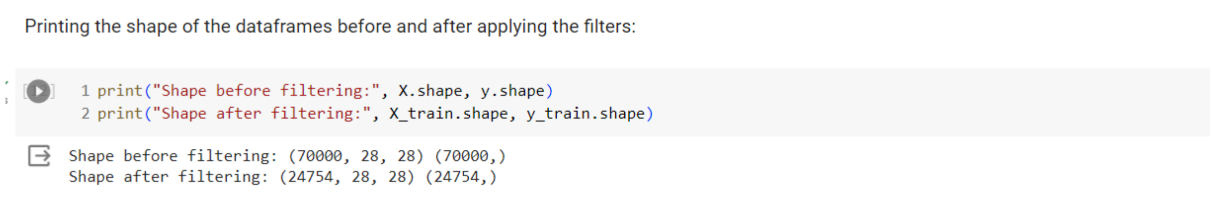
- randomly select 20% of the training images along with their corresponding labels to be the validation data.



```
X_train, X_val, y_train, y_val = train_test_split(X_TR, Y_TR, test_size=0.2, random_state=42)
```



```
X_train=np.array(X_TR)
y_train=np.array(Y_TR)
X_test=np.array(X_TS)
y_test=np.array(Y_TS)
```



Printing the shape of the dataframes before and after applying the filters:

```
print("Shape before filtering:", X.shape, y.shape)
print("Shape after filtering:", X_train.shape, y_train.shape)
```

```
Shape before filtering: (70000, 28, 28) (70000,)
Shape after filtering: (24754, 28, 28) (24754,)
```

b) Feature extraction: average the pixel values in the quadrants in each image to generate a feature vector of 4 values for each image.

```
[13]
# Calculate the average pixel values in each quadrant
X_train_avg = np.array([[
    np.mean(image[:X_train.shape[1]/2, :X_train.shape[1]/2]),  # Top-left quadrant
    np.mean(image[:X_train.shape[1]/2, X_train.shape[1]/2:]),  # Top-right quadrant
    np.mean(image[X_train.shape[1]/2:, :X_train.shape[1]/2]),  # Bottom-left quadrant
    np.mean(image[X_train.shape[1]/2:, X_train.shape[1]/2:])   # Bottom-right quadrant
] for image in X_train])
X_val_avg = np.array([[
    np.mean(image[:X_train.shape[1]/2, :X_train.shape[1]/2]),
    np.mean(image[:X_train.shape[1]/2, X_train.shape[1]/2:]),
    np.mean(image[X_train.shape[1]/2:, :X_train.shape[1]/2]),
    np.mean(image[X_train.shape[1]/2:, X_train.shape[1]/2:])
] for image in X_val])
X_test_avg = np.array([[
    np.mean(image[:X_train.shape[1]/2, :X_train.shape[1]/2]),
    np.mean(image[:X_train.shape[1]/2, X_train.shape[1]/2:]),
    np.mean(image[X_train.shape[1]/2:, :X_train.shape[1]/2]),
    np.mean(image[X_train.shape[1]/2:, X_train.shape[1]/2:])
] for image in X_test])

```

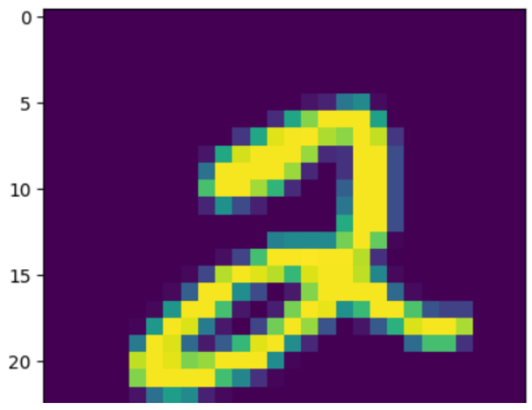
c) Convert the label vectors for all the sets to binary class matrices using to_categorical() Keras function

```
y_train = to_categorical(y_train, num_classes=4)
y_val = to_categorical(y_val, num_classes=4)
y_test = to_categorical(y_test, num_classes=4)
```

Verifying both the actual and predicted digit

```
[15]
X_train[2]
plt.imshow(X_train[2])
```

```
<matplotlib.image.AxesImage at 0x783c854bbeb0>
```



```
[16]
# Make a prediction for X_train[0]
model = Sequential()
predicted_probabilities = model.predict(X_train_avg[2].reshape(1, -1))

# Convert the predicted probabilities to the corresponding digit
predicted_digit = np.argmax(predicted_probabilities)

# Display the actual and predicted digit
actual_digit = np.argmax(y_train[2])
print(f"Actual Digit: {actual_digit}, Predicted Digit: {predicted_digit}")
```

```
1/1 [==============================] - 0s 75ms/step
Actual Digit: 2, Predicted Digit: 2
```

d) Build, compile, train, and then evaluate:

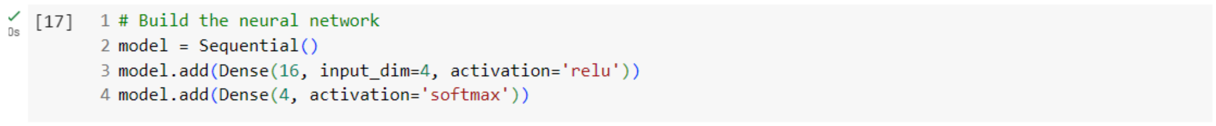
i. Build a neural network with 1 layer that contains 16 nodes using the Keras library.

ii. Compile the network. Make sure to select a correct loss function for this classification problem. Use stochastic gradient descent learning (SGD, learning rate of 0.0001). Explain your selection of the loss function.

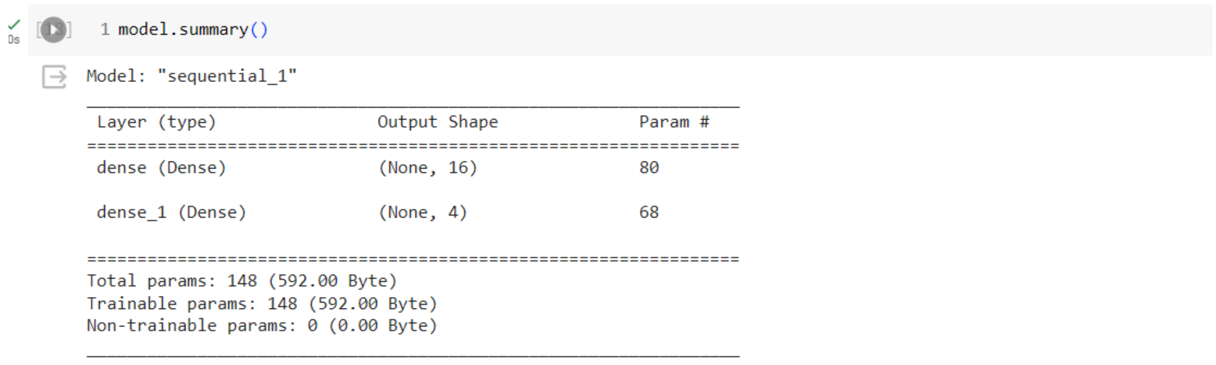
- **We used categorical_crossentropy loss function**
- **categorical_crossentropy is used for multi-class classification problems, where each input sample can belong to multiple classes**

iii. Train the network for 30 epochs and a batch size of 16.

iv. Plot the training loss (i.e., the learning curve) for all the epochs. v. Use the evaluate() Keras function to find the training and validation loss and accuracy.

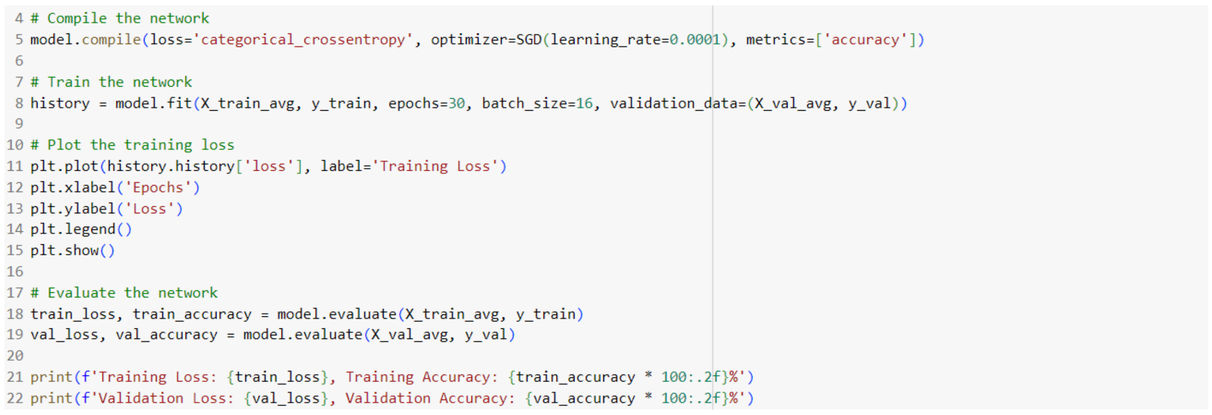


```
[17]
# Build the neural network
model = Sequential()
model.add(Dense(16, input_dim=4, activation='relu'))
model.add(Dense(4, activation='softmax'))
```

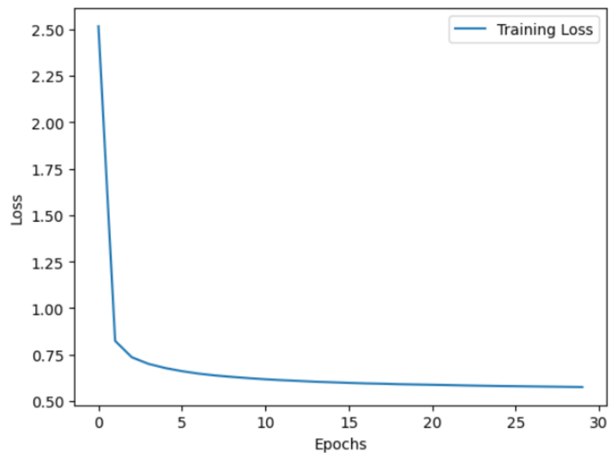


```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 16)                80

 dense_1 (Dense)             (None, 4)                 68

=================================================================
Total params: 148 (592.00 Byte)
Trainable params: 148 (592.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```



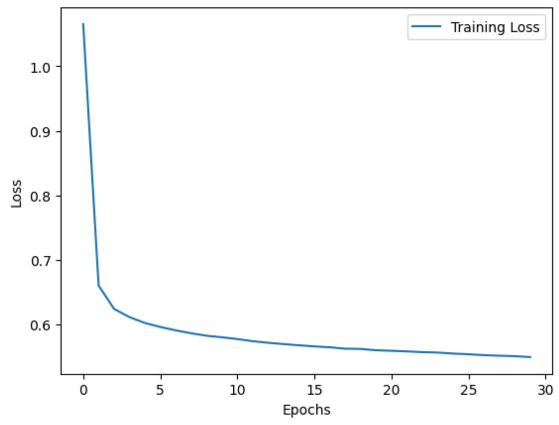
```
# Compile the network
model.compile(loss='categorical_crossentropy', optimizer=SGD(learning_rate=0.0001), metrics=['accuracy'])

# Train the network
history = model.fit(X_train_avg, y_train, epochs=30, batch_size=16, validation_data=(X_val_avg, y_val))

# Plot the training loss
plt.plot(history.history['loss'], label='Training Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

# Evaluate the network
train_loss, train_accuracy = model.evaluate(X_train_avg, y_train)
val_loss, val_accuracy = model.evaluate(X_val_avg, y_val)

print(f'Training Loss: {train_loss}, Training Accuracy: {train_accuracy * 100:.2f}%')
print(f'Validation Loss: {val_loss}, Validation Accuracy: {val_accuracy * 100:.2f}%')
```



```
774/774 [==============================] - 1s 1ms/step - loss: 0.5797 - accuracy: 0.7849
155/155 [==============================] - 0s 2ms/step - loss: 0.6015 - accuracy: 0.7710
Training Loss: 0.5797455906867981, Training Accuracy: 78.49%
Validation Loss: 0.6015303134918213, Validation Accuracy: 77.10%
```

For 1 layer and 64 nodes:

```python
# Build the neural network
model = Sequential()
model.add(Dense(64, input_dim=4, activation='relu'))
model.add(Dense(4, activation='softmax'))
print(model.summary())
# Compile the network
model.compile(loss='categorical_crossentropy', optimizer=SGD(learning_rate=0.0001), metrics=['accuracy'])

# Train the network
history = model.fit(X_train_avg, y_train, epochs=30, batch_size=16, validation_data=(X_val_avg, y_val))

# Plot the training loss
plt.plot(history.history['loss'], label='Training Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

# Evaluate the network
train_loss, train_accuracy = model.evaluate(X_train_avg, y_train)
val_loss, val_accuracy = model.evaluate(X_val_avg, y_val)

print(f'Training Loss: {train_loss}, Training Accuracy: {train_accuracy * 100:.2f}%')
print(f'Validation Loss: {val_loss}, Validation Accuracy: {val_accuracy * 100:.2f}%')
```

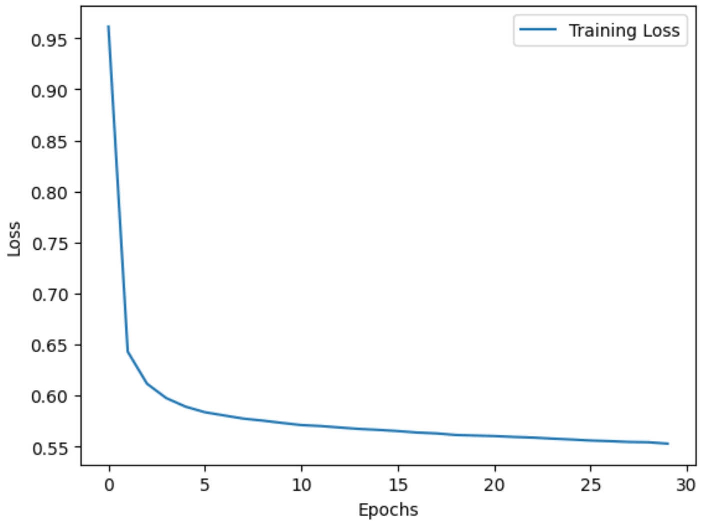


```
774/774 [==============================] - 1s 2ms/step - loss: 0.5480 - accuracy: 0.7942
155/155 [==============================] - 0s 3ms/step - loss: 0.5672 - accuracy: 0.7833
Training Loss: 0.5480408072471619, Training Accuracy: 79.42%
Validation Loss: 0.5672495365142822, Validation Accuracy: 78.33%
```

For 1 layer and 128 nodes:

For 1 layer and 128 nodes:

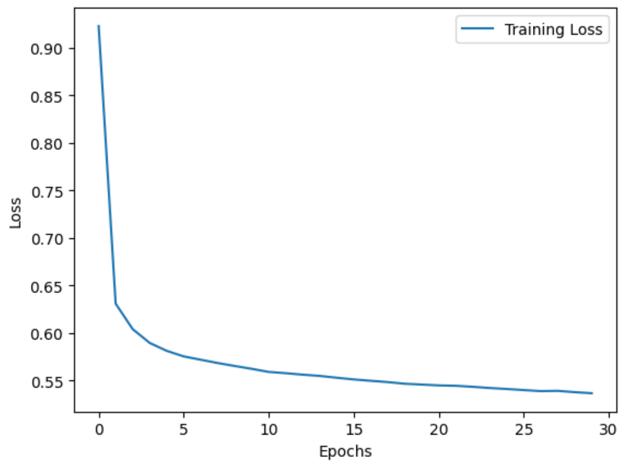
```python
# Build the neural network
model = Sequential()
model.add(Dense(128, input_dim=4, activation='relu'))
model.add(Dense(4, activation='softmax'))
print(model.summary())
# Compile the network
model.compile(loss='categorical_crossentropy', optimizer=SGD(learning_rate=0.0001), metrics=['accuracy'])

# Train the network
history = model.fit(X_train_avg, y_train, epochs=30, batch_size=16, validation_data=(X_val_avg, y_val))

# Plot the training loss
plt.plot(history.history['loss'], label='Training Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

# Evaluate the network
train_loss, train_accuracy = model.evaluate(X_train_avg, y_train)
val_loss, val_accuracy = model.evaluate(X_val_avg, y_val)

print(f'Training Loss: {train_loss}, Training Accuracy: {train_accuracy * 100:.2f}%')
print(f'Validation Loss: {val_loss}, Validation Accuracy: {val_accuracy * 100:.2f}%')
```



```
774/774 [==============================] - 1s 1ms/step - loss: 0.5495 - accuracy: 0.7940
155/155 [==============================] - 0s 1ms/step - loss: 0.5706 - accuracy: 0.7829
Training Loss: 0.5494880080223083, Training Accuracy: 79.40%
Validation Loss: 0.570569634437561, Validation Accuracy: 78.29%
```

For 2 layers 128 nodes, 16 nodes

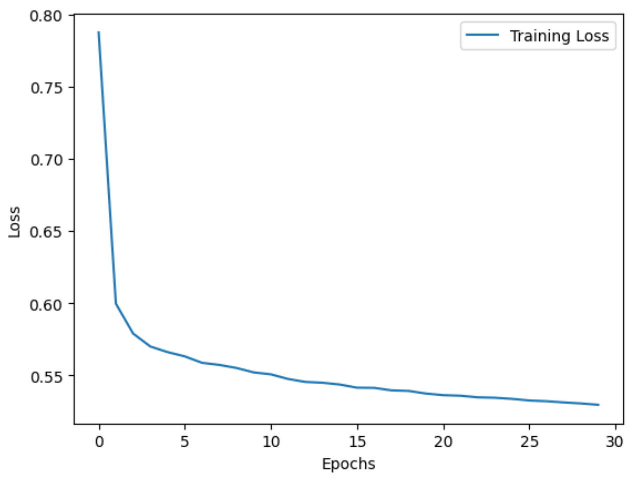
```python
# Build the neural network with two hidden layers
model = Sequential()
model.add(Dense(128, input_dim=4, activation='relu'))  # First hidden layer with 128 nodes
model.add(Dense(16, activation='relu'))  # Second hidden layer with 16 nodes
model.add(Dense(4, activation='softmax'))  # Output layer with 4 nodes (for 0-3 digits) and softmax activation

# Compile the network
model.compile(loss='categorical_crossentropy', optimizer=SGD(learning_rate=0.0001), metrics=['accuracy'])

# Train the network
history = model.fit(X_train_avg, y_train, epochs=30, batch_size=16, validation_data=(X_val_avg, y_val))

# Plot the training loss
plt.plot(history.history['loss'], label='Training Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

# Evaluate the network
train_loss, train_accuracy = model.evaluate(X_train_avg, y_train)
val_loss, val_accuracy = model.evaluate(X_val_avg, y_val)

print(f'Training Loss: {train_loss}, Training Accuracy: {train_accuracy * 100:.2f}%')
print(f'Validation Loss: {val_loss}, Validation Accuracy: {val_accuracy * 100:.2f}%')

```



```
774/774 [==============================] - 1s 2ms/step - loss: 0.5343 - accuracy: 0.7977
155/155 [==============================] - 0s 2ms/step - loss: 0.5514 - accuracy: 0.7859
Training Loss: 0.5342774987220764, Training Accuracy: 79.77%
Validation Loss: 0.5514405369758606, Validation Accuracy: 78.59%
```

For 2 layers 128 nodes, 64 nodes

```python

# Build the neural network with two hidden layers
model = Sequential()
model.add(Dense(128, input_dim=4, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(4, activation='softmax'))

# Compile the network
model.compile(loss='categorical_crossentropy', optimizer=SGD(learning_rate=0.0001), metrics=['accuracy'])

# Train the network
history = model.fit(X_train_avg, y_train, epochs=30, batch_size=16, validation_data=(X_val_avg, y_val))

# Plot the training loss
plt.plot(history.history['loss'], label='Training Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

# Evaluate the network
train_loss, train_accuracy = model.evaluate(X_train_avg, y_train)
val_loss, val_accuracy = model.evaluate(X_val_avg, y_val)

print(f'Training Loss: {train_loss}, Training Accuracy: {train_accuracy * 100:.2f}%')
print(f'Validation Loss: {val_loss}, Validation Accuracy: {val_accuracy * 100:.2f}%')

```



```
774/774 [==============================] - 2s 2ms/step - loss: 0.5268 - accuracy: 0.7981
155/155 [==============================] - 0s 2ms/step - loss: 0.5468 - accuracy: 0.7883
Training Loss: 0.5267793536186218, Training Accuracy: 79.81%
Validation Loss: 0.5467689037322998, Validation Accuracy: 78.83%
```

Model #	Details	Training		Validation	
		loss	accuracy	loss	accuracy
1	1 layer 16 nodes	0.5653371810913086	79.21%	0.6004406809806824	77.68%
2	1 layer 64 nodes	0.5409508943557739	79.72%	0.5718497037887573	78.02%
3	1 layer 128 nodes	0.5416780710220337	79.88%	0.5668755769729614	78.59%
4	2 layers 128 nodes, 16 nodes	0.5319947600364685	80.00%	0.5584332942962646	78.25%
5	2 layers 128 nodes, 64 nodes	0.5406889319419861	79.77%	0.5699062943458557	78.43%

f) What behavior do you observe in the training loss and the validation loss when you increase the number layers and nodes in the previous table. Which model is more suitable in this problem? Explain.

From the results we get, it is clear that the behaviour of training and validation loss changes when we change the number of layers and nodes in the neural network. Based on the provided results, the 2-layer model with 128 nodes in the first layer and 16 nodes in the second layer appears to be the most suitable for this problem. It offers the following advantages:

It achieves the lowest validation loss among the models.

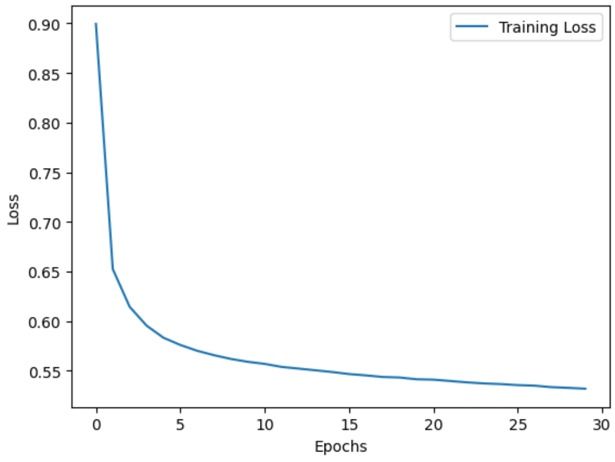
It has the highest validation accuracy, indicating better generalization.

It exhibits less overfitting compared to the 1-layer models, as evidenced by the smaller gap between training and validation losses.

a) Evaluate the selected model in part (e) on the testing set and report the testing loss and accuracy.

Testing Loss: 0.5267793536186218, Testing Accuracy: 79.81%

```python
# Build the neural network with two hidden layers
model = Sequential()
model.add(Dense(128, input_dim=4, activation='relu'))  # First hidden layer with 128 nodes
model.add(Dense(16, activation='relu'))  # Second hidden layer with 16 nodes
model.add(Dense(4, activation='softmax'))  # Output layer with 4 nodes (for 0-3 digits) and softmax activation

# Compile the network
model.compile(loss='categorical_crossentropy', optimizer=SGD(learning_rate=0.0001), metrics=['accuracy'])

# Train the network
history = model.fit(X_train_avg, y_train, epochs=30, batch_size=16, validation_data=(X_val_avg, y_val))

# Plot the training loss
plt.plot(history.history['loss'], label='Training Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

# Evaluate the network
test_loss, test_accuracy = model.evaluate(X_test_avg, y_test)

print(f'Testing Loss: {train_loss}, Testing Accuracy: {train_accuracy * 100:.2f}%')
```



```
130/130 [==============================] - 0s 2ms/step - loss: 0.5182 - accuracy: 0.8090
Testing Loss: 0.5267793536186218, Testing Accuracy: 79.81%
```