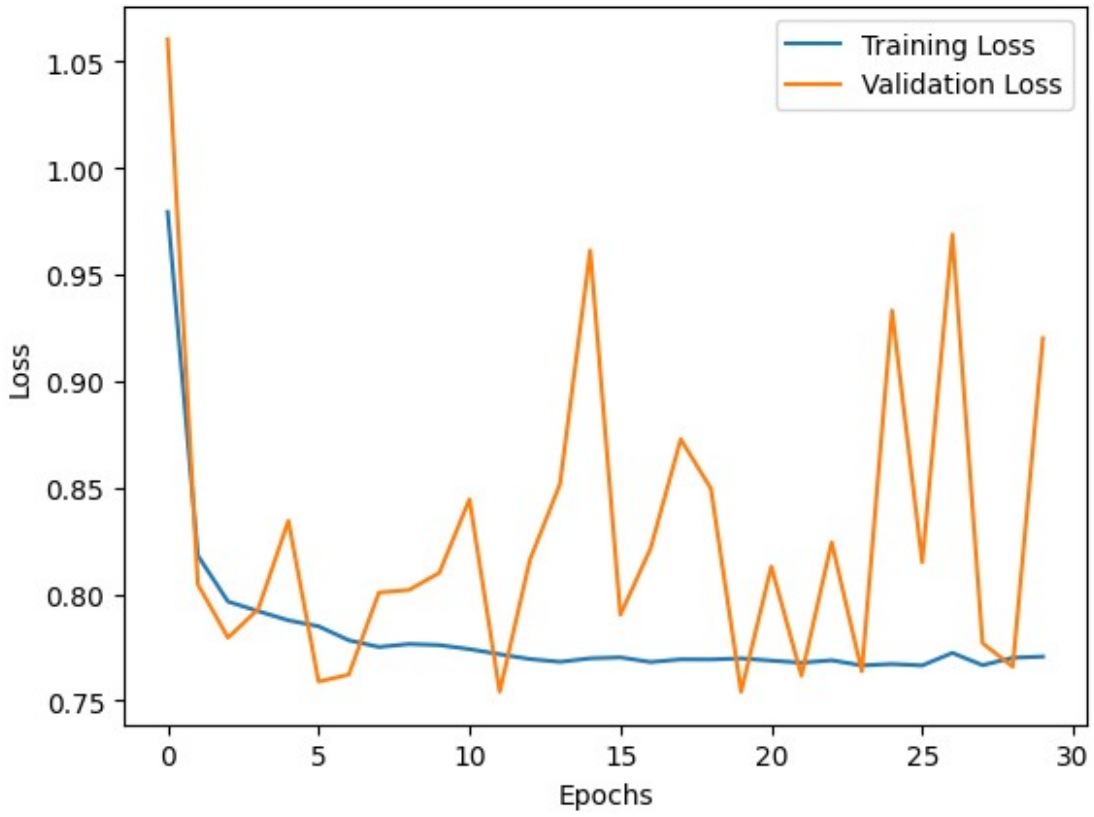
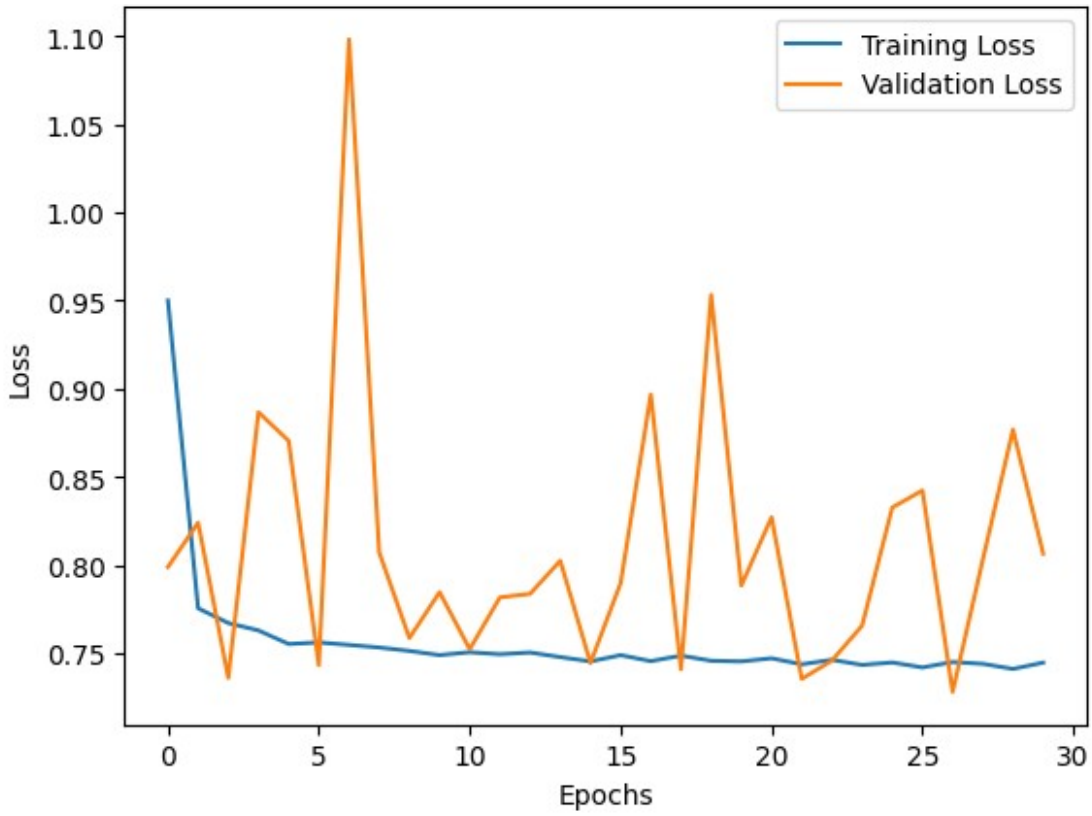
```
import numpy as np
import matplotlib.pyplot as plt
from keras.datasets import mnist
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import SGD

#load MNIST dataset and split into training and testing sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()

#filter dataset for digits 0, 3, and 8
train_filter = np.where((y_train == 0) | (y_train == 3) | (y_train ==
8))
test_filter = np.where((y_test == 0) | (y_test == 3) | (y_test == 8))

x_train, y_train = x_train[train_filter], y_train[train_filter]
x_test, y_test = x_test[test_filter], y_test[test_filter]

#split training data into training and validation sets
validation_split = 0.2
split_index = int((1 - validation_split) * len(x_train))

x_val, y_val = x_train[split_index:], y_train[split_index:]
x_train, y_train = x_train[:split_index], y_train[:split_index]

#feature extraction: average the pixel values in the quadrants
def extract_features(data):
    num_samples = data.shape[0]
    features = np.zeros((num_samples, 4))
    for i in range(num_samples):
        img = data[i]
        features[i][0] = np.mean(img[:14, :14])  # Top left quadrant
        features[i][1] = np.mean(img[:14, 14:])  # Top right quadrant
        features[i][2] = np.mean(img[14:, :14])  # Bottom left
quadrant
        features[i][3] = np.mean(img[14:, 14:])  # Bottom right
quadrant
    return features

x_train_features = extract_features(x_train)
x_val_features = extract_features(x_val)
x_test_features = extract_features(x_test)

#remap labels to start from 0 and be continuous
label_map = {0: 0, 3: 1, 8: 2}
y_train = np.array([label_map[label] for label in y_train])
y_val = np.array([label_map[label] for label in y_val])
y_test = np.array([label_map[label] for label in y_test])
```

```
#convert labels to binary class matrices
num_classes = len(label_map)
y_train = to_categorical(y_train, num_classes)
y_val = to_categorical(y_val, num_classes)
y_test = to_categorical(y_test, num_classes)

#define function to build, compile, and train the model
def build_model(num_layers, nodes):
    model = Sequential()
    model.add(Dense(nodes, activation='relu', input_shape=(4,)))
    if num_layers == 2:
        model.add(Dense(16, activation='relu'))
    model.add(Dense(3, activation='softmax'))

    model.compile(optimizer=SGD(lr=0.0001),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
    return model

#define function to train the model and plot the learning curve
def train_model(model, x_train, y_train, x_val, y_val):
    history = model.fit(x_train, y_train, epochs=30, batch_size=16,
                        validation_data=(x_val, y_val), verbose=0)
    plt.plot(history.history['loss'], label='Training Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.show()
    return history

#define function to evaluate the model on testing set
def evaluate_model(model, x_test, y_test):
    loss, accuracy = model.evaluate(x_test, y_test)
    print("Testing Loss:", loss)
    print("Testing Accuracy:", accuracy)

#train and evaluate models
models = [(1, 16), (1, 64), (1, 128), (2, 128), (2, 64)]

for i, (num_layers, nodes) in enumerate(models, 1):
    print("Model", i)
    model = build_model(num_layers, nodes)
    print("Training:")
    history = train_model(model, x_train_features, y_train,
x_val_features, y_val)
    print("Evaluation:")
    evaluate_model(model, x_test_features, y_test)
```

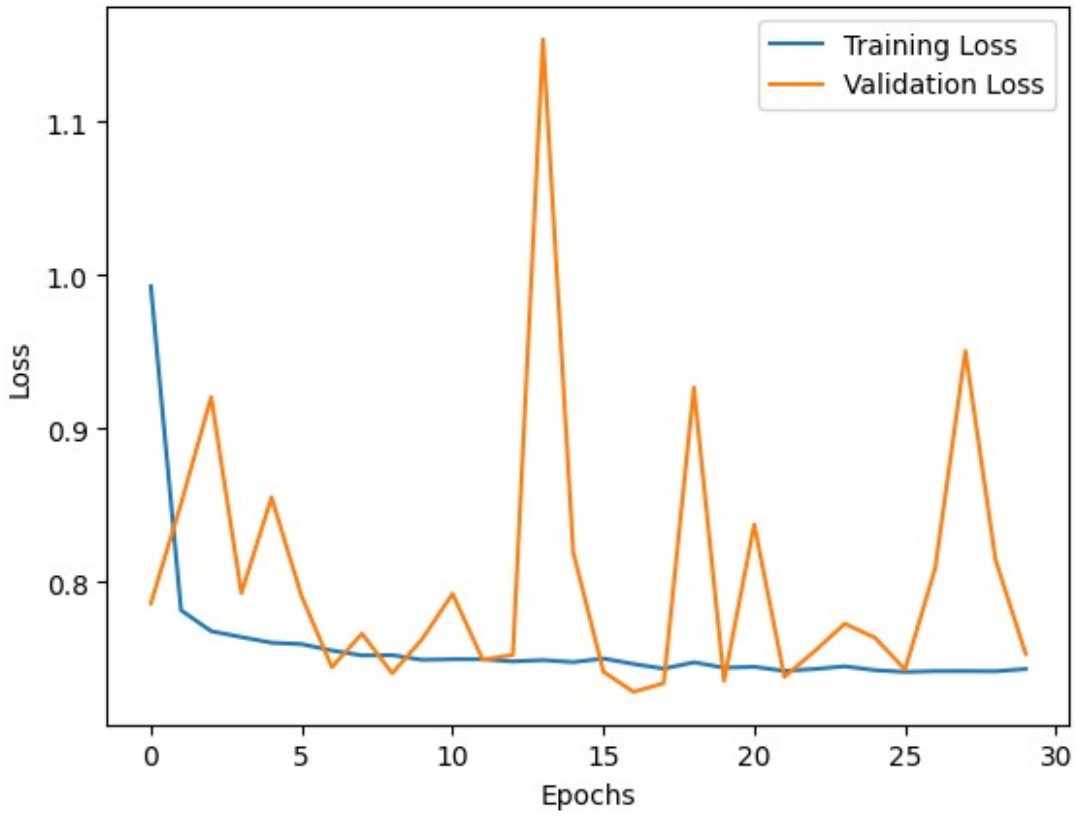
```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use
`learning_rate` or use the legacy optimizer,
e.g.,tf.keras.optimizers.legacy.SGD.

Model 1
Training:
```



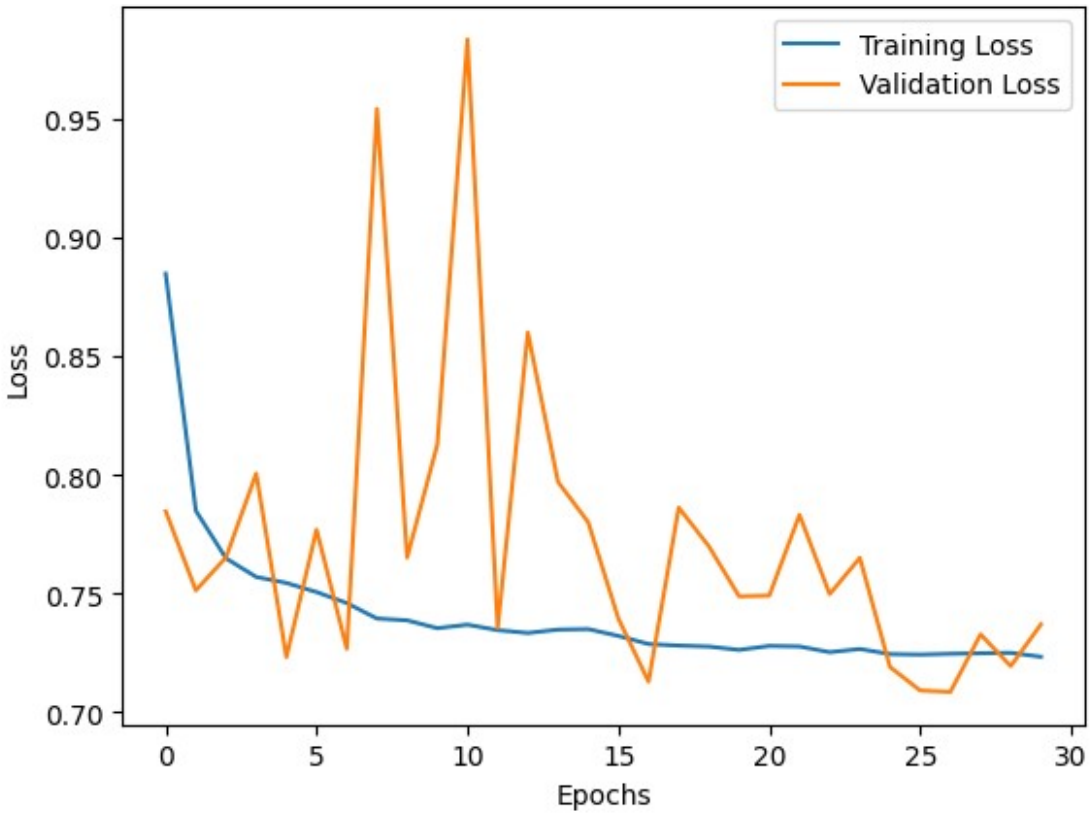

```
Evaluation:
93/93 [==============================] - 0s 2ms/step - loss: 0.9093 -
accuracy: 0.5992
Testing Loss: 0.909250795841217
Testing Accuracy: 0.5991902947425842
Model 2

WARNING:absl:`lr` is deprecated in Keras optimizer, please use
`learning_rate` or use the legacy optimizer,
e.g.,tf.keras.optimizers.legacy.SGD.

Training:
```



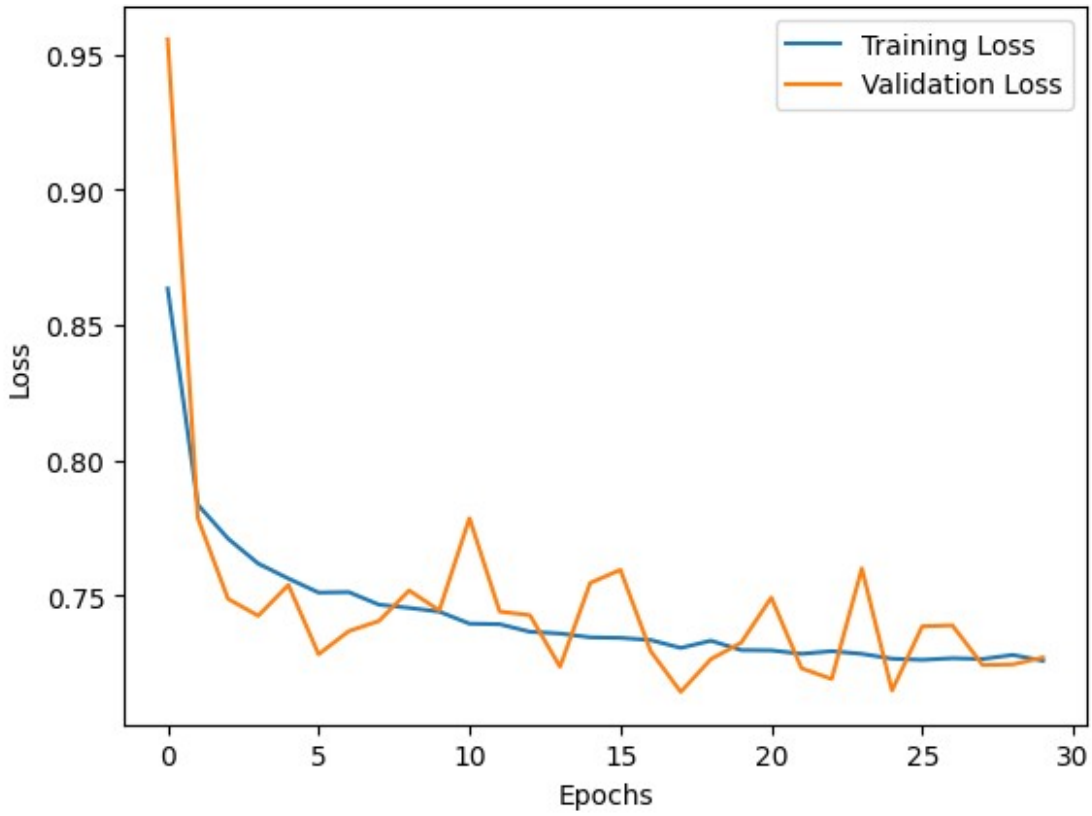

```
Evaluation:
93/93 [==============================] - 0s 1ms/step - loss: 0.8140 -
accuracy: 0.6336

WARNING:absl:`lr` is deprecated in Keras optimizer, please use
`learning_rate` or use the legacy optimizer,
e.g.,tf.keras.optimizers.legacy.SGD.

Testing Loss: 0.8139521479606628
Testing Accuracy: 0.6336032152175903
Model 3
Training:
```




```
Evaluation:
93/93 [==============================] - 0s 2ms/step - loss: 0.7747 -
accuracy: 0.6302
Testing Loss: 0.7747158408164978
Testing Accuracy: 0.6302294135093689
Model 4

WARNING:absl:`lr` is deprecated in Keras optimizer, please use
`learning_rate` or use the legacy optimizer,
e.g.,tf.keras.optimizers.legacy.SGD.

Training:
```




```
Evaluation:
93/93 [==============================] - 0s 2ms/step - loss: 0.7393 -
accuracy: 0.6515
Testing Loss: 0.7393301129341125
Testing Accuracy: 0.651484489440918
Model 5

WARNING:absl:`lr` is deprecated in Keras optimizer, please use
`learning_rate` or use the legacy optimizer,
e.g.,tf.keras.optimizers.legacy.SGD.

Training:
```




```
Evaluation:
93/93 [==============================] - 0s 1ms/step - loss: 0.7495 -
accuracy: 0.6441
Testing Loss: 0.7495449185371399
Testing Accuracy: 0.6440621018409729
```