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

```
# Loading MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 2s 0us/step
```

```
# Filter for classes 0, 3, and 8
filter_classes = [0, 3, 8]
train_filter = np.isin(y_train, filter_classes)
test_filter = np.isin(y_test, filter_classes)

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

```
# Map class labels to 0, 1, 2
class_mapping = {0: 0, 3: 1, 8: 2}
y_train = np.array([class_mapping[label] for label in y_train])
y_test = np.array([class_mapping[label] for label in y_test])
```

```
# Extract features by averaging pixel values in each quadrant
def extract_features(images):
    features = np.zeros((images.shape[0], 4))
    for i, img in enumerate(images):
        upper_left = img[:14, :14]
        upper_right = img[:14, 14:]
        lower_left = img[14:, :14]
        lower_right = img[14:, 14:]

        features[i, 0] = np.mean(upper_left)
        features[i, 1] = np.mean(upper_right)
        features[i, 2] = np.mean(lower_left)
        features[i, 3] = np.mean(lower_right)

    return features
```

```
# Extract features
x_train_features = extract_features(x_train)
x_test_features = extract_features(x_test)
```

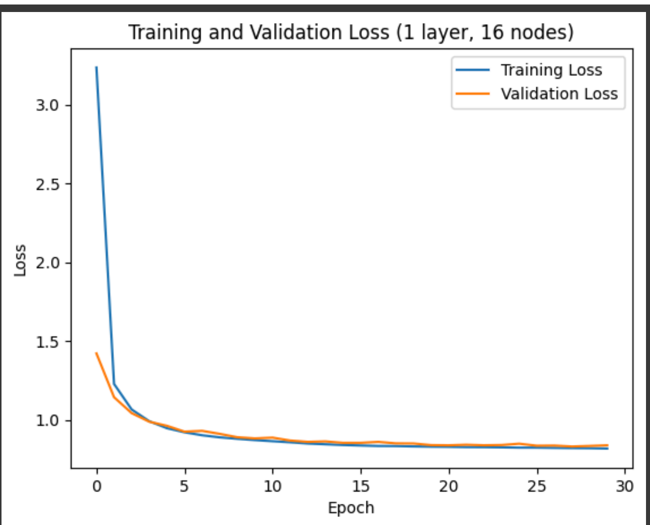
```
# Split the training set into 80% training and 20% validation sets
x_train_features, x_val_features, y_train, y_val = train_test_split(x_train_features, y_train, test_size=0.2, random_state=42)
```

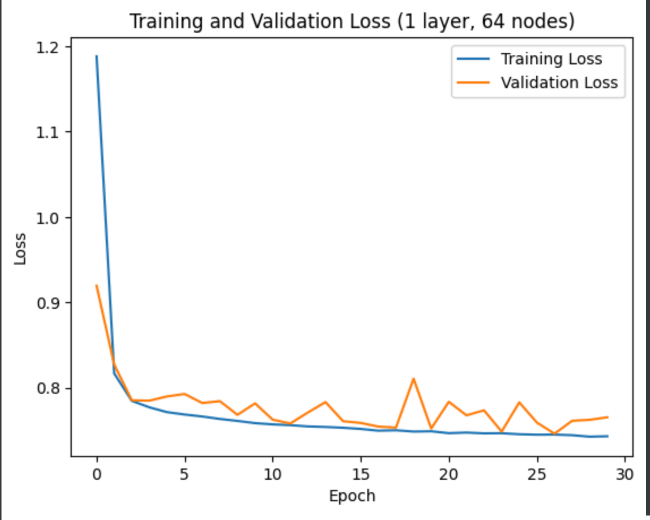
```
# Convert labels to categorical
y_train = to_categorical(y_train, 3)
y_val = to_categorical(y_val, 3)
y_test = to_categorical(y_test, 3)
```

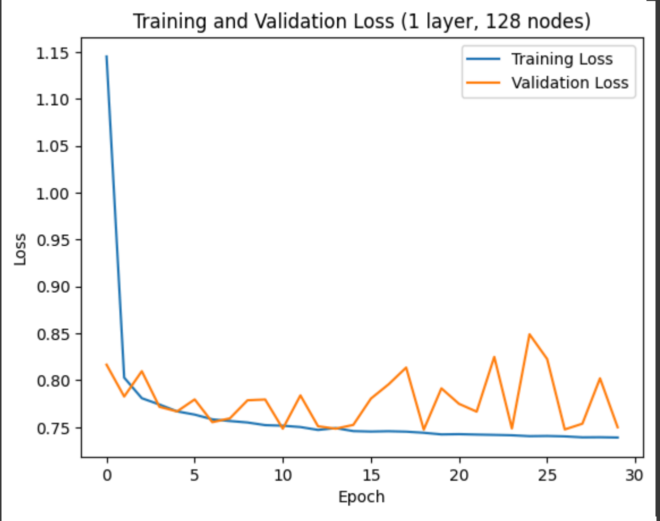
```
# Build, compile, and train a model
def build_and_train_model(layers):
    model = Sequential()
    for units in layers:
        if model.layers:
            model.add(Dense(units, activation='relu'))
        else:
            model.add(Dense(units, input_dim=4, activation='relu'))
    model.add(Dense(3, activation='softmax'))

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

    history = model.fit(x_train_features, y_train,
                        epochs=30,
                        batch_size=16,
                        validation_data=(x_val_features, y_val),
                        verbose=0)
    train_loss, train_accuracy = model.evaluate(x_train_features, y_train, verbose=0)
    val_loss, val_accuracy = model.evaluate(x_val_features, y_val, verbose=0)

    return train_loss, train_accuracy, val_loss, val_accuracy, history
```

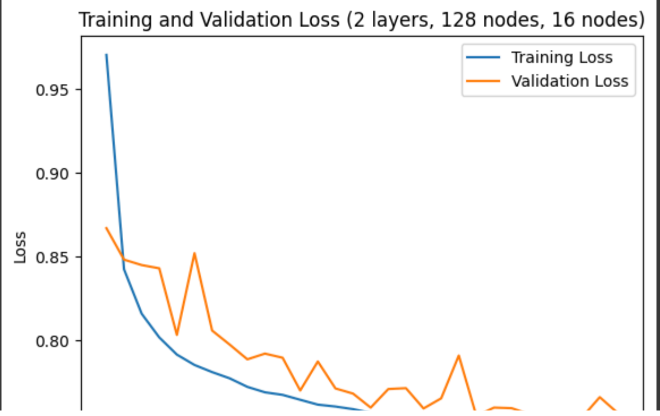
```
# Model configurations
models = [
    {"details": "1 layer, 16 nodes", "layers": [16]},
    {"details": "1 layer, 64 nodes", "layers": [64]},
    {"details": "1 layer, 128 nodes", "layers": [128]},
    {"details": "2 layers, 128 nodes, 16 nodes", "layers": [128, 16]},
    {"details": "2 layers, 128 nodes, 64 nodes", "layers": [128, 64]}
]

results = []
```

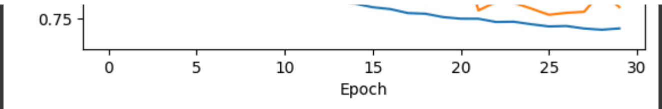
```
# Train and evaluate each model
for model_config in models:
    train_loss, train_accuracy, val_loss, val_accuracy, history = build_and_train_model(model_config["layers"])
    results.append({
        "Model #": len(results) + 1,
        "Details": model_config["details"],
        "Training Loss": train_loss,
        "Training Accuracy": train_accuracy,
        "Validation Loss": val_loss,
        "Validation Accuracy": val_accuracy
    })

    # Plot the training losses
    plt.figure()
    plt.plot(history.history['loss'], label='Training Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.title(f'Training and Validation Loss ({model_config["details"]})')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()
    plt.show()
```

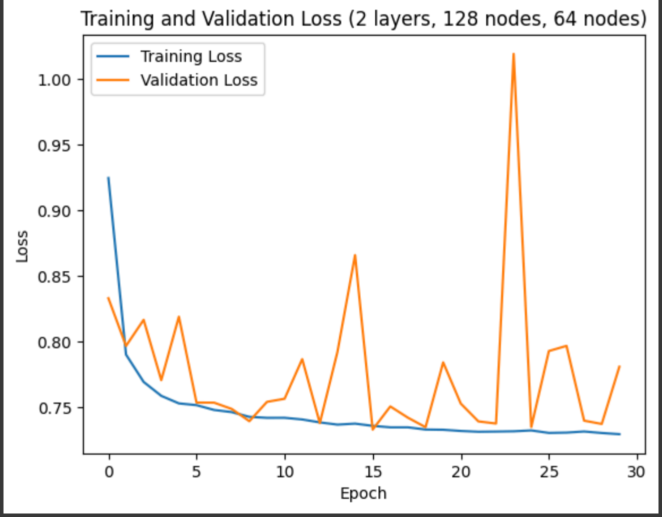

Training and Validation Loss (1 layer, 16 nodes)
Training Loss
Validation Loss
Loss
Epoch
3.0
2.5
2.0
1.5
1.0
0
5
10
15
20
25
30


Training and Validation Loss (1 layer, 64 nodes)
Training Loss
Validation Loss
Loss
Epoch
1.2
1.1
1.0
0.9
0.8
0
5
10
15
20
25
30


Training and Validation Loss (1 layer, 128 nodes)
Training Loss
Validation Loss
Loss
Epoch
1.15
1.10
1.05
1.00
0.95
0.90
0.85
0.80
0.75
0
5
10
15
20
25
30


Training and Validation Loss (2 layers, 128 nodes, 16 nodes)
Training Loss
Validation Loss
Loss
0.95
0.90
0.85
0.80





```
# Display the results table
results_df = pd.DataFrame(results)
print(results_df)
```

```
   Model #                      Details  Training Loss  Training Accuracy  \
0        1             1 layer, 16 nodes       0.825930           0.592991
1        2             1 layer, 64 nodes       0.750790           0.657009
2        3            1 layer, 128 nodes       0.734693           0.661128
3        4  2 layers, 128 nodes, 16 nodes       0.742867           0.659802
4        5  2 layers, 128 nodes, 64 nodes       0.765703           0.649330

   Validation Loss  Validation Accuracy
0         0.838022             0.583077
1         0.765326             0.649818
2         0.749927             0.651215
3         0.756786             0.646747
4         0.780949             0.641162
```

Increasing the number of layers and nodes in the neural network models generally led to improved performance, as shown by decreased training and validation loss and increased accuracy. Model 5, which consists of two layers with 128 nodes in the first layer and 64 nodes in the