MSDS 422-57 | Module 7 Assignment 1 | Group 5
Aug 7, 2022 | Digit Recognizer | Scott Jue
Dr. Anil Chaturvedi | | Zach Watson

We built our models using the MNIST ("Modified National Institute of Standards and Technology") data. The data set contains gray-scale images of hand-drawn digits ranging from zero to nine. A group of 784 pixels (24 pixels high and 24 pixels wide) represents each image; pixel values range from 0 to 255, corresponding to darkness (higher numbers are darker pixels). Our EDA found that the training dataset included 42,000 rows and 785 columns and the testing data included 28,000 rows and 784 columns. Each row represents a hand-drawn digit. The first column corresponds to the image label (not included in the testing data), and subsequent columns correspond to pixel values.

We began preparing the training data by removing the ‘label’ column to use as our output variable. Next, we scaled the data by dividing each value by 255 (the maximum value), so the values were between 0 and 1. Additionally, we flattened the data to use it in our artificial neural networks (ANN). We split the train.csv data into training and testing groups of 80% and 20%, respectively, for all of our models. We built two types of artificial neural networks and evaluated their performance using 13 combinations of layers and nodes. We first explored using the Sequential model from the Keras/TensorFlow library. Then we built additional neural network models using the MLPClassifer model from the sci-kit learn library. We measure the performance of each model using the time it takes to train the model, the accuracy of the model predicting the training set, and the accuracy of the model predicting the testing set.

We tested four Sequential Neural Network models with early stopping in a 2x2 crossed design benchmark experiment with {2, 4} layers and {50, 100} nodes. The models with more nodes ran slower but performed better than their counterparts with 50 nodes. Model 2 (seq2; layers 2; nodes 50) received a testing accuracy score of 0.974 and a Kaggle score of 0.970, making it our second best performing model. Model 4 (seq4; layers 4; nodes 100) received a testing accuracy score of 0.972 and a Kaggle score of 0.969, making it our third-best model.

The second ANN we tested was the MLPClassifier from sklearn. We ultimately tested {1, 3, 5} layers and {10, 50, 100} nodes for nine combinations of nodes and layers. Our models ranged from one to five layers and 10 to 100 nodes. Increasing the number of nodes in an MLPClassifier model generally corresponds to an increase in the amount of time it takes to train the model, regardless of the number of layers. We found that the models with 100 nodes generally performed at around 0.99 accuracy scores. Models with 50 nodes performed equally well if they had 1 or 3 layers but performed poorly if they had 5 layers (0.11 accuracy score). Five-layer models with 10 nodes perform similarly poorly, while 10 node models with 1 or 3 layers receive accuracy scores of around 0.95. We selected three models for Kaggle, the models with the highest testing accuracy in each layer group (1, 3, and 5), using training accuracy to break ties. Model 7 (mlp3; 1 layer; 100 nodes) received a Kaggle score of 0.973, making it our best performing model. Model 10 (mlp6; 3 layers; 100 nodes) received a Kaggle score of 0.966, and Model 13 (mlp9; 5 layers; 100 nodes) received a Kaggle score of 0.967, making them our fourth- and fifth- best scoring models.

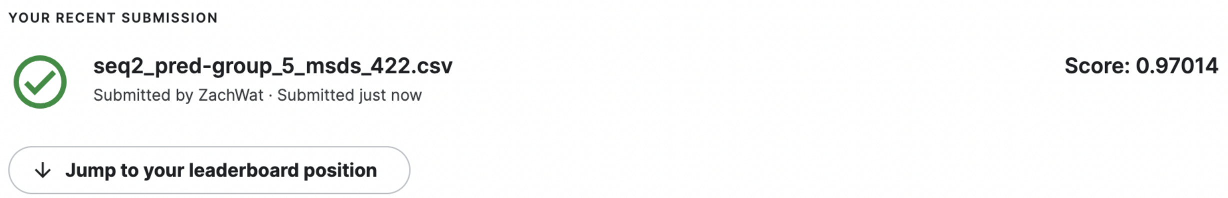
We were generally happy with our results this week. For a list of our full results, see the appendix. Our previous models received Kaggle scores ranging from 0.896 to 0.966, so our worst-performing ANN performed as well as our best-performing previous model. Additional areas for exploration include more methodical tuning of the hyperparameters of each model (layer-by-layer) and further investigation of the limits of the models. For example, would a model with 500 or 1000 nodes continue the trend of increasing accuracy for our ANN models, and what would the computational cost of implementing more advanced models be? Another area for exploration would be why our MLP models with 5 layers and nodes of 10 and 50 performed poorly.

MSDS 422-57
Aug 7, 2022
Dr. Anil Chaturvedi

Module 7 Assignment 1
Digit Recognizer

Group 5
Scott Jue
Zach Watson

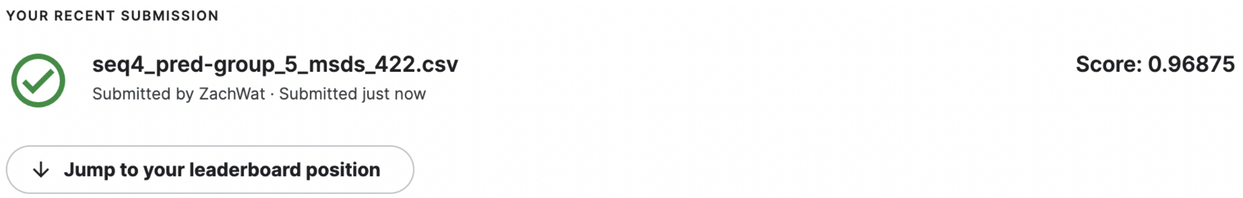
Index:



YOUR RECENT SUBMISSION

✓ **seq2_pred-group_5_msds_422.csv**
Submitted by ZachWat · Submitted just now

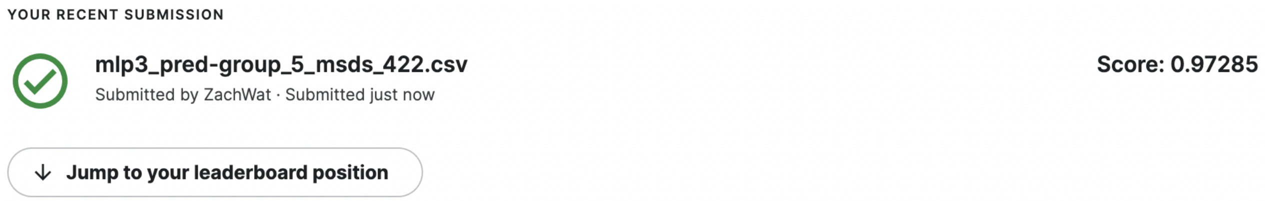
Score: 0.97014



YOUR RECENT SUBMISSION

✓ **seq4_pred-group_5_msds_422.csv**
Submitted by ZachWat · Submitted just now

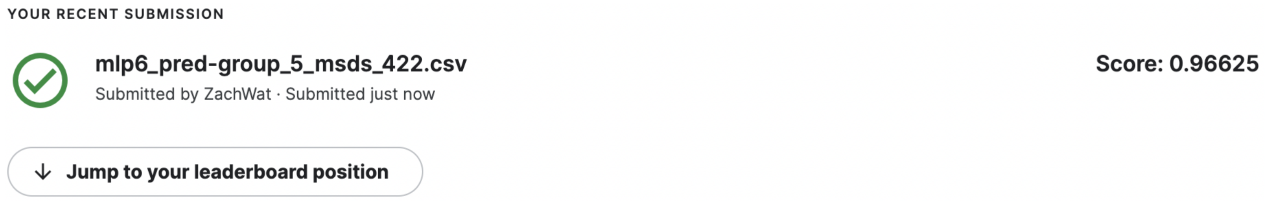
Score: 0.96875



YOUR RECENT SUBMISSION

✓ **mlp3_pred-group_5_msds_422.csv**
Submitted by ZachWat · Submitted just now

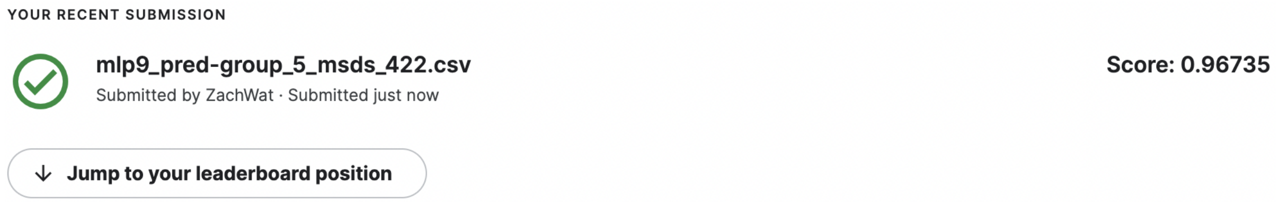
Score: 0.97285



YOUR RECENT SUBMISSION

✓ **mlp6_pred-group_5_msds_422.csv**
Submitted by ZachWat · Submitted just now

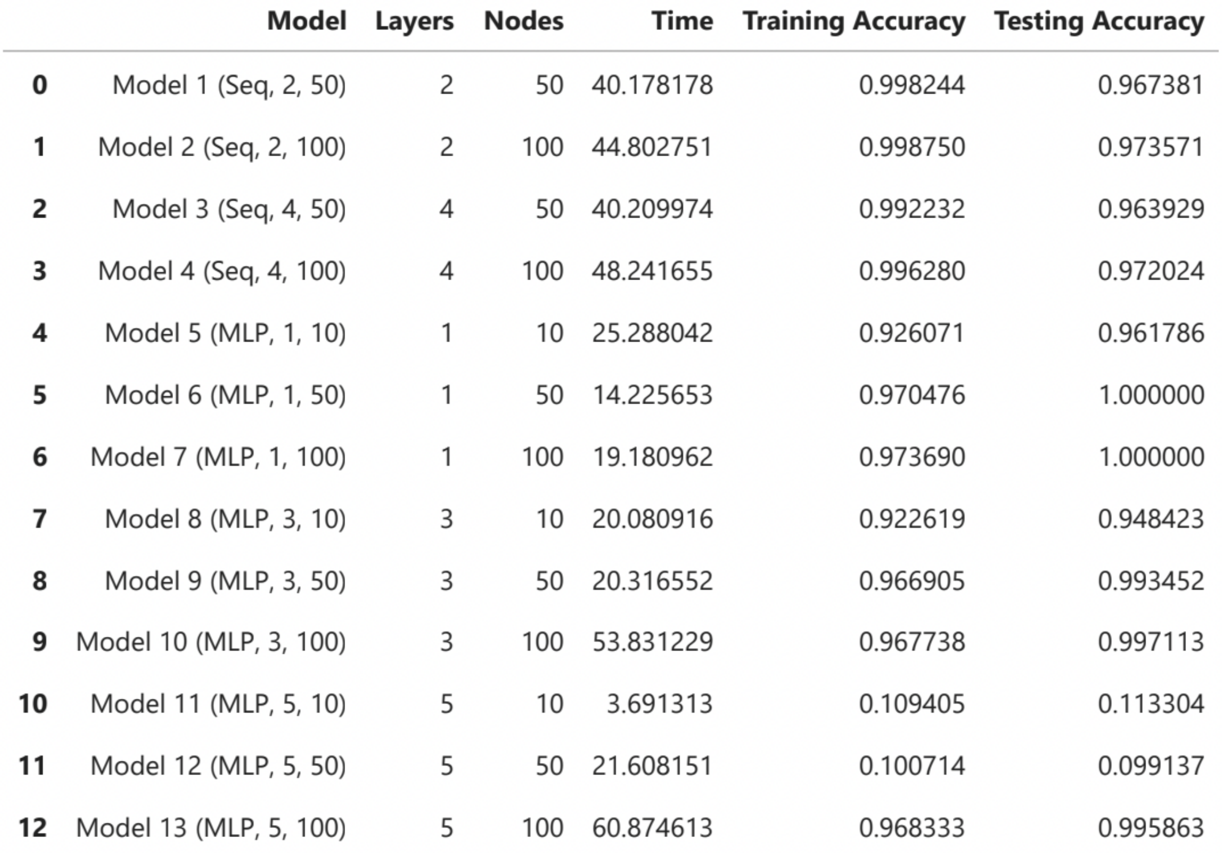
Score: 0.96625



YOUR RECENT SUBMISSION

✓ **mlp9_pred-group_5_msds_422.csv**
Submitted by ZachWat · Submitted just now

Score: 0.96735



	Model	Layers	Nodes	Time	Training Accuracy	Testing Accuracy
0	Model 1 (Seq, 2, 50)	2	50	40.178178	0.998244	0.967381
1	Model 2 (Seq, 2, 100)	2	100	44.802751	0.998750	0.973571
2	Model 3 (Seq, 4, 50)	4	50	40.209974	0.992232	0.963929
3	Model 4 (Seq, 4, 100)	4	100	48.241655	0.996280	0.972024
4	Model 5 (MLP, 1, 10)	1	10	25.288042	0.926071	0.961786
5	Model 6 (MLP, 1, 50)	1	50	14.225653	0.970476	1.000000
6	Model 7 (MLP, 1, 100)	1	100	19.180962	0.973690	1.000000
7	Model 8 (MLP, 3, 10)	3	10	20.080916	0.922619	0.948423
8	Model 9 (MLP, 3, 50)	3	50	20.316552	0.966905	0.993452
9	Model 10 (MLP, 3, 100)	3	100	53.831229	0.967738	0.997113
10	Model 11 (MLP, 5, 10)	5	10	3.691313	0.109405	0.113304
11	Model 12 (MLP, 5, 50)	5	50	21.608151	0.100714	0.099137
12	Model 13 (MLP, 5, 100)	5	100	60.874613	0.968333	0.995863

Intro

Links

Canvas: https://canvas.northwestern.edu/courses/167719/assignments/1078608?module_item_id=2319275

Kaggle: https://www.kaggle.com/c/digit-recognizer

Modules

In [127…

```
#For data manipulation and visualization
#from google.colab import files

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure
from matplotlib.pyplot import subplots_adjust
import seaborn as sns


from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.decomposition import PCA
from sklearn.metrics import accuracy_score, f1_score, classification_report, confusion_
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import MiniBatchKMeans
from sklearn import cluster

from datetime import datetime
```

Import Data

In [128…

```
# #Import data.csv from the Kaggle page linked above
# from google.colab import files
# files.upload()
```

In [129…

```
df = pd.read_csv("train.csv")
```

EDA

Intro Stats

In [130…

```
df.shape
```

```
(42000, 785)
```

Out[130…

In [131…

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 42000 entries, 0 to 41999
Columns: 785 entries, label to pixel783
dtypes: int64(785)
memory usage: 251.5 MB
```

In [132…

```
# check for missing values
print(df.isna().sum().sum())
print(np.isnan(df).sum().sum())
print(df.isnull().sum().sum())
```

```
0
0
0
```

In [133…

```
df.head(10)
```

Out[133…

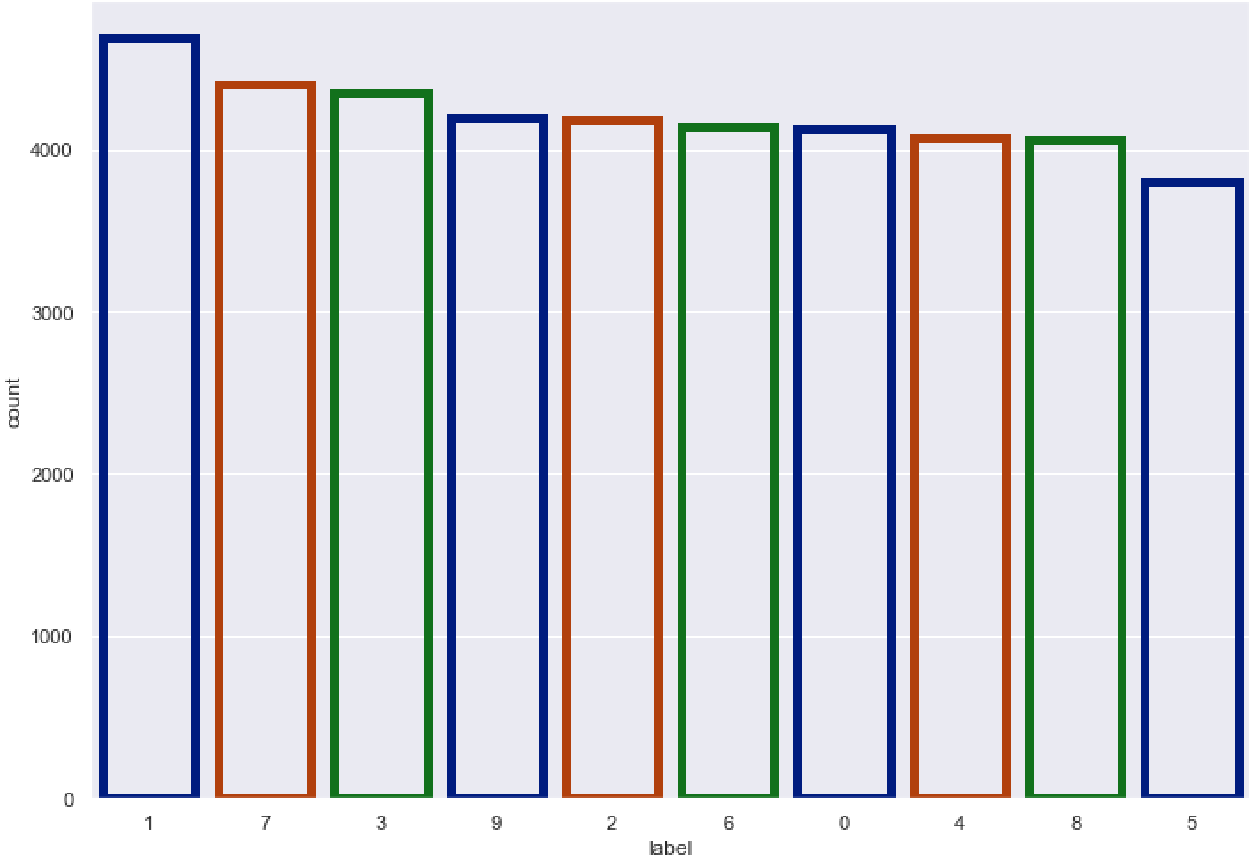
	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	pixel774	pixel775	p
0	1	0	0	0	0	0	0	0	0	0	...	0	0	
1	0	0	0	0	0	0	0	0	0	0	...	0	0	
2	1	0	0	0	0	0	0	0	0	0	...	0	0	
3	4	0	0	0	0	0	0	0	0	0	...	0	0	
4	0	0	0	0	0	0	0	0	0	0	...	0	0	
5	0	0	0	0	0	0	0	0	0	0	...	0	0	
6	7	0	0	0	0	0	0	0	0	0	...	0	0	
7	3	0	0	0	0	0	0	0	0	0	...	0	0	
8	5	0	0	0	0	0	0	0	0	0	...	0	0	
9	3	0	0	0	0	0	0	0	0	0	...	0	0	

10 rows × 785 columns

In [134…

```
#Setting plot size
sns.set(rc={'figure.figsize':(11.7,8.27)})

#Countplot
ax = sns.countplot(x="label", data=df,
                   facecolor=(0, 0, 0, 0),
                   linewidth=5,
                   edgecolor=sns.color_palette("dark", 3),
                   order = df['label'].value_counts().index)
```



Data Prep

In [135…

```
y = df['label']
X = df.drop(columns = ['label'])
```

Scale Data

In [136…

```
# Conversion to float
X = X.astype('float32')

# Normalization
X = X/255.0
```

Flatten Data

https://thedatafrog.com/en/articles/handwritten-digit-recognition-scikit-learn/

In [137…

```
X = X.to_numpy().reshape((len(X), -1))
X.shape
```

Out[137…

```
(42000, 784)
```

Split Data

In [138…

```
# split data in to training and test data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=4
```

Neural Networks

Sequential Model

https://hackernoon.com/how-to-perform-mnist-digit-recognition-with-a-multi-layer-neural-network-xn223td8

In [139…

```
import tensorflow as tf
import keras
from keras.models import Sequential
from keras.layers import Flatten, Dense
```

Model 1: 2 Layers, 50 Nodes

In [140…

```
# creating model (2 layers, 50 nodes)
model1 = Sequential()
hidden_layer1 = Dense(50, activation='relu')
model1.add(hidden_layer1)
hidden_layer2 = Dense(50, activation='relu')
model1.add(hidden_layer2)
output_layer=Dense(10, activation='softmax')
model1.add(output_layer)
```

In [141…

```
# compiling the sequential model
model1.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

In [142…

```
# early stopping callback to interrupt training when it measures no progress on the val
early_stopping_cb = keras.callbacks.EarlyStopping(patience=20)
```

In [143…

```
start = datetime.now()
model1.fit(X_train, y_train, epochs=100,
          validation_data=(X_test, y_test),
          callbacks=[early_stopping_cb])
end = datetime.now()
```

```
Epoch 1/100
1050/1050 [==============================] - 1s 1ms/step - loss: 0.3638 - accuracy: 0.89
54 - val_loss: 0.2054 - val_accuracy: 0.9404
Epoch 2/100
1050/1050 [==============================] - 1s 1ms/step - loss: 0.1683 - accuracy: 0.95
05 - val_loss: 0.1615 - val_accuracy: 0.9519
Epoch 3/100
1050/1050 [==============================] - 1s 1ms/step - loss: 0.1237 - accuracy: 0.96
30 - val_loss: 0.1318 - val_accuracy: 0.9608
Epoch 4/100
1050/1050 [==============================] - 1s 1ms/step - loss: 0.0966 - accuracy: 0.97
00 - val_loss: 0.1337 - val_accuracy: 0.9594
```

```
Epoch 5/100
1050/1050 [==============================] - 1s 1ms/step - loss: 0.0796 - accuracy: 0.97
48 - val_loss: 0.1230 - val_accuracy: 0.9623
Epoch 6/100
1050/1050 [==============================] - 1s 1ms/step - loss: 0.0651 - accuracy: 0.97
99 - val_loss: 0.1392 - val_accuracy: 0.9605
Epoch 7/100
1050/1050 [==============================] - 1s 1ms/step - loss: 0.0575 - accuracy: 0.98
16 - val_loss: 0.1262 - val_accuracy: 0.9637
Epoch 8/100
1050/1050 [==============================] - 1s 1ms/step - loss: 0.0484 - accuracy: 0.98
48 - val_loss: 0.1288 - val_accuracy: 0.9638
Epoch 9/100
1050/1050 [==============================] - 1s 1ms/step - loss: 0.0408 - accuracy: 0.98
70 - val_loss: 0.1262 - val_accuracy: 0.9651
Epoch 10/100
1050/1050 [==============================] - 1s 1ms/step - loss: 0.0355 - accuracy: 0.98
76 - val_loss: 0.1226 - val_accuracy: 0.9669
Epoch 11/100
1050/1050 [==============================] - 2s 1ms/step - loss: 0.0298 - accuracy: 0.99
08 - val_loss: 0.1457 - val_accuracy: 0.9615
Epoch 12/100
1050/1050 [==============================] - 1s 1ms/step - loss: 0.0278 - accuracy: 0.99
05 - val_loss: 0.1328 - val_accuracy: 0.9648
Epoch 13/100
1050/1050 [==============================] - 1s 1ms/step - loss: 0.0231 - accuracy: 0.99
24 - val_loss: 0.1610 - val_accuracy: 0.9640
Epoch 14/100
1050/1050 [==============================] - 1s 1ms/step - loss: 0.0211 - accuracy: 0.99
31 - val_loss: 0.1361 - val_accuracy: 0.9679
Epoch 15/100
1050/1050 [==============================] - 1s 1ms/step - loss: 0.0169 - accuracy: 0.99
47 - val_loss: 0.1367 - val_accuracy: 0.9687
Epoch 16/100
1050/1050 [==============================] - 1s 1ms/step - loss: 0.0204 - accuracy: 0.99
31 - val_loss: 0.1475 - val_accuracy: 0.9685
Epoch 17/100
1050/1050 [==============================] - 1s 1ms/step - loss: 0.0169 - accuracy: 0.99
47 - val_loss: 0.1605 - val_accuracy: 0.9664
Epoch 18/100
1050/1050 [==============================] - 1s 1ms/step - loss: 0.0143 - accuracy: 0.99
52 - val_loss: 0.1741 - val_accuracy: 0.9654
Epoch 19/100
1050/1050 [==============================] - 1s 1ms/step - loss: 0.0157 - accuracy: 0.99
45 - val_loss: 0.1507 - val_accuracy: 0.9679
Epoch 20/100
1050/1050 [==============================] - 1s 1ms/step - loss: 0.0110 - accuracy: 0.99
66 - val_loss: 0.1595 - val_accuracy: 0.9685
Epoch 21/100
1050/1050 [==============================] - 1s 1ms/step - loss: 0.0141 - accuracy: 0.99
52 - val_loss: 0.1811 - val_accuracy: 0.9664
Epoch 22/100
1050/1050 [==============================] - 1s 1ms/step - loss: 0.0115 - accuracy: 0.99
60 - val_loss: 0.1691 - val_accuracy: 0.9671
Epoch 23/100
1050/1050 [==============================] - 1s 1ms/step - loss: 0.0120 - accuracy: 0.99
61 - val_loss: 0.1829 - val_accuracy: 0.9669
Epoch 24/100
1050/1050 [==============================] - 1s 1ms/step - loss: 0.0123 - accuracy: 0.99
57 - val_loss: 0.1892 - val_accuracy: 0.9681
Epoch 25/100
1050/1050 [==============================] - 1s 1ms/step - loss: 0.0075 - accuracy: 0.99
77 - val_loss: 0.1873 - val_accuracy: 0.9662
Epoch 26/100
1050/1050 [==============================] - 1s 1ms/step - loss: 0.0132 - accuracy: 0.99
```

```
57 - val_loss: 0.1859 - val_accuracy: 0.9677
Epoch 27/100
1050/1050 [==============================] - 1s 1ms/step - loss: 0.0112 - accuracy: 0.99
58 - val_loss: 0.1962 - val_accuracy: 0.9665
Epoch 28/100
1050/1050 [==============================] - 1s 1ms/step - loss: 0.0080 - accuracy: 0.99
71 - val_loss: 0.2016 - val_accuracy: 0.9652
Epoch 29/100
1050/1050 [==============================] - 1s 1ms/step - loss: 0.0100 - accuracy: 0.99
69 - val_loss: 0.1874 - val_accuracy: 0.9690
Epoch 30/100
1050/1050 [==============================] - 1s 1ms/step - loss: 0.0094 - accuracy: 0.99
69 - val_loss: 0.2034 - val_accuracy: 0.9674
```

In [144…

```
delta = end-start
print(delta) #4:21.933538
```

```
0:00:40.178178
```

In [145…

```
train_loss_and_acc = model1.evaluate(X_train, y_train, verbose=2)
print('Training Loss:', train_loss_and_acc[0])
print('Training Accuracy:', train_loss_and_acc[1])
```

```
1050/1050 - 1s - loss: 0.0054 - accuracy: 0.9982 - 906ms/epoch - 863us/step
Training Loss: 0.005433934275060892
Training Accuracy: 0.998244047164917
```

In [146…

```
test_loss_and_acc = model1.evaluate(X_test, y_test, verbose=2)
print('Test Loss:', test_loss_and_acc[0])
print('Test Accuracy:', test_loss_and_acc[1])
```

```
263/263 - 0s - loss: 0.2034 - accuracy: 0.9674 - 204ms/epoch - 777us/step
Test Loss: 0.20336014032363892
Test Accuracy: 0.967380940914154
```

In [147…

```
results = pd.DataFrame(columns = ['Model', 'Layers', 'Nodes', 'Time', 'Training Accurac

#eval
Model = 'Model 1 (Seq, 2, 50)'
Layers = 2
Nodes = 50
Time = delta.total_seconds()
Training_Accuracy = train_loss_and_acc[1]
Testing_Accuracy = test_loss_and_acc[1]
row = [Model, Layers, Nodes, Time, Training_Accuracy, Testing_Accuracy]
results = results.append(pd.DataFrame([row], columns=results.columns), ignore_index=Tru
results
```

Out[147…

	Model	Layers	Nodes	Time	Training Accuracy	Testing Accuracy
0	Model 1 (Seq, 2, 50)	2	50	40.178178	0.998244	0.967381

Model 2: 2 Layers, 100 Nodes

In [148…

```
# creating model (2 layers, 100 nodes)
model2 = Sequential()
hidden_layer1 = Dense(100, activation='relu')
```

```
model2.add(hidden_layer1)
hidden_layer2 = Dense(100, activation='relu')
model2.add(hidden_layer2)
output_layer=Dense(10, activation='softmax')
model2.add(output_layer)
```

In [149…

```
# compiling the sequential model
model2.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

In [150…

```
start = datetime.now()
model2.fit(X_train, y_train, epochs=100,
          validation_data=(X_test, y_test),
          callbacks=[early_stopping_cb])
end = datetime.now()
```

```
Epoch 1/100
1050/1050 [==============================] - 2s 2ms/step - loss: 0.3193 - accuracy: 0.90
61 - val_loss: 0.1877 - val_accuracy: 0.9442
Epoch 2/100
1050/1050 [==============================] - 2s 2ms/step - loss: 0.1364 - accuracy: 0.95
91 - val_loss: 0.1485 - val_accuracy: 0.9545
Epoch 3/100
1050/1050 [==============================] - 2s 2ms/step - loss: 0.0942 - accuracy: 0.97
07 - val_loss: 0.1118 - val_accuracy: 0.9640
Epoch 4/100
1050/1050 [==============================] - 2s 2ms/step - loss: 0.0673 - accuracy: 0.97
90 - val_loss: 0.1167 - val_accuracy: 0.9661
Epoch 5/100
1050/1050 [==============================] - 2s 2ms/step - loss: 0.0555 - accuracy: 0.98
22 - val_loss: 0.1304 - val_accuracy: 0.9605
Epoch 6/100
1050/1050 [==============================] - 2s 2ms/step - loss: 0.0421 - accuracy: 0.98
60 - val_loss: 0.1060 - val_accuracy: 0.9693
Epoch 7/100
1050/1050 [==============================] - 2s 2ms/step - loss: 0.0320 - accuracy: 0.98
96 - val_loss: 0.1144 - val_accuracy: 0.9696
Epoch 8/100
1050/1050 [==============================] - 2s 2ms/step - loss: 0.0287 - accuracy: 0.99
05 - val_loss: 0.1179 - val_accuracy: 0.9685
Epoch 9/100
1050/1050 [==============================] - 2s 2ms/step - loss: 0.0243 - accuracy: 0.99
26 - val_loss: 0.1219 - val_accuracy: 0.9710
Epoch 10/100
1050/1050 [==============================] - 2s 2ms/step - loss: 0.0207 - accuracy: 0.99
34 - val_loss: 0.1438 - val_accuracy: 0.9655
Epoch 11/100
1050/1050 [==============================] - 2s 2ms/step - loss: 0.0192 - accuracy: 0.99
38 - val_loss: 0.1082 - val_accuracy: 0.9735
Epoch 12/100
1050/1050 [==============================] - 2s 2ms/step - loss: 0.0135 - accuracy: 0.99
54 - val_loss: 0.1290 - val_accuracy: 0.9735
Epoch 13/100
1050/1050 [==============================] - 2s 2ms/step - loss: 0.0145 - accuracy: 0.99
51 - val_loss: 0.1363 - val_accuracy: 0.9701
Epoch 14/100
1050/1050 [==============================] - 2s 2ms/step - loss: 0.0154 - accuracy: 0.99
49 - val_loss: 0.1327 - val_accuracy: 0.9733
Epoch 15/100
```

```
1050/1050 [==============================] - 2s 1ms/step - loss: 0.0130 - accuracy: 0.99
60 - val_loss: 0.1223 - val_accuracy: 0.9739
Epoch 16/100
1050/1050 [==============================] - 1s 1ms/step - loss: 0.0120 - accuracy: 0.99
63 - val_loss: 0.1438 - val_accuracy: 0.9708
Epoch 17/100
1050/1050 [==============================] - 2s 2ms/step - loss: 0.0146 - accuracy: 0.99
51 - val_loss: 0.1400 - val_accuracy: 0.9736
Epoch 18/100
1050/1050 [==============================] - 2s 2ms/step - loss: 0.0101 - accuracy: 0.99
65 - val_loss: 0.1520 - val_accuracy: 0.9714
Epoch 19/100
1050/1050 [==============================] - 2s 2ms/step - loss: 0.0110 - accuracy: 0.99
64 - val_loss: 0.1425 - val_accuracy: 0.9748
Epoch 20/100
1050/1050 [==============================] - 2s 2ms/step - loss: 0.0087 - accuracy: 0.99
71 - val_loss: 0.1896 - val_accuracy: 0.9674
Epoch 21/100
1050/1050 [==============================] - 2s 2ms/step - loss: 0.0124 - accuracy: 0.99
59 - val_loss: 0.1773 - val_accuracy: 0.9714
Epoch 22/100
1050/1050 [==============================] - 1s 1ms/step - loss: 0.0073 - accuracy: 0.99
76 - val_loss: 0.1766 - val_accuracy: 0.9708
Epoch 23/100
1050/1050 [==============================] - 2s 1ms/step - loss: 0.0111 - accuracy: 0.99
65 - val_loss: 0.1903 - val_accuracy: 0.9699
Epoch 24/100
1050/1050 [==============================] - 1s 1ms/step - loss: 0.0097 - accuracy: 0.99
68 - val_loss: 0.1677 - val_accuracy: 0.9729
Epoch 25/100
1050/1050 [==============================] - 1s 1ms/step - loss: 0.0099 - accuracy: 0.99
67 - val_loss: 0.1702 - val_accuracy: 0.9719
Epoch 26/100
1050/1050 [==============================] - 1s 1ms/step - loss: 0.0091 - accuracy: 0.99
73 - val_loss: 0.1765 - val_accuracy: 0.9736
```

In [151…

```
delta = end-start
print(delta)
```

```
0:00:44.802751
```

In [152…

```
train_loss_and_acc = model2.evaluate(X_train, y_train, verbose=2)
print('Training Loss:', train_loss_and_acc[0])
print('Training Accuracy:', train_loss_and_acc[1])
```

```
1050/1050 - 1s - loss: 0.0034 - accuracy: 0.9987 - 893ms/epoch - 851us/step
Training Loss: 0.003429529257118702
Training Accuracy: 0.9987499713897705
```

In [153…

```
test_loss_and_acc = model2.evaluate(X_test, y_test, verbose=2)
print('Test Loss:', test_loss_and_acc[0])
print('Test Accuracy:', test_loss_and_acc[1])
```

```
263/263 - 0s - loss: 0.1765 - accuracy: 0.9736 - 203ms/epoch - 770us/step
Test Loss: 0.17647650837898254
Test Accuracy: 0.9735714197158813
```

In [154…

```
#eval
Model = 'Model 2 (Seq, 2, 100)'
Layers = 2
Nodes = 100
```

```
Time = delta.total_seconds()
Training_Accuracy = train_loss_and_acc[1]
Testing_Accuracy = test_loss_and_acc[1]
row = [Model, Layers, Nodes, Time, Training_Accuracy, Testing_Accuracy]
results = results.append(pd.DataFrame([row], columns=results.columns), ignore_index=Tru
results
```

Out[154…

	Model	Layers	Nodes	Time	Training Accuracy	Testing Accuracy
0	Model 1 (Seq, 2, 50)	2	50	40.178178	0.998244	0.967381
1	Model 2 (Seq, 2, 100)	2	100	44.802751	0.998750	0.973571

Model 3: 4 Layers, 50 Nodes

In [155…

```
# creating model (4 layers, 50 nodes)
model3 = Sequential()
hidden_layer1 = Dense(50, activation='relu')
model3.add(hidden_layer1)
hidden_layer2 = Dense(50, activation='relu')
model3.add(hidden_layer2)
hidden_layer3 = Dense(50, activation='relu')
model3.add(hidden_layer3)
hidden_layer4 = Dense(50, activation='relu')
model3.add(hidden_layer4)
output_layer=Dense(10, activation='softmax')
model3.add(output_layer)
```

In [156…

```
# compiling the sequential model
model3.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

In [157…

```
start = datetime.now()
model3.fit(X_train, y_train, epochs=100,
          validation_data=(X_test, y_test),
          callbacks=[early_stopping_cb])
end = datetime.now()
```

```
Epoch 1/100
1050/1050 [==============================] - 2s 1ms/step - loss: 0.3715 - accuracy: 0.88
71 - val_loss: 0.2024 - val_accuracy: 0.9390
Epoch 2/100
1050/1050 [==============================] - 1s 1ms/step - loss: 0.1600 - accuracy: 0.95
17 - val_loss: 0.1748 - val_accuracy: 0.9494
Epoch 3/100
1050/1050 [==============================] - 1s 1ms/step - loss: 0.1203 - accuracy: 0.96
34 - val_loss: 0.1411 - val_accuracy: 0.9596
Epoch 4/100
1050/1050 [==============================] - 2s 1ms/step - loss: 0.0958 - accuracy: 0.97
08 - val_loss: 0.1445 - val_accuracy: 0.9593
Epoch 5/100
1050/1050 [==============================] - 2s 2ms/step - loss: 0.0806 - accuracy: 0.97
42 - val_loss: 0.1821 - val_accuracy: 0.9499
Epoch 6/100
1050/1050 [==============================] - 1s 1ms/step - loss: 0.0705 - accuracy: 0.97
72 - val_loss: 0.1148 - val_accuracy: 0.9669
```

```
Epoch 7/100
1050/1050 [==============================] - 2s 2ms/step - loss: 0.0592 - accuracy: 0.98
12 - val_loss: 0.1538 - val_accuracy: 0.9586
Epoch 8/100
1050/1050 [==============================] - 2s 2ms/step - loss: 0.0522 - accuracy: 0.98
32 - val_loss: 0.1252 - val_accuracy: 0.9658
Epoch 9/100
1050/1050 [==============================] - 2s 1ms/step - loss: 0.0464 - accuracy: 0.98
43 - val_loss: 0.1461 - val_accuracy: 0.9638
Epoch 10/100
1050/1050 [==============================] - 1s 1ms/step - loss: 0.0423 - accuracy: 0.98
55 - val_loss: 0.1377 - val_accuracy: 0.9656
Epoch 11/100
1050/1050 [==============================] - 1s 1ms/step - loss: 0.0395 - accuracy: 0.98
69 - val_loss: 0.1393 - val_accuracy: 0.9663
Epoch 12/100
1050/1050 [==============================] - 1s 1ms/step - loss: 0.0363 - accuracy: 0.98
81 - val_loss: 0.1535 - val_accuracy: 0.9629
Epoch 13/100
1050/1050 [==============================] - 1s 1ms/step - loss: 0.0287 - accuracy: 0.99
00 - val_loss: 0.1487 - val_accuracy: 0.9657
Epoch 14/100
1050/1050 [==============================] - 1s 1ms/step - loss: 0.0287 - accuracy: 0.98
99 - val_loss: 0.1540 - val_accuracy: 0.9662
Epoch 15/100
1050/1050 [==============================] - 2s 1ms/step - loss: 0.0284 - accuracy: 0.99
05 - val_loss: 0.1758 - val_accuracy: 0.9640
Epoch 16/100
1050/1050 [==============================] - 2s 1ms/step - loss: 0.0249 - accuracy: 0.99
15 - val_loss: 0.1579 - val_accuracy: 0.9699
Epoch 17/100
1050/1050 [==============================] - 2s 2ms/step - loss: 0.0219 - accuracy: 0.99
29 - val_loss: 0.1915 - val_accuracy: 0.9646
Epoch 18/100
1050/1050 [==============================] - 2s 2ms/step - loss: 0.0237 - accuracy: 0.99
24 - val_loss: 0.1543 - val_accuracy: 0.9705
Epoch 19/100
1050/1050 [==============================] - 2s 2ms/step - loss: 0.0204 - accuracy: 0.99
29 - val_loss: 0.1666 - val_accuracy: 0.9661
Epoch 20/100
1050/1050 [==============================] - 2s 1ms/step - loss: 0.0207 - accuracy: 0.99
32 - val_loss: 0.1862 - val_accuracy: 0.9654
Epoch 21/100
1050/1050 [==============================] - 2s 1ms/step - loss: 0.0220 - accuracy: 0.99
26 - val_loss: 0.2063 - val_accuracy: 0.9612
Epoch 22/100
1050/1050 [==============================] - 2s 2ms/step - loss: 0.0196 - accuracy: 0.99
35 - val_loss: 0.1689 - val_accuracy: 0.9683
Epoch 23/100
1050/1050 [==============================] - 2s 1ms/step - loss: 0.0181 - accuracy: 0.99
43 - val_loss: 0.1778 - val_accuracy: 0.9679
Epoch 24/100
1050/1050 [==============================] - 1s 1ms/step - loss: 0.0176 - accuracy: 0.99
46 - val_loss: 0.1710 - val_accuracy: 0.9694
Epoch 25/100
1050/1050 [==============================] - 1s 1ms/step - loss: 0.0159 - accuracy: 0.99
47 - val_loss: 0.1883 - val_accuracy: 0.9671
Epoch 26/100
1050/1050 [==============================] - 1s 1ms/step - loss: 0.0145 - accuracy: 0.99
50 - val_loss: 0.2159 - val_accuracy: 0.9639
```

In [158…

```
delta = end-start
print(delta)
```

```
0:00:40.209974
```

In [159...

```
train_loss_and_acc = model3.evaluate(X_train, y_train, verbose=2)
print('Training Loss:', train_loss_and_acc[0])
print('Training Accuracy:', train_loss_and_acc[1])
```

```
1050/1050 - 1s - loss: 0.0254 - accuracy: 0.9922 - 921ms/epoch - 877us/step
Training Loss: 0.025426795706152916
Training Accuracy: 0.9922321438789368
```

In [160...

```
test_loss_and_acc = model3.evaluate(X_test, y_test, verbose=2)
print('Test Loss:', test_loss_and_acc[0])
print('Test Accuracy:', test_loss_and_acc[1])
```

```
263/263 - 0s - loss: 0.2159 - accuracy: 0.9639 - 206ms/epoch - 783us/step
Test Loss: 0.215900257229805
Test Accuracy: 0.9639285802841187
```

In [161...

```
#eval
Model = 'Model 3 (Seq, 4, 50)'
Layers = 4
Nodes = 50
Time = delta.total_seconds()
Training_Accuracy = train_loss_and_acc[1]
Testing_Accuracy = test_loss_and_acc[1]
row = [Model, Layers, Nodes, Time, Training_Accuracy, Testing_Accuracy]
results = results.append(pd.DataFrame([row], columns=results.columns), ignore_index=Tru
results
```

Out[161...

	Model	Layers	Nodes	Time	Training Accuracy	Testing Accuracy
0	Model 1 (Seq, 2, 50)	2	50	40.178178	0.998244	0.967381
1	Model 2 (Seq, 2, 100)	2	100	44.802751	0.998750	0.973571
2	Model 3 (Seq, 4, 50)	4	50	40.209974	0.992232	0.963929

Model 4: 4 Layers, 100 Nodes

In [162...

```
# creating model (4 layers, 100 nodes)
model4 = Sequential()
hidden_layer1 = Dense(100, activation='relu')
model4.add(hidden_layer1)
hidden_layer2 = Dense(100, activation='relu')
model4.add(hidden_layer2)
hidden_layer3 = Dense(100, activation='relu')
model4.add(hidden_layer3)
hidden_layer4 = Dense(100, activation='relu')
model4.add(hidden_layer4)
output_layer=Dense(10, activation='softmax')
model4.add(output_layer)
```

In [163...

```
# compiling the sequential model
model4.compile(optimizer='adam',
               loss='sparse_categorical_crossentropy',
               metrics=['accuracy'])
```

In [164…

```
start = datetime.now()
model4.fit(X_train, y_train, epochs=100,
          validation_data=(X_test, y_test),
          callbacks=[early_stopping_cb])
end = datetime.now()
```

```
Epoch 1/100
1050/1050 [==============================] - 2s 2ms/step - loss: 0.3174 - accuracy: 0.90
38 - val_loss: 0.1676 - val_accuracy: 0.9496
Epoch 2/100
1050/1050 [==============================] - 2s 2ms/step - loss: 0.1311 - accuracy: 0.95
95 - val_loss: 0.1218 - val_accuracy: 0.9632
Epoch 3/100
1050/1050 [==============================] - 2s 2ms/step - loss: 0.0912 - accuracy: 0.97
23 - val_loss: 0.1551 - val_accuracy: 0.9539
Epoch 4/100
1050/1050 [==============================] - 2s 2ms/step - loss: 0.0750 - accuracy: 0.97
62 - val_loss: 0.1083 - val_accuracy: 0.9660
Epoch 5/100
1050/1050 [==============================] - 2s 2ms/step - loss: 0.0611 - accuracy: 0.98
07 - val_loss: 0.1043 - val_accuracy: 0.9688
Epoch 6/100
1050/1050 [==============================] - 2s 2ms/step - loss: 0.0480 - accuracy: 0.98
46 - val_loss: 0.1023 - val_accuracy: 0.9710
Epoch 7/100
1050/1050 [==============================] - 2s 2ms/step - loss: 0.0429 - accuracy: 0.98
70 - val_loss: 0.1130 - val_accuracy: 0.9676
Epoch 8/100
1050/1050 [==============================] - 2s 2ms/step - loss: 0.0375 - accuracy: 0.98
81 - val_loss: 0.1166 - val_accuracy: 0.9693
Epoch 9/100
1050/1050 [==============================] - 2s 2ms/step - loss: 0.0345 - accuracy: 0.98
89 - val_loss: 0.1237 - val_accuracy: 0.9699
Epoch 10/100
1050/1050 [==============================] - 2s 2ms/step - loss: 0.0326 - accuracy: 0.98
93 - val_loss: 0.1204 - val_accuracy: 0.9696
Epoch 11/100
1050/1050 [==============================] - 2s 2ms/step - loss: 0.0258 - accuracy: 0.99
23 - val_loss: 0.1327 - val_accuracy: 0.9727
Epoch 12/100
1050/1050 [==============================] - 2s 2ms/step - loss: 0.0254 - accuracy: 0.99
24 - val_loss: 0.1442 - val_accuracy: 0.9695
Epoch 13/100
1050/1050 [==============================] - 3s 3ms/step - loss: 0.0244 - accuracy: 0.99
25 - val_loss: 0.1122 - val_accuracy: 0.9742
Epoch 14/100
1050/1050 [==============================] - 2s 2ms/step - loss: 0.0239 - accuracy: 0.99
30 - val_loss: 0.1270 - val_accuracy: 0.9704
Epoch 15/100
1050/1050 [==============================] - 2s 2ms/step - loss: 0.0197 - accuracy: 0.99
45 - val_loss: 0.1637 - val_accuracy: 0.9682
Epoch 16/100
1050/1050 [==============================] - 2s 2ms/step - loss: 0.0158 - accuracy: 0.99
50 - val_loss: 0.1419 - val_accuracy: 0.9714
Epoch 17/100
1050/1050 [==============================] - 2s 2ms/step - loss: 0.0259 - accuracy: 0.99
23 - val_loss: 0.1369 - val_accuracy: 0.9701
Epoch 18/100
1050/1050 [==============================] - 2s 2ms/step - loss: 0.0146 - accuracy: 0.99
57 - val_loss: 0.1480 - val_accuracy: 0.9706
Epoch 19/100
1050/1050 [==============================] - 2s 2ms/step - loss: 0.0136 - accuracy: 0.99
```

```
60 - val_loss: 0.1524 - val_accuracy: 0.9725
Epoch 20/100
1050/1050 [==============================] - 2s 2ms/step - loss: 0.0157 - accuracy: 0.99
48 - val_loss: 0.1641 - val_accuracy: 0.9693
Epoch 21/100
1050/1050 [==============================] - 2s 2ms/step - loss: 0.0180 - accuracy: 0.99
46 - val_loss: 0.1445 - val_accuracy: 0.9689
Epoch 22/100
1050/1050 [==============================] - 2s 2ms/step - loss: 0.0149 - accuracy: 0.99
56 - val_loss: 0.1193 - val_accuracy: 0.9762
Epoch 23/100
1050/1050 [==============================] - 2s 2ms/step - loss: 0.0115 - accuracy: 0.99
71 - val_loss: 0.1505 - val_accuracy: 0.9717
Epoch 24/100
1050/1050 [==============================] - 2s 2ms/step - loss: 0.0175 - accuracy: 0.99
48 - val_loss: 0.1671 - val_accuracy: 0.9724
Epoch 25/100
1050/1050 [==============================] - 2s 2ms/step - loss: 0.0128 - accuracy: 0.99
59 - val_loss: 0.1426 - val_accuracy: 0.9739
Epoch 26/100
1050/1050 [==============================] - 2s 2ms/step - loss: 0.0139 - accuracy: 0.99
65 - val_loss: 0.1502 - val_accuracy: 0.9720
```

In [165…

```
delta = end-start
print(delta)
```

```
0:00:48.241655
```

In [166…

```
train_loss_and_acc = model4.evaluate(X_train, y_train, verbose=2)
print('Training Loss:', train_loss_and_acc[0])
print('Training Accuracy:', train_loss_and_acc[1])
```

```
1050/1050 - 1s - loss: 0.0113 - accuracy: 0.9963 - 1s/epoch - 994us/step
Training Loss: 0.01132137794047594
Training Accuracy: 0.996279776096344
```

In [167…

```
test_loss_and_acc = model4.evaluate(X_test, y_test, verbose=2)
print('Test Loss:', test_loss_and_acc[0])
print('Test Accuracy:', test_loss_and_acc[1])
```

```
263/263 - 0s - loss: 0.1502 - accuracy: 0.9720 - 261ms/epoch - 992us/step
Test Loss: 0.1502307504415512
Test Accuracy: 0.9720237851142883
```

In [168…

```
#eval
Model = 'Model 4 (Seq, 4, 100)'
Layers = 4
Nodes = 100
Time = delta.total_seconds()
Training_Accuracy = train_loss_and_acc[1]
Testing_Accuracy = test_loss_and_acc[1]
row = [Model, Layers, Nodes, Time, Training_Accuracy, Testing_Accuracy]
results = results.append(pd.DataFrame([row], columns=results.columns), ignore_index=Tru
results
```

Out[168…

	Model	Layers	Nodes	Time	Training Accuracy	Testing Accuracy
0	Model 1 (Seq, 2, 50)	2	50	40.178178	0.998244	0.967381

	Model	Layers	Nodes	Time	Training Accuracy	Testing Accuracy
1	Model 2 (Seq, 2, 100)	2	100	44.802751	0.998750	0.973571
2	Model 3 (Seq, 4, 50)	4	50	40.209974	0.992232	0.963929
3	Model 4 (Seq, 4, 100)	4	100	48.241655	0.996280	0.972024

MLP Classfier Model

In [169…

```
import warnings
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_openml
from sklearn.exceptions import ConvergenceWarning
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
```

In [170…

```
from sklearn.neural_network import MLPClassifier
layer_sizes_10x1 = (10,)
layer_sizes_50x1 = (50,)
layer_sizes_100x1 = (100,)
layer_sizes_10x3 = (10,10,10)
layer_sizes_50x3 = (50,50,50)
layer_sizes_100x3 = (100,100,100)
layer_sizes_10x5 = (10,10,10,10,10)
layer_sizes_50x5 = (50,50,50,50,50)
layer_sizes_100x5 = (100,100,100,100,100)
```

1 Layer

10 Nodes

In [171…

```
mlp1 = MLPClassifier(
    hidden_layer_sizes=layer_sizes_10x1,
    max_iter=100,
    alpha=1e-4,
    solver="sgd",
    verbose=10,
    random_state=42,
    learning_rate_init=0.2,
)

start = datetime.now()
mlp1.fit(X_train, y_train)
end = datetime.now()
print("Training set score: %f" % mlp1.score(X_train, y_train))
print("Test set score: %f" % mlp1.score(X_test, y_test))
```

```
Iteration 1, loss = 0.46799789
Iteration 2, loss = 0.28337209
Iteration 3, loss = 0.26004510
Iteration 4, loss = 0.24543112
Iteration 5, loss = 0.23922180
Iteration 6, loss = 0.23108309
Iteration 7, loss = 0.23014608
Iteration 8, loss = 0.21869644
Iteration 9, loss = 0.21827797
Iteration 10, loss = 0.21678182
Iteration 11, loss = 0.21008014
Iteration 12, loss = 0.20553513
Iteration 13, loss = 0.20457491
Iteration 14, loss = 0.19738307
Iteration 15, loss = 0.19873571
Iteration 16, loss = 0.19578736
Iteration 17, loss = 0.19436931
Iteration 18, loss = 0.19221754
Iteration 19, loss = 0.18757736
Iteration 20, loss = 0.18641514
Iteration 21, loss = 0.18723098
Iteration 22, loss = 0.18257645
Iteration 23, loss = 0.18053602
Iteration 24, loss = 0.18286422
Iteration 25, loss = 0.17846300
Iteration 26, loss = 0.17725075
Iteration 27, loss = 0.17867561
Iteration 28, loss = 0.17128354
Iteration 29, loss = 0.17497341
Iteration 30, loss = 0.16668537
Iteration 31, loss = 0.17080538
Iteration 32, loss = 0.16887190
Iteration 33, loss = 0.16410179
Iteration 34, loss = 0.16463814
Iteration 35, loss = 0.16336772
Iteration 36, loss = 0.16395531
Iteration 37, loss = 0.16170666
Iteration 38, loss = 0.16577991
Iteration 39, loss = 0.16077357
Iteration 40, loss = 0.15725368
Iteration 41, loss = 0.15527804
Iteration 42, loss = 0.15517271
Iteration 43, loss = 0.15480795
Iteration 44, loss = 0.15452442
Iteration 45, loss = 0.15767771
Iteration 46, loss = 0.15563256
Iteration 47, loss = 0.15388912
Iteration 48, loss = 0.14967470
Iteration 49, loss = 0.15146286
Iteration 50, loss = 0.15292202
Iteration 51, loss = 0.14993475
Iteration 52, loss = 0.14900185
Iteration 53, loss = 0.14653540
Iteration 54, loss = 0.15025065
Iteration 55, loss = 0.14929180
Iteration 56, loss = 0.14675219
Iteration 57, loss = 0.15061054
Iteration 58, loss = 0.14927762
Iteration 59, loss = 0.15083743
Iteration 60, loss = 0.14438355
Iteration 61, loss = 0.14205400
Iteration 62, loss = 0.14743674
Iteration 63, loss = 0.13996177
Iteration 64, loss = 0.14431671
Iteration 65, loss = 0.14527572
Iteration 66, loss = 0.14160146
Iteration 67, loss = 0.14380693
Iteration 68, loss = 0.14414260
Iteration 69, loss = 0.14214776
Iteration 70, loss = 0.14230648
Iteration 71, loss = 0.14140478
Iteration 72, loss = 0.13778057
Iteration 73, loss = 0.13949324
Iteration 74, loss = 0.14034866
Iteration 75, loss = 0.13802927
Iteration 76, loss = 0.14341789
Iteration 77, loss = 0.13859883
Iteration 78, loss = 0.13650487
Iteration 79, loss = 0.13386188
Iteration 80, loss = 0.13689800
Iteration 81, loss = 0.13715766
Iteration 82, loss = 0.13643560
Iteration 83, loss = 0.14068233
Iteration 84, loss = 0.13448595
Iteration 85, loss = 0.13549297
Iteration 86, loss = 0.13506786
Iteration 87, loss = 0.13726068
Iteration 88, loss = 0.13295125
Iteration 89, loss = 0.13365700
Iteration 90, loss = 0.13551863
Iteration 91, loss = 0.13414394
Iteration 92, loss = 0.13262250
Iteration 93, loss = 0.13323267
Iteration 94, loss = 0.13416315
Iteration 95, loss = 0.13160296
Iteration 96, loss = 0.13289975
Iteration 97, loss = 0.13365570
Iteration 98, loss = 0.13217884
Iteration 99, loss = 0.13416245
Iteration 100, loss = 0.13264885
Training set score: 0.961786
Test set score: 0.926071
```

```
C:\Users\sjue\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.
py:702: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (100) reached and t
he optimization hasn't converged yet.
  warnings.warn(
```

In [172…

```
delta = end-start
print(delta)
```

```
0:00:25.288042
```

In [173…

```
mlp1_test_pred = mlp1.score(X_test, y_test)
mlp1_train_pred = mlp1.score(X_train, y_train)
```

In [174…

```
#eval
Model = 'Model 5 (MLP, 1, 10)'
Layers = 1
Nodes = 10
Time = delta.total_seconds()
Training_Accuracy = mlp1_test_pred
Testing_Accuracy = mlp1_train_pred
row = [Model, Layers, Nodes, Time, Training_Accuracy, Testing_Accuracy]
results = results.append(pd.DataFrame([row], columns=results.columns), ignore_index=Tru
results
```

Out[174…

	Model	Layers	Nodes	Time	Training Accuracy	Testing Accuracy
0	Model 1 (Seq, 2, 50)	2	50	40.178178	0.998244	0.967381
1	Model 2 (Seq, 2, 100)	2	100	44.802751	0.998750	0.973571

	Model	Layers	Nodes	Time	Training Accuracy	Testing Accuracy
2	Model 3 (Seq, 4, 50)	4	50	40.209974	0.992232	0.963929
3	Model 4 (Seq, 4, 100)	4	100	48.241655	0.996280	0.972024
4	Model 5 (MLP, 1, 10)	1	10	25.288042	0.926071	0.961786

50 Nodes

In [175…

```
mlp2 = MLPClassifier(
    hidden_layer_sizes=layer_sizes_50x1,
    max_iter=100,
    alpha=1e-4,
    solver="sgd",
    verbose=10,
    random_state=42,
    learning_rate_init=0.2,
)

start = datetime.now()
mlp2.fit(X_train, y_train)
end = datetime.now()

print("Training set score: %f" % mlp2.score(X_train, y_train))
print("Test set score: %f" % mlp2.score(X_test, y_test))
```

```
Iteration 1, loss = 0.33812117
Iteration 2, loss = 0.13744666
Iteration 3, loss = 0.10501313
Iteration 4, loss = 0.08500181
Iteration 5, loss = 0.06995283
Iteration 6, loss = 0.06114768
Iteration 7, loss = 0.05183904
Iteration 8, loss = 0.04595707
Iteration 9, loss = 0.03593902
Iteration 10, loss = 0.03226649
Iteration 11, loss = 0.02851820
Iteration 12, loss = 0.02372226
Iteration 13, loss = 0.02137845
Iteration 14, loss = 0.01935074
Iteration 15, loss = 0.01543476
Iteration 16, loss = 0.01155923
Iteration 17, loss = 0.01227434
Iteration 18, loss = 0.00864990
Iteration 19, loss = 0.00715568
Iteration 20, loss = 0.00479896
Iteration 21, loss = 0.00317698
Iteration 22, loss = 0.00211373
Iteration 23, loss = 0.00177557
Iteration 24, loss = 0.00151011
Iteration 25, loss = 0.00139539
Iteration 26, loss = 0.00129437
Iteration 27, loss = 0.00125639
Iteration 28, loss = 0.00119228
Iteration 29, loss = 0.00114205
Iteration 30, loss = 0.00110769
Iteration 31, loss = 0.00105695
Iteration 32, loss = 0.00102615
Iteration 33, loss = 0.00100543
Iteration 34, loss = 0.00097124
Iteration 35, loss = 0.00094304
Iteration 36, loss = 0.00091842
Iteration 37, loss = 0.00089950
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stoppin
g.
Training set score: 1.000000
Test set score: 0.970476
```

In [176…

```
delta = end-start
print(delta)
```

```
0:00:14.225653
```

In [177…

```
mlp2_test_pred = mlp2.score(X_test, y_test)
mlp2_train_pred = mlp2.score(X_train, y_train)
```

In [178…

```
#eval
Model = 'Model 6 (MLP, 1, 50)'
Layers = 1
Nodes = 50
Time = delta.total_seconds()
Training_Accuracy = mlp2_test_pred
Testing_Accuracy = mlp2_train_pred
row = [Model, Layers, Nodes, Time, Training_Accuracy, Testing_Accuracy]
results = results.append(pd.DataFrame([row], columns=results.columns), ignore_index=Tru
results
```

Out[178…

	Model	Layers	Nodes	Time	Training Accuracy	Testing Accuracy
0	Model 1 (Seq, 2, 50)	2	50	40.178178	0.998244	0.967381
1	Model 2 (Seq, 2, 100)	2	100	44.802751	0.998750	0.973571
2	Model 3 (Seq, 4, 50)	4	50	40.209974	0.992232	0.963929
3	Model 4 (Seq, 4, 100)	4	100	48.241655	0.996280	0.972024
4	Model 5 (MLP, 1, 10)	1	10	25.288042	0.926071	0.961786
5	Model 6 (MLP, 1, 50)	1	50	14.225653	0.970476	1.000000

100 Nodes

In [179…

```
mlp3 = MLPClassifier(
    hidden_layer_sizes=layer_sizes_100x1,
    max_iter=100,
    alpha=1e-4,
    solver="sgd",
    verbose=10,
    random_state=42,
    learning_rate_init=0.2,
)

start = datetime.now()
mlp3.fit(X_train, y_train)
end = datetime.now()
```

```
print("Training set score: %f" % mlp3.score(X_train, y_train))
print("Test set score: %f" % mlp3.score(X_test, y_test))
```

```
Iteration 1, loss = 0.32167365
Iteration 2, loss = 0.12200411
Iteration 3, loss = 0.08494004
Iteration 4, loss = 0.06455943
Iteration 5, loss = 0.04928552
Iteration 6, loss = 0.03773648
Iteration 7, loss = 0.03102117
Iteration 8, loss = 0.02460724
Iteration 9, loss = 0.01995751
Iteration 10, loss = 0.01349303
Iteration 11, loss = 0.01251578
Iteration 12, loss = 0.00956929
Iteration 13, loss = 0.00542790
Iteration 14, loss = 0.00443356
Iteration 15, loss = 0.00293126
Iteration 16, loss = 0.00219065
Iteration 17, loss = 0.00186604
Iteration 18, loss = 0.00166548
Iteration 19, loss = 0.00151996
Iteration 20, loss = 0.00141091
Iteration 21, loss = 0.00134208
Iteration 22, loss = 0.00129608
Iteration 23, loss = 0.00118608
Iteration 24, loss = 0.00112978
Iteration 25, loss = 0.00109271
Iteration 26, loss = 0.00104337
Iteration 27, loss = 0.00100573
Iteration 28, loss = 0.00097580
Iteration 29, loss = 0.00095161
Iteration 30, loss = 0.00091180
Iteration 31, loss = 0.00089629
Iteration 32, loss = 0.00086426
Iteration 33, loss = 0.00084528
Iteration 34, loss = 0.00082363
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stoppin
g.
Training set score: 1.000000
Test set score: 0.973690
```

In [180…

```
delta = end-start
print(delta)
```

```
0:00:19.180962
```

In [181…

```
mlp3_test_pred = mlp3.score(X_test, y_test)
mlp3_train_pred = mlp3.score(X_train, y_train)
```

In [182…

```
#eval
Model = 'Model 7 (MLP, 1, 100)'
Layers = 1
Nodes = 100
Time = delta.total_seconds()
Training_Accuracy = mlp3_test_pred
Testing_Accuracy = mlp3_train_pred
row = [Model, Layers, Nodes, Time, Training_Accuracy, Testing_Accuracy]
```

```
results = results.append(pd.DataFrame([row], columns=results.columns), ignore_index=Tru
results
```

Out[182…

	Model	Layers	Nodes	Time	Training Accuracy	Testing Accuracy
0	Model 1 (Seq, 2, 50)	2	50	40.178178	0.998244	0.967381
1	Model 2 (Seq, 2, 100)	2	100	44.802751	0.998750	0.973571
2	Model 3 (Seq, 4, 50)	4	50	40.209974	0.992232	0.963929
3	Model 4 (Seq, 4, 100)	4	100	48.241655	0.996280	0.972024
4	Model 5 (MLP, 1, 10)	1	10	25.288042	0.926071	0.961786
5	Model 6 (MLP, 1, 50)	1	50	14.225653	0.970476	1.000000
6	Model 7 (MLP, 1, 100)	1	100	19.180962	0.973690	1.000000

3 Layers

10 Nodes

In [183…

```
mlp4 = MLPClassifier(
    hidden_layer_sizes=layer_sizes_10x3,
    max_iter=100,
    alpha=1e-4,
    solver="sgd",
    verbose=10,
    random_state=42,
    learning_rate_init=0.2,
)

start = datetime.now()
mlp4.fit(X_train, y_train)
end = datetime.now()

print("Training set score: %f" % mlp4.score(X_train, y_train))
print("Test set score: %f" % mlp4.score(X_test, y_test))
```

```
Iteration 1, loss = 0.95200348
Iteration 2, loss = 0.49537485
Iteration 3, loss = 0.42375075
Iteration 4, loss = 0.36630830
Iteration 5, loss = 0.34513270
Iteration 6, loss = 0.32719876
Iteration 7, loss = 0.31331080
Iteration 8, loss = 0.31718798
Iteration 9, loss = 0.30375754
Iteration 10, loss = 0.29615480
Iteration 11, loss = 0.29065328
Iteration 12, loss = 0.28912365
Iteration 13, loss = 0.28349875
Iteration 14, loss = 0.28413518
Iteration 15, loss = 0.27920261
Iteration 16, loss = 0.27726201
Iteration 17, loss = 0.26929211
Iteration 18, loss = 0.27078912
Iteration 19, loss = 0.26838341
Iteration 20, loss = 0.26876391
Iteration 21, loss = 0.26670843
Iteration 22, loss = 0.26207022
Iteration 23, loss = 0.26472387
Iteration 24, loss = 0.25967084
Iteration 25, loss = 0.25990658
Iteration 26, loss = 0.26050995
Iteration 27, loss = 0.25320391
Iteration 28, loss = 0.25541233
Iteration 29, loss = 0.25730853
Iteration 30, loss = 0.25246449
Iteration 31, loss = 0.24903140
Iteration 32, loss = 0.24509865
Iteration 33, loss = 0.24133469
Iteration 34, loss = 0.24180556
Iteration 35, loss = 0.23978958
Iteration 36, loss = 0.23895530
Iteration 37, loss = 0.23489046
Iteration 38, loss = 0.23072863
Iteration 39, loss = 0.22581056
Iteration 40, loss = 0.23381132
Iteration 41, loss = 0.22169284
Iteration 42, loss = 0.22709626
Iteration 43, loss = 0.22327762
Iteration 44, loss = 0.22147225
Iteration 45, loss = 0.21730356
Iteration 46, loss = 0.21627244
Iteration 47, loss = 0.21637593
Iteration 48, loss = 0.21416641
Iteration 49, loss = 0.21047881
Iteration 50, loss = 0.21779899
Iteration 51, loss = 0.21671118
Iteration 52, loss = 0.21425793
Iteration 53, loss = 0.21136803
Iteration 54, loss = 0.20835240
Iteration 55, loss = 0.20721170
Iteration 56, loss = 0.20844192
Iteration 57, loss = 0.20772467
Iteration 58, loss = 0.20413349
Iteration 59, loss = 0.20510409
Iteration 60, loss = 0.20453301
Iteration 61, loss = 0.20619270
Iteration 62, loss = 0.20376114
Iteration 63, loss = 0.20181528
Iteration 64, loss = 0.20847558
Iteration 65, loss = 0.20119562
Iteration 66, loss = 0.20543950
Iteration 67, loss = 0.19740208
Iteration 68, loss = 0.19886402
Iteration 69, loss = 0.20305287
Iteration 70, loss = 0.19295018
Iteration 71, loss = 0.19509171
Iteration 72, loss = 0.19782918
Iteration 73, loss = 0.19588872
Iteration 74, loss = 0.19435749
Iteration 75, loss = 0.19423982
Iteration 76, loss = 0.19689540
Iteration 77, loss = 0.19312529
Iteration 78, loss = 0.19491269
Iteration 79, loss = 0.19691840
Iteration 80, loss = 0.19529473
Iteration 81, loss = 0.19397768
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stoppin
g.
Training set score: 0.948423
Test set score: 0.922619
```

In [184…

```
delta = end-start
print(delta)
```

```
0:00:20.080916
```

In [185…

```
mlp4_test_pred = mlp4.score(X_test, y_test)
mlp4_train_pred = mlp4.score(X_train, y_train)
```

In [186…

```
#eval
Model = 'Model 8 (MLP, 3, 10)'
Layers = 3
Nodes = 10
Time = delta.total_seconds()
Training_Accuracy = mlp4_test_pred
Testing_Accuracy = mlp4_train_pred
row = [Model, Layers, Nodes, Time, Training_Accuracy, Testing_Accuracy]
results = results.append(pd.DataFrame([row], columns=results.columns), ignore_index=Tru
results
```

Out[186…

	Model	Layers	Nodes	Time	Training Accuracy	Testing Accuracy
0	Model 1 (Seq, 2, 50)	2	50	40.178178	0.998244	0.967381
1	Model 2 (Seq, 2, 100)	2	100	44.802751	0.998750	0.973571
2	Model 3 (Seq, 4, 50)	4	50	40.209974	0.992232	0.963929
3	Model 4 (Seq, 4, 100)	4	100	48.241655	0.996280	0.972024
4	Model 5 (MLP, 1, 10)	1	10	25.288042	0.926071	0.961786
5	Model 6 (MLP, 1, 50)	1	50	14.225653	0.970476	1.000000
6	Model 7 (MLP, 1, 100)	1	100	19.180962	0.973690	1.000000
7	Model 8 (MLP, 3, 10)	3	10	20.080916	0.922619	0.948423

50 Nodes

In [187…

```
mlp5 = MLPClassifier(
    hidden_layer_sizes=layer_sizes_50x3,
    max_iter=100,
    alpha=1e-4,
    solver="sgd",
    verbose=10,
    random_state=42,
    learning_rate_init=0.2,
)

start = datetime.now()
mlp5.fit(X_train, y_train)
end = datetime.now()

print("Training set score: %f" % mlp5.score(X_train, y_train))
print("Test set score: %f" % mlp5.score(X_test, y_test))
```

```
Iteration 1, loss = 0.45871904
Iteration 2, loss = 0.17437994
Iteration 3, loss = 0.14009937
Iteration 4, loss = 0.12222580
Iteration 5, loss = 0.10187893
Iteration 6, loss = 0.09147139
Iteration 7, loss = 0.08225719
Iteration 8, loss = 0.07842556
Iteration 9, loss = 0.07434273
Iteration 10, loss = 0.06572201
Iteration 11, loss = 0.06337553
Iteration 12, loss = 0.06245802
Iteration 13, loss = 0.06310806
Iteration 14, loss = 0.05442029
Iteration 15, loss = 0.05853271
Iteration 16, loss = 0.04964991
Iteration 17, loss = 0.04539171
Iteration 18, loss = 0.04298113
Iteration 19, loss = 0.05096939
Iteration 20, loss = 0.04377763
Iteration 21, loss = 0.04091264
Iteration 22, loss = 0.04343825
Iteration 23, loss = 0.04497801
Iteration 24, loss = 0.04454707
Iteration 25, loss = 0.04035827
Iteration 26, loss = 0.04309277
Iteration 27, loss = 0.04109823
Iteration 28, loss = 0.04410669
Iteration 29, loss = 0.04623591
Iteration 30, loss = 0.04106406
Iteration 31, loss = 0.04423567
Iteration 32, loss = 0.03186252
Iteration 33, loss = 0.02720727
Iteration 34, loss = 0.04704205
Iteration 35, loss = 0.03888651
Iteration 36, loss = 0.03517685
Iteration 37, loss = 0.03219988
Iteration 38, loss = 0.03721443
Iteration 39, loss = 0.03154602
Iteration 40, loss = 0.04794059
Iteration 41, loss = 0.03522032
Iteration 42, loss = 0.04999745
Iteration 43, loss = 0.03857026
Iteration 44, loss = 0.02964172
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stoppin
g.
Training set score: 0.993452
Test set score: 0.966905
```

In [188…

```
delta = end-start
print(delta)
```

```
0:00:20.316552
```

In [189…

```
mlp5_test_pred = mlp5.score(X_test, y_test)
mlp5_train_pred = mlp5.score(X_train, y_train)
```

In [190…

```
#eval
Model = 'Model 9 (MLP, 3, 50)'
Layers = 3
Nodes = 50
Time = delta.total_seconds()
```

```
Training_Accuracy = mlp5_test_pred
Testing_Accuracy = mlp5_train_pred
row = [Model, Layers, Nodes, Time, Training_Accuracy, Testing_Accuracy]
results = results.append(pd.DataFrame([row], columns=results.columns), ignore_index=Tru
results
```

Out[190…

	Model	Layers	Nodes	Time	Training Accuracy	Testing Accuracy
0	Model 1 (Seq, 2, 50)	2	50	40.178178	0.998244	0.967381
1	Model 2 (Seq, 2, 100)	2	100	44.802751	0.998750	0.973571
2	Model 3 (Seq, 4, 50)	4	50	40.209974	0.992232	0.963929
3	Model 4 (Seq, 4, 100)	4	100	48.241655	0.996280	0.972024
4	Model 5 (MLP, 1, 10)	1	10	25.288042	0.926071	0.961786
5	Model 6 (MLP, 1, 50)	1	50	14.225653	0.970476	1.000000
6	Model 7 (MLP, 1, 100)	1	100	19.180962	0.973690	1.000000
7	Model 8 (MLP, 3, 10)	3	10	20.080916	0.922619	0.948423
8	Model 9 (MLP, 3, 50)	3	50	20.316552	0.966905	0.993452

100 Nodes

In [191…

```
mlp6 = MLPClassifier(
    hidden_layer_sizes=layer_sizes_100x3,
    max_iter=100,
    alpha=1e-4,
    solver="sgd",
    verbose=10,
    random_state=42,
    learning_rate_init=0.2,
)

start = datetime.now()
mlp6.fit(X_train, y_train)
end = datetime.now()

print("Training set score: %f" % mlp6.score(X_train, y_train))
print("Test set score: %f" % mlp6.score(X_test, y_test))
```

```
Iteration 1, loss = 0.52164476
Iteration 2, loss = 0.16402378
Iteration 3, loss = 0.12330468
Iteration 4, loss = 0.10269050
Iteration 5, loss = 0.08547243
Iteration 6, loss = 0.07110801
Iteration 7, loss = 0.06503341
Iteration 8, loss = 0.06123416
Iteration 9, loss = 0.05784609
Iteration 10, loss = 0.04944880
Iteration 11, loss = 0.05467753
Iteration 12, loss = 0.04675435
Iteration 13, loss = 0.03782403
Iteration 14, loss = 0.03772119
Iteration 15, loss = 0.03686301
Iteration 16, loss = 0.03476929
Iteration 17, loss = 0.03707047
Iteration 18, loss = 0.03141235
Iteration 19, loss = 0.02953099
Iteration 20, loss = 0.02785696
Iteration 21, loss = 0.02868018
Iteration 22, loss = 0.03017021
Iteration 23, loss = 0.02582090
Iteration 24, loss = 0.02934057
Iteration 25, loss = 0.03017075
Iteration 26, loss = 0.03563241
Iteration 27, loss = 0.02531422
Iteration 28, loss = 0.03070848
Iteration 29, loss = 0.01830584
Iteration 30, loss = 0.02320620
Iteration 31, loss = 0.01657743
Iteration 32, loss = 0.01536999
Iteration 33, loss = 0.02340464
Iteration 34, loss = 0.02702605
Iteration 35, loss = 0.02032714
Iteration 36, loss = 0.02627075
Iteration 37, loss = 0.01428419
Iteration 38, loss = 0.01827018
Iteration 39, loss = 0.02238833
Iteration 40, loss = 0.01796843
Iteration 41, loss = 0.01250539
Iteration 42, loss = 0.02029697
Iteration 43, loss = 0.01864376
Iteration 44, loss = 0.02039657
Iteration 45, loss = 0.02202145
Iteration 46, loss = 0.01223888
Iteration 47, loss = 0.01492640
Iteration 48, loss = 0.01606106
Iteration 49, loss = 0.01659242
Iteration 50, loss = 0.01653984
Iteration 51, loss = 0.01395894
Iteration 52, loss = 0.02120525
Iteration 53, loss = 0.02010606
Iteration 54, loss = 0.01606660
Iteration 55, loss = 0.01965114
Iteration 56, loss = 0.01672052
Iteration 57, loss = 0.01064272
Iteration 58, loss = 0.00980046
Iteration 59, loss = 0.01114746
Iteration 60, loss = 0.01737239
Iteration 61, loss = 0.01563515
Iteration 62, loss = 0.01964429
Iteration 63, loss = 0.02012231
Iteration 64, loss = 0.01970246
Iteration 65, loss = 0.02374014
Iteration 66, loss = 0.02634564
Iteration 67, loss = 0.01698695
Iteration 68, loss = 0.01333169
Iteration 69, loss = 0.01792261
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stoppin
g.
Training set score: 0.997113
Test set score: 0.967738
```

In [192…

```
delta = end-start
print(delta)
```

```
0:00:53.831229
```

In [193…

```
mlp6_test_pred = mlp6.score(X_test, y_test)
mlp6_train_pred = mlp6.score(X_train, y_train)
```

In [194…

```
#eval
Model = 'Model 10 (MLP, 3, 100)'
Layers = 3
Nodes = 100
Time = delta.total_seconds()
Training_Accuracy = mlp6_test_pred
Testing_Accuracy = mlp6_train_pred
row = [Model, Layers, Nodes, Time, Training_Accuracy, Testing_Accuracy]
results = results.append(pd.DataFrame([row], columns=results.columns), ignore_index=Tru
results
```

Out[194…

	Model	Layers	Nodes	Time	Training Accuracy	Testing Accuracy
0	Model 1 (Seq, 2, 50)	2	50	40.178178	0.998244	0.967381
1	Model 2 (Seq, 2, 100)	2	100	44.802751	0.998750	0.973571
2	Model 3 (Seq, 4, 50)	4	50	40.209974	0.992232	0.963929
3	Model 4 (Seq, 4, 100)	4	100	48.241655	0.996280	0.972024
4	Model 5 (MLP, 1, 10)	1	10	25.288042	0.926071	0.961786
5	Model 6 (MLP, 1, 50)	1	50	14.225653	0.970476	1.000000
6	Model 7 (MLP, 1, 100)	1	100	19.180962	0.973690	1.000000
7	Model 8 (MLP, 3, 10)	3	10	20.080916	0.922619	0.948423
8	Model 9 (MLP, 3, 50)	3	50	20.316552	0.966905	0.993452
9	Model 10 (MLP, 3, 100)	3	100	53.831229	0.967738	0.997113

5 Layers

10 Nodes

In [195…

```
mlp7 = MLPClassifier(
    hidden_layer_sizes=layer_sizes_10x5,
    max_iter=100,
    alpha=1e-4,
    solver="sgd",
    verbose=10,
    random_state=42,
    learning_rate_init=0.2,
)

start = datetime.now()
mlp7.fit(X_train, y_train)
end = datetime.now()

print("Training set score: %f" % mlp7.score(X_train, y_train))
print("Test set score: %f" % mlp7.score(X_test, y_test))
```

```
Iteration 1, loss = 1.65926315
Iteration 2, loss = 2.18176474
Iteration 3, loss = 2.29935690
Iteration 4, loss = 2.30176772
Iteration 5, loss = 2.30245862
Iteration 6, loss = 2.30233233
Iteration 7, loss = 2.30212881
Iteration 8, loss = 2.30174824
Iteration 9, loss = 2.30194928
Iteration 10, loss = 2.30217997
Iteration 11, loss = 2.30183751
Iteration 12, loss = 2.30174213
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stoppin
g.
Training set score: 0.113304
Test set score: 0.109405
```

In [196…

```
delta = end-start
print(delta)
```

```
0:00:03.691313
```

In [197…

```
mlp7_test_pred = mlp7.score(X_test, y_test)
mlp7_train_pred = mlp7.score(X_train, y_train)
```

In [198…

```
#eval
Model = 'Model 11 (MLP, 5, 10)'
Layers = 5
Nodes = 10
Time = delta.total_seconds()
Training_Accuracy = mlp7_test_pred
Testing_Accuracy = mlp7_train_pred
row = [Model, Layers, Nodes, Time, Training_Accuracy, Testing_Accuracy]
results = results.append(pd.DataFrame([row], columns=results.columns), ignore_index=Tru
results
```

Out[198…

	Model	Layers	Nodes	Time	Training Accuracy	Testing Accuracy
0	Model 1 (Seq, 2, 50)	2	50	40.178178	0.998244	0.967381
1	Model 2 (Seq, 2, 100)	2	100	44.802751	0.998750	0.973571
2	Model 3 (Seq, 4, 50)	4	50	40.209974	0.992232	0.963929
3	Model 4 (Seq, 4, 100)	4	100	48.241655	0.996280	0.972024
4	Model 5 (MLP, 1, 10)	1	10	25.288042	0.926071	0.961786
5	Model 6 (MLP, 1, 50)	1	50	14.225653	0.970476	1.000000
6	Model 7 (MLP, 1, 100)	1	100	19.180962	0.973690	1.000000
7	Model 8 (MLP, 3, 10)	3	10	20.080916	0.922619	0.948423
8	Model 9 (MLP, 3, 50)	3	50	20.316552	0.966905	0.993452
9	Model 10 (MLP, 3, 100)	3	100	53.831229	0.967738	0.997113
10	Model 11 (MLP, 5, 10)	5	10	3.691313	0.109405	0.113304

50 Nodes

In [199…

```
mlp8 = MLPClassifier(
    hidden_layer_sizes=layer_sizes_50x5,
    max_iter=100,
    alpha=1e-4,
    solver="sgd",
    verbose=10,
    random_state=42,
    learning_rate_init=0.2,
)

start = datetime.now()
mlp8.fit(X_train, y_train)
end = datetime.now()

print("Training set score: %f" % mlp8.score(X_train, y_train))
print("Test set score: %f" % mlp8.score(X_test, y_test))
```

```
Iteration 1, loss = 1.33298166
Iteration 2, loss = 0.56228986
Iteration 3, loss = 0.37054011
Iteration 4, loss = 0.30428159
Iteration 5, loss = 0.27811590
Iteration 6, loss = 0.25683773
Iteration 7, loss = 0.24304994
Iteration 8, loss = 0.23908765
Iteration 9, loss = 0.22275476
Iteration 10, loss = 0.21691503
Iteration 11, loss = 0.20732844
Iteration 12, loss = 0.20797553
Iteration 13, loss = 0.19814034
Iteration 14, loss = 0.20593115
Iteration 15, loss = 0.18688355
Iteration 16, loss = 0.18373304
Iteration 17, loss = 0.18307489
Iteration 18, loss = 0.18259575
Iteration 19, loss = 0.17538491
Iteration 20, loss = 0.16984887
Iteration 21, loss = 0.16828121
Iteration 22, loss = 0.16565114
Iteration 23, loss = 0.16180297
Iteration 24, loss = 0.16187776
Iteration 25, loss = 0.15270026
Iteration 26, loss = 0.14888043
Iteration 27, loss = 0.15526094
Iteration 28, loss = 0.15593925
Iteration 29, loss = 0.14624261
Iteration 30, loss = 0.16611589
Iteration 31, loss = 1583758.33619908
Iteration 32, loss = 26365321.13103580
Iteration 33, loss = 26630903.27309979
Iteration 34, loss = 26621956.72254743
Iteration 35, loss = 26613013.17178402
Iteration 36, loss = 26604072.62479221
Iteration 37, loss = 26595135.08229299
Iteration 38, loss = 26586200.54163133
Iteration 39, loss = 26577269.00345315
Iteration 40, loss = 26568340.46524443
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stoppin
g.
Training set score: 0.099137
Test set score: 0.100714
```

In [200…

```
delta = end-start
```

```
print(delta)
```

```
0:00:21.608151
```

In [201…

```
mlp8_test_pred = mlp8.score(X_test, y_test)
mlp8_train_pred = mlp8.score(X_train, y_train)
```

In [202…

```
#eval
Model = 'Model 12 (MLP, 5, 50)'
Layers = 5
Nodes = 50
Time = delta.total_seconds()
Training_Accuracy = mlp8_test_pred
Testing_Accuracy = mlp8_train_pred
row = [Model, Layers, Nodes, Time, Training_Accuracy, Testing_Accuracy]
results = results.append(pd.DataFrame([row], columns=results.columns), ignore_index=Tru
results
```

Out[202…

	Model	Layers	Nodes	Time	Training Accuracy	Testing Accuracy
0	Model 1 (Seq, 2, 50)	2	50	40.178178	0.998244	0.967381
1	Model 2 (Seq, 2, 100)	2	100	44.802751	0.998750	0.973571
2	Model 3 (Seq, 4, 50)	4	50	40.209974	0.992232	0.963929
3	Model 4 (Seq, 4, 100)	4	100	48.241655	0.996280	0.972024
4	Model 5 (MLP, 1, 10)	1	10	25.288042	0.926071	0.961786
5	Model 6 (MLP, 1, 50)	1	50	14.225653	0.970476	1.000000
6	Model 7 (MLP, 1, 100)	1	100	19.180962	0.973690	1.000000
7	Model 8 (MLP, 3, 10)	3	10	20.080916	0.922619	0.948423
8	Model 9 (MLP, 3, 50)	3	50	20.316552	0.966905	0.993452
9	Model 10 (MLP, 3, 100)	3	100	53.831229	0.967738	0.997113
10	Model 11 (MLP, 5, 10)	5	10	3.691313	0.109405	0.113304
11	Model 12 (MLP, 5, 50)	5	50	21.608151	0.100714	0.099137

100 Nodes

In [203…

```
mlp9 = MLPClassifier(
    hidden_layer_sizes=layer_sizes_100x5,
    max_iter=65,
    alpha=1e-4,
    solver="sgd",
    verbose=10,
    random_state=42,
    learning_rate_init=0.2,
)

start = datetime.now()
mlp9.fit(X_train, y_train)
end = datetime.now()
```

```
print("Training set score: %f" % mlp9.score(X_train, y_train))
print("Test set score: %f" % mlp9.score(X_test, y_test))
```

```
Iteration 1, loss = 0.81648253
Iteration 2, loss = 0.21923203
Iteration 3, loss = 0.16311275
Iteration 4, loss = 0.13162621
Iteration 5, loss = 0.11535752
Iteration 6, loss = 0.09349293
Iteration 7, loss = 0.08865608
Iteration 8, loss = 0.08041453
Iteration 9, loss = 0.06911561
Iteration 10, loss = 0.06949015
Iteration 11, loss = 0.05939087
Iteration 12, loss = 0.06019510
Iteration 13, loss = 0.05525488
Iteration 14, loss = 0.05353015
Iteration 15, loss = 0.04777815
Iteration 16, loss = 0.04135481
Iteration 17, loss = 0.04212481
Iteration 18, loss = 0.03968593
Iteration 19, loss = 0.03838957
Iteration 20, loss = 0.03899387
Iteration 21, loss = 0.03745069
Iteration 22, loss = 0.03796576
Iteration 23, loss = 0.03380817
Iteration 24, loss = 0.03332531
Iteration 25, loss = 0.03562480
Iteration 26, loss = 0.03022161
Iteration 27, loss = 0.03203888
Iteration 28, loss = 0.02593825
Iteration 29, loss = 0.03003061
Iteration 30, loss = 0.02810734
Iteration 31, loss = 0.02606915
Iteration 32, loss = 0.03021289
Iteration 33, loss = 0.02431933
Iteration 34, loss = 0.02463925
Iteration 35, loss = 0.02775000
Iteration 36, loss = 0.02490856
Iteration 37, loss = 0.03479933
Iteration 38, loss = 0.02630577
Iteration 39, loss = 0.02504796
Iteration 40, loss = 0.02385996
Iteration 41, loss = 0.02570856
Iteration 42, loss = 0.01835720
Iteration 43, loss = 0.01670299
Iteration 44, loss = 0.02055414
Iteration 45, loss = 0.01756446
Iteration 46, loss = 0.01263776
Iteration 47, loss = 0.01776810
Iteration 48, loss = 0.01535573
Iteration 49, loss = 0.01630602
Iteration 50, loss = 0.01717806
Iteration 51, loss = 0.01075398
Iteration 52, loss = 0.02093156
Iteration 53, loss = 0.02009459
Iteration 54, loss = 0.01275531
Iteration 55, loss = 0.01594768
Iteration 56, loss = 0.01317940
Iteration 57, loss = 0.00917176
Iteration 58, loss = 0.01090047
Iteration 59, loss = 0.02058604
Iteration 60, loss = 0.02117412
Iteration 61, loss = 0.02109674
Iteration 62, loss = 0.01589923
Iteration 63, loss = 0.01431999
Iteration 64, loss = 0.01131108
Iteration 65, loss = 0.01513471
```

```
C:\Users\sjue\Anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.
py:702: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (65) reached and th
e optimization hasn't converged yet.
  warnings.warn(
Training set score: 0.995863
Test set score: 0.968333
```

In [204…

```
delta = end-start
print(delta)
```

```
0:01:00.874613
```

In [205…

```
mlp9_test_pred = mlp9.score(X_test, y_test)
mlp9_train_pred = mlp9.score(X_train, y_train)
```

Model Results Table

In [206…

```
#eval
Model = 'Model 13 (MLP, 5, 100)'
Layers = 5
Nodes = 100
Time = delta.total_seconds()
Training_Accuracy = mlp9_test_pred
Testing_Accuracy = mlp9_train_pred
row = [Model, Layers, Nodes, Time, Training_Accuracy, Testing_Accuracy]
results = results.append(pd.DataFrame([row], columns=results.columns), ignore_index=Tru
results
```

Out[206…

	Model	Layers	Nodes	Time	Training Accuracy	Testing Accuracy
0	Model 1 (Seq, 2, 50)	2	50	40.178178	0.998244	0.967381
1	Model 2 (Seq, 2, 100)	2	100	44.802751	0.998750	0.973571
2	Model 3 (Seq, 4, 50)	4	50	40.209974	0.992232	0.963929
3	Model 4 (Seq, 4, 100)	4	100	48.241655	0.996280	0.972024
4	Model 5 (MLP, 1, 10)	1	10	25.288042	0.926071	0.961786
5	Model 6 (MLP, 1, 50)	1	50	14.225653	0.970476	1.000000
6	Model 7 (MLP, 1, 100)	1	100	19.180962	0.973690	1.000000
7	Model 8 (MLP, 3, 10)	3	10	20.080916	0.922619	0.948423
8	Model 9 (MLP, 3, 50)	3	50	20.316552	0.966905	0.993452
9	Model 10 (MLP, 3, 100)	3	100	53.831229	0.967738	0.997113
10	Model 11 (MLP, 5, 10)	5	10	3.691313	0.109405	0.113304
11	Model 12 (MLP, 5, 50)	5	50	21.608151	0.100714	0.099137
12	Model 13 (MLP, 5, 100)	5	100	60.874613	0.968333	0.995863

Predictions

In [207…

```
# Predicting the number labels

# sequential model 1
seq1_y_pred = model1.predict(X_test)
seq1_y_pred = np.argmax(seq1_y_pred, axis=1) # Here we get the index of maximum value i

# sequential model 2
seq2_y_pred = model2.predict(X_test)
seq2_y_pred = np.argmax(seq2_y_pred, axis=1)
# sequential model 3
seq3_y_pred = model3.predict(X_test)
seq3_y_pred = np.argmax(seq3_y_pred, axis=1)

# sequential model 4
seq4_y_pred = model4.predict(X_test)
seq4_y_pred = np.argmax(seq4_y_pred, axis=1)
```

```
263/263 [==============================] - 0s 924us/step
263/263 [==============================] - 0s 938us/step
263/263 [==============================] - 0s 915us/step
263/263 [==============================] - 0s 930us/step
```

In [208…

```
mlp1_test_pred = mlp1.predict(X_test)
mlp1_train_pred = mlp1.predict(X_train)
```

In [209…

```
mlp2_test_pred = mlp2.predict(X_test)
mlp2_train_pred = mlp2.predict(X_train)
```

In [210…

```
mlp3_test_pred = mlp3.predict(X_test)
mlp3_train_pred = mlp3.predict(X_train)
```

In [211…

```
mlp4_test_pred = mlp4.predict(X_test)
mlp4_train_pred = mlp4.predict(X_train)
```

In [212…

```
mlp5_test_pred = mlp5.predict(X_test)
mlp5_train_pred = mlp5.predict(X_train)
```

In [213…

```
mlp6_test_pred = mlp6.predict(X_test)
mlp6_train_pred = mlp6.predict(X_train)
```

In [214…

```
mlp7_test_pred = mlp7.predict(X_test)
mlp7_train_pred = mlp7.predict(X_train)
```

In [215…

```
mlp8_test_pred = mlp8.predict(X_test)
mlp8_train_pred = mlp8.predict(X_train)
```

In [216…

```
mlp9_test_pred = mlp9.predict(X_test)
mlp9_train_pred = mlp9.predict(X_train)
```

Confusion Matrices

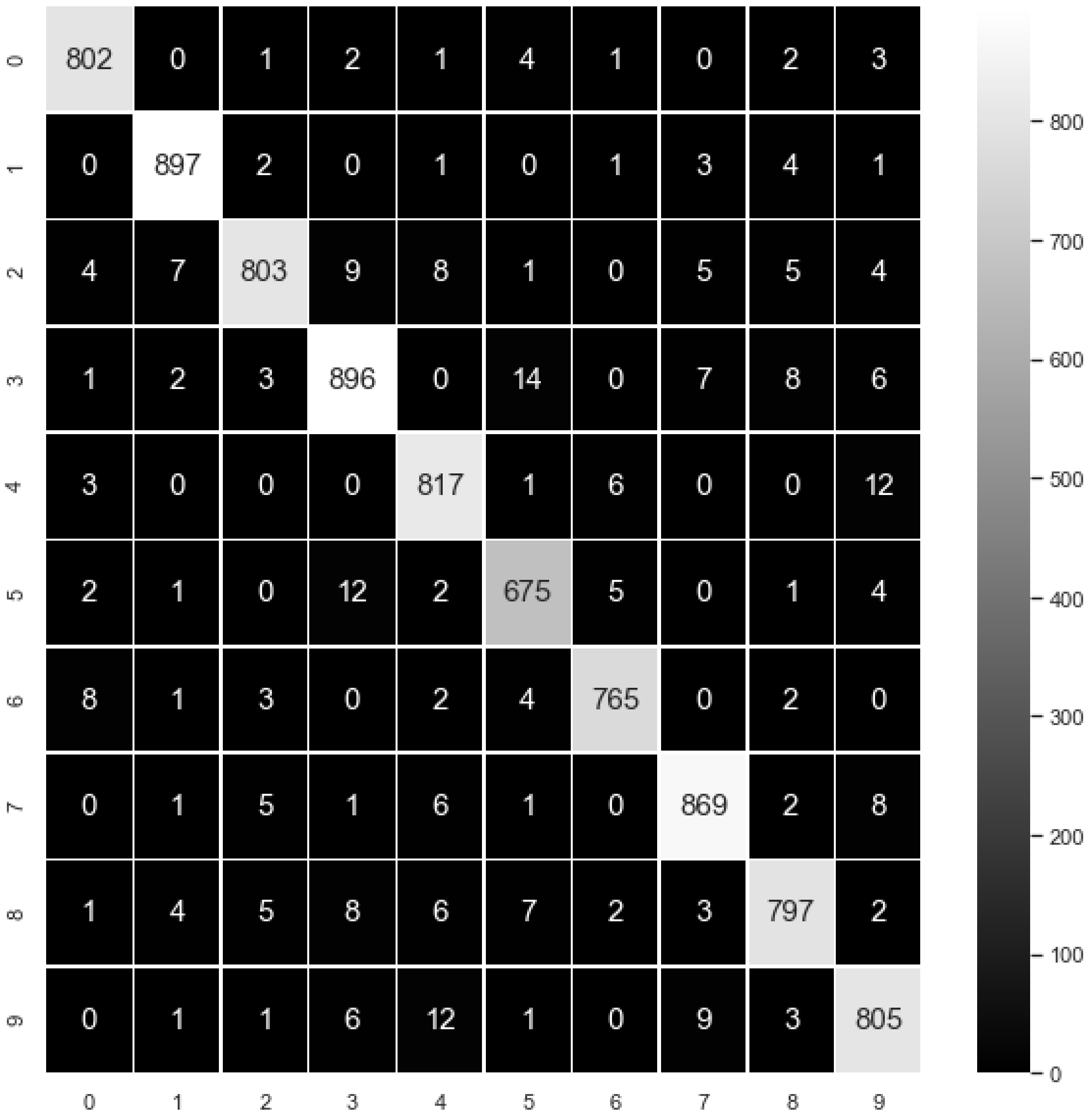
Sequential Models

2 Layers, 50 Nodes

In [217…

```
#Confusion matrix for sequential model 1
con_mat=confusion_matrix(y_test, seq1_y_pred)
plt.style.use('seaborn-deep')
plt.figure(figsize=(10,10))
sns.heatmap(con_mat,annot=True,annot_kws={'size': 15},linewidths=0.5,fmt="d",cmap="gray
plt.title('Sequential Model 1 True or False predicted digit MNIST\n',fontweight='bold',
plt.show()
```

Sequential Model 1 True or False predicted digit MNIST



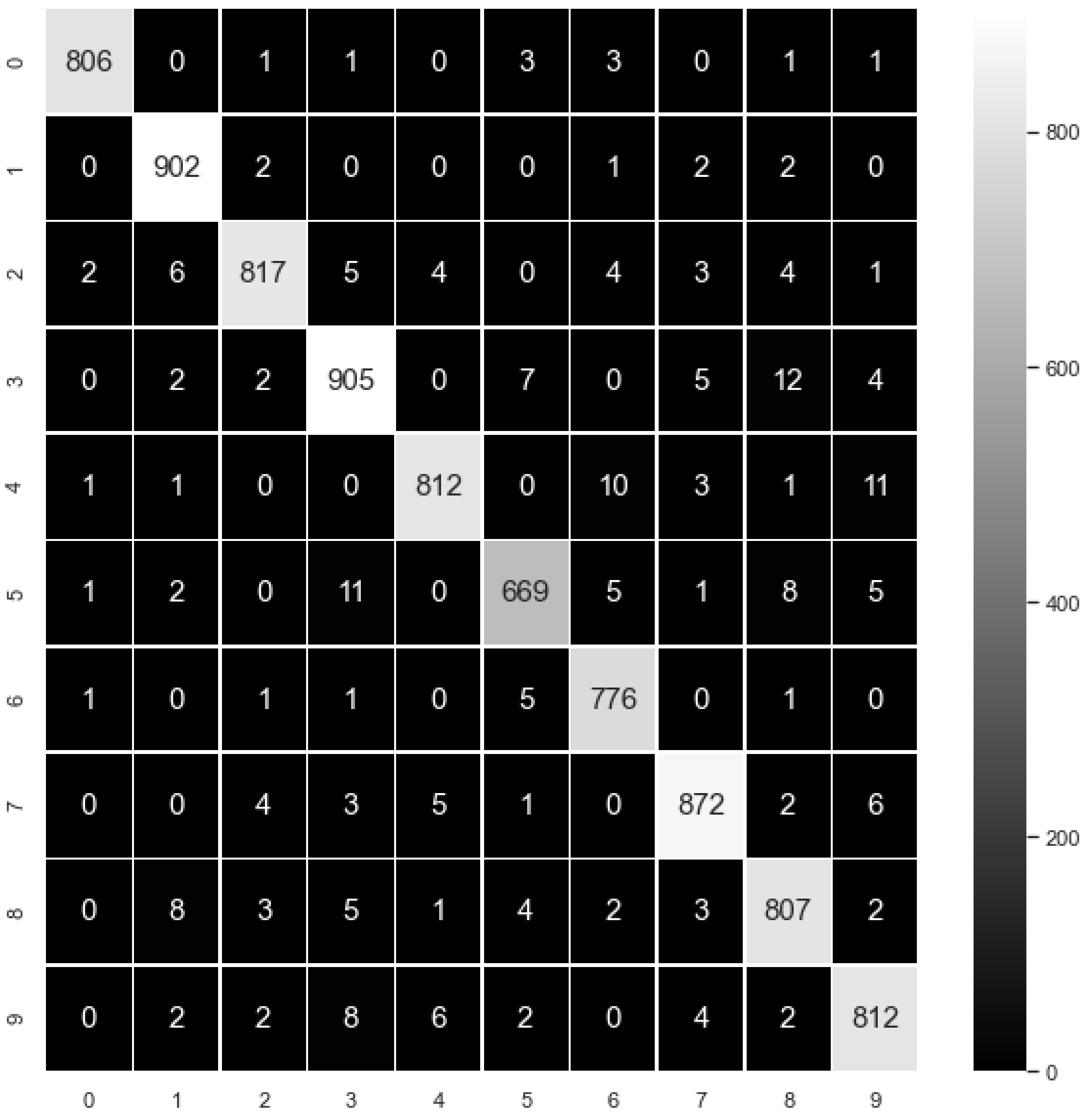
	0	1	2	3	4	5	6	7	8	9
0	802	0	1	2	1	4	1	0	2	3
1	0	897	2	0	1	0	1	3	4	1
2	4	7	803	9	8	1	0	5	5	4
3	1	2	3	896	0	14	0	7	8	6
4	3	0	0	0	817	1	6	0	0	12
5	2	1	0	12	2	675	5	0	1	4
6	8	1	3	0	2	4	765	0	2	0
7	0	1	5	1	6	1	0	869	2	8
8	1	4	5	8	6	7	2	3	797	2
9	0	1	1	6	12	1	0	9	3	805

2 layers, 100 Nodes

In [218…

```
#Confusion matrix for sequential model 2
con_mat=confusion_matrix(y_test, seq2_y_pred)
plt.style.use('seaborn-deep')
plt.figure(figsize=(10,10))
sns.heatmap(con_mat,annot=True,annot_kws={'size': 15},linewidths=0.5,fmt="d",cmap="gray
plt.title('Sequential Model 2 True or False predicted digit MNIST\n',fontweight='bold',
plt.show()
```

Sequential Model 2 True or False predicted digit MNIST



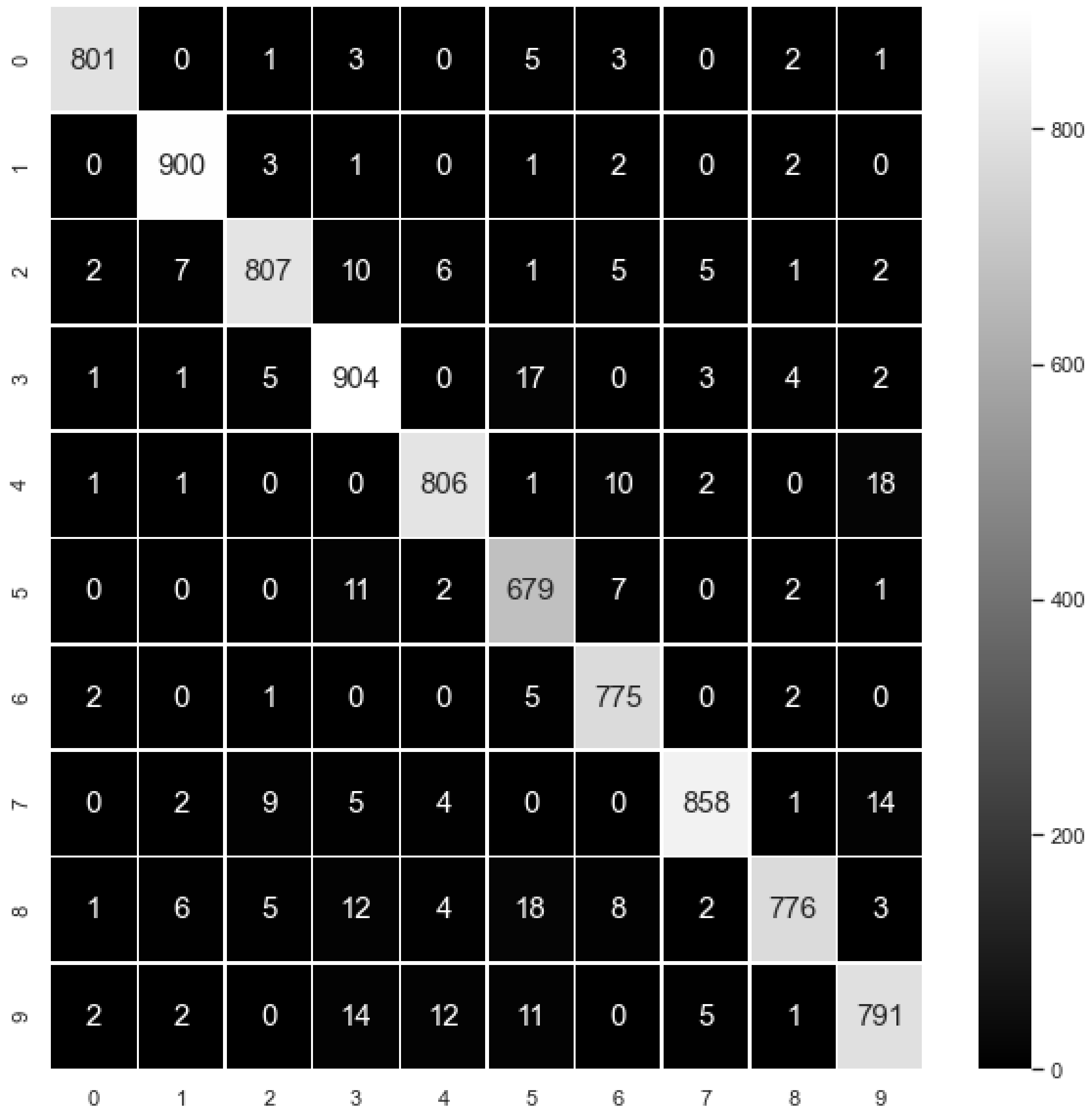
	0	1	2	3	4	5	6	7	8	9
0	806	0	1	1	0	3	3	0	1	1
1	0	902	2	0	0	0	1	2	2	0
2	2	6	817	5	4	0	4	3	4	1
3	0	2	2	905	0	7	0	5	12	4
4	1	1	0	0	812	0	10	3	1	11
5	1	2	0	11	0	669	5	1	8	5
6	1	0	1	1	0	5	776	0	1	0
7	0	0	4	3	5	1	0	872	2	6
8	0	8	3	5	1	4	2	3	807	2
9	0	2	2	8	6	2	0	4	2	812

4 Layers, 50 Nodes

In [219…

```
#Confusion matrix for sequential model 3
con_mat=confusion_matrix(y_test, seq3_y_pred)
plt.style.use('seaborn-deep')
plt.figure(figsize=(10,10))
sns.heatmap(con_mat,annot=True,annot_kws={'size': 15},linewidths=0.5,fmt="d",cmap="gray
plt.title('Sequential Model 3 True or False predicted digit MNIST\n',fontweight='bold',
plt.show()
```

Sequential Model 3 True or False predicted digit MNIST



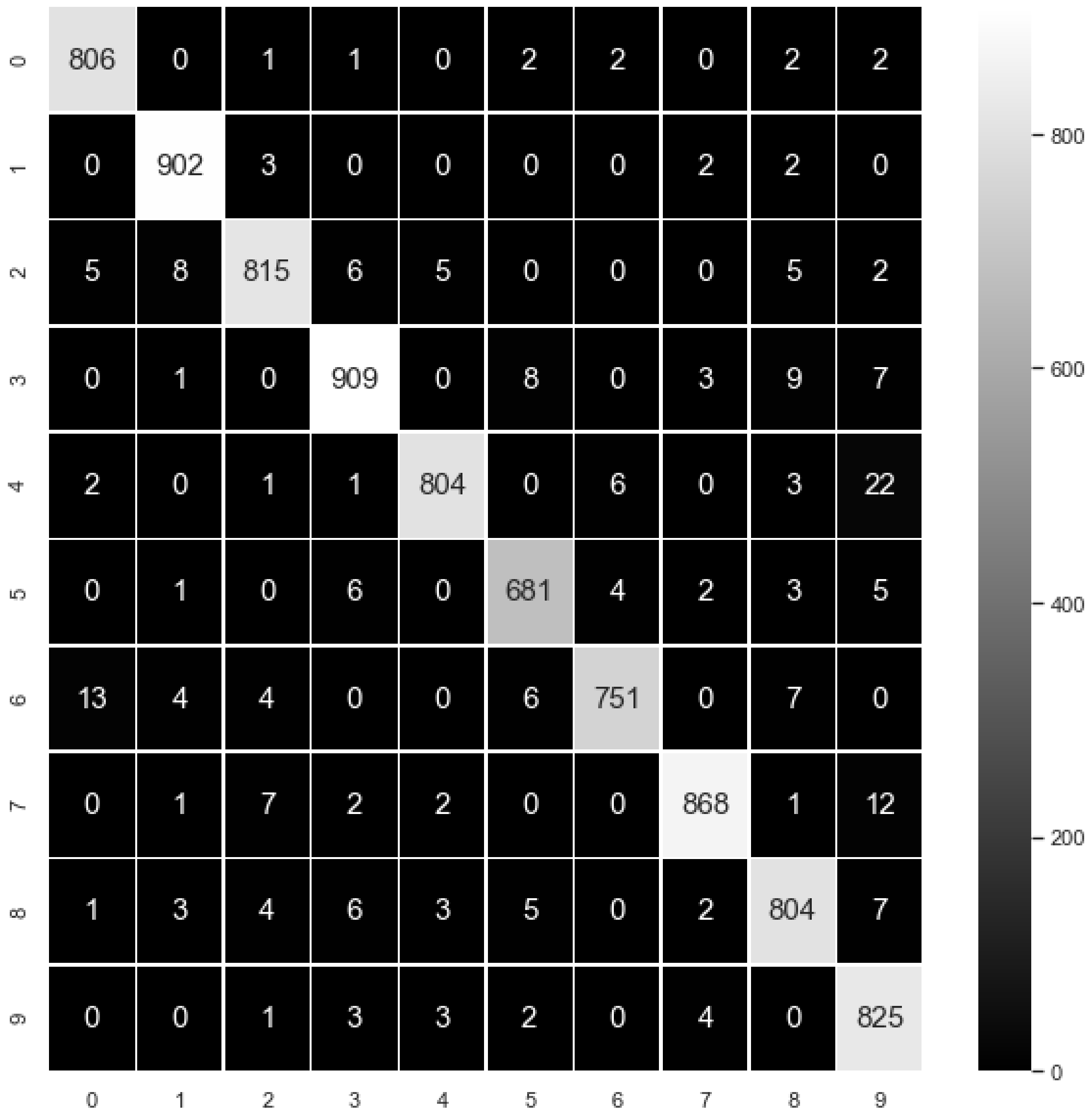
	0	1	2	3	4	5	6	7	8	9
0	801	0	1	3	0	5	3	0	2	1
1	0	900	3	1	0	1	2	0	2	0
2	2	7	807	10	6	1	5	5	1	2
3	1	1	5	904	0	17	0	3	4	2
4	1	1	0	0	806	1	10	2	0	18
5	0	0	0	11	2	679	7	0	2	1
6	2	0	1	0	0	5	775	0	2	0
7	0	2	9	5	4	0	0	858	1	14
8	1	6	5	12	4	18	8	2	776	3
9	2	2	0	14	12	11	0	5	1	791

4 Layers, 100 Nodes

In [220…

```
#Confusion matrix for sequential model 4
con_mat=confusion_matrix(y_test, seq4_y_pred)
plt.style.use('seaborn-deep')
plt.figure(figsize=(10,10))
sns.heatmap(con_mat,annot=True,annot_kws={'size': 15},linewidths=0.5,fmt="d",cmap="gray
plt.title('Sequential Model 4 True or False predicted digit MNIST\n',fontweight='bold',
plt.show()
```

Sequential Model 4 True or False predicted digit MNIST



	0	1	2	3	4	5	6	7	8	9
0	806	0	1	1	0	2	2	0	2	2
1	0	902	3	0	0	0	0	2	2	0
2	5	8	815	6	5	0	0	0	5	2
3	0	1	0	909	0	8	0	3	9	7
4	2	0	1	1	804	0	6	0	3	22
5	0	1	0	6	0	681	4	2	3	5
6	13	4	4	0	0	6	751	0	7	0
7	0	1	7	2	2	0	0	868	1	12
8	1	3	4	6	3	5	0	2	804	7
9	0	0	1	3	3	2	0	4	0	825

MLP Classifier Models

1 Layer

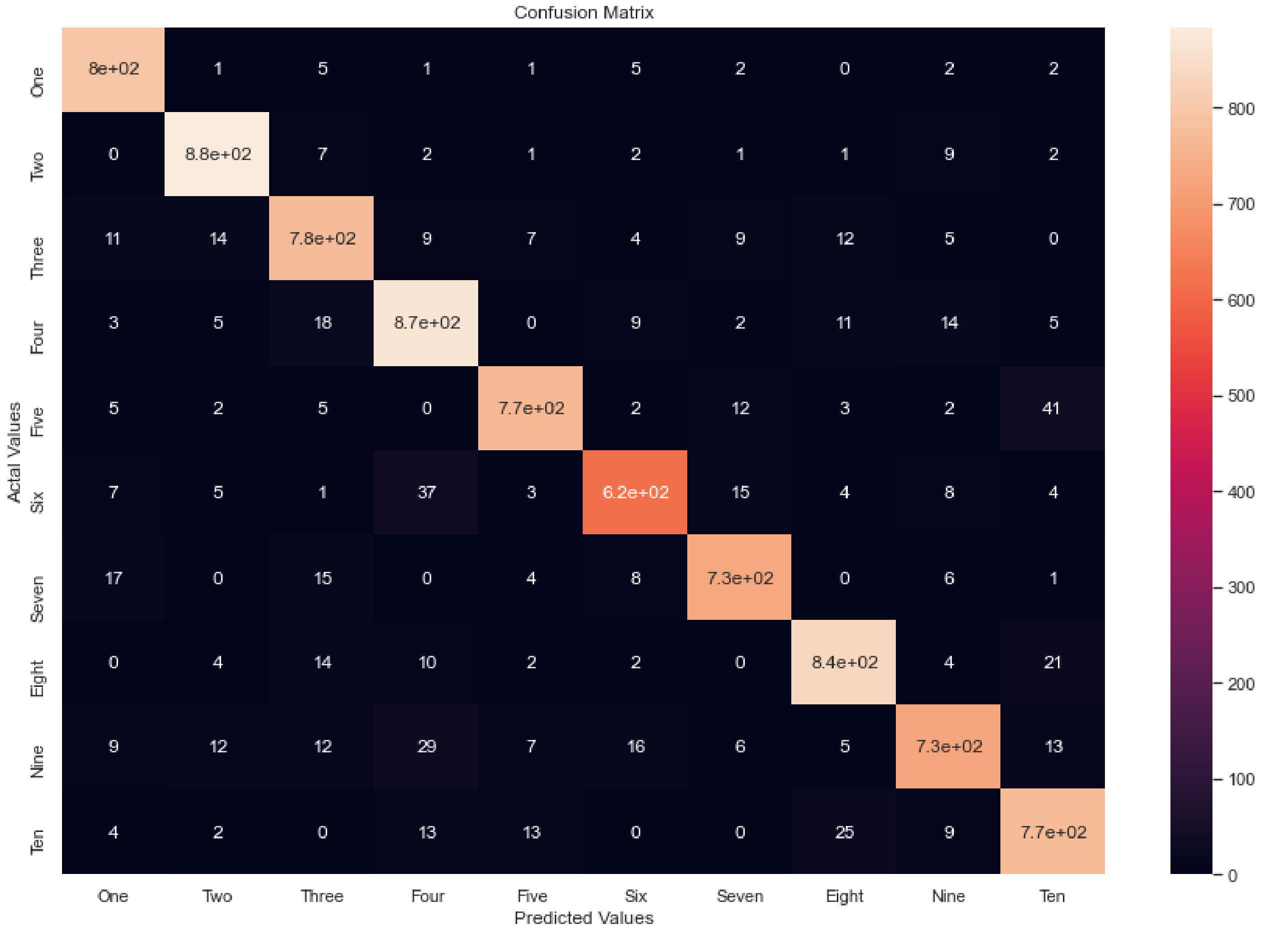
10 Nodes

In [221…

```
cm1 = confusion_matrix(y_test, mlp1_test_pred)

cm1_df = pd.DataFrame(cm1,
                     index = ['One','Two','Three','Four','Five','Six','Seven','Eight','
                     columns = ['One','Two','Three','Four','Five','Six','Seven','Eight'

#Plotting the confusion matrix
plt.figure(figsize=(15,10))
sns.heatmap(cm1_df, annot=True)
plt.title('Confusion Matrix')
plt.ylabel('Actal Values')
plt.xlabel('Predicted Values')
plt.show()
```



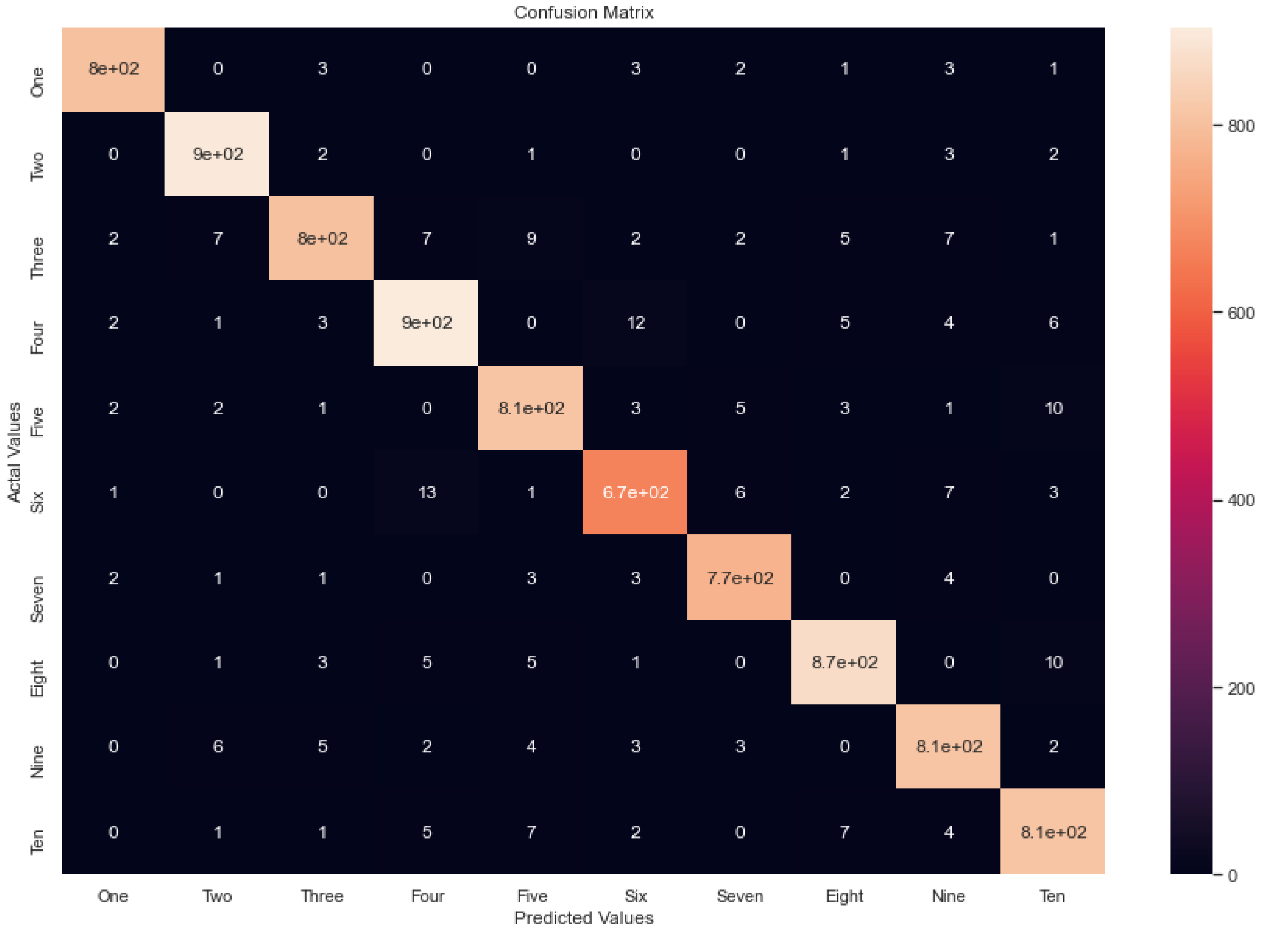
50 Nodes

In [222…

```
cm2 = confusion_matrix(y_test, mlp2_test_pred)

cm2_df = pd.DataFrame(cm2,
                     index = ['One','Two','Three','Four','Five','Six','Seven','Eight','
                     columns = ['One','Two','Three','Four','Five','Six','Seven','Eight'

#Plotting the confusion matrix
plt.figure(figsize=(15,10))
sns.heatmap(cm2_df, annot=True)
plt.title('Confusion Matrix')
plt.ylabel('Actal Values')
plt.xlabel('Predicted Values')
plt.show()
```



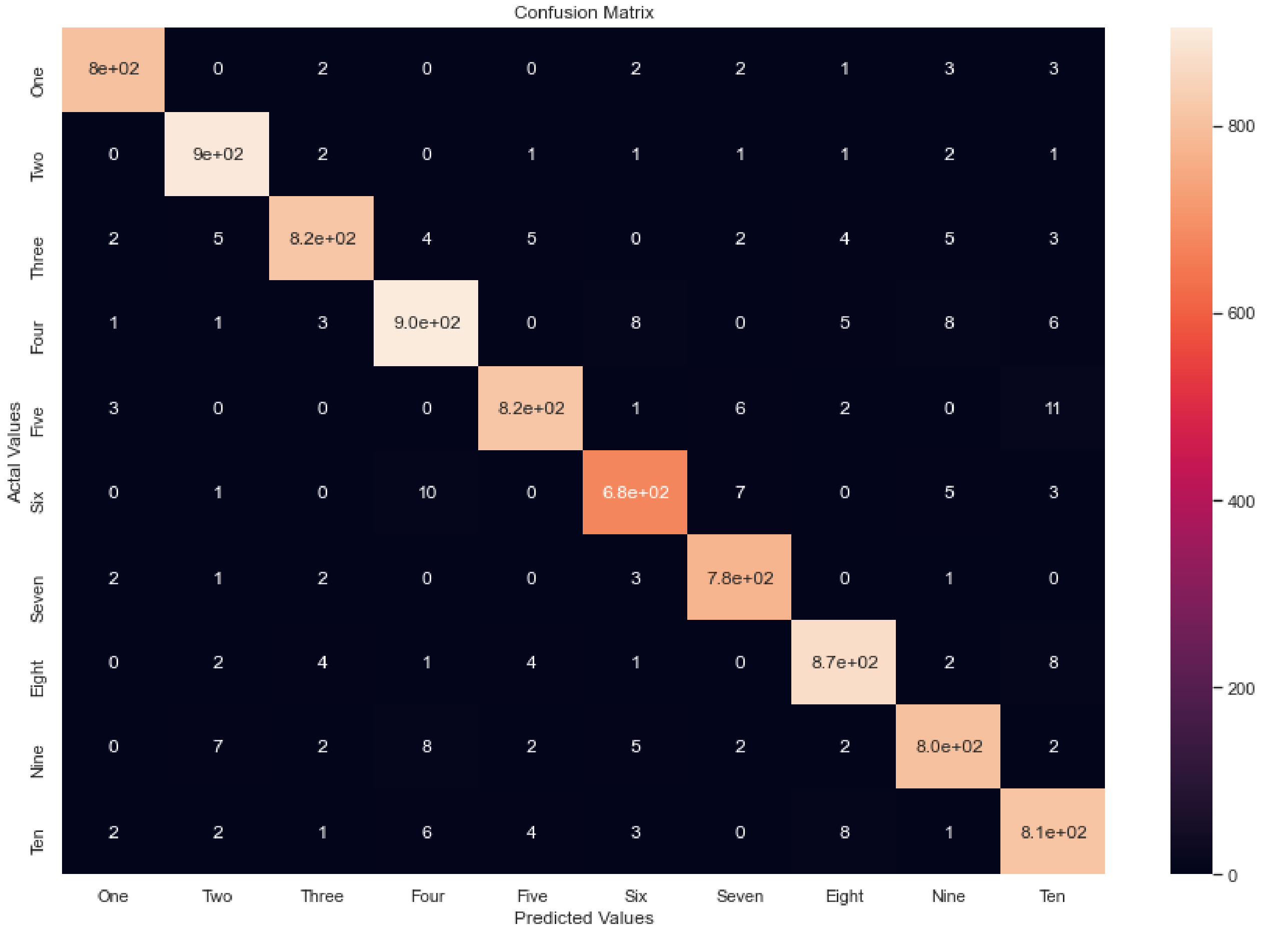
100 Nodes

In [223…

```
cm3 = confusion_matrix(y_test, mlp3_test_pred)

cm3_df = pd.DataFrame(cm3,
                     index = ['One','Two','Three','Four','Five','Six','Seven','Eight','
                     columns = ['One','Two','Three','Four','Five','Six','Seven','Eight'

#Plotting the confusion matrix
plt.figure(figsize=(15,10))
sns.heatmap(cm3_df, annot=True)
plt.title('Confusion Matrix')
plt.ylabel('Actal Values')
plt.xlabel('Predicted Values')
plt.show()
```



3 Layers

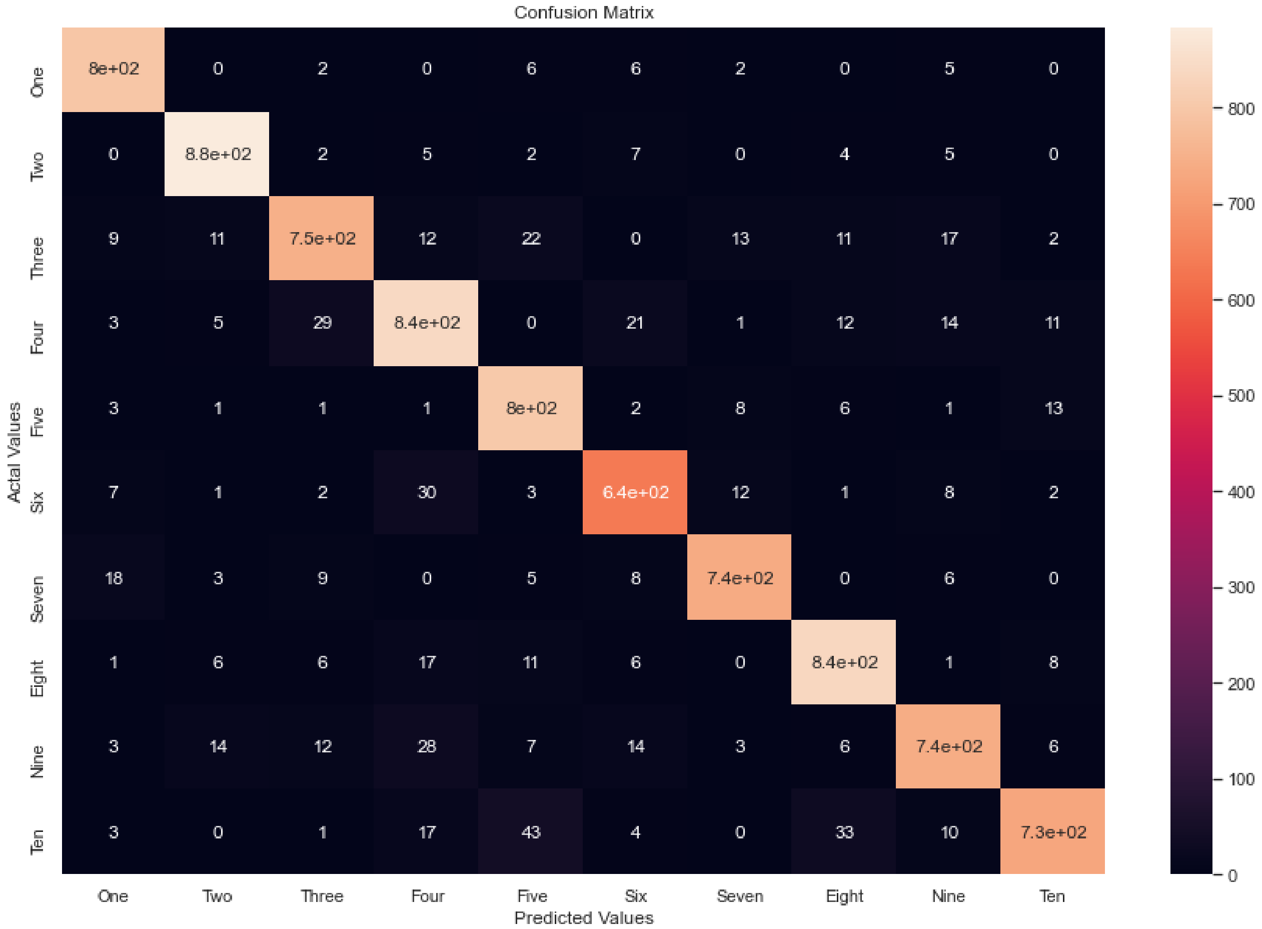
10 Nodes

In [224…

```
cm4 = confusion_matrix(y_test, mlp4_test_pred)

cm4_df = pd.DataFrame(cm4,
                     index = ['One','Two','Three','Four','Five','Six','Seven','Eight','
                     columns = ['One','Two','Three','Four','Five','Six','Seven','Eight'

#Plotting the confusion matrix
plt.figure(figsize=(15,10))
sns.heatmap(cm4_df, annot=True)
plt.title('Confusion Matrix')
plt.ylabel('Actal Values')
plt.xlabel('Predicted Values')
plt.show()
```



	One	Two	Three	Four	Five	Six	Seven	Eight	Nine	Ten
One	8e+02	0	2	0	6	6	2	0	5	0
Two	0	8.8e+02	2	5	2	7	0	4	5	0
Three	9	11	7.5e+02	12	22	0	13	11	17	2
Four	3	5	29	8.4e+02	0	21	1	12	14	11
Five	3	1	1	1	8e+02	2	8	6	1	13
Six	7	1	2	30	3	6.4e+02	12	1	8	2
Seven	18	3	9	0	5	8	7.4e+02	0	6	0
Eight	1	6	6	17	11	6	0	8.4e+02	1	8
Nine	3	14	12	28	7	14	3	6	7.4e+02	6
Ten	3	0	1	17	43	4	0	33	10	7.3e+02

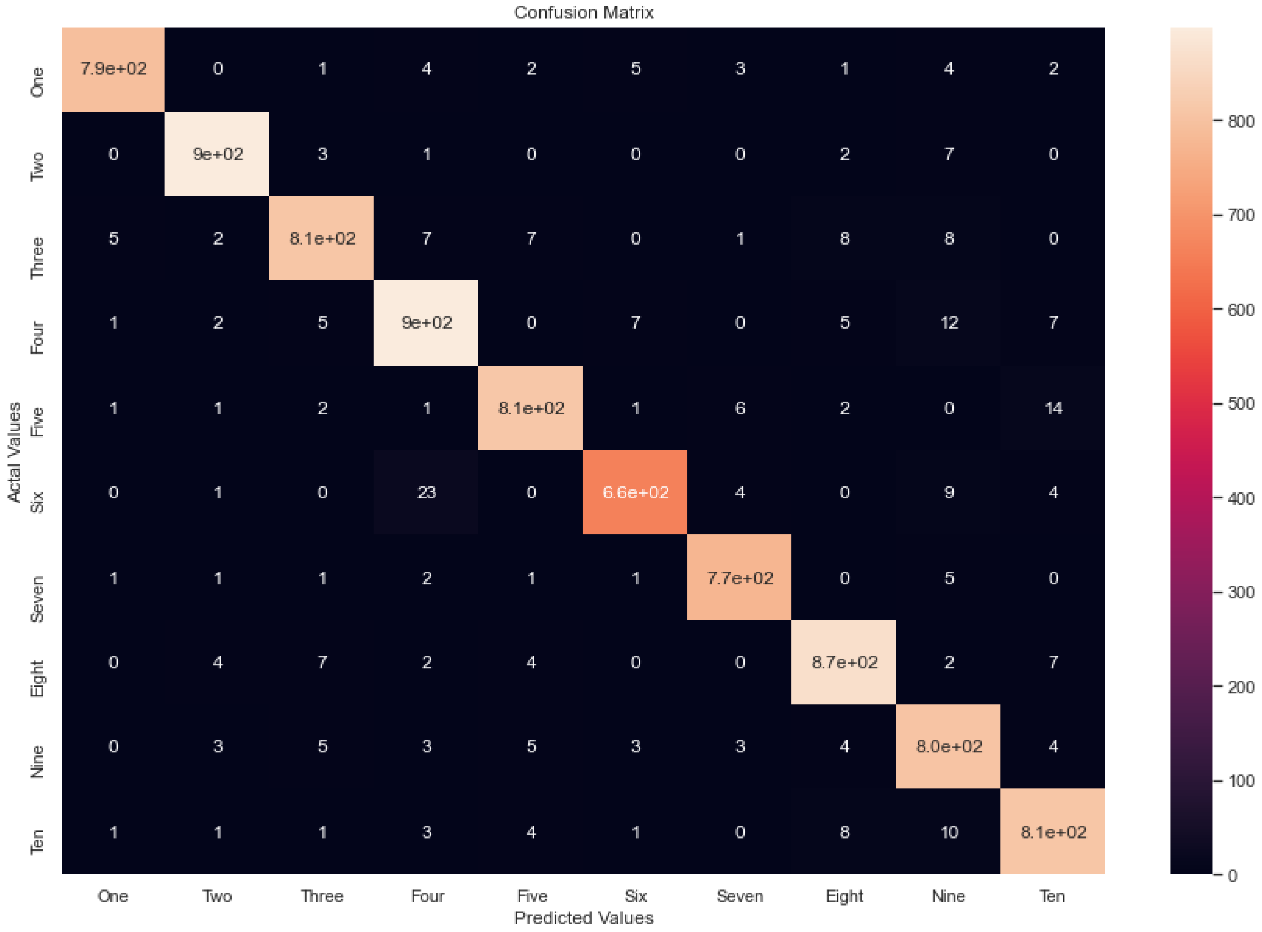
50 Nodes

In [225…

```
cm5 = confusion_matrix(y_test, mlp5_test_pred)

cm5_df = pd.DataFrame(cm5,
                     index = ['One','Two','Three','Four','Five','Six','Seven','Eight','
                     columns = ['One','Two','Three','Four','Five','Six','Seven','Eight'

#Plotting the confusion matrix
plt.figure(figsize=(15,10))
sns.heatmap(cm5_df, annot=True)
plt.title('Confusion Matrix')
plt.ylabel('Actal Values')
plt.xlabel('Predicted Values')
plt.show()
```



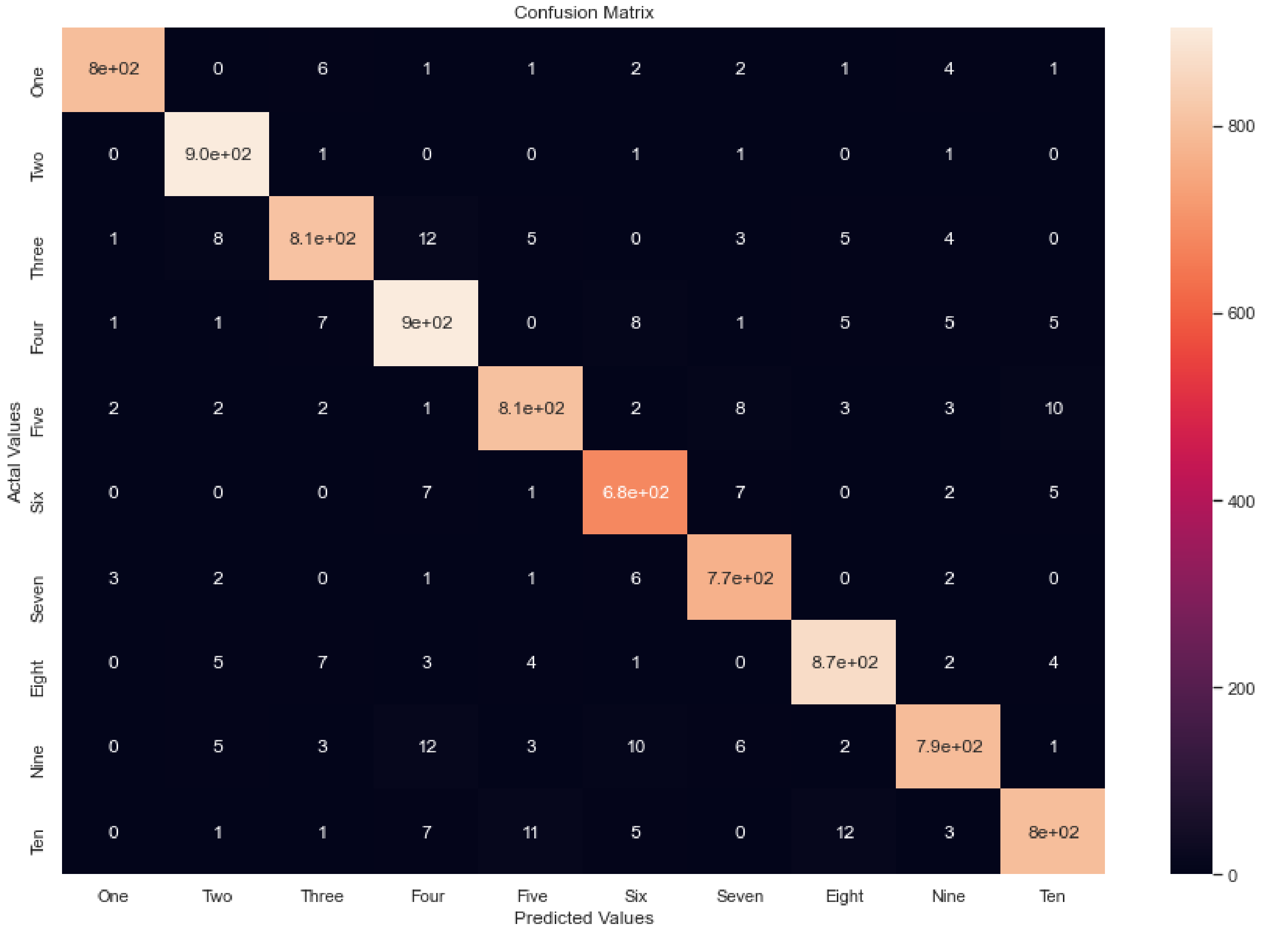
100 Nodes

In [226…

```
cm6 = confusion_matrix(y_test, mlp6_test_pred)

cm6_df = pd.DataFrame(cm6,
                     index = ['One','Two','Three','Four','Five','Six','Seven','Eight','
                     columns = ['One','Two','Three','Four','Five','Six','Seven','Eight'

#Plotting the confusion matrix
plt.figure(figsize=(15,10))
sns.heatmap(cm6_df, annot=True)
plt.title('Confusion Matrix')
plt.ylabel('Actal Values')
plt.xlabel('Predicted Values')
plt.show()
```



5 Layers

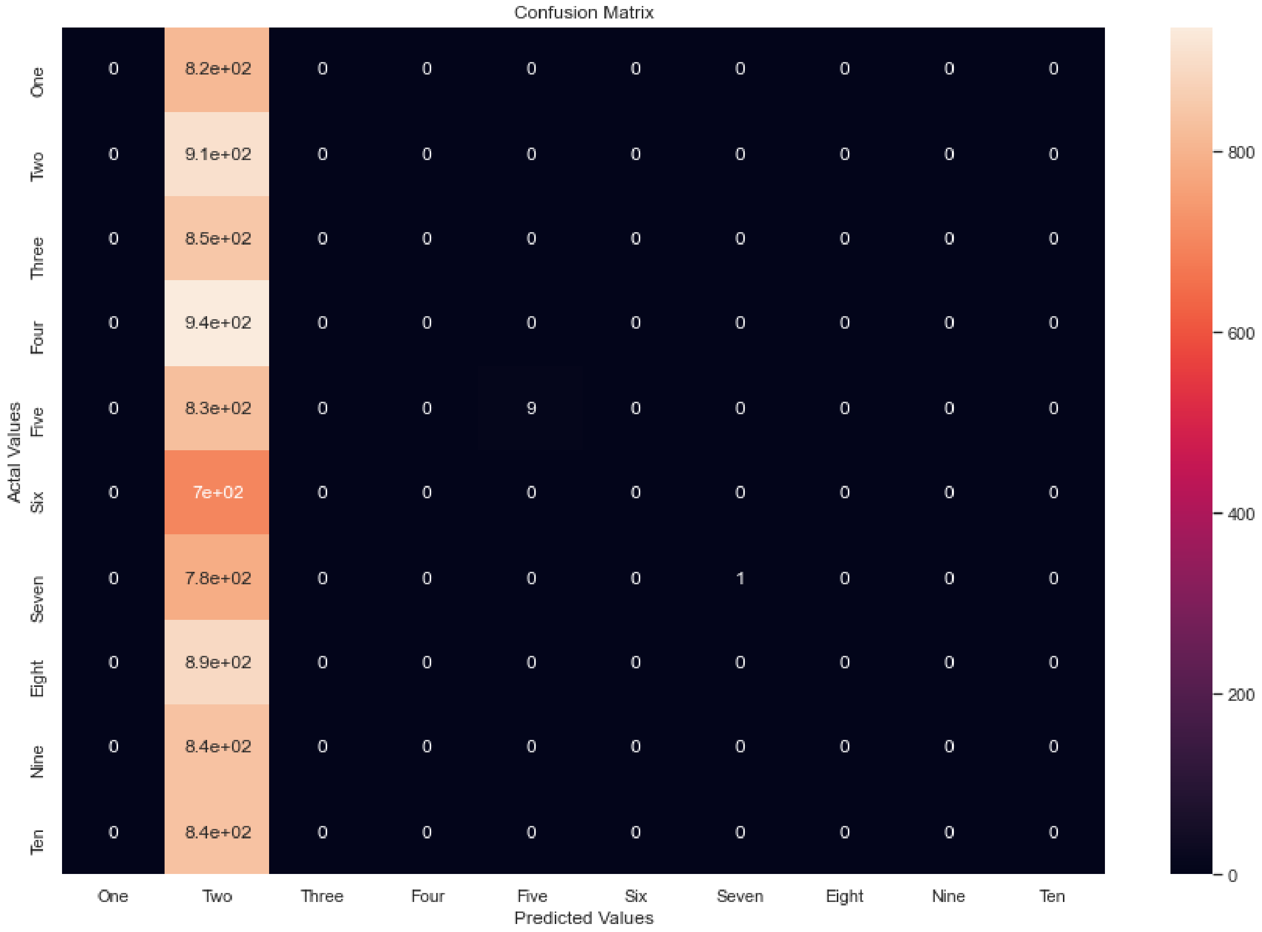
10 Nodes

In [227…

```
cm7 = confusion_matrix(y_test, mlp7_test_pred)

cm7_df = pd.DataFrame(cm7,
                     index = ['One','Two','Three','Four','Five','Six','Seven','Eight','
                     columns = ['One','Two','Three','Four','Five','Six','Seven','Eight'

#Plotting the confusion matrix
plt.figure(figsize=(15,10))
sns.heatmap(cm7_df, annot=True)
plt.title('Confusion Matrix')
plt.ylabel('Actal Values')
plt.xlabel('Predicted Values')
plt.show()
```



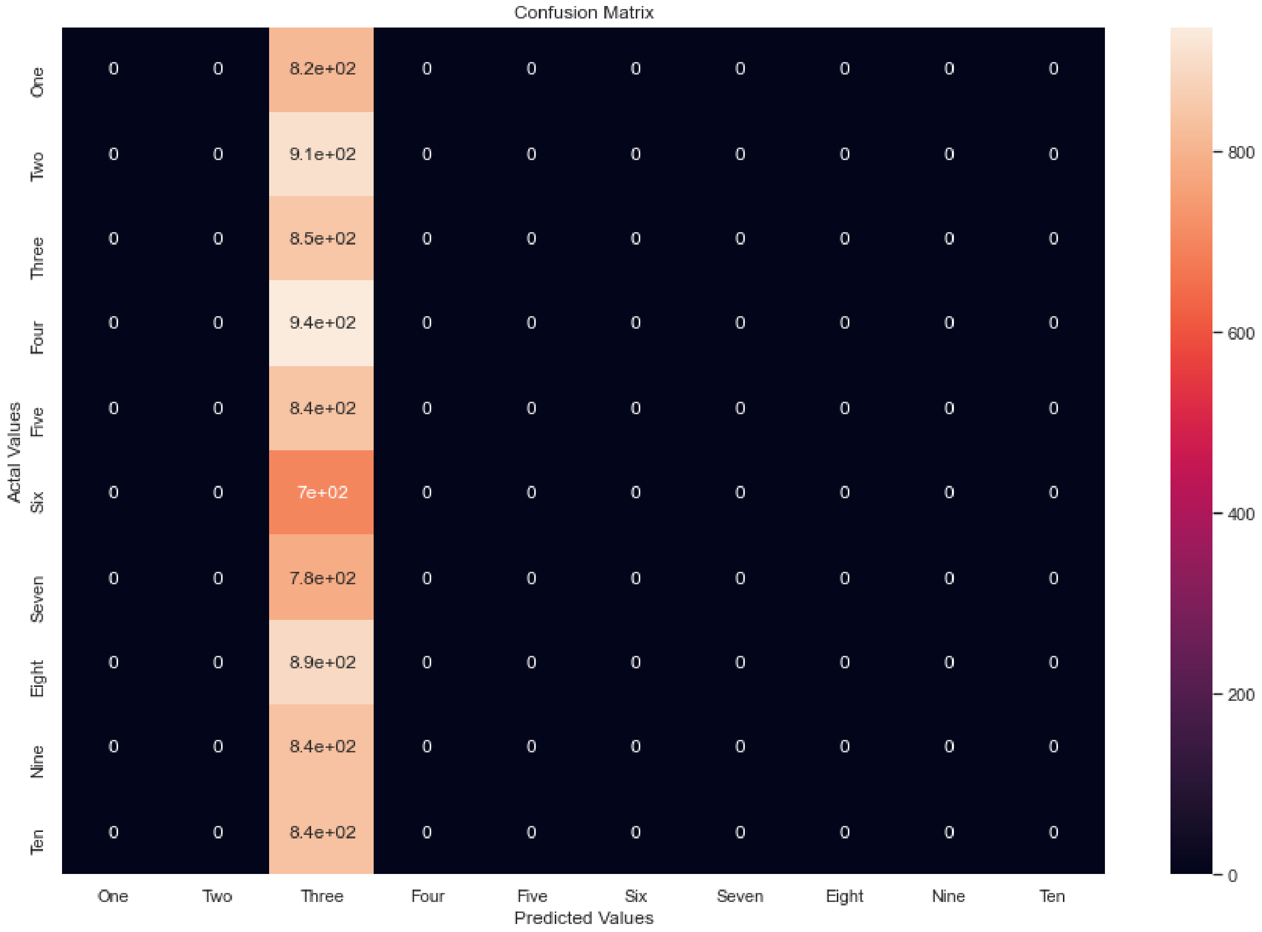
50 Nodes

In [228…

```
cm8 = confusion_matrix(y_test, mlp8_test_pred)

cm8_df = pd.DataFrame(cm8,
                     index = ['One','Two','Three','Four','Five','Six','Seven','Eight','
                     columns = ['One','Two','Three','Four','Five','Six','Seven','Eight'

#Plotting the confusion matrix
plt.figure(figsize=(15,10))
sns.heatmap(cm8_df, annot=True)
plt.title('Confusion Matrix')
plt.ylabel('Actal Values')
plt.xlabel('Predicted Values')
plt.show()
```



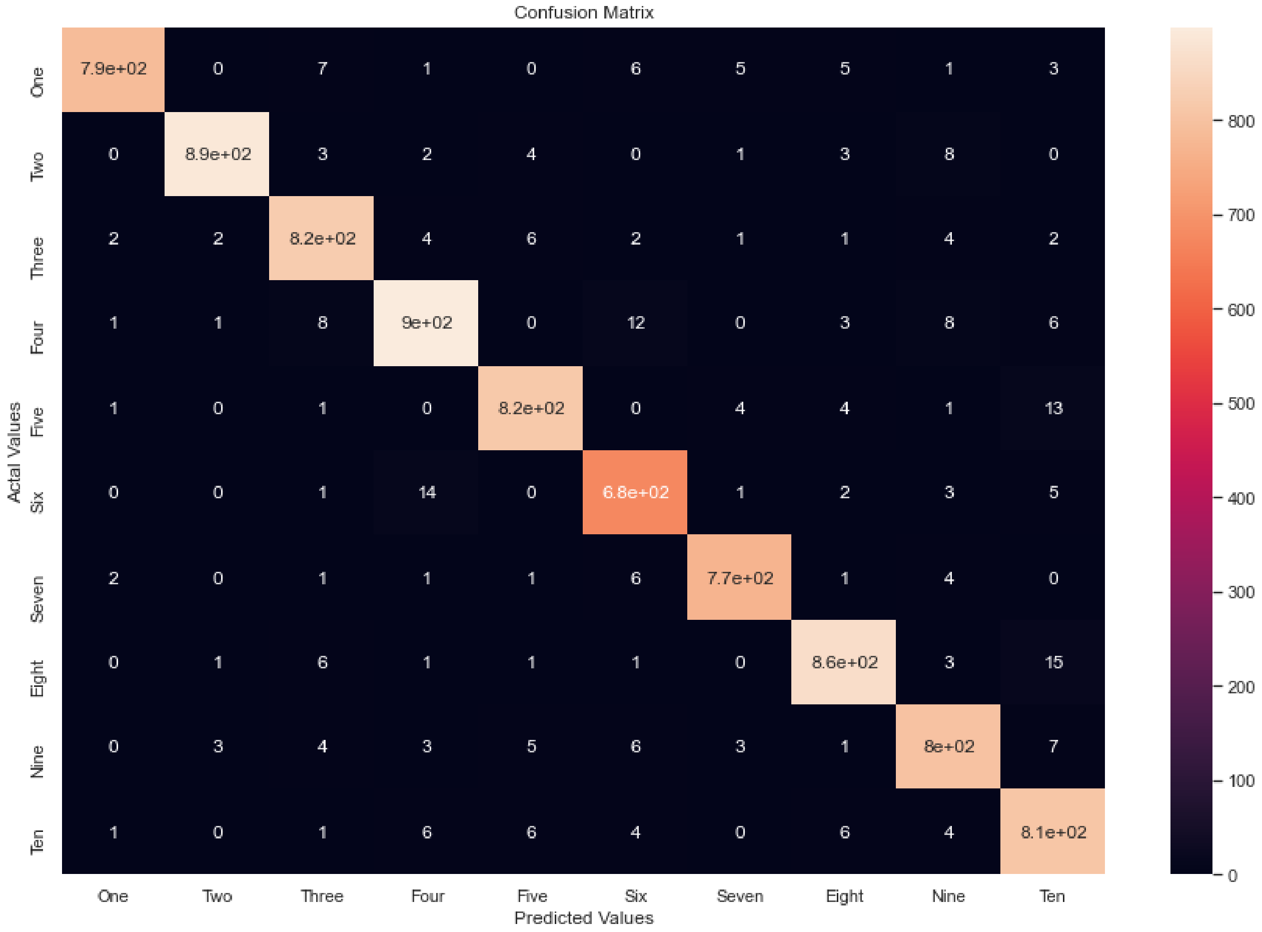
100 Nodes

In [229…

```
cm9 = confusion_matrix(y_test, mlp9_test_pred)

cm9_df = pd.DataFrame(cm9,
                     index = ['One','Two','Three','Four','Five','Six','Seven','Eight','
                     columns = ['One','Two','Three','Four','Five','Six','Seven','Eight'

#Plotting the confusion matrix
plt.figure(figsize=(15,10))
sns.heatmap(cm9_df, annot=True)
plt.title('Confusion Matrix')
plt.ylabel('Actal Values')
plt.xlabel('Predicted Values')
plt.show()
```



Testing for Kaggle Submisson

In [230…

```
#create dataframe using test data from kaggle
df_test = pd.read_csv("test.csv")
```

In [231…

```
len(df_test)
```

Out[231…

```
28000
```

Scale Data

In [232…

```
# Conversion to float
df_float = df_test.astype('float32')

# Normalization
X = df_float/255.0

X.head()
```

Out[232…

	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel774	pixel775
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0

	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel774	pixel775
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0

5 rows × 784 columns

Flatten Data

In [233…

```
X = X.to_numpy().reshape((len(X), -1))
X.shape
```

Out[233…

```
(28000, 784)
```

Test Models

In [234…

```
# Sequential Model 2
seq2_pred = model2.predict(X)
seq2_pred = np.argmax(seq2_pred, axis=1)
seq2_pred = pd.DataFrame(seq2_pred ,columns = ['Label'])
seq2_pred.insert(0, 'ImageId', range(1, 1 + len(X)))

seq2_pred.head()
```

```
875/875 [==============================] - 1s 876us/step
```

Out[234…

	ImageId	Label
0	1	2
1	2	0
2	3	9
3	4	4
4	5	3

In [235…

```
# Sequential Model 4
seq4_pred = model4.predict(X)
seq4_pred = np.argmax(seq4_pred, axis=1)
seq4_pred = pd.DataFrame(seq4_pred ,columns = ['Label'])
seq4_pred.insert(0, 'ImageId', range(1, 1 + len(X)))

seq4_pred.head()
```

```
875/875 [==============================] - 1s 979us/step
```

Out[235…

	ImageId	Label

	ImageId	Label
0	1	2
1	2	0
2	3	9
3	4	9
4	5	3

In [236…

```
# Model 7 (MLP model 3)
mlp3_pred = pd.DataFrame(mlp3.predict(X), columns = ['Label'])
mlp3_pred.insert(0, 'ImageId', range(1, 1 + len(X)))

mlp3_pred.head()
```

Out[236…

	ImageId	Label
0	1	2
1	2	0
2	3	9
3	4	9
4	5	3

In [237…

```
# Model 10 (MLP model 6)
mlp6_pred = pd.DataFrame(mlp6.predict(X), columns = ['Label'])
mlp6_pred.insert(0, 'ImageId', range(1, 1 + len(X)))

mlp6_pred.head()
```

Out[237…

	ImageId	Label
0	1	2
1	2	0
2	3	9
3	4	9
4	5	3

In [238…

```
# Model 12 (MLP model 9)
mlp9_pred = pd.DataFrame(mlp9.predict(X), columns = ['Label'])
mlp9_pred.insert(0, 'ImageId', range(1, 1 + len(X)))

mlp9_pred.head()
```

Out[238…

	ImageId	Label
0	1	2

	ImageId	Label
1	2	0
2	3	9
3	4	9
4	5	3

Download the Files

Leave these commented out unless downloading a final version.

In [239…

```
# seq2_pred.to_csv('seq2_pred-group_5_msds_422.csv', index=False)
# files.download('seq2_pred-group_5_msds_422.csv')
```

In [240…

```
# seq4_pred.to_csv('seq4_pred-group_5_msds_422.csv', index=False)
# files.download('seq4_pred-group_5_msds_422.csv')
```

In [241…

```
# mlp3_pred.to_csv('mlp3_pred-group_5_msds_422.csv', index=False)
# files.download('mlp3_pred-group_5_msds_422.csv')
```

In [242…

```
# mlp6_pred.to_csv('mlp6_pred-group_5_msds_422.csv', index=False)
# files.download('mlp6_pred-group_5_msds_422.csv')
```

In [243…

```
# mlp9_pred.to_csv('mlp9_pred-group_5_msds_422.csv', index=False)
# files.download('mlp9_pred-group_5_msds_422.csv')
```

In [244…

```
# %%capture
# !wget -nc https://raw.githubusercontent.com/brpy/colab-pdf/master/colab_pdf.py
# from colab_pdf import colab_pdf
# colab_pdf('Module7_Assignment_1.ipynb')
```