MNIST Digit Recognizer: Neural Networks

According to the Kaggle overview, “MNIST ("Modified National Institute of Standards and Technology") is the de facto “hello world” dataset of computer vision.” It is a classic dataset to use for measuring and benchmarking classification algorithms. In this research, we will identify correct digits from a dataset of tens of thousands of handwritten images. The dataset contains 700+ columns each indicating a pixel of the chart. We will run a completely-crossed design experiments with neural networks to make predictions on the test set. First, we performed basic exploratory data analysis to better understand the dataset. Figures 1-5 shows 785 columns and int values that range from 0 (black pixel) to 255 (white). Figure 6 shows that there is a fairly even distribution of labels from 0-9. There are no null values. Figures 7-9 show a few more graphs exploring the frequency plot of pixel values, top 10 pixels across the different labels, and then the density plot for the average pixel intensities of each label. These allow us to get a fuller idea of the dataset and what pixel values go into it. Finally, Figure 10 shows that 76 columns have constant values (all 255 or all 0). These will not contribute to any of the classification models we create so we drop these columns. The dataset now has 709 columns and is ready to be worked with.

After splitting the dataset into an 80/20 training/testing split we conducted our experiments. We used a completely crossed design to run three different experiments. The first experiment was without any standard scaling and tested 2 and 5 layers and 10 and 20 nodes. What this means is that we will train and fit neural network models that use 2 hidden layers and 10 nodes, 2 hidden layers and 20 nodes, 5 hidden layers and 10 nodes, and lastly 5 hidden layers and 20 nodes. We will get the training and testing accuracies of each of these models and then use this to create a model to use on the Kaggle testing data. We found that the model that performed the best was 5 hidden layers with 20 nodes. The results and multi-class confusion matrix are shown in figures 11-16. Though this had better results than our random forest classifier from last week, we knew we could improve upon it. So, next we ran the exact same experiment design with the same number of layers and neurons but first transformed the data using a standard scaler. These results can be seen in figures 17-20. Our Kaggle score for this round of experimentation, using a model with 2 hidden layers and 20 nodes, was 0.94264.

This is a great improvement from last week as well but we wanted to finally test the limits of increasing the layers and nodes before we run into different issues. When building neural networks there are a few common pitfalls to be wary of while designing them. The first is that with too many neurons some may not activate. This is not necessarily going to have negative effects on classification performance and accuracy but instead will result in excess in the model that we do not need. This is fine for a dataset like MNIST digit classifier but for huge datasets can increase the amount of time to run. This can also occur with too many hidden layers. Many hidden layers vastly increases the amount of time it takes to fit along with overfitting in some cases. Lastly, it is easy to run into what is called the vanishing gradient problem in which the gradient becomes so small the neurons stop backpropagating.

With that in mind, we ran one more experiment with 3 and 5 hidden layers and 20 and 30 neurons. As expected, all of the training accuracies were super high with 5 hidden layers and 30 neurons resulting in a perfect score (a sign of overfitting though the testing accuracy still was high at ~0.95). We tested a neural network model of 3 hidden layers and 30 neurons resulting in our highest Kaggle score yet of 0.95157 (figure 26). To test our theory that the 5 hidden layer, 30 neuron model overfit we also tested this on our Kaggle test data where we saw a slight regression and a score of 0.9498. A 2x2 completely crossed experiment design shows a great improvement in classification capabilities. In the future we can further fine tune these parameters to get even better scores.

Appendix

```
from google.colab import drive
drive.mount('/content/drive')

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn
%cd /content/drive/My Drive/
df = pd.read_csv('MNIST_Digit/train.csv')
df.head(5)
```

```
df = pd.read_csv('MNIST_Digit/train.csv')
df.head(5)
```

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	pixel774	pixel775	pixel776	pixel777	pixel778	pixel779	pixel780	pixel781	pixe
0	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	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	0	0	0	0	0	0	
3	4	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	

5 rows × 785 columns

Figure 1

```
# Check datatypes, all numeric data
data_type_counts = df.dtypes.value_counts()
print(data_type_counts)

int64 785 Name: count, dtype: int64
```

Figure 2

```
len(df.columns)

785
```

Figure 3

```
#Null Value Columns
nullseries= df.isna().sum()
```

```python
print(nullseries[nullseries > 0])
```

```
Series([], dtype: int64)
```

Figure 4

```python
# Describe the data
df.describe()
```

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	pixel774	pixel775	pixel776	pixel777	pixel778	pixel779
count	42000.000000	42000.0	42000.0	42000.0	42000.0	42000.0	42000.0	42000.0	42000.0	42000.0	...	42000.000000	42000.000000	42000.000000	42000.00000	42000.000000	42000.000000
mean	4.456643	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.219286	0.117095	0.059024	0.02019	0.017238	0.002857
std	2.887730	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	6.312890	4.633819	3.274488	1.75987	1.894498	0.414264
min	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.000000	0.000000	0.00000	0.000000	0.000000
25%	2.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.000000	0.000000	0.00000	0.000000	0.000000
50%	4.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.000000	0.000000	0.00000	0.000000	0.000000
75%	7.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.000000	0.000000	0.00000	0.000000	0.000000
max	9.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	254.000000	254.000000	253.000000	253.00000	254.000000	62.000000

8 rows × 785 columns

Figure 5

```python
#Create Histogram
import matplotlib.pyplot as plt

# Assuming 'df' is your DataFrame and 'Bankrupt?' is the column name
plt.hist(df['label'], bins=10, edgecolor='black') # Assuming binary data,
adjust 'bins' as needed
plt.xlabel('label')
plt.ylabel('Frequency')
plt.title('Histogram of Digits')
plt.show()
```

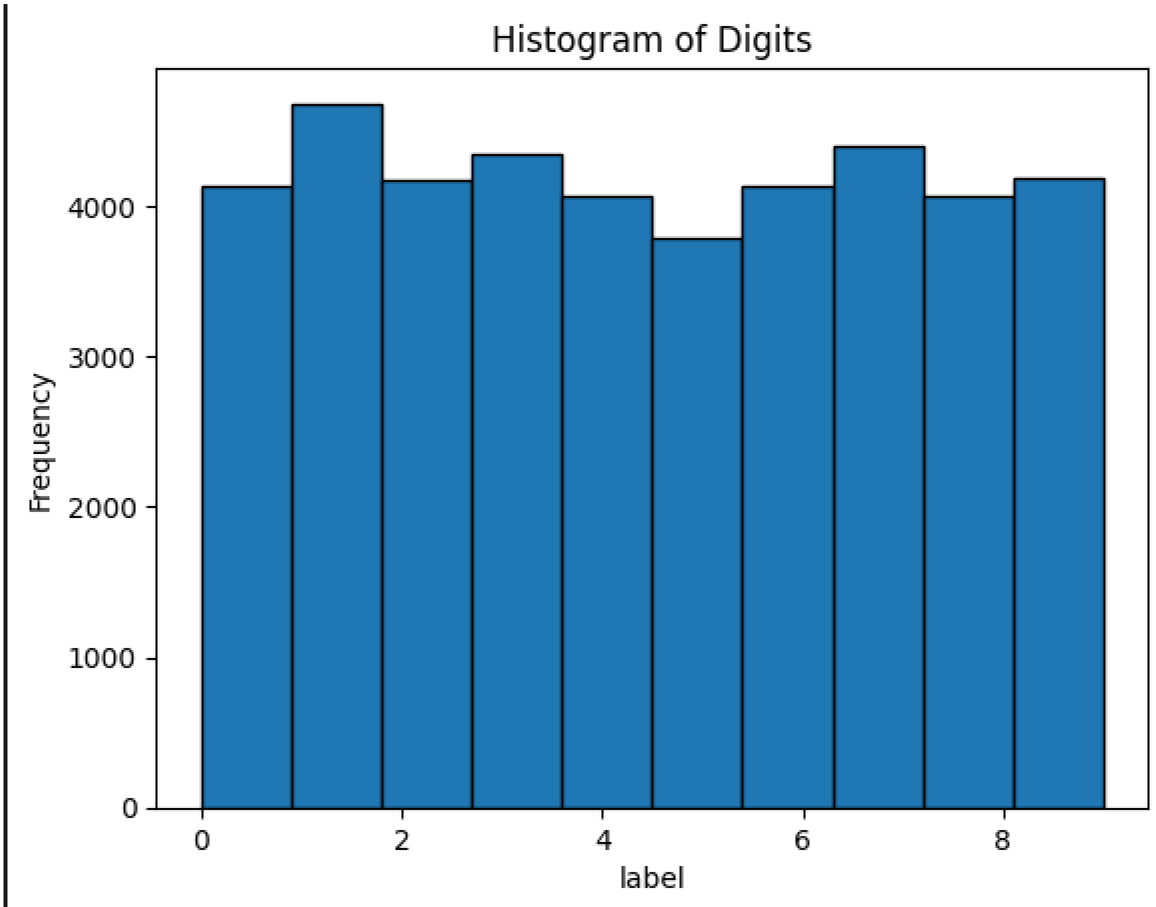

Figure 6

```
#create a heat map of all labels
columns_to_plot = df.columns # Add all relevant column names

# Flatten the selected columns into a single array
values = df[columns_to_plot].values.flatten()

# Plot the frequency of values
plt.figure(figsize=(10, 6))
plt.hist(values, bins=50, color='skyblue', edgecolor='black') # Adjust
bins as needed
plt.title('Frequency Plot of Pixel Values')
plt.xlabel('Pixel Value')
plt.ylabel('Frequency')
plt.grid(False)
plt.show()
```

```
#Majority of the pixels in the dataset are white while some are black, not many other pixel values.
```

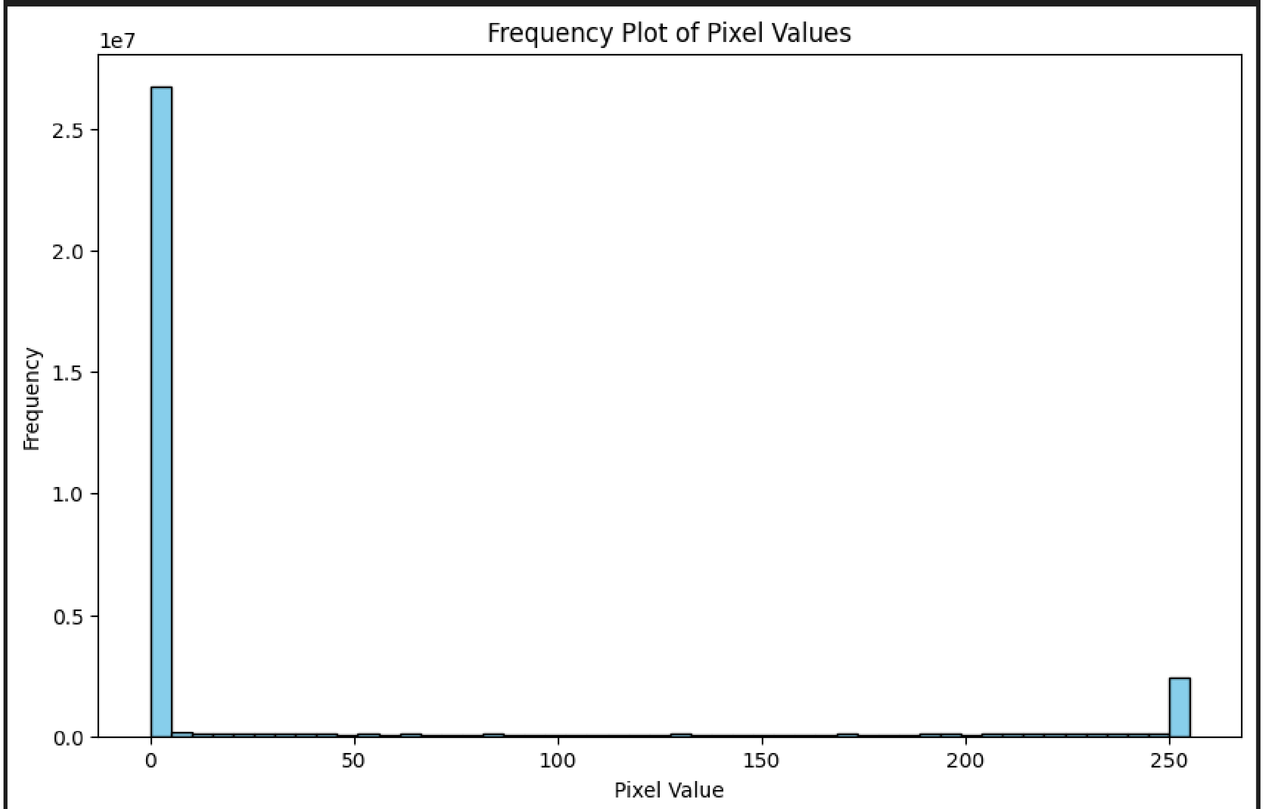

Figure 7

```
average_pixel_values_grouped = df.groupby('label').mean()
# Create a subplot with 10 plots (2 each row)
fig, axs = plt.subplots(5, 2, figsize=(15, 12))

for index, row in average_pixel_values_grouped.iterrows():
largest_10 = row.nlargest(10)
largest_10 = largest_10.sort_values(ascending=False)
ax = axs[index // 2, index % 2] # Calculate row and column index for subplot
ax.bar(largest_10.index, largest_10.values, color='skyblue')
ax.set_xlabel('Pixel Value')
ax.set_ylabel('Pixel Column')
```

```
ax.set_title('Top 10 Pixels of Label {}'.format(index))
ax.tick_params(axis='x', labelrotation = 45) # Rotate label


plt.tight_layout()
plt.show()

#Top 10 pixels are different across every labels
```

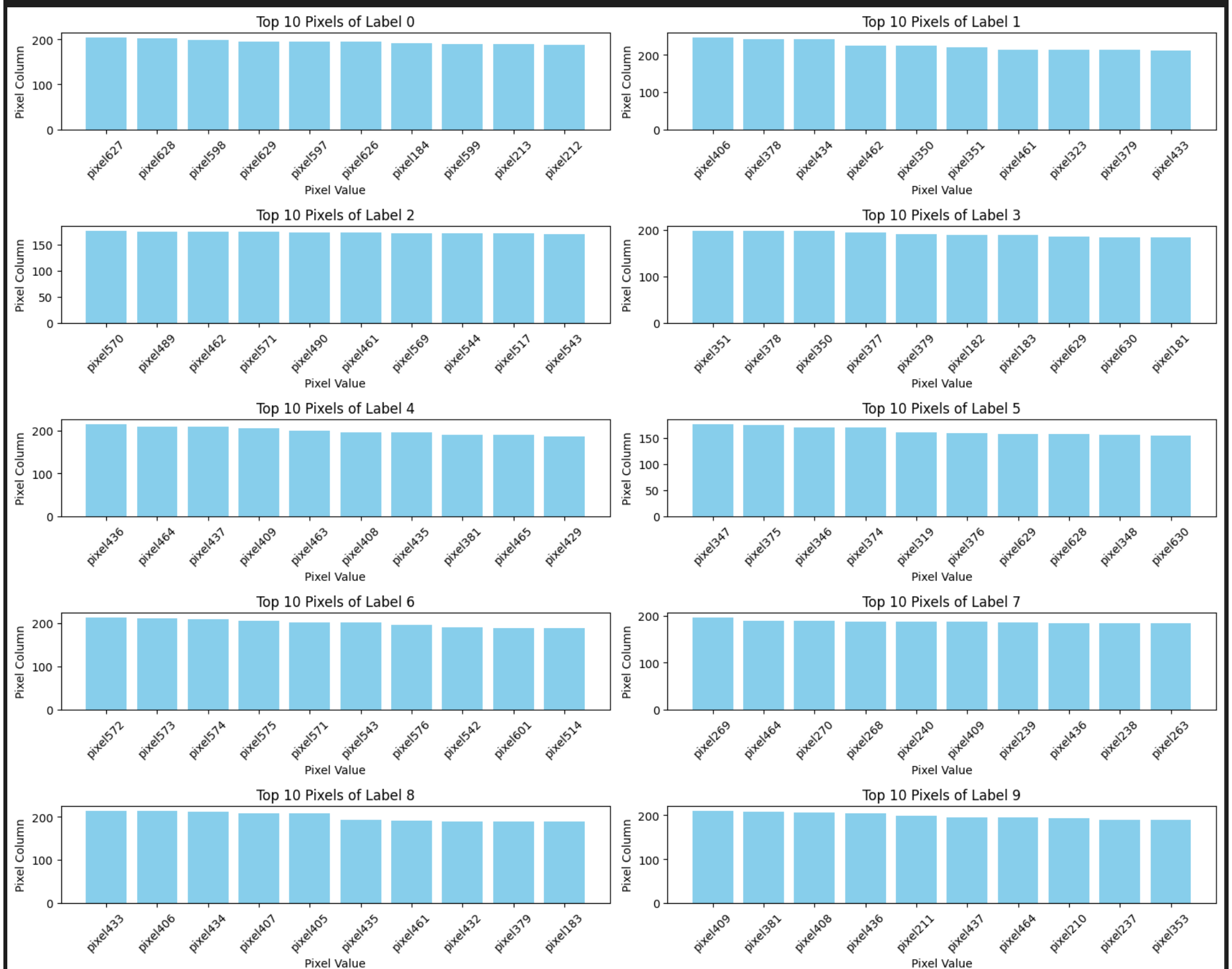

Figure 8

```
plt.figure(figsize=(10, 6))
for label in sorted(df['label'].unique()):
```

```
sns.kdeplot(df[df['label'] == label].mean(axis=1), label=label,
fill=False)
plt.title('Density Plot of Average Pixel Intensities by Label')
plt.xlabel('Average Pixel Intensity')
plt.ylabel('Density')
plt.legend(title='Label')
plt.grid(False)
plt.show()

#Most of label 1 has lower intensity than other labels; label 0 has the
highest pixel intensity
```

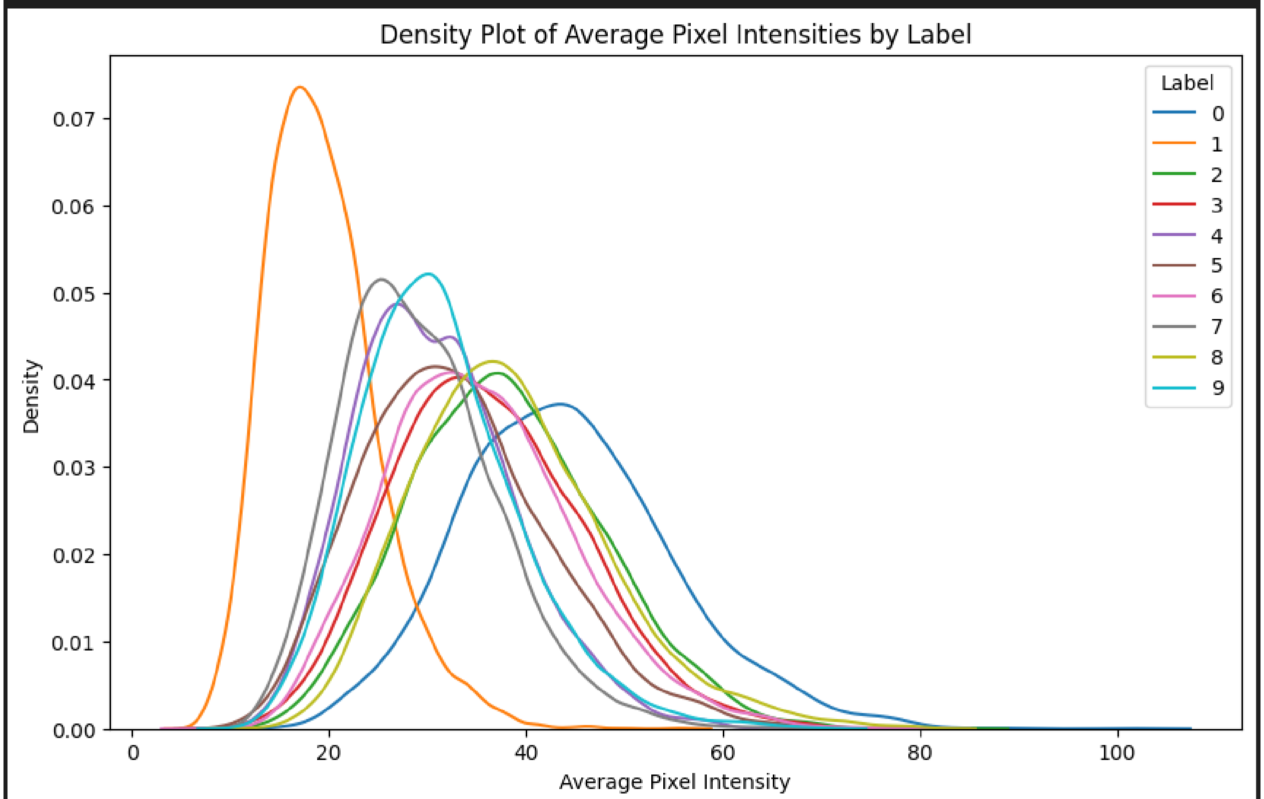

Figure 9

```
# Remove any columns with constant values. They won't contribute to the
classifiers
clean_df = df.copy()
black_pixels = []
white_pixels = []
print(len(df.columns))
```

```python
for pixel in df.columns:
if max(df[pixel]) == 0:
black_pixels.append(pixel)
if min(df[pixel]) == 255:
white_pixels.append(pixel)

clean_df = clean_df.drop(black_pixels, axis=1)
clean_df = clean_df.drop(white_pixels, axis=1)
print(len(black_pixels))
print(len(white_pixels))
print(len(clean_df.columns))
```

```
785 76 0 709
```

Figure 10

Split Training and Testing

```python
x = clean_df.drop(columns=['label'])
y = clean_df['label']

# Use K-Fold Later
from sklearn.model_selection import train_test_split
# Split the dataset into 80% train and 20% test
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,
random_state=42)
```

Neural Network Without Standard Scaling

```python
import time
from sklearn.neural_network import MLPClassifier

# Define experimental design
layers = [2, 5]
nodes = [10, 20]
```

```
# Prepare data (Assuming x_train and y_train are already loaded)

# Initialize lists to store results
time_taken = []
training_accuracy = []
testing_accuracy = []

# Iterate over each combination of layers and nodes
for layer in layers:
for node in nodes:
# Create neural network with current layer and node configuration
neural_net = MLPClassifier(hidden_layer_sizes=(node,) * layer,
max_iter=1000)

# Start training timer
start_time = time.time()

# Train neural network
neural_net.fit(x_train, y_train)

# Record time taken for training
time_taken.append(time.time() - start_time)

# Record training accuracy
training_accuracy.append(neural_net.score(x_train, y_train))

# Record testing accuracy
testing_accuracy.append(neural_net.score(x_test, y_test))

# Print results
for i in range(len(layers)):
for j in range(len(nodes)):
print(f"Layers: {layers[i]}, Nodes: {nodes[j]}, Time:
{time_taken[i*len(nodes)+j]}, Training Accuracy:
{training_accuracy[i*len(nodes)+j]}, Testing Accuracy:
{testing_accuracy[i*len(nodes)+j]}")

Layers: 2, Nodes: 10, Time: 313.3716700077057, Training Accuracy:
0.9114583333333334, Testing Accuracy: 0.8783333333333333
```

```
Layers: 2, Nodes: 20, Time: 126.3457510471344, Training Accuracy:
0.9778571428571429, Testing Accuracy: 0.9339285714285714
Layers: 5, Nodes: 10, Time: 285.35211634635925, Training Accuracy:
0.9396130952380952, Testing Accuracy: 0.8958333333333334
Layers: 5, Nodes: 20, Time: 176.1007297039032, Training Accuracy:
0.9904761904761905, Testing Accuracy: 0.9355952380952381
```

Figure 11

```
for layer in layers:
for node in nodes:
print((node,)* layer)
```

```
(10, 10)
(20, 20)
(10, 10, 10, 10, 10)
(20, 20, 20, 20, 20)
```

Figure 12

```
for i in range(len(layers)):
for j in range(len(nodes)):
print(f"Layers: {layers[i]}, Nodes: {nodes[j]}, Time:
{time_taken[i*len(nodes)+j]}, Training Accuracy:
{training_accuracy[i*len(nodes)+j]}, Testing Accuracy:
{testing_accuracy[i*len(nodes)+j]}")
```

```
Layers: 2, Nodes: 10, Time: 313.3716700077057, Training Accuracy:
0.9114583333333334, Testing Accuracy: 0.8783333333333333
Layers: 2, Nodes: 20, Time: 126.3457510471344, Training Accuracy:
0.9778571428571429, Testing Accuracy: 0.9339285714285714
Layers: 5, Nodes: 10, Time: 285.35211634635925, Training Accuracy:
0.9396130952380952, Testing Accuracy: 0.8958333333333334
Layers: 5, Nodes: 20, Time: 176.1007297039032, Training Accuracy:
0.9904761904761905, Testing Accuracy: 0.9355952380952381
```

Figure 13

```python
start_time = time.time()
neural_net = MLPClassifier(hidden_layer_sizes=(20, 20, 20, 20, 20),
max_iter=1000)
neural_net.fit(x_train, y_train)
print(f'It takes {time.time() - start_time} seconds to train 20 nodes 5
layers NetWork')

It takes 109.3866958618164 seconds to train 20 nodes 5 layers NetWork
```

Figure 14

```python
from sklearn.metrics import confusion_matrix, classification_report

# Testing Data confusion matrix
neural_pred = neural_net.predict(x_test)
test_conf_matrix = confusion_matrix(y_test, neural_pred)
class_names=[0,1] # name of classes
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
# create heatmap
sns.heatmap(pd.DataFrame(test_conf_matrix), annot=True,
cmap="YlGnBu" ,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Testing data confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

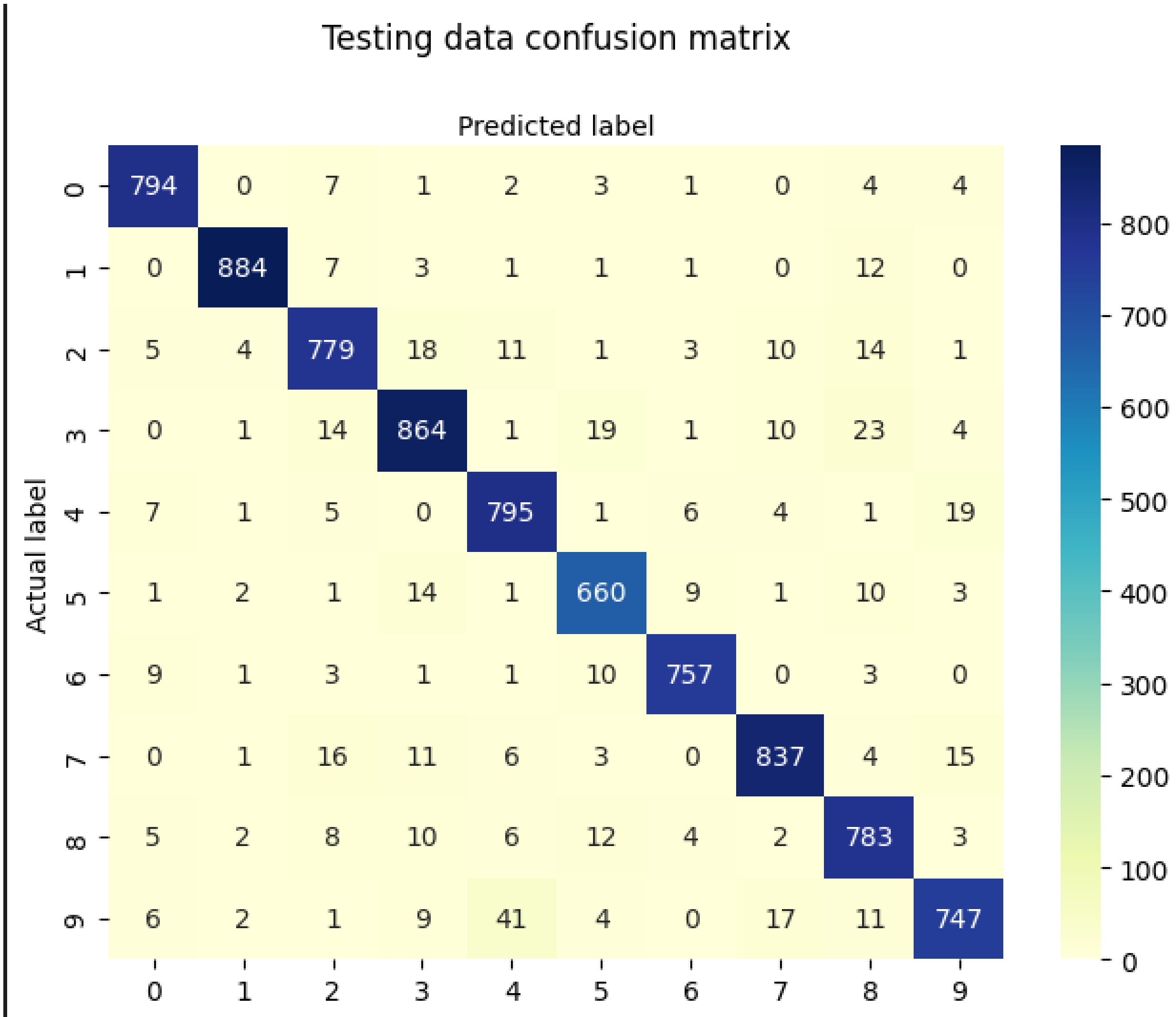

Figure 15

Prediction on Test Dataset

```
# Make prediction on test dataset
test_df = pd.read_csv('MNIST_Digit/test.csv')

# Remove any columns with constant values. They won't contribute to the
classifiers
clean_test_df = test_df.copy()
black_pixels = []
white_pixels = []
for pixel in df.columns:
```

```
if max(df[pixel]) == 0:
black_pixels.append(pixel)
if min(df[pixel]) == 255:
white_pixels.append(pixel)

clean_test_df = clean_test_df.drop(black_pixels, axis=1)
clean_test_df = clean_test_df.drop(white_pixels, axis=1)
print(len(clean_df.columns))
print(len(clean_test_df.columns)) # should be 1 less than clean_df because
there is no label column

709
708
```

Figure 16

```
predictions = neural_net.predict(clean_test_df)

imageId = pd.Series(range(1, len(predictions)+1)).astype(int)
result = {'ImageId': imageId, 'Label': predictions}
result_df = pd.DataFrame(result)
result_df.to_csv('MNIST_Digit/Neural_Network_Unscaled_prediction.csv',inde
x=False)
```

Neural Network Standard Scale

```
from sklearn.preprocessing import StandardScaler

# Standard Scale
scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)

import time
from sklearn.neural_network import MLPClassifier

# Define experimental design
layers = [2, 5]
nodes = [10, 20]
```

```
# Prepare data (Assuming x_train and y_train are already loaded)

# Initialize lists to store results
time_taken = []
training_accuracy = []
testing_accuracy = []

# Iterate over each combination of layers and nodes
for layer in layers:
for node in nodes:
# Create neural network with current layer and node configuration
neural_net = MLPClassifier(hidden_layer_sizes=(node,) * layer,
max_iter=1000)

# Start training timer
start_time = time.time()

# Train neural network
neural_net.fit(x_train_scaled, y_train)

# Record time taken for training
time_taken.append(time.time() - start_time)

# Record training accuracy
training_accuracy.append(neural_net.score(x_train_scaled, y_train))

# Record testing accuracy
testing_accuracy.append(neural_net.score(x_test_scaled, y_test))

# Print results
for i in range(len(layers)):
for j in range(len(nodes)):
print(f"Layers: {layers[i]}, Nodes: {nodes[j]}, Time:
{time_taken[i*len(nodes)+j]}, Training Accuracy:
{training_accuracy[i*len(nodes)+j]}, Testing Accuracy:
{testing_accuracy[i*len(nodes)+j]}")
```

```
Layers: 2, Nodes: 10, Time: 218.75375127792358, Training Accuracy:
0.9859821428571428, Testing Accuracy: 0.9082142857142858
Layers: 2, Nodes: 20, Time: 107.57854437828064, Training Accuracy:
0.9985416666666667, Testing Accuracy: 0.9458333333333333
Layers: 5, Nodes: 10, Time: 283.8109927177429, Training Accuracy:
0.9836309523809523, Testing Accuracy: 0.9082142857142858
Layers: 5, Nodes: 20, Time: 141.67477869987488, Training Accuracy:
0.9997619047619047, Testing Accuracy: 0.9432142857142857
```

Figure 17

```
for i in range(len(layers)):
for j in range(len(nodes)):
print(f"Layers: {layers[i]}, Nodes: {nodes[j]}, Time:
{time_taken[i*len(nodes)+j]}, Training Accuracy:
{training_accuracy[i*len(nodes)+j]}, Testing Accuracy:
{testing_accuracy[i*len(nodes)+j]}")
```


```
Layers: 2, Nodes: 10, Time: 218.75375127792358, Training Accuracy:
0.9859821428571428, Testing Accuracy: 0.9082142857142858
Layers: 2, Nodes: 20, Time: 107.57854437828064, Training Accuracy:
0.9985416666666667, Testing Accuracy: 0.9458333333333333
Layers: 5, Nodes: 10, Time: 283.8109927177429, Training Accuracy:
0.9836309523809523, Testing Accuracy: 0.9082142857142858
Layers: 5, Nodes: 20, Time: 141.67477869987488, Training Accuracy:
0.9997619047619047, Testing Accuracy: 0.9432142857142857
```


Figure 18

```
start_time = time.time()
neural_net = MLPClassifier(hidden_layer_sizes=(20, 20), max_iter=1000)
neural_net.fit(x_train_scaled, y_train)
print(f'It takes {time.time() - start_time} seconds to train 20 nodes 2
layers NetWork')
```

```
It takes 103.00234508514404 seconds to train 20 nodes 2 layers NetWork
```

```python
from sklearn.metrics import confusion_matrix, classification_report

# Testing Data confusion matrix
neural_pred = neural_net.predict(x_test_scaled)
test_conf_matrix = confusion_matrix(y_test, neural_pred)
class_names=[0,1] # name of classes
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
# create heatmap
sns.heatmap(pd.DataFrame(test_conf_matrix), annot=True,
cmap="YlGnBu" ,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Testing data confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

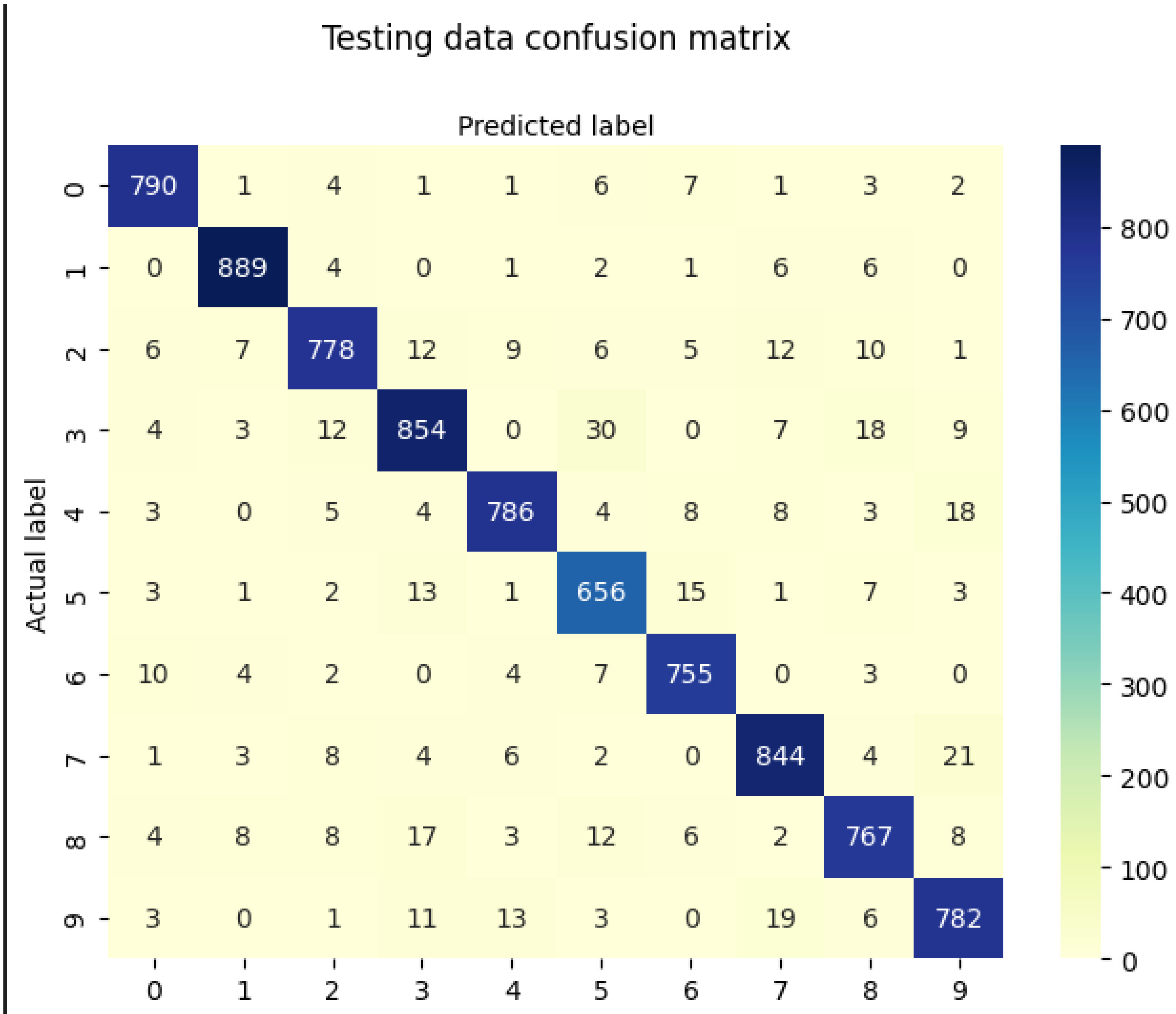

Figure 19

Prediction on Test Dataset

```
# Make prediction on test dataset
test_df = pd.read_csv('MNIST_Digit/test.csv')

# Remove any columns with constant values. They won't contribute to the classifiers
clean_test_df = test_df.copy()
black_pixels = []
white_pixels = []
for pixel in df.columns:
```

```
if max(df[pixel]) == 0:
black_pixels.append(pixel)
if min(df[pixel]) == 255:
white_pixels.append(pixel)

clean_test_df = clean_test_df.drop(black_pixels, axis=1)
clean_test_df = clean_test_df.drop(white_pixels, axis=1)
print(len(clean_df.columns))
print(len(clean_test_df.columns)) # should be 1 less than clean_df because
there is no label column

709
708
```

Figure 20

```
# Standard Scale
scaler = StandardScaler()
predict_scaled = scaler.fit_transform(clean_test_df)

predictions = neural_net.predict(predict_scaled)

imageId = pd.Series(range(1, len(predictions)+1)).astype(int)
result = {'ImageId': imageId, 'Label': predictions}
result_df = pd.DataFrame(result)
result_df.to_csv('MNIST_Digit/Neural_Network_scaled_prediction.csv',index=
False)
```

Neural Net Experiment 2

```
from sklearn.preprocessing import StandardScaler

# Standard Scale
scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)
```

```python
import time
from sklearn.neural_network import MLPClassifier

# Define experimental design
layers = [3, 5]
nodes = [20, 30]

# Prepare data (Assuming x_train and y_train are already loaded)

# Initialize lists to store results
time_taken = []
training_accuracy = []
testing_accuracy = []

# Iterate over each combination of layers and nodes
for layer in layers:
for node in nodes:
# Create neural network with current layer and node configuration
neural_net = MLPClassifier(hidden_layer_sizes=(node,) * layer,
max_iter=1000)

# Start training timer
start_time = time.time()

# Train neural network
neural_net.fit(x_train_scaled, y_train)

# Record time taken for training
time_taken.append(time.time() - start_time)

# Record training accuracy
training_accuracy.append(neural_net.score(x_train_scaled, y_train))

# Record testing accuracy
testing_accuracy.append(neural_net.score(x_test_scaled, y_test))

# Print results
for i in range(len(layers)):
for j in range(len(nodes)):
```

```
print(f"Layers: {layers[i]}, Nodes: {nodes[j]}, Time:
{time_taken[i*len(nodes)+j]}, Training Accuracy:
{training_accuracy[i*len(nodes)+j]}, Testing Accuracy:
{testing_accuracy[i*len(nodes)+j]}")
```

```
Layers: 3, Nodes: 20, Time: 89.13008999824524, Training Accuracy:
0.9999107142857143, Testing Accuracy: 0.9435714285714286
Layers: 3, Nodes: 30, Time: 55.43762469291687, Training Accuracy:
0.9999404761904762, Testing Accuracy: 0.9566666666666667
Layers: 5, Nodes: 20, Time: 108.07192993164062, Training Accuracy:
0.9999107142857143, Testing Accuracy: 0.9433333333333334
Layers: 5, Nodes: 30, Time: 68.57372498512268, Training Accuracy: 1.0,
Testing Accuracy: 0.9541666666666667
```

Figure 21

```
for i in range(len(layers)):
for j in range(len(nodes)):
print(f"Layers: {layers[i]}, Nodes: {nodes[j]}, Time:
{time_taken[i*len(nodes)+j]}, Training Accuracy:
{training_accuracy[i*len(nodes)+j]}, Testing Accuracy:
{testing_accuracy[i*len(nodes)+j]}")
```

```
Layers: 3, Nodes: 20, Time: 89.13008999824524, Training Accuracy:
0.9999107142857143, Testing Accuracy: 0.9435714285714286
Layers: 3, Nodes: 30, Time: 55.43762469291687, Training Accuracy:
0.9999404761904762, Testing Accuracy: 0.9566666666666667
Layers: 5, Nodes: 20, Time: 108.07192993164062, Training Accuracy:
0.9999107142857143, Testing Accuracy: 0.9433333333333334
Layers: 5, Nodes: 30, Time: 68.57372498512268, Training Accuracy: 1.0,
Testing Accuracy: 0.9541666666666667
```

Figure 22

```
start_time = time.time()
neural_net = MLPClassifier(hidden_layer_sizes=(30, 30, 30), max_iter=1000)
```

```
neural_net.fit(x_train_scaled, y_train)
print(f'It takes {time.time() - start_time} seconds to train 30 nodes 3 
layers NetWork')

It takes 59.0290744304657 seconds to train 30 nodes 3 layers NetWork
```

Figure 23

```
from sklearn.metrics import confusion_matrix, classification_report

# Testing Data confusion matrix
neural_pred = neural_net.predict(x_test_scaled)
test_conf_matrix = confusion_matrix(y_test, neural_pred)
class_names=[0,1] # name of classes
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
# create heatmap
sns.heatmap(pd.DataFrame(test_conf_matrix), annot=True, 
cmap="YlGnBu" ,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Testing data confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

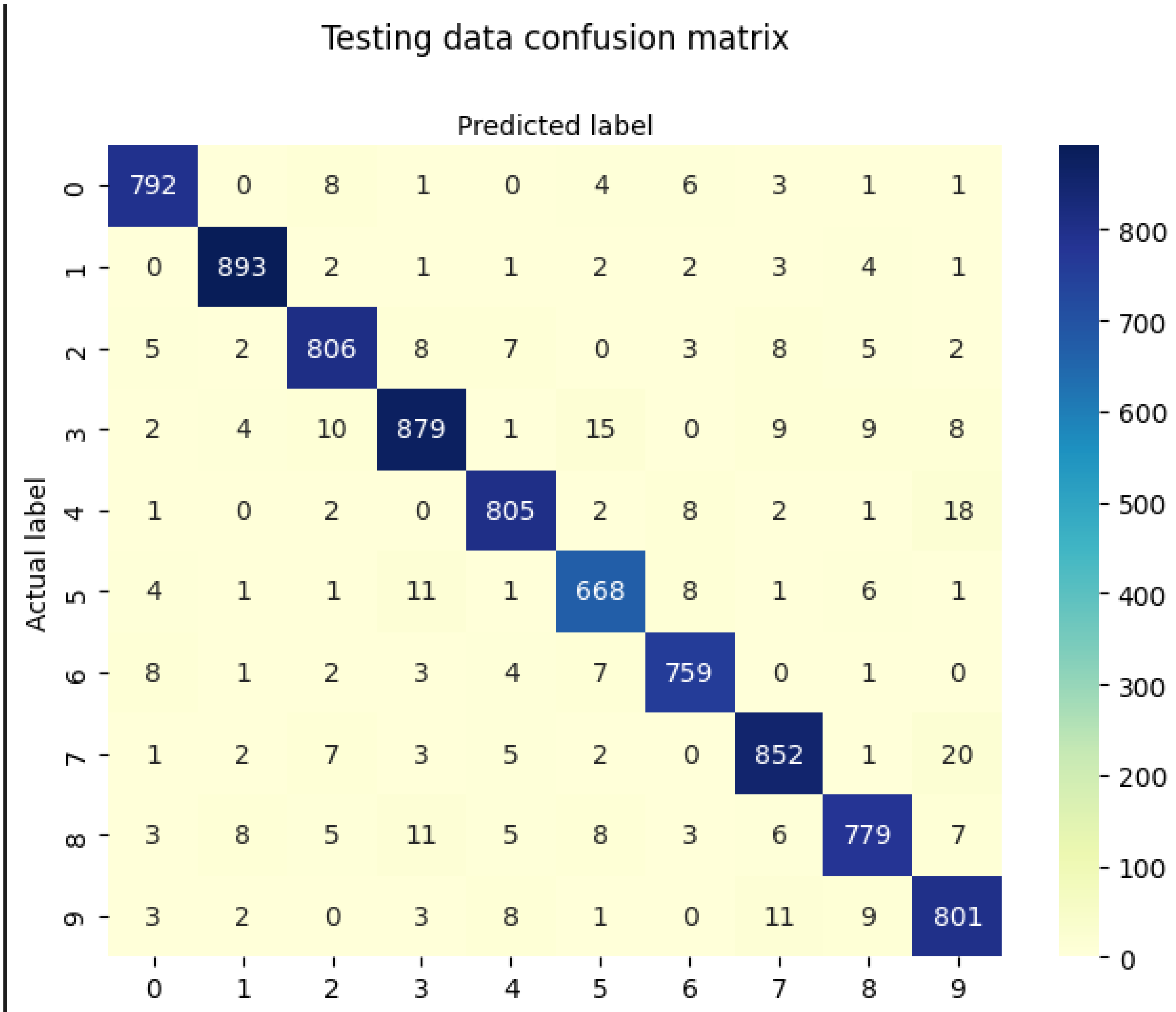

Figure 24

Prediction on Test Dataset

```
# Make prediction on test dataset
test_df = pd.read_csv('MNIST_Digit/test.csv')

# Remove any columns with constant values. They won't contribute to the
classifiers
clean_test_df = test_df.copy()
black_pixels = []
white_pixels = []
for pixel in df.columns:
```

```
if max(df[pixel]) == 0:
black_pixels.append(pixel)
if min(df[pixel]) == 255:
white_pixels.append(pixel)

clean_test_df = clean_test_df.drop(black_pixels, axis=1)
clean_test_df = clean_test_df.drop(white_pixels, axis=1)
print(len(clean_df.columns))
print(len(clean_test_df.columns)) # should be 1 less than clean_df because
there is no label column

709
708
```

Figure 25

```
# Standard Scale
scaler = StandardScaler()
predict_scaled = scaler.fit_transform(clean_test_df)

predictions = neural_net.predict(predict_scaled)

imageId = pd.Series(range(1, len(predictions)+1)).astype(int)
result = {'ImageId': imageId, 'Label': predictions}
result_df = pd.DataFrame(result)
result_df.to_csv('MNIST_Digit/Neural_Network_scaled_2_prediction.csv',inde
x=False)
```

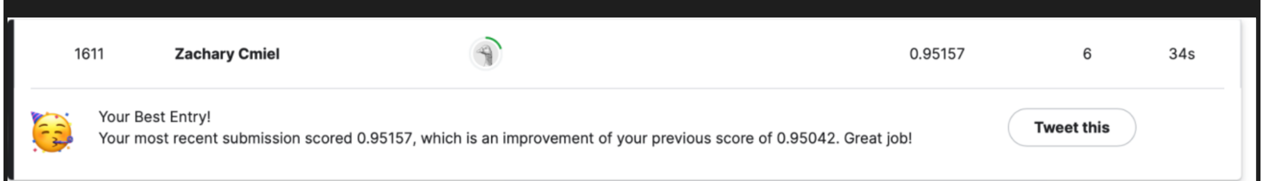

Figure 26