Analysis of K-Nearest Neighbors and Neural Networks on MNIST Fashion Dataset

Vernicia Gage
R I-P
AN

Introduction

As technology has improved, machine vision has become more and more prominent in data analytics. Computer vision is commonly used in data classification problems, and the results are then analyzed. In this project we use machine visualization to classify sets of images of clothes, we have compared different methods of machine learning as to explore which method would be best. For the project data set, we have used a MNIST data set, which consists of several gray scale 28X28 pixel images to test our code and different methods. The two methods that we decided to compare are KNN, and neural networks, which are both commonly used methods in machine learning.

There is the concept that biological systems can be represented as networks that have interactions and entity relationships in biology. These networks are similar to process flowcharts, where steps occur based on entity interactions and relations. One example is the brain, where information is flows between neurons and the brain makes decisions based on this information. This idea is then transferred over to [Artificial] Neural Networks. A neural network has "simple processing nodes" that contain layers that takes in input data, processes it through the layers, making decisions to define the data, then returns the output.

K-Nearest Neighbors is a machine learning method that compares a data point to the K nearest data points to it. KNN assumes that points that are similar will be near each other in a dataset and that if an unknown point is introduced in a dataset, it can be classified based on the classification of the points closest to it.

Using this information, we set out to apply these models to the Fashion MNIST dataset.

Problem Statement

The MNIST dataset is a consists of 60,000 28x28 grayscaled images that consist of different clothes types. Each image contains an article of clothing that can be assigned to one of the ten following categories:

1. T-shirt
2. Trouser
3. Pullover
4. Dress
5. Coat
6. Sandal
7. Shirt
8. Sneaker
9. Bag
10. Ankle Boot

If this were a case where the dataset contained a lesser number of images, it could be possible for the information to be categorized manually. However, because this specific data set contain 60,000 images, categorizing each image by hand would take a significantly longer time to complete and has a higher chance of mistakes relating to human error. In cases that involving categorizing larger sets of data, it is best to use some type of program that was trained to work with said data and has the ability to correctly sort its information. The benefit of this would time reduction of the sorting/categorization along with higher rates of accurately categorizing the data (if the program was trained and set up correctly in relation to the data).

As mentioned previously, we are provided the MNIST dataset which is made up of 60,000 greyscaled images of different clothes. For this assignment, we must develop a program that has the ability to read and categorize the datasets image under its specific clothing type In order to attain this goal we must build a code that can successfully categorize and input image of clothing from MNIST and determine the best set of parameters for its categorization filters that provide the highest accuracy level in terms of correct categorization.

Methodology

Initially we tried running our KNN and MLP codes on google colab. Fortunately, we were able to successfully develop and run the code for the MLP classification. KNN on the other hand was not cooperating well on google colab and produce multiple technological issues. The main one being error runtime which resulted in us redoing multiple runs. This was very time consuming considering that some of the KNN lines of code (such as the cell containing startifiedkfold and gridsearch) typically take longer amounts of time to run compared to the lines of code that make up of the MLP classification.

In this project, the 2 codes we are requested to work with are KNN and MLP. KNN (K-nearest neighbor) is a classification method in which the “neighbors” surrounded by a portion of data are analyzed. After this analysis, it develops a label for the lone piece of data based on an average of the most common label found amongst its neighbors. The other method classification we are using, the MLP, involves applying data to multiple hidden layers that analyze specific traits of data. Once the data is worked the the layers, a final outcome is calculated based on what the filters gather and convert that datas traits into the best fit category.

We started off the code by implementing the necessary packages that would be beneficial toward importing the data, training it(80/20), and building the code overall. This step is the most important because the imports are what allows most of our cells to run. If a sell requires and input to programmed prior to it’s run, then it will be unable to function. This is something we had to be very careful of when transitioning out KNN data from google colab to jupyter python. Unlike google colab, some imports of Jupyter python are not automatically installed. For example, we had to apple a pip install code for keras and seaborn since they are not pre-included in our jupyter python notebooks. As soon as we had all the imports set up and installed, we were ready to prepare and train the MNIST data for further coding.

As mentioned previously, some lines of code took longer times to run. To improve the speed of the data we incorporated a preprocessing technique called flattening. When large data sets are flattened, they are converted to a narrower single dimensional format. This is extremely beneficial and useful for codes that involve categorizing large sets of visual data. Aside from flattening, we did not do much to alter the MNIST data set in terms of null values and colorization. The data provided was made solely of visual components, based on this we concluded that it was unnecessary to perform a null look up function due to it only being able to be applied to numerical dataset. In some of our labs, we experimented with converting grayscaled images to other scales such as rbg and other color scales. This process was also

skipped in our code since the purpose of the project was to categorize data. Categorization codes can still run and function well with a given, therefore including a line of code that performs a change in color scale would be an unnecessary extra step.

In the early steps of MLP and KNN we performed a random seed processing code, specifically a random seed of 42. This specific combination of random samples remains the same for every single testing and training run we do in its current coding session. Therefore, our overall coding system will rely on the same 42 numbers generated in the beginning of our classification coding system (unless the code is completely restarted) The random seed 42 code creates a random sample of 42 numbers connected to our data set. The purpose behind this randomization is to ensure that our random code is able to reproduce the correct/valid results the same set of numbers/inputs on other systems. Meaning, if our code can successfully run any random input in our machine, than it can do so on others. In part of our code, we performed a 3-fold cross validation. For our crossvalidation, we split the model trained data into 3 sets where further training and validation are applied. In our case it works by splitting the data into 3 different folds, each fold applied is used as testing data while the surrounding folds function as training data. In our case we have 11 points of data(candidates) running for 3 folds, as a result all 11 points will simultaneously experience fitting with each fold giving us a total of each 33 fits. This is basically to confirm/validate that our model is performing and running the way it is supposed to and helps further ensure that all data sets are. This throughout analysis ensures that every single data point in our code has experienced some type of testing/training which is necessary for applying it to our KNN and MLP methods.

Before we begin our analysis predictions, mean test/train scores, error values, and other analytical behaviors of our KNN and MLP models regarding MNIST, we must ensure that we have the appropriate parameter values in our filters to perform classification. Each individual parameter must be finetuned and adjusted to a specific value that promotes the best possible accuracy of result production. Fixing these parameters to the appropriate values allows the input data to be correctly analyzed through each filter/hidden layer of the code. These layers are where the traits of the data are gathered and combined until it reaches a point where a classification for it can be recognized and assigned. Making sure the parameters being set up correctly is very crucial in terms of making sure our code can correctly categorize data for our KNN and MLP method. However, doing this by hand can be tedious, take a long time, and can encounter human error. To avoid these problems, we applied a line of gridsearch to our code. This gridsearch analysis all possible combinations of parameters and returns the best set of them that provides the highest functioning accuracy for our code. Once these parameters are found and the necessary data and import prepping steps are performed, we can move further in running our developed KNN and MLP classification code in application to the MNIST dataset.

Results Neural Network

Two different machine learning algorithms were compared when developing the best model for analyzing the data: a neural network and K-Nearest Neighbors classification algorithm. The SciKit Learn library was used for both models, using the MLPClassifier for the neural network and the KNeighborsClassifier for the KNN classification. Some hyperparameters for the models

were left constant while others were adjusted, this is to reduce the time that we are testing hyperparameters as both methods have long runtimes.

When considering hyperparameter adjustment for a neural network there are many to choose from such as number of hidden layers and learning rate. For the purpose of our analysis, the defaults of all the hyperparameters were left while only the activation function, alpha (L2 penalty) and hidden layer sizes were adjusted. Values for the adjusted hyperparameters are listed below in Figure 1.

Adjusted Hyperparameters	
Activation Function	tanh, relu
Alpha (L2 Penalty)	0.0001, 0.05
Hidden Layer Sizes	1, 5, 10, 50

Figure 1: Table of hyperparameters adjusted in the Neural Network model

When running the neural network, there are 16 hyperparameter combinations, all can be seen along with graphs of their training and test accuracy in Appendix D and Appendix E. The mean training score and the mean testing score for each model was graphed and the mean testing scores of all the models were graphed together for comparison. From Figure 2, it can be shown that the line belonging to activation function tanh, at alpha equal 0.0001 and with 50 hidden layers appears to have the best accuracy.

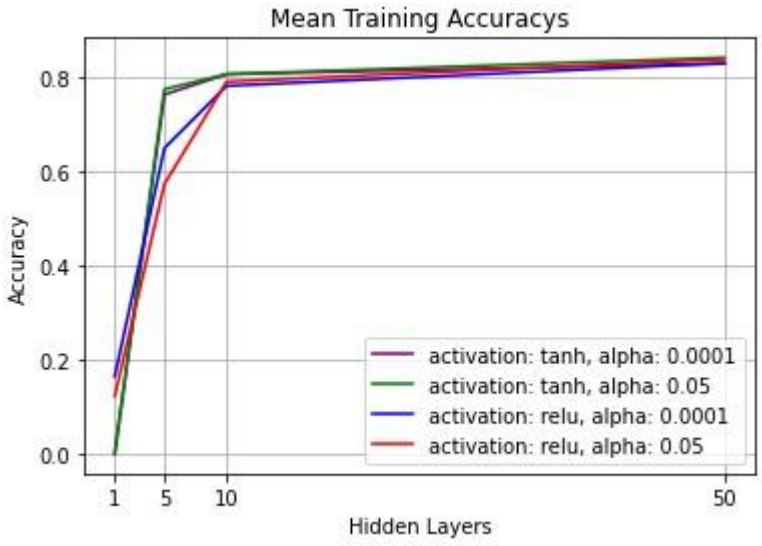


Figure 2: Graph of Mean Training Accuracies displaying model performance at different hidden layer values

For this model, the best combination of hyperparameters was found to be use the tanh activation function, an alpha value of 0.05, and a hidden layer size of 50. When using these values, most classes have over 78% of the images correctly classified except for class 6 which has 58% of the images correctly classified. The confusion matrix for the best model is shown below in Figure 3.

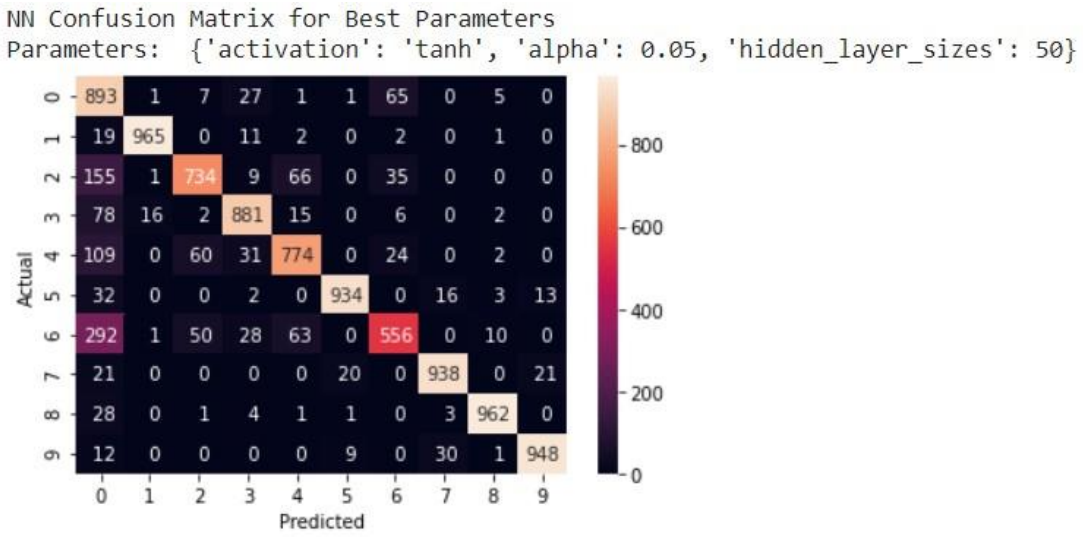


Figure 3: Confusion matrix of the Neural Network model for best parameters

K-Nearest Neighbors

The second machine learning algorithm that was analyzed when developing the best model for analyzing the data was K-Nearest Neighbors classification algorithm. The Sci-Kit Learn library's KNeighborsClassifier was used for the KNN classification. The only hyperparameter that was adjusted for the KNN model was the number of neighbors.

The number of neighbors that were used in the algorithm are the numbers 1 – 11.

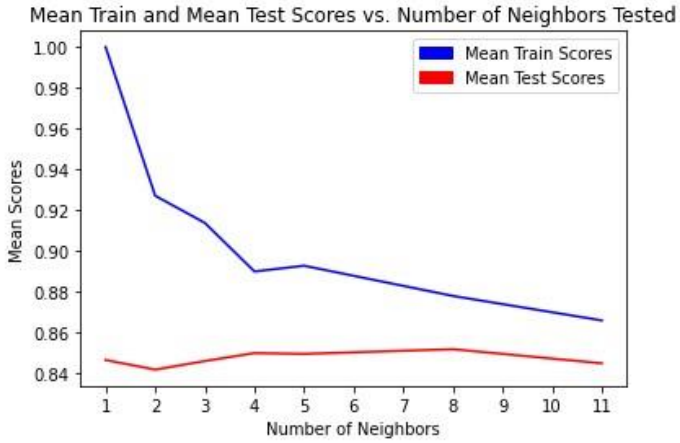


Figure 4: Graph of Mean Training and Test Accuracies displaying model performance

From Figure 4, it can be shown that for this model, the best combination value of k is 8. When using these values, most classes have over 80% of the images correctly classified except for class

6 which has 60% of the images correctly classified. The confusion matrix for the best model is shown below in Figure 5.

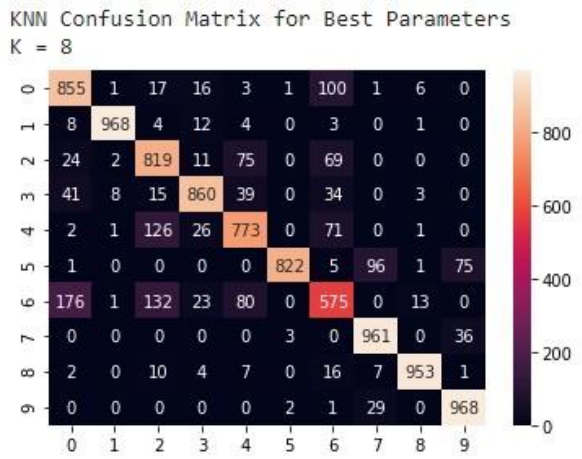


Figure 5: Confusion matrix of the KNN model for best parameters

Items that were the most challenging to classify were the pieces of clothing that went on the torso, such as shirts, and coats. The reason there is a high misclassification among class 0 predictions is because the three classes that have the highest amount of those predicted to be class 0 are pullovers, coats, and shirts. Class 0 (t-shirts) and the rest of those highly predicted to be class 0 are all clothing that is worn on the torso, thus have similar shapes. This can cause wrong predictions among those predicted to be class 0.

The neural network took less time to run at 121 mins to train. The KNN training time is 858 mins and it had to be split up in batches to run without timeout errors from Google Colab. **Conclusion**

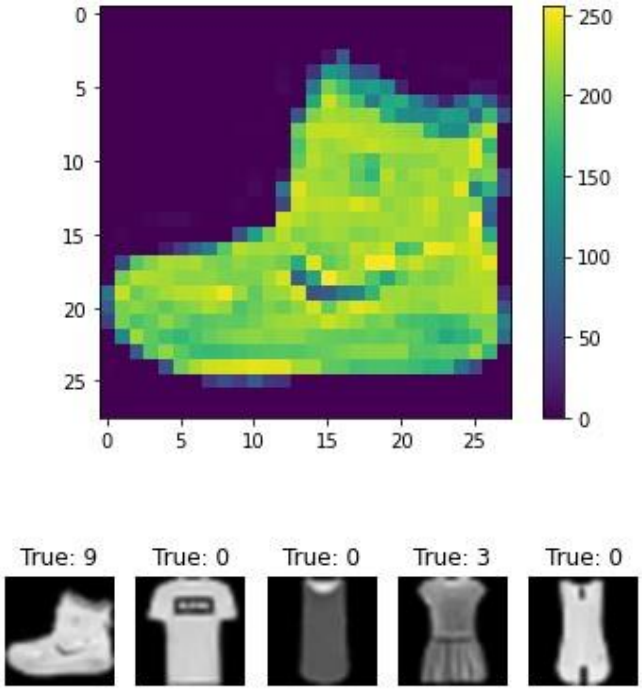
To summarize, we have used several machine learning techniques to classify images of clothes into separate categories, since classifying them by hand would not be efficient. We compared two machine learning methods to determine which one would be best, neural networks or KNN. After analyzing the results, we were able to conclude that the method that yielded highest accuracy was the KNN method. The best performing model for the neural network had a testing accuracy of 83%, while the KNN had a test accuracy of 85%. For future improvement of the project, we would recommend that other methods are explored to see if any yield any higher accuracy than the ones that have already been tested.

References

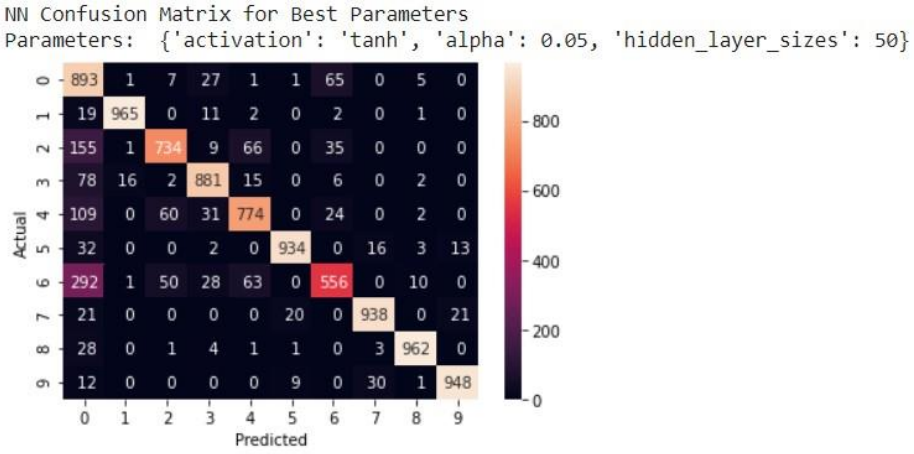
1. https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414
2. https://www.brainfunction.edu.au/research/research-themes/neural-circuits/
3. https://towardsdatascience.com/understanding-neural-networks-19020b758230 4. https://www.ebi.ac.uk/training-beta/online/courses/network-analysis-of-proteininteraction-data-an-introduction/network-analysis-in-biology/
5. Piccinini, G. The First Computational Theory of Mind and Brain: A Close Look at Mcculloch and Pitts's "Logical Calculus of Ideas Immanent in Nervous Activity". Synthese 141, 175–215 (2004). https://doi.org/10.1023/B:SYNT.0000043018.52445.3e
6. https://www.sciencedirect.com/science/article/pii/S0898122110002555

Appendices

Appendix A: Output example of Fashion mnist files



Appendix B: Neural Network Confusion Matrix



Appendix C: Neural Network Classification Report

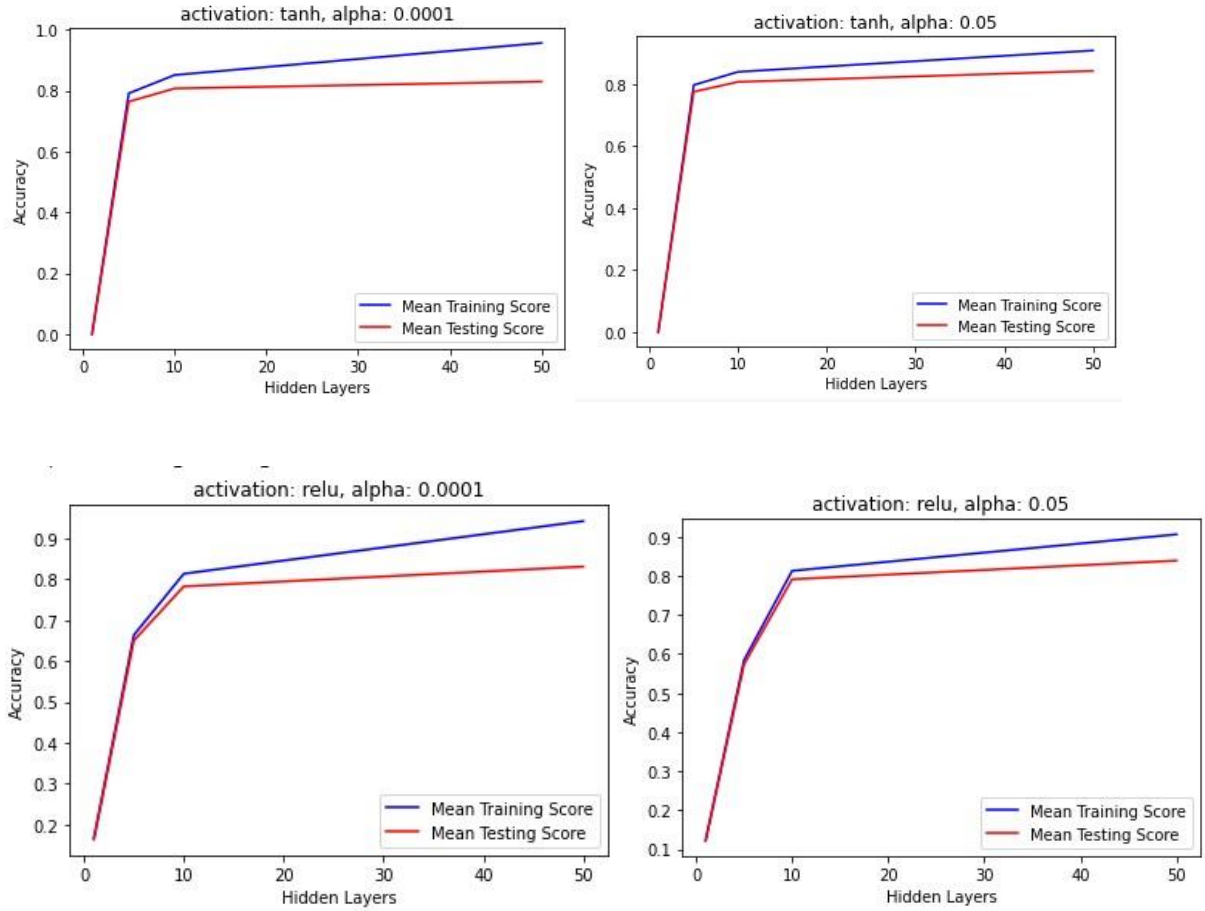
NN Classification Report

	precision	recall	f1-score	support
0	0.86	0.78	0.82	1000
1	0.98	0.96	0.97	1000
2	0.86	0.74	0.79	1000
3	0.88	0.89	0.88	1000
4	0.81	0.79	0.80	1000
5	0.97	0.93	0.95	1000
6	0.77	0.58	0.66	1000
7	0.95	0.94	0.95	1000
8	0.97	0.96	0.97	1000
9	0.96	0.95	0.96	1000
micro avg	0.90	0.85	0.88	10000
macro avg	0.90	0.85	0.88	10000
weighted avg	0.90	0.85	0.88	10000
samples avg	0.85	0.85	0.85	10000

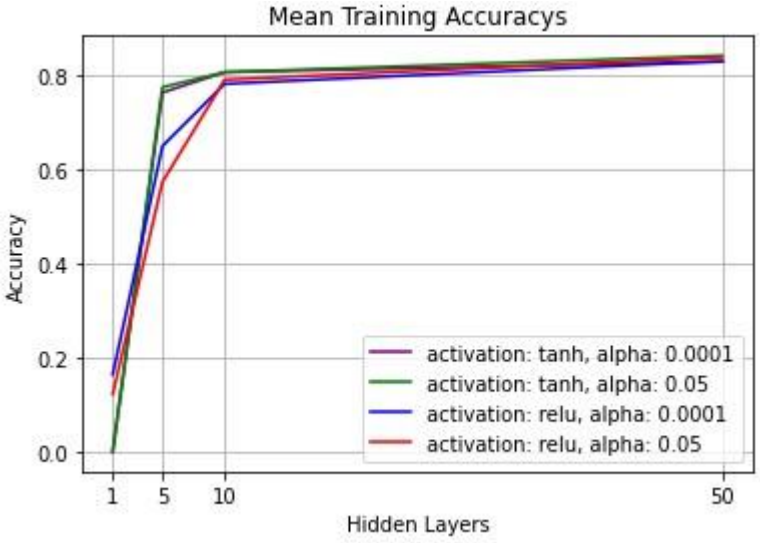
Appendix D: Neural Network Parameter Combinations

```
NN Parameters
[{'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_sizes': 1},
 {'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_sizes': 5},
 {'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_sizes': 10},
 {'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_sizes': 50},
 {'activation': 'tanh', 'alpha': 0.05, 'hidden_layer_sizes': 1},
 {'activation': 'tanh', 'alpha': 0.05, 'hidden_layer_sizes': 5},
 {'activation': 'tanh', 'alpha': 0.05, 'hidden_layer_sizes': 10},
 {'activation': 'tanh', 'alpha': 0.05, 'hidden_layer_sizes': 50},
 {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': 1},
 {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': 5},
 {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': 10},
 {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': 50},
 {'activation': 'relu', 'alpha': 0.05, 'hidden_layer_sizes': 1},
 {'activation': 'relu', 'alpha': 0.05, 'hidden_layer_sizes': 5},
 {'activation': 'relu', 'alpha': 0.05, 'hidden_layer_sizes': 10},
 {'activation': 'relu', 'alpha': 0.05, 'hidden_layer_sizes': 50}]
```

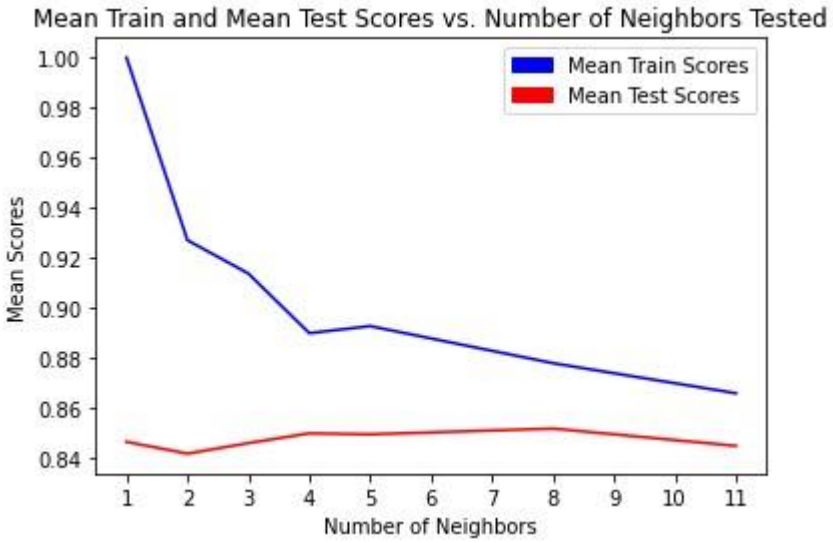
Appendix E: Neural Network Training vs. Test Mean Graphs



Appendix F: Neural Network Mean Test Accuracies



Appendix G: KNN Confusion Matrix



Appendix H: KNN Confusion Matrix