Ilan Barr

December 8, 2023

Data Mining 514

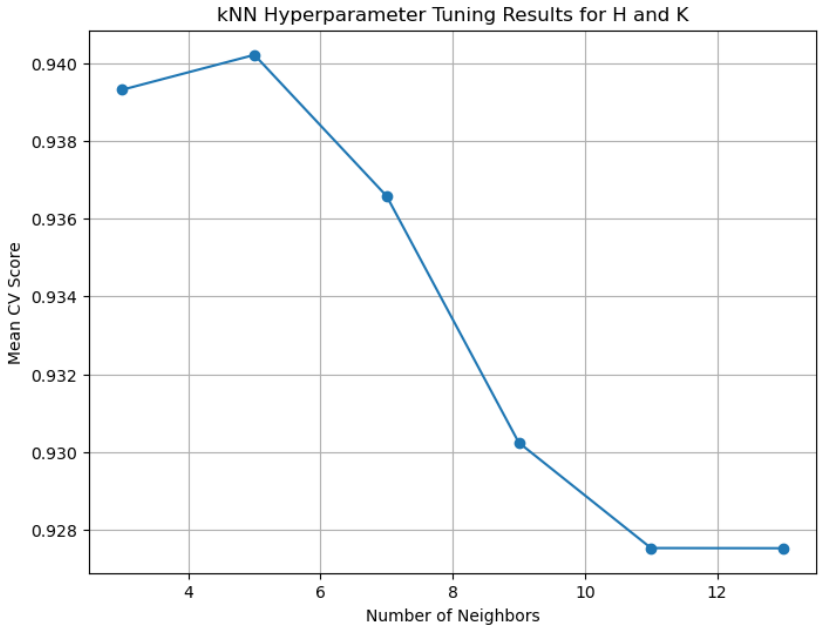
Data Mining on UCI's Letter Recognition Dataset

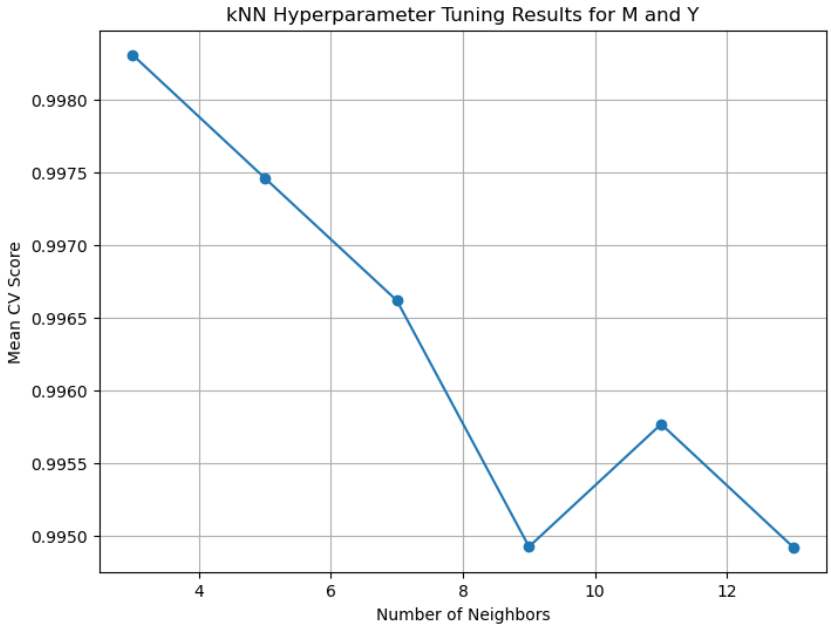
Letter recognition is becoming increasingly more common in the modern world. The use cases are seemingly endless. A few examples that have become prominent in recent years are autonomous cars being able to read signs, artificial intelligence being able to scan in documents and read them, and assistive technology for the visually impaired. The ability for machines to recognize letters, whether they are handwritten or printed, allows the further integration of machines to increase the efficiency of everyday life for humans. Training machines to do this effectively, however, can be a challenging task, however, and if trained incorrectly, could lead to great negative consequences. Many letters can look the same, especially if they are presented in different fonts, or handwriting that may not be the clearest, thus, and easy way to approach this problem is by first training models on binary classifiers, instead of jumping straight to classifying a letter out of 26 options. Training and testing multiple classifiers, instead of just one, can increase the resilience of letter recognition models because where one classification model may fail, another would predict a letter correctly. A rudimentary thought process in examining how well a model performs would be to only consider the accuracy of the model and exclaim that the one with the highest accuracy score is the best model. Failing to factor in other characteristics such as computational complexity would be bad practice. While the accuracy is probably the most important characteristic, if two models have near similar accuracy rates, one

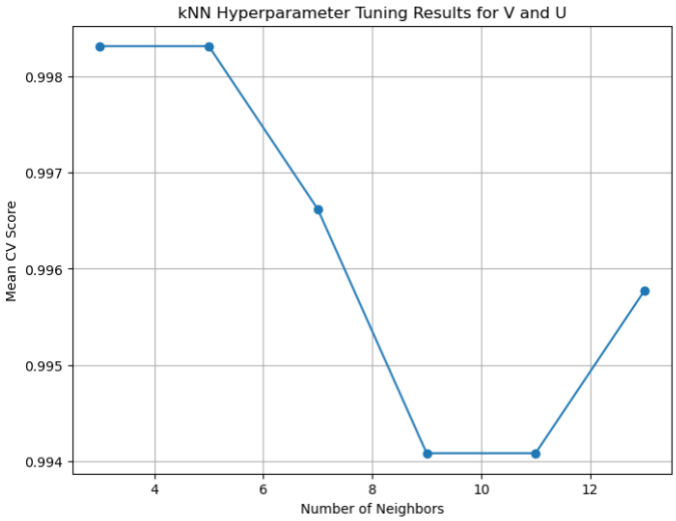
of them being slightly higher, however, was much more computationally complex, leading to longer run times and increased resource consumption, it would not necessarily be correct to choose the model with the higher accuracy score as the “better” model. Decreasing the dimensionality of the models through PCA can be helpful because it can simplify the model, increasing speed and decreasing resource consumption, while, if done correctly, maintain similar levels of accuracy. Ensuring a proper trade-off between the decrease in accuracy and increase in efficiency of the model is important while considering dimension reduction. I chose to do a PCA dimension reduction for the classifications. The reason I chose PCA is because the variables that remain after the reduction of dimensionality are the most significant for the classification problem. For the classification problem where I could choose my own two letters I chose ‘U’ and ‘V’. These letters have a very similar shape, and I have gotten the two mixed up before, specifically, in handwritten text. I would have guessed that the ML models might have trouble in distinguishing between the two letters. I predict the easiest would be H and K because they have vastly different shapes.

The two classifiers I chose to use for the binary classification problems were K-Nearest Neighbors and Artificial Neural Networks. KNN is a relatively simple classifier to understand. It looks at the data set, and after using some distance metric, it makes its prediction based on the data points that are most like the one in question. There is no training phase in a KNN classifier, which might lead to faster computations when dealing with real time data. Additionally, it is very adaptable to feature changes. However, some disadvantages that come with KNN are it is sensitive to noisy data, or irrelevant features; and it requires significant memory storage because it needs to access the entire data set during the testing phase.

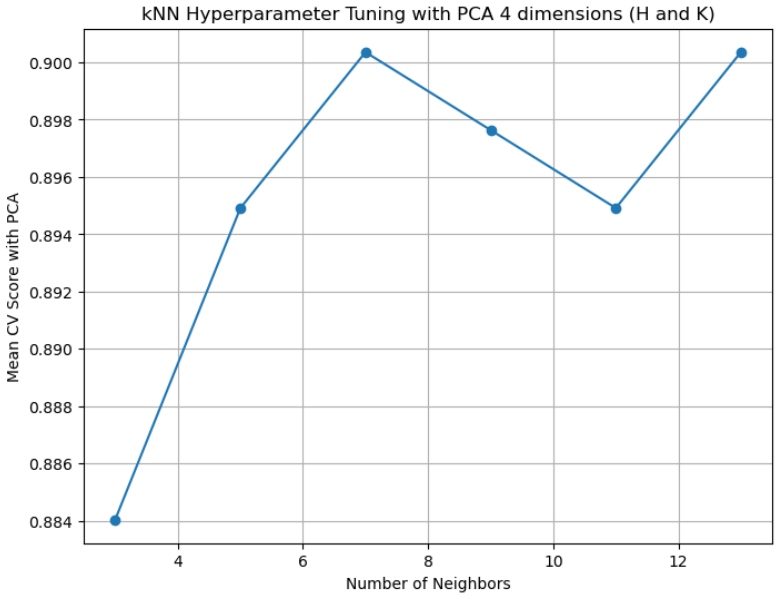
The following graphs show the hyperparameter tuning of KNN for each classifier, H and K, M and Y, and, U and V respectively. I chose the number of neighbors as my hyperparameter for KNN:

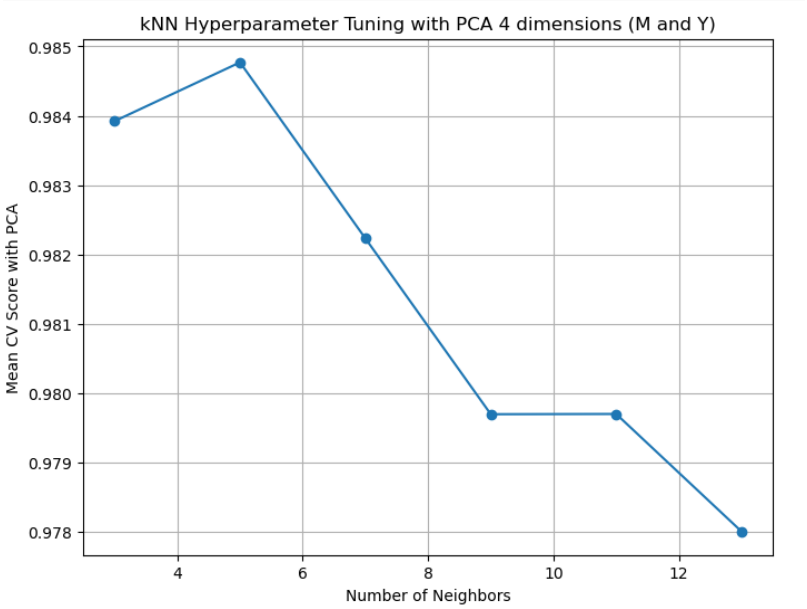


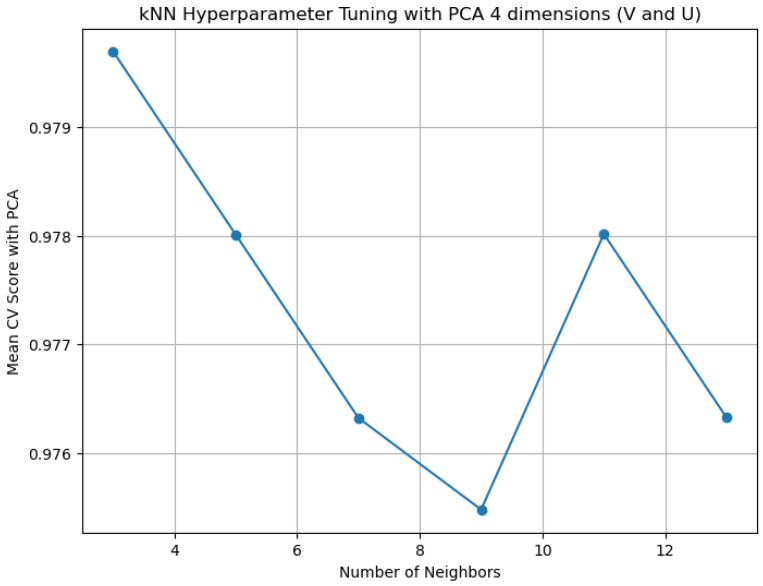




Here is the results from tuning the hyperparameter after the dimension reduction:

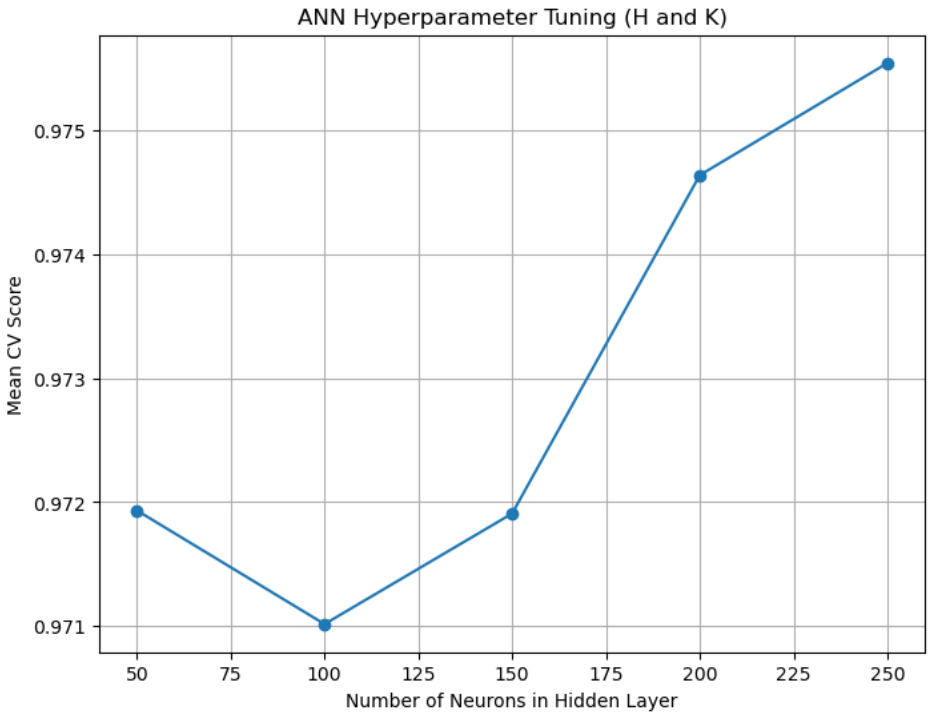


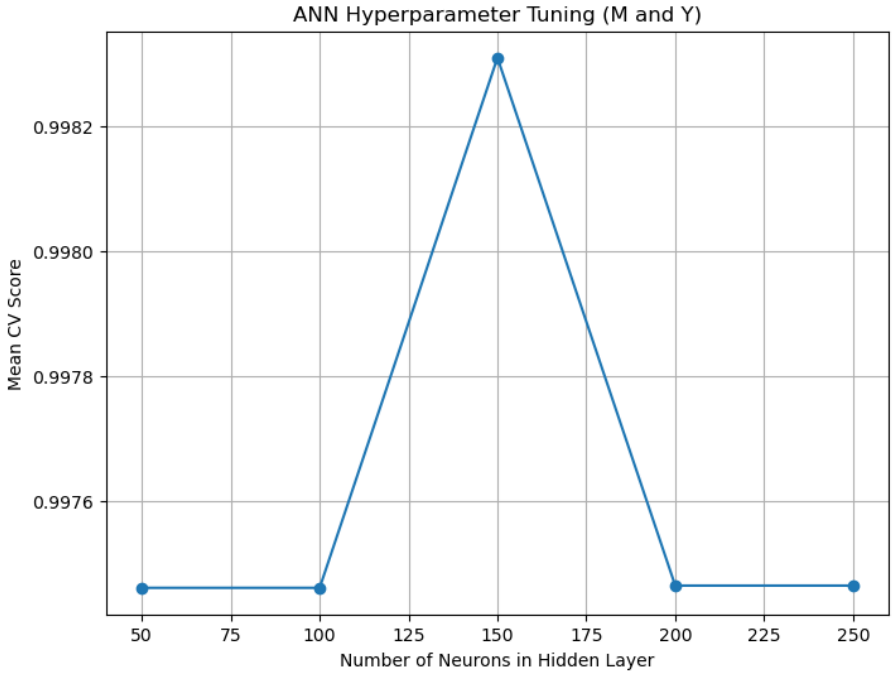


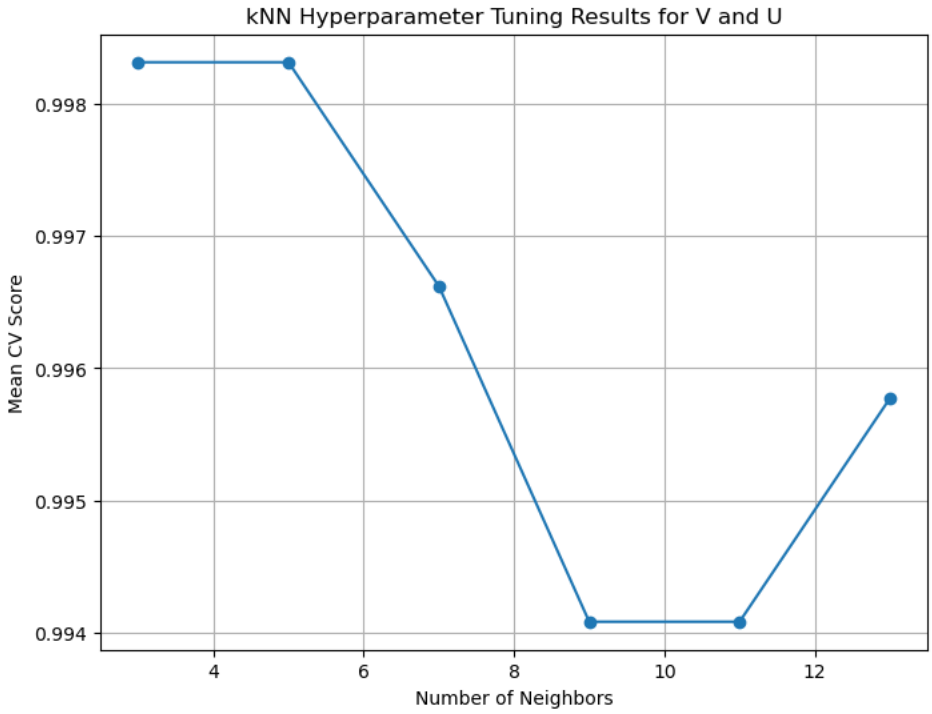


Some advantages that come with ANN is that ANN is well equipped at handling non-linear data. ANNs are also great at flexibility. They can adjust to input data by adjusting weights, which makes it versatile for many different data types and structures. Some disadvantages of ANNs are since they are very complex, they have increased risk of overfitting on the data. Additionally, training ANNs can be very computationally expensive and time consuming. The

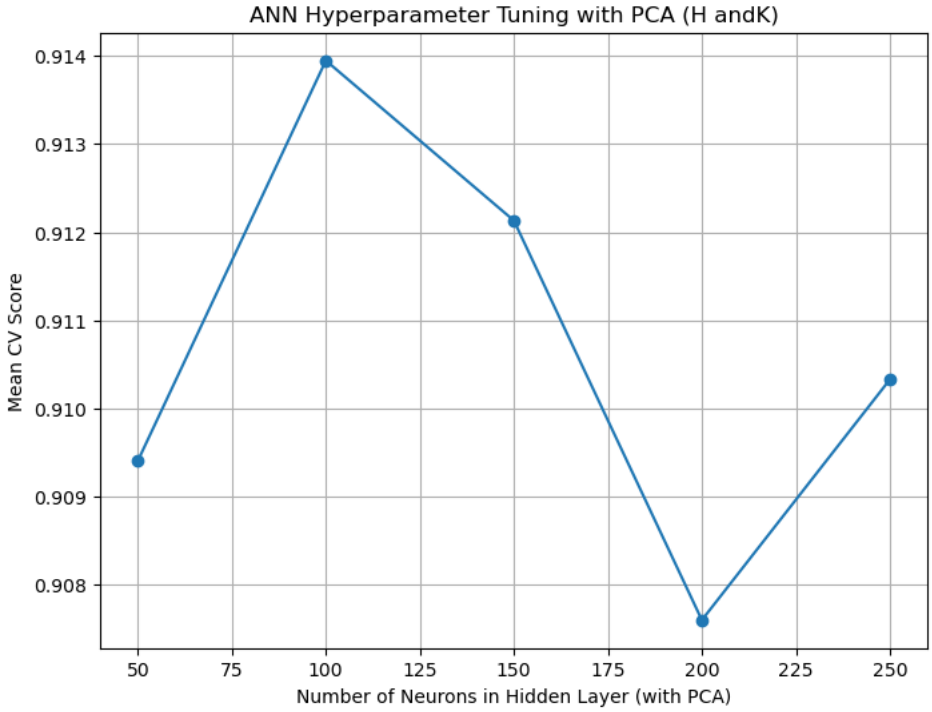
following graphs show the hyperparameter tuning of ANN for each classifier, H and K, M and Y, and U and V respectively. I chose the number of neurons in hidden layer as my hyperparameter for ANN:

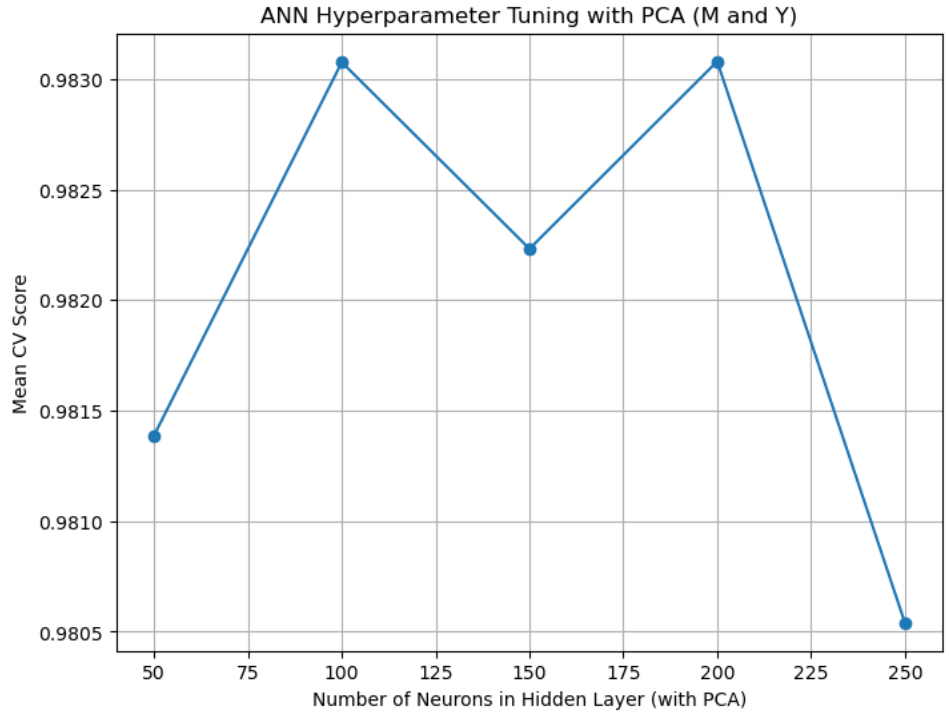


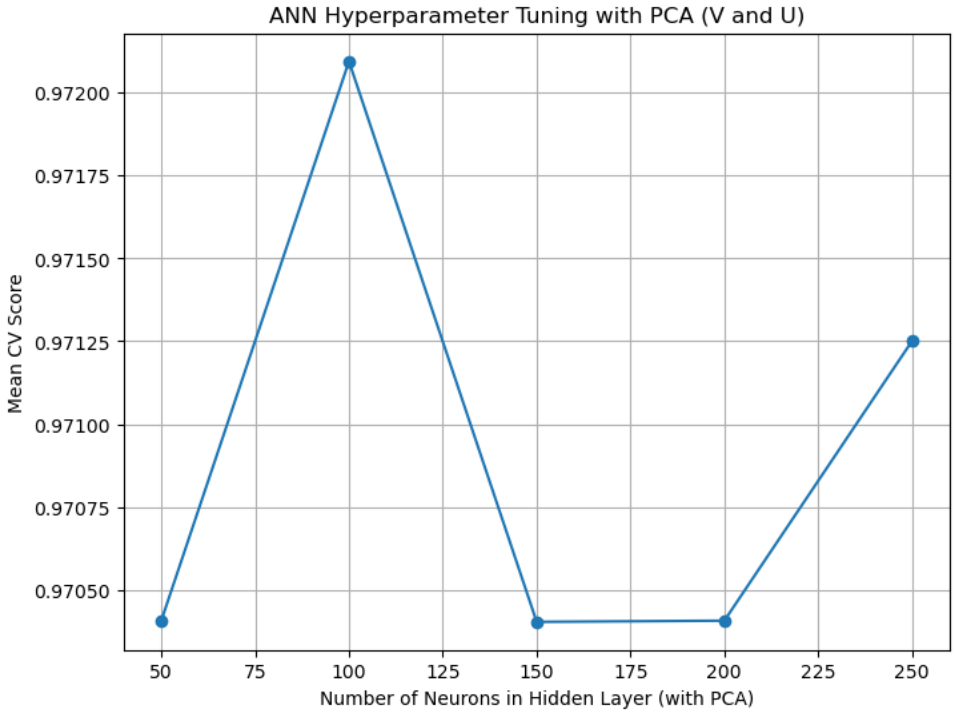




Here is the results from tuning the hyperparameter after the dimension reduction:




ANN Hyperparameter Tuning with PCA (M and Y)
Mean CV Score
0.9830
0.9825
0.9820
0.9815
0.9810
0.9805
50
75
100
125
150
175
200
225
250
Number of Neurons in Hidden Layer (with PCA)


ANN Hyperparameter Tuning with PCA (V and U)
Mean CV Score
0.97200
0.97175
0.97150
0.97125
0.97100
0.97075
0.97050
50
75
100
125
150
175
200
225
250
Number of Neurons in Hidden Layer (with PCA)

The performance and runtime table of the three classification problems is:

Classifier	Accuracy (H,K)	Accuracy (M,Y)	Accuracy (V,U)	Run Time (H,K) (seconds)	Run Time (M,Y) (seconds)	Run Time (V,U) (seconds)
kNN	0.938776	1	1	0.011287	0.017131	0.00684
kNN wit PCA	0.877551	0.981013	0.993671	0.009814	0.010659	0.004492
ANN	0.986395	1	0.993671	0.9965	0.674735	0.828071
ANN with PC	0.897959	0.981013	0.987342	1.486929	0.760097	1.183329

The model I would choose for this problem would be probably be the kNN without reduction of dimensionality. It performed 2nd best behind ANN with an average accuracy rating of 98% and average run time of .011 seconds which was also the 2nd best. The ANN model returned a 99.3% accuracy, however, the time it took was roughly 7000% longer than the Knn, at 0.83 seconds. The tradeoff between accuracy and run time does not seem worth it for only a 1% increase in accuracy. The dimension reduction decreased accuracy but also decreased the run time, however, in this case, the accuracy to run time trade off did not seem to be worth it. For example, for the KNN, average accuracy decreased by roughly 3% while run time only would have improved by 30%. At 0.01 seconds, 30% is not worth losing that 3% accuracy for. If I was given this same task for another dataset, I would likely investigate exploring the other classifiers more deeply. While these classifiers did perform well for this problem, the other one's may have performed even better, and perhaps even more efficiently. As an up and coming data scientist it is best to explore all different types of classifiers. Something I am interested in is seeing how accurate machine learning classifiers might be at predicting a letter out of the standard 26 possibilities. It is easier to make a prediction if there are only two choices, but with 26, which is what it would be in the real world, it might limit how successful the models are.