OMSCS_CS7641 HW1

Clement Li (cli620) Fall 2020

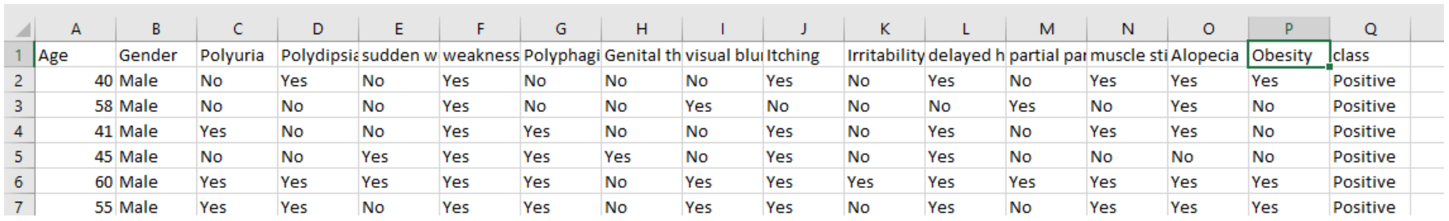
Introduction:

The purpose of this project is to explore some techniques in supervised learning. These techniques include: decision tree with pruning, neural networks, boosting on decision trees, support vector machines, and k-nearest neighbors. The first data set, early diabetes detection, are a set of patients with 16 fields describing the patient and labeled on whether they have diabetes or not. The second data set, coral reef identification, are 128 by 128 by 3 images split up into their folders labeled with inhabitants of the coral reef seafloor.

Dataset 1 – Diabetes:

About/Preprocessing:

This early diabetes detection dataset, pulled down from the UC Irving machine learning repository, has each row represents an individual patient. Each row describes whether they have any of the 16 characteristics of diabetes and a flag of if they were diagnosed with diabetes. Some example characteristics includes age, gender, obese, etc. There are 521 patients in this dataset. The data is mostly binary besides the age field; this would make the dataset not as interesting. To make the problem more interesting, as part of the preprocessing phase, the data is introduced to noise and compare the effects of the learning algorithms. To introduce noise, 25% of the data will be uniformly randomly replaced with 'NA'. This is realistic because symptoms of patients may be unclear (i.e.: irritability, muscle stiffness, delayed healing) or the patient declines to disclose certain information (i.e.: genital thrush, obesity, gender).



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	Age	Gender	Polyuria	Polydipsia	sudden w	weakness	Polyphagi	Genital th	visual blu	Itching	Irritability	delayed h	partial pa	muscle sti	Alopecia	Obesity	class
2	40	Male	No	Yes	No	Yes	No	No	No	Yes	No	Yes	No	Yes	Yes	Yes	Positive
3	58	Male	No	No	No	Yes	No	No	Yes	No	No	No	Yes	No	Yes	No	Positive
4	41	Male	Yes	No	No	Yes	Yes	No	No	Yes	No	Yes	No	Yes	Yes	No	Positive
5	45	Male	No	No	Yes	Yes	Yes	Yes	No	Yes	No	Yes	No	No	No	No	Positive
6	60	Male	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Positive
7	55	Male	Yes	Yes	No	Yes	Yes	No	Yes	Yes	No	Yes	No	Yes	Yes	Yes	Positive

Decision Trees:

Figures:

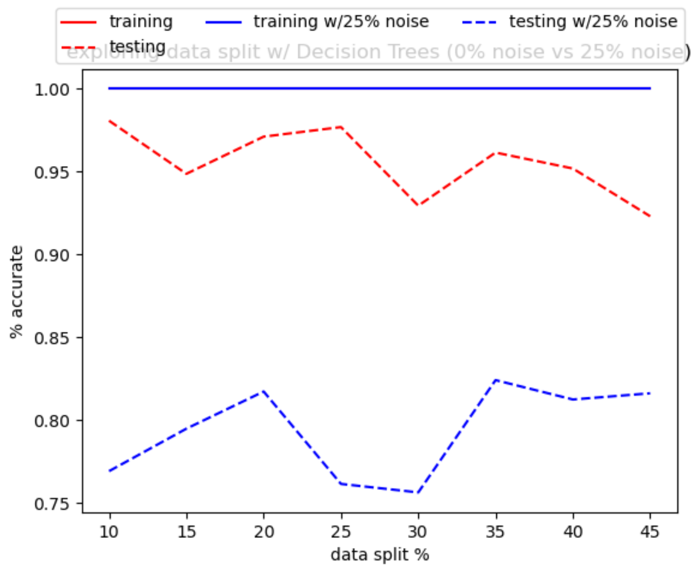


Figure 1: effects of test data size on training/testing accuracy with/without 25% noise

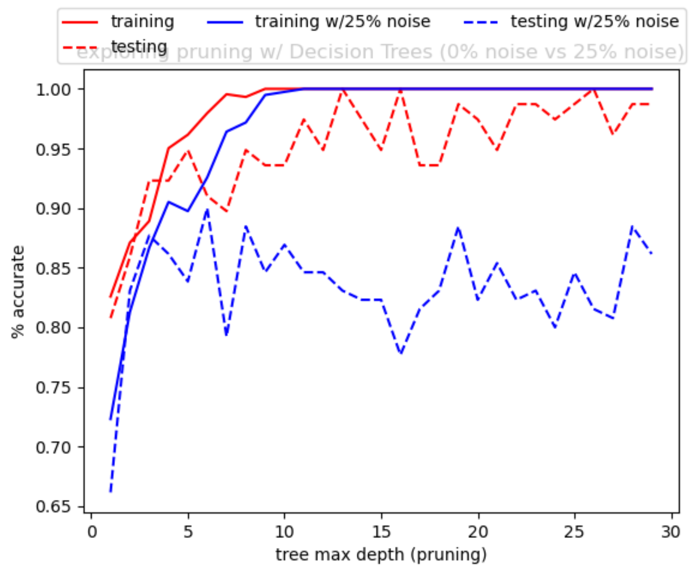


Figure 2: effects of pruning max tree depth on training/testing accuracy with/without 25% noise

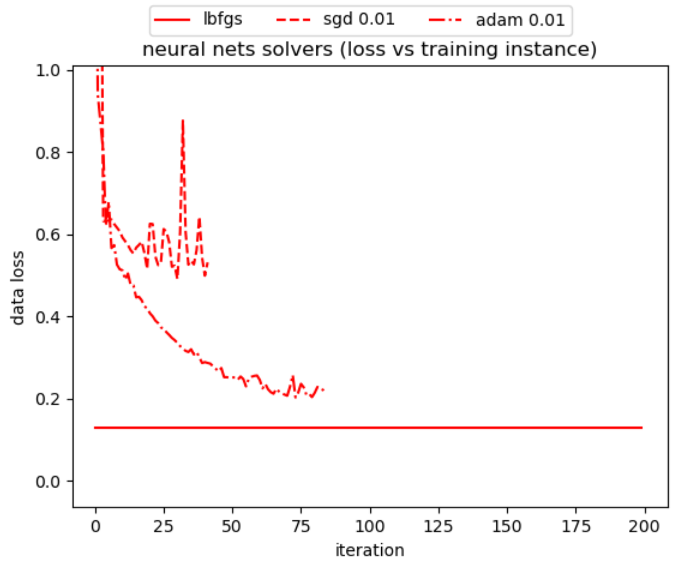
Analysis:

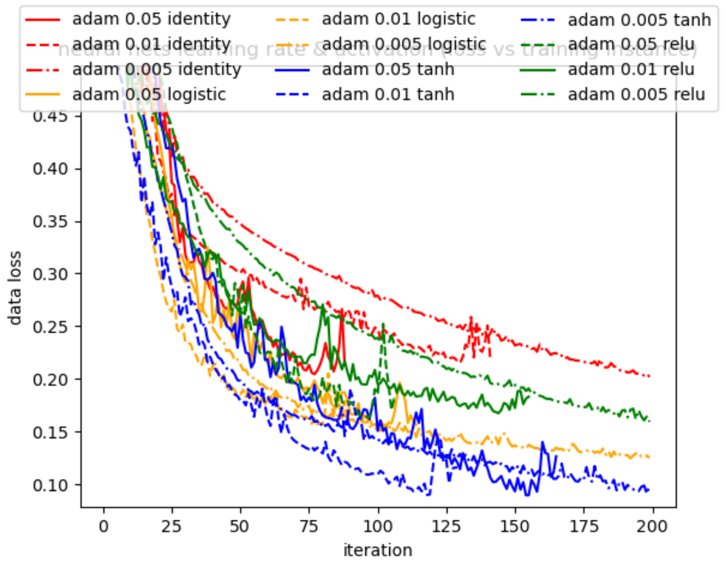
The first figure is trying to find empirically which data split/ testing size performs the best. The figure shows that training data for both with and without noise capped at 100% accuracy for all data split percentages. The testing data without noise is mostly the same with all data splits with a slight negative slope over increasing data split. When it is introduced to noise, there is a peak at 35% but overall, the accuracy spread is mostly the same as without noise. I chose 35% because the 35% has a higher accuracy when considering of noise and without noise

The second figure is trying to find which max tree depth to use in attempt to prune the decision tree. An observation is that the training and testing performance converges after a certain tree depth. Therefore, for efficiency purposes that will be the max depth to prune to. Without noise, the accuracy converges at around 7 nodes and with noise the accuracy converged at around 10 nodes. The convergence is due to the characteristics of the data, this dataset can be fully characterized with 7 nodes. In 7 nodes, without noise, the test dataset accuracy begins to converge at around 10 nodes at around 95% accuracy. With noise, the testing dataset start converging too early at 3 nodes at 85%. This clearly shows the effects of noise on the dataset and the limit of only using max depth to improve the accuracy. By only increasing max depth the training data still reached 100% accuracy, but the testing data capped out at 95% and 85% respective to with and without noise. This shows that something else needs to be introduced for the testing accuracy to increase even higher. For example, another pruning parameter such as min sample leaf or boosting.

Neural Networks:

Figures:





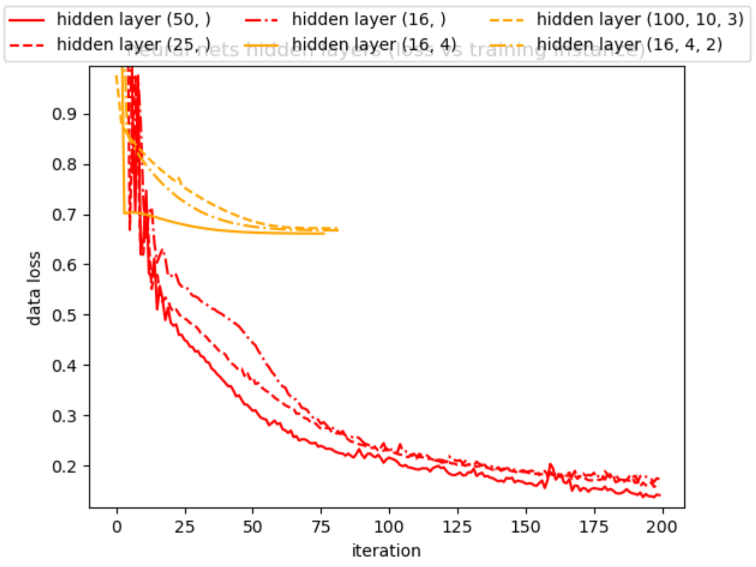


Figure 3-5: Figure 3 (top left) explores different neural net solvers to train with notably lbfgs, sgd with 0.01 learning rate and adam with 0.01 learning rate. Figure 4 (top right) uses adam as the solver to explore both learning rates (0.05, 0.01 and 0.005) and activation methods (identity, logistic, tanh and relu). Figure 5 (bottom left) attempts to span a range of hidden layers. The default layer is (100,) and from there we decreased the one-layer size from 100 to 50 to 25 to 16. Then tried a couple multi-layers using the 16 nodes as the top layer to (16,4) and (16,4,2) along with (100,10,3) to get a larger multi-layer. These plots show the data loss based on training iteration. It uses adam of 0.01 learning rate and relu activation.

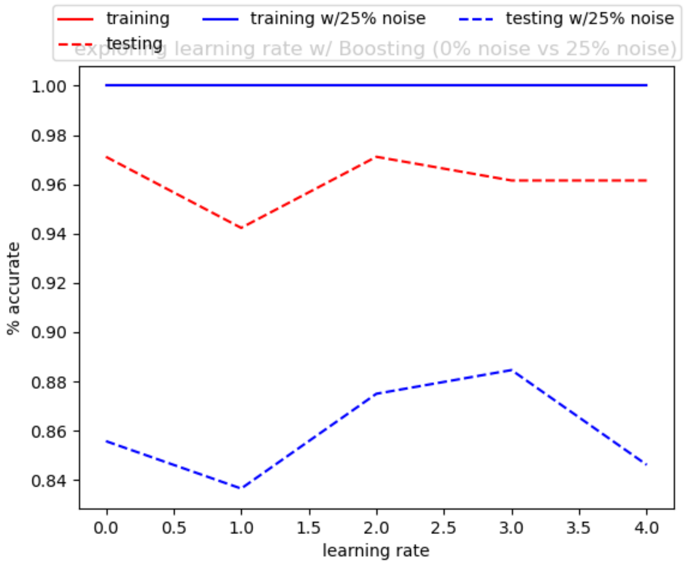
Accuracy Train/test	*Identity*	*Logistics*	*Tanh*	*Relu*
0.05	*0.943/0.938*	*0.962/0.923*	*0.944/0.923*	*0.933/0.931*
w/noise	*0.762/0.785*	*0.903/0.769*	*0.621/0.6*	*0.8/0.708*
0.01	*0.949/0.892*	*0.954/0.915*	*0.977/0.946*	*0.879/0.9*
w/noise	*0.859/0.9*	*0.869/0.823*	*0.987/0.769*	*0.928/0.838*
0.005	*0.933/0.931*	*0.946/0.931*	*0.974/0.969*	*0.946/0.938*
w/noise	*0.785/0.792*	*0.823/0.7*	*0.974/0.754*	*0.895/0.815*

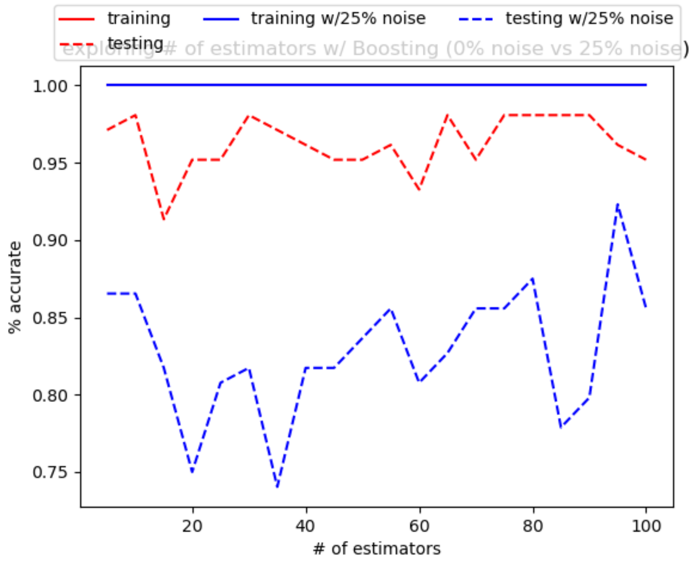
Analysis:

These plots analyses the parameters for the multilayer perceptron (MLP) such as which solver, activation, and hidden layers to use for the diabetes detection data. These plots show data loss over the default 200 iterations. The default 200 iterations are enough to show the performance of the different parameters against each other and do not need to see the final convergence to do a comparison. Figure 3 shows that that lbfgs has the lowest loss at the end of 200 iterations. This is likely because lbfgs algorithms perform best for training smaller data sets (sklearn mlp description page). Adam was used to explore learning rate and hidden layers to see if we can perform better than lbfgs by modifying parameters empirically and the loss for sgd was too big. In Figure 4, we will describe a good training session by the data loss at the end of the training session. The reason why a lower data loss is important at convergence is because it would mean a lower percentage of the weights are being constantly changed. This indicates the uncertainty in the model because the uncertain weighs will be changed. Figure 4 shows that with 200 iterations the learning rate of 0.005 performs the best for all activation modes. Lower learning rates converges the slowest and continues to improve. The activation mode is grouped to identity, relu, logistic and tanh for having the highest to lowest data loss. At the end, an adam solver with 0.001 learning rate and tanh activation mode performed the best of all the ones explores for this data set. This performance, via the accuracy chart, outperformed the accuracy of a simple decision tree algorithm. It also outperformed using an lbfgs MLP solver. Figure 5 shows clear performance between multiple layer models and single models. The single layer models clearly show a healthier data loss. Out of the three, the highest neuron counts performed the best. This is likely due to the simplicity of the dataset; each feature is binary and does not required a complicated MLP.

Boosting:

Figures:





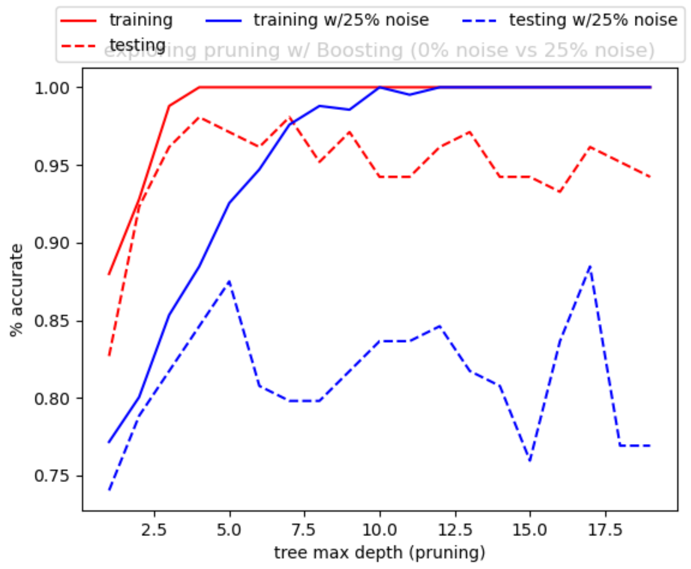


Figure 6-8: Figure 6 (top left) Shows training/testing accuracy with/without noise varying based on learning rate from 10^-1, 10^-2, 10^-3, 10^-4, 10^-5. Figure 7 (top right) Shows training/testing accuracy with/without noise varying based on number of estimators from 0-100 with increments of 5. Figure 8 (bottom left)Shows training/testing accuracy with/without noise varying max tree depth to explore pruning with boosting.

Analysis:

Figure 6 shows that learning rates of 10^-2 gives the highest accuracy without data and 10^-3 gives the highest accuracy with data. This shows that it would take longer for the boosting model to train to reach its optimal performance when introduced to noise.

Figure 7 shows roughly the same performance for all testing data without noise. However, it shows a slightly upward slope for testing results with noise. This indicates that higher number of estimators aids accuracy of the model when introduced to noise. This is likely because noise adds variability and averaging over more estimators would mean averaging over the noise added; average of white noise is 0 over infinite estimators .

Figure 8 shows similar better results compared to using one decision tree. From before, without noise, the accuracy converges at around 7 nodes and with noise the accuracy converged at around 10 nodes. By introducing boosting, the accuracy converges at 3 without noise and 10 with noise. This demonstrates the benefits of using boosting when training models. It prevents the data from overfitting based on one specific set of training data.

Support Vector Machines:

Figures:

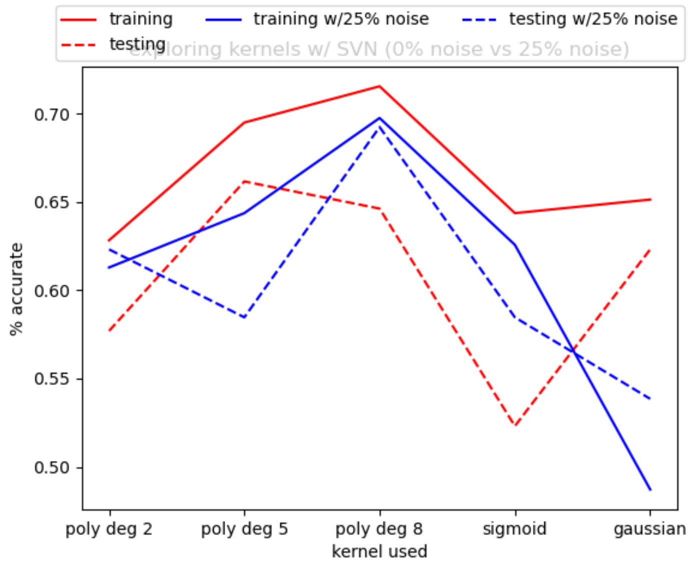


Figure 9: Shows the percent accuracy based on different kernels

Analysis:

This figure explores the three different type of kernel. The kernel used are to try and map the inputs to a higher dimension feature space. Here we use polynomial with degrees of 2, 5 and 8, the sigmoid function and the gaussian function. The best training/testing performance is polynomial function with degree of 8. It also performs better with introduction of noise and being able to predict the testing data better. This is likely because by adding noise (one output of 'NA'), it requires another degree. The sigmoid and gaussian functions performed approximately the same with the data without noise. However, the gaussian kernel's performance dropped significantly with noise. This indicates that the gaussian kernel is more prone to error when noise is introduced. Note that the performance is worst compared to other algorithms.

K-nearest neighbors:

Figures:

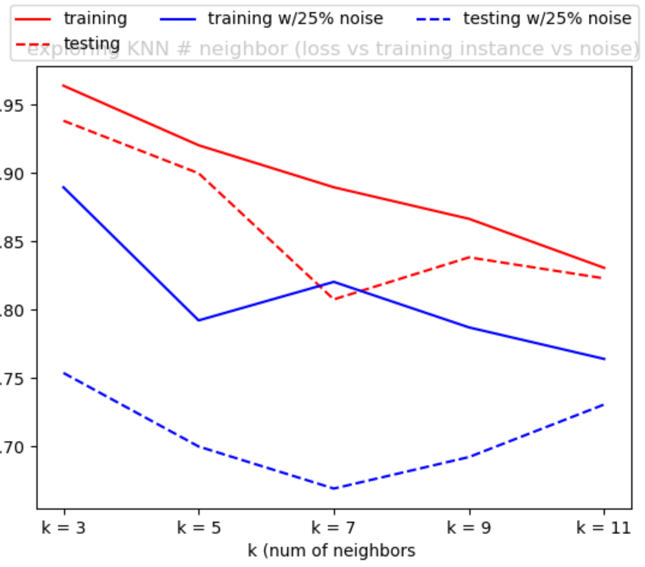


Figure 10: Shows the percent accuracy based on different neighbors for k-nearest neighbors

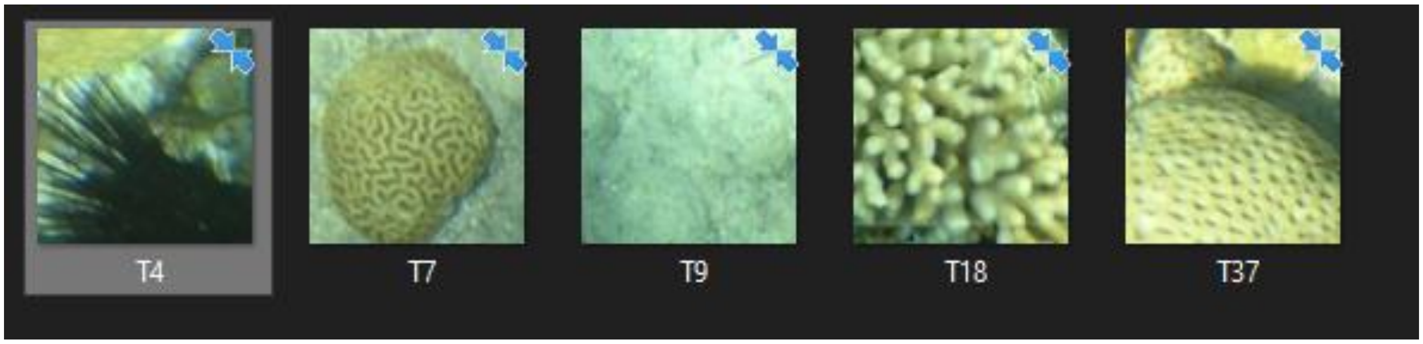
Analysis:

This figure explores the affects k (the number of neighbors to group together) on an KNN algorithm for our diabetes prediction data. The results show an obvious negative slope in performance as we increase the number of neighbors used in the KNN. The number of kernels required is likely less due to the simplicity of the data. Compared to other algorithms, KNN with k=3 has competitive performance without noise. However, it does worse than DT, Boosting and MLP when noise is introduced. The training time is quicker than its counterparts. This indicates that there might be room for improvement decreasing the leaf sizes.

Dataset 2 – Coral Reef:

About/Preprocessing:

This dataset, pulled down from NARCIS data archiving services, is a set of images categorized into five different types of images seen on the coral reef ocean floors: brain corals, branching, favids, sand pavements and urchins. There are 72 brain corals, 49 branchings, 89 favids, 80 sand pavements and 14 urchins. Each 128 by 128 by 3 RGB image is read in, flattened to an 49152 by 1 array, and appended by the label value. This data is interesting because the specimens on the seafloor are normally very tightly clustered and images provided often contains other specimens. The specimens are unique in patterns, but majority of the images are often similar in color and texture. This provides an interesting problem in how each supervised learning algorithm would performs for image recognition of the coral seafloor. Here there will be many different types of pre-processing that happens to try and give the algorithm higher detail features to train on. Some methods include, histogram of oriented gradients, sobel filter for edge detections and morphology dilation to bring out any edges.



Decision Trees:

Figures:

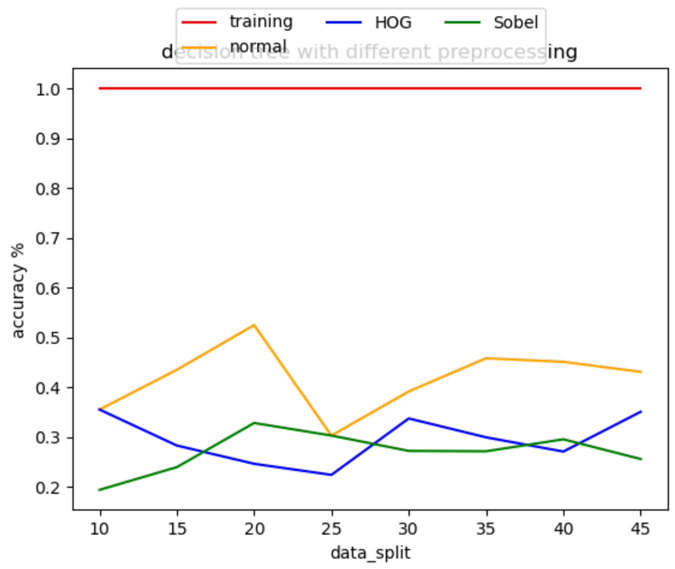


Figure 11: This shows the training and testing accuracy of the data set with different pre-processing methods and varying data splits.

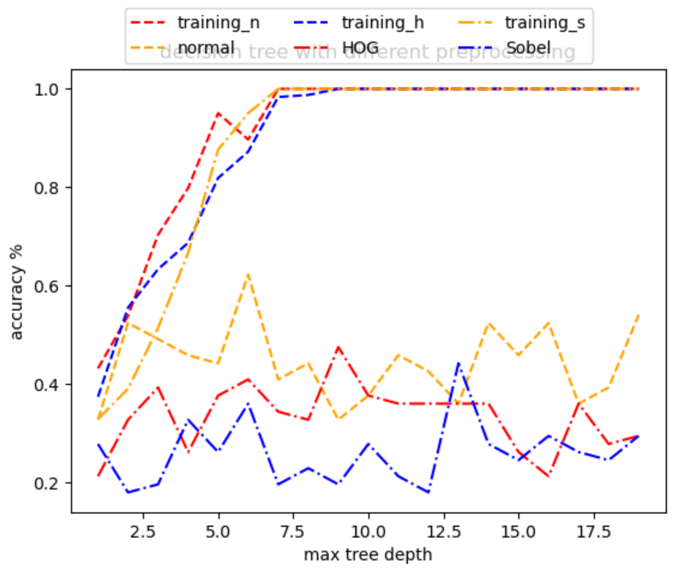
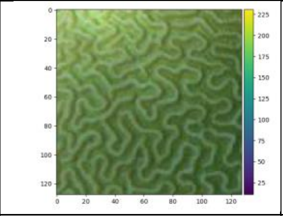
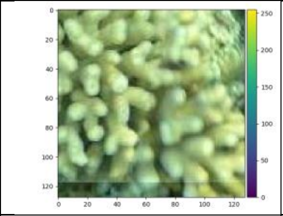
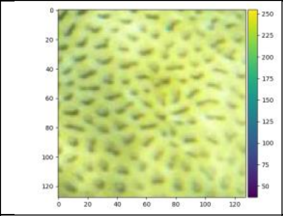
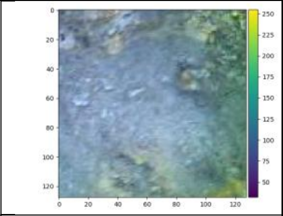
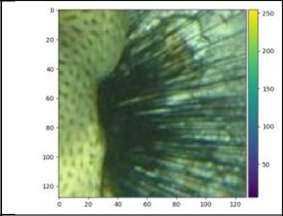
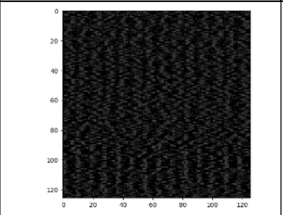
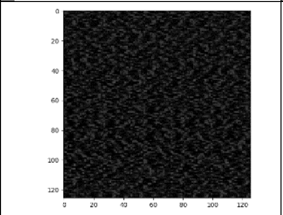
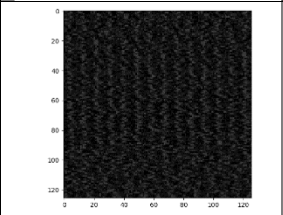
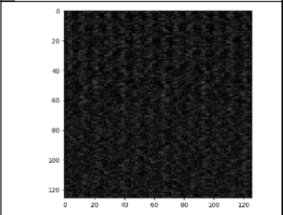
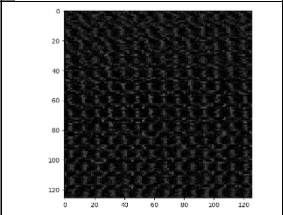
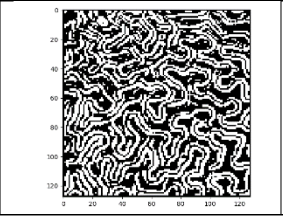
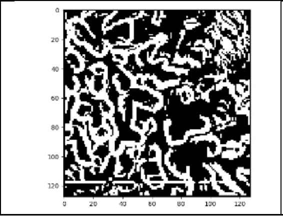
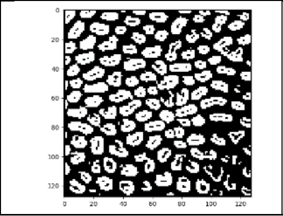
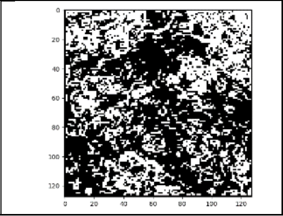
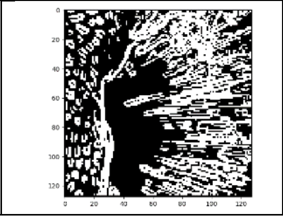


Figure 12: This shows the training and testing accuracy of the data set with different pre-processing methods on varying max tree depth for pruning.

Analysis:

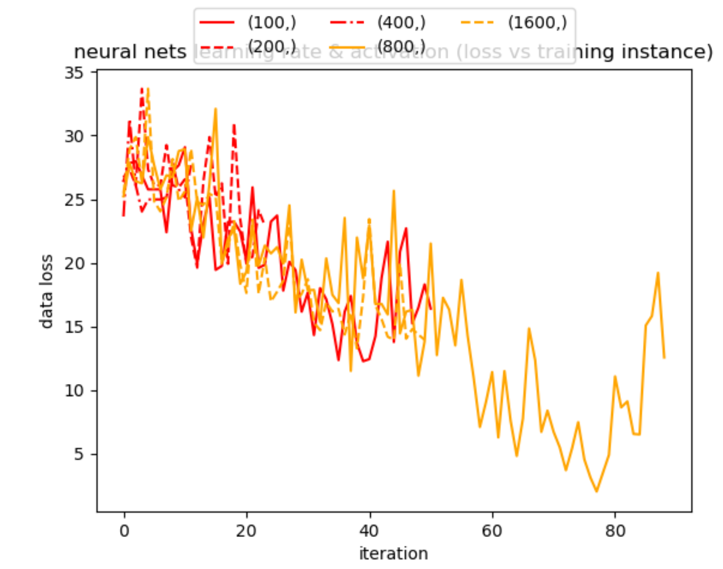
Figure 1 explores the effects of pre-processing in attempt to improve the testing accuracy of the data set. Here it appears that 20% performs the best with this data set. A possible situation can be that this dataset requires more data to train on for higher performance. With the original data, using decision trees, the training data converged but the testing data remained statistically unchanged. This demonstrates how hard of a problem this dataset is. More in-depth preprocessing is needed to raise the testing performance. The preprocessing used are Histogram of oriented gradients (HOG) and Sobel filter. The goal of these techniques is to pull out the texture of the specimens.

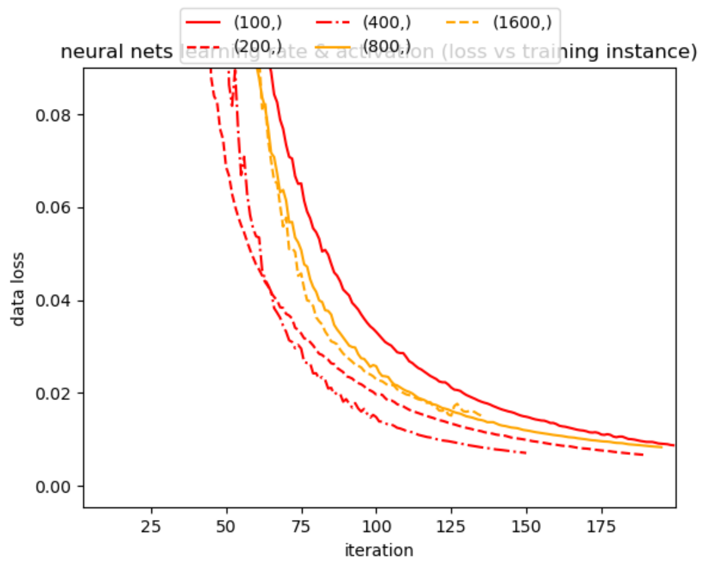
REG					
HOG					
SOBEL					

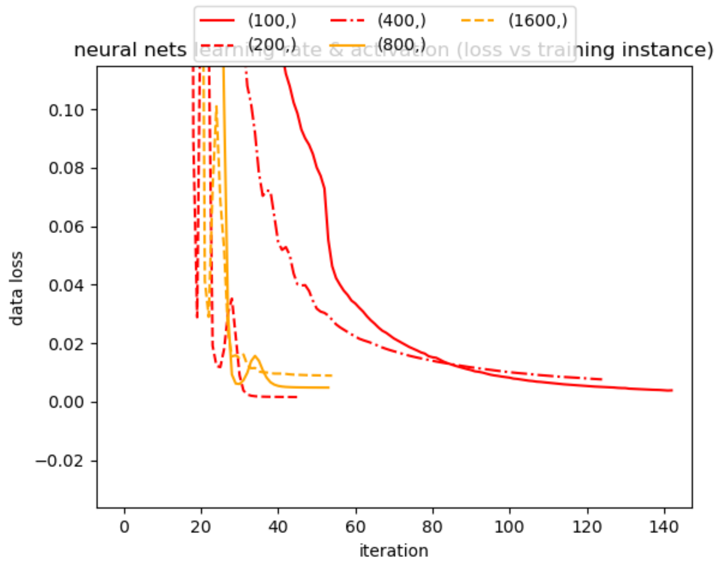
The reason for poor performance for HOG is because we lost a lot of information by transforming the images into a histogram of gradients. The original hope was to be able to capture the textures of each type of specimen better, more tinkering with the parameters of HOG would be needed. As an alternative, a Sobel filter was applied to pull out the patterns. The reason for poor performance of the Sobel filter is because though it did pull out the major details of each specimen, the color details are lost. This can be crucial for example between the last three images.

Neural Networks:

Figures:







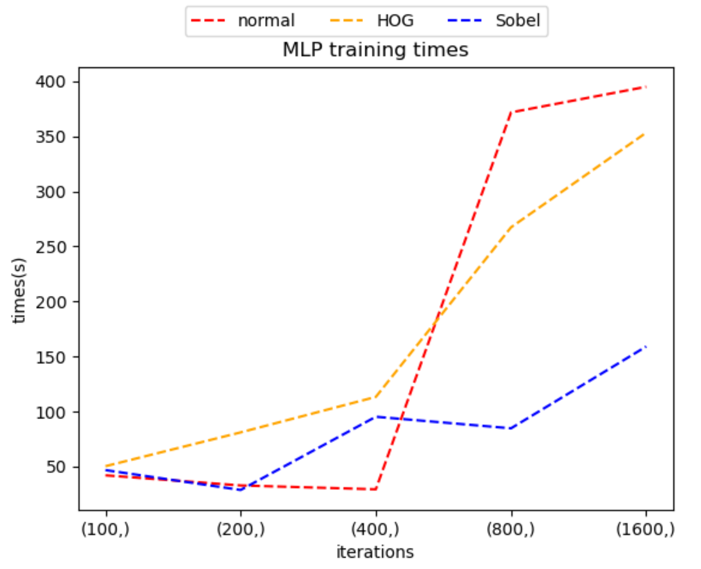


Figure 13-16:
Figure 13 (top left) Shows training/testing accuracy varying based increasing hidden layer size with the original images
Figure 14 (top right) Shows training/testing accuracy varying based increasing hidden layer size with the HOG images
Figure 15 (bottom left) Shows training/testing accuracy varying based increasing hidden layer size with the Sobel filtered images
Figure 16 (bottom right) Shows training time with each hidden layer for each pre-processing technique.

Analysis:

I tried using the same experiment as the previous dataset and came across several problems. The first is that it takes a ridiculous amount of time to train the same number of tests. This is because of the number of features it has; with each additional feature it will exponentially grow in testing time because of the number of neurons needed to be fired and travel through the net. The second is that the accuracy plummeted from the prior experiment; the previous experiment's accuracy varied from 65% to 85% for the training data and 50 to 65% with the testing data. With this data set the training and testing accuracy was between 40% to 60%. This indicates that the model is underfitting, to solve this we need to play with other parameters such as the hidden layers used.

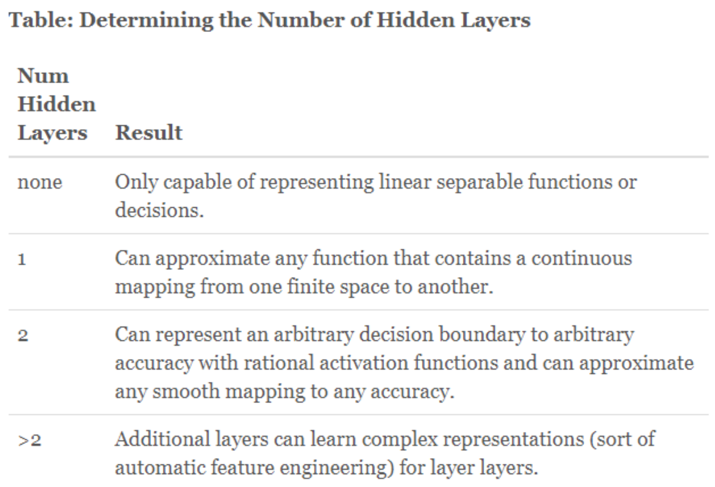


Table: Determining the Number of Hidden Layers

Num Hidden Layers	Result
none	Only capable of representing linear separable functions or decisions.
1	Can approximate any function that contains a continuous mapping from one finite space to another.
2	Can represent an arbitrary decision boundary to arbitrary accuracy with rational activation functions and can approximate any smooth mapping to any accuracy.
>2	Additional layers can learn complex representations (sort of automatic feature engineering) for layer layers.

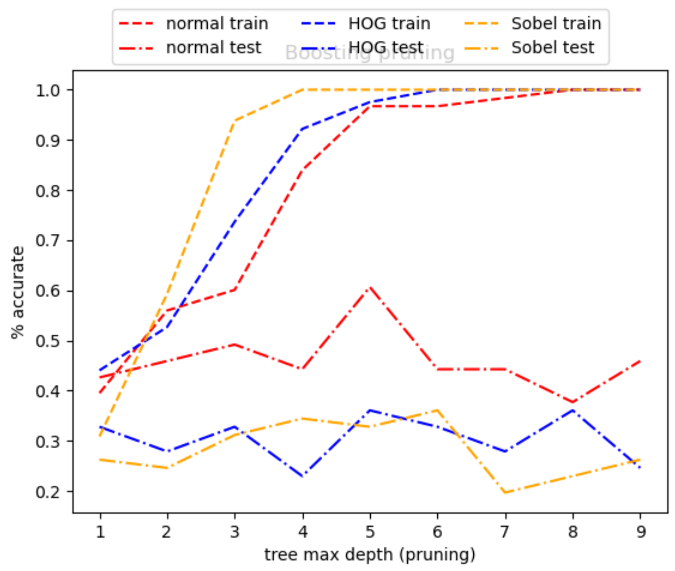
From heaton research on hidden layers , they explains the effects of number of hidden layers and number of neurons in the hidden layers.

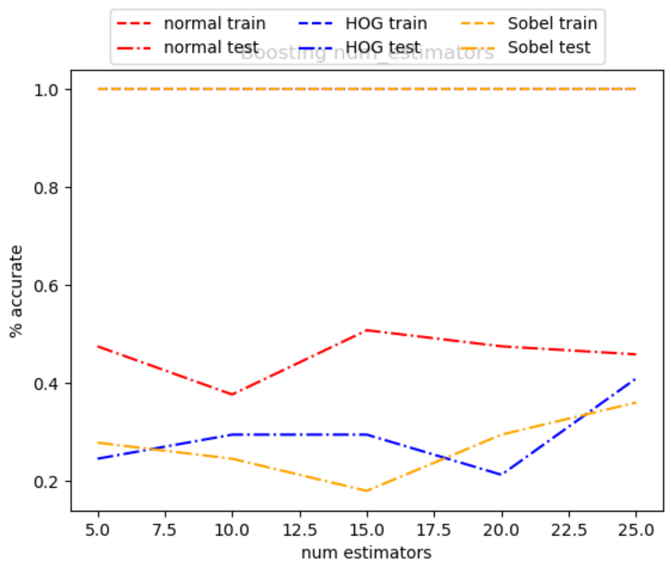
The figure to the right detailed how to determine the number of hidden layers. They explained that too few neurons in the hidden layers results in underfitting and too many neurons result in overfitting and time consumption.

Based on that I decided to test on a single hidden layer with exponentially increasing neurons in the layer. Here I will deem a better training session as the one that converged the smoothest. The training for the original images is significantly worst than the others because there is too much details to learn. The HOG and Sobel images is smaller in size due to it being a gray scale image and can converge faster. The training times reflect the sizes and details of the images. The original images are 128x128x3 images hence taking three times as long as the other images. The HOG images take longer than the Sobel images because of the noisiness of the images. Referring back to the image table in Decision tree analysis, the HOG images has much less defined details than the Sobel images.

Boosting:

Figures:





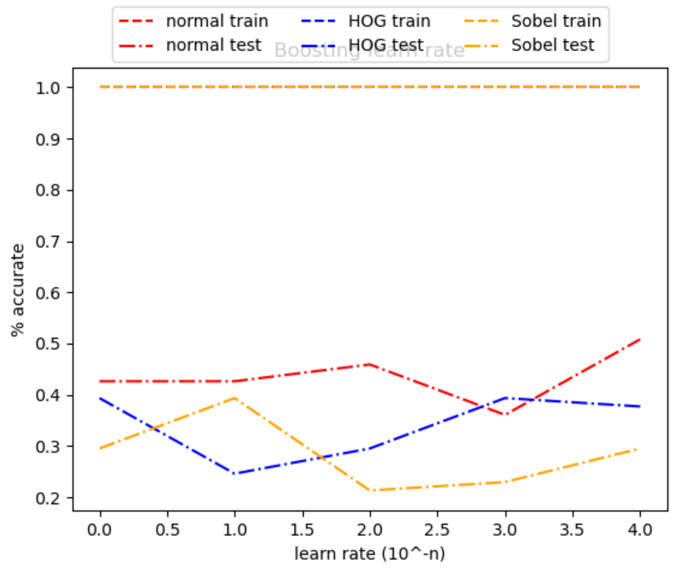


Figure 17-19:
Figure 17 (top left) Shows training/testing accuracy with normal images and the preprocessing images with varying tree max depth (pruning)
Figure 18 (top right) Shows training/testing accuracy with normal images and the preprocessing images with varying number of estimators
Figure 19 (bottom left) Shows training/testing accuracy with normal images and the preprocessing images with varying learning rates (10^-n)

Analysis:

We can compare the accuracy without boosting and conclude that to reach the same accuracy levels, boosting can achieve it with just 4 tree depth compared to 8 tree depth without boosting. This shows that boosting still works on image classification problems. Learning rate seem to have little effects but higher learning rates perform slightly higher. Figure 19 shows that by increasing the number of estimators the performance increases slightly. One idea to make the boosting experiment better maybe to use neural nets as the base model instead of decision trees. Another idea goes back to being able to preprocess better.

Support Vector Machines:

Figures:

Figures:

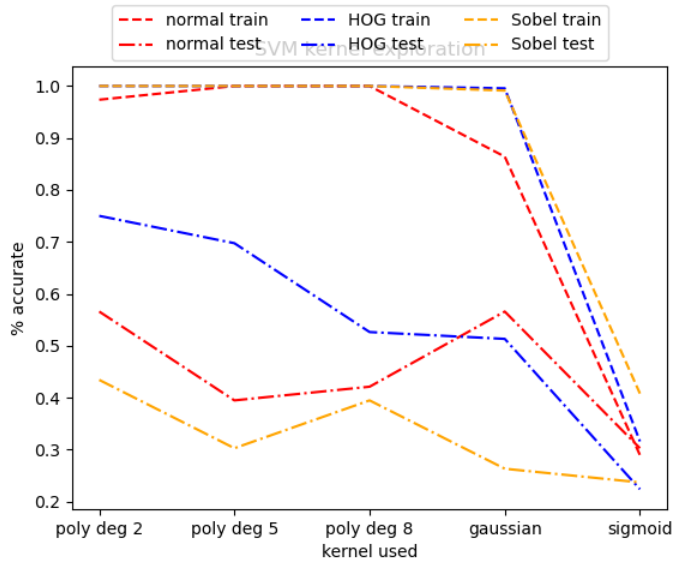


Figure 9: Shows the percent accuracy based on different kernels

Analysis:

This figure explores the three different type of kernel. The kernel used are to try and map the inputs to a higher dimension feature space. Here we use polynomial with degrees of 2, 5 and 8, the sigmoid function and the gaussian function. All three preprocessing methods are included in the plots to compare. The performance decreases with increasing degree used for the polynomial kernels. HOG dataset managed to obtain an 80% accuracy rate with a polynomial kernel with degree of 2; in addition, HOG data set performed the best for all three polynomial kernels. An explanation, based off the image inputs, of this is maybe the kernel matches better with inputs that has less details but still have very defining patterns. The sigmoid kernel's bad performance may be due to it being underfitted. The training accuracy has not even reached 100% yet; a way to fix this can be to decrease the learning rate.

K-nearest neighbors:

Figures:

Figures:

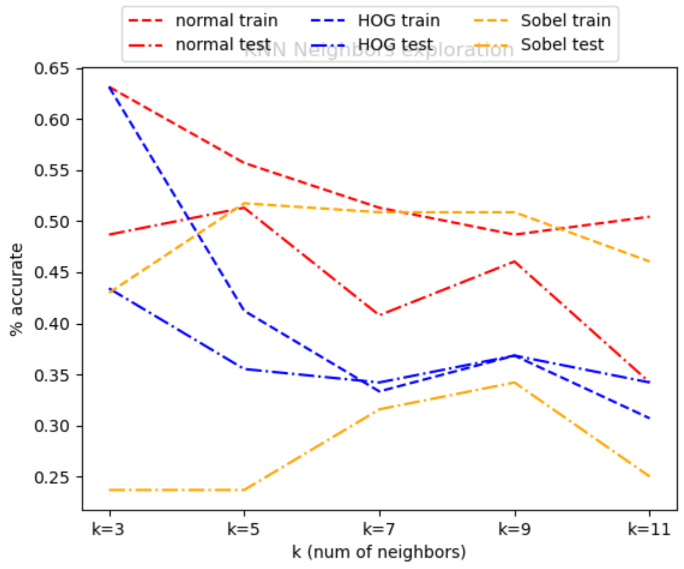


Figure 9: Shows the percent accuracy based on different kernels

Analysis:

This figure explores the 5 different neighbor numbers. The HOG inputs perform poorly with KNNs because of the sparseness of the image. The original images trained well but is underfitted due to the lower accuracy; to attempt to obtain better performance I would test out the weights and algorithm parameters next time. Some other explanations of this graph is that the images are 128x128, a neighbor size of 11 would essentially cover 20% of the image. This would end up accounting for too much of the image.

Overall:

In conclusion, image classification is a much harder problem to solve than classifying labels with 17 features. The higher the amount of features the more data required to properly train the model. I was trying to find a proper method to preprocess the images to pull out better features.

In terms of algorithms, boosting decision trees with pruning did the best with and without noise. It is able to prevent overfitting of the training data and save processing power by being able to use a much more aggressive pruning method than traditional decision trees. For a simple plug in image classifier, SVM with a polynomial kernel of degree 2 with HOG preprocessing performed the best. This is likely due to the simplifying the images by pulling the HOG of the image, something that a SVM with a polynomial kernel of degree 2 is likely best for.

References:

https://archive.ics.uci.edu/ml/datasets/Early+stage+diabetes+risk+prediction+dataset

https://www.narcis.nl/dataset/RecordID/oai%3Aeasy.dans.knaw.nl%3Aeasy-dataset%3A75772

https://towardsdatascience.com/decision-tree-build-prune-and-visualize-it-using-python-12ceee9af752

https://scikit-.org/stable/auto_examples/neural_networks/plot_mlp_training_curves.html?fbclid=IwAR239XIIIqYEVLia--s-p91epOrH3TjrXFnzq7h5z2Uw12T3vNCedzNh_FM

https://towardsdatascience.com/decision-tree-build-prune-and-visualize-it-using-python-12ceee9af752

https://towardsdatascience.com/machine-learning-part-17-boosting-algorithms-adaboost-in-python-d00faac6c464

https://www.heatonresearch.com/2017/06/01/hidden-layers.html

https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html