Journal of Machine Learning Research 4 (2022) 15 Submitted 4/2022; Published

Implementing Linear and Deep Neural Networks for Classification and Regression

William S. Ventura WVENTUR1@JHU.EDU
Department of Computer Science
Johns Hopkins University
Baltimore, VA 21218-2682, USA

Editor: William S. Ventura

Abstract

This paper takes look at the implementations of linear neural networks, deep neural networks, and auto encoders. This paper will implement these networks to handle classification and regression tasks. For the regression tasks, the neural networks will use Mean Square Error as the loss function and for classification tasks, the loss function will be Cross Entropy Loss. These neural networks will also implement the usage of L2 regularization and momentum, as well as Stochastic Gradient Descent to train the networks. These networks will then be tested on six data sets from the UCI Machine Learning Repository.

Keywords: Neural Networks, Perceptrons, Deep Learning, Feed Forward-Back Propagation

1. Introduction

Neural Networks are the core of machine learning. Having been modeled after the biological human neuron, especially around the same time the Hodgkin-Huxley model was proposed for neuronal action potentials. Taking ideas from Hodgkin and Huxley and their work on a squid axon, it gave rise to the idea that these neuron's action potentials can be thought of as an all or nothing scenario. This later influence the machine learning cohort to make an artificial neural network (ANN), based on their model with the outputs being binary, 1 signifying on and 0 signifying off. This initially led to the development of the Linear Perceptron, which is the artificial parallel to one single neuron. While this was good stepping stone, biological neural networks do not operate as singular neurons, instead they are pathways that cascade inhibitory, and excitatory information to the synapses, which either further propagates down the pathway, or the synaptic vesicles release neurotransmitters that back-propagates the network. Having this idea in mind, the next step in ANNs was to model a network where there are multiple layers just like in the biological pathways and that is able to then back-propagate infromation up the network. This led to the development of Deep Neural Networks, with the methodology of feeding forward all the information and based on the output at the last layer, the weights in respect to each layer will be adjusted accordingly through back-propagation.

This paper attempts to implement a Linear Network, a Deep Neural Network, and as well as an Autoencoding neural network(AENN). While the first two types of of networks have a supervised learning method, the Autoencoder in the Autoencoder Neural Network has a

type of supervised learning method. More specifically, the goal of the Autoencoder is to reduced the dimensionality of the input data through data compression. The Autoencoder then seeks to learn an accurate compressed representation of the data, through reducing the data by encoding it and then decoding it to compare the outputs to the initial values, this is referred to as self-supervising. Once the data is able to be compressed it is then attached to another network, which will attempt to predict the output of the data based on the compressed X data.

These three different types of networks will then be tested on data from the UCI Machine Learning Repository. Further more in attempting to minimize Mean Square Error (MSE) and Cross-Entropy Loss (CE) in the regression and classification tasks, it is hypothesizes that the Linear Networks will be able to perform relatively well, although since there are only a linear set of weights it may be prone to overfitting the data. This tends to occur on data that is substantially noisy and those noises in the training data is picked up learned by the model, this in turn makes it difficult for the model to generalize, resulting in the need to use specialized techniques. For the Deep Neural Network, in is also hypothesized that the model well perform well on the training data, but that model could be overfitted due to the noisiness of the training data, and also the complexity of the model, since it is now a neural network with multiple hidden layers instead of just a linear network. This also introduced the possibility that the Deep Neural Network experiences the phenomenon known as the Vanishing Gradient Problem. This is also a similar concern hypothesized for the Autoencoding neural network, since by compressing the data in the Autoencoder, in order to get the exact results as the with the regular uncompressed data, the Autoencoder has to be trained to be 100% accurate. However it's hypothesized that is not the case and there is information loss in compressing the X input data, which in turn will result in lower performance when it is attached to the other network. This will then in turn experience lower performance noise in the compressed X input data, which could overfit the model, and for the same reasons as above, the increase in complexity will then in turn create the possibility for the Vanish Gradient Problem.

2. Methods

2.1 Mean Square Error

For the regression tasks, the goal is the minimize the Mean Square Error, by updating the weights on the three different kind of neural networks. Mean Square Error is defined as such:

$$\mathcal{L}_{MSE} = \frac{1}{N}\sum_{n=1}^{N}(Y - \hat{Y})^2$$

2.2 Cross Entropy Loss

For the classification tasks, $Softmax$ was used as the activation function for the networks, which allows for the interpretation of the outputs as probabilities. $Softmax$ is defined as such:

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

Cross Entropy Loss is then used to measure the error at the $Softmax$ output layer. The goal of the networks for the classification task was to minimize Cross Entropy Loss, L_{CE}. Cross Entropy Loss is defined as such:

$$\mathcal{L}_{CE}(w) = \frac{1}{N}\sum_{n=1}^{N} H(p_n, p_q) = -\frac{1}{N}\sum_{n=1}^{N}[y_n \log \hat{y}_n + (1-y_n)\log(1-\hat{y})]$$

2.3 Forward

In training the networks, first the inputs are passed through the network, activity, activation, and an output are calculated. Those there could be subtle differences depending on the type of network and whether they use a non-linear or linear activation function. In general Feeding-forward the information to the network looks like this, where the weighted z_k^l is the weighted sum of the activations in the previous layer.

$$z_k^l = b_k^l + \sum_j W_{kj}^l a_j^{l-1}$$

a_k^l refers to the neurons activation, where $a_k^l = f(z_k^l)$, where $f()$ is the activation function. Depending on the type of network used, the activation function can vary from being a $Softmax$, $Linear$ or $tanh$ activation function.

2.4 Gradient Descent

The learning rate, ϵ is used to control the increment size. Defined below is the common weight update rule:

$$\Delta W \propto -\frac{\partial \mathcal{L}}{\partial W}$$

$$\Delta W_{kj}^l = -\epsilon \frac{\partial \mathcal{L}}{\partial a_k^l} = -\epsilon \frac{\partial \mathcal{L}}{\partial a_k^l}\frac{\partial a_k^l}{\partial z_k^l}\frac{\partial z_k^l}{\partial W_{kj}^l}$$

2.5 Backpropagation

Through the use of back propagation, it provides a way to calculate $\frac{\partial \mathcal{L}}{\partial a_k^l}$, for each layer recursively in an attempt to find $\frac{\partial \mathcal{L}}{\partial W_k^l}$ and $\frac{\partial \mathcal{L}}{\partial b_k^l}$, for the weight and bias terms. In short it looks something like this for the last layer:

$$\frac{\partial \mathcal{L}}{\partial W_{kj}^L} = \frac{\partial \mathcal{L}}{\partial z_k^L}\frac{\partial z_k^L}{\partial W_{kj}^L} = \delta_k^L a_j^{L-1}$$

This is with respect to weight and with respect to bias is shown below:

$$\frac{\partial \mathcal{L}}{\partial b_k^L} = \frac{\partial \mathcal{L}}{\partial z_k^L}\frac{\partial z_k^L}{\partial b_k^L} = \delta_k^L(1) = \delta_k^L$$

2.6 Additional Methods

In order to tackle the possibility of the model being overfitted and avoiding the possibility the vanishing gradient phenomenon these extra precautions were applied:

2.6.1 He-et-al Initialization:

The weights are initialized taking into consideration the size of the previous layer. This helps attain a global minimum of the loss function fast and more efficiently. The initialized weights are still random but in this type of initialization the range differs depending on the size of the previous layer. This is similar to Xavier initialization, except instead of using a scale factor of 1, He-et-al initialization uses a scaling factor of 2.

2.6.2 L2-Regularization:

In addition to using the He-et-al initialization, L2 Weight Decay Regularization was also implemented as a means to prevent overfitting the neural network. It works by multiplying the weights by a factor less than 1 after each update. This prevents the weights from growing too large, which if not applied could lead to an exploding gradient situation. The L2 weight decay is as followed:

$$\mathcal{L}_2(w) = \mathcal{L}(w) + \frac{\lambda}{2}w^2$$

2.6.3 Momentum:

Momentum was used as a way to speed up the learning and help prevent the neural network from getting stuck at a local minima. Momentum, γ was applied as such during the weight updates:

$$\Delta W_{ij}^l = (\epsilon * \frac{\partial \mathcal{L}}{\partial W_{ij}^l}) + (\gamma * \Delta W_{ij}^{t-1})$$

2.7 Cross Validation and Hyper-Parameter Tuning

Some assumptions made beforehand about the data was that all the instances in the data had a random distribution and that these feature variables, x'_1, ... x'_n were independent of each other and identically distributed (IID). With those assumptions in mind, all of the regression data was normalized in the data preprocessing steps, with the output , Y made to be within range -1 and +1. For the classification tasks the output Y, was one-hot encoded. All networks underwent 5 fold cross validation with same values of $regStrength = 0.001$

and *momentum* =0.05. *eta* was the only hyper-parameter that was tuned. For all the classification dataset, *eta* = [0.01, 0.015, 0.02, 0.03, 0.035] were used in finding out which is the best learning rate for the data. The regression tasks however had different *eta* values during tuning. For the abalone data, *eta* = [0.01, 0.015, 0.02, 0.03, 0.035] were used. For the forest fire and computer hardware data, *eta* = [0.01, 0.0125, 0.015, 0.0175, 0.02] were used during tuning.

3. Results

3.1 Car Classification via Linear Network

3.1.1 Tuning Results

Figure 1: Graph: Car Linear Network Tuning Performance for each k-Fold, Eta = 0.01

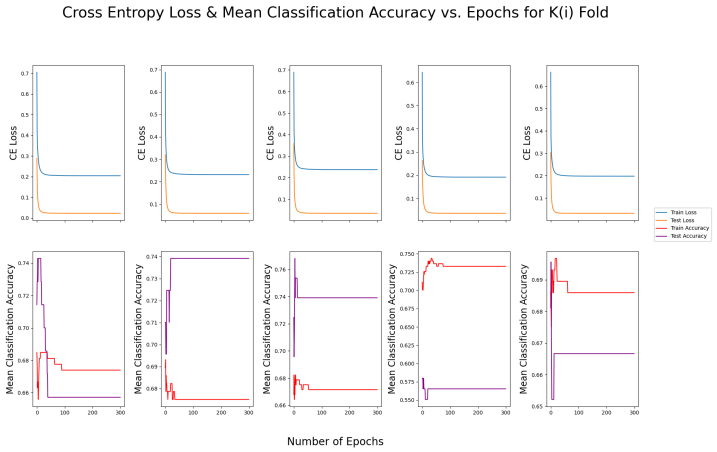


Figure 2: Graph: Car LN Tuning Avg Performance For Cross Validation, Eta = 0.01

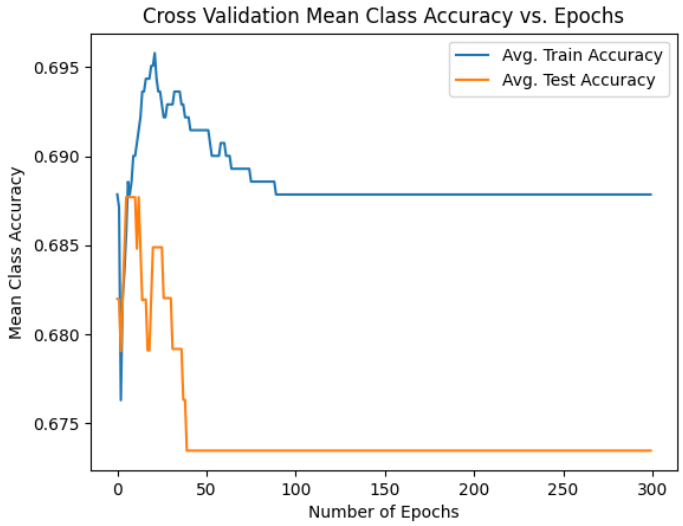


Figure 3: Graph: Car Linear Network Tuning Training Confusion Matrix , Eta = 0.01

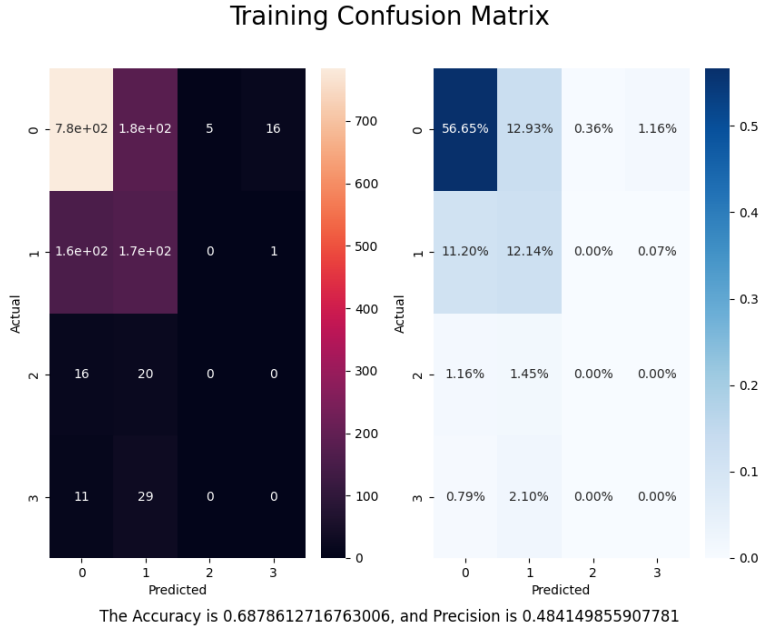


Figure 4: Graph: Car Linear Network Tuning Testing Confusion Matrix , Eta = 0.01

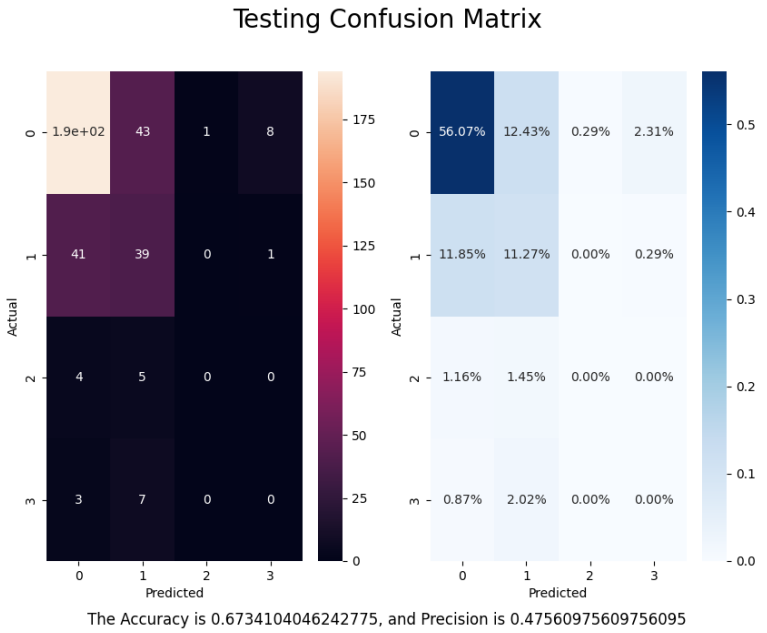


Figure 5: Graph: Car Linear Network Tuning Performance for each k-Fold, Eta = 0.015

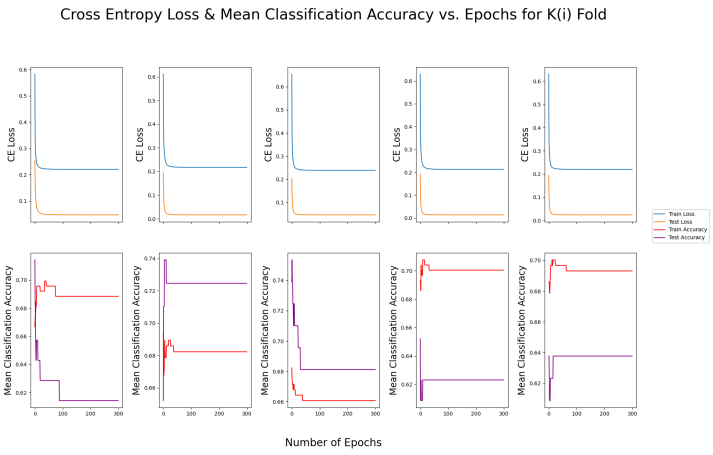


Figure 6: Graph: Car LN Tuning Avg Performance For Cross Validation, Eta = 0.015

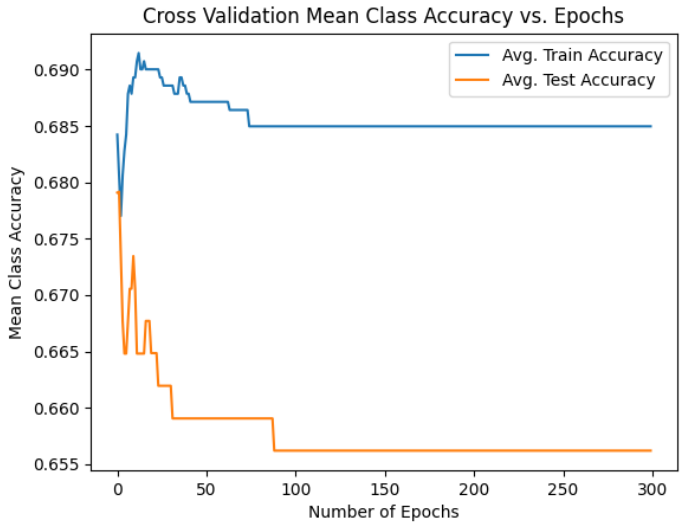


Figure 7: Graph: Car Linear Network Tuning Training Confusion Matrix , Eta = 0.015

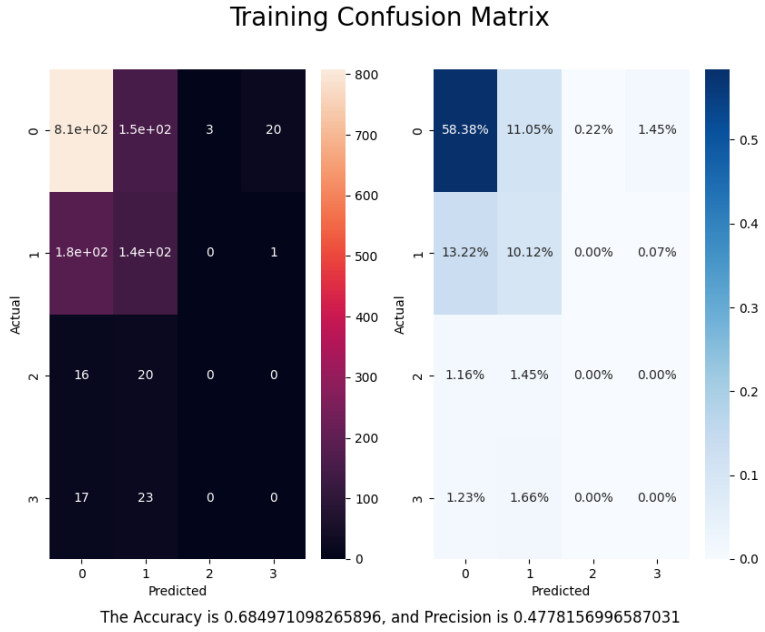


Figure 8: Graph: Car Linear Network Tuning Testing Confusion Matrix , Eta = 0.015

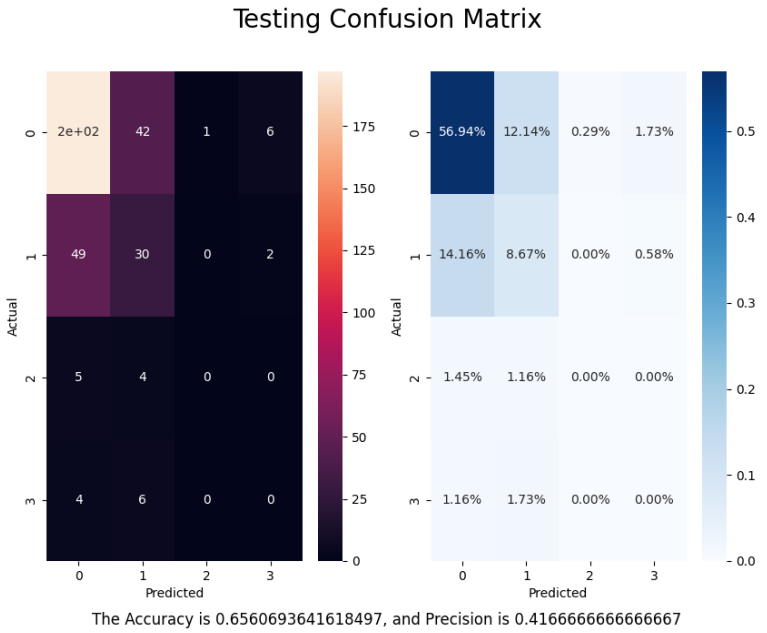


Figure 9: Graph: Car Linear Network Tuning Performance for each k-Fold, Eta = 0.02

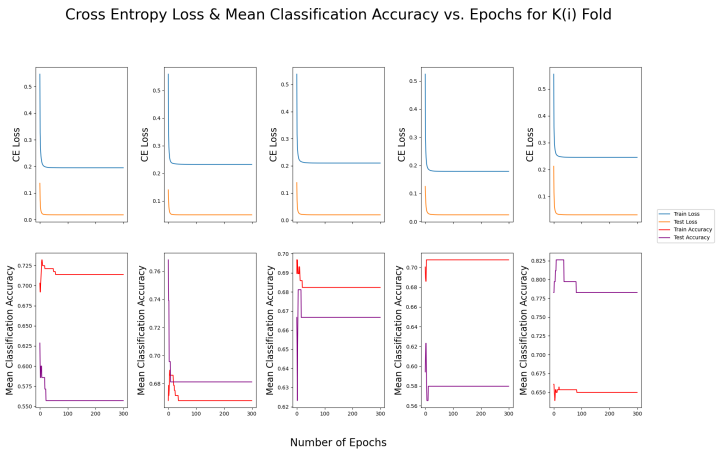


Figure 10: Graph: Car LN Tuning Avg Performance For Cross Validation, Eta = 0.02

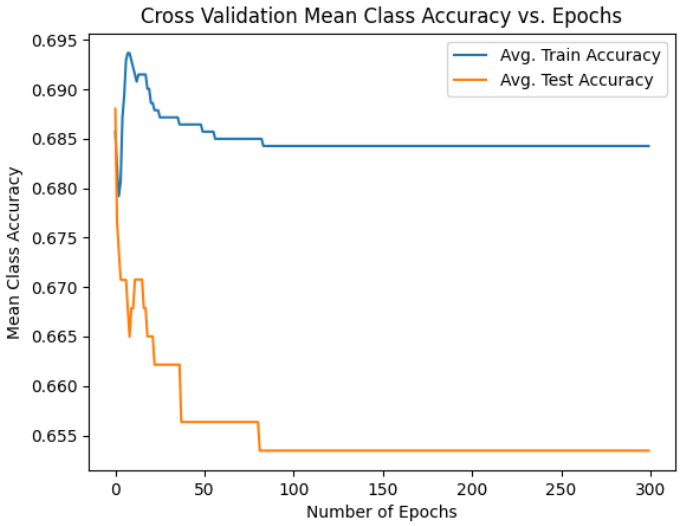


Figure 11: Graph: Car Linear Network Tuning Training Confusion Matrix , Eta = 0.02

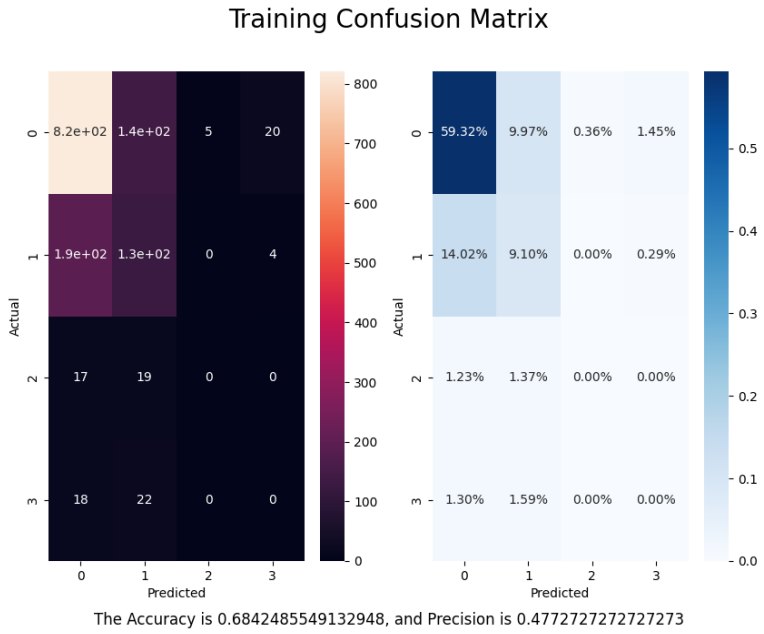


Figure 12: Graph: Car Linear Network Tuning Testing Confusion Matrix , Eta = 0.02

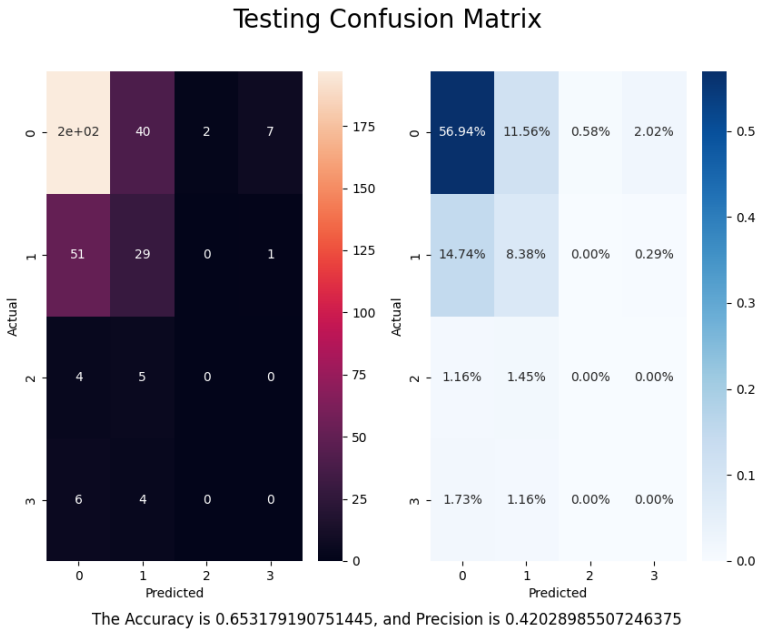


Figure 13: Graph: Car Linear Network Tuning Performance for each k-Fold, Eta = 0.03

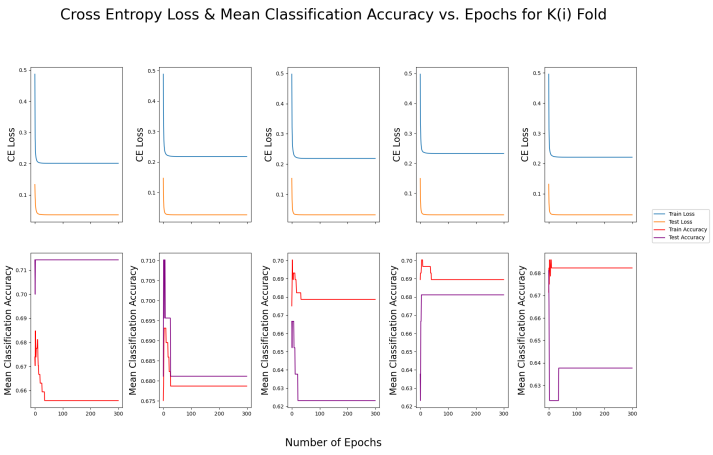


Figure 14: Graph: Car LN Tuning Avg Performance For Cross Validation, Eta = 0.03

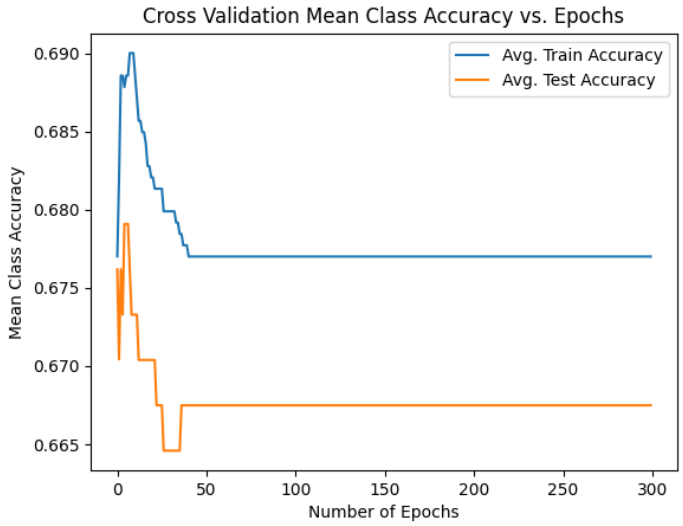


Figure 15: Graph: Car Linear Network Tuning Training Confusion Matrix , Eta = 0.03

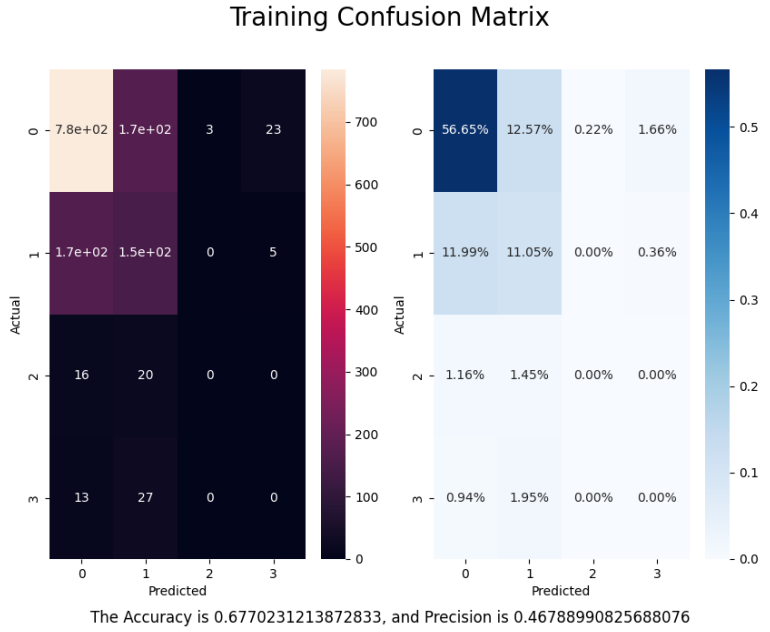


Figure 16: Graph: Car Linear Network Tuning Testing Confusion Matrix , Eta = 0.03

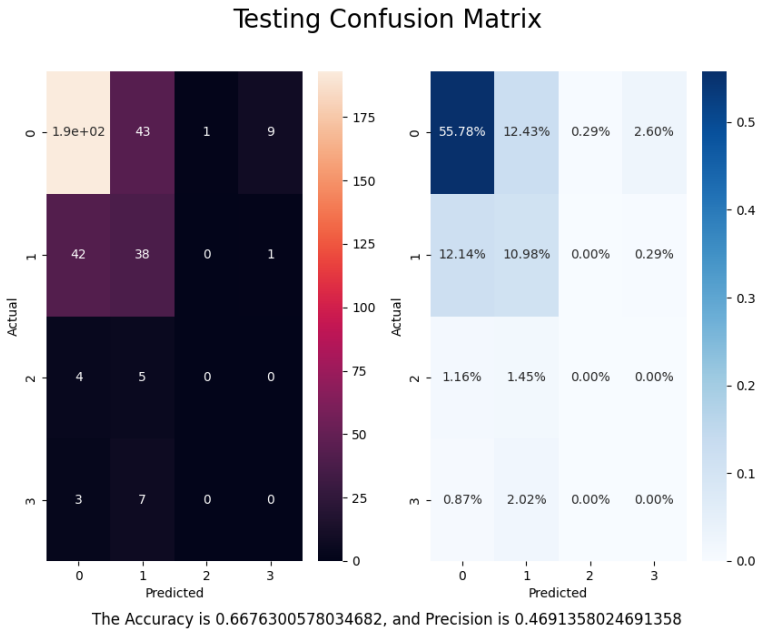


Figure 17: Graph: Car Linear Network Tuning Performance for each k-Fold, Eta = 0.035

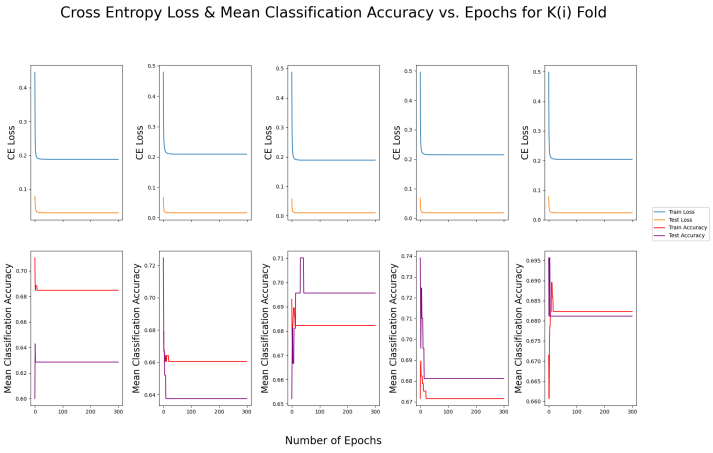


Figure 18: Graph: Car LN Tuning Avg Performance For Cross Validation, Eta = 0.035

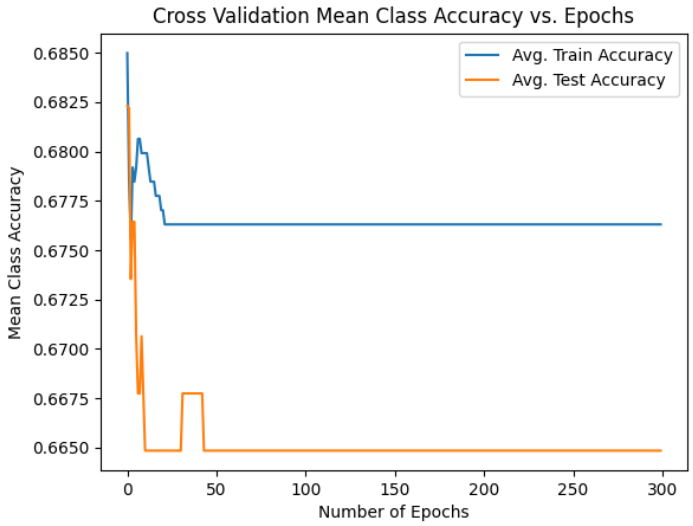


Figure 19: Graph: Car Linear Network Tuning Training Confusion Matrix , Eta = 0.035

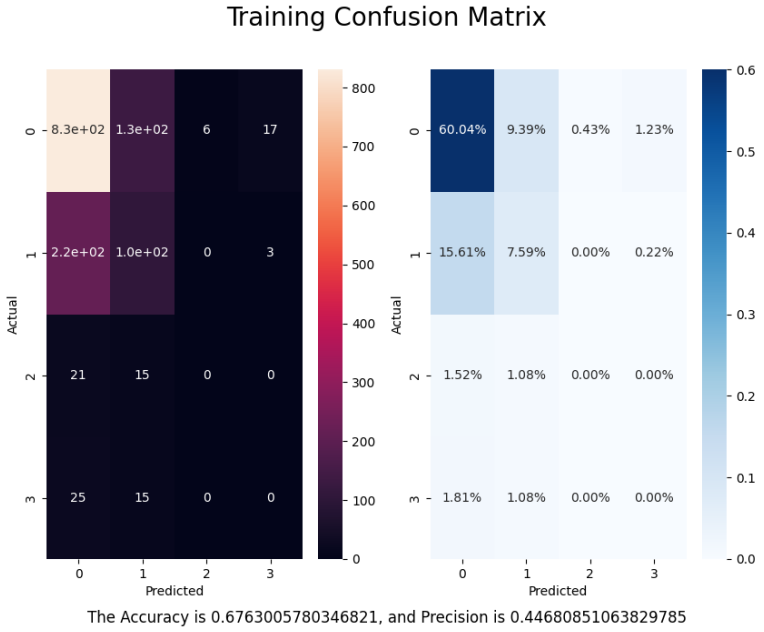
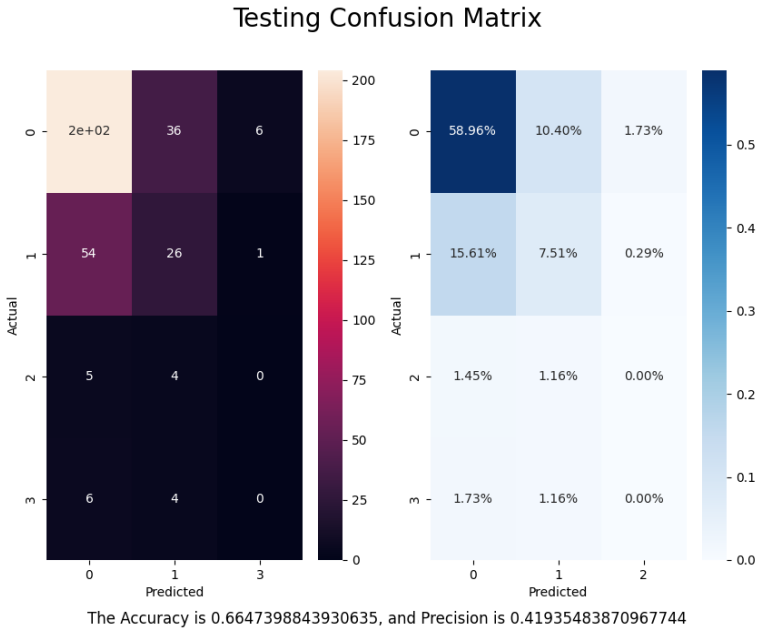


Figure 20: Graph: Car Linear Network Tuning Testing Confusion Matrix , Eta = 0.035



3.1.2 Testing Results

ETA = 0.01 gave the best accuracy during tuning at Accuracy = 67.341%.

Figure 21: Graph: Car LN 80% Data Performance for each k-Fold, Eta = 0.01

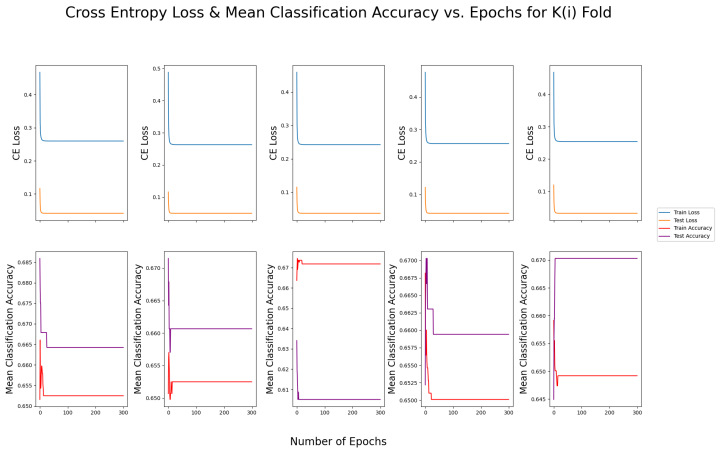


Figure 22: Graph: Car LN 80% Data Avg Performance For Cross Validation, Eta = 0.01

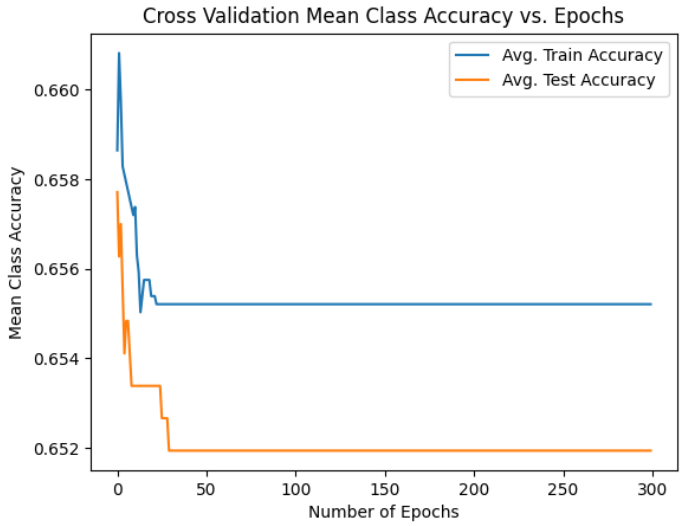


Figure 23: Graph: Car LN 80% Training Confusion Matrix , Eta = 0.01

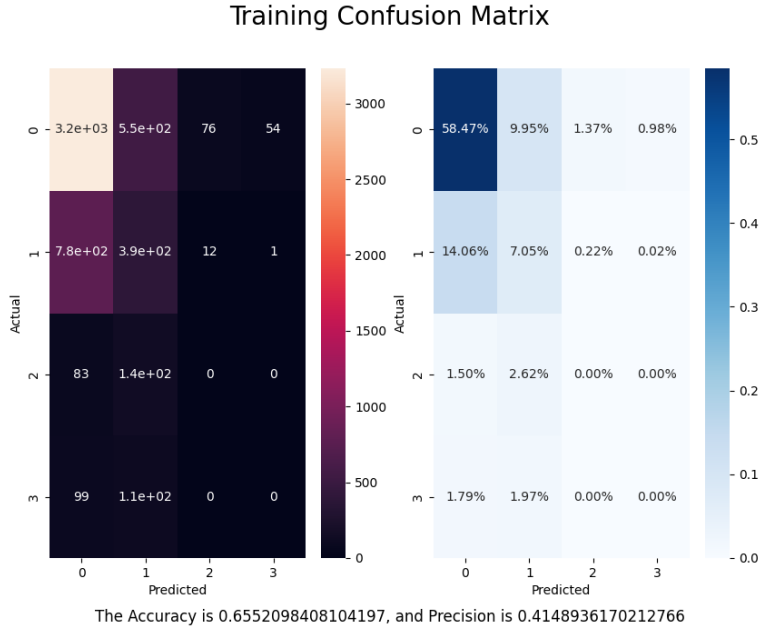
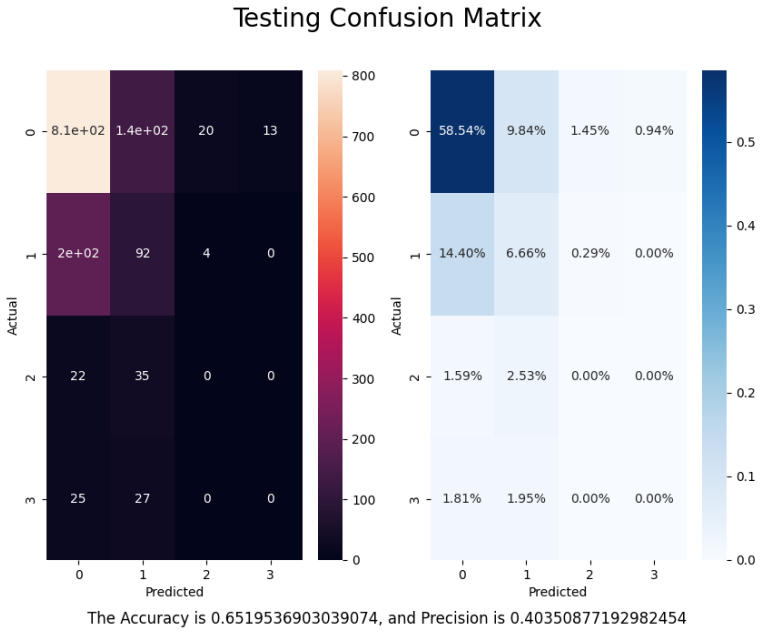


Figure 24: Graph: Car LN 80% Data Testing Confusion Matrix , Eta = 0.01



3.2 Voting Classification via Linear Network

3.2.1 Tuning Results

Figure 25: Graph: Voting LN Tuning Performance for each k-Fold, Eta = 0.01

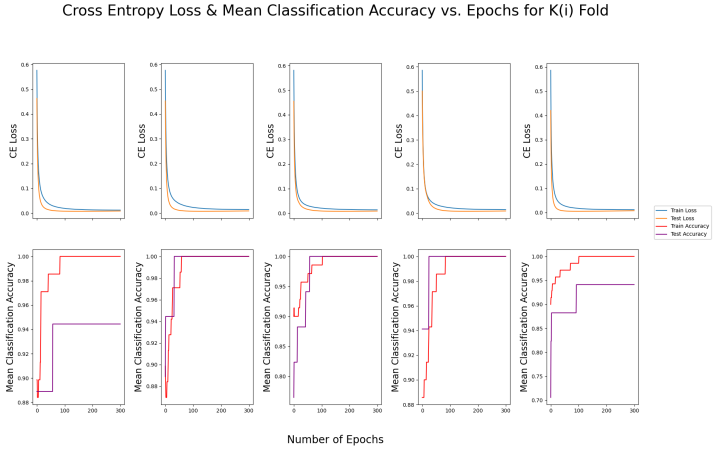


Figure 26: Graph: Voting LN Tuning Avg Performance For Cross Validation, Eta = 0.01

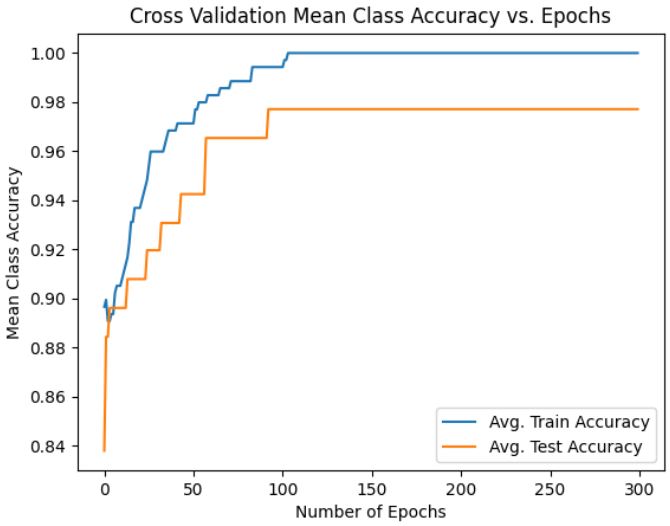


Figure 27: Graph: Voting Ln Tuning Training Confusion Matrix , Eta = 0.01

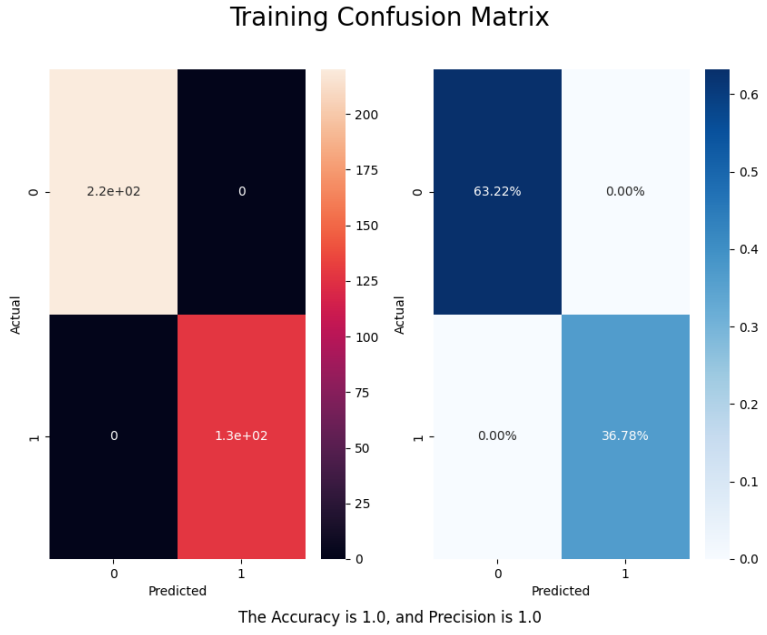


Figure 28: Graph: Voting LN Tuning Testing Confusion Matrix , Eta = 0.01

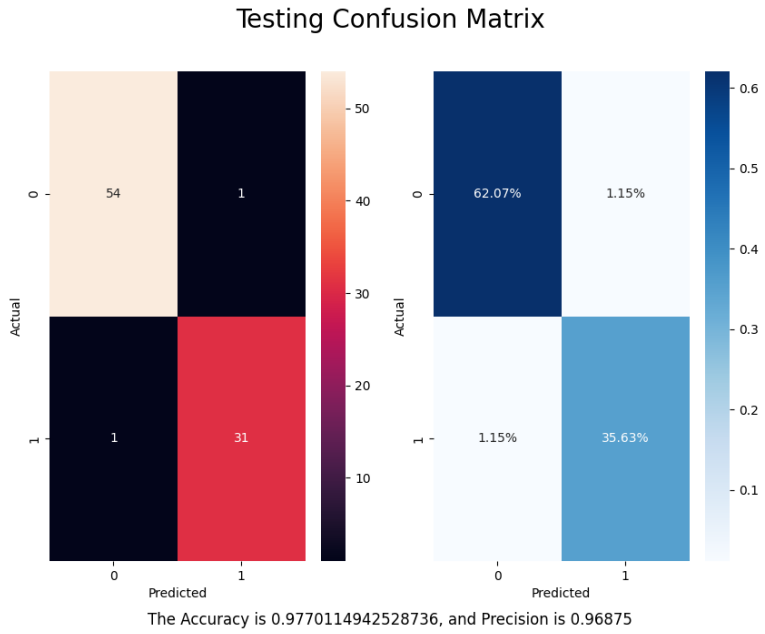


Figure 29: Graph: Voting LN Tuning Performance for each k-Fold, Eta = 0.015

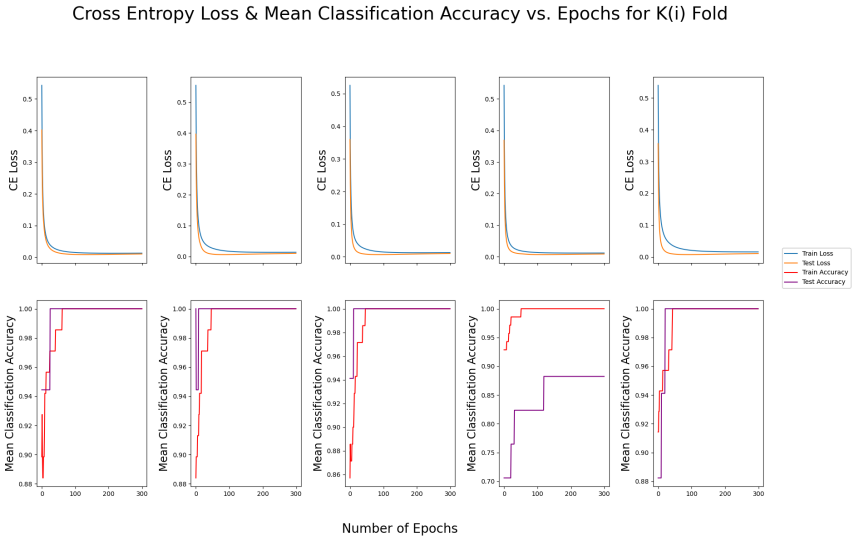


Figure 30: Graph: Voting LN Tuning Avg Performance For Cross Validation, Eta = 0.015

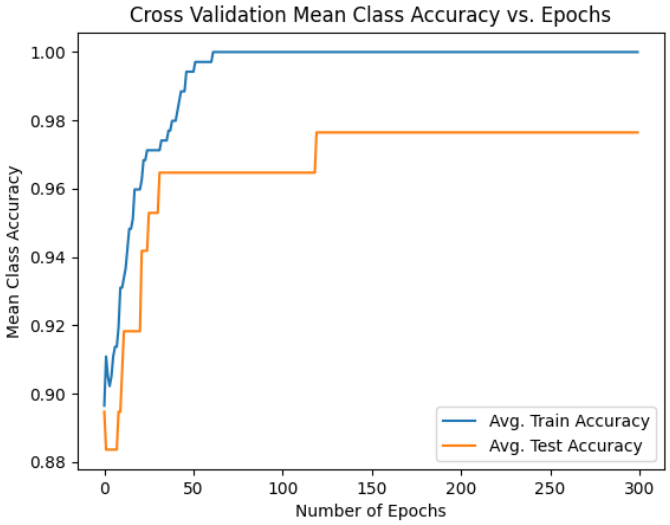


Figure 31: Graph: Voting LN Tuning Training Confusion Matrix , Eta = 0.015

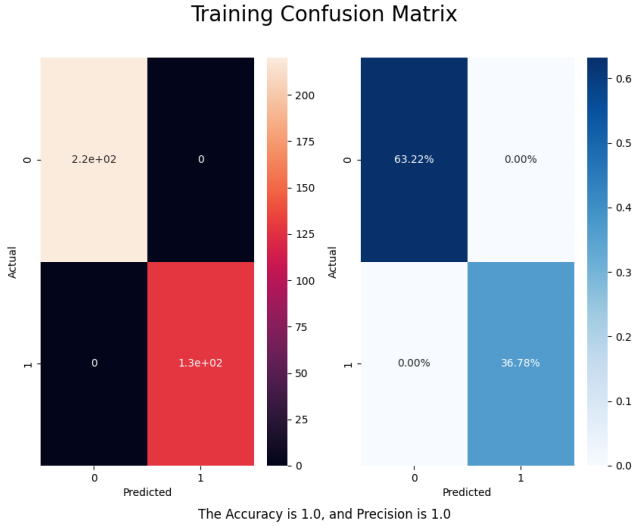


Figure 32: Graph: Voting LN Tuning Testing Confusion Matrix , Eta = 0.015

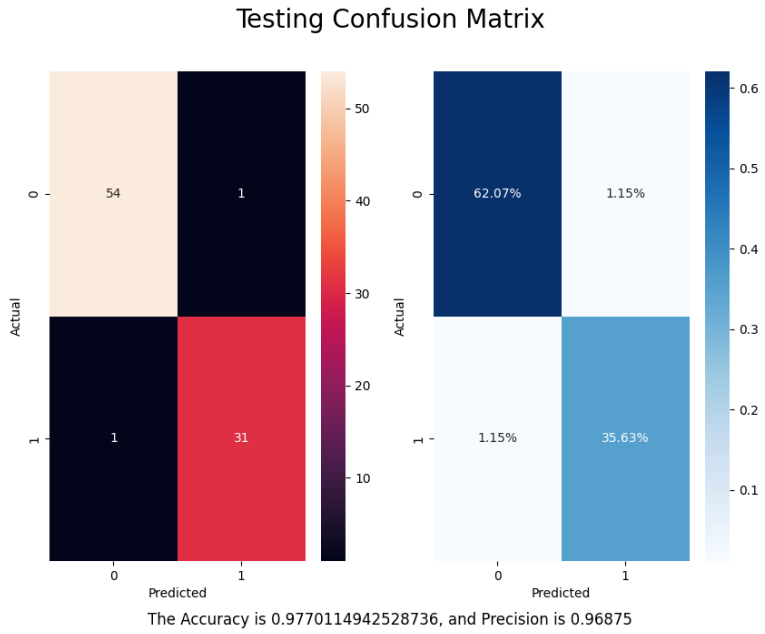


Figure 33: Graph: Voting LN Tuning Performance for each k-Fold, Eta = 0.02

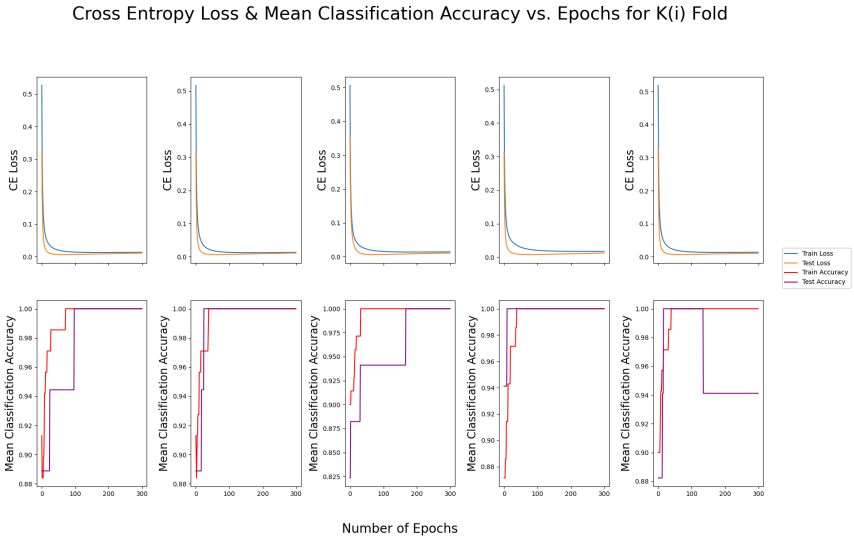


Figure 34: Graph: Voting LN Tuning Avg Performance For Cross Validation, Eta = 0.02

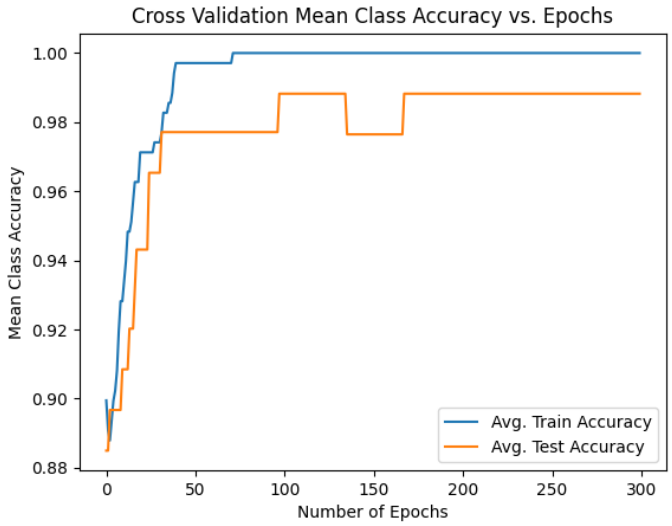


Figure 35: Graph: Voting LN Tuning Training Confusion Matrix , Eta = 0.02

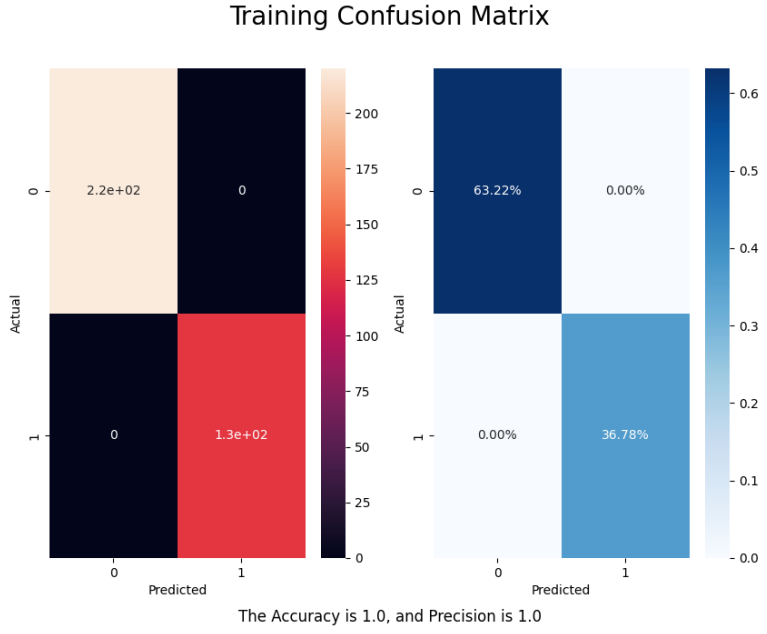


Figure 36: Graph: Voting LN Tuning Testing Confusion Matrix , Eta = 0.02

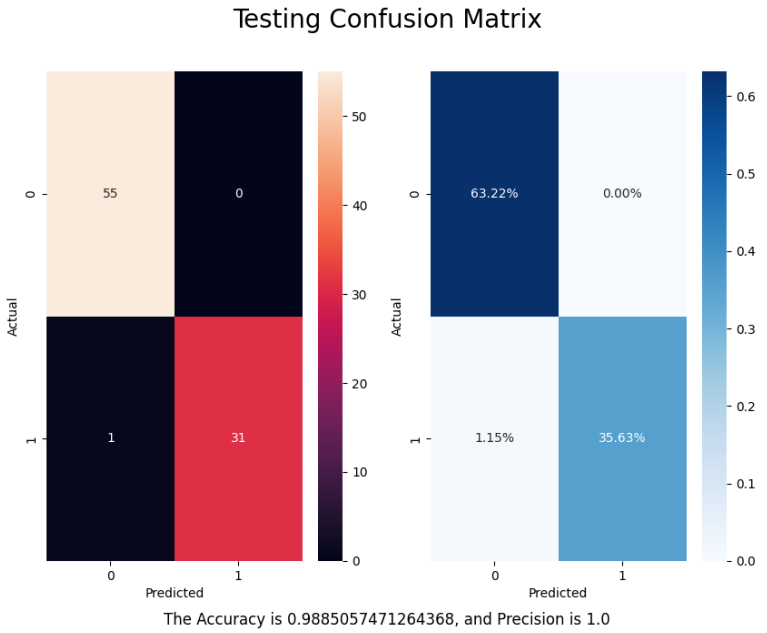


Figure 37: Graph: Voting LN Tuning Performance for each k-Fold, Eta = 0.03

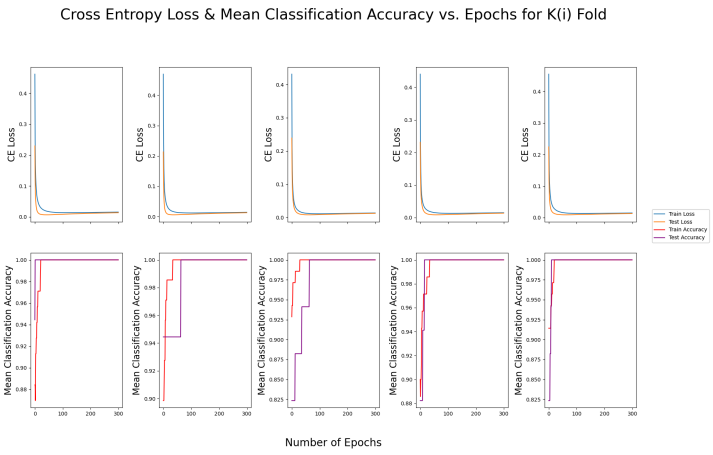


Figure 38: Graph: Voting LN Tuning Avg Performance For Cross Validation, Eta = 0.03

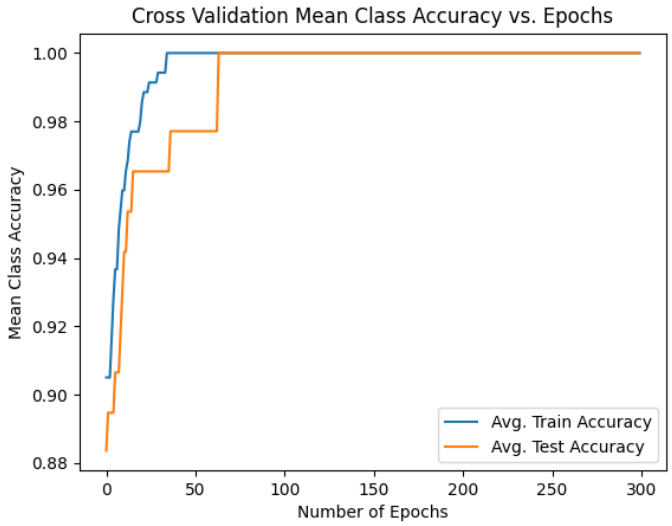


Figure 39: Graph: Voting LN Tuning Training Confusion Matrix , Eta = 0.03

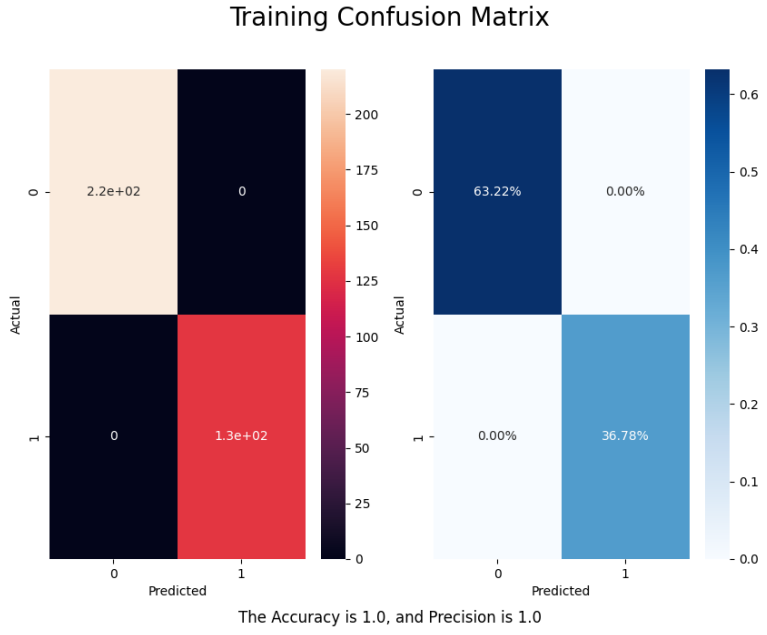


Figure 40: Graph: Voting LN Tuning Testing Confusion Matrix , Eta = 0.03

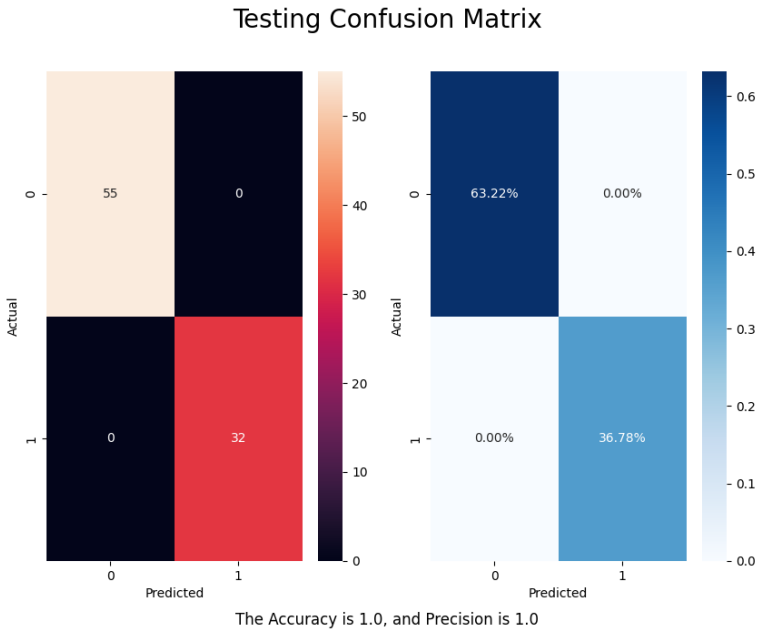


Figure 41: Graph: Voting LN Tuning Performance for each k-Fold, Eta = 0.035

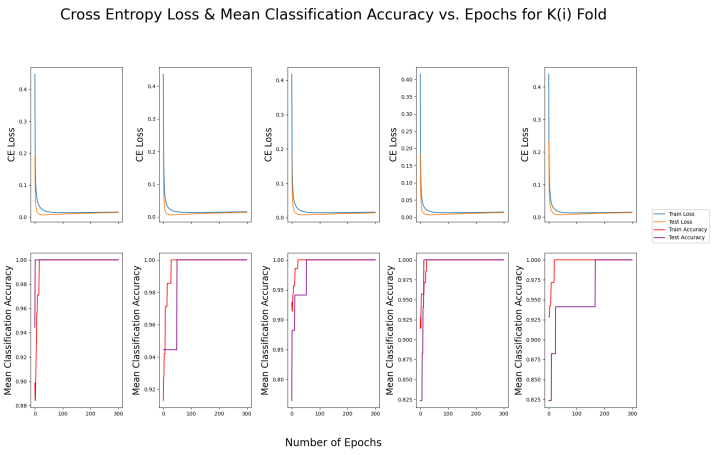


Figure 42: Graph:Voting LN Tuning Avg Performance For Cross Validation, Eta = 0.035

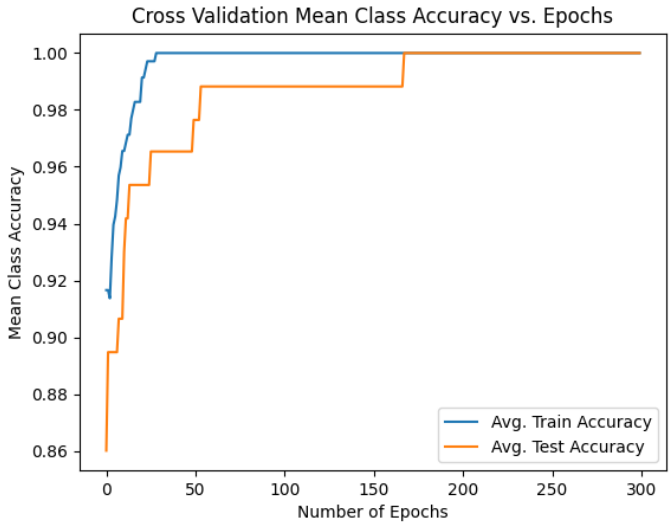


Figure 43: Graph: Voting LN Tuning Training Confusion Matrix , Eta = 0.035

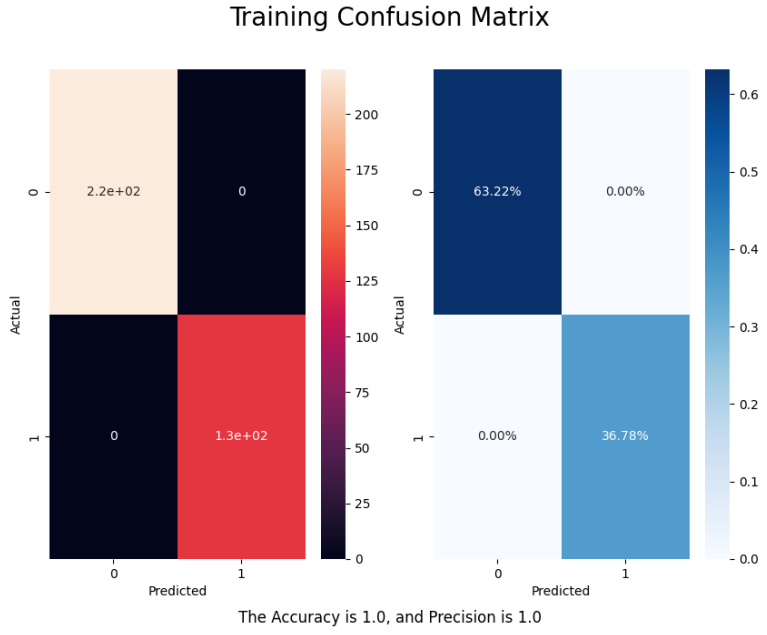
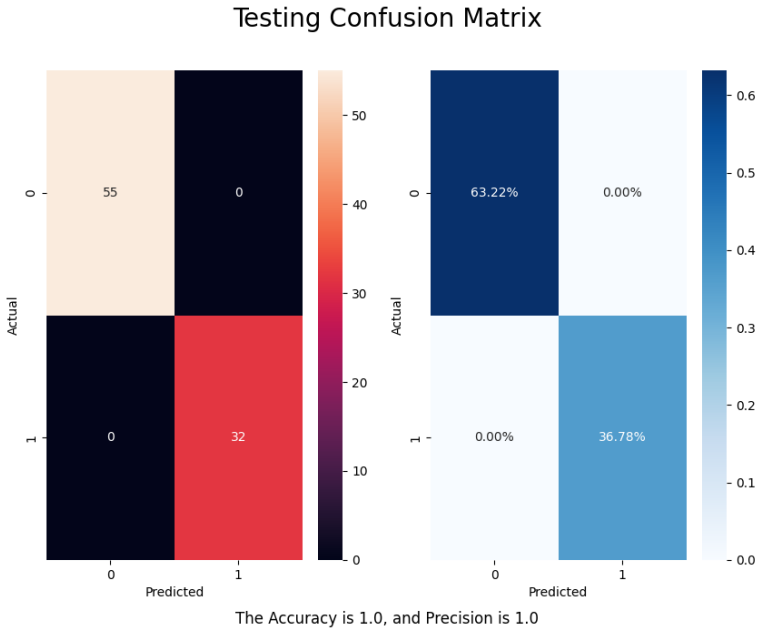


Figure 44: Graph: Voting LN Tuning Testing Confusion Matrix , Eta = 0.035



3.2.2 Testing Results

Both ETA = 0.03 and 0.035 gave the highest accuracy at 100 percent.

Figure 45: Graph: Voting LN 80% Data Performance for each k-Fold, Eta = 0.03

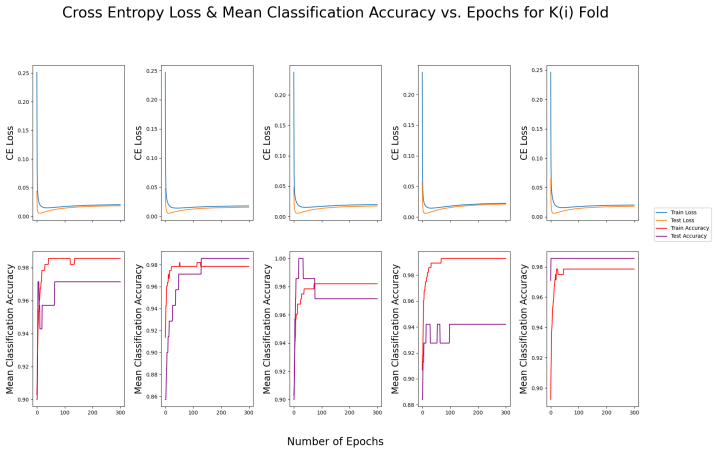


Figure 46: Graph: Voting LN 80% Data Avg Performance For Cross Validation, Eta = 0.03

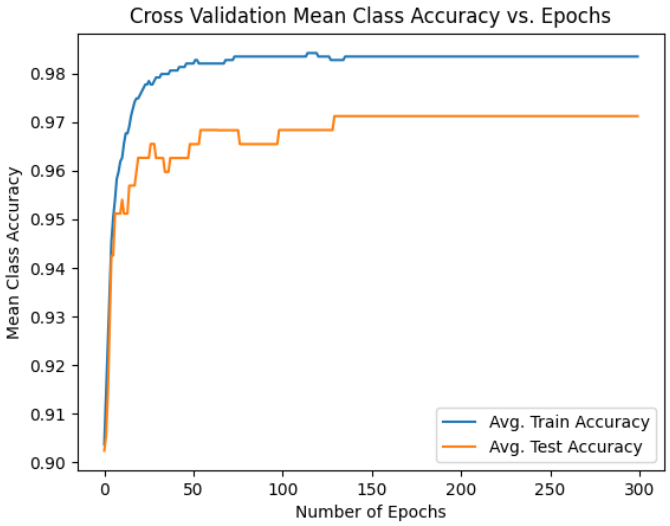


Figure 47: Graph: Voting LN 80% Data Training Confusion Matrix , Eta = 0.03

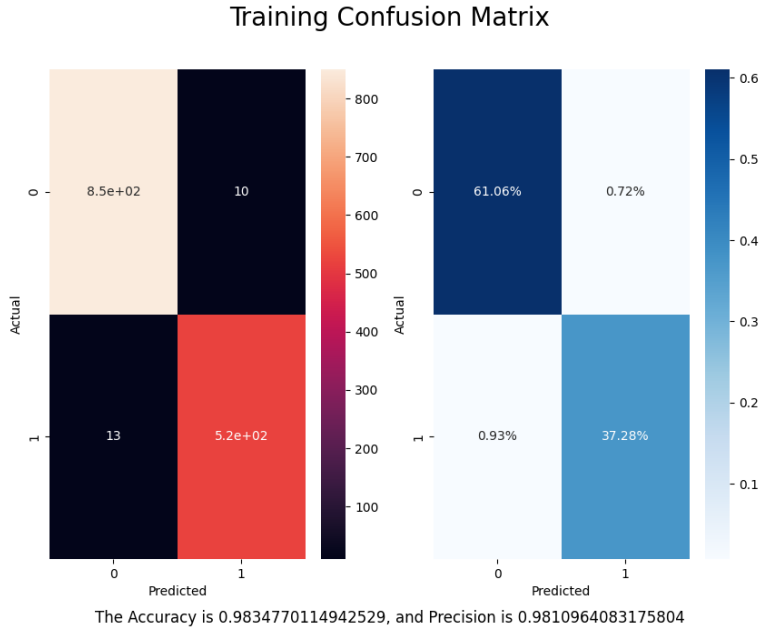
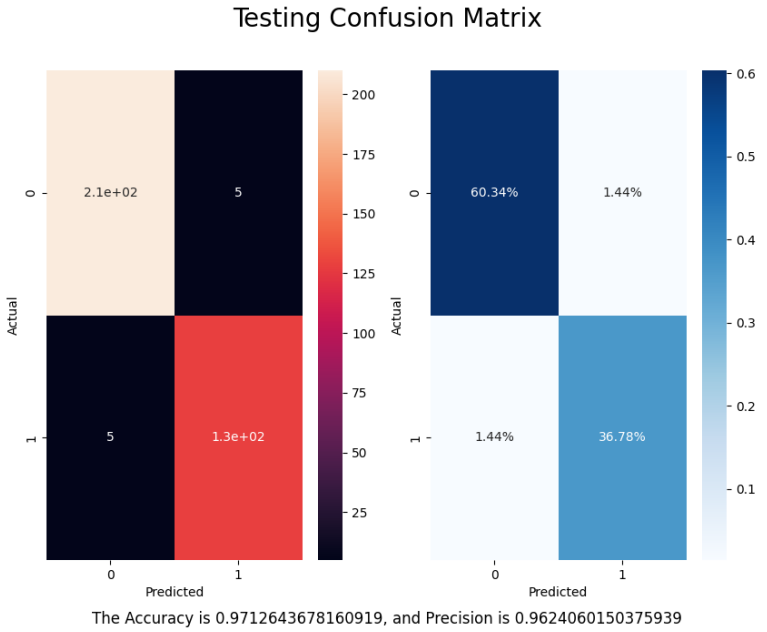


Figure 48: Graph: Voting LN 80% Data Testing Confusion Matrix , Eta = 0.03



3.3 Breast Cancer Classification via Linear Network

3.3.1 Tuning Results

Figure 49: Graph: Breast LN Cancer Tuning Performance for each k-Fold, Eta = 0.01

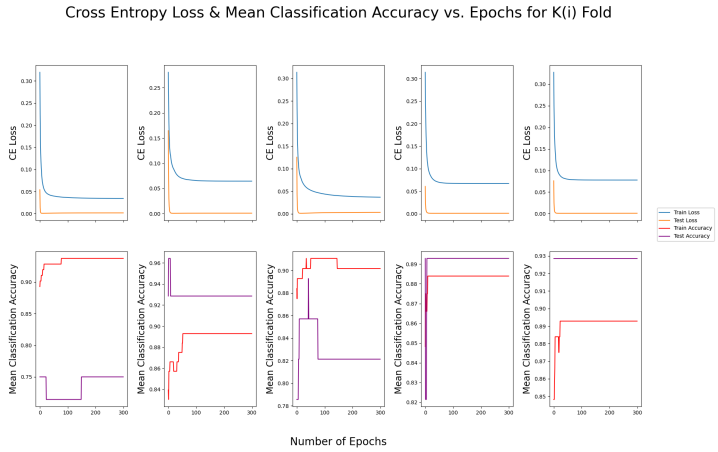


Figure 50: Graph: Breast LN Cancer Tuning Avg Performance For Cross Validation, Eta = 0.01

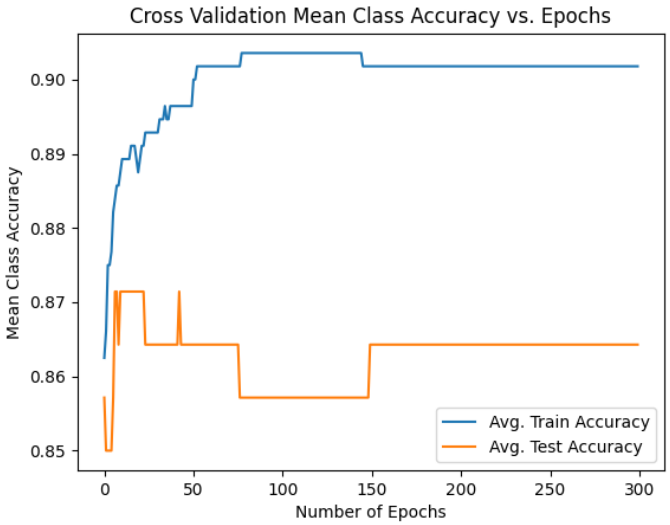


Figure 51: Graph: Breast LN Cancer Tuning Training Confusion Matrix , Eta = 0.01

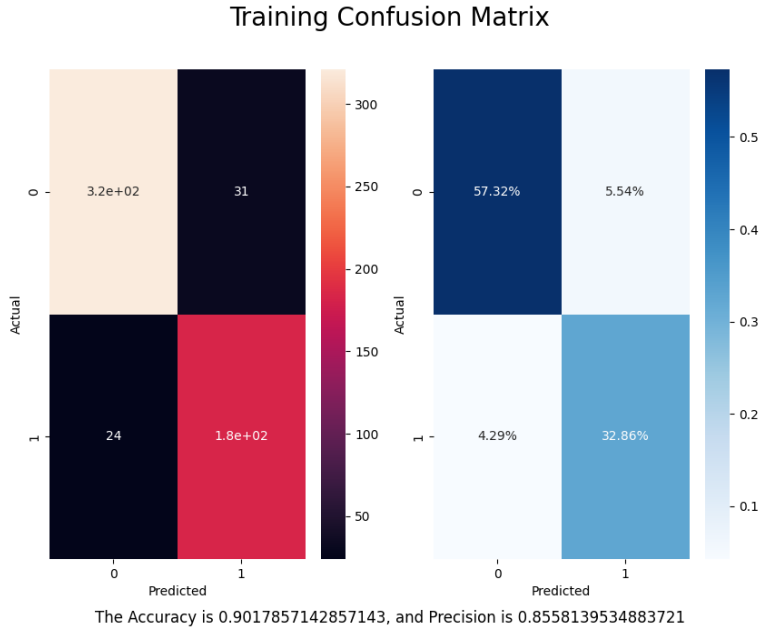


Figure 52: Graph: Breast LN Cancer Tuning Testing Confusion Matrix , Eta = 0.01

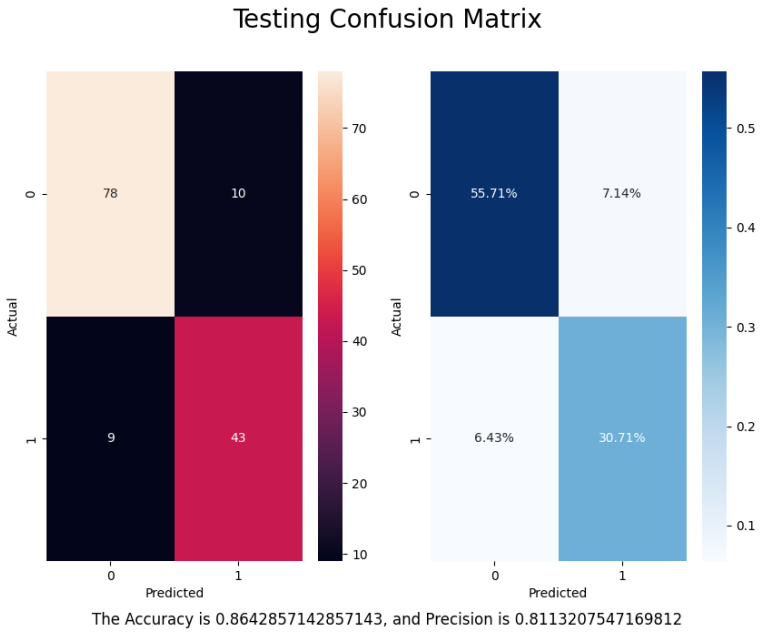


Figure 53: Graph: Breast LN Cancer Tuning Performance for each k-Fold, Eta = 0.015

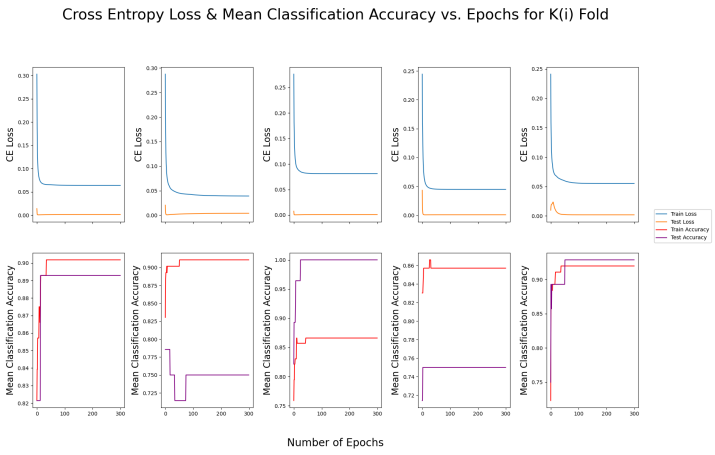


Figure 54: Graph: Breast LN Cancer Tuning Avg Performance For Cross Validation, Eta = 0.015

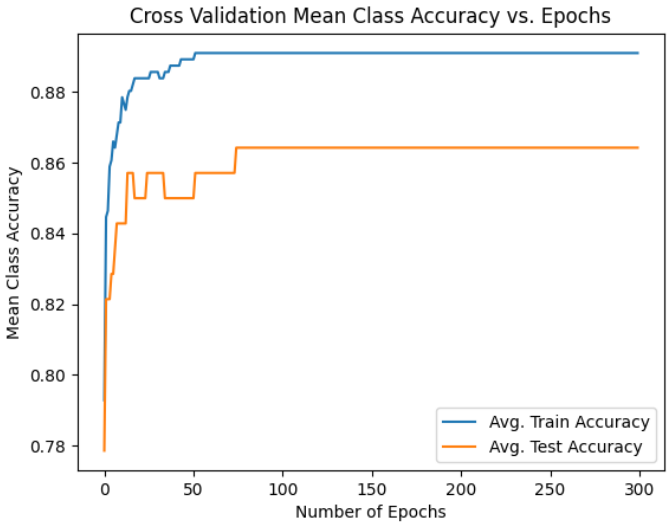


Figure 55: Graph: Breast LN Cancer Tuning Training Confusion Matrix , Eta = 0.015

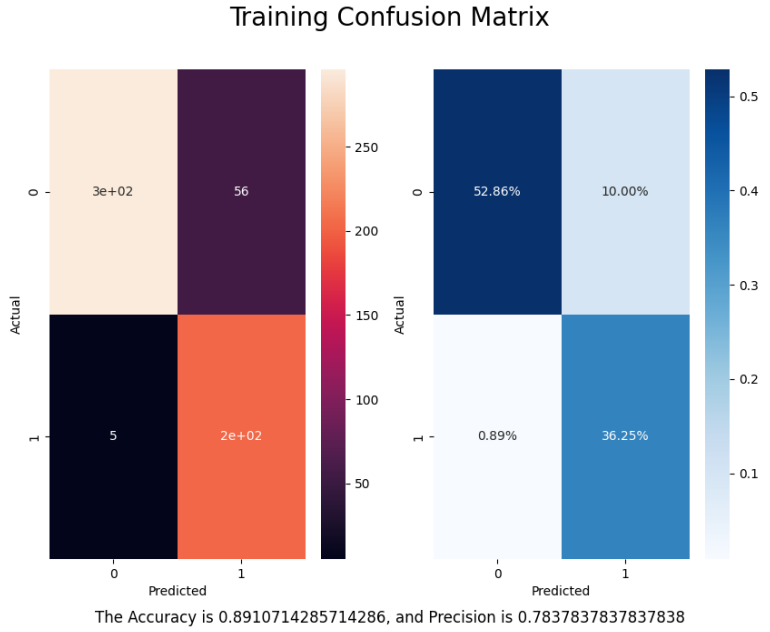


Figure 56: Graph: Breast LN Cancer Tuning Testing Confusion Matrix , Eta = 0.015

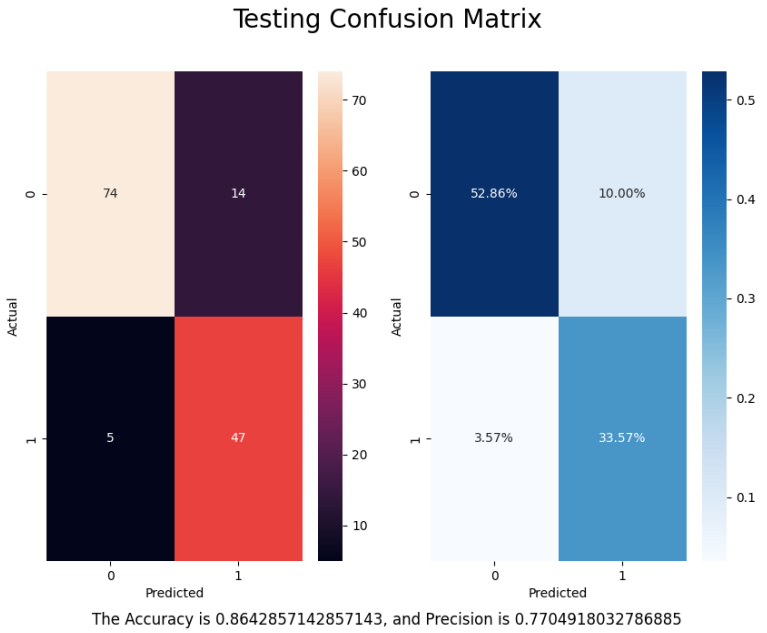


Figure 57: Graph: Breast LN Cancer Tuning Performance for each k-Fold, Eta = 0.02

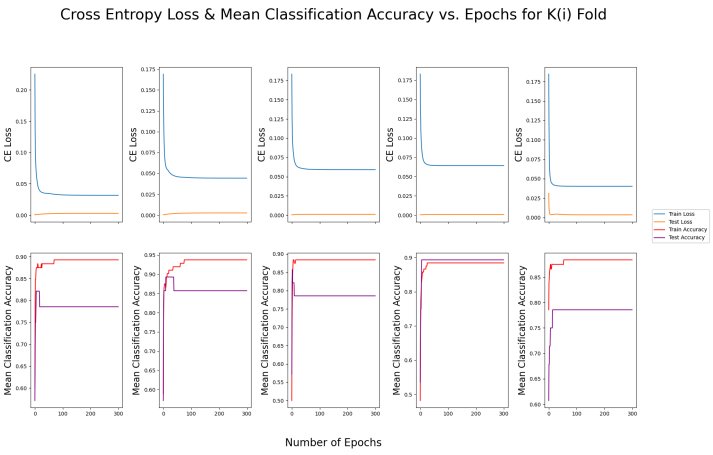


Figure 58: Graph: Breast LN Cancer Tuning Avg Performance For Cross Validation, Eta = 0.02

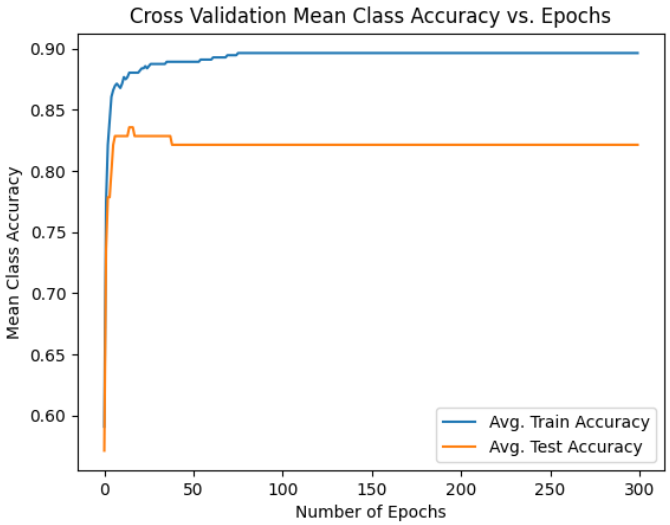


Figure 59: Graph: Breast LN Cancer Tuning Training Confusion Matrix , Eta = 0.02

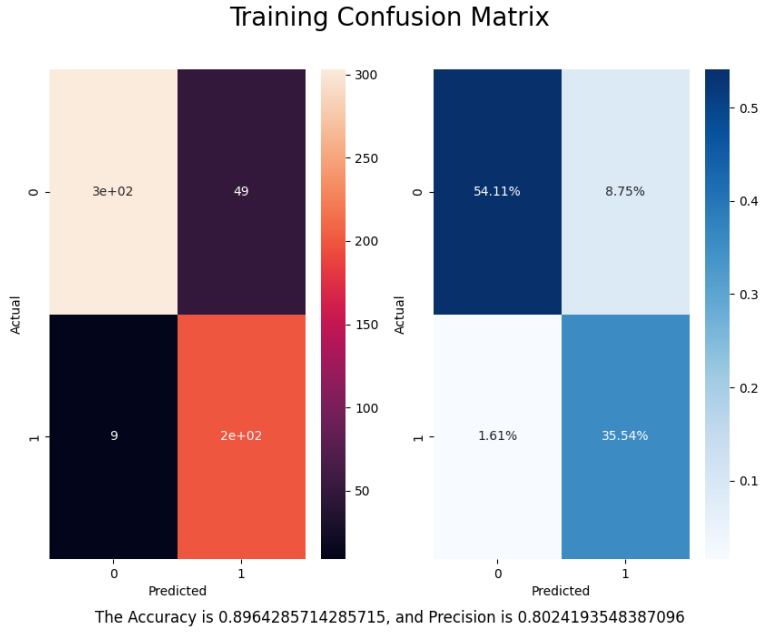


Figure 60: Graph: Breast LN Cancer Tuning Testing Confusion Matrix , Eta = 0.02

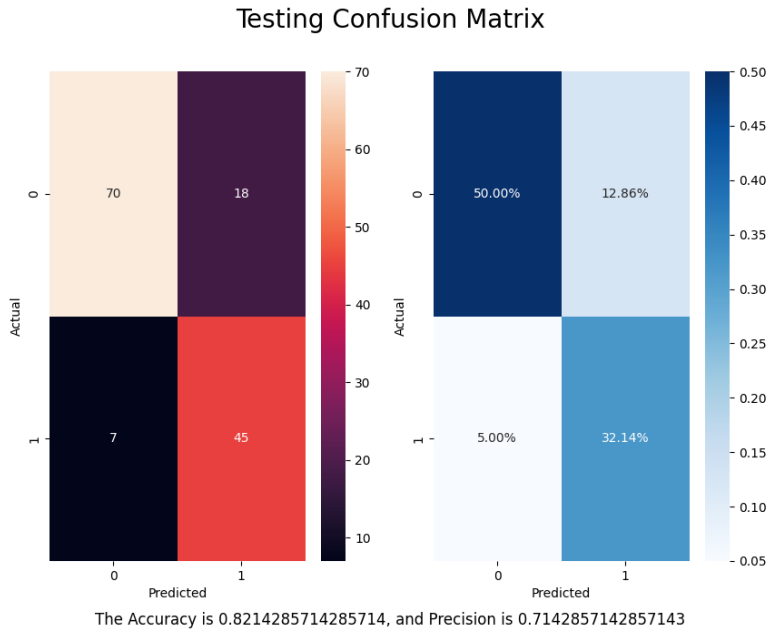


Figure 61: Graph: Breast LN Cancer Tuning Performance for each k-Fold, Eta = 0.03

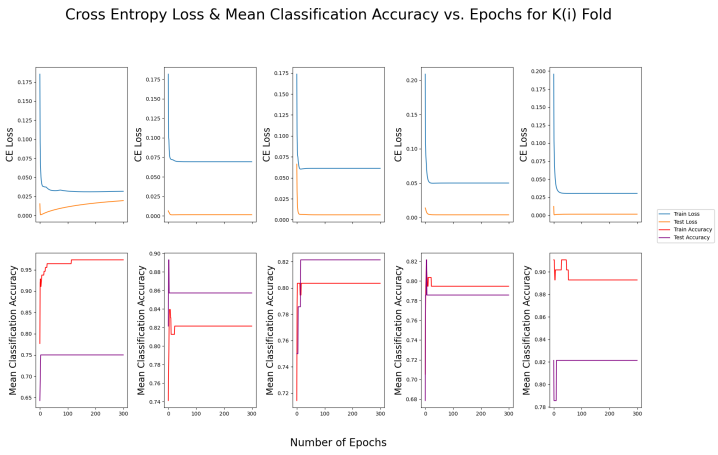


Figure 62: Graph: Breast LN Cancer Tuning Avg Performance For Cross Validation, Eta = 0.03

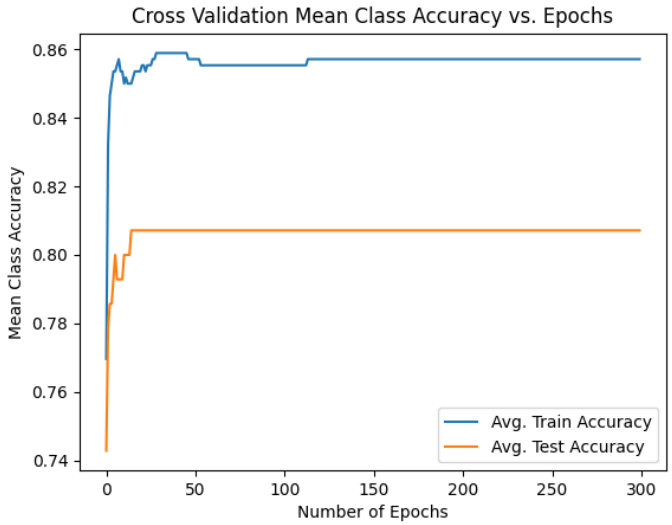


Figure 63: Graph: Breast LN Cancer Tuning Training Confusion Matrix , Eta = 0.03

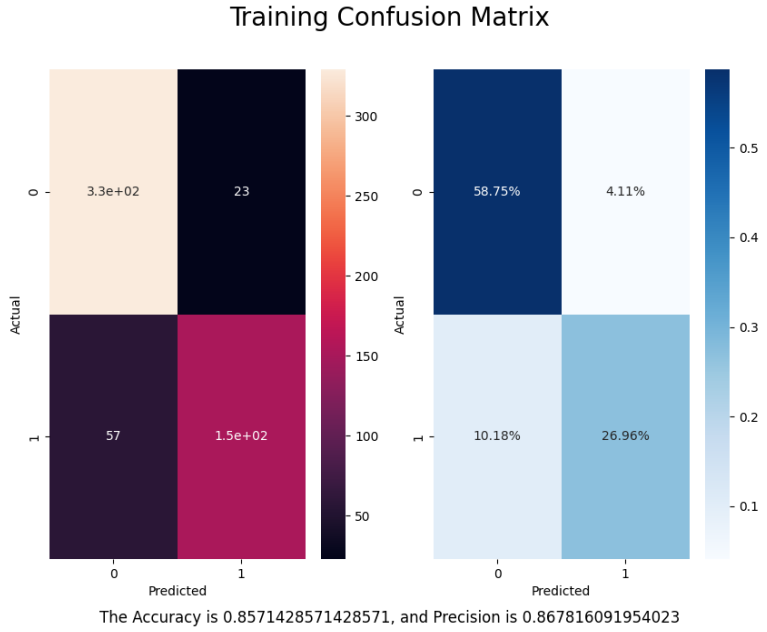


Figure 64: Graph: Breast LN Cancer Tuning Testing Confusion Matrix , Eta = 0.03

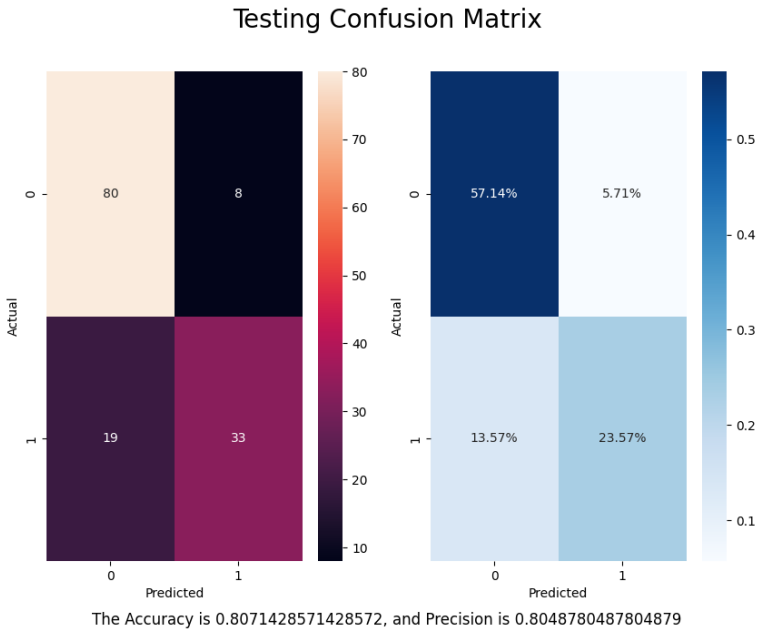


Figure 65: Graph: Breast LN Cancer Tuning Performance for each k-Fold, Eta = 0.035

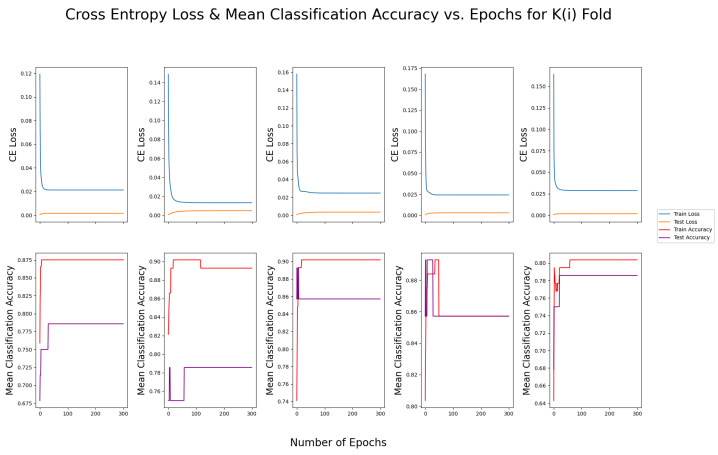


Figure 66: Graph: Breast LN Cancer Tuning Avg Performance For Cross Validation, Eta = 0.035

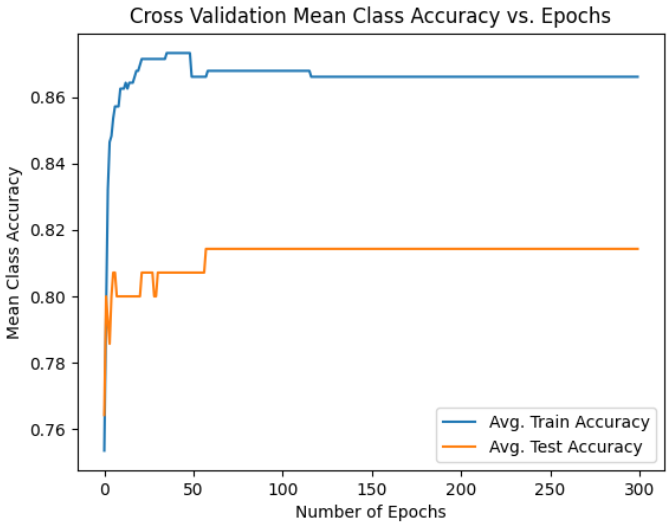


Figure 67: Graph: Breast LN Cancer Tuning Training Confusion Matrix , Eta = 0.035

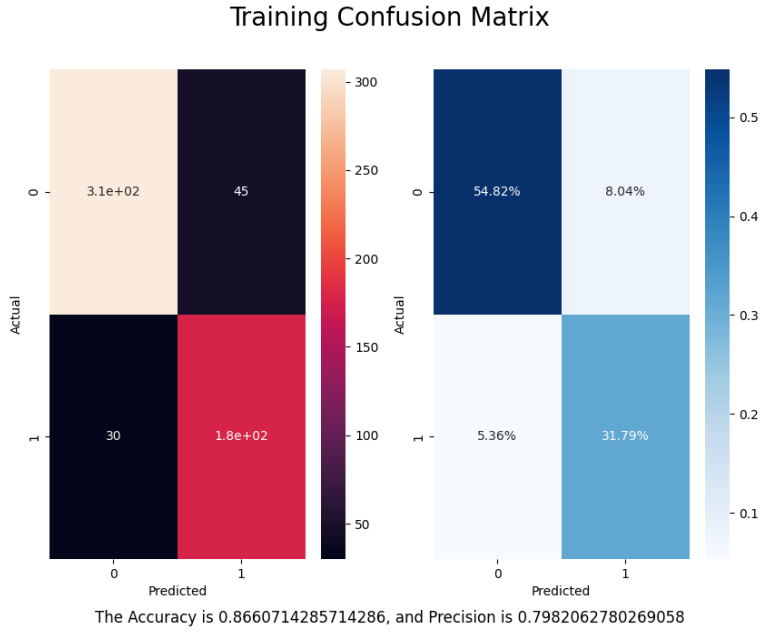
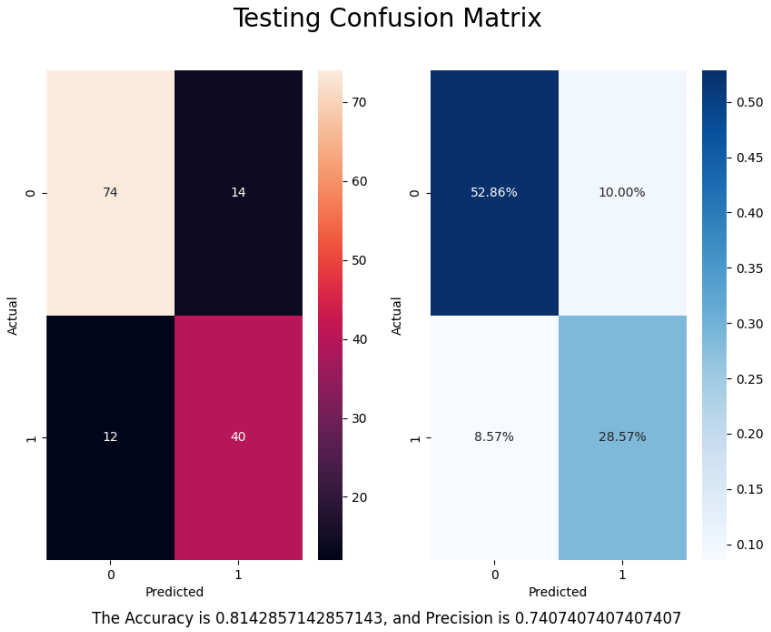


Figure 68: Graph: Breast LN Cancer Tuning Testing Confusion Matrix , Eta = 0.035



3.3.2 Testing Results

Both ETA = 0.01 and 0.015 give the same results for the testing accuracy, but precision is higher in ETA = 0.01 so that value is chosen as the tuned hyper parameter value.

Figure 69: Graph: Breast LN Cancer 80% Data Performance for each k-Fold, Eta = 0.01

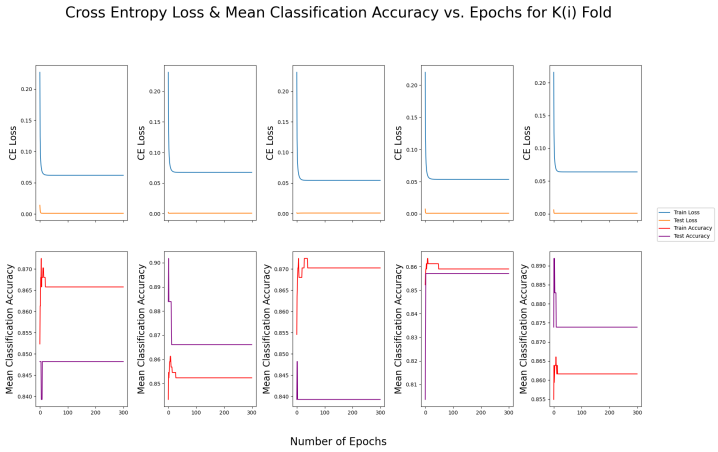


Figure 70: Graph: Breast LN Cancer 80% Data Avg Performance For Cross Validation, Eta = 0.01

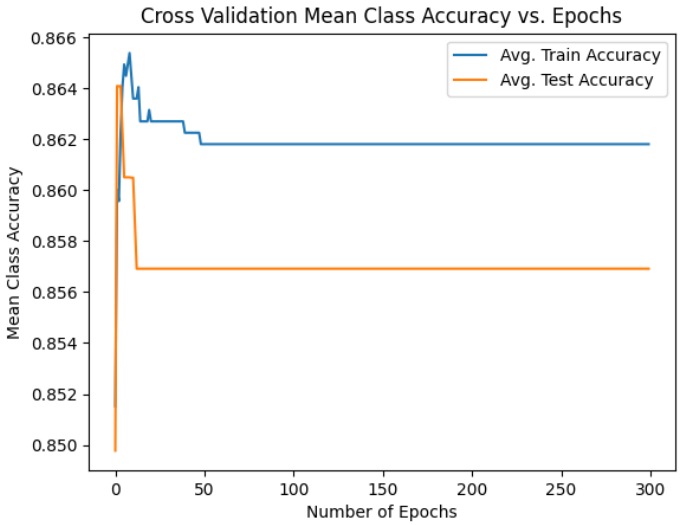


Figure 71: Graph: Breast LN Cancer 80% Data Training Confusion Matrix , Eta = 0.01

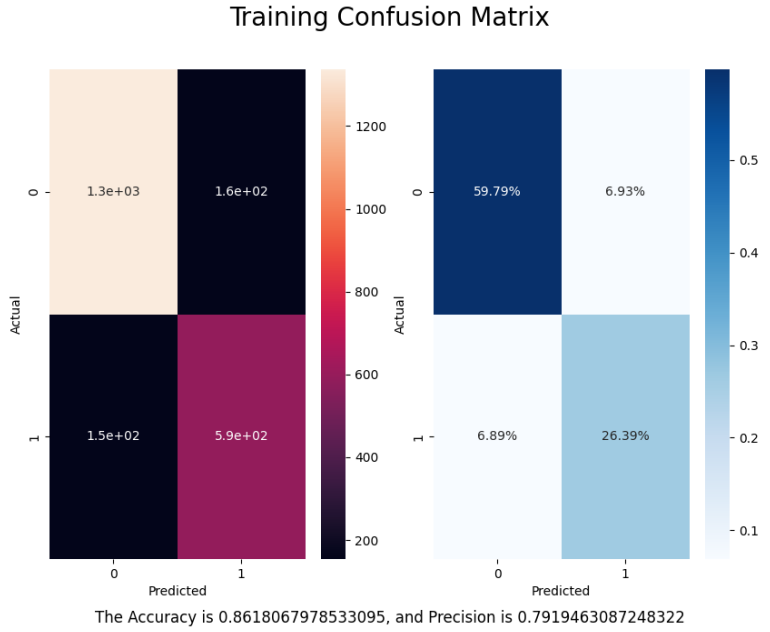
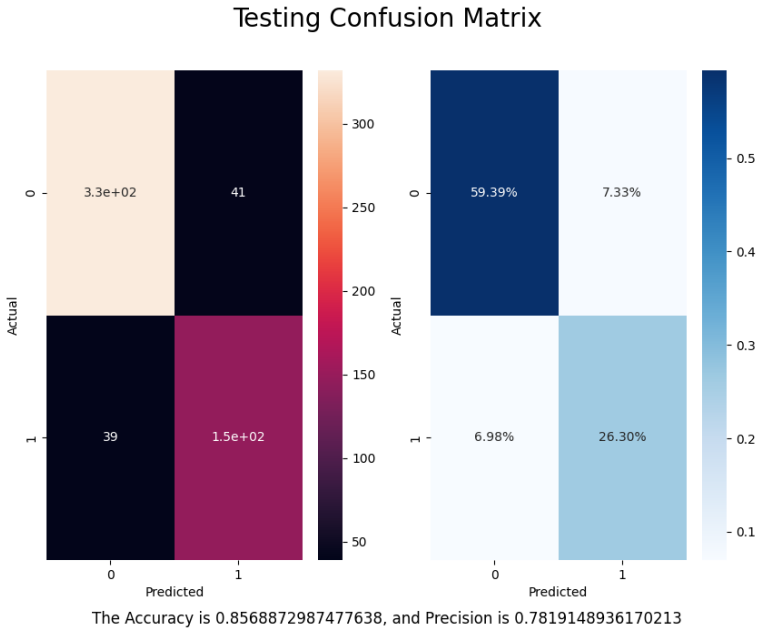


Figure 72: Graph: Breast LN Cancer 80% Data Testing Confusion Matrix , Eta = 0.01



3.4 Abalone Regression via Linear Network

3.4.1 Tuning Results

Figure 73: Graph: Abalone LN Tuning Performance for each k-Fold, Eta = 0.01

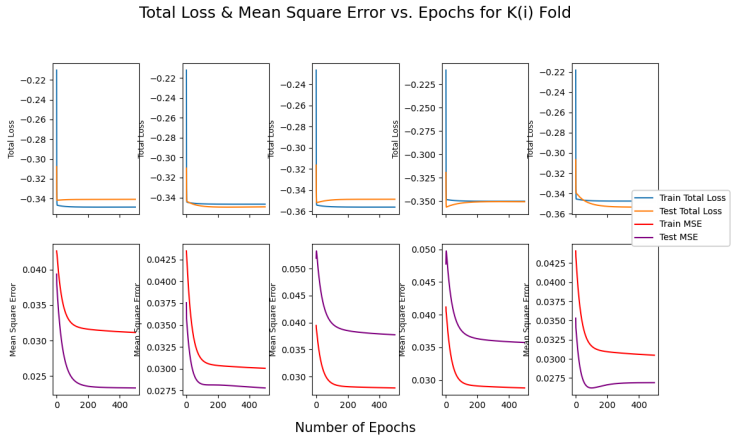


Figure 74: Graph: Abalone LN Tuning Avg Performance For Cross Validation, Eta = 0.01

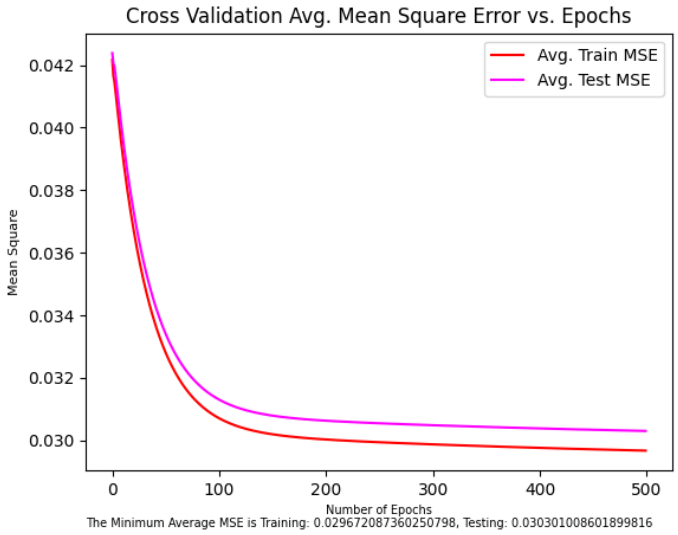


Figure 75: Graph: Abalone LN Tuning yPred vs. yActual , Eta = 0.01

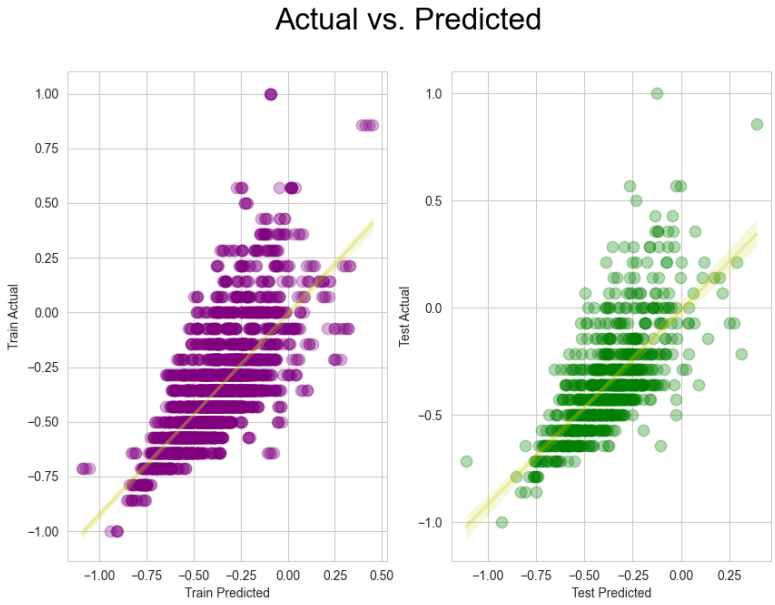


Figure 76: Graph: Abalone LN Tuning Performance for each k-Fold, Eta = 0.015

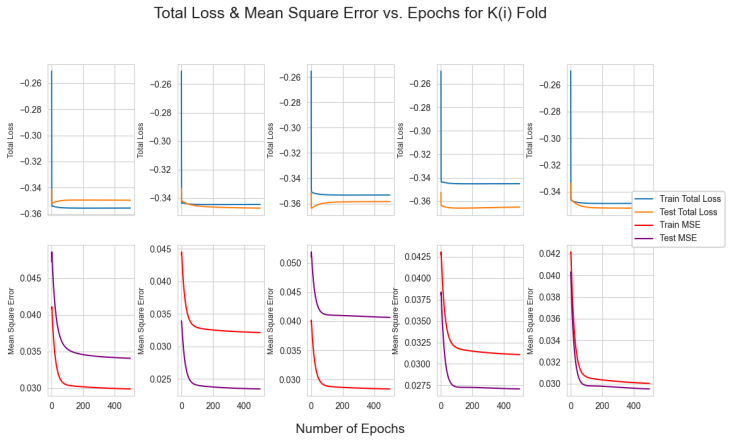


Figure 77: Graph: Abalone LN Avg Tuning Performance For Cross Validation, Eta = 0.015

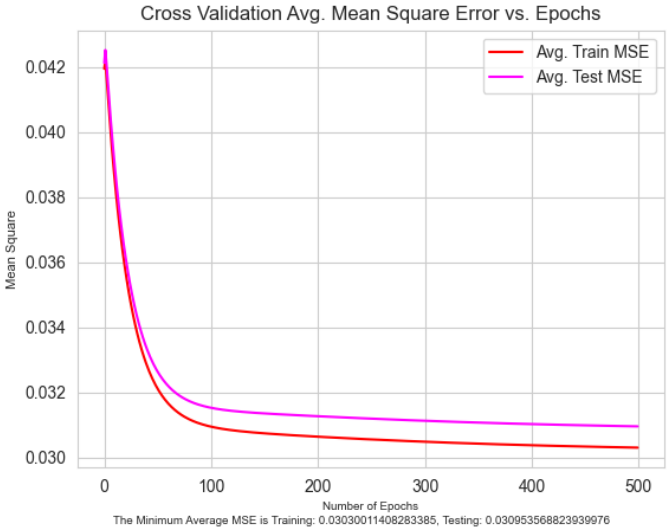


Figure 78: Graph: Abalone LN Tuning yPred vs. yActual , Eta = 0.015

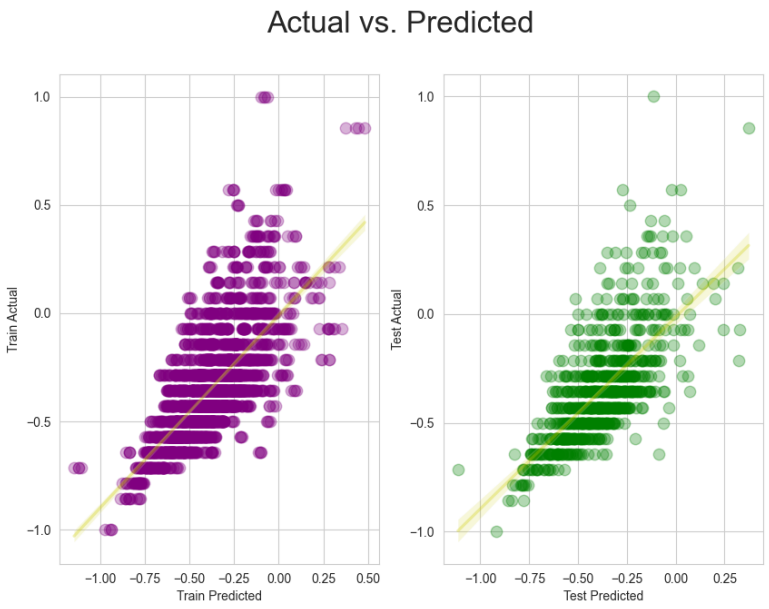


Figure 79: Graph: Abalone LN Tuning Performance for each k-Fold, Eta = 0.02

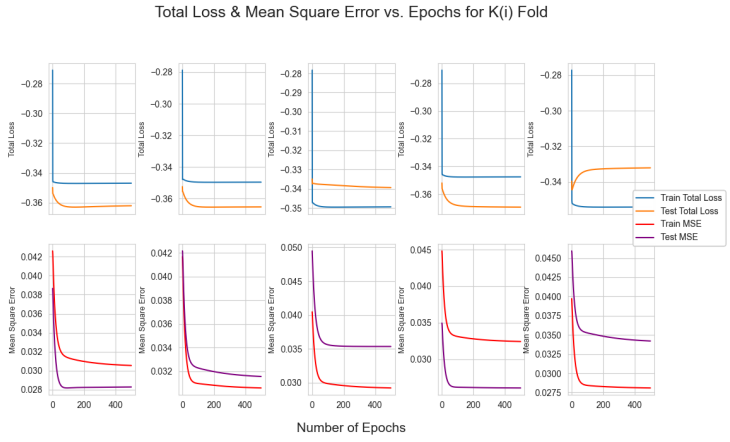


Figure 80: Graph: Abalone LN Tuning Avg Performance For Cross Validation, Eta = 0.02

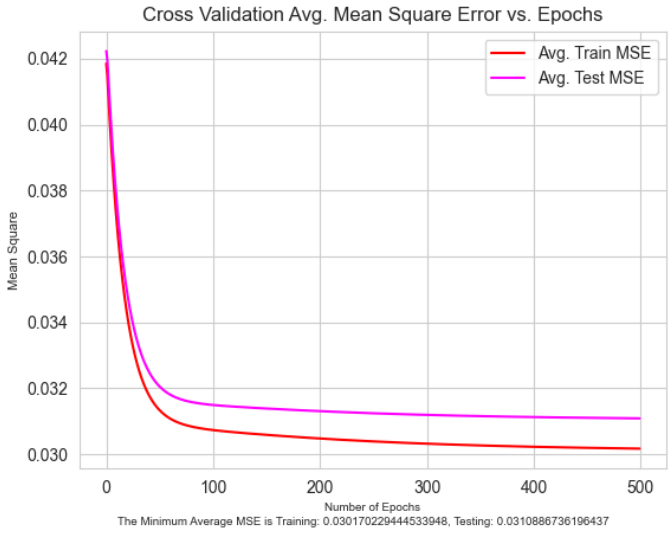


Figure 81: Graph: Abalone LN Tuning yPred vs. yActual , Eta = 0.02

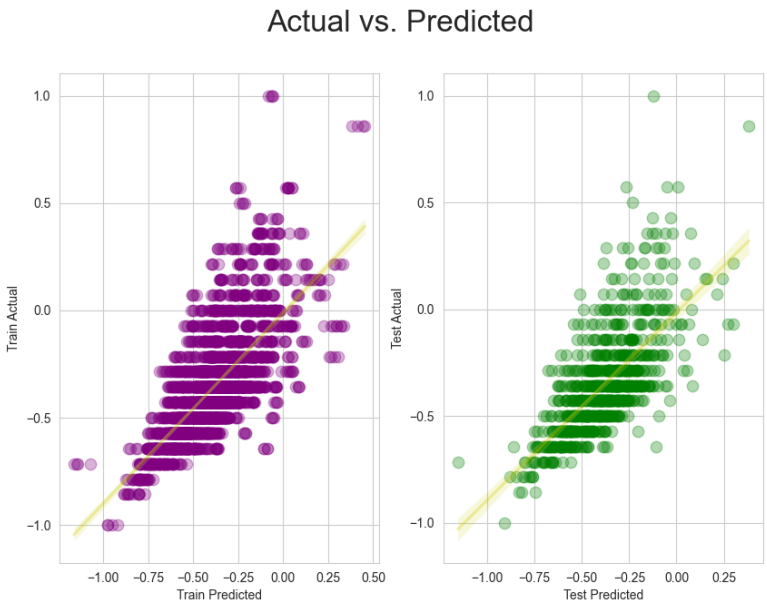


Figure 82: Graph: Abalone LN Tuning Performance for each k-Fold, Eta = 0.03

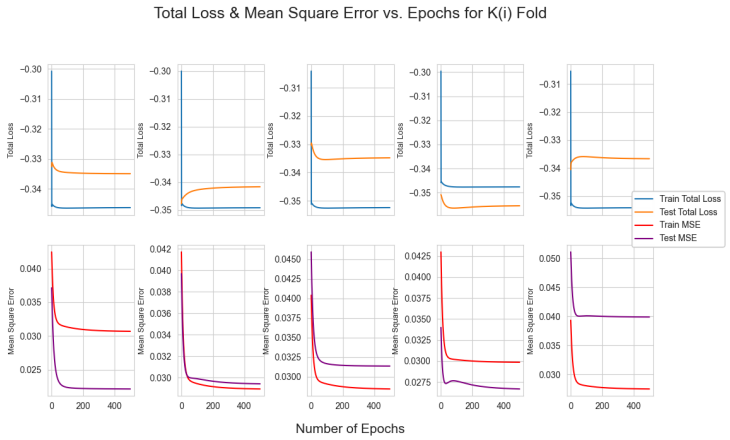


Figure 83: Graph: Abalone LN Tuning Avg Performance For Cross Validation, Eta = 0.03

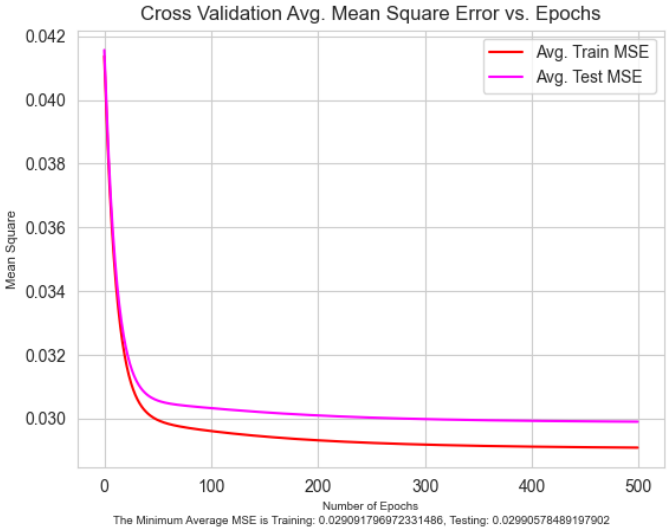


Figure 84: Graph: Abalone LN Tuning yPred vs. yActual , Eta = 0.03

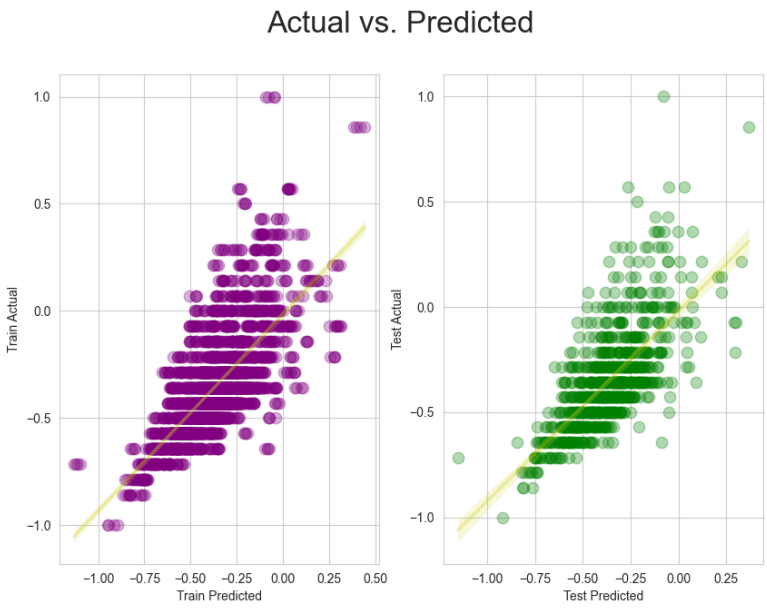


Figure 85: Graph: Abalone LN Performance for each k-Fold, Eta = 0.035

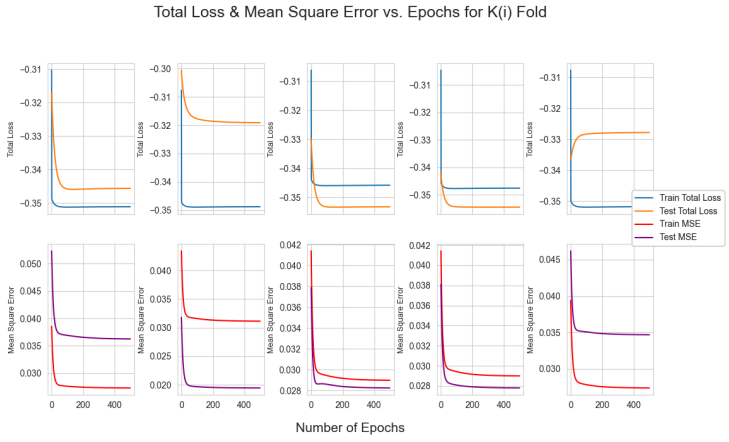


Figure 86: Graph: Abalone LN Avg Performance For Cross Validation, Eta = 0.035

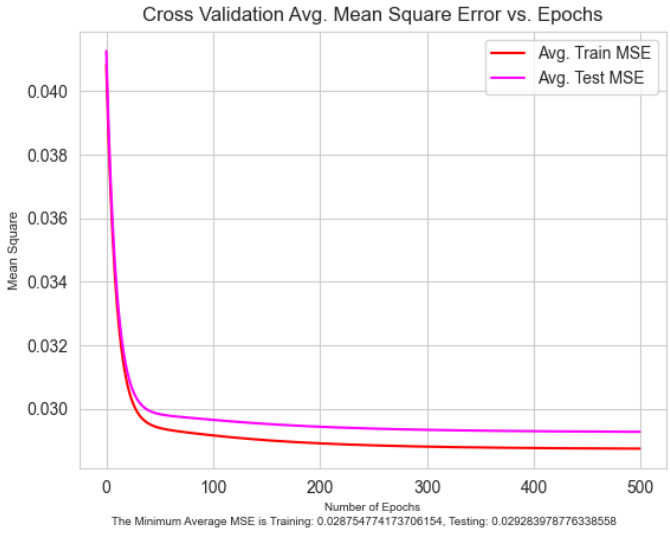
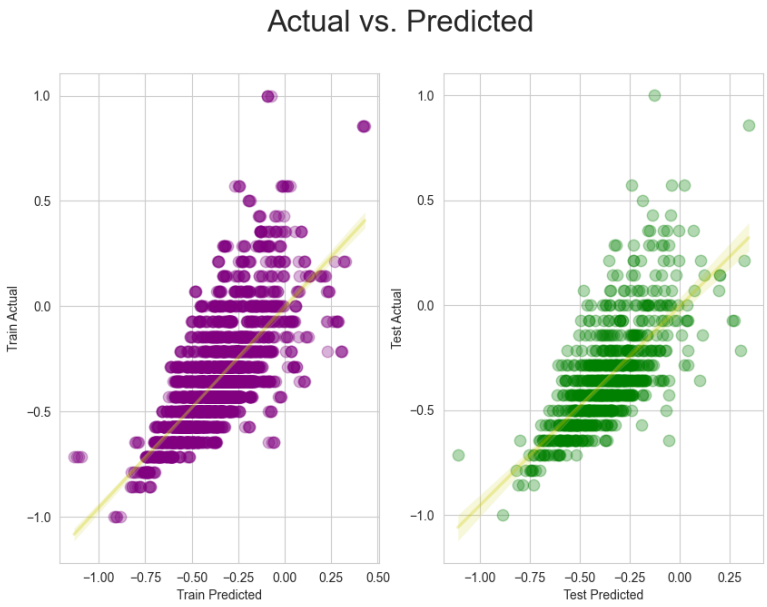


Figure 87: Graph: Abalone LN Tuning yPred vs. yActual , Eta = 0.035



3.4.2 Testing Results

Eta = 0.035 gives the lowest MSE on the tuning testing at MSE = 0.02928.

Figure 88: Graph: Abalone LN 80% Data Performance for each k-Fold, Eta = 0.035

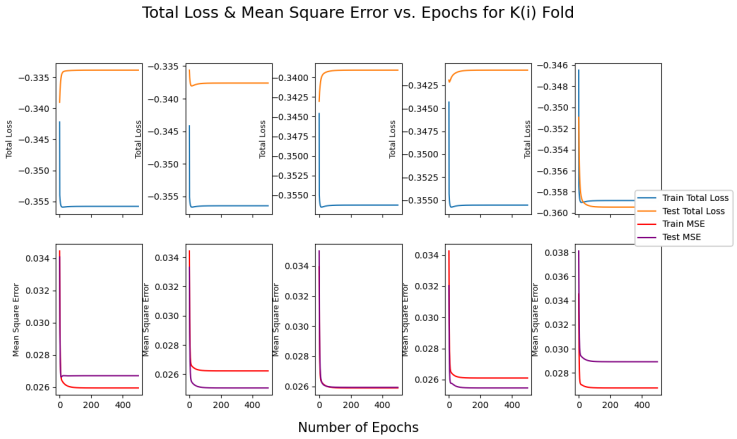


Figure 89: Graph: Abalone LN 80% Data Avg Performance For Cross Validation, Eta = 0.035

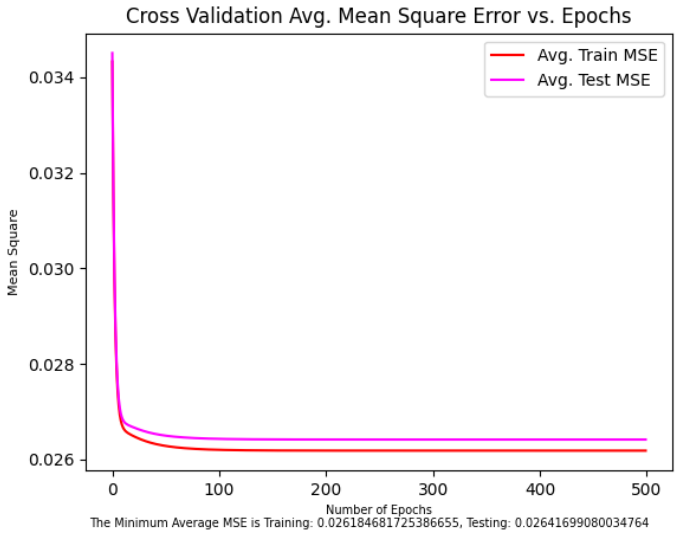
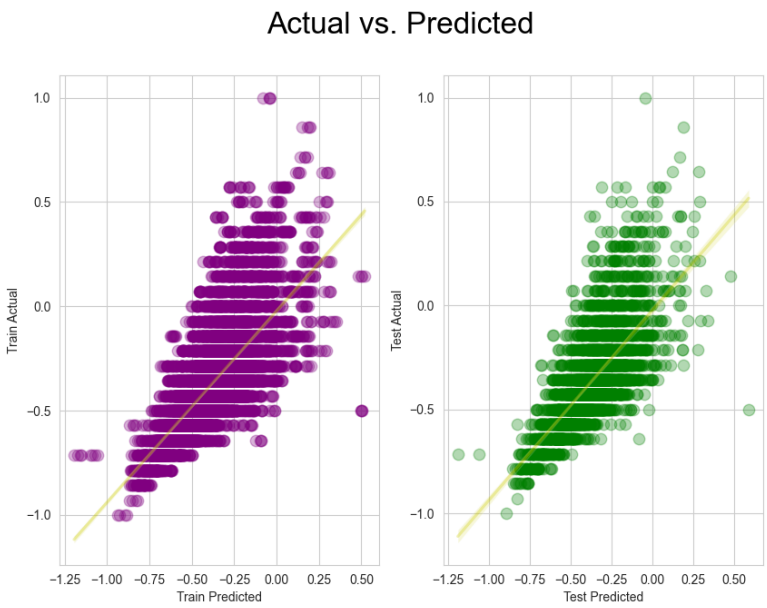


Figure 90: Graph: Abalone Remainder Data yPred vs. yActual , Eta = 0.035



3.5 Computer Regression via Linear Network

3.5.1 Tuning Results

Figure 91: Graph: Computer LN Tuning Performance for each k-Fold, Eta = 0.01

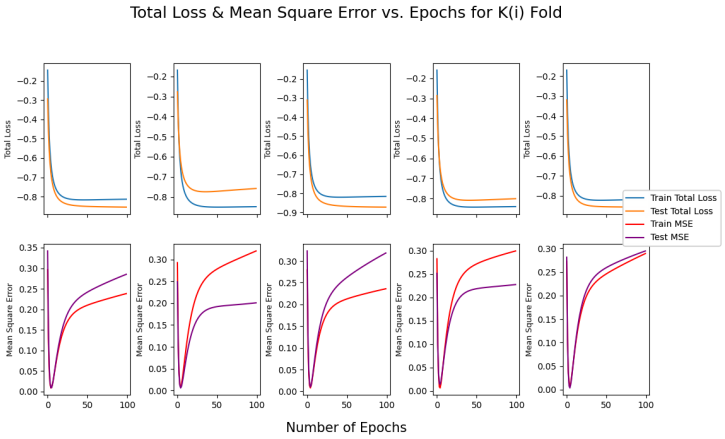


Figure 92: Graph: Computer LN Tuning Avg Performance For Cross Validation, Eta = 0.01

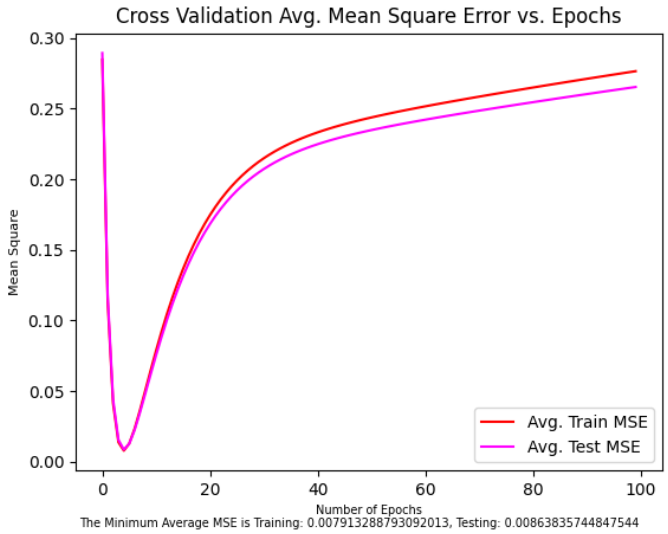


Figure 93: Graph: Computer LN Tuning yPred vs. yActual , Eta = 0.01

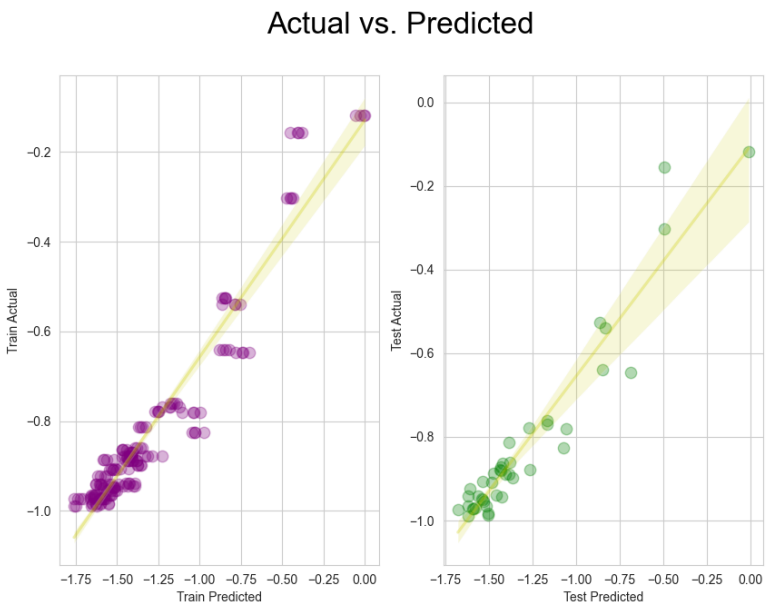


Figure 94: Graph: Computer LN Tuning Performance for each k-Fold, Eta = 0.0125

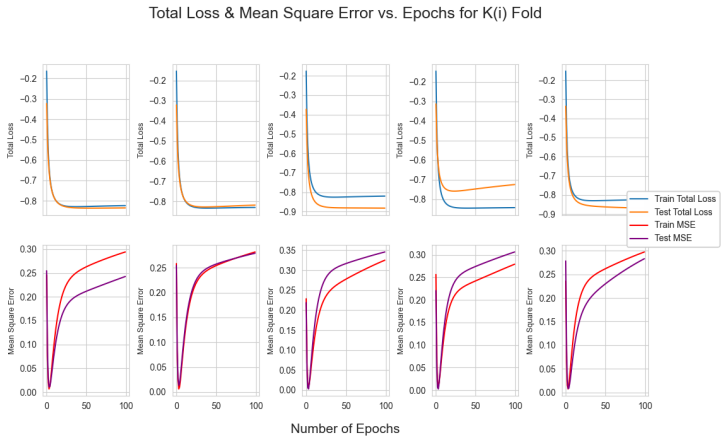


Figure 95: Graph: Computer LN Avg Tuning Performance For Cross Validation, Eta = 0.0125

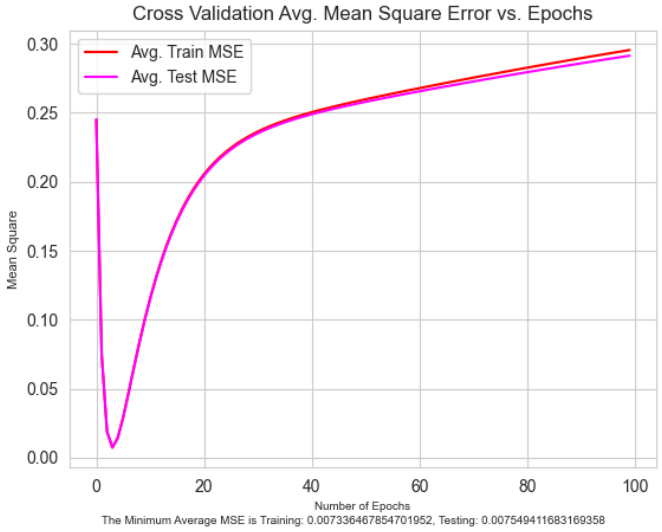


Figure 96: Graph: Computer LN Tuning yPred vs. yActual , Eta = 0.0125

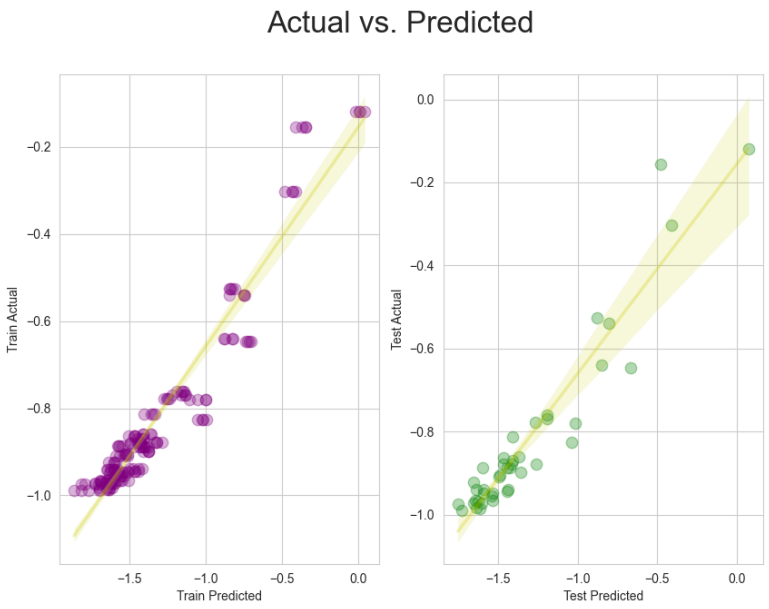


Figure 97: Graph: Computer LN Tuning Performance for each k-Fold, Eta = 0.015

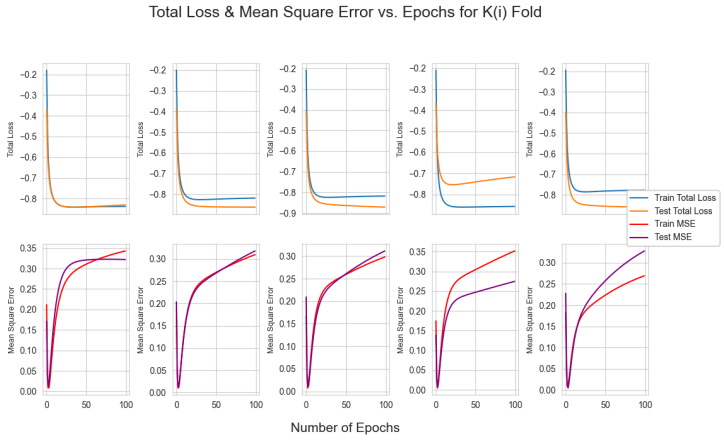


Figure 98: Graph: Computer LN Tuning Avg Performance For Cross Validation, Eta = 0.015

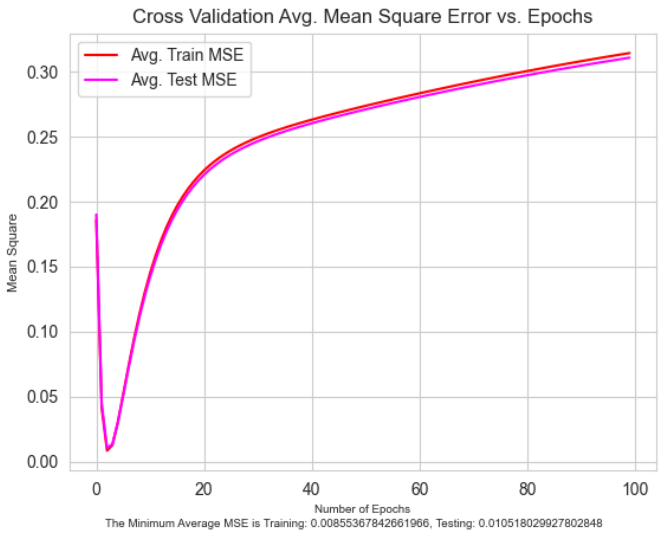


Figure 99: Graph: Computer LN Tuning yPred vs. yActual , Eta = 0.015

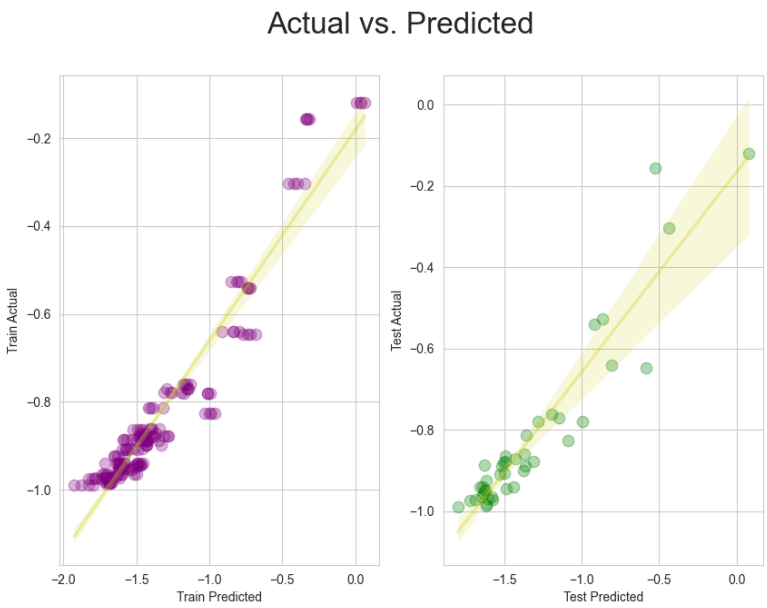


Figure 100: Graph: Computer LN Tuning Performance for each k-Fold, Eta = 0.0175

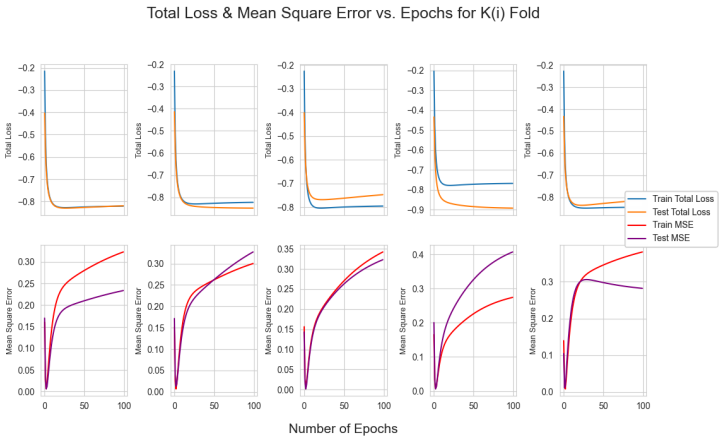


Figure 101: Graph: Computer LN Tuning Avg Performance For Cross Validation, Eta = 0.0175

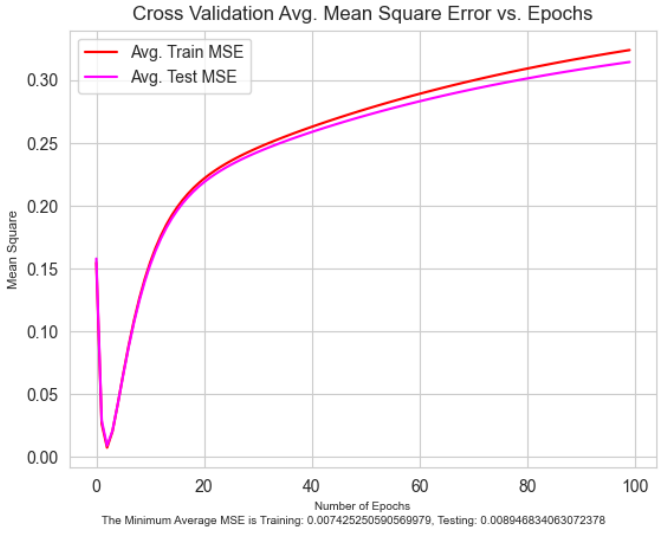


Figure 102: Graph: Computer LN Tuning yPred vs. yActual , Eta = 0.0175

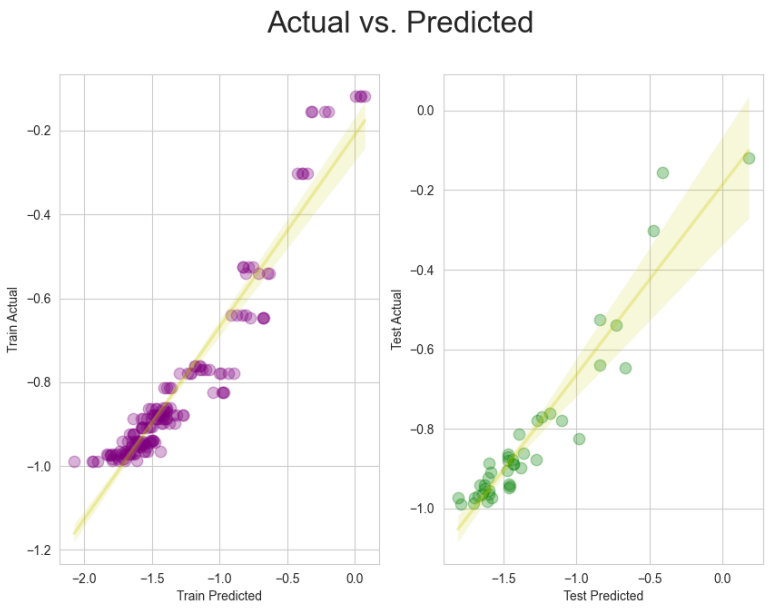


Figure 103: Graph: Computer LN Tuning Performance for each k-Fold, Eta = 0.02

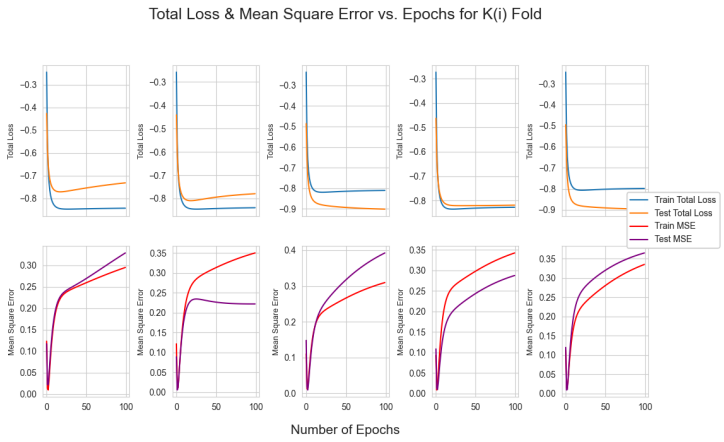


Figure 104: Graph: Computer LN Tuning Avg Performance For Cross Validation, Eta = 0.02

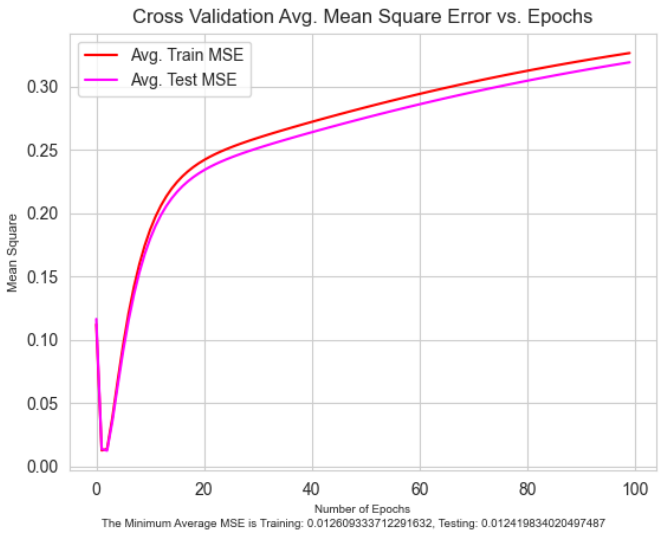
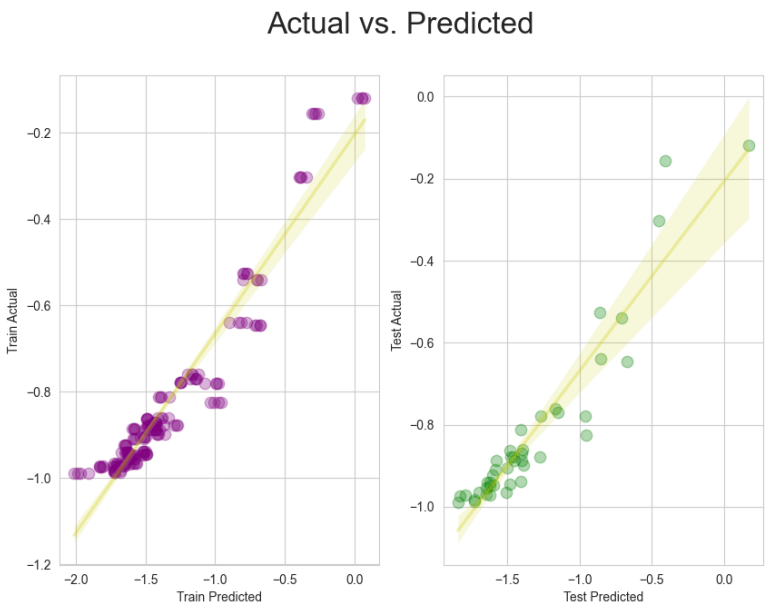


Figure 105: Graph: Computer LN Tuning yPred vs. yActual , Eta = 0.02



3.5.2 Testing Results

ETA = 0.0125 gives us the lowest MSE during tuning at 0.0075494.

Figure 106: Graph: Computer LN 80% Data Performance for each k-Fold, Eta = 0.0125

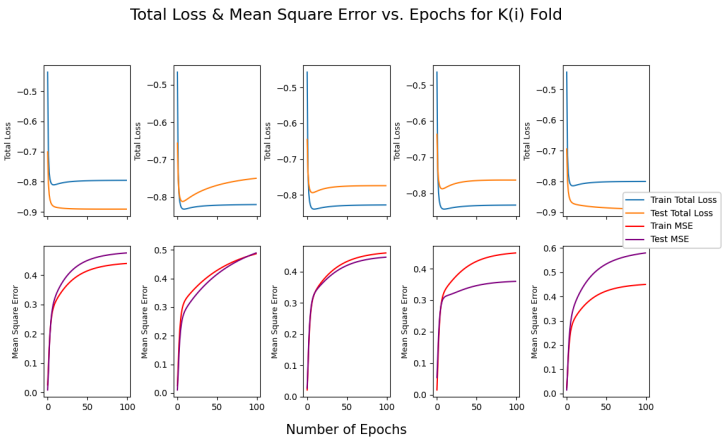


Figure 107: Graph: Computer LN 80% Data Avg Performance For Cross Validation, Eta = 0.0125)

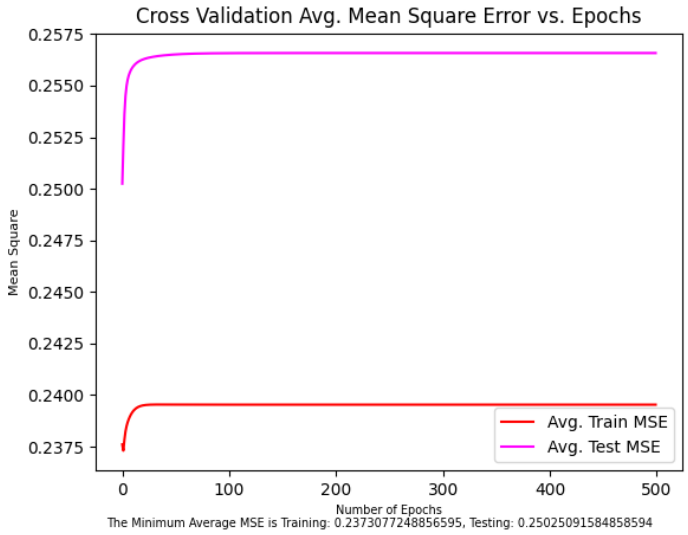
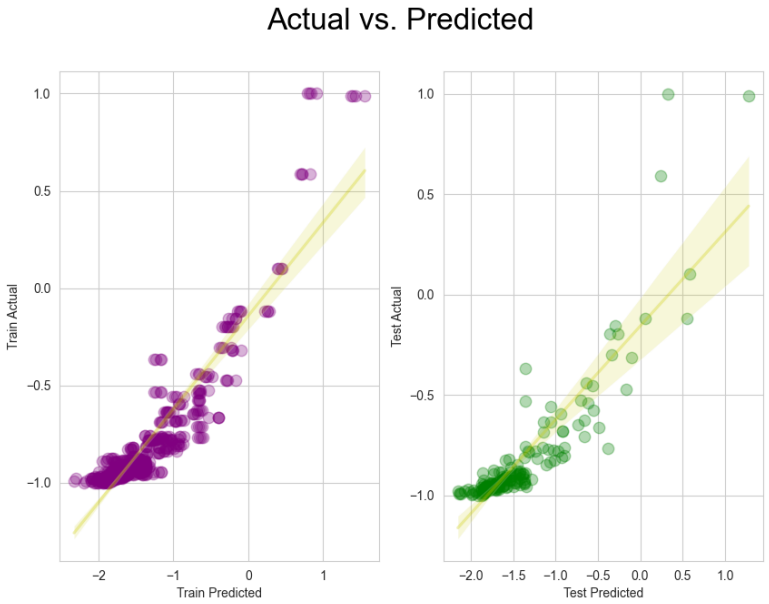


Figure 108: Graph: Computer LN 80% Data yPred vs. yActual , Eta = 0.0125



3.6 Forest Regression via Linear Network

3.6.1 Tuning Results

Figure 109: Graph: Forest LN Tuning Performance for each k-Fold, Eta = 0.01

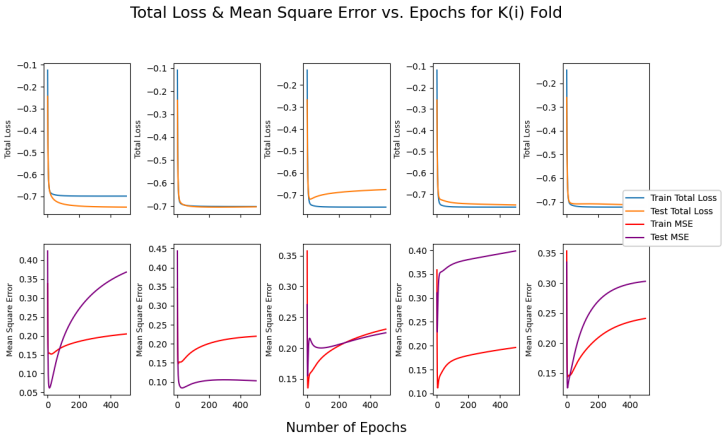


Figure 110: Graph: Forest LN Tuning Avg Performance For Cross Validation, Eta = 0.01

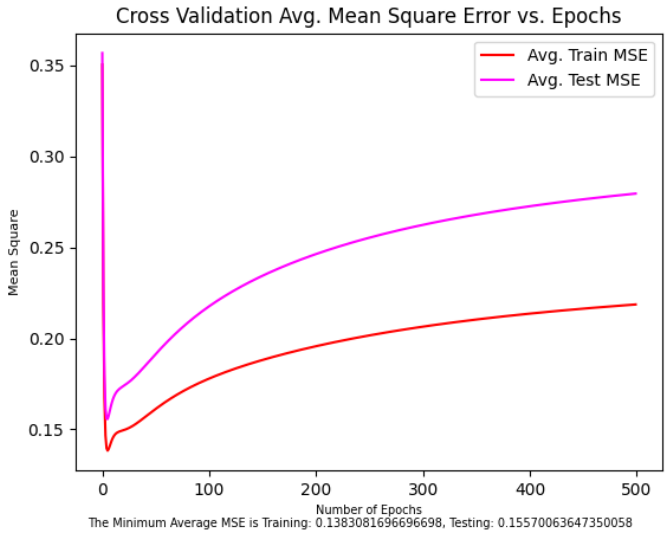


Figure 111: Graph: Forest LN Tuning yPred vs. yActual , Eta = 0.01

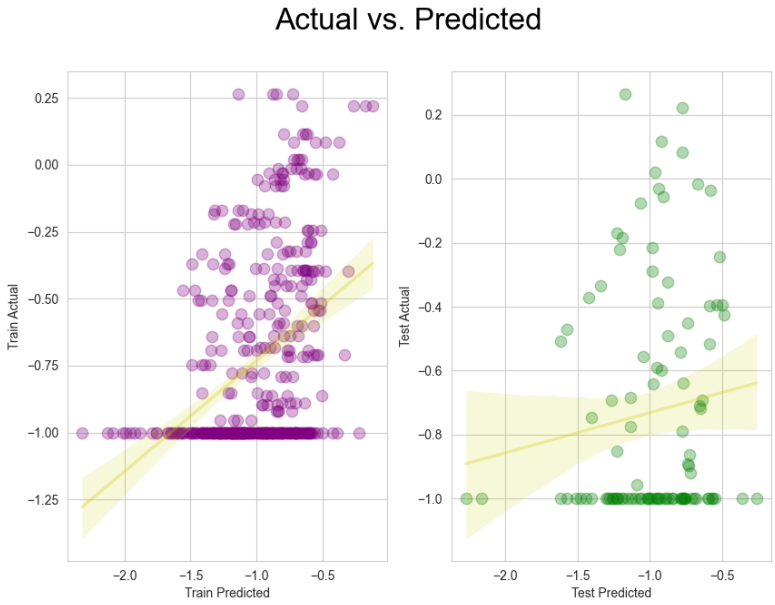


Figure 112: Graph: Forest LN Tuning Performance for each k-Fold, Eta = 0.0125

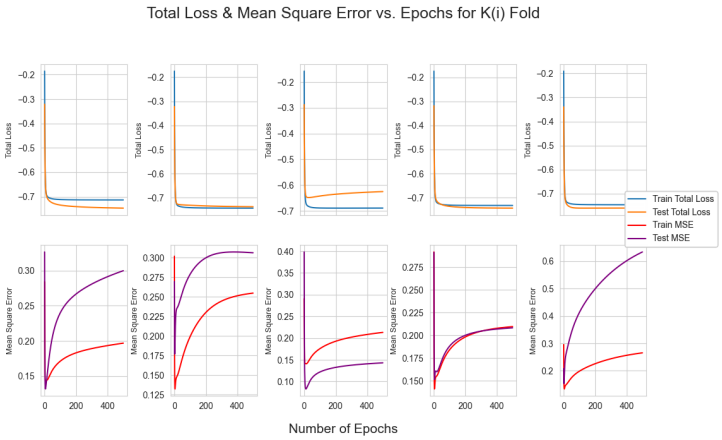


Figure 113: Graph: Forest LN Avg Tuning Performance For Cross Validation, Eta = 0.01225

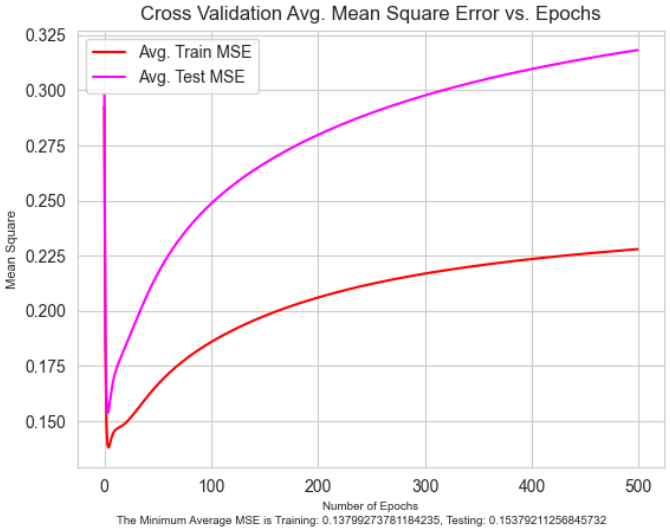


Figure 114: Graph: Forest LN Tuning yPred vs. yActual , Eta = 0.0125

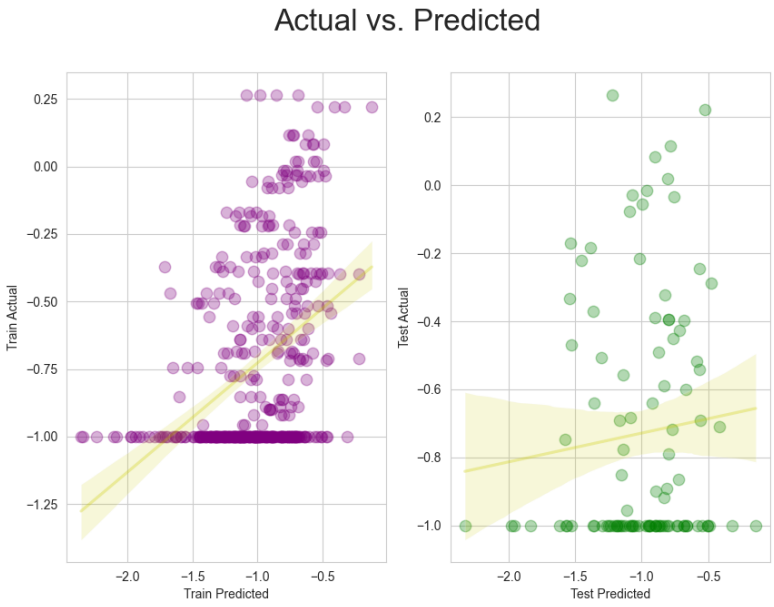


Figure 115: Graph: Forest LN Tuning Performance for each k-Fold, Eta = 0.015

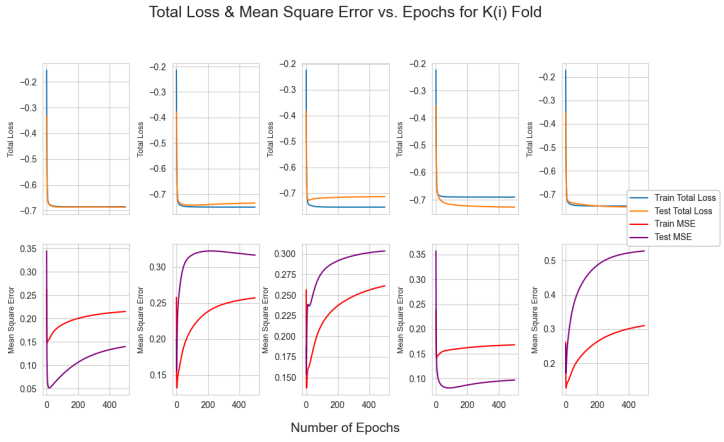


Figure 116: Graph: Forest LN Tuning Avg Performance For Cross Validation, Eta = 0.015

Figure 117: Graph: Forest LN Tuning yPred vs. yActual , Eta = 0.015

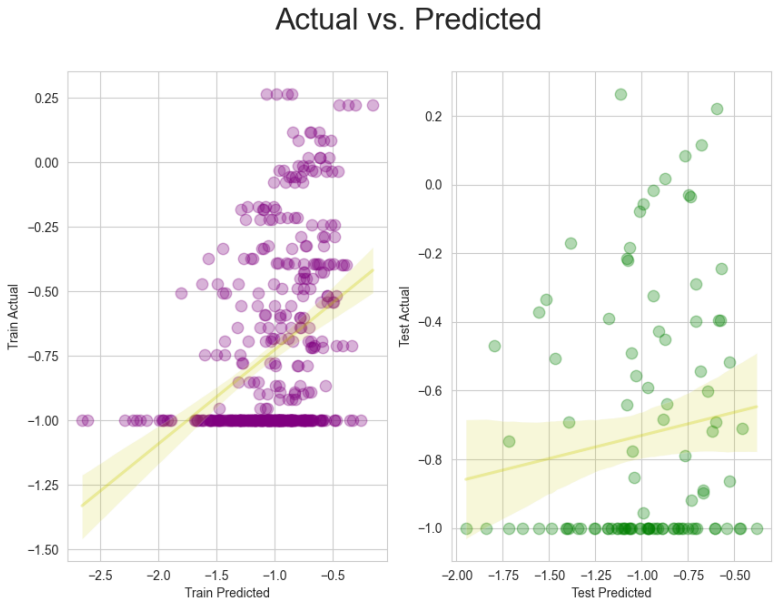


Figure 118: Graph: Forest LN Tuning Performance for each k-Fold, Eta = 0.0175

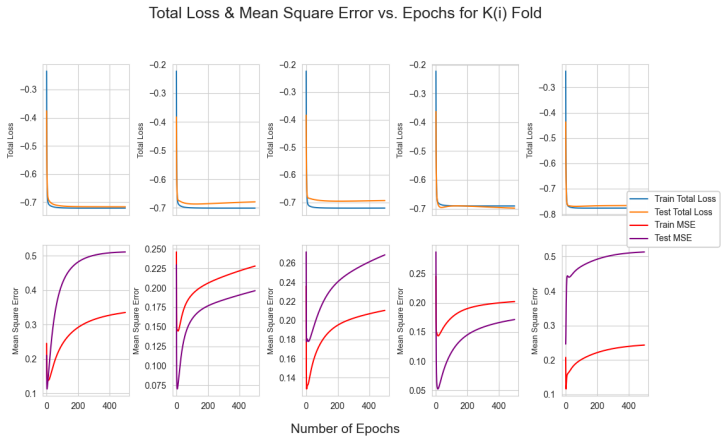


Figure 119: Graph: Forest LN Tuning Avg Performance For Cross Validation, Eta = 0.0175

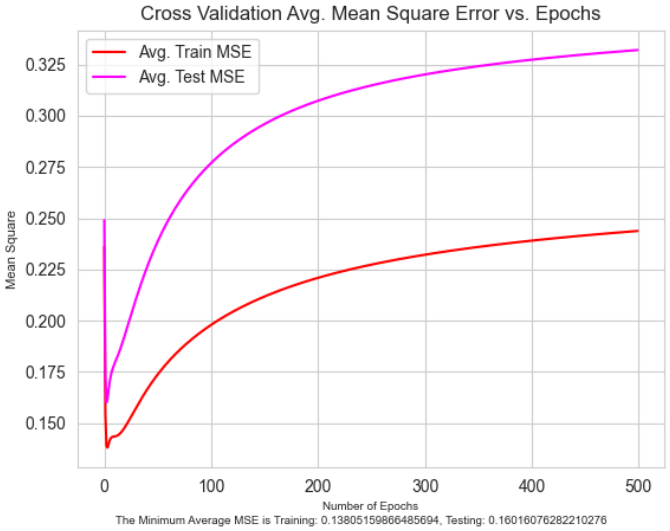


Figure 120: Graph: Forest LN Tuning yPred vs. yActual , Eta = 0.0175

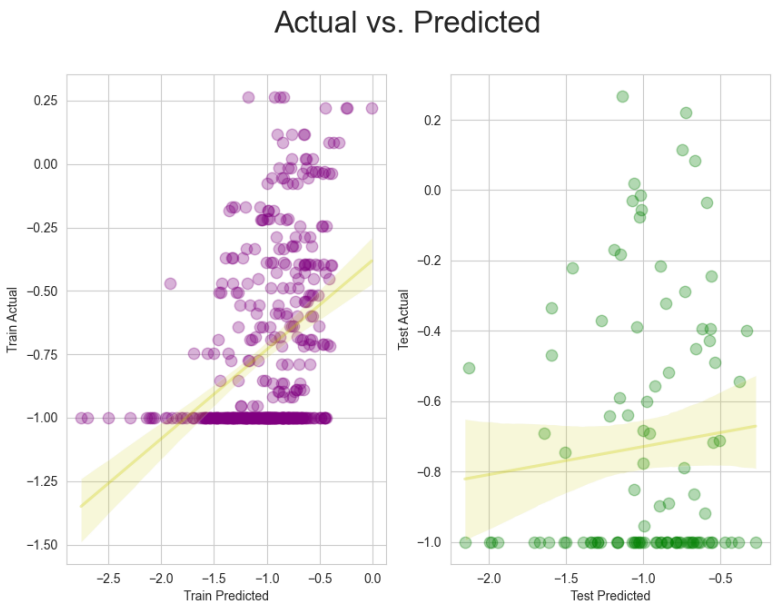


Figure 121: Graph: Forest LN Tuning Performance for each k-Fold, Eta = 0.02

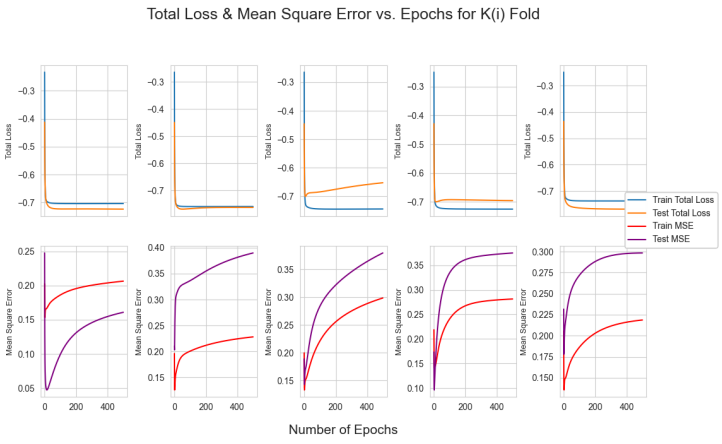


Figure 122: Graph: Forest LN Tuning Avg Performance For Cross Validation, Eta = 0.02

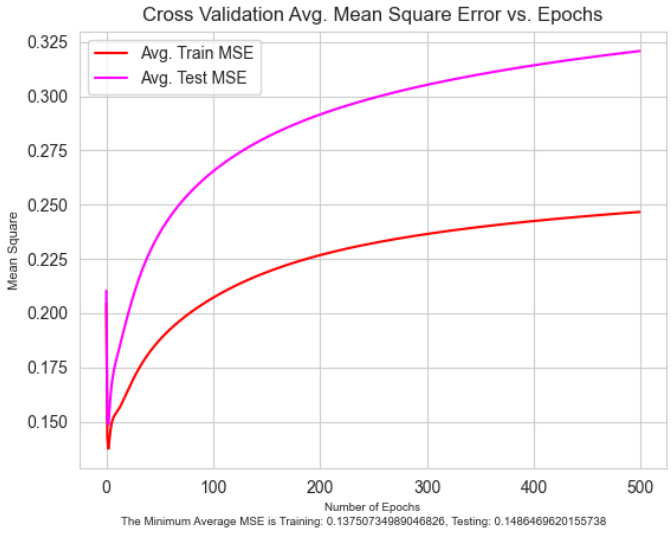
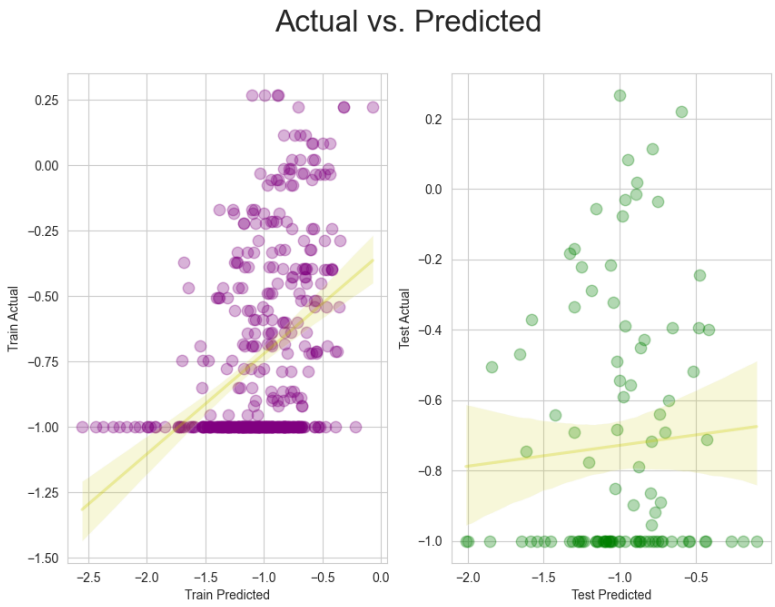


Figure 123: Graph: Forest LN Tuning yPred vs. yActual, Eta = 0.02



3.6.2 Testing Results

ETA = 0.02 gives the lowest MSE on the tuning testing at MSE = 0.1486469.

Figure 124: Graph: Forest LN 80% Data Performance for each k-Fold, Eta = 0.02

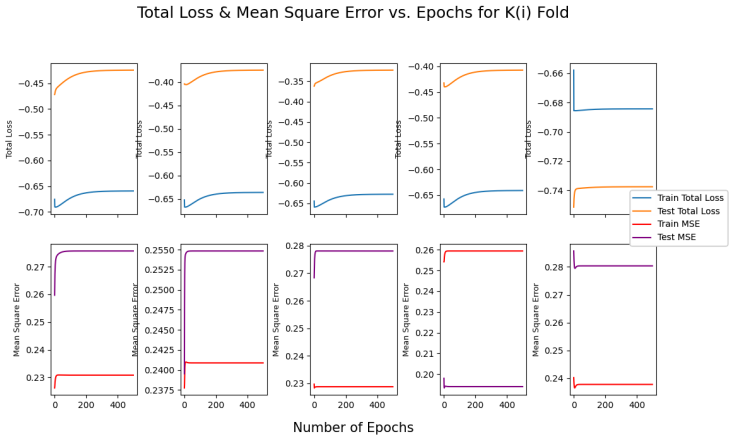


Figure 125: Graph: Forest LN 80% Data Avg Performance For Cross Validation, Eta = 0.02

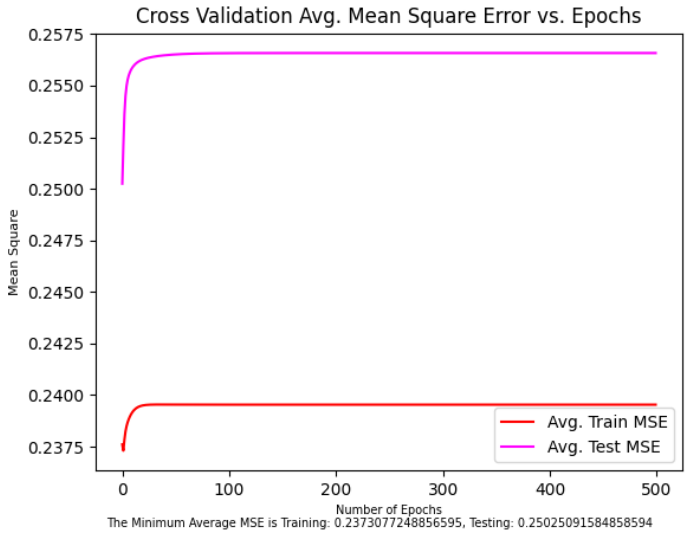
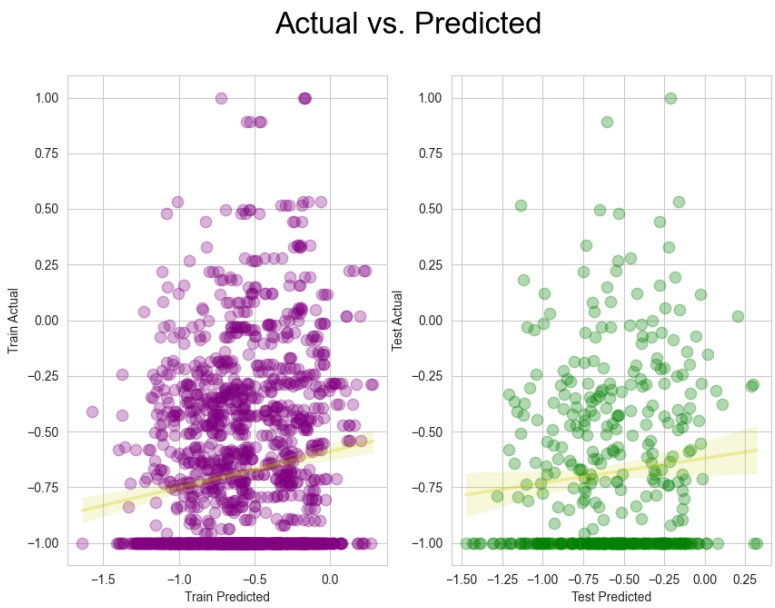


Figure 126: Graph: Forest LN 80% Data yPred vs. yActual, Eta = 0.02



3.7 Car Classification via Deep Neural Network

3.7.1 Tuning Results

Figure 127: Graph: Car DNN Tuning Performance for each k-Fold, Eta = 0.01

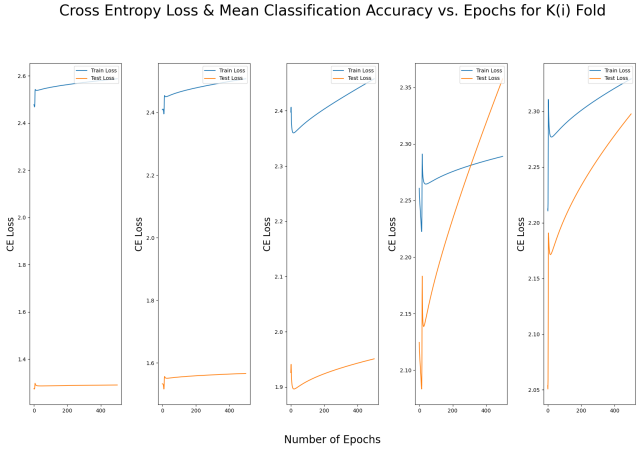


Figure 128: Graph: Car DNN Tuning Train Test Confusion Matrix , Eta = 0.01

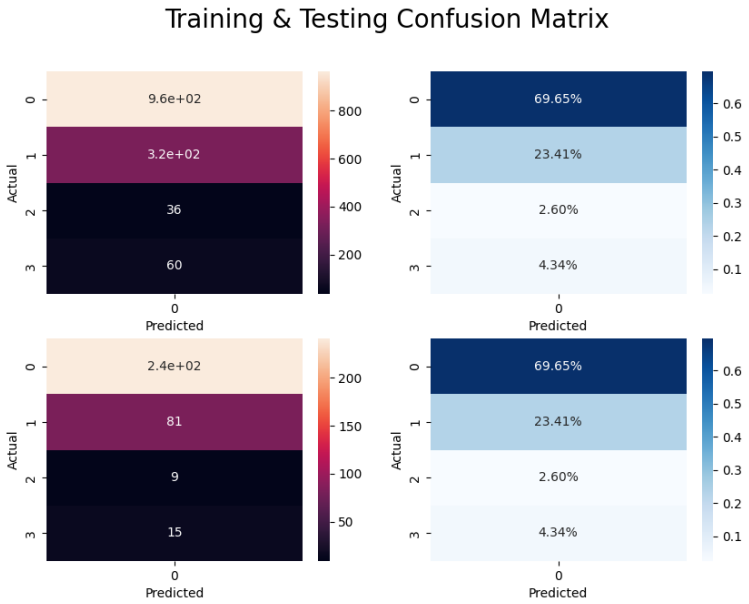


Figure 129: Graph: Car DNN Tuning Performance for each k-Fold, Eta = 0.015

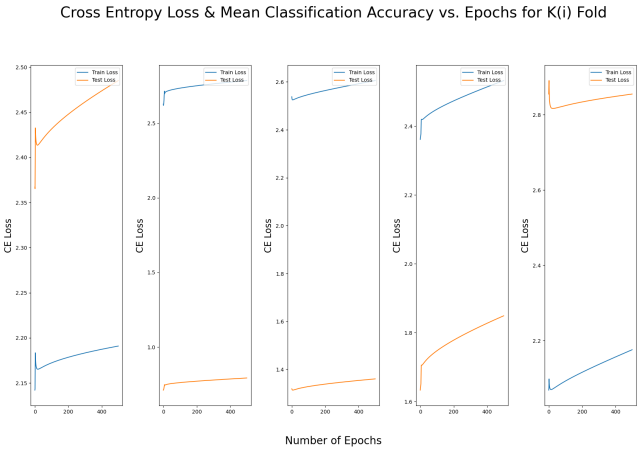


Figure 130: Graph: Car DNN Tuning Train Train Confusion Matrix , Eta = 0.015

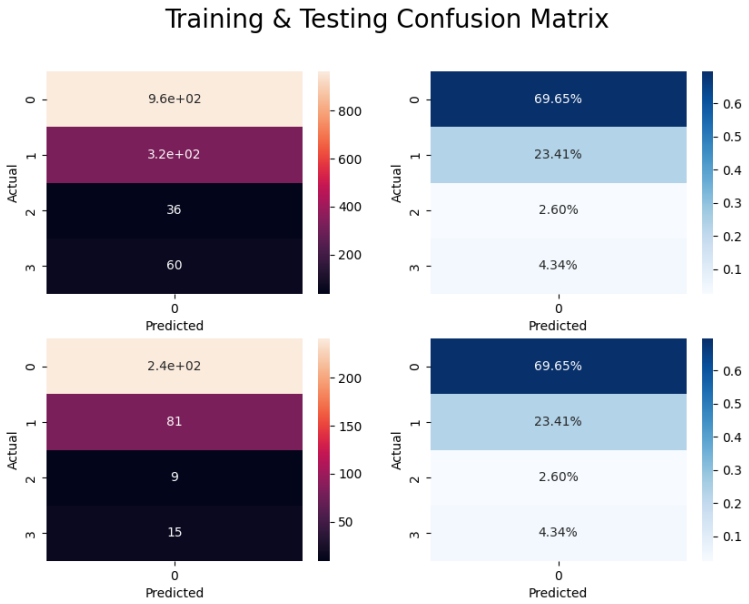


Figure 131: Graph: Car DNN Tuning Performance for each k-Fold, Eta = 0.02

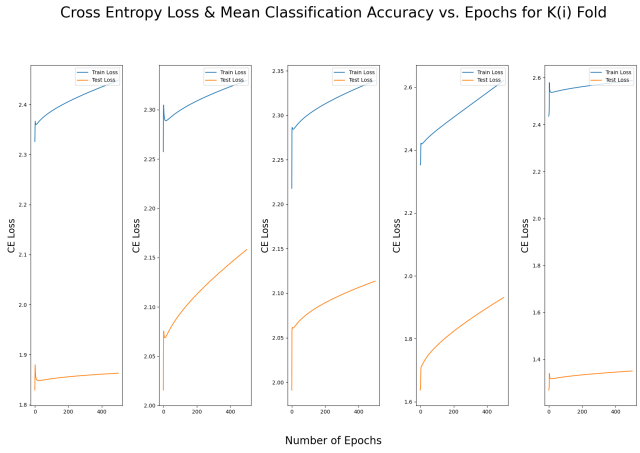


Figure 132: Graph: Car DNN Tuning Train Train Confusion Matrix , Eta = 0.02

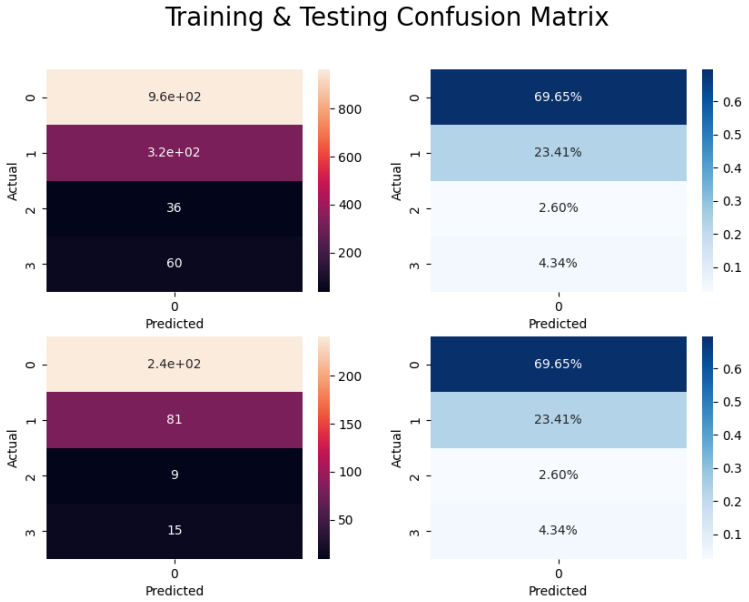


Figure 133: Graph: Car DNN Tuning Performance for each k-Fold, Eta = 0.03

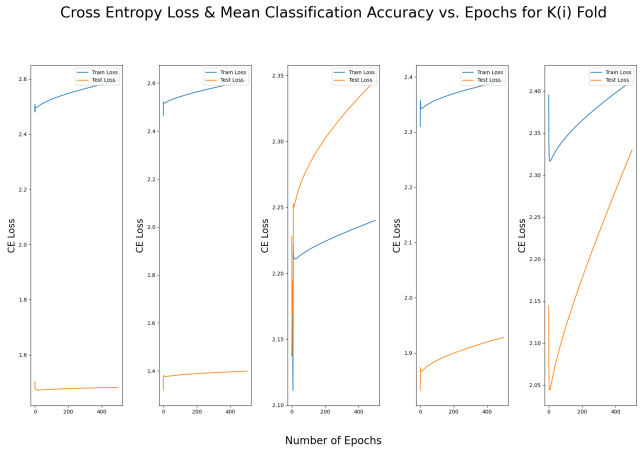


Figure 134: Graph: Car DNN Tuning Train Test Confusion Matrix , Eta = 0.03

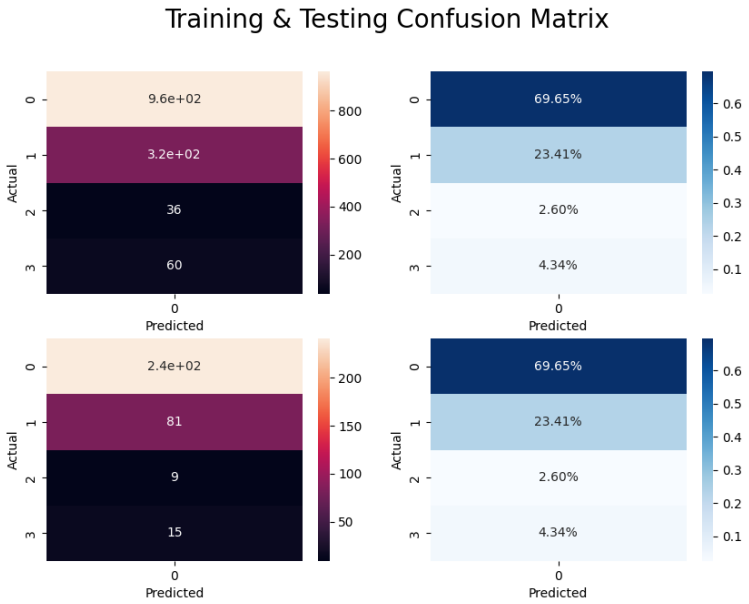


Figure 135: Graph: Car DNN Tuning Performance for each k-Fold, Eta = 0.035

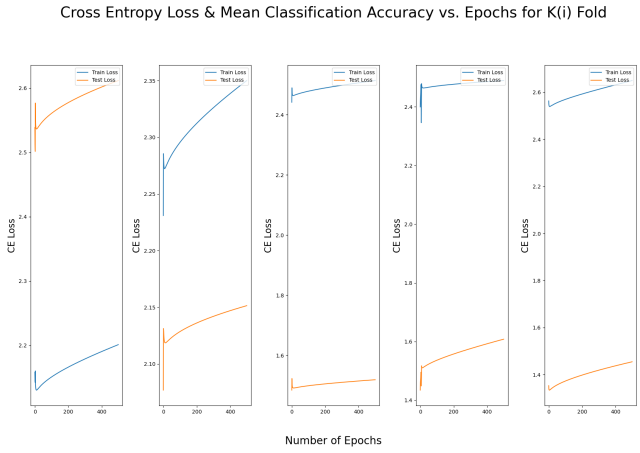
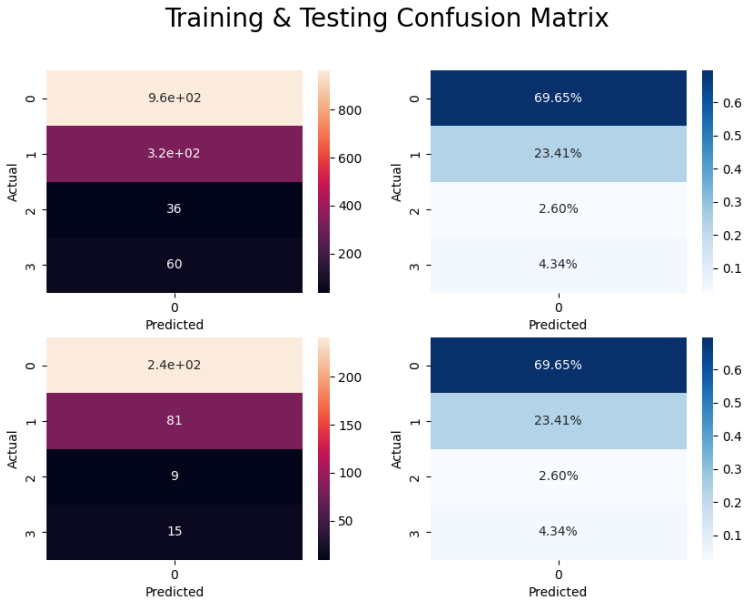


Figure 136: Graph: Car DNN Tuning Train Test Confusion Matrix , Eta = 0.035



3.7.2 TESTING RESULTS

ETA = 0.03 was used for the testing as the accuracy was the same across the etas.

Figure 137: Graph: Car DNN 80% Data Performance for each k-Fold, Eta = 0.03

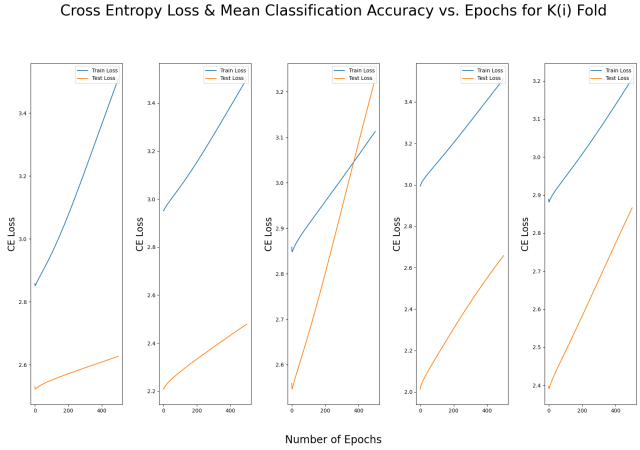


Figure 138: Graph: Car DNN 80% Training Confusion Matrix , Eta = 0.03

3.8 Voting Classification via Deep Neural Network

3.8.1 Tuning Results

Figure 139: Graph: Vote DNN Tuning Performance for each k-Fold, Eta = 0.01

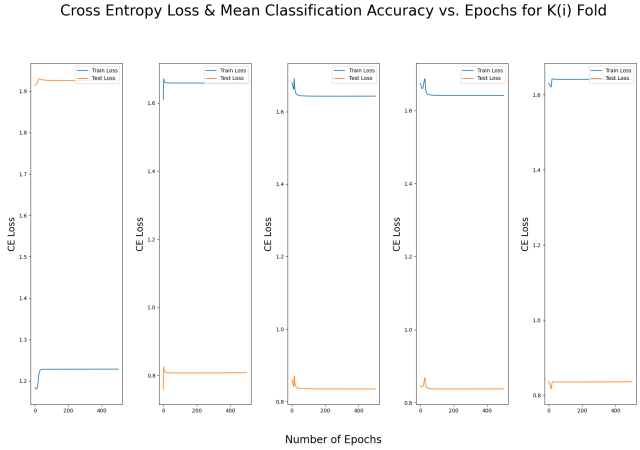


Figure 140: Graph: Vote DNN Tuning Train Test Confusion Matrix , Eta = 0.01

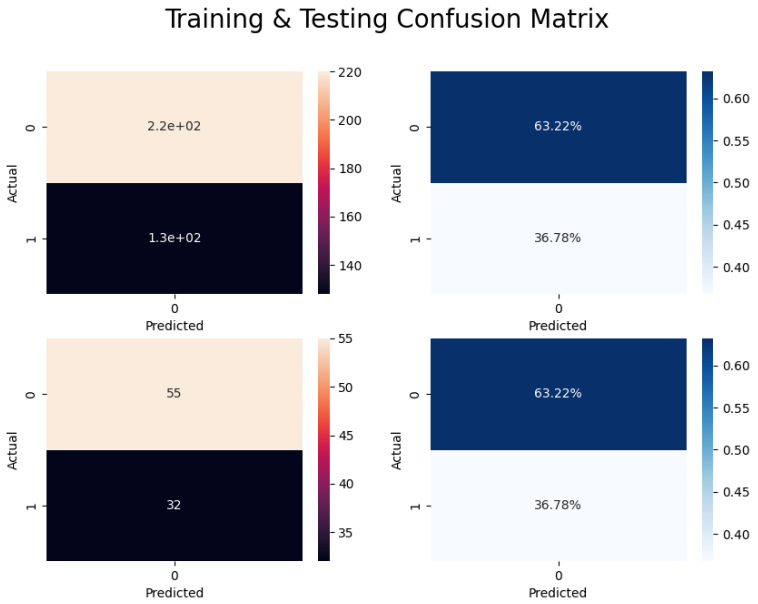


Figure 141: Graph: Vote DNN Tuning Performance for each k-Fold, Eta = 0.015

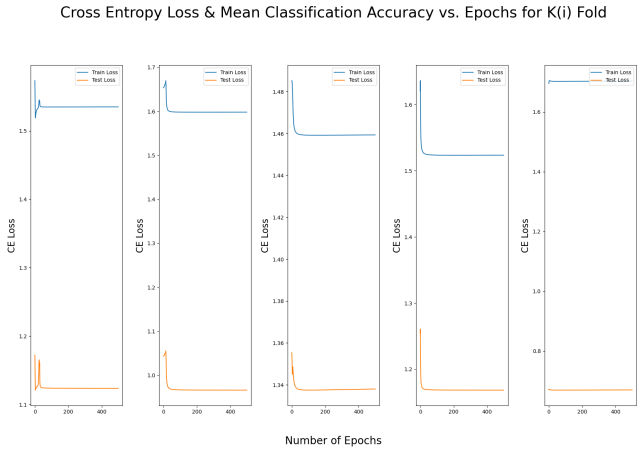


Figure 142: Graph: Vote DNN Tuning Train Train Confusion Matrix , Eta = 0.015

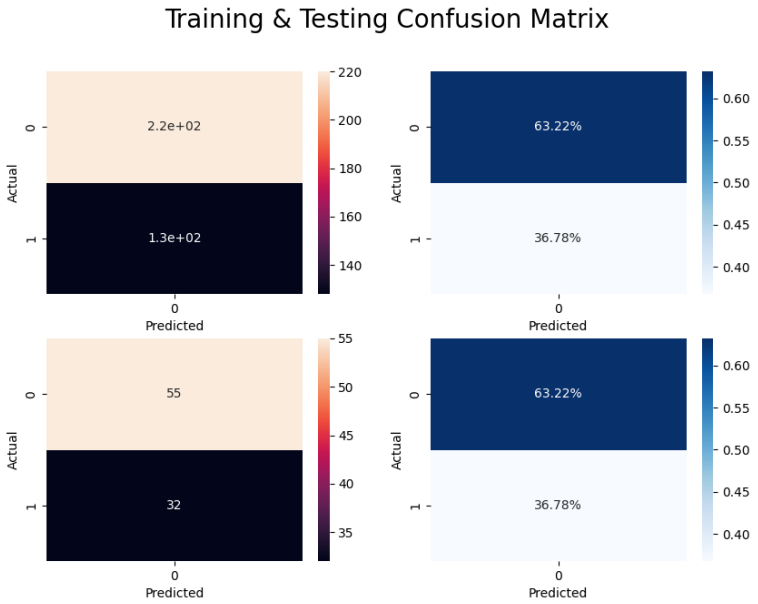


Figure 143: Graph: Vote DNN Tuning Performance for each k-Fold, Eta = 0.02

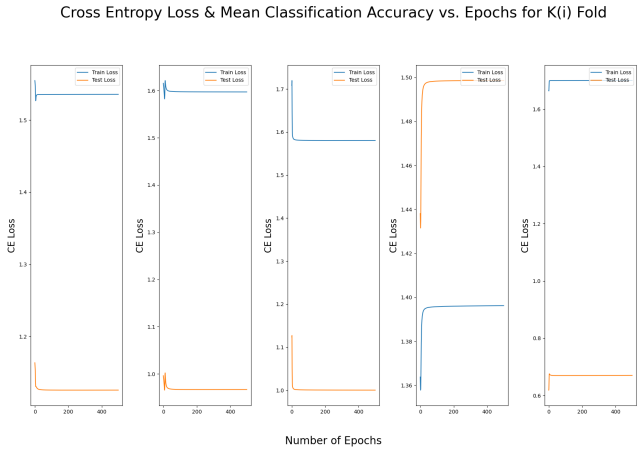


Figure 144: Graph: Vote DNN Tuning Train Train Confusion Matrix , Eta = 0.02

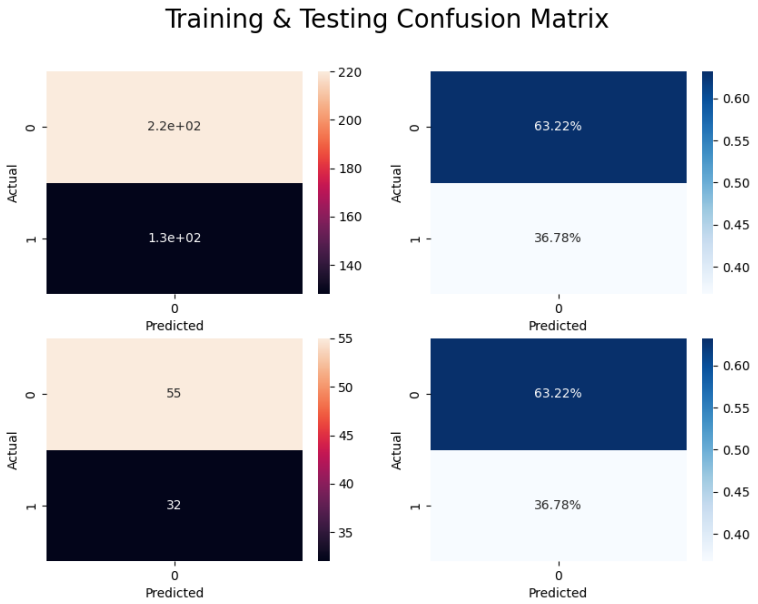


Figure 145: Graph: Vote DNN Tuning Performance for each k-Fold, Eta = 0.03

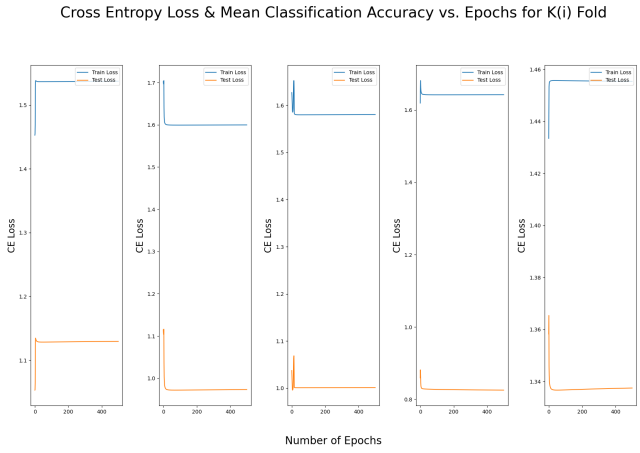


Figure 146: Graph: Vote DNN Tuning Train Test Confusion Matrix , Eta = 0.03

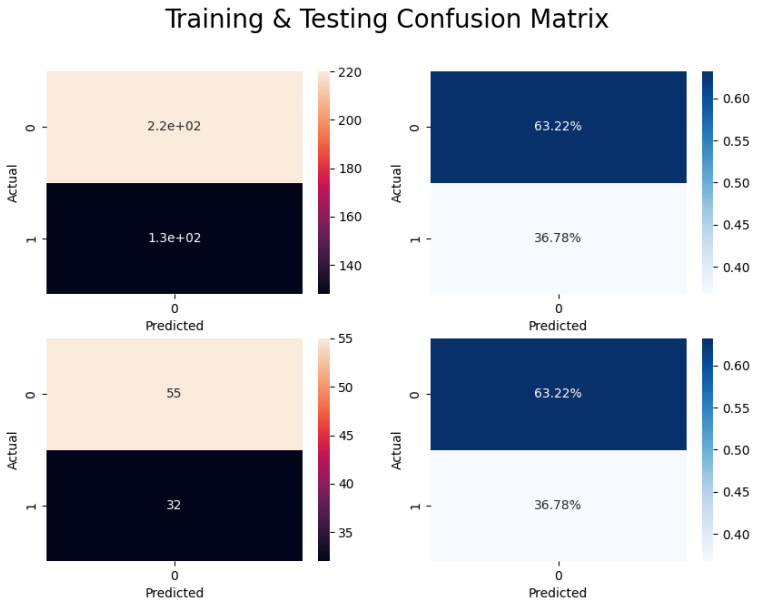


Figure 147: Graph: Vote DNN Tuning Performance for each k-Fold, Eta = 0.035

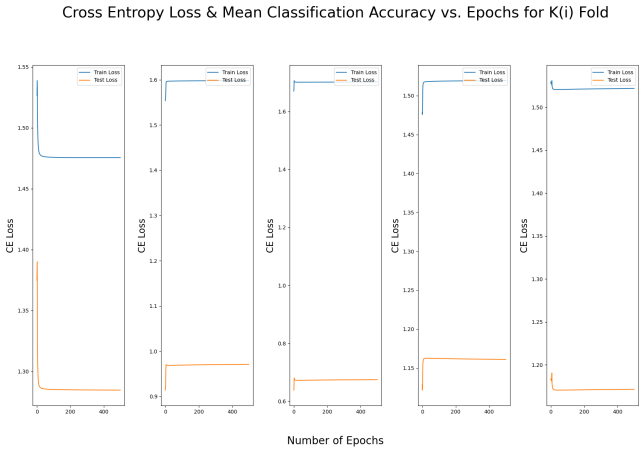
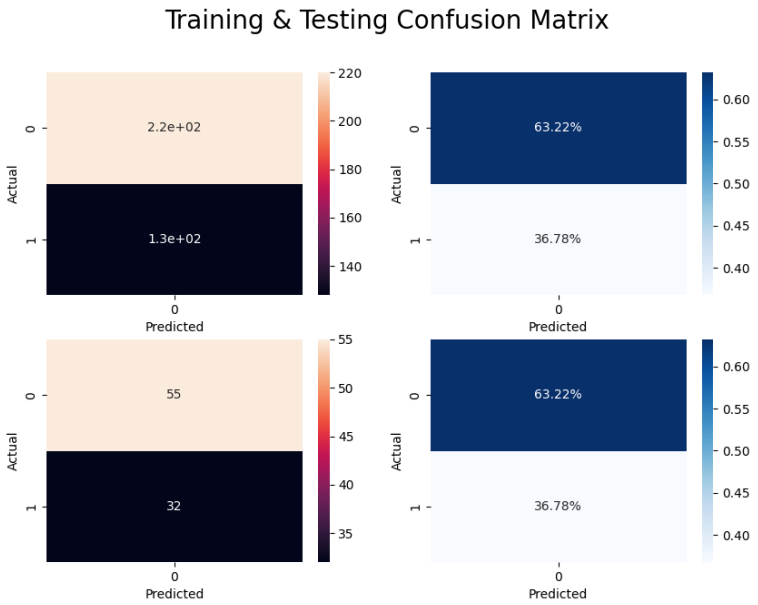


Figure 148: Graph: Vote DNN Tuning Train Test Confusion Matrix , Eta = 0.035



3.8.2 Testing Results

Eta = 0.02 was used as the eta value for the testing on the remaining data.

Figure 149: Graph: Vote DNN 80% Data Performance for each k-Fold, Eta = 0.02

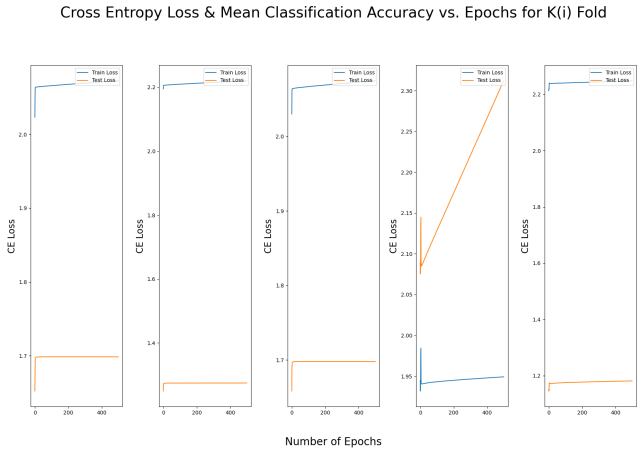
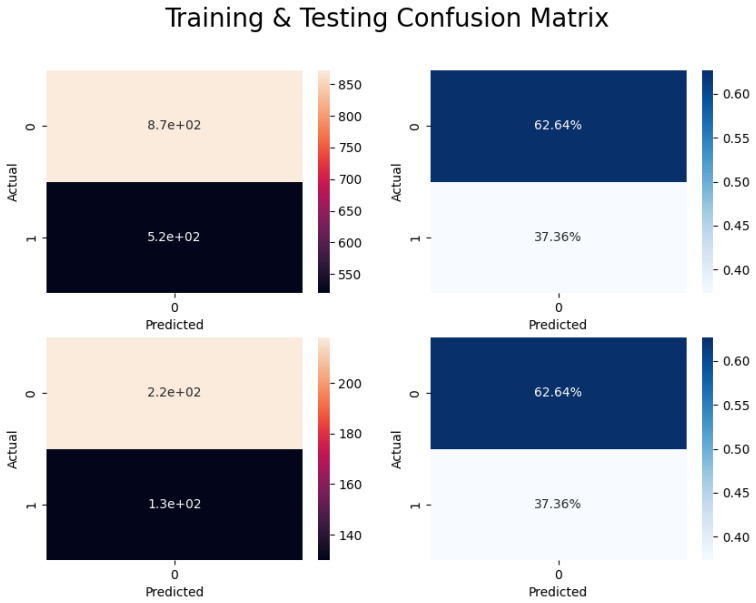


Figure 150: Graph: Vote DNN 80% Training Confusion Matrix , Eta = 0.02



3.9 Breast Cancer Classification via Deep Neural Network

3.9.1 Tuning Results

Figure 151: Graph: Breast DNN Tuning Performance for each k-Fold, Eta = 0.01

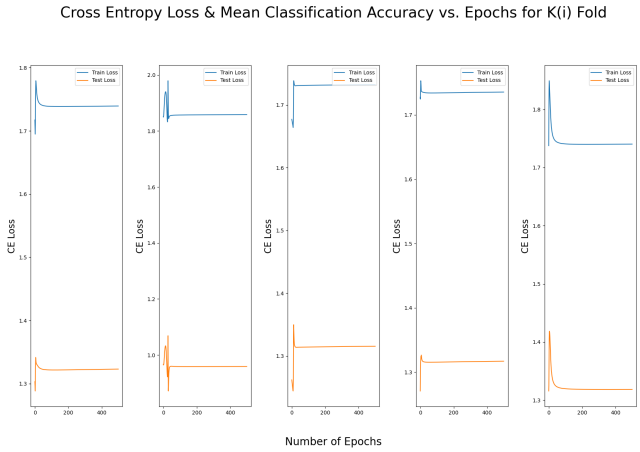


Figure 152: Graph: Breast DNN Tuning Train Test Confusion Matrix , Eta = 0.01

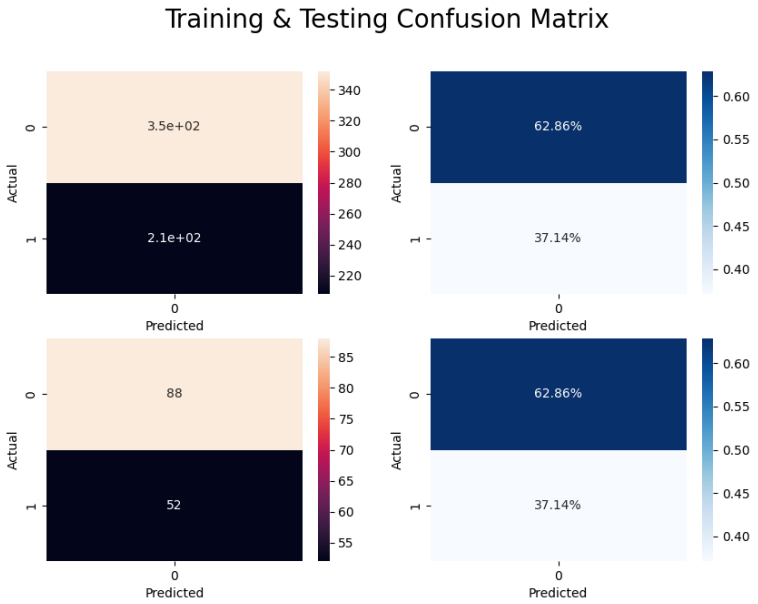


Figure 153: Graph: Breast DNN Tuning Performance for each k-Fold, Eta = 0.015

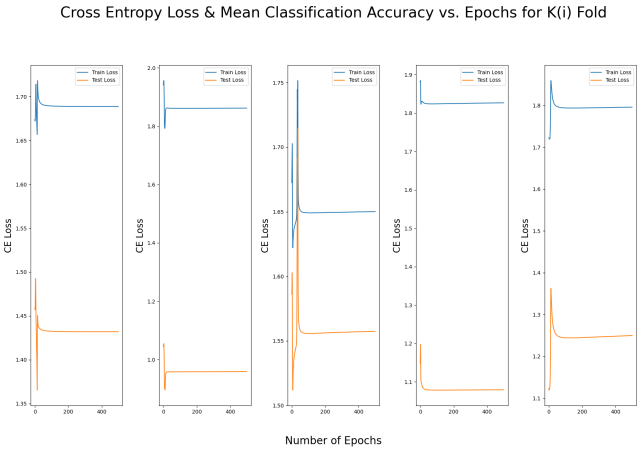


Figure 154: Graph: Breast DNN Tuning Train Train Confusion Matrix , Eta = 0.015

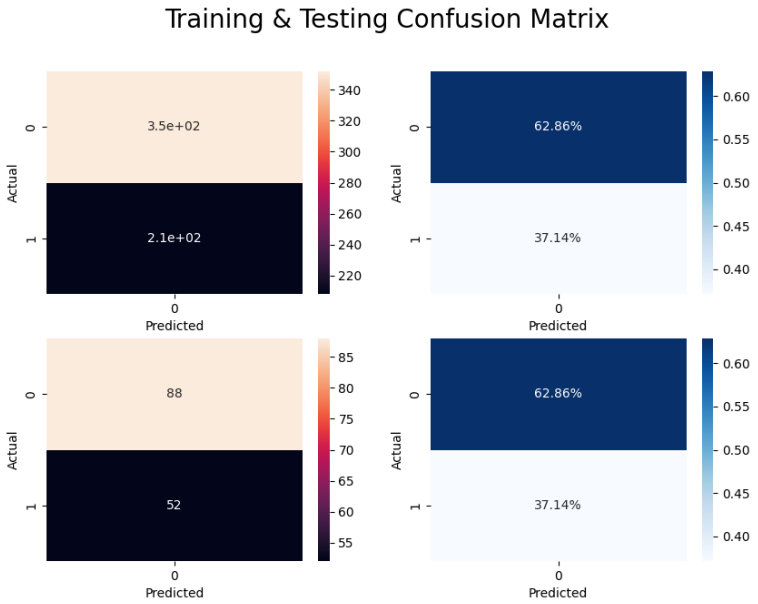


Figure 155: Graph: Breast DNN Tuning Performance for each k-Fold, Eta = 0.02

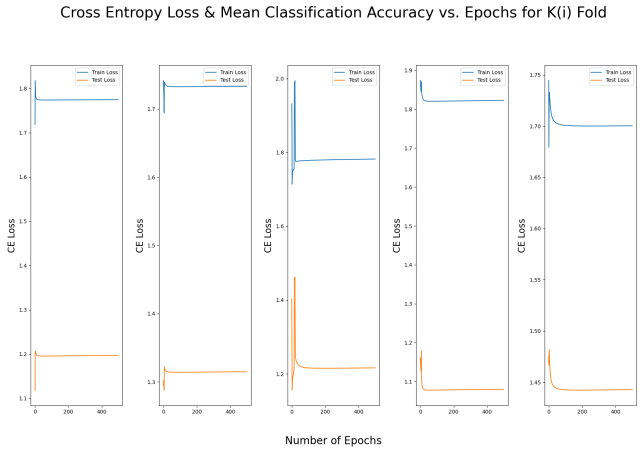


Figure 156: Graph: Breast DNN Tuning Train Train Confusion Matrix , Eta = 0.02

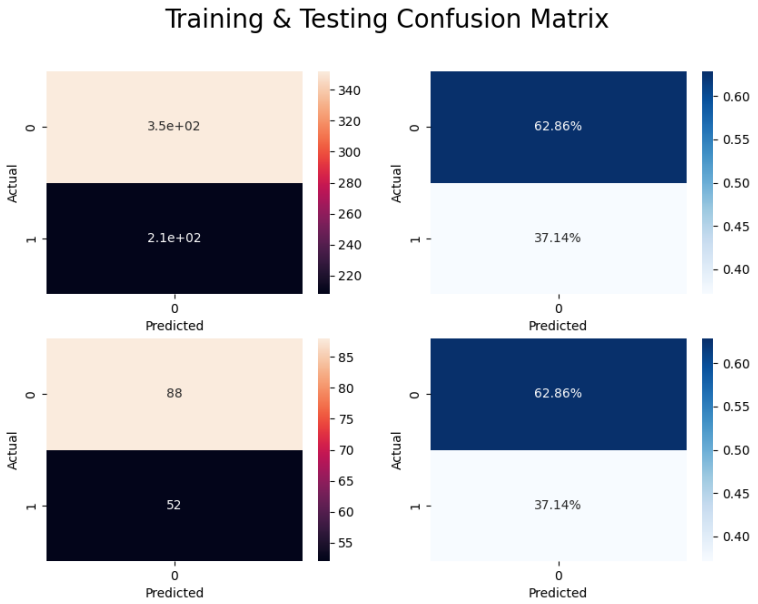


Figure 157: Graph: Breast DNN Tuning Performance for each k-Fold, Eta = 0.03

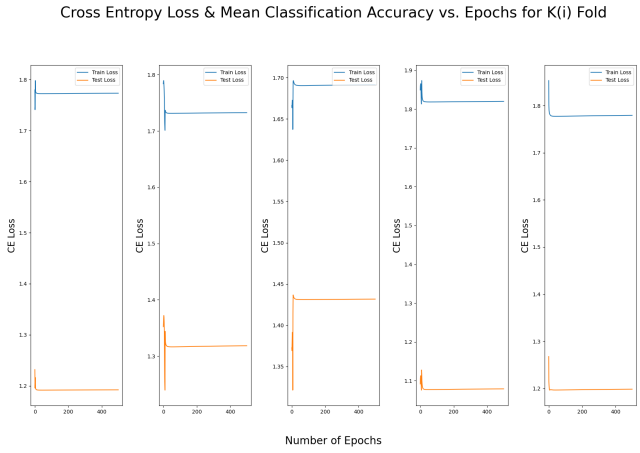


Figure 158: Graph: Breast DNN Tuning Train Test Confusion Matrix , Eta = 0.03

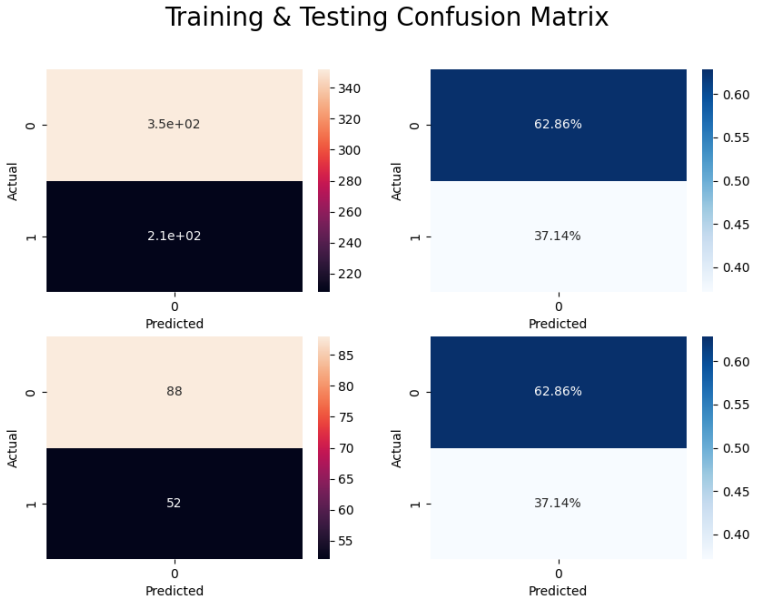


Figure 159: Graph: Breast DNN Tuning Performance for each k-Fold, Eta = 0.035

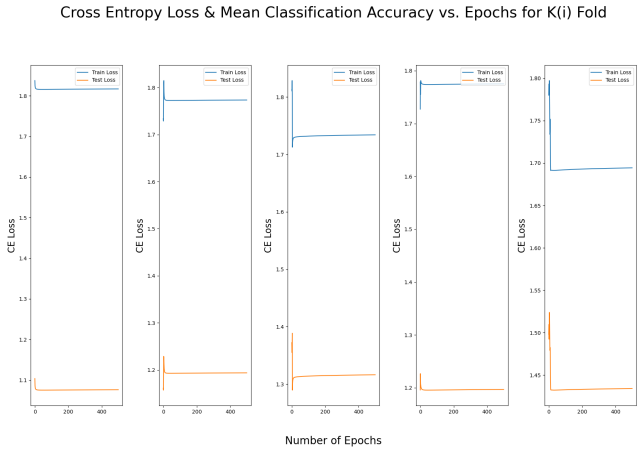
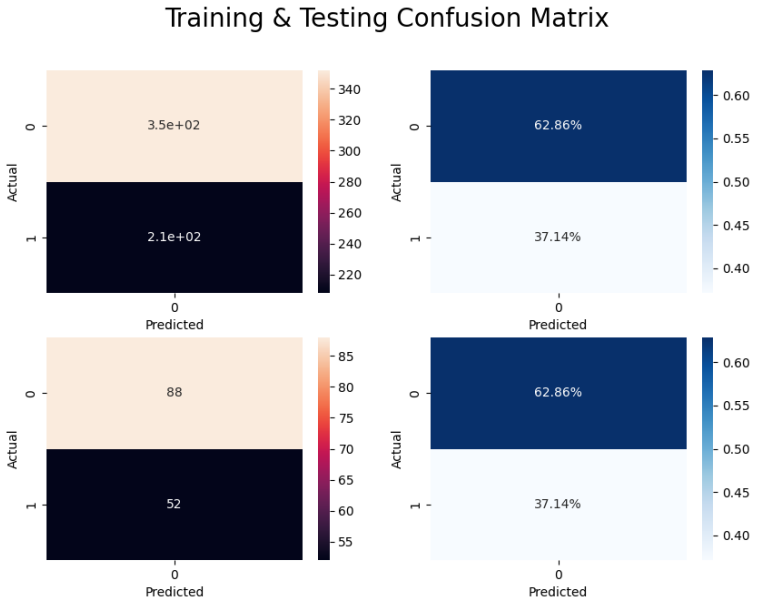


Figure 160: Graph: Breast DNN Tuning Train Test Confusion Matrix , Eta = 0.035



3.9.2 TESTING RESULTS

ETA = 0.015 was used as the eta value for testing it on the remaining data.

Figure 161: Graph: Breast DNN 80% Data Performance for each k-Fold, Eta = 0.015

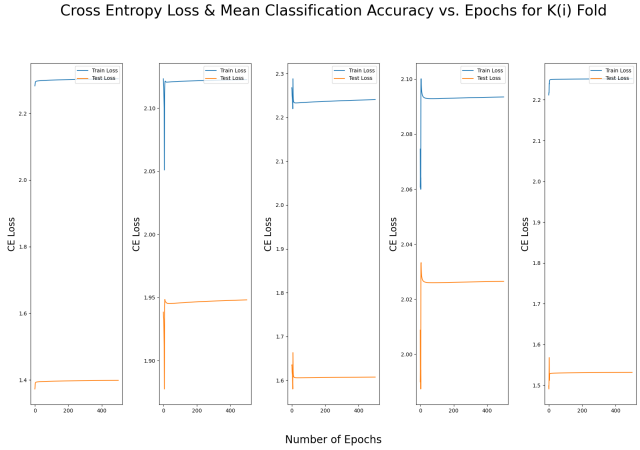
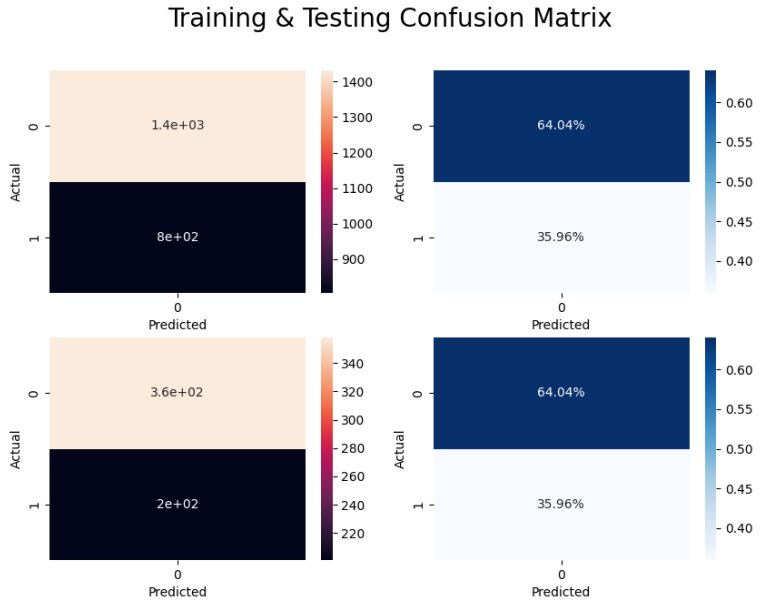


Figure 162: Graph: Breast DNN 80% Training Confusion Matrix , Eta = 0.015



3.10 Abalone Regression via Deep Neural Network

3.10.1 Tuning Results

Figure 163: Graph: Abalone DNN Tuning Avg Performance For Cross Validation, Eta = 0.01

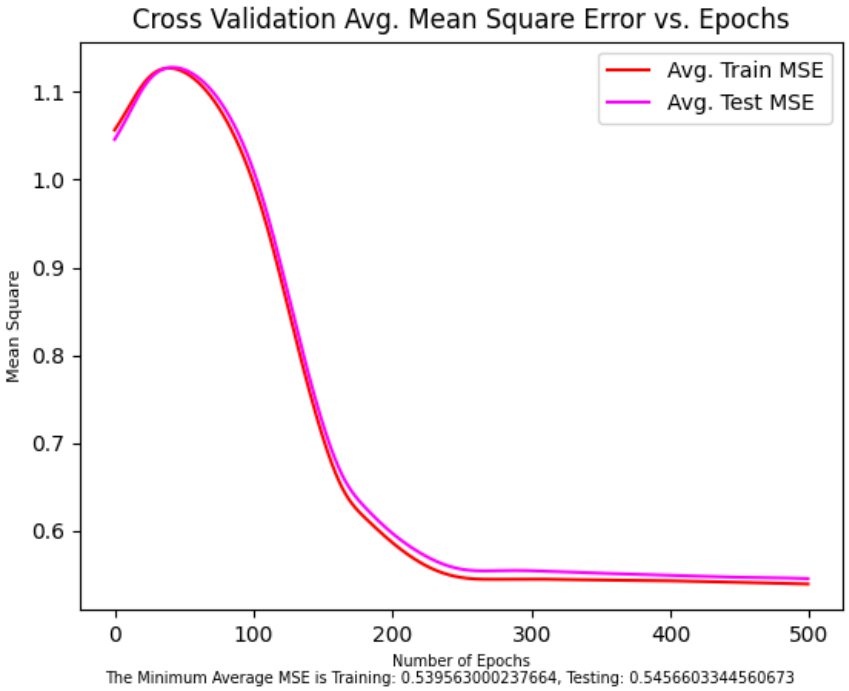


Figure 164: Graph: Abalone DNN Tuning yPred vs. yActual, Eta = 0.01

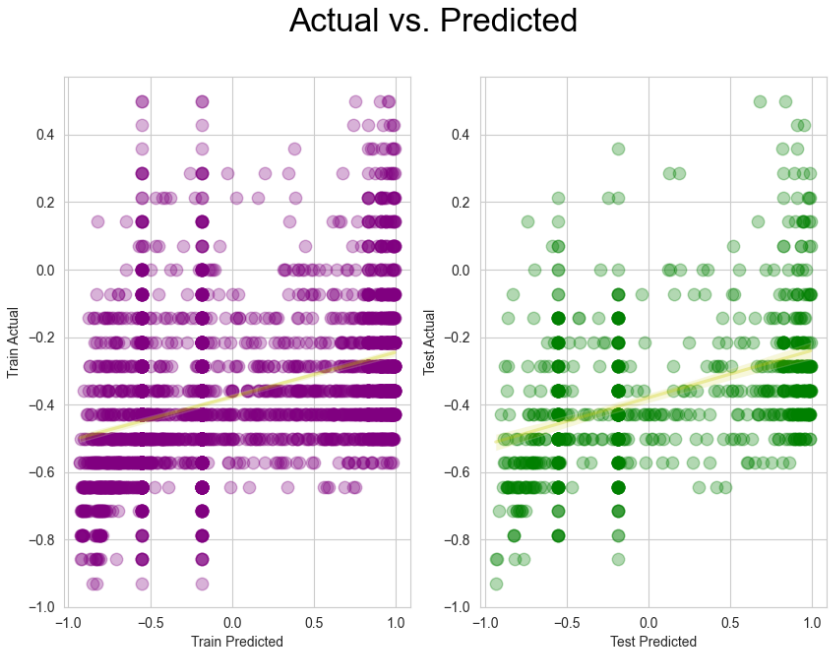


Figure 165: Graph: Abalone DNN Avg Tuning Performance For Cross Validation, Eta = 0.015

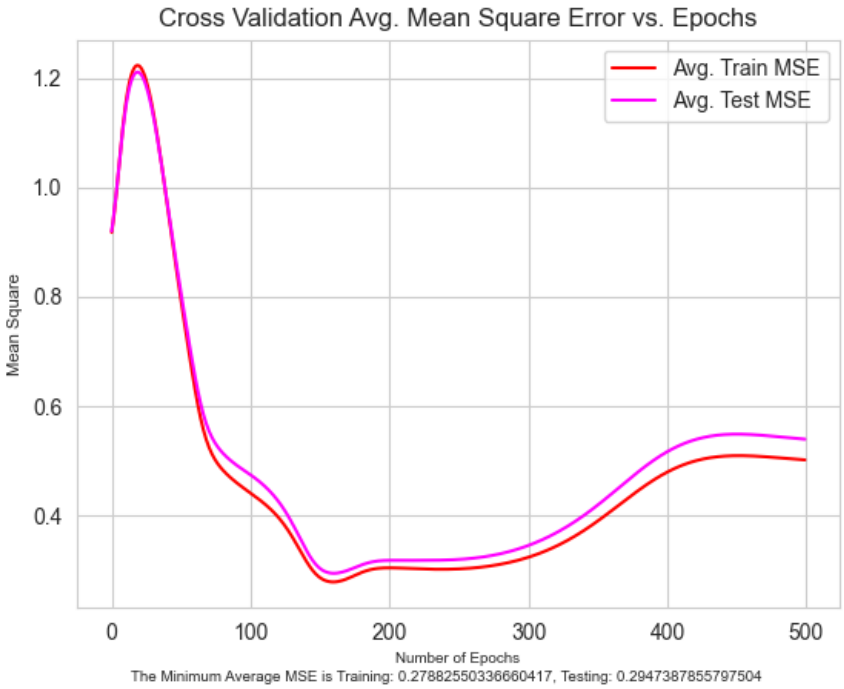


Figure 166: Graph: Abalone DNN Tuning yPred vs. yActual, Eta = 0.015

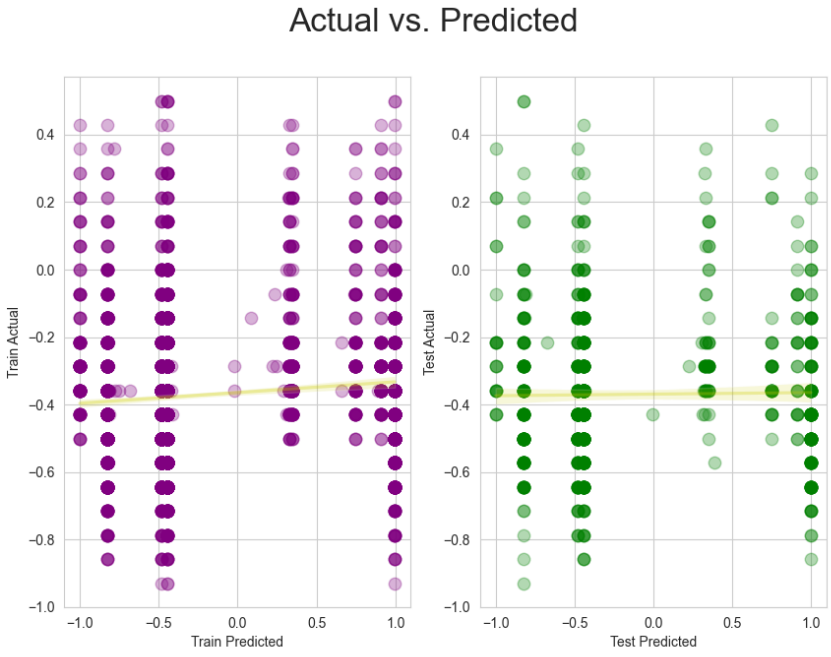


Figure 167: Graph: Abalone DNN Tuning Avg Performance For Cross Validation, Eta = 0.02

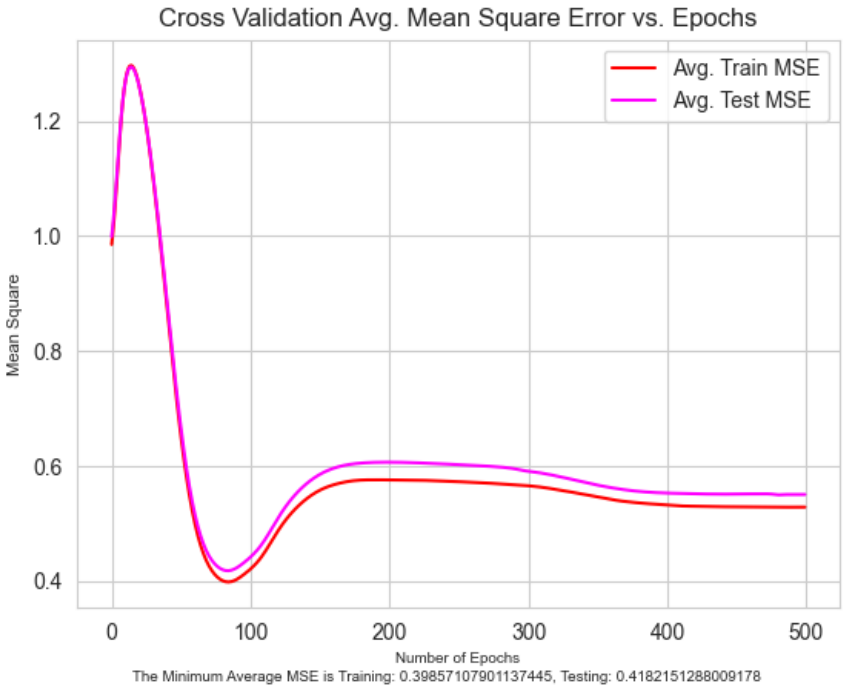


Figure 168: Graph: Abalone DNN Tuning yPred vs. yActual, Eta = 0.02

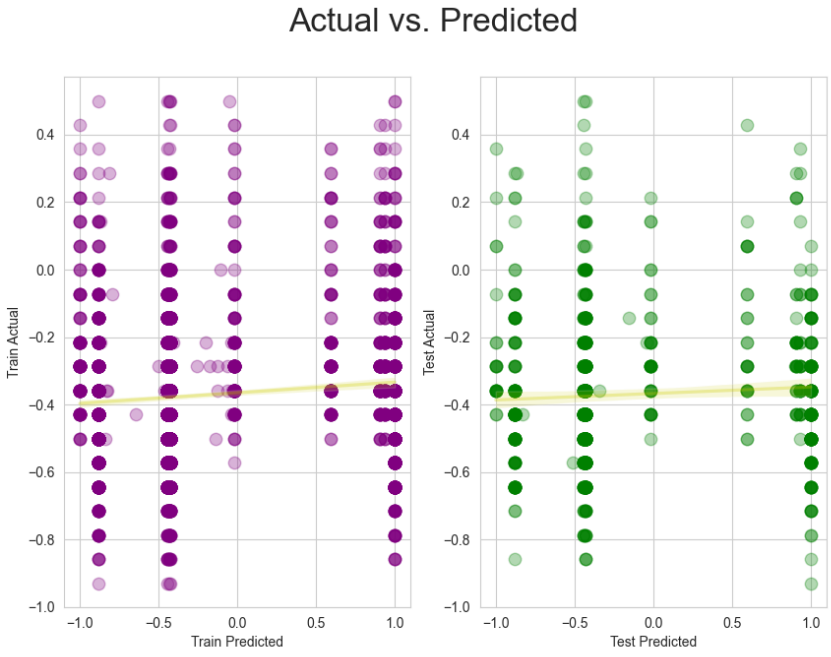


Figure 169: Graph: Abalone DNN Tuning Avg Performance For Cross Validation, Eta = 0.03

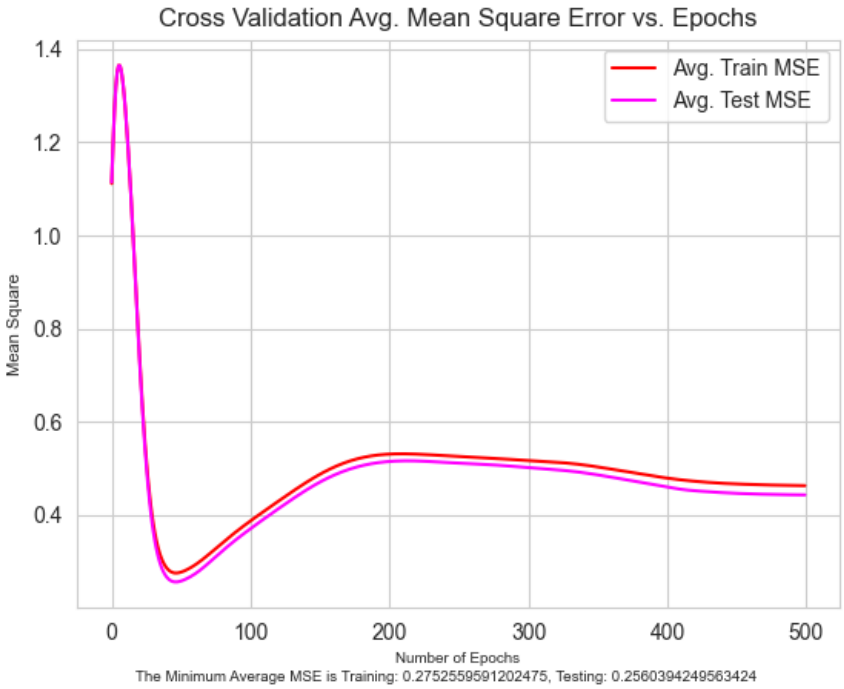


Figure 170: Graph: Abalone DNN Tuning yPred vs. yActual, Eta = 0.03

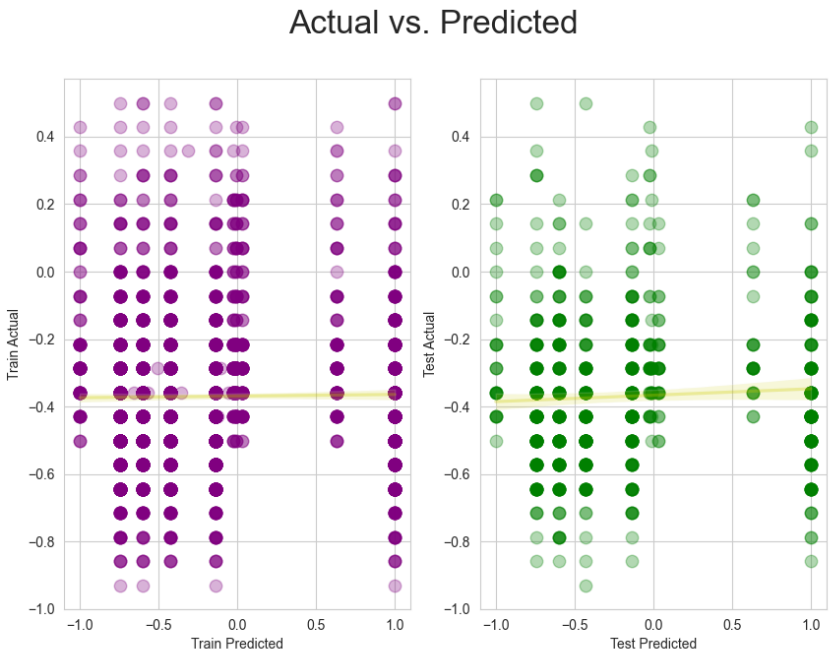


Figure 171: Graph: Abalone DNN Tuning Avg Performance For Cross Validation, Eta = 0.035

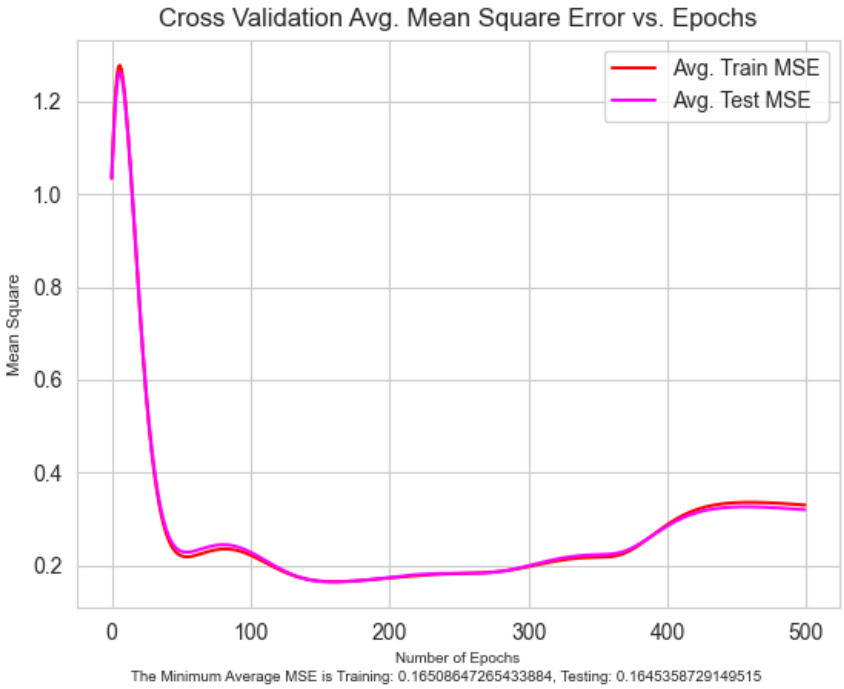
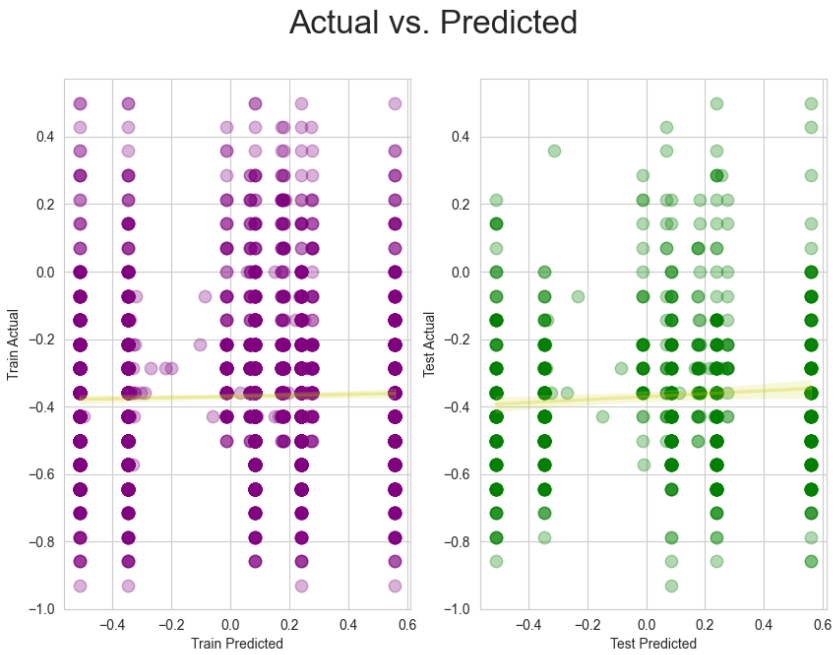


Figure 172: Graph: Abalone DNN Tuning yPred vs. yActual, Eta = 0.035



3.10.2 Testing Results

Eta = 0.035 gave the lowest MSE at MSE = 0.1645358 on the tuning data.

Figure 173: Graph: Abalone DNN 80% Data Avg Performance For Cross Validation, Eta = 0.035

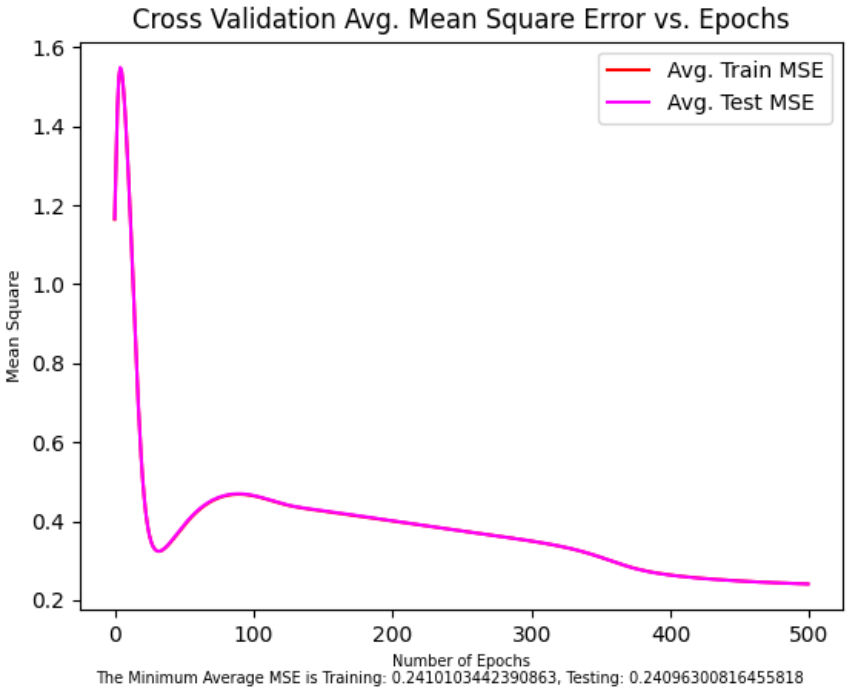
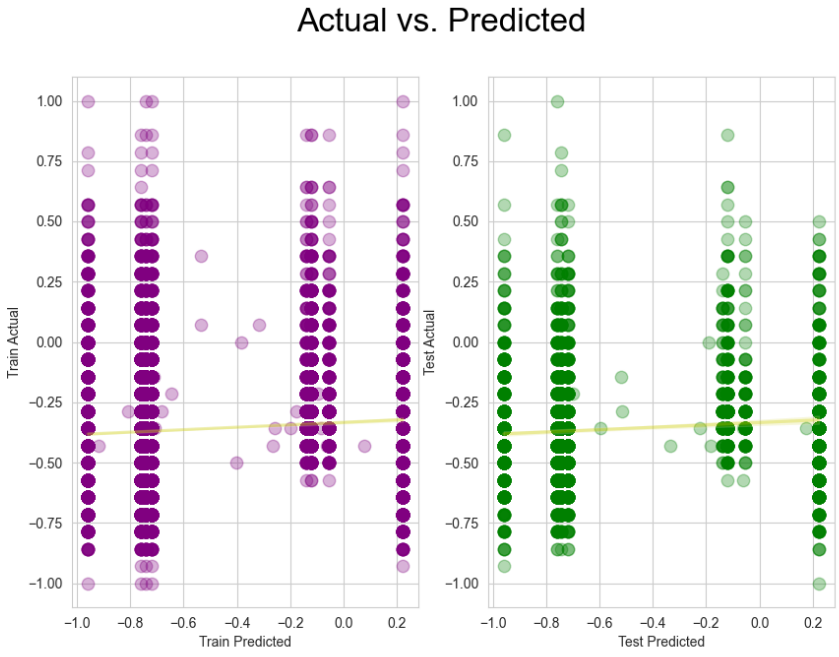


Figure 174: Graph: Abalone DNN 80% Data yPred vs. yActual, Eta = 0.035



3.11 Computer Regression via Deep Neural Network

3.11.1 Tuning Results

Figure 175: Graph: Computer DNN Tuning Avg Performance For Cross Validation, Eta = 0.01

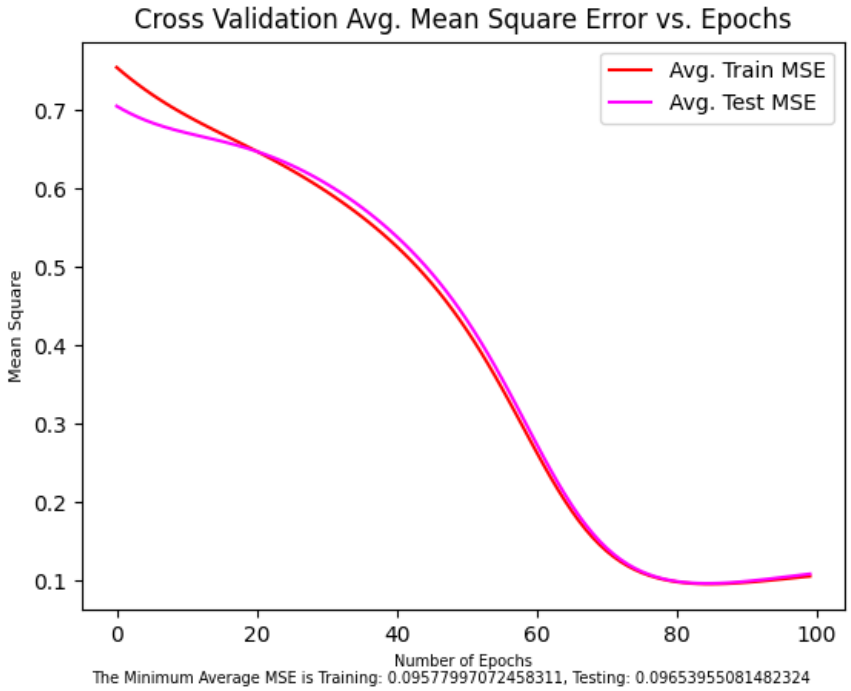


Figure 176: Graph: Computer DNN Tuning yPred vs. yActual, Eta = 0.01

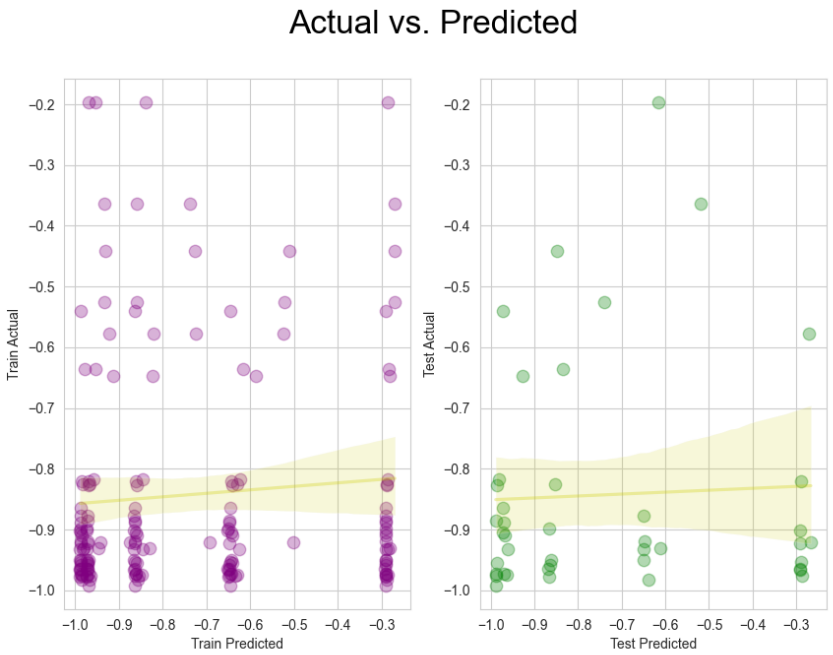


Figure 177: Graph: Computer DNN Avg Tuning Performance For Cross Validation, Eta = 0.0125

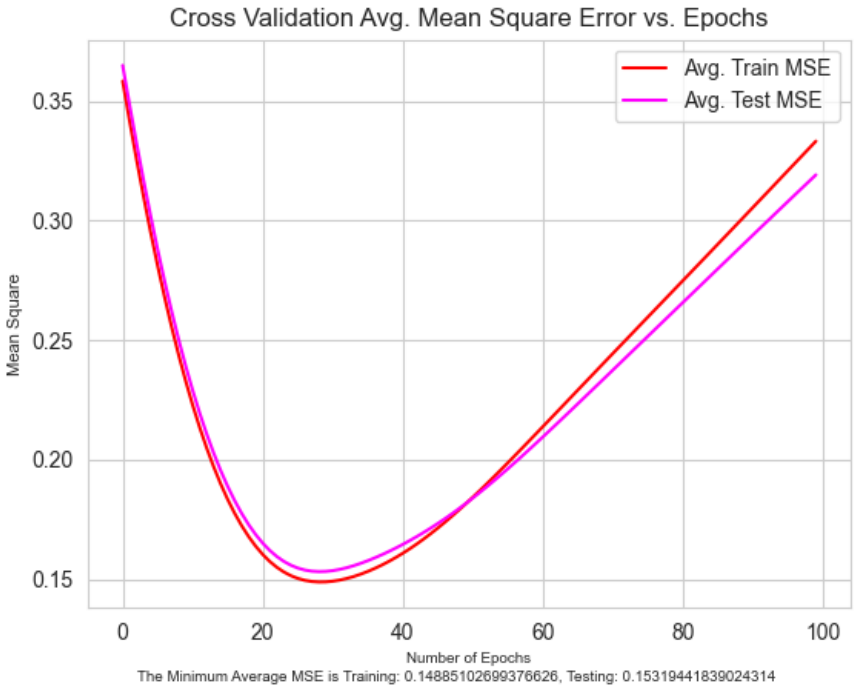


Figure 178: Graph: Computer DNN Tuning yPred vs. yActual, Eta = 0.0125

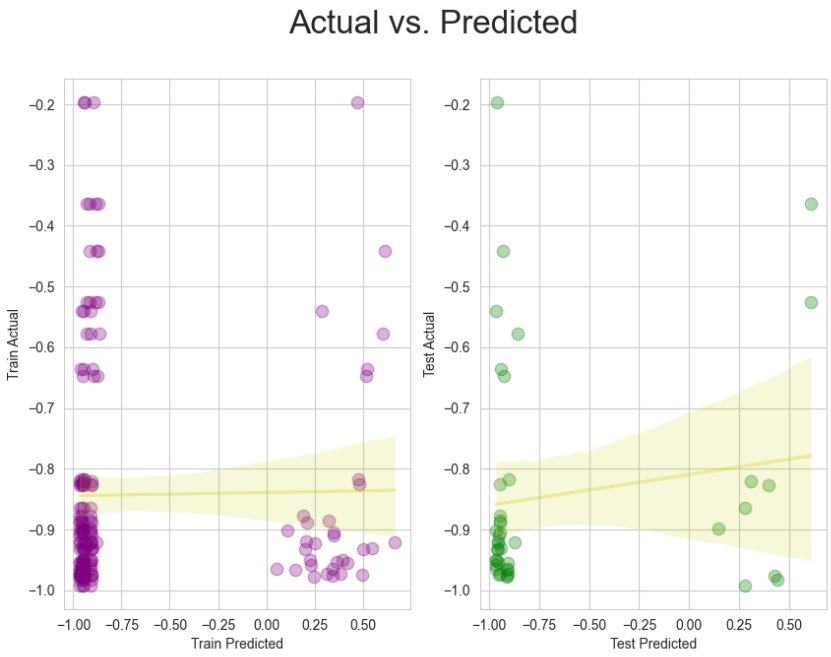


Figure 179: Graph: Computer DNN Tuning Avg Performance For Cross Validation, Eta = 0.015

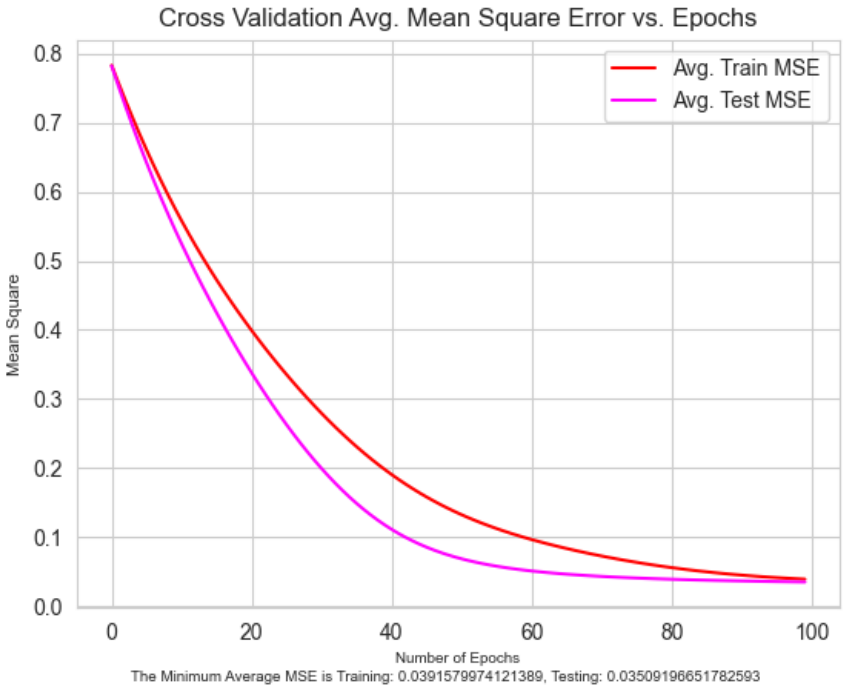


Figure 180: Graph: Computer DNN Tuning yPred vs. yActual, Eta = 0.015

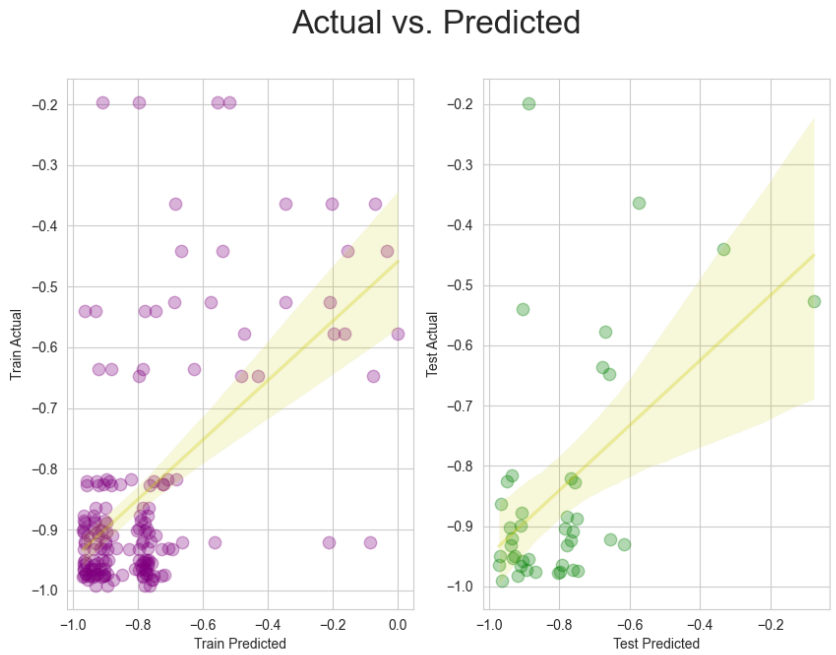


Figure 181: Graph: Computer DNN Tuning Avg Performance For Cross Validation, Eta = 0.0175

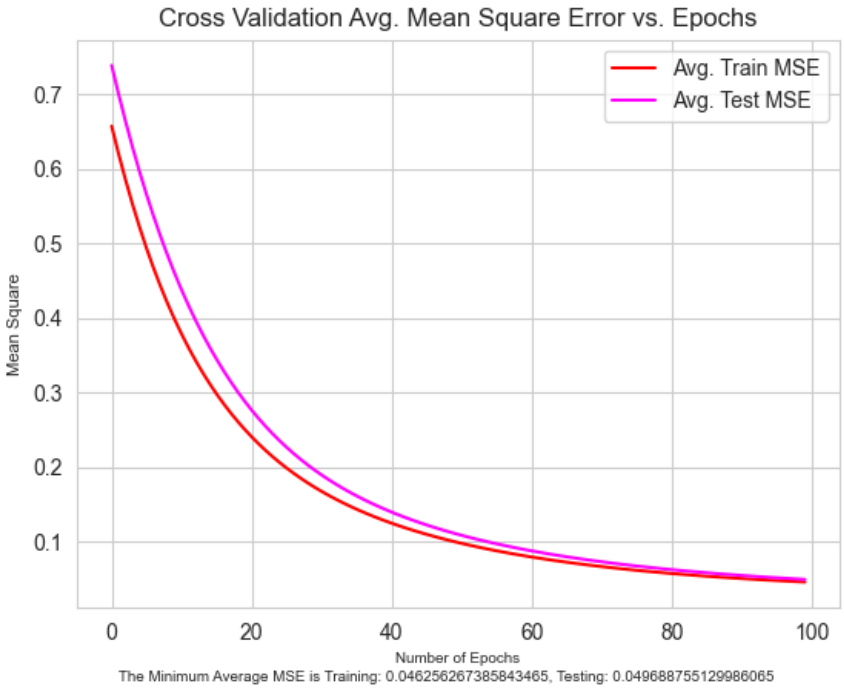


Figure 182: Graph: Computer DNN Tuning yPred vs. yActual, Eta = 0.0175

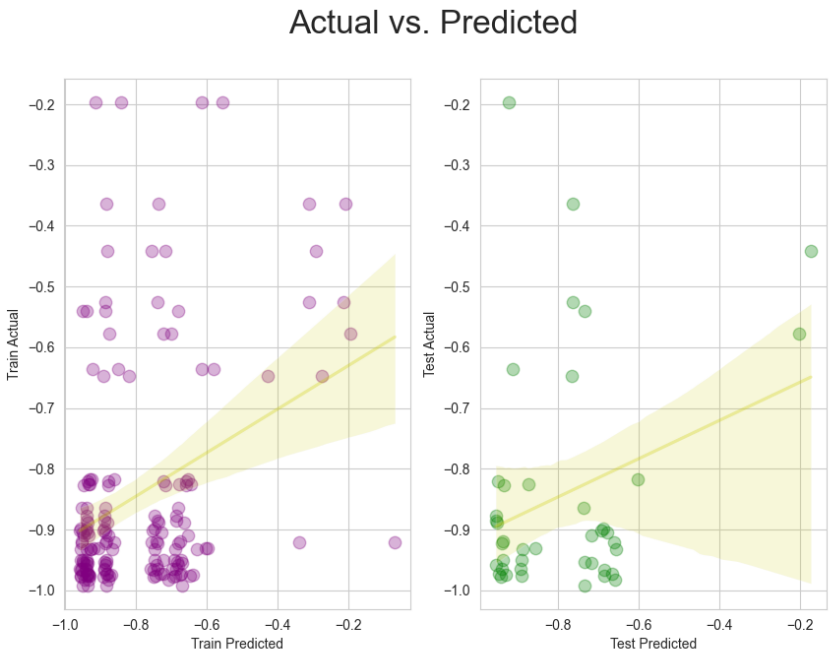


Figure 183: Graph: Computer DNN Tuning Avg Performance For Cross Validation, Eta = 0.02

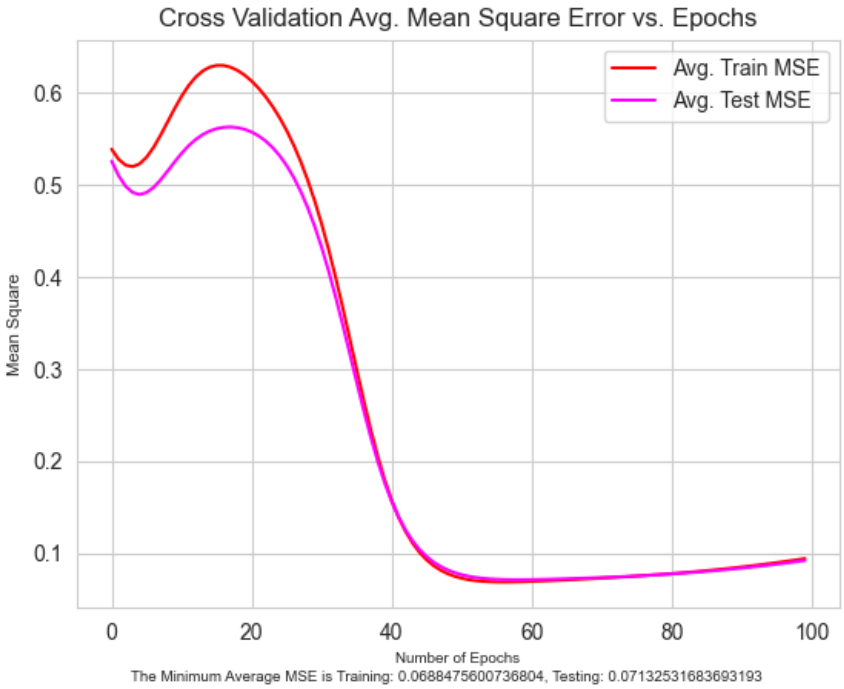
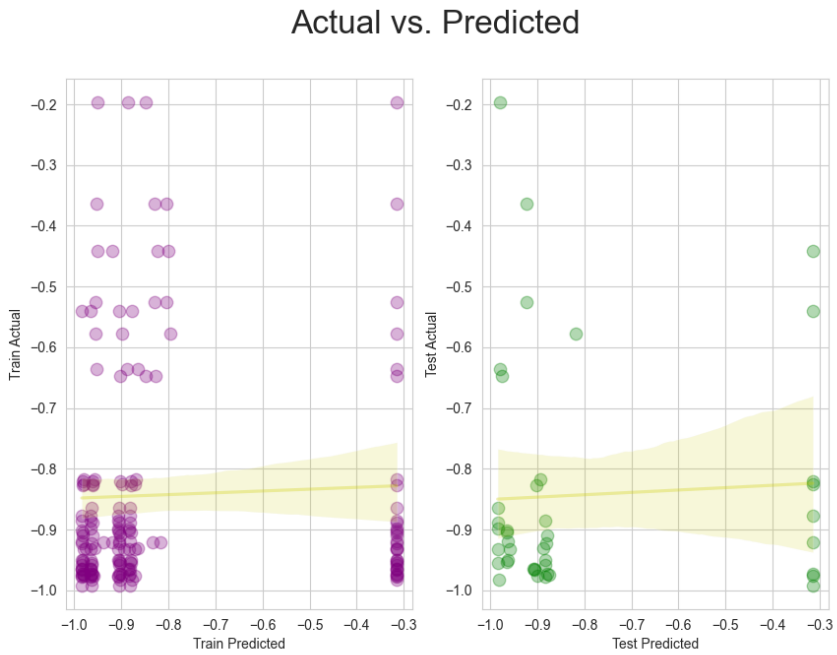


Figure 184: Graph: Computer DNN Tuning yPred vs. yActual, Eta = 0.02



3.11.2 Testing Results

ETA = 0.015 resulted in the lowest MSE for the tuning data at MSE = 0.03509.

Figure 185: Graph: Computer DNN 80% Data Avg Performance For Cross Validation, Eta = 0.015

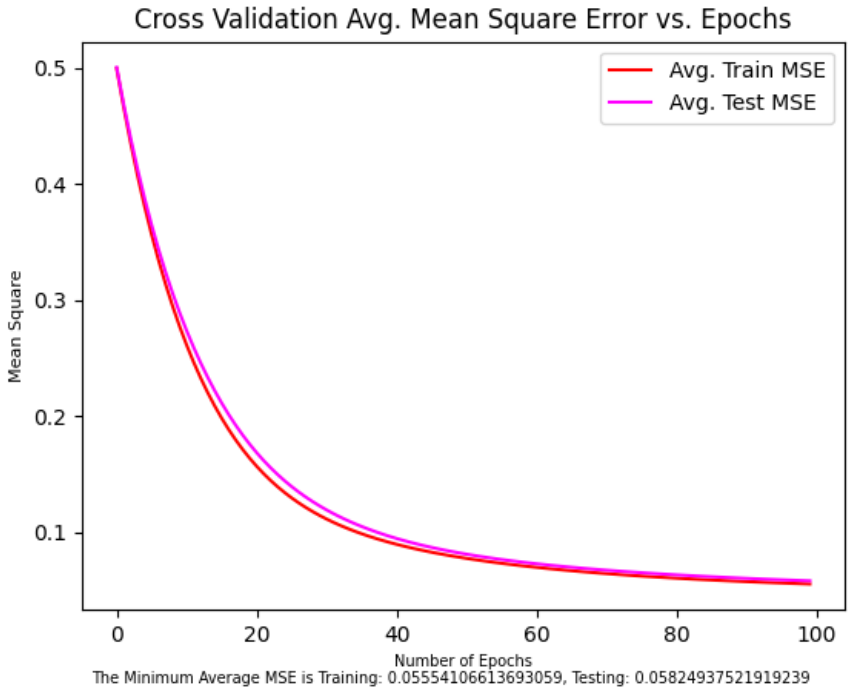
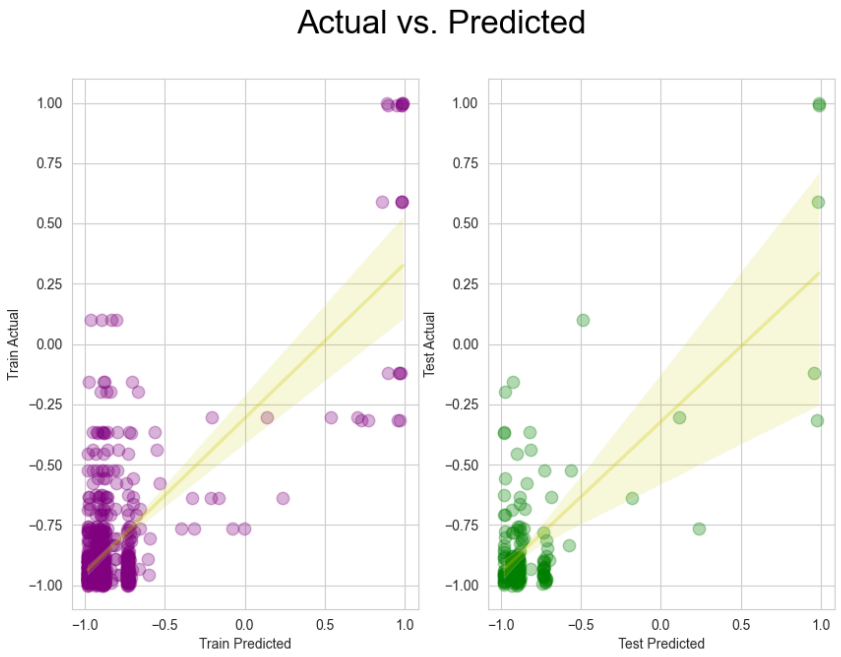


Figure 186: Graph: Computer DNN 80% Data yPred vs. yActual, Eta = 0.015



3.12 Forest Regression via Deep Neural Network

3.12.1 Tuning Results

Figure 187: Graph: Forest DNN Tuning Avg Performance For Cross Validation, Eta = 0.01

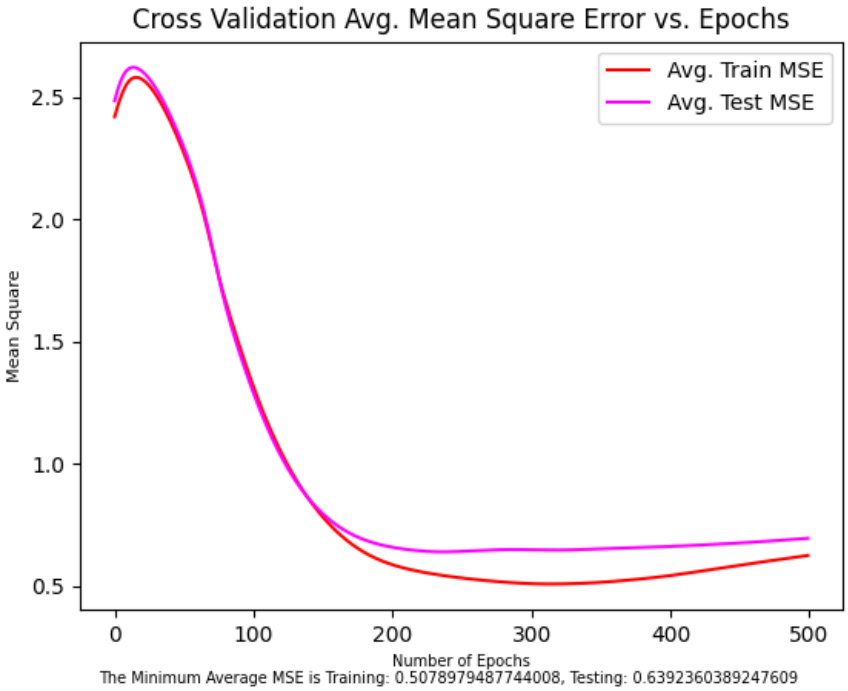


Figure 188: Graph: Forest DNN Tuning yPred vs. yActual, Eta = 0.01

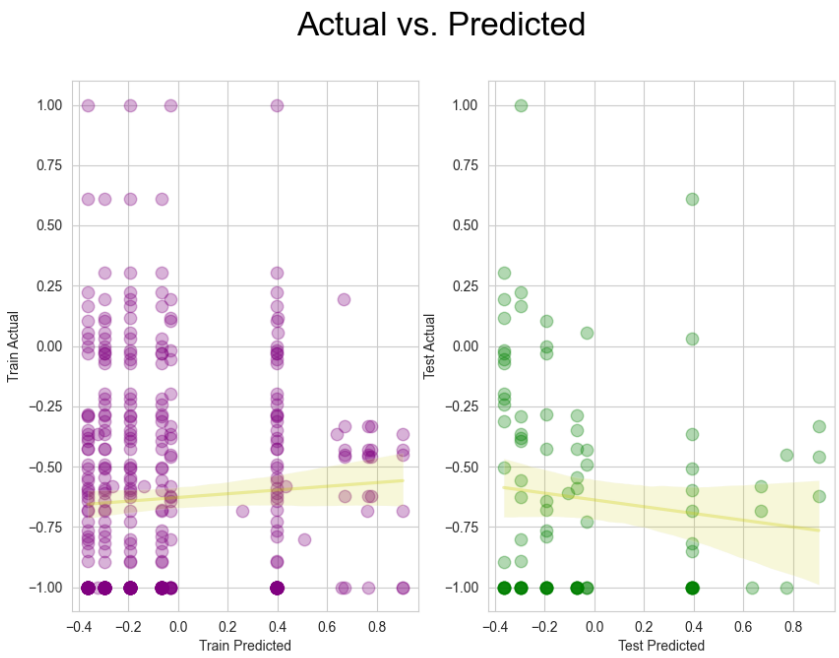


Figure 189: Graph: Forest DNN Avg Tuning Performance For Cross Validation, Eta = 0.0125

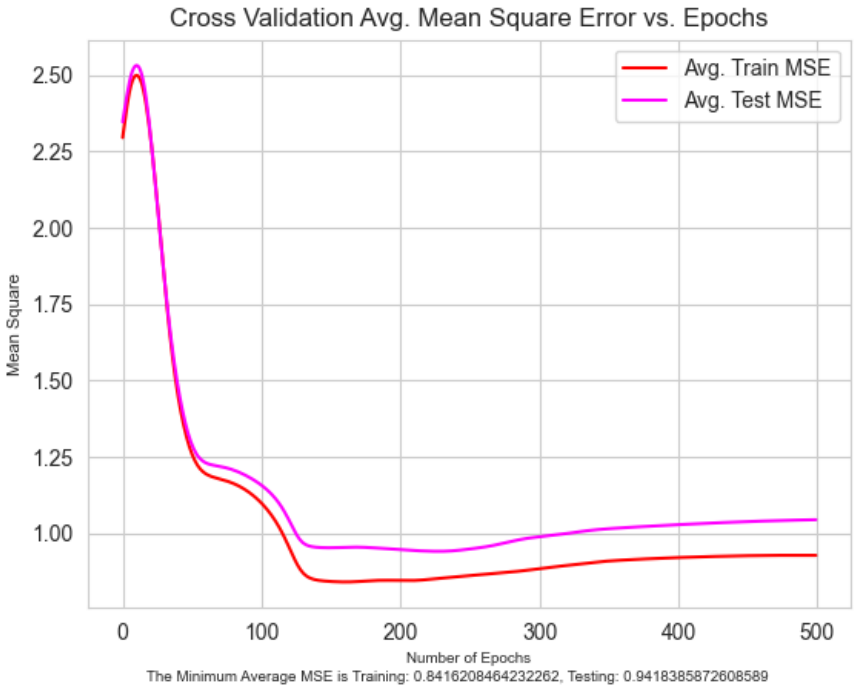


Figure 190: Graph: Forest DNN Tuning yPred vs. yActual, Eta = 0.0125

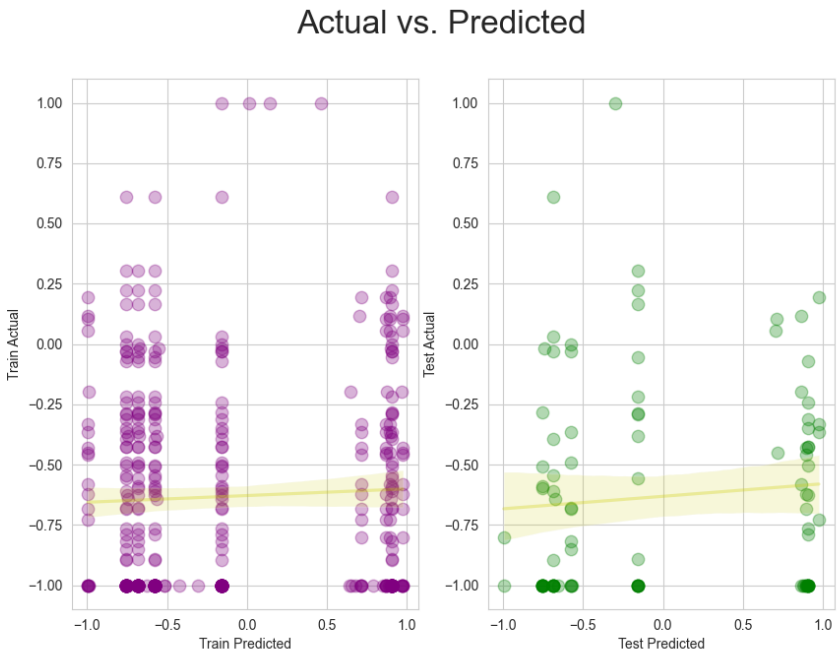


Figure 191: Graph: Forest DNN Tuning Avg Performance For Cross Validation, Eta = 0.015

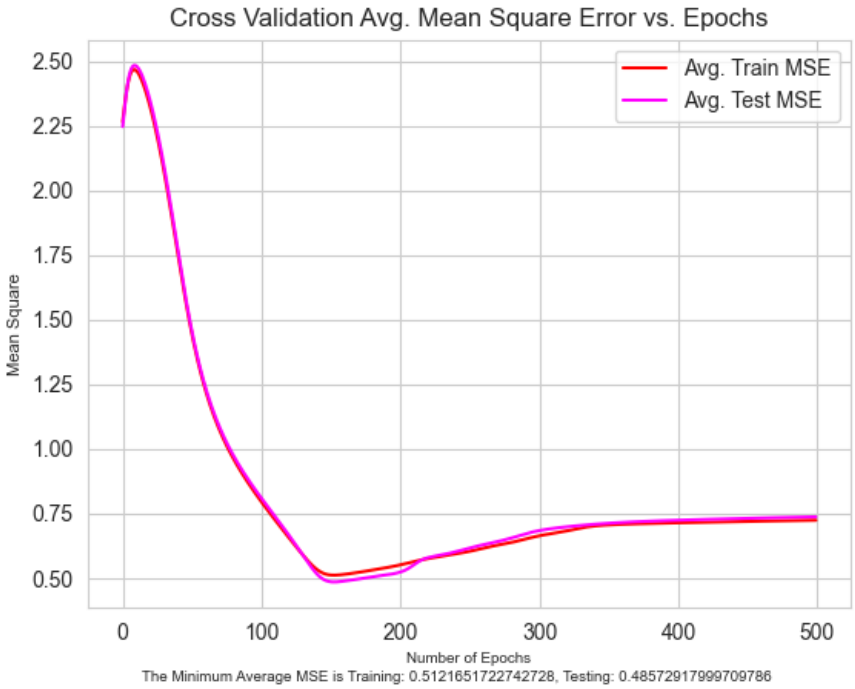


Figure 192: Graph: Forest DNN Tuning yPred vs. yActual, Eta = 0.015

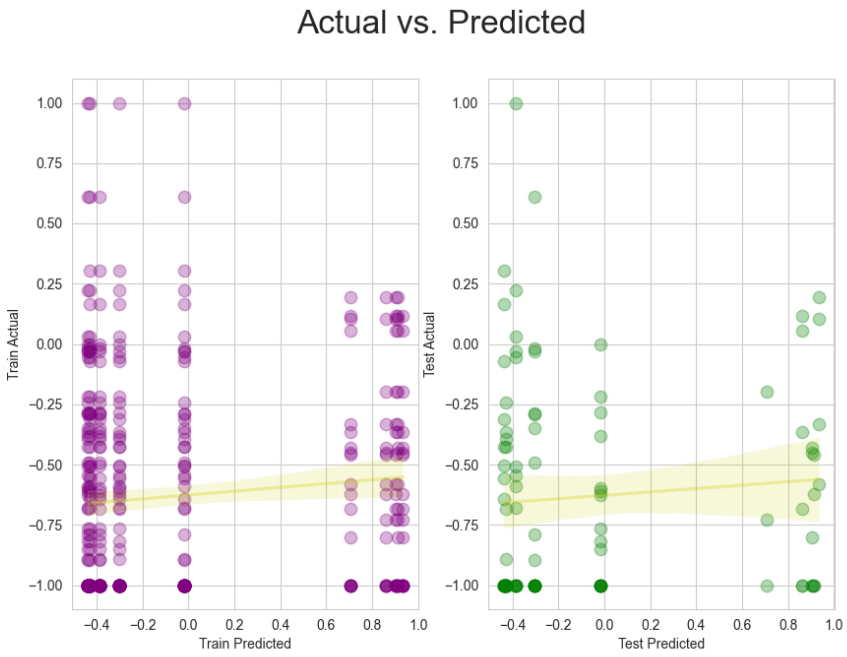


Figure 193: Graph: Forest DNN Tuning Avg Performance For Cross Validation, Eta = 0.0175

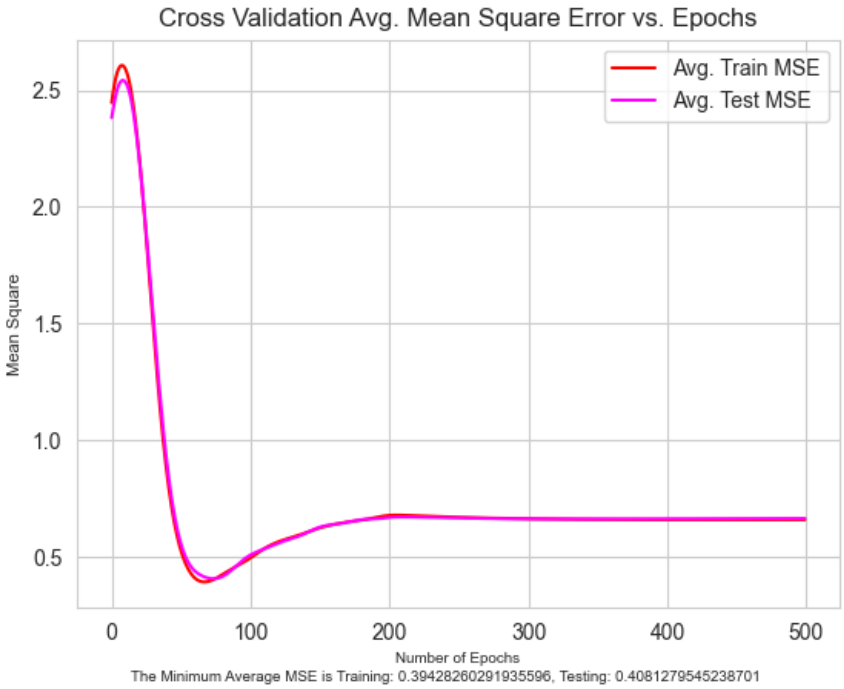


Figure 194: Graph: Forest DNN Tuning yPred vs. yActual, Eta = 0.0175

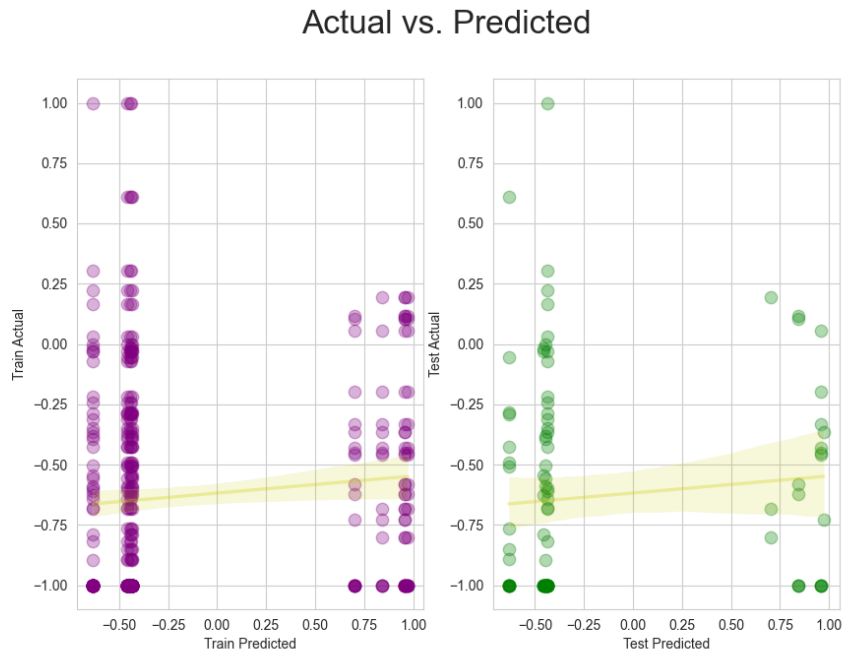


Figure 195: Graph: Forest DNN Tuning Avg Performance For Cross Validation, Eta = 0.02

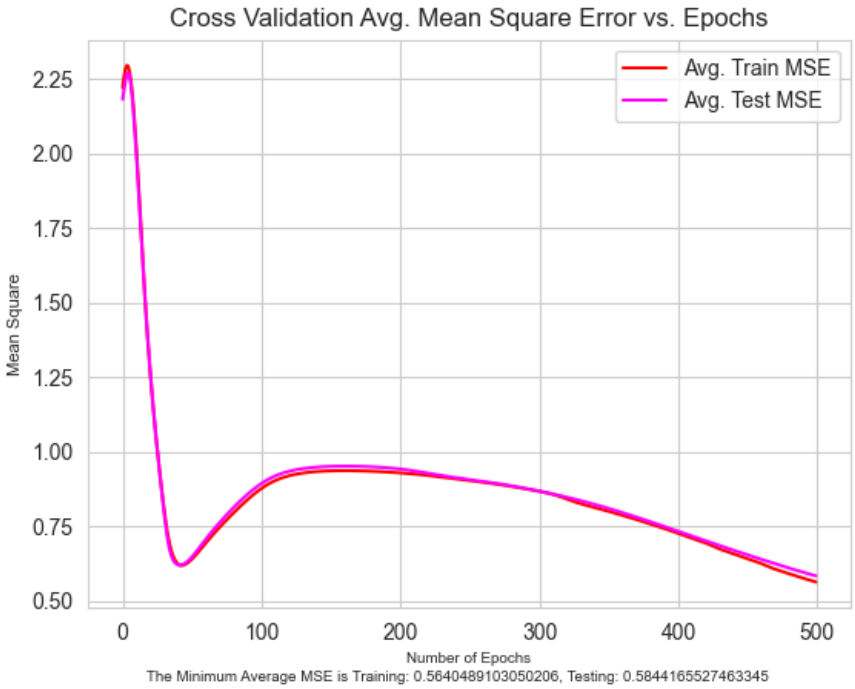
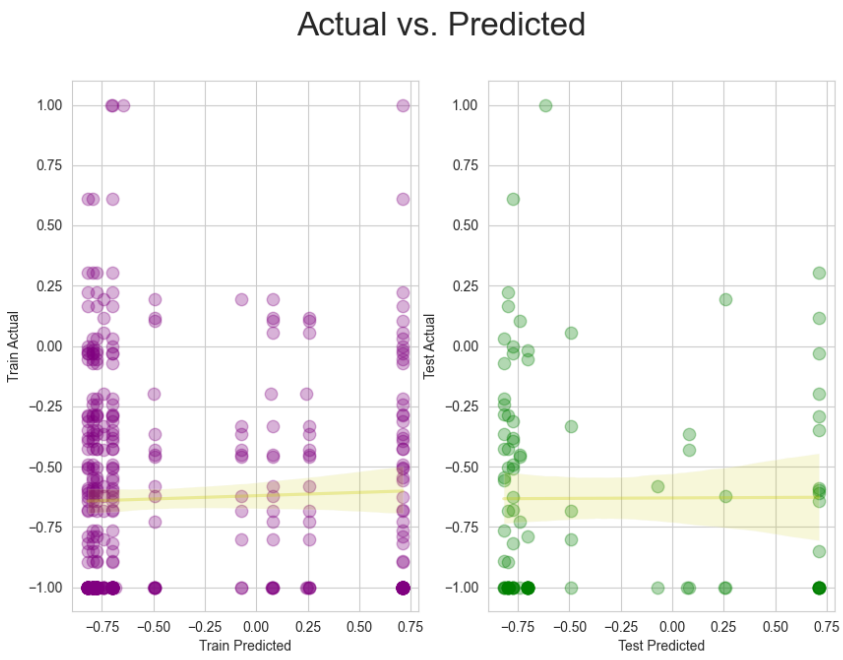


Figure 196: Graph: Forest DNN Tuning yPred vs. yActual, Eta = 0.02



3.12.2 Testing Results

ETA = 0.0175 gives the lowest MSE at MSE = 0.4081.

Figure 197: Graph: Forest DNN 80% Data Avg Performance For Cross Validation, Eta = 0.0175)

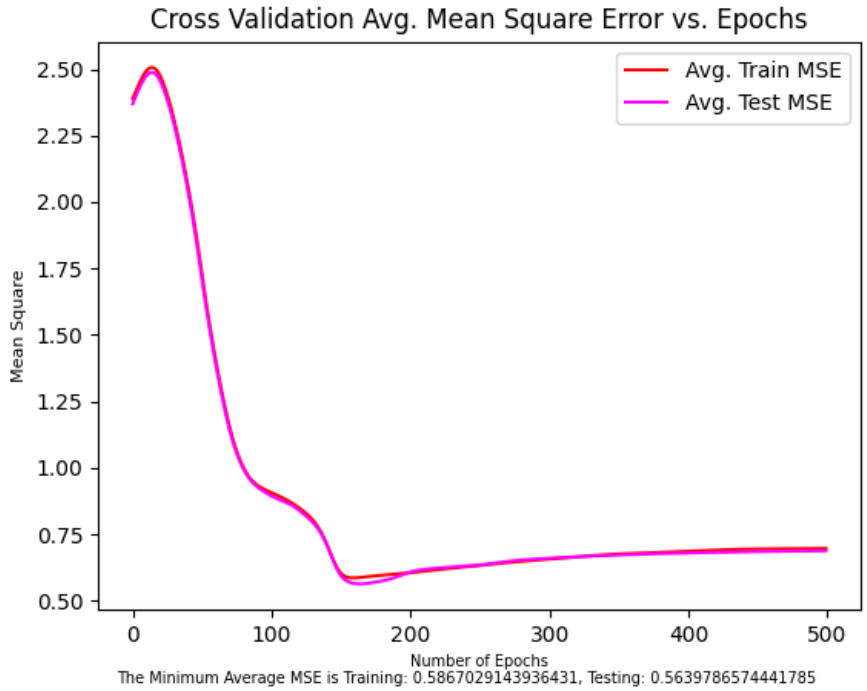
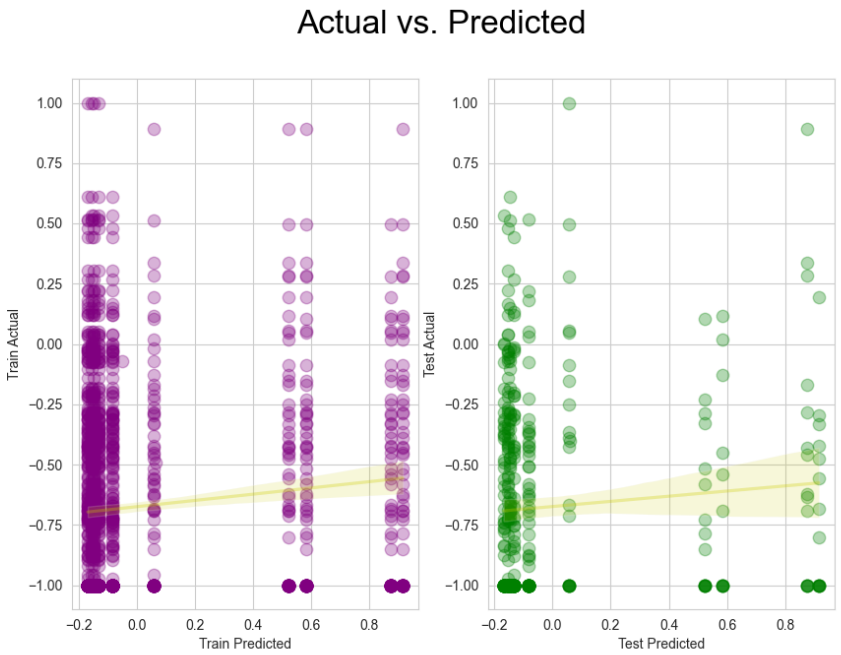


Figure 198: Graph: Forest DNN 80% Data yPred vs. yActual, Eta = 0.0175



3.13 Car Classification via Autoencoder to DNN

3.13.1 Tuning Results

Figure 199: Graph: Car AEDNN Network Tuning Training Confusion Matrix , Eta = 0.01

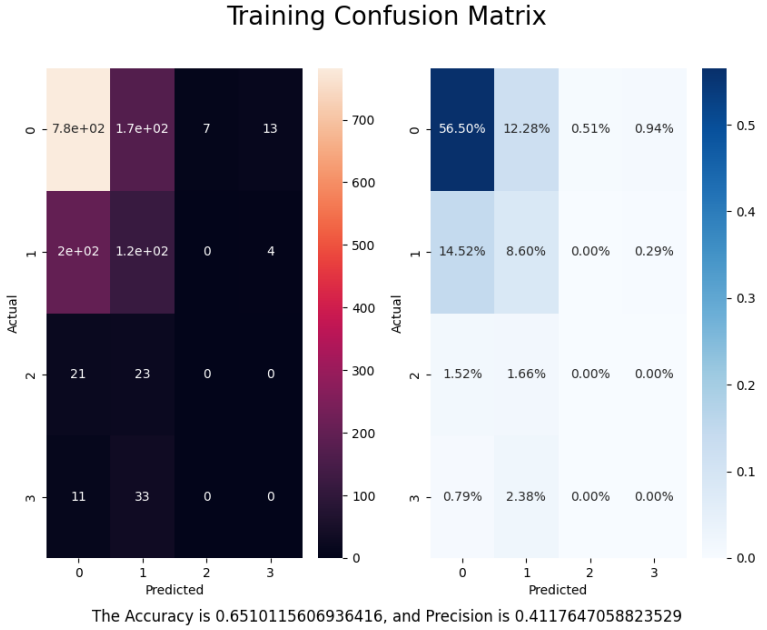


Figure 200: Graph: Car AEDNN Tuning Testing Confusion Matrix , Eta = 0.01

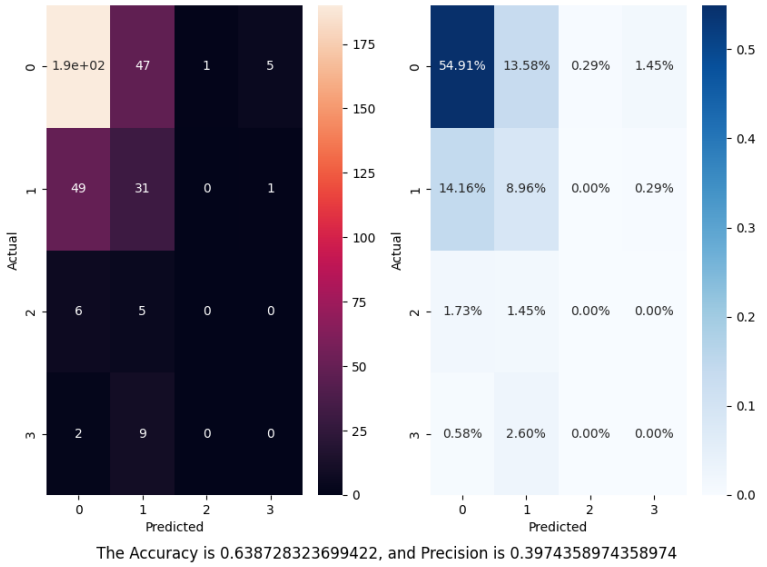


Figure 201: Graph: Car AEDNN Tuning Training Confusion Matrix , Eta = 0.015

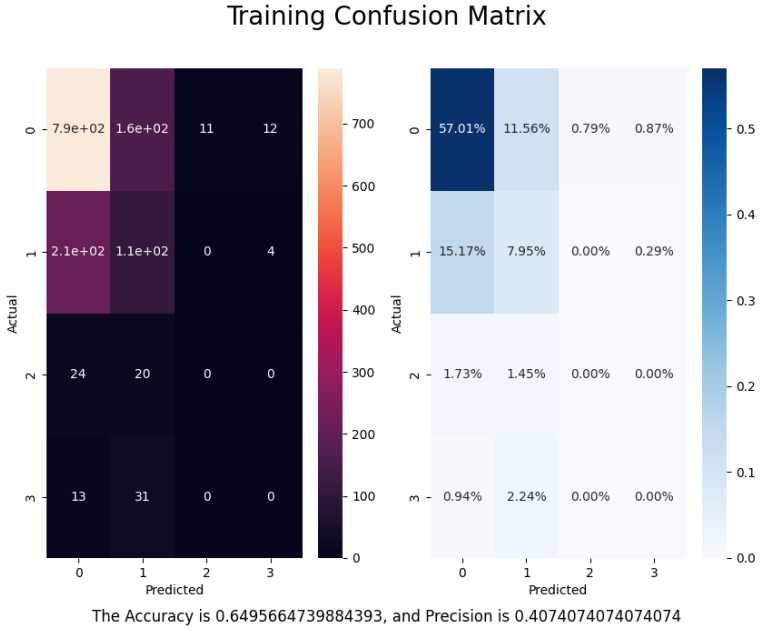


Figure 202: Graph: Car AEDNN Tuning Testing Confusion Matrix , Eta = 0.015

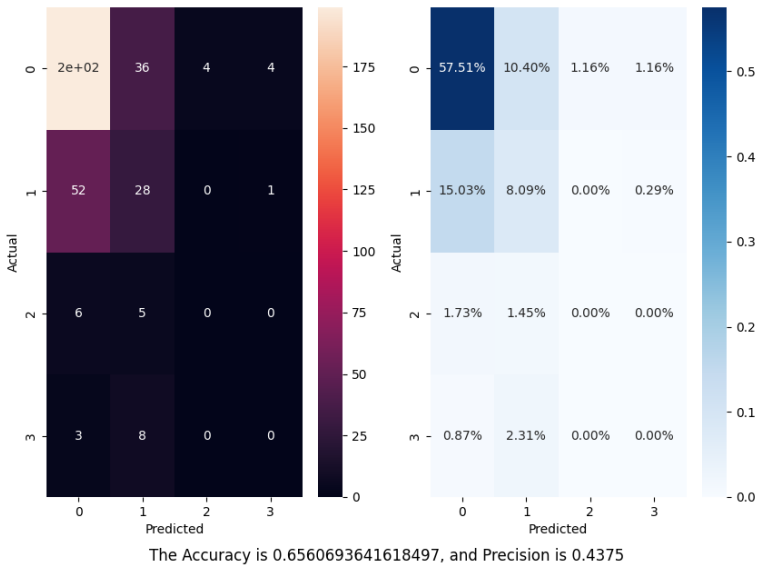


Figure 203: Graph: Car AEDNN Tuning Training Confusion Matrix , Eta = 0.02

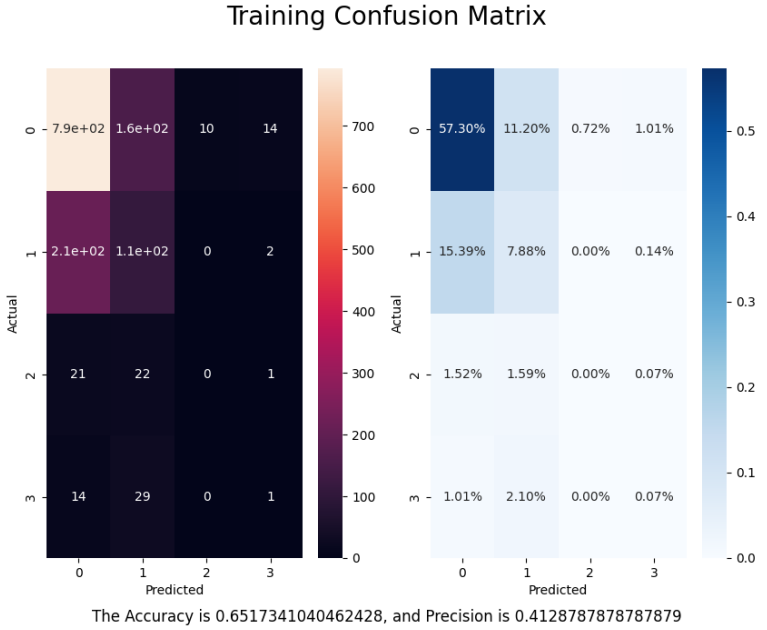


Figure 204: Graph: Car AEDNN Tuning Testing Confusion Matrix , Eta = 0.02

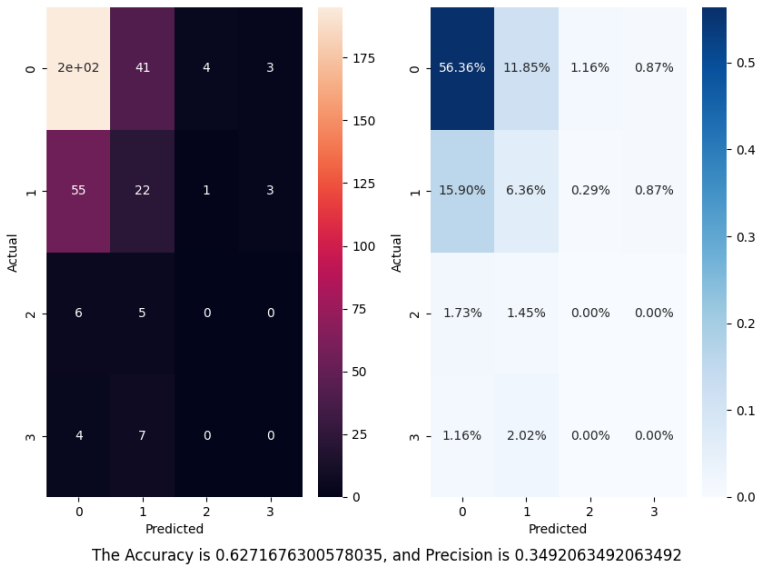


Figure 205: Graph: Car AEDNN Tuning Training Confusion Matrix , Eta = 0.03

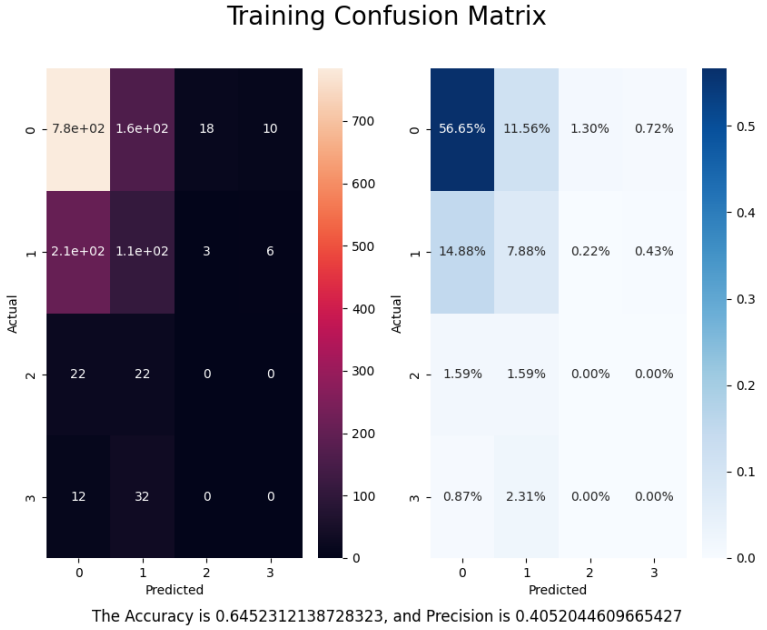


Figure 206: Graph: Car AEDNN Tuning Testing Confusion Matrix , Eta = 0.03

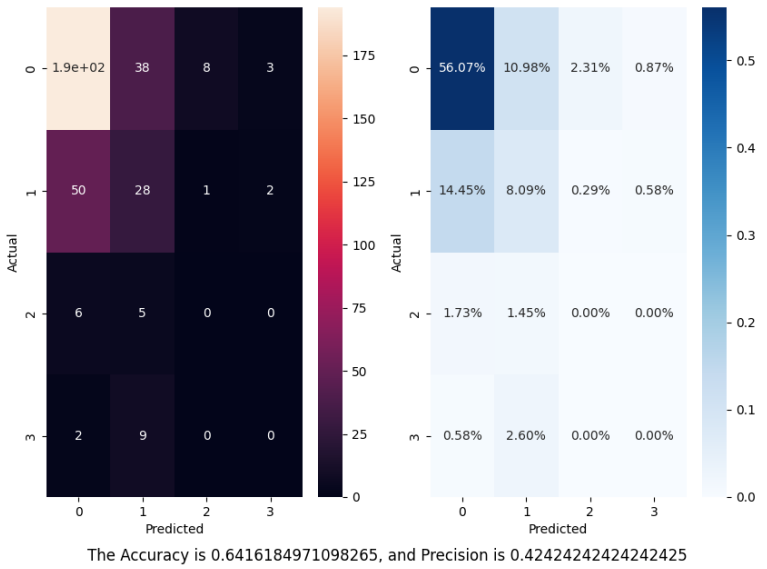


Figure 207: Graph: Car AEDNN Tuning Training Confusion Matrix , Eta = 0.035

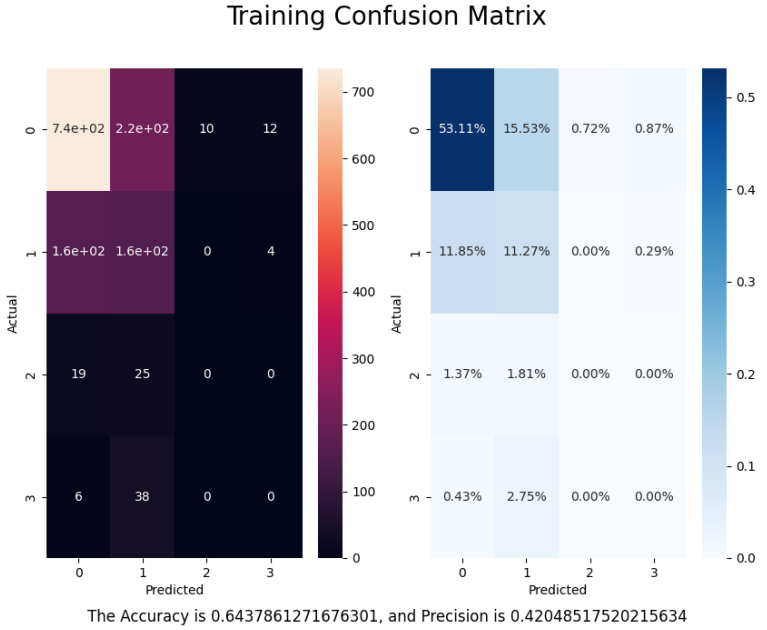
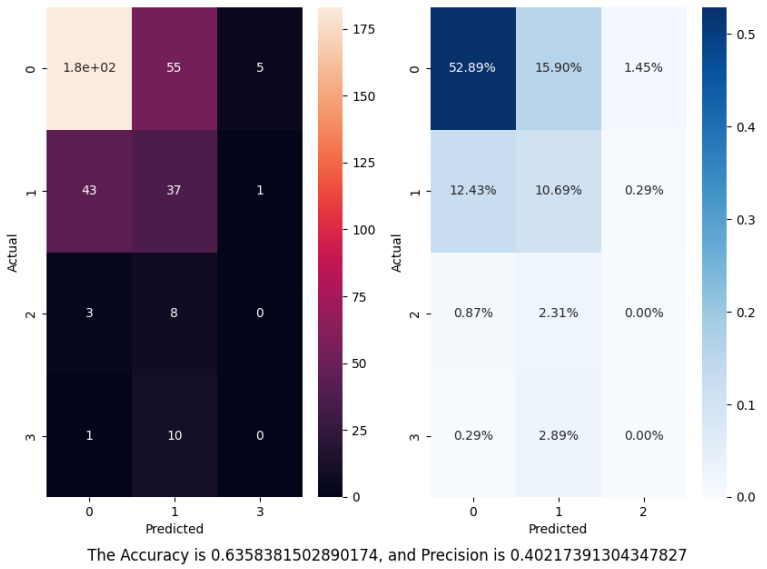


Figure 208: Graph: Car AEDNN Tuning Testing Confusion Matrix , Eta = 0.035



3.13.2 Testing Results

ETA = 0.015 gave the best accuracy at 0.6560.

Figure 209: Graph: Car AEDNN 80% Training Confusion Matrix , Eta = 0.015

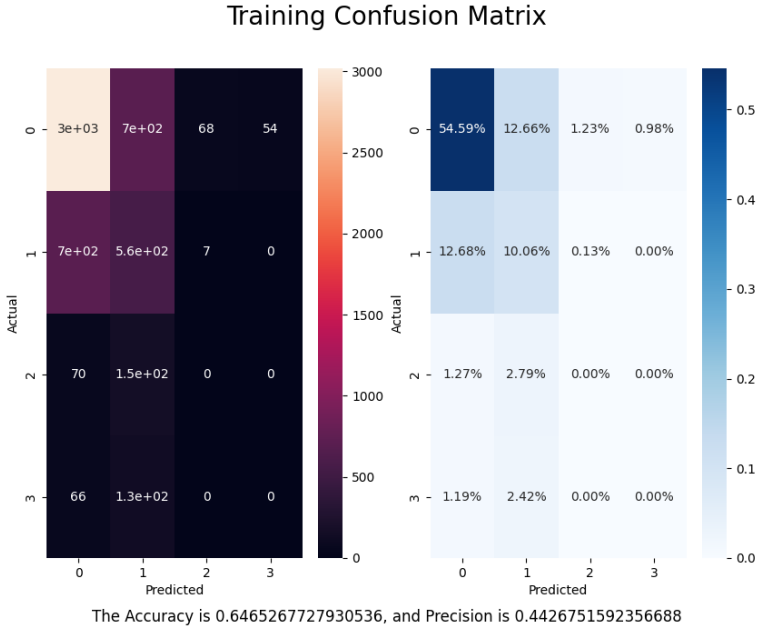
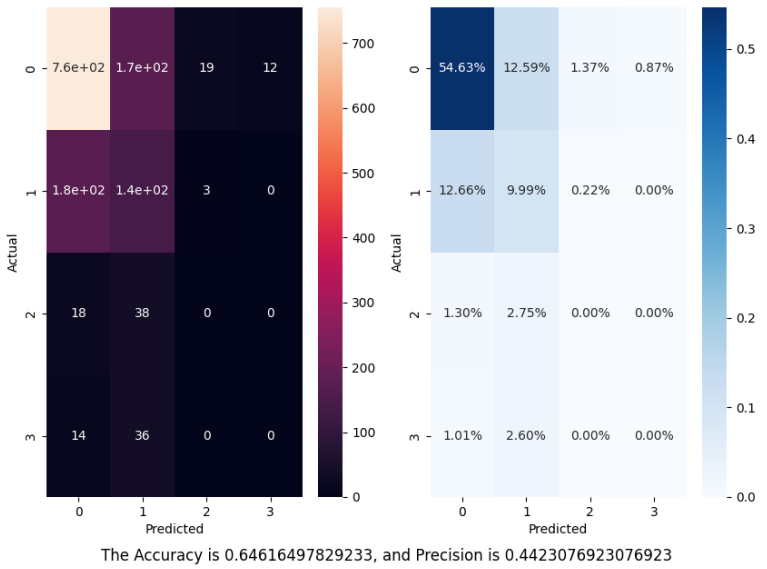


Figure 210: Graph: Car AEDNN 80% Data Testing Confusion Matrix , Eta = 0.015



3.14 Voting Classification via Autoencoder to DNN

3.14.1 Tuning Results

Figure 211: Graph: Vote AEDNN Network Tuning Training Confusion Matrix , Eta = 0.01

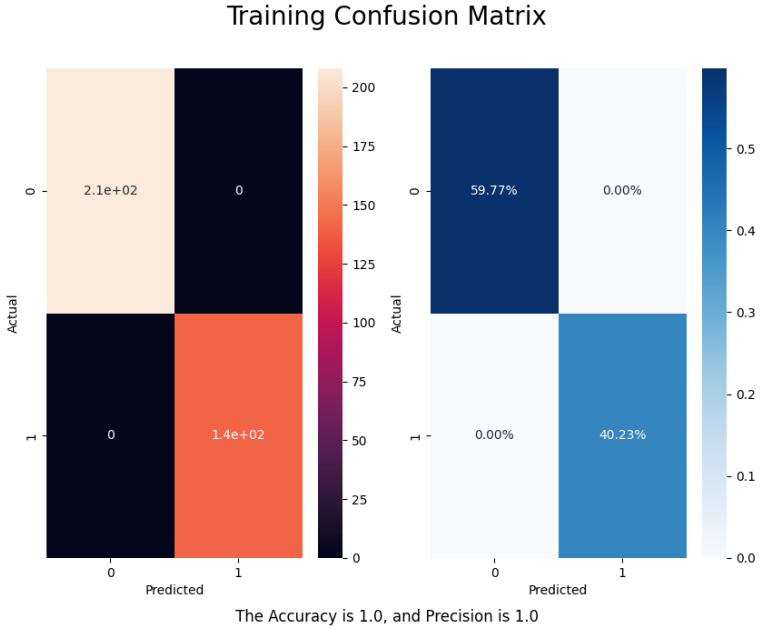


Figure 212: Graph: Vote AEDNN Tuning Testing Confusion Matrix , Eta = 0.01

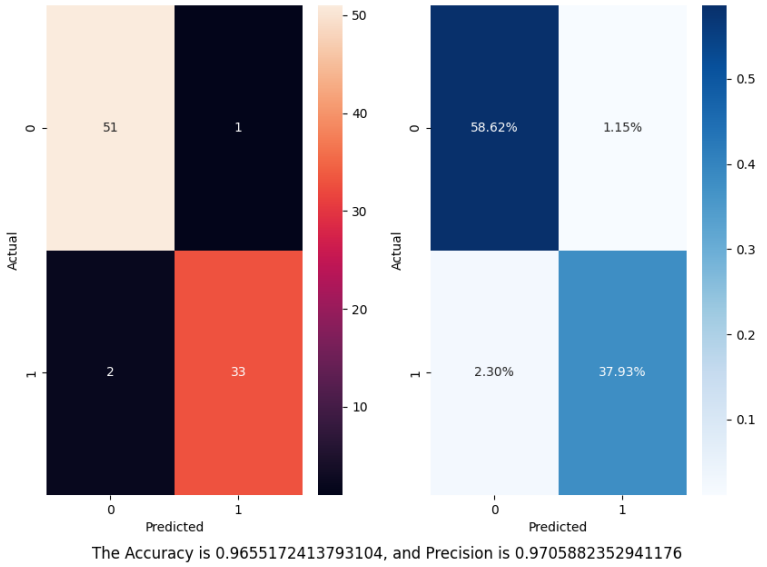


Figure 213: Graph: Vote AEDNN Tuning Training Confusion Matrix , Eta = 0.015

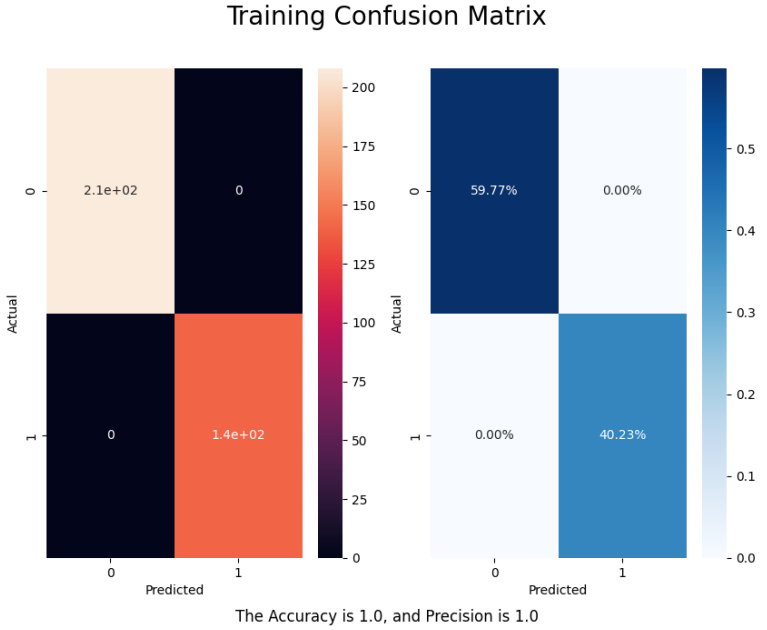


Figure 214: Graph: Vote AEDNN Tuning Testing Confusion Matrix , Eta = 0.015

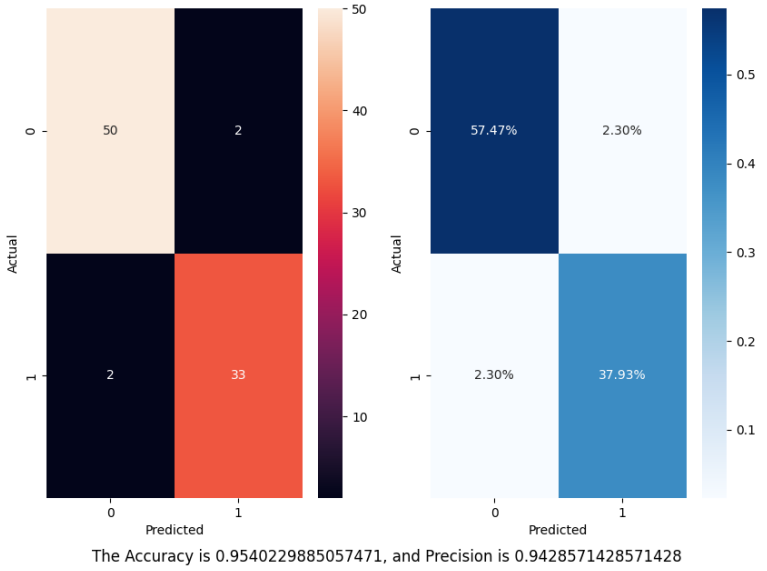


Figure 215: Graph: Vote AEDNN Tuning Training Confusion Matrix , Eta = 0.02

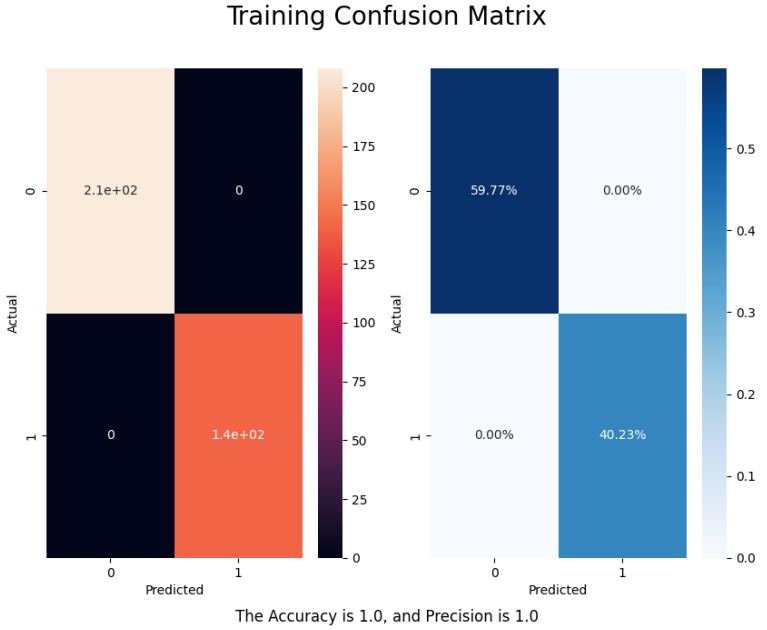


Figure 216: Graph: Vote AEDNN Tuning Testing Confusion Matrix , Eta = 0.02

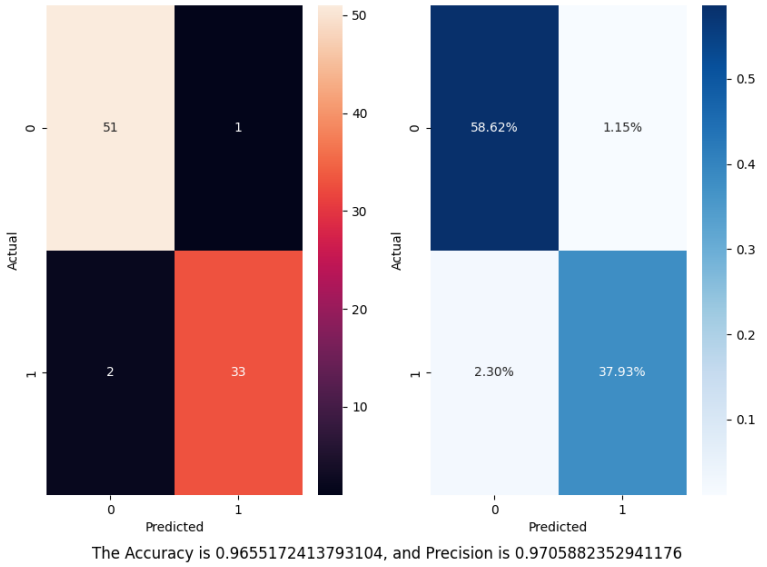


Figure 217: Graph: Vote AEDNN Tuning Training Confusion Matrix , Eta = 0.03

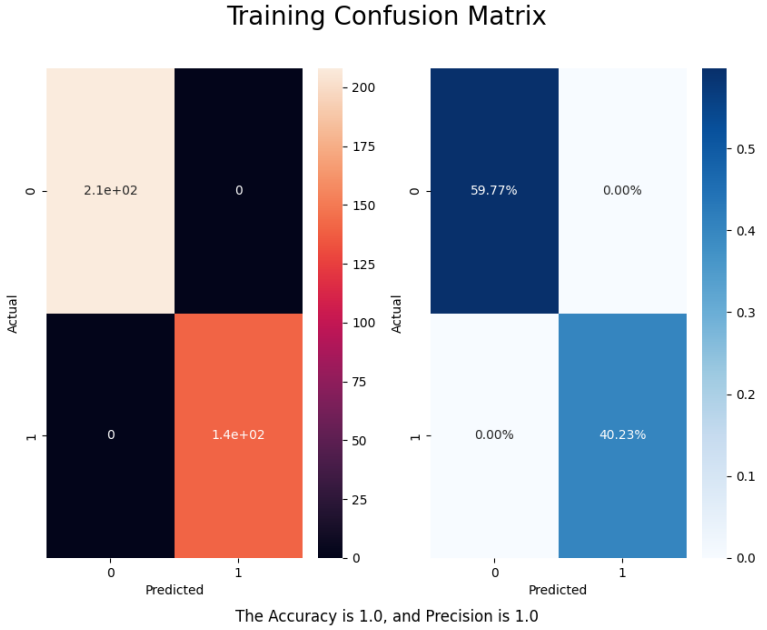


Figure 218: Graph: Vote AEDNN Tuning Testing Confusion Matrix , Eta = 0.03

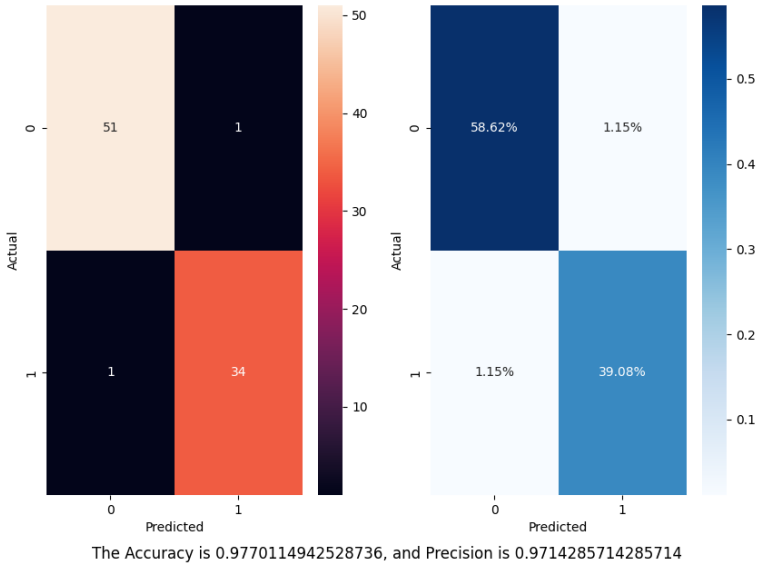


Figure 219: Graph: Vote AEDNN Tuning Training Confusion Matrix , Eta = 0.035

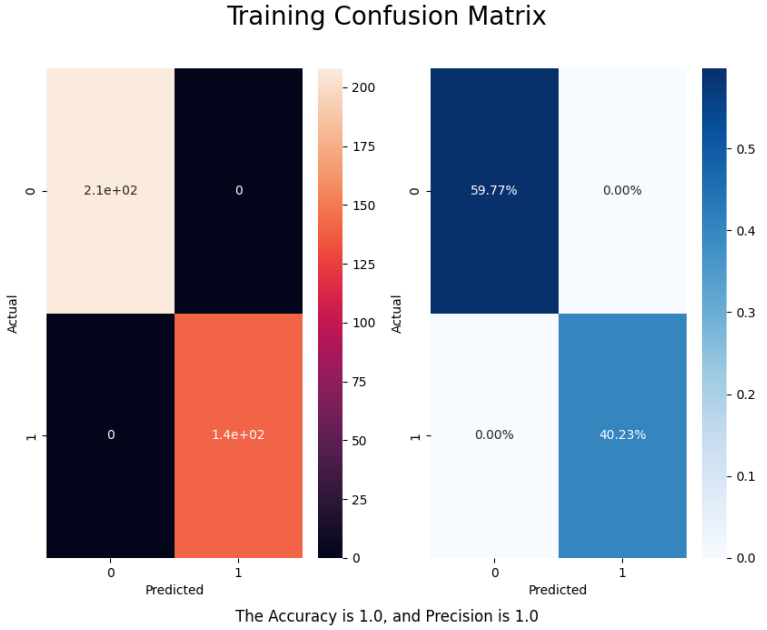
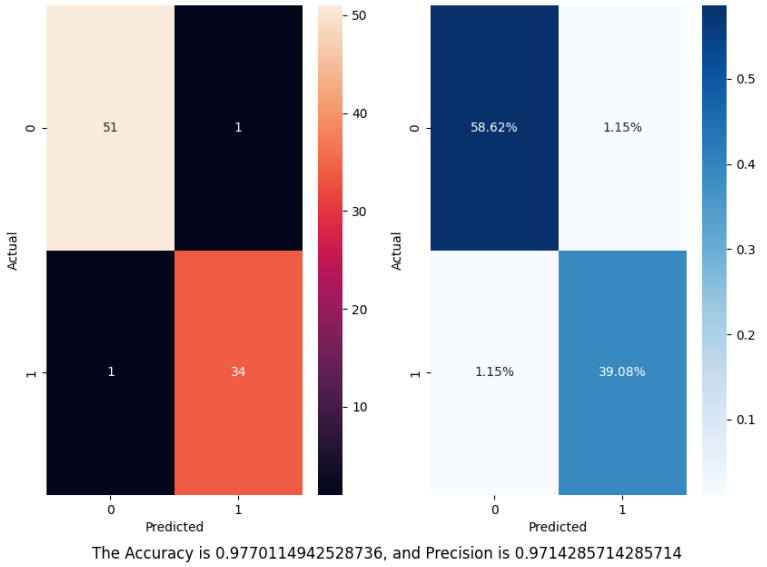


Figure 220: Graph: Vote AEDNN Tuning Testing Confusion Matrix , Eta = 0.035



3.14.2 Testing Results

ETA = 0.03 gave the best accuracy at 97.70%.

Figure 221: Graph: Vote AEDNN 80% Training Confusion Matrix , Eta = 0.03

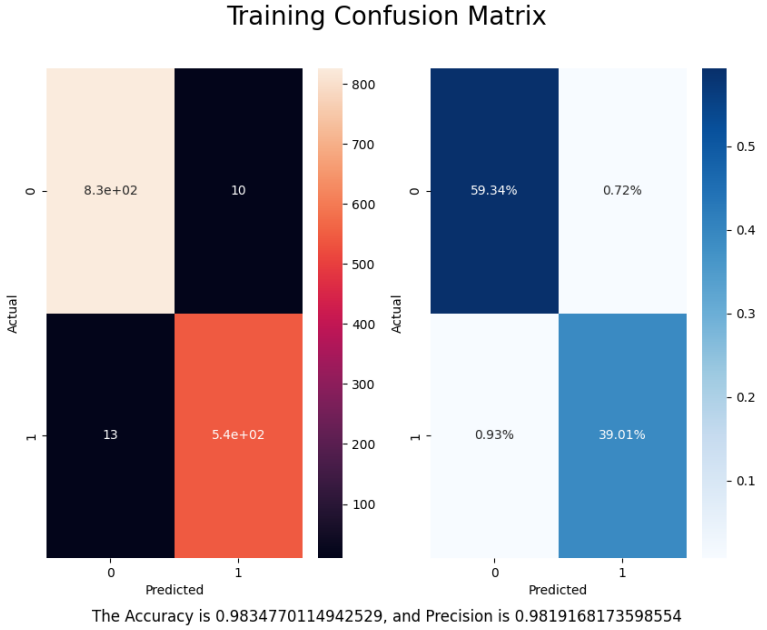
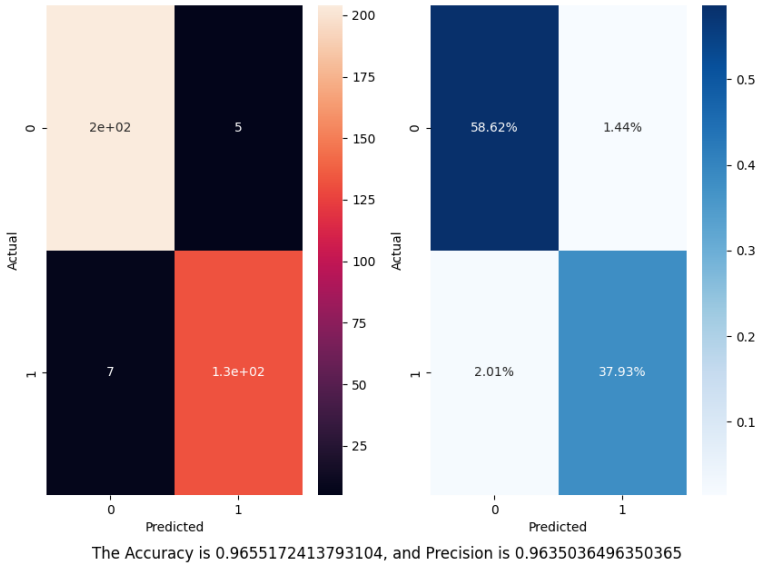


Figure 222: Graph: Vote AEDNN 80% Data Testing Confusion Matrix , Eta = 0.03



3.15 Breast Cancer Classification via Autoencoder to DNN

3.15.1 Tuning Results

Figure 223: Graph: Breast AEDNN Network Tuning Training Confusion Matrix , Eta = 0.01

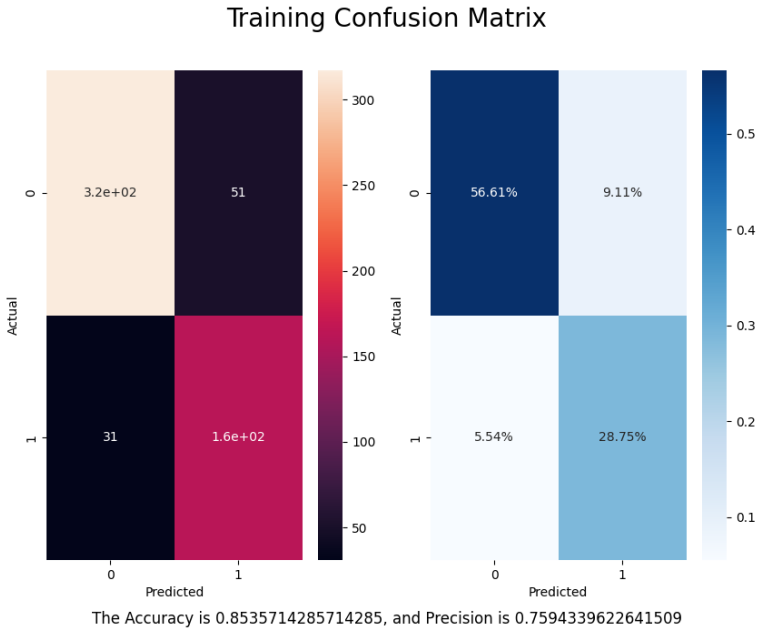


Figure 224: Graph: Breast AEDNN Tuning Testing Confusion Matrix , Eta = 0.01

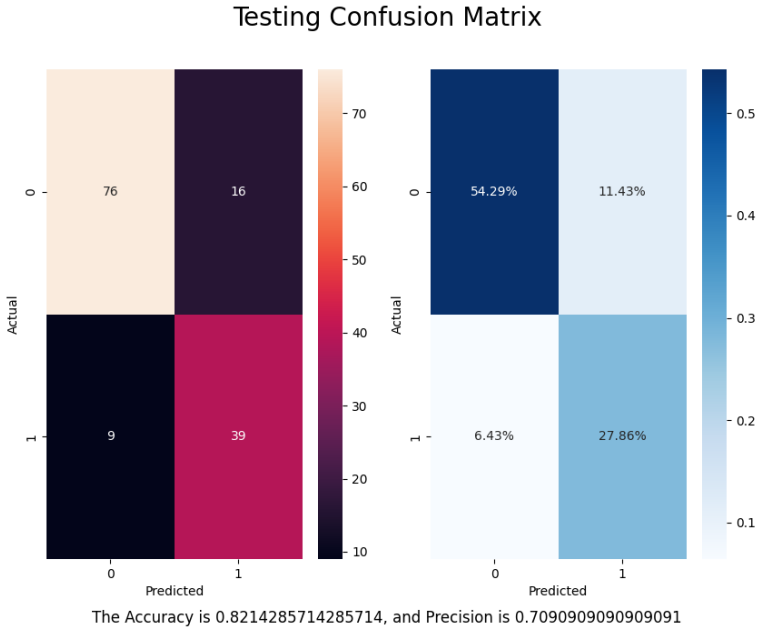


Figure 225: Graph: Breast AEDNN Tuning Training Confusion Matrix , Eta = 0.015

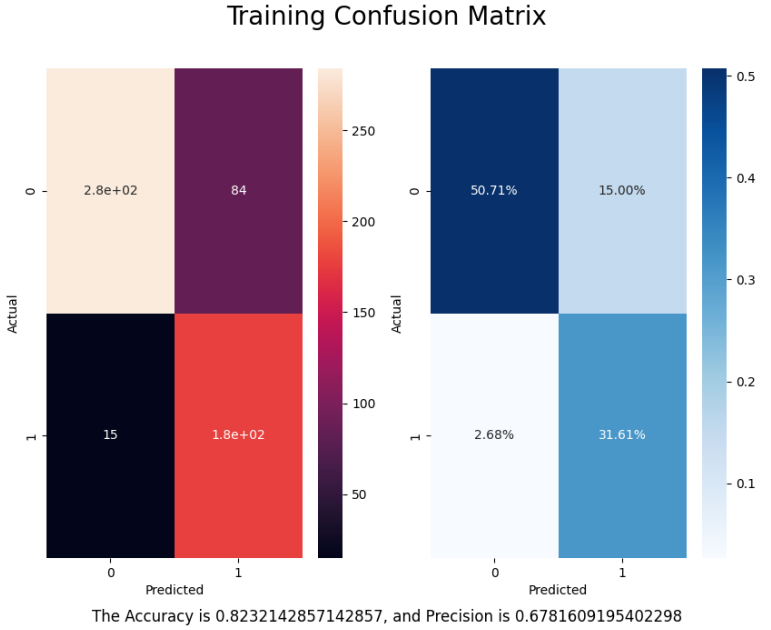


Figure 226: Graph: Breast AEDNN Tuning Testing Confusion Matrix , Eta = 0.015

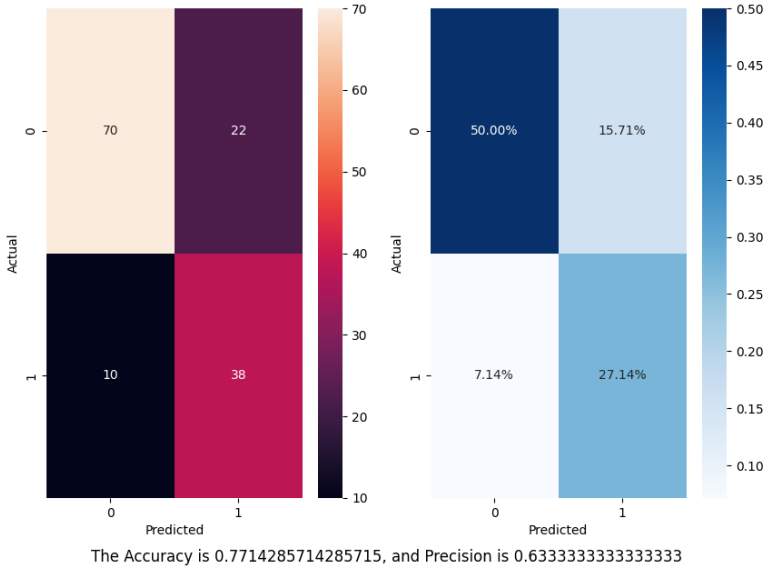


Figure 227: Graph: Breast AEDNN Tuning Training Confusion Matrix , Eta = 0.02

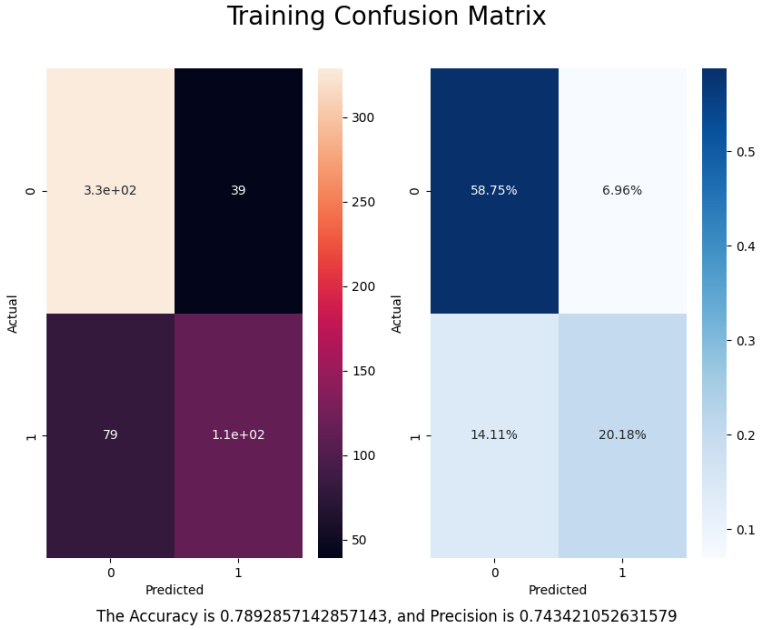


Figure 228: Graph: Breast AEDNN Tuning Testing Confusion Matrix , Eta = 0.02

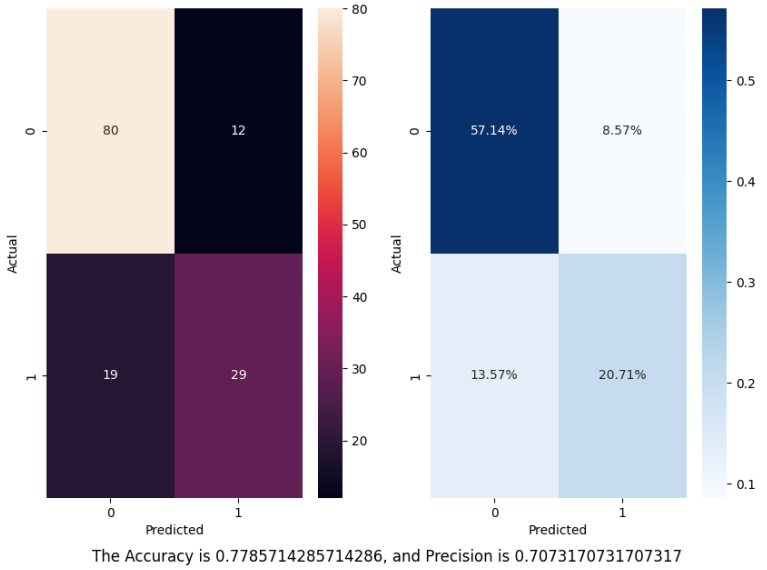


Figure 229: Graph: Breast AEDNN Tuning Training Confusion Matrix , Eta = 0.03

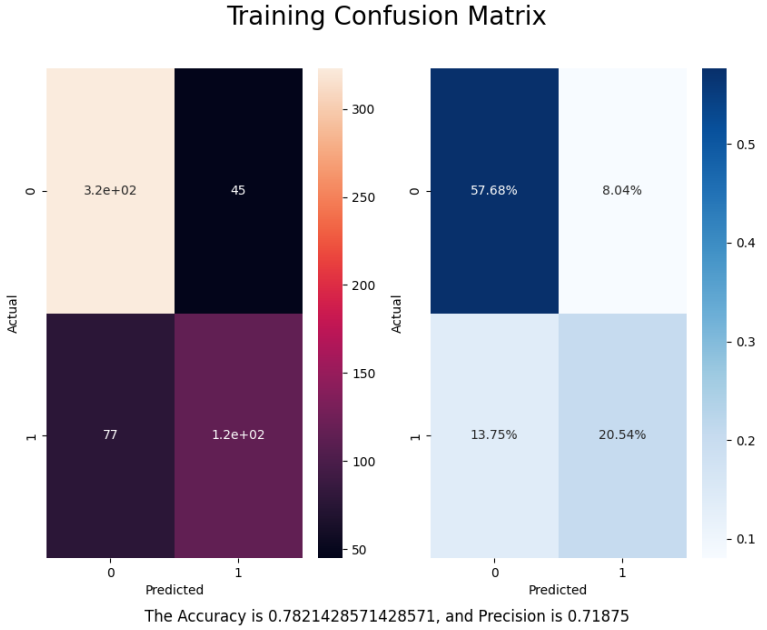


Figure 230: Graph: Breast AEDNN Tuning Testing Confusion Matrix , Eta = 0.03

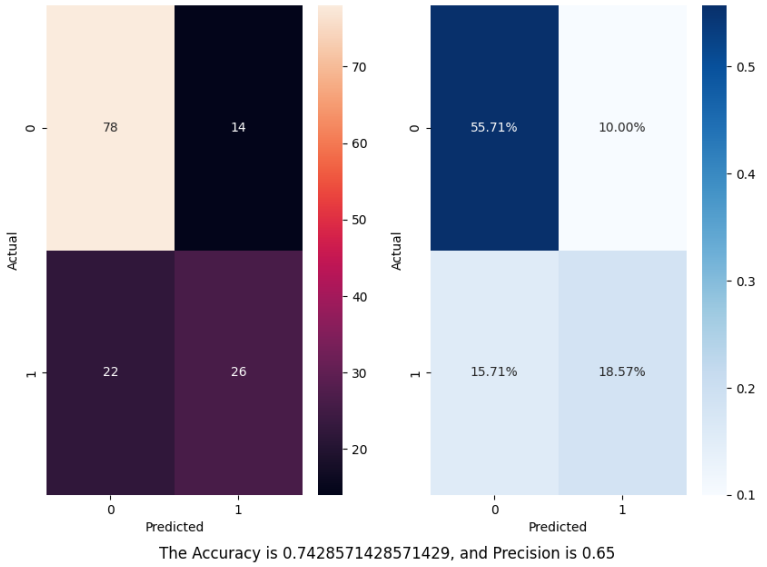


Figure 231: Graph: Breast AEDNN Tuning Training Confusion Matrix , Eta = 0.035

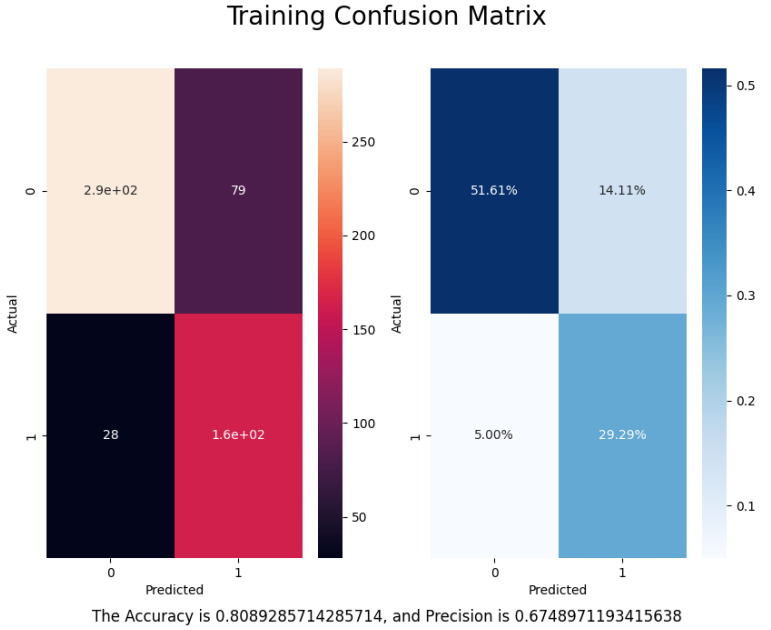
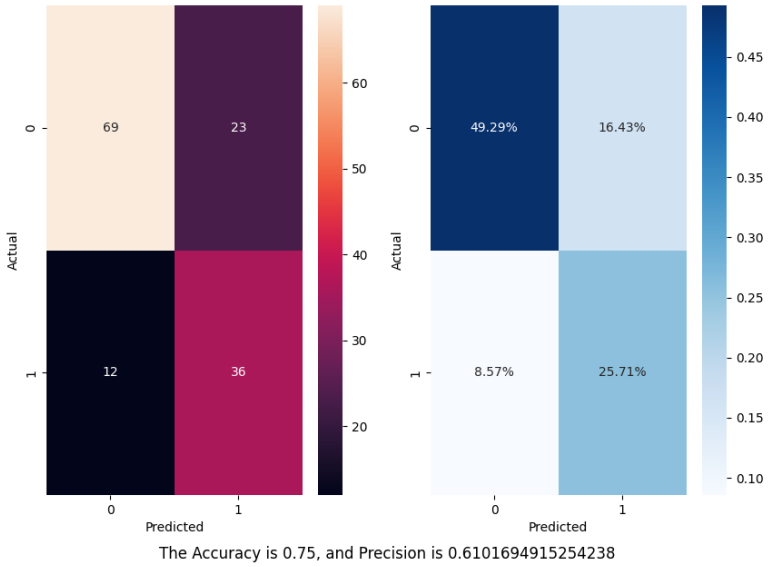


Figure 232: Graph: Breast AEDNN Tuning Testing Confusion Matrix , Eta = 0.035



3.15.2 Testing Results

ETA = 0.01 gave the best accuracy at 0.8214 in during tuning.

Figure 233: Graph: Breast AEDNN 80% Training Confusion Matrix , Eta = 0.01

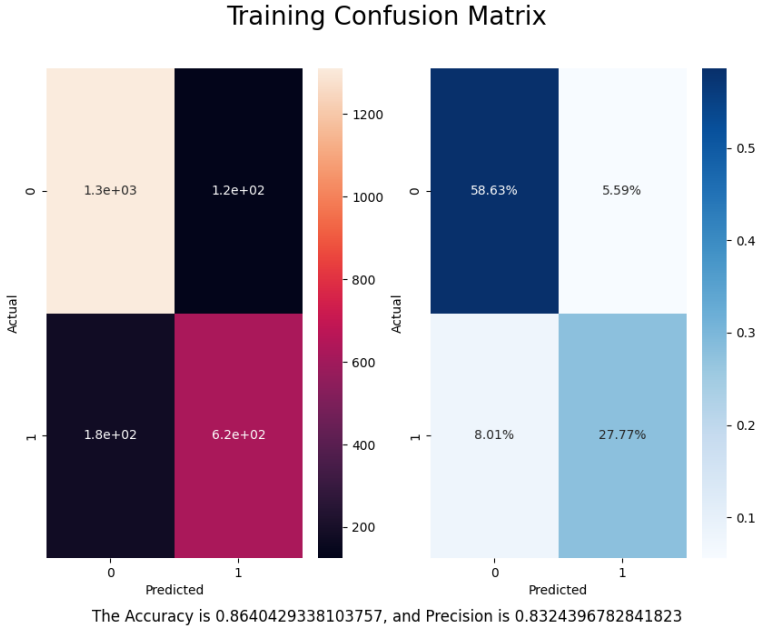
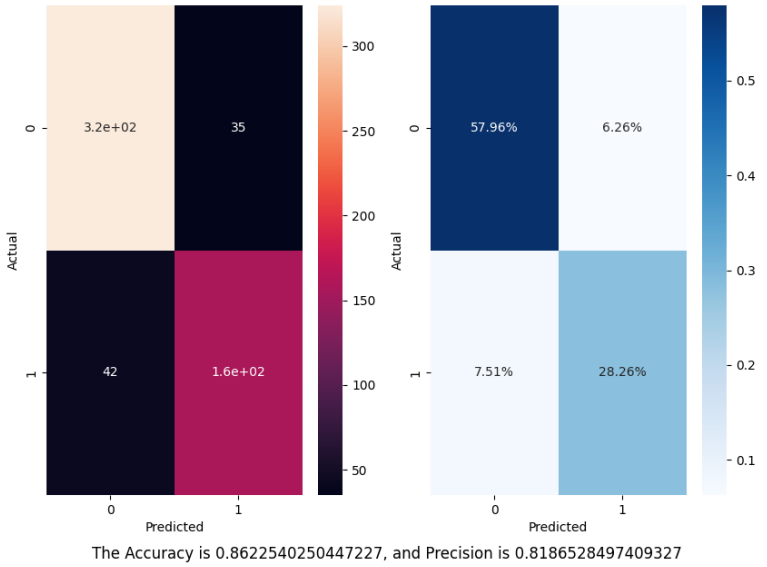


Figure 234: Graph: Breast AEDNN 80% Data Testing Confusion Matrix , Eta = 0.01



3.16 Abalone Regression via Autoencoder to DNN

3.16.1 Tuning Results

Figure 235: Graph: Abalone AEDNN Avg Tuning Performance For Cross Validation, Eta = 0.015

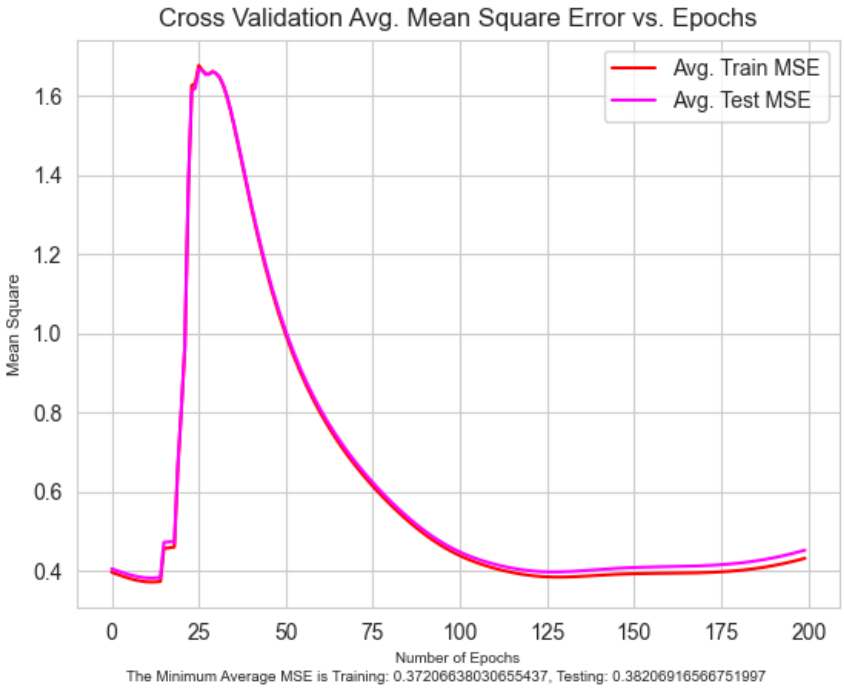


Figure 236: Graph: Abalone AEDNN Tuning yPred vs. yActual, Eta = 0.015

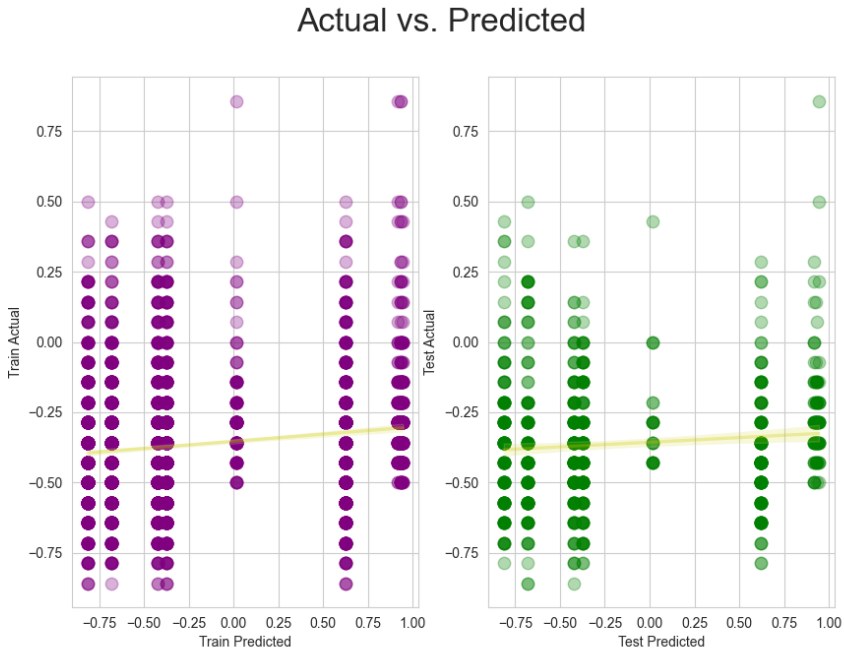


Figure 237: Graph: Abalone AEDNN Tuning Avg Performance For Cross Validation, Eta = 0.02

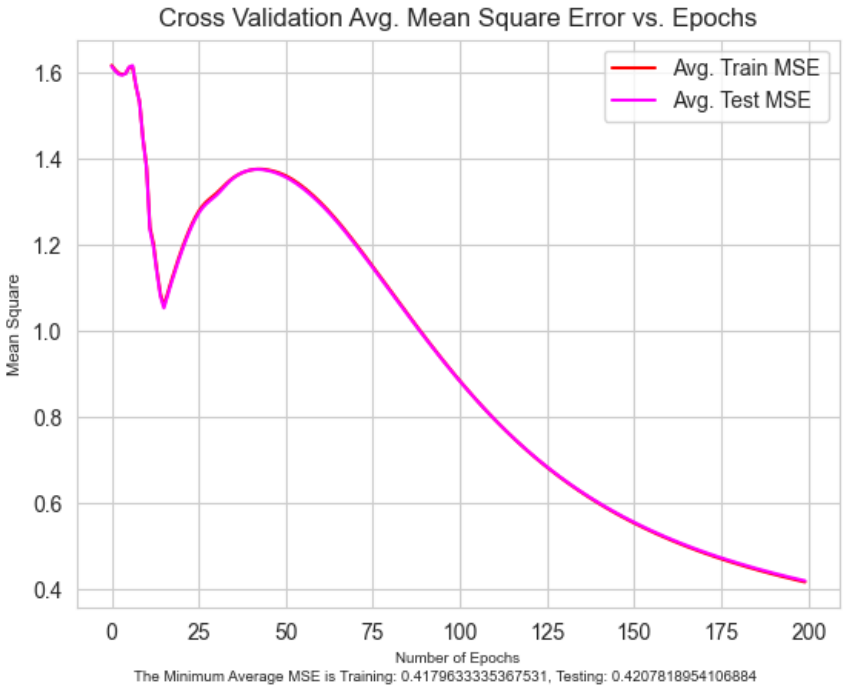
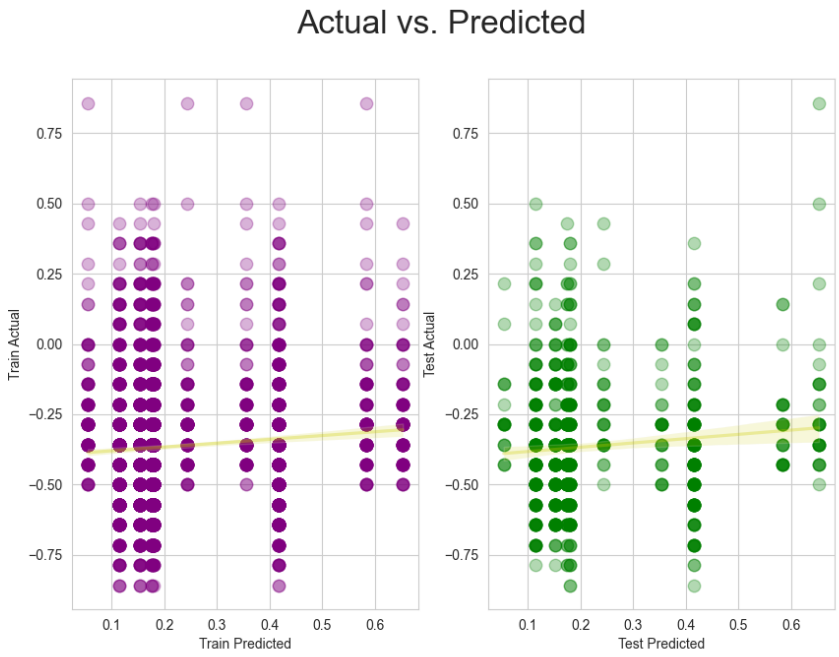


Figure 238: Graph: Abalone AEDNN Tuning yPred vs. yActual, Eta = 0.02



3.16.2 Testing Results

Eta = 0.015 gave the lowest MSE at MSE = 0.3820. Eta = [0.01, 0.03, 0.035] experienced a diminishing gradient, resulting in the task not being able to be performed.

Figure 239: Graph: Abalone AEDNN 80% Data Avg Performance For Cross Validation, Eta = 0.015

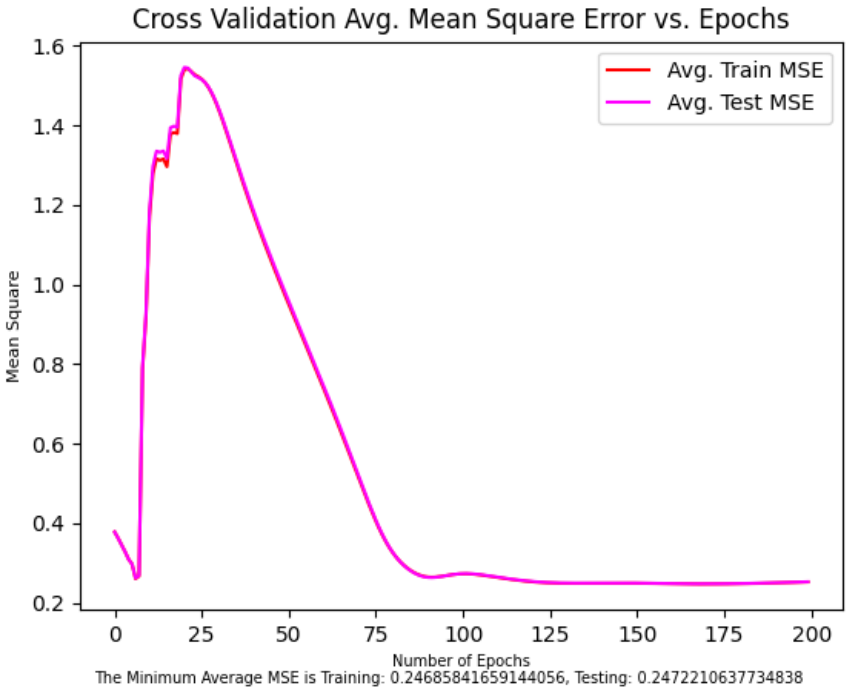
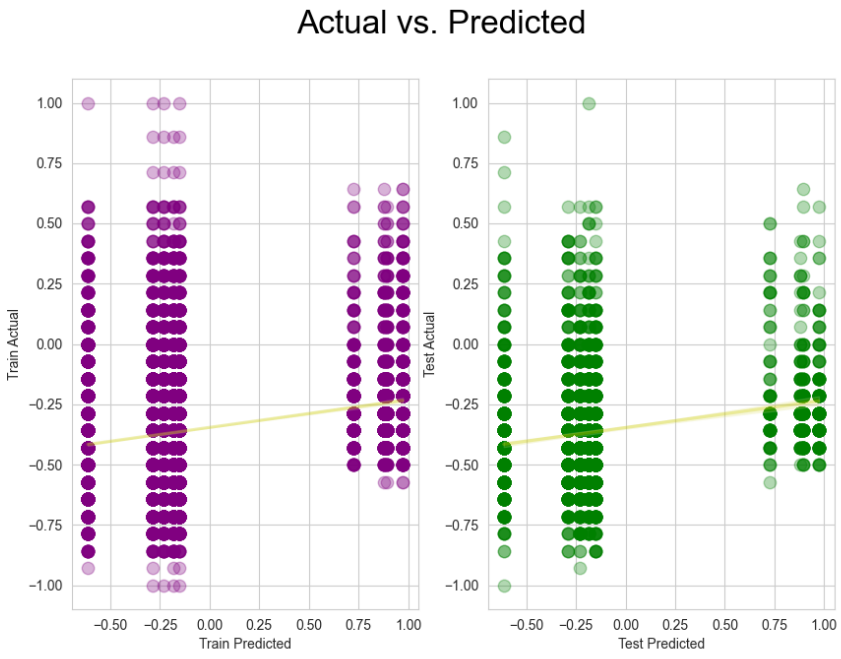


Figure 240: Graph: Abalone AEDNN 80% Data yPred vs. yActual, Eta = 0.015



3.17 Computer Regression via Autoencoder to DNN

3.17.1 Tuning Results

Figure 241: Graph: Computer AEDNN Tuning Avg Performance For Cross Validation, Eta = 0.01

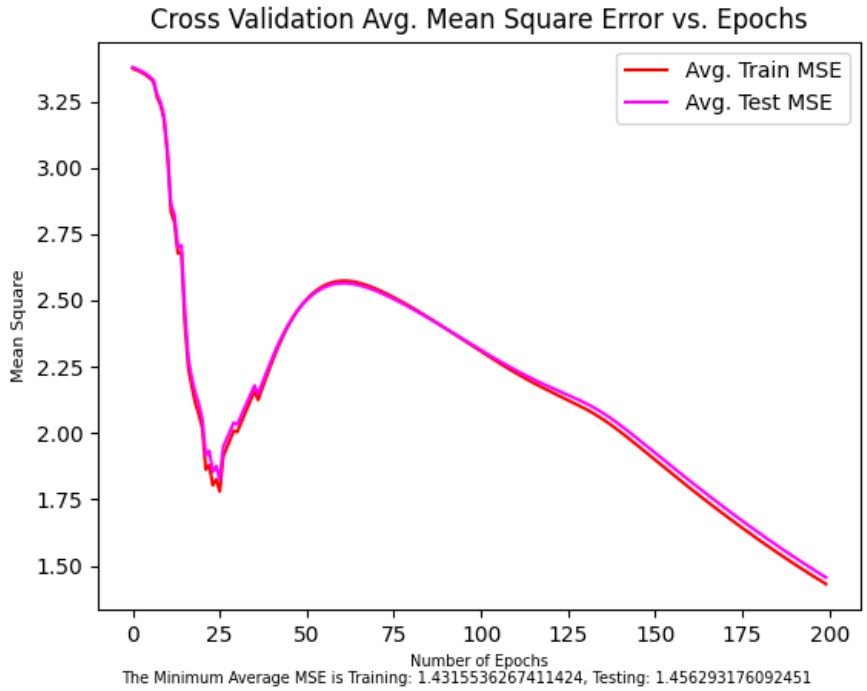


Figure 242: Graph: Computer AEDNN Tuning yPred vs. yActual, Eta = 0.01

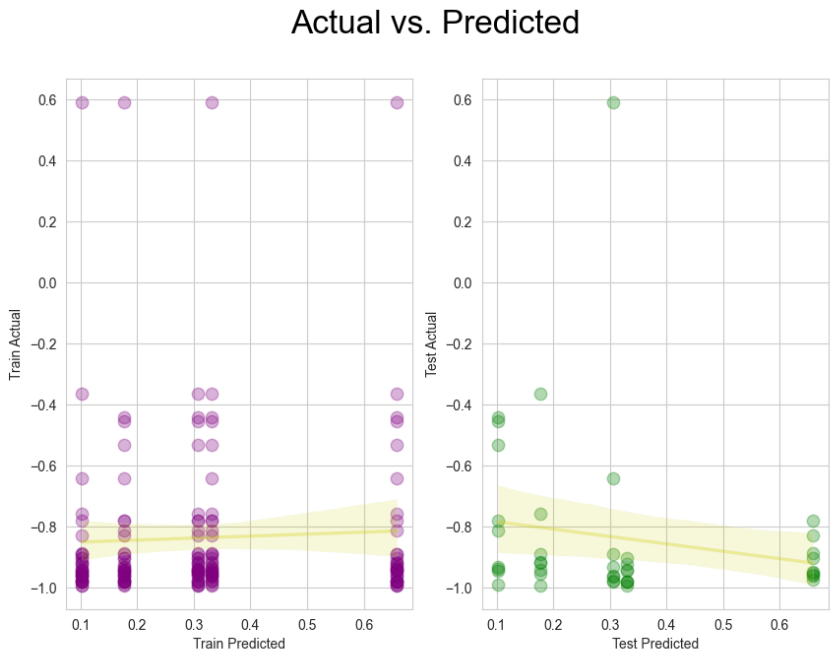


Figure 243: Graph: Computer AEDNN Avg Tuning Performance For Cross Validation, Eta = 0.0125

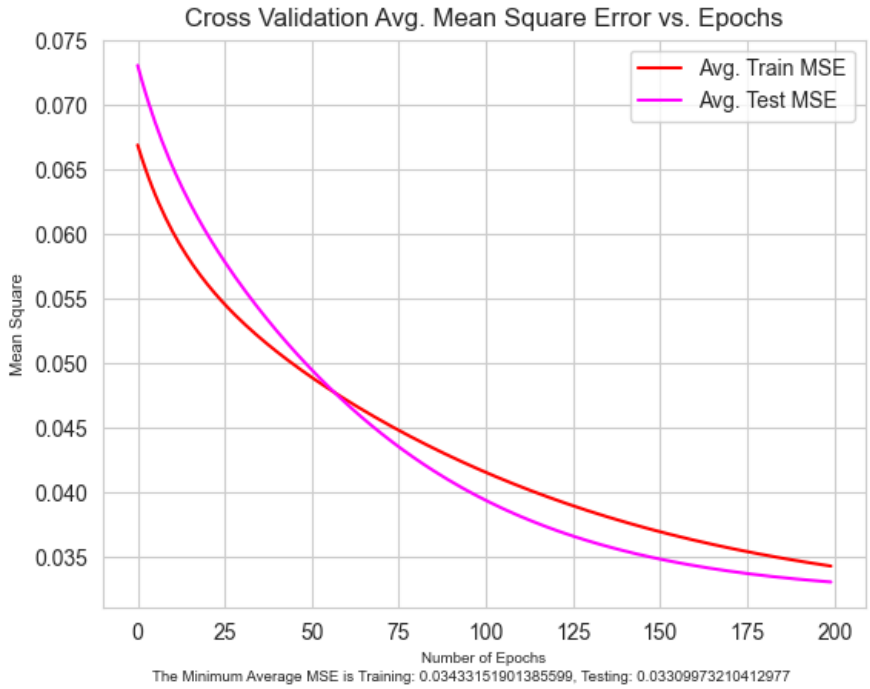


Figure 244: Graph: Computer AEDNN Tuning yPred vs. yActual, Eta = 0.0125

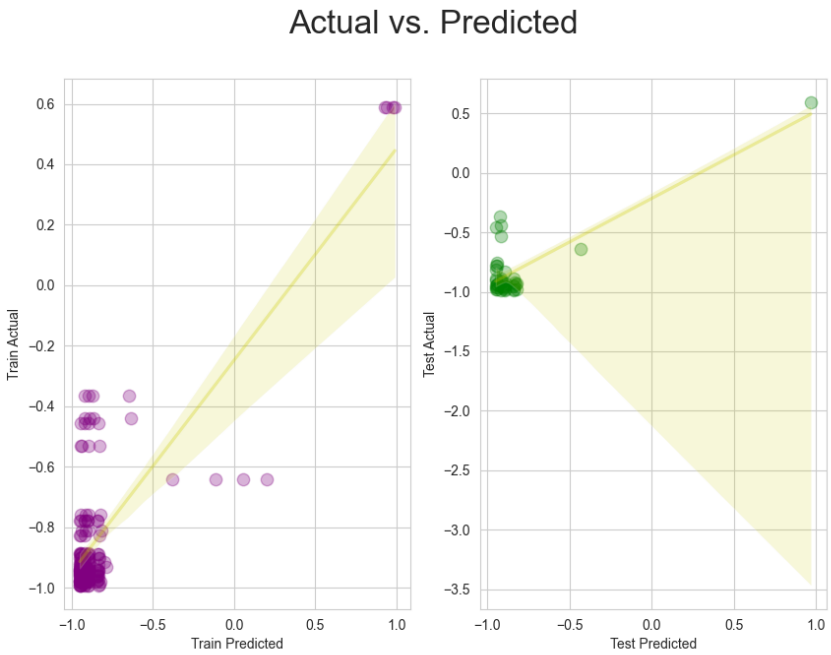


Figure 245: Graph: Computer AEDNN Tuning Avg Performance For Cross Validation, Eta = 0.015

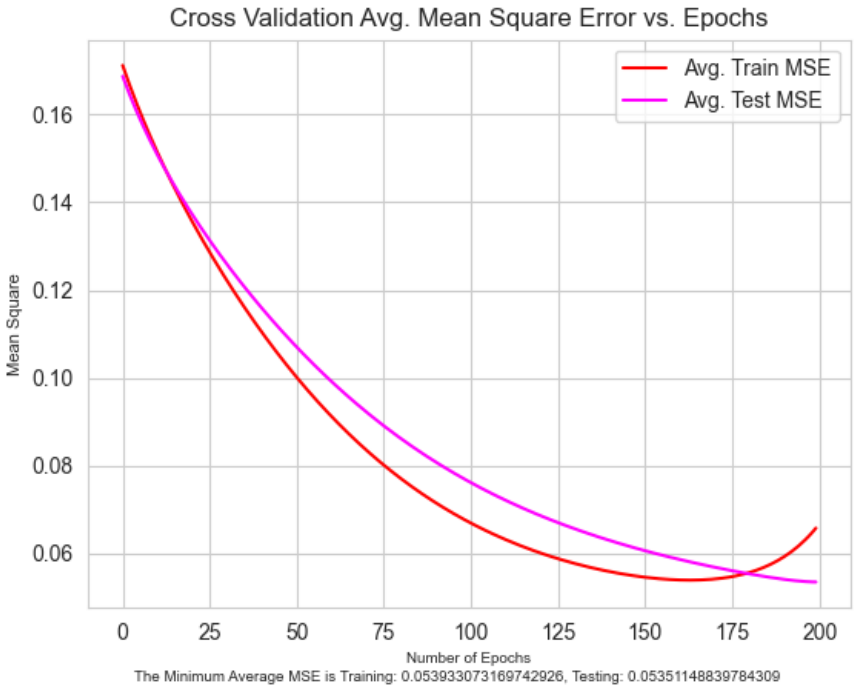


Figure 246: Graph: Computer AEDNN Tuning yPred vs. yActual, Eta = 0.015

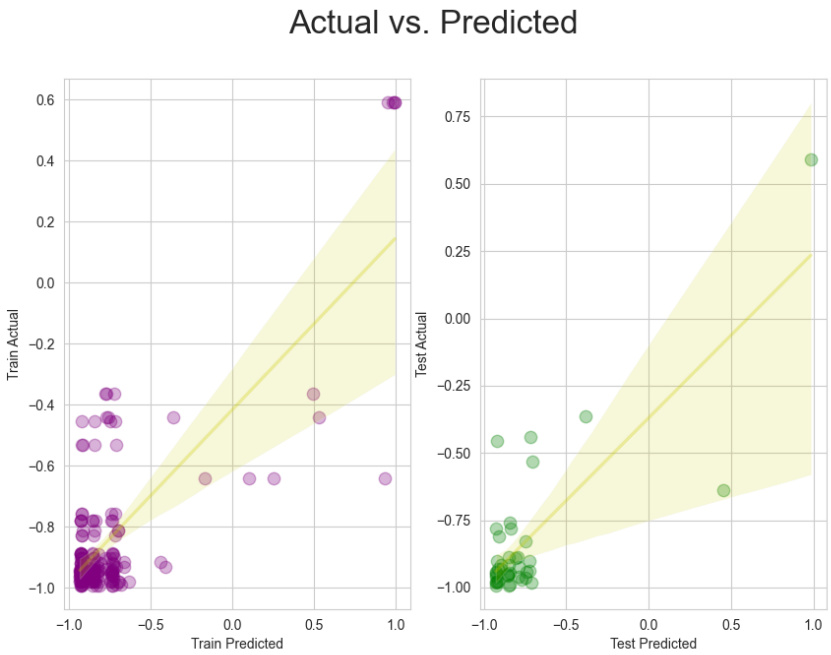


Figure 247: Graph: Computer AEDNN Tuning Avg Performance For Cross Validation, Eta = 0.0175

Figure 248: Graph: Computer AEDNN Tuning yPred vs. yActual, Eta = 0.0175

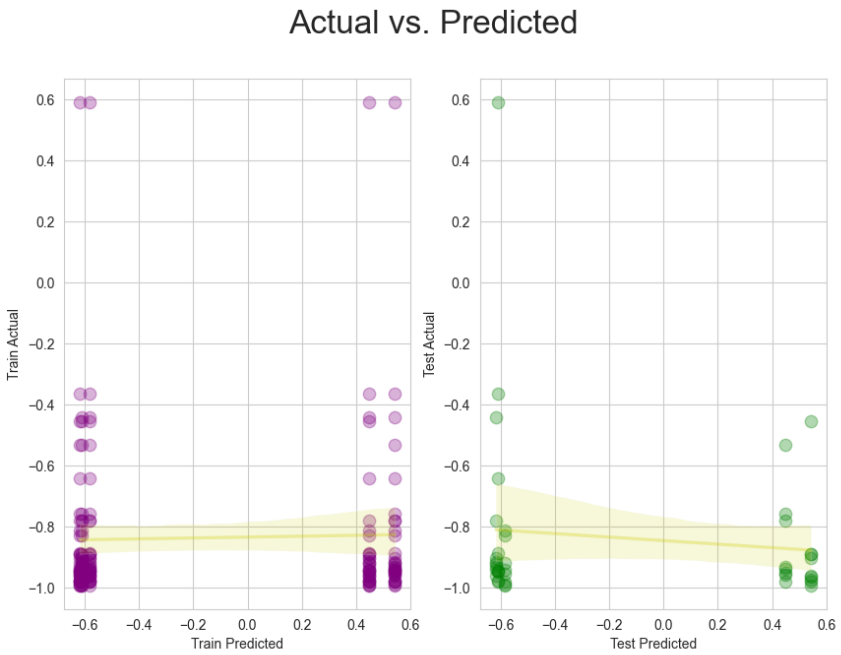


Figure 249: Graph: Computer AEDNN Tuning Avg Performance For Cross Validation, Eta = 0.02

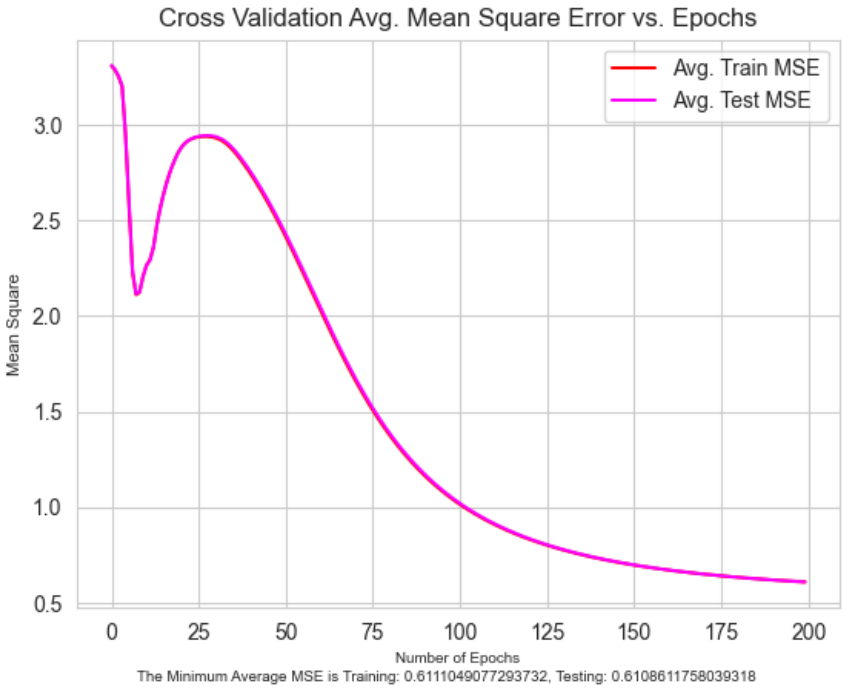
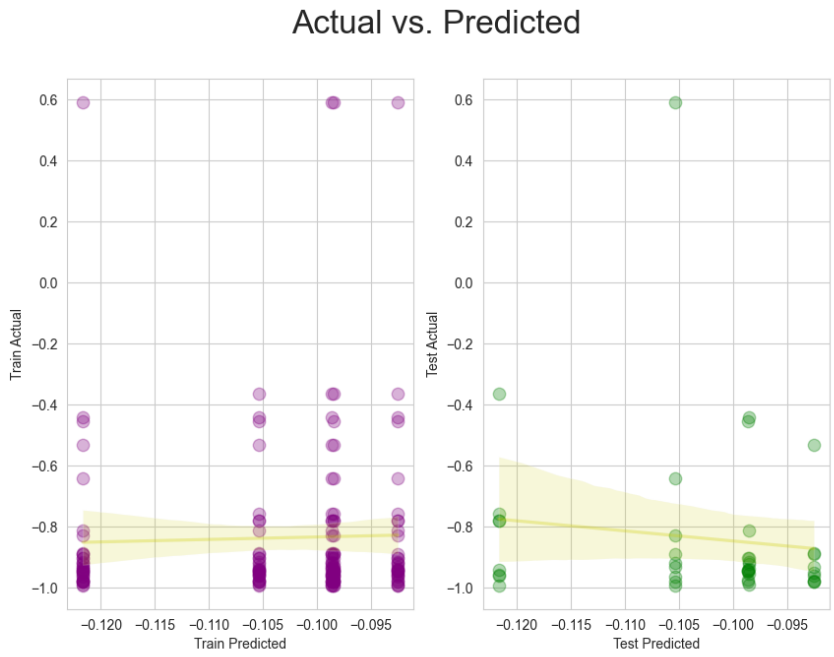


Figure 250: Graph: Computer AEDNN Tuning yPred vs. yActual, Eta = 0.02



3.17.2 Testing Results

ETA = 0.0125 resulted in MSE during tuning to be MSE = 0.0331.

Figure 251: Graph: Computer AEDNN 80% Data Avg Performance For Cross Validation, Eta = 0.0125

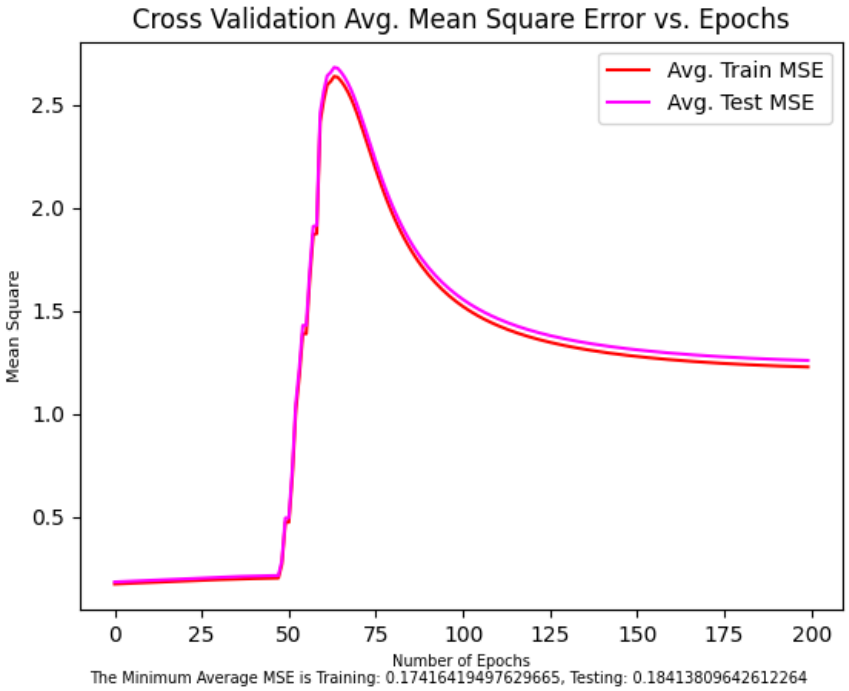
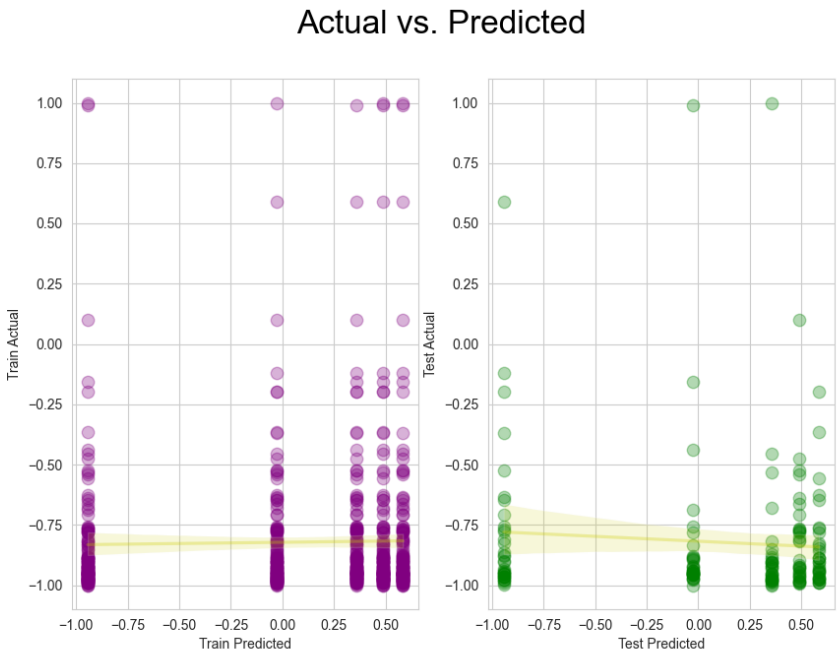


Figure 252: Graph: Computer AEDNN 80% Data yPred vs. yActual, Eta = 0.015



3.18 Forest Regression via Autoencoder to DNN

3.18.1 Tuning Results

Figure 253: Graph: Forest AEDNN Tuning Avg Performance For Cross Validation, Eta = 0.02

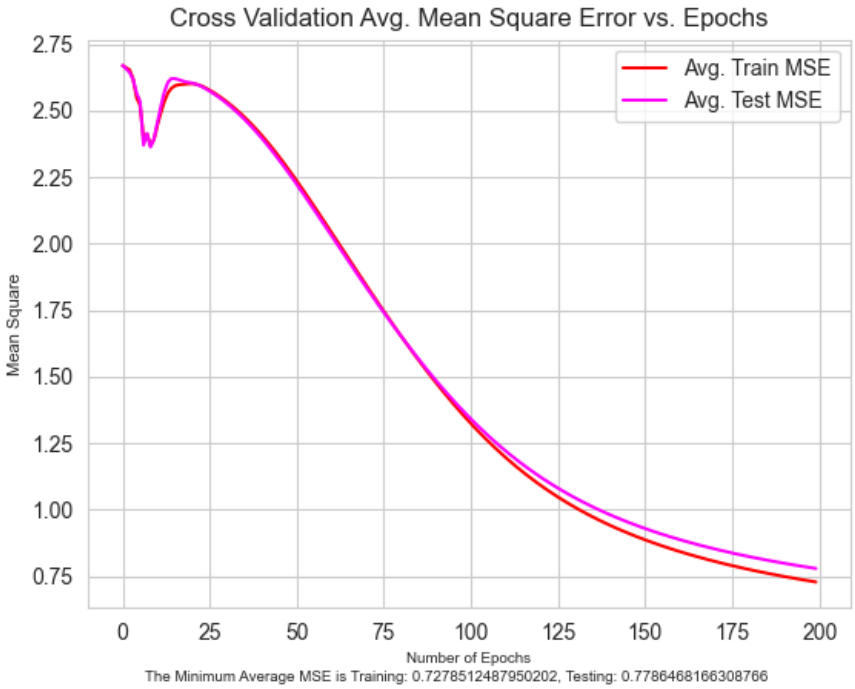
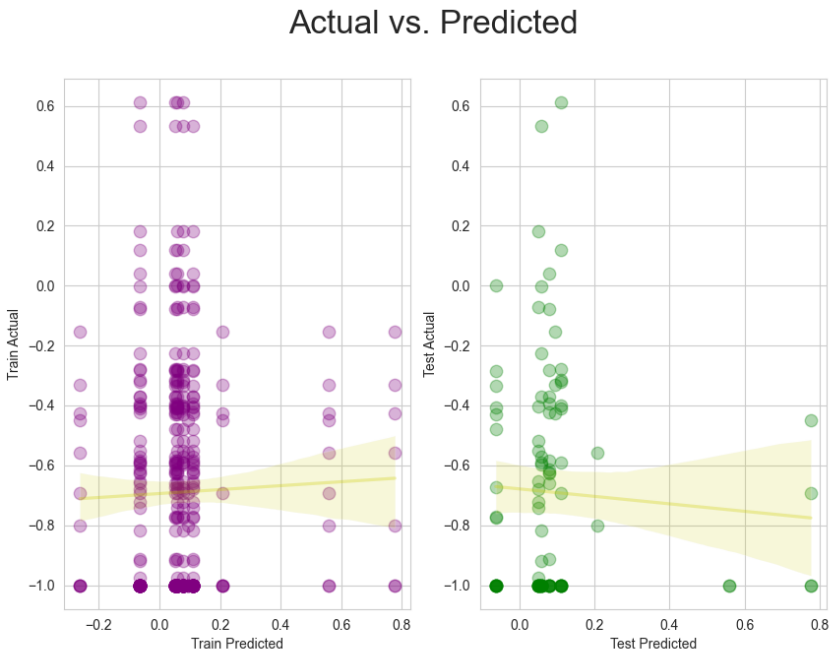


Figure 254: Graph: Forest AeDNN Tuning yPred vs. yActual, Eta = 0.02



3.18.2 Testing Results

Eta = 0.02 gave the lowest MSE. The other Etas = [0.01, 0.0125, 0.015, 0.0175] resulted in a vanishing gradient situation.

Figure 255: Graph: Forest AEDNN 80% Data Avg Performance For Cross Validation, Eta = 0.02)

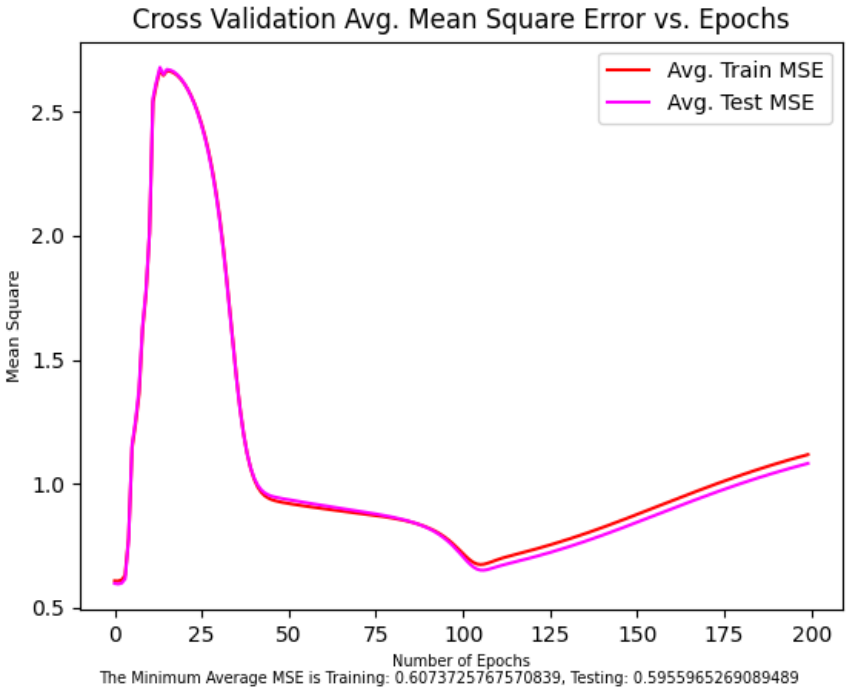
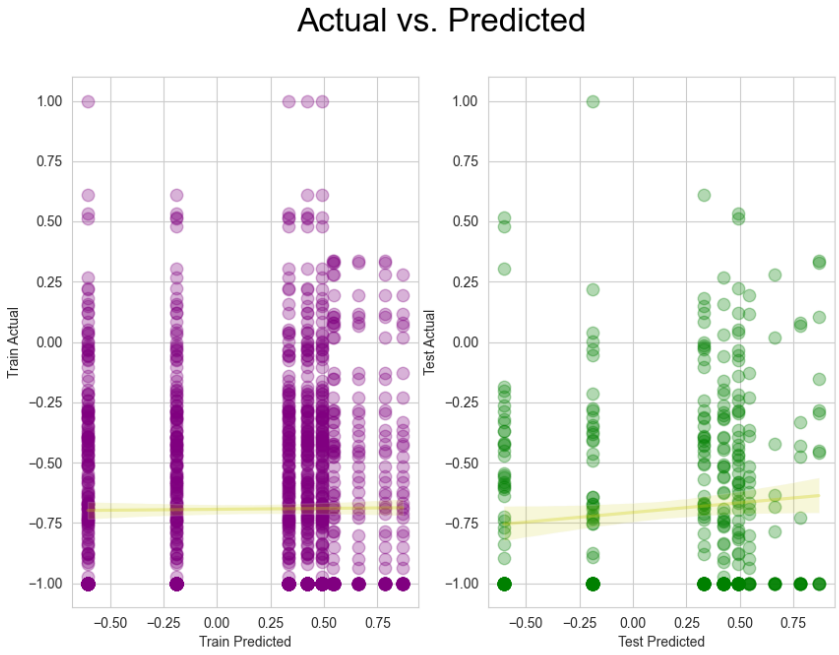


Figure 256: Graph: Forest AEDNN 80% Data yPred vs. yActual, Eta = 0.02



4. Discussion

4.1 Classification Tasks

4.1.1 Car Evaluation Data

Looking at the results from implementing the Linear Perceptron Classifier, Deep Neural Network and the Auto encoding network that then attaches to another network. For the Linear Perceptron network, after tuning the best *eta* hyper-parameter was $eta = 0.01$, having achieved 67.34% accuracy on the tuning data set. This *eta* value resulted in the the linear network achieving a 65.19% accuracy for the remaining 80% of the Car Evaluation Data.

In running the tuning car dataset on the Deep Neural Network, it was discovered that $eta = 0.03$ resulted in the tuning dataset achieving 69.56% accuracy. The $eta = 0.03$ was then used as the hyper parameter for the deep neural network with two hidden layers. After running a 5- fold cross validation, the accuracy achieved on the remaining 80% of the car dataset was 41.05 %.

Finally, when applying the auto encoding network as a form of data compression and then attaching it to another network it was found during tuning that the best *eta* hyper parameter was $eta = 0.015$ which resulted in 65.60 % accuracy. When using $eta = 0.015$ as the learning rate for the remaining 80% for the data the average accuracy from the 5-fold cross validation was 64.61%.

When looking at the *car.name* file, it is important to note that the class distribution among the class was skewed more to the unacceptable class, *unacc*. This skewed distribution is often attributed to under sampling. To reduce overfitting of the networks, $L2 - Regularization$ and $momentum$ were additionally used. In the cases theses three networks, the linear network and the auto encoding neural network performed really well and were capable of reducing the cross entropy loss. In regards to the standalone deep neural network, it appeared that it was not capable of lowering the cross entropy loss. It also appears that the network itself had issues in dealing with the one hot encoded output and calculating the appropriate cross entropy loss and gradients. This is a problem that is only carried across the other classification data sets for the standalone deep neural network.

4.1.2 House Voting Data

The house voting data underwent 5 fold cross validation tuning for $eta = [0.01, 0.015, 002, 0.03, 0.035]$. In running the tuning dataset through the Linear Perceptron Classifier for $epochs = 300$, $regStrength = 0.001$, and $momentum = 0.05$, the optimal hyper-parameter *eta*, was $eta = 0.03$ and 0.035 which resulted in 100% accuracy during tuning. In using that same *eta* value on the remaining 80% of the Voting data, the average accuracy achieved from a 5- fold Cross Validation was 97.13%.

In tuning the deep neural network for $epochs = 500$ and the same values of $regStrength$, and $momentum$, the best *eta* value was $eta = 0.02$, which resulted in 63.21% accuracy. Keep in mind this DNN appears to not handle the one hot encoded variables like the others and therefore the confusion matrix is messed up. In using $eta = 0.02$ on the remaining 80% of the data the accuracy achieved was 62.86%.

Lastly when running the Voting tuning data through the auto encoding neural network and the the same *eta* hyper-parameter values, the best *eta* value was $eta = 0.03$ having achieved an accuracy of 97.70% . Using the *eta* value found from tuning on the remaining 80% of data the average accuracy achieved during a 5-fold Cross Validation was 96.55%.

When comparing the distributions from the confusion matrices in the Linear Perceptron and the the Auto encoding neural network, the auto encoding neural networks seems to best guess the distribution of the actual Democrat and Republican distribution from the house-votes-84.names file, with the actual distribution being 54.2% Republican and 45.2% Democrat and the predicted distribution being 58.62% Republican and 37.92% Democrat, with around a 3.45% discrepancy from false positives and false negatives.

4.1.3 Breast Cancer Data

The breast data underwent 5 fold cross validation tuning for eta = [0.01, 0.015, 002, 0.03, 0.035]. In running the tuning dataset through the Linear Perceptron Classifier for *epochs* = 300, $regStrength = 0.001$, and $momentum = 0.05$, the optimal hyper-parameter *eta*, was $eta = 0.01$ and 0.0015 which resulted in same accuracy during tuning. The precision was higher when $eta = 0.01$ was used. In using that eta=0.01 on the remaining 80% of the breast data, the average accuracy achieved from a 5- fold Cross Validation was 85.69%.

In using tuning the deep neural network for *epochs* = 500 and the same values of $regStrength$, and $momentum$, the best *eta* value was $eta = 0.015$, which resulted in 62.5% accuracy. Keep in mind this DNN appears to not handle the one hot encoded variables like the others and therefore the confusion matrix is messed up. In using $eta = 0.015$ on the remaining 80% of the data the accuracy achieved was 64.28%.

Lastly when running the Breast tuning data through the auto encoding neural network and the the same *eta* hyper-parameter values, the best *eta* value was $eta = 0.01$ having achieved an accuracy of 82.14% during tuning . Using the $eta = 0.01$ found from tuning on the remaining 80% of data the average accuracy achieved during a 5-fold Cross Validation was 86.23%.

4.2 Regression Tasks

4.2.1 Abalone Data

For the linear network, after tuning the abalone data for 500 epochs, the *eta* that returned the lowest mse was $eta = 0.035$, having achieved an $mse = 0.02928$. When using $eta = 0.035$ on the remaining 80% of data, the mse achieved was $mse = 0.0262$.

For the Deep neural network the best *eta* was $eta = 0.035$ and it resulted in $mse = 0.1645$ for tuning. When that learning rate was applied to the remaining 80% of data the mse achieved was $mse = 0.24096$ which is significantly greater than during tuning.

For the Autoencoding neural network, the best *eta* hyper-parameter value was $eta = 0.015$ and it resulted in mse=0.3820. It is also important to note that eta = [0.01, 0.03, 0.035] resulted in the network experiencing the vanishing gradient problem. Due to this network experiencing the vanishing gradient problem it is possible that the weights were not updated adequately and thus led to poorer performance on unseen data. There could also be the possibility that the network was over-trained and overfitted to the training data which led to poor generalization on unseen data.

4.2.2 Computer Data

In tuning the linear network for the computer data, the best learning rate found was $eta = 0.0125$ which resulted in the tuning dataset achieving an $mse = 0.0075494$. When evaluating that learning rate on the remaining data, the mse achieved was $mse = 0.2502$. Which is significantly poorer than in the tuning data.

For the deep neural network, the best hyper-parameter eta, was $eta = 0.015$ and on the tuning dataset, it achieved $mse = 0.03509$. In using that eta =0.015 on the remaining data the mse achieved was $mse = 0.0582$ which is not terrible comparing it's performance on unseen data.

For the auto encoding neural network, the best eta found from tuning was $eta = 0.012$ and it resulted in $mse = 0.0331$ for the tuning data. On the remaining data, an mse of $mse = 0.18441$ was attained. The computer hardware did not experience the vanishing gradient problem as the abalone data, but on the auto encoding neural network it could be that information was lost in the compressed X inputs and led to a poorer performance on the unseen data.

4.2.3 Forest Fire Data

For the linear network the best eta value found was $eta = 0.02$ which gave an mse of $mse = 0.1486$ on the tuning data. In using that eta value on the remaining data, the mse was $mse = 0.2502$, which is significantly greater than on our training data, but not too significant to clearly say it is overfitted.

In the deep neural network, the best eta found from tuning was $eta = 0.0175$ and attained an mse of $mse = 0.4081$ on the tuning data. Using that eta value, on the remaining data an $mse = 0.5639$ was achieved which is greater than on the tuning, but it is expected to have a greater error on unseen data.

For the auto encoding neural network, the best eta found during tuning was eta=0.02 which gave the lowest mse, $mse = 0.7786$. This network however did experience the vanishing gradient problem on $eta = [0.01, 0.0125, 0.015, 0.0175]$ which resulted in network not being able to performed as expected. When using $eta = 0.02$ on the remaining data the lowest mse was $mse = 0.596$ which is lower than during tuning. Here it can be assumed that the model is underfitted as it performs poorly on the tuning data and also on the remaining 80% of data.

5. Conclusion

In the process of implementing three different kinds of neural networks, it was shown that while some linear and deep neural networks were able to perform well on the UCI data, there are challenges to overcome in terms of underfitting, overfitting and information loss during compression of input data. To overcome some of those challenges extra measures such as the He-et-al weight initialization, L2-weight decay regularization and momentum were included to help optimize the performance of the networks. Through this project it was shown that while linear neural networks may have relatively good performance, it could be due to overfitting the training data, thus preventing the unseen data to be evaluated correctly and preventing the model from being able to generalize. When looking at the

performance of the deep neural networks, it was shown that in raising the complexity of the network, it could improve the performance of it or decrease it depending on the data set. This is mainly evident in the performance of the forest data which overall has high *mse* results. Lastly when compressing the data and then feeding it to a classification or regression network the performance decreases through the increase in complexity and the information loss in compressing the X input data. This type of network also experience the effects of the vanishing gradient problem. While some extra measures were utilized to prevent this, in future studies, it would be best to examine alternative solutions such as the Xavier-et-al weight initialization, Batch normalization, further adjust the learning rates in the tuning process, gradient clipping, and changing the architecture of the neural network. This paper find that overall networks performed relatively well as a linear network, but through the increase of complexity in the deep neural network and the auto encoding neural network then performance fluctuates on the unseen data, as well as there a situations that lead to the vanishing gradient problem.