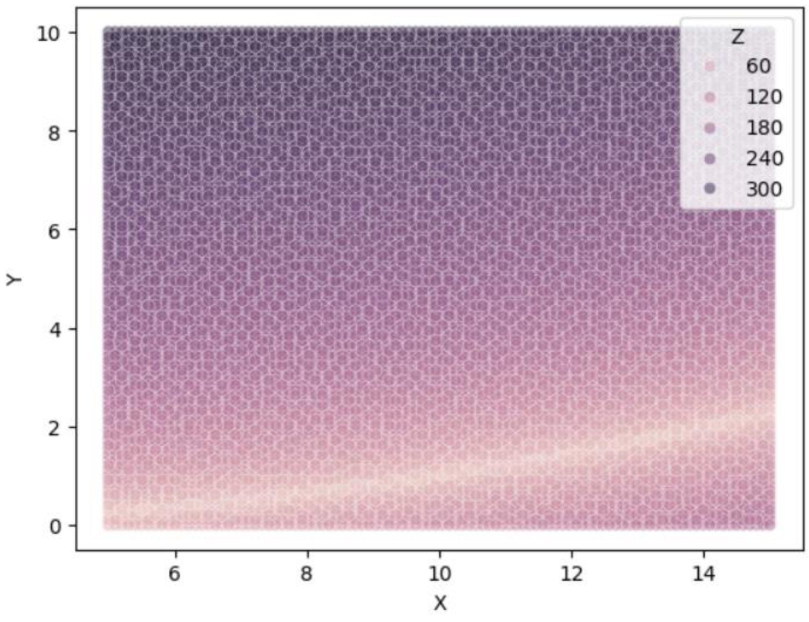
GENERATION OF AN ARTIFICIAL NEURAL NETWORK(ANN) MODEL

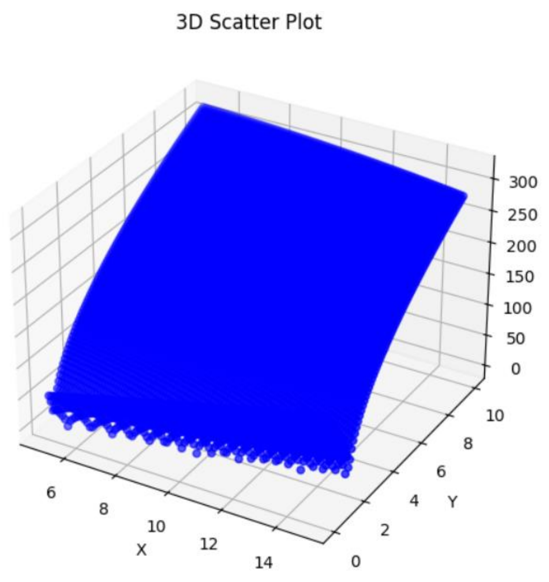
FUNCTION ALLOTED: BUKIN FUNCTION N.6

GENERATION OF THE RANDOM DATA: Random data was generated using the allotted Bukin Function N.6 in python using NumPy and the range was defined as X (5,15) and Y (0,10) and Z was calculated which is the output. In total 10000 data points were generated and through Pandas they were saved as Excel file.

READING THE DATA FROM THE EXCEL FILE: Data in the excel file was read by **Pandas.read_excel** by giving the location of the file. Descriptive statistics were calculated for the generated dataset and the distribution of the data points were plotted.

	X	Y	Z
count	10000.00000	10000.00000	10000.000000
mean	10.00000	5.00000	186.449699
std	2.91591	2.91591	77.967372
min	5.00000	0.00000	0.660101
25%	7.50000	2.50000	122.948932
50%	10.00000	5.00000	198.081263
75%	12.50000	7.50000	253.989705
max	15.00000	10.00000	312.399900





NORMALIZING THE DATA: Input and output data points were normalized to scale the feature in the dataset to a specific range, typically between 0 and 1. Normalization was done by **Minmax Scaler**. Normalization is crucial while training the data with ANN because Fast convergence, Improved generalization, Numerical stability and Smoother optimization landscape.

SPLIT THE DATA INTO TRAIN, VALIDATION AND TEST:

Splitting the data into training, validation and test sets is a fundamental practice in machine learning to evaluate and tune the models.

Training Data: This is the largest portion of dataset and is used to train machine learning model. During training, the model learns the underlying patterns and relationships in the data. It's used for updating the models' parameters such as weights and biases through optimization process.

Validation Data: The validation set is used during the model training process to tune hyperparameters and assess the model's performance on data it has never seen before. By monitoring the model's performance on validation set, one can adjust the hyperparameters and prevent overfitting.

Test Data: The test set is kept separate from both the training and validation sets. It's used to evaluate the model's performance once you've finished training and hyperparameter tuning.

It's a general practice to divide the dataset into Train, validation and test dataset in the proportion (70: 15: 15) and the same has been adopted in the current model.

GENERATING THE ANN MODEL:

- To generate an ANN model, Model must define using TensorFlow(keras).
- The architecture of the model must be defined. This includes specifying the number of layers, the number of neurons in each layer, activation function etc.,
- In the model **Input layer contains one layer**; Two hidden layers were defined with **First hidden layer with 12 nodes**, **Second hidden layer with 8 nodes**.
- For the hidden layers **relu activation function** was defined.
- Output layer contains only one node, and the activation function was defined to **linear.**
- After defining the model, it must be compiled with appropriate optimizer and metrics also known as Loss function.
- In the model, **Adam optimizer** was used and Metrics being **Mean squared error.**
- Models should be trained with the train data that was divided before and validated with the Validation data.
- While training the model, Epochs must be defined which implies the number of times the entire training dataset is presented to the neural network for learning, in this case, The number of epochs was set to 100.

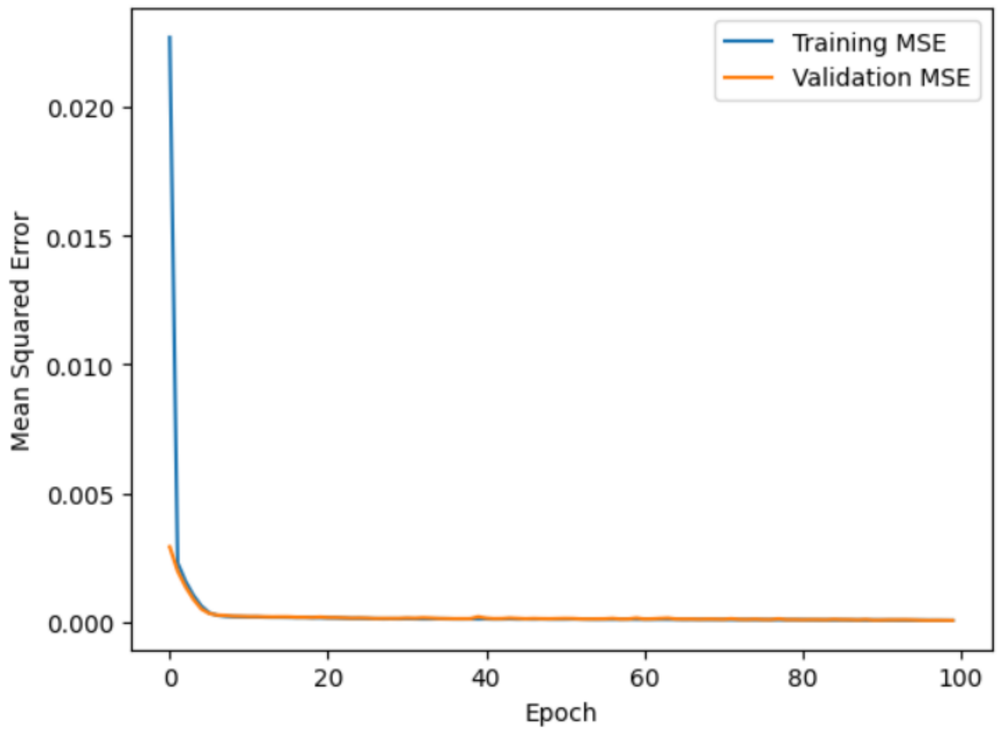
- After training the model, the model was evaluated with validation data and then with test data.

MODEL METRIC VALUES:

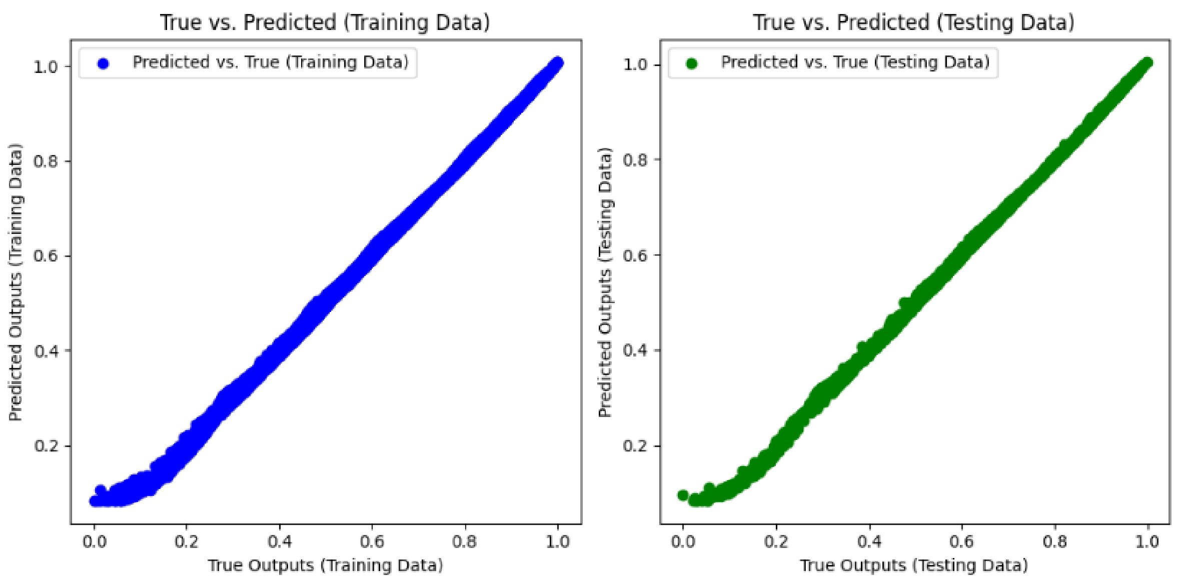
At first, the model was optimized with **Adam optimizer** and metric values obtained as follows:

	Mean Squared error (MSE)	**R-Squared error**
Training data	0.000063	0.998985
Test data	0.000066	0.998959

The following graph depicts the changes in Mean squared error of training and validation data upon multiple epochs.



The predicted output v/s True output in the testing data as well as training data depicts how well the model has been trained and how accurately is predicting the output.



In the above graph, we can see a diagonal line with the slope almost equal to 1 summarizes that model has been trained well and making the accurate predictions in test data as well. The R-squared value is also very close to 1 which supports the result.

The model was then trained to keep all the parameters same except the optimizer which is **RMSProp** instead of **Adam** to determine which works best for this specific use case. both Adam and RMSprop are popular optimization algorithms for training neural networks, and their effectiveness can depend on the specific problem and the choice of hyperparameters. While Adam combines elements of momentum and RMSprop to offer more adaptability in adjusting learning rates, RMSprop focuses solely on the magnitude of gradients.

Network architecture = [1, 2, 1]

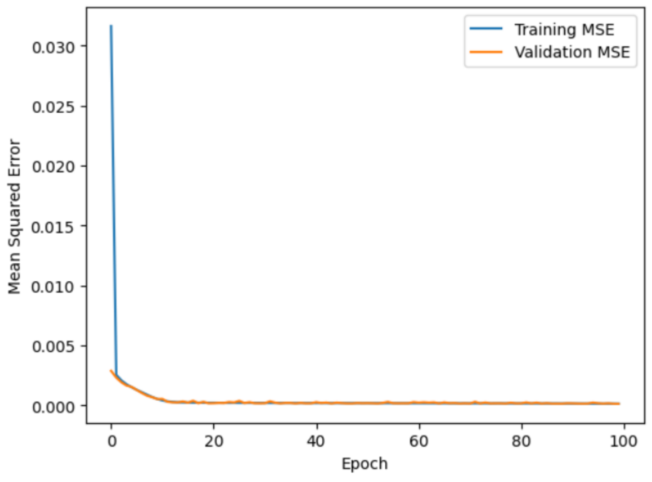
Nodes = 12 in 1st Hidden layer, 8 in 2nd Hidden layer (Same as previous one)

Activation function = relu, Epochs =100

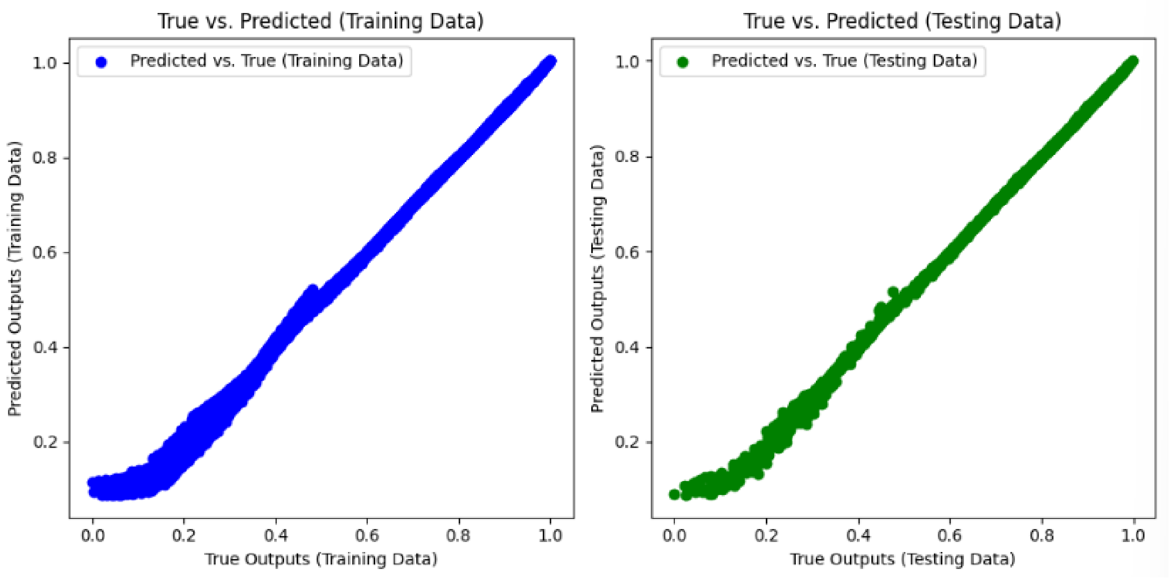
Optimizer = **RMSprop**

Metric values for RMSProp optimizer:

	Mean Squared error (MSE)	**R-Squared error**
Training data	0.000112	0.998191
Test data	0.000108	0.998283



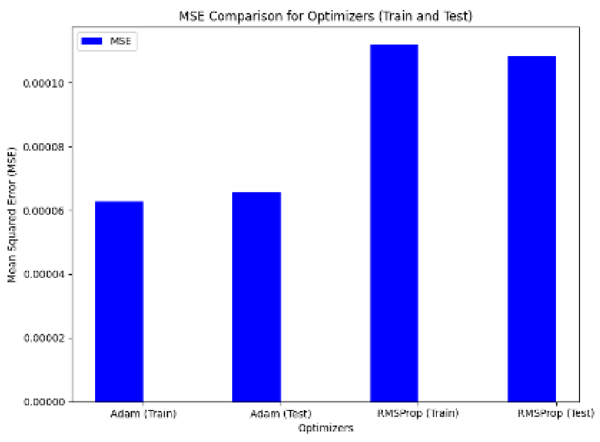
Plot for True v/s Predicted output for Training data and training data:

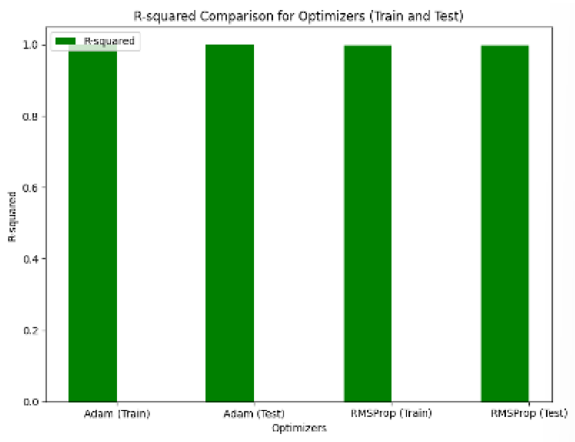


Here also, as observed, Model has been trained well and the predicted outputs are very close to True outputs in both Training and Testing data which is again supported by MSE value and R-squared values.

Comparatively between Adam and RMSProp, Metric values obtained are as follows:

	Adam		**RMSProp**	
	MSE	R-squared	MSE	R-squared
Training data	0.000063	0.998985	0.000112	0.998191
Testing data	0.000066	0.998959	0.000108	0.998283





As we can observe clearly from the above graph, MSE values of Adam is very less than RMSProp and R-squared values in each case are almost similar with negligible differences. This can be concluded as:

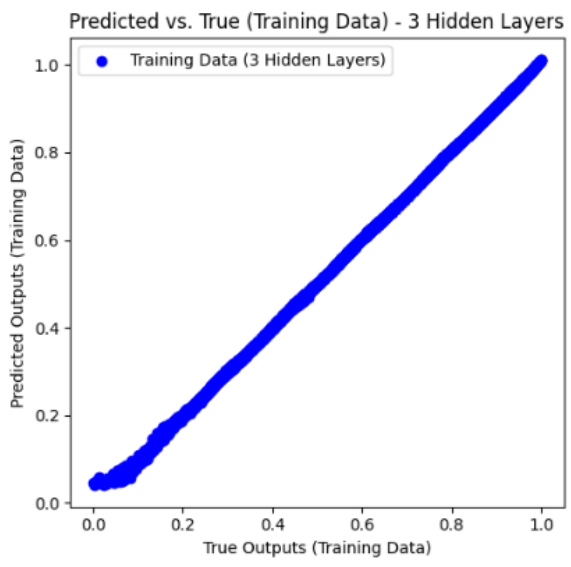
- **Adam achieves better performance than RMSProp**: Lower MSE values indicate that Adam produced a model that accurately predicts both training and Testing data.
- It is possible that Adam optimizer led to faster convergence.
- It is also possible that the weights initialized for the Adam-trained model were more favorable, leading to better results.

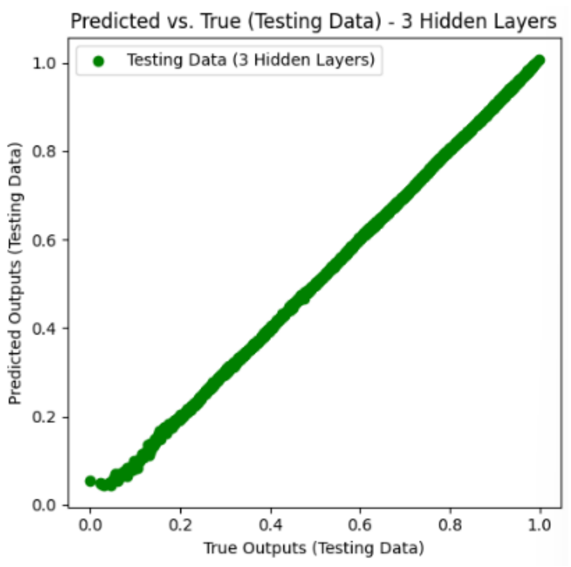
ANALYSIS DONE BY VARYING NUMBER OF HIDDEN LAYERS:

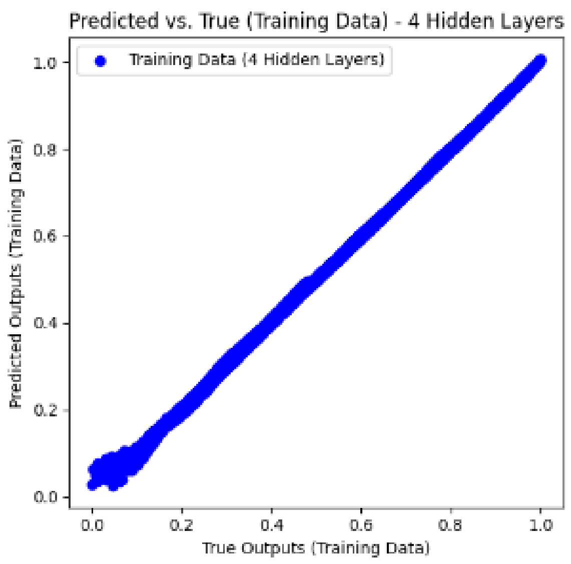
PARAMETERS:

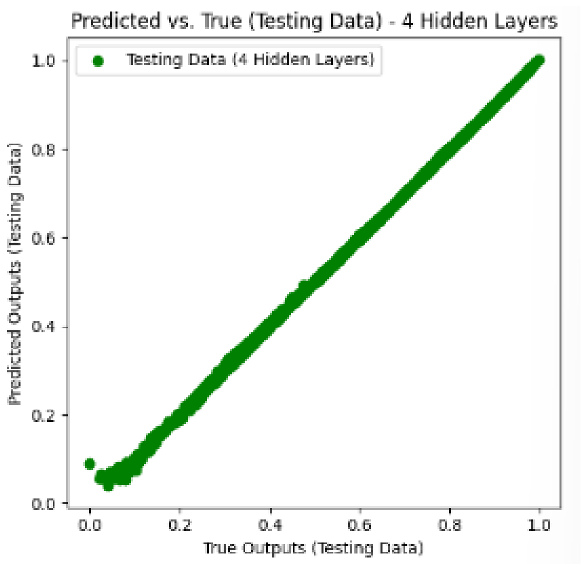
Different No. of Hidden layers	**3,4,5**
Activation function	Relu
Optimizer	Adam
Number of Nodes in each hidden layer	12
Epochs	100

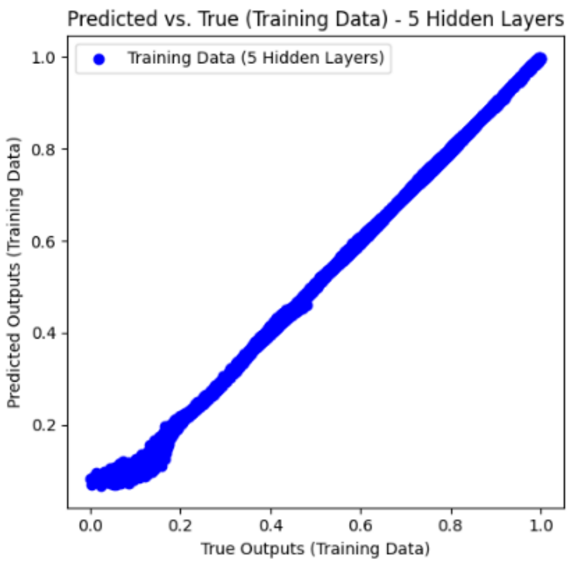
No. Hidden layers	**Training**		**Testing**	
	MSE	**R-Squared**	**MSE**	**R-Squared**
3	1.587234	0.99974	1.675426	0.9997347
4	2.274091	0.999632	2.568226	0.999593
5	4.106707	0.999335	4.10670	0.999320

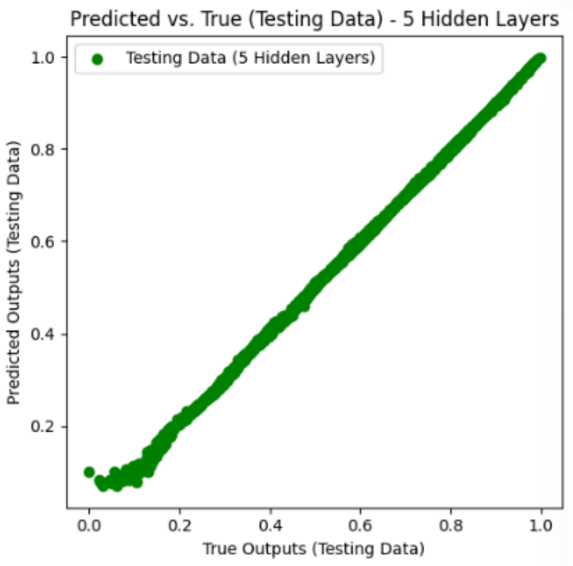






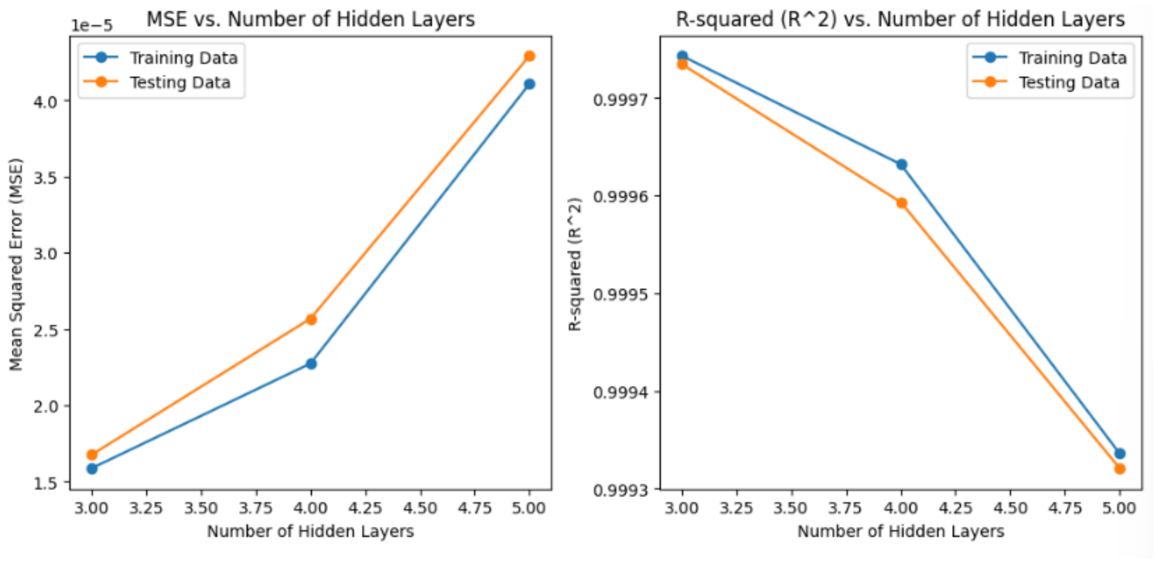






From the above graphs it can be concluded that the model has performed well in every case i.e., Hidden layers 3,4 and 5.

However, Detailed analysis found that there was a little variation in the MSE upon changing the number of Hidden layers.



Increase in MSE upon increasing the number of Hidden layers has been observed. This might be due to:

- Overfitting: Model becomes too complex for the amount of data available.
- Model complexity: **Adding more hidden layers increases the model's capacity to fit the training data**.
- Increasing the depth of a neural network can make it more susceptible to vanishing gradients or exploding gradients.

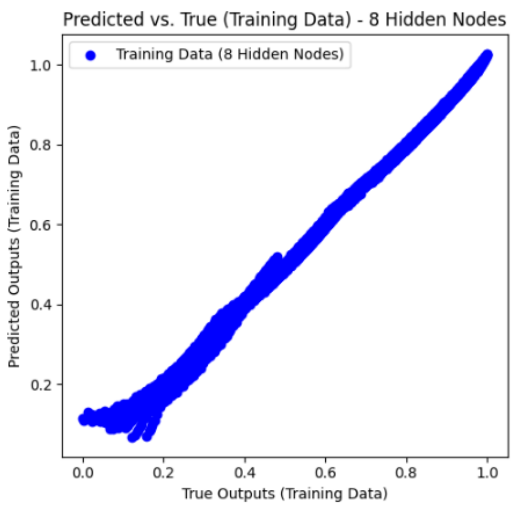
Reducing the number of hidden layers or applying regularization techniques might help mitigate the overfitting issue and improve model performance.

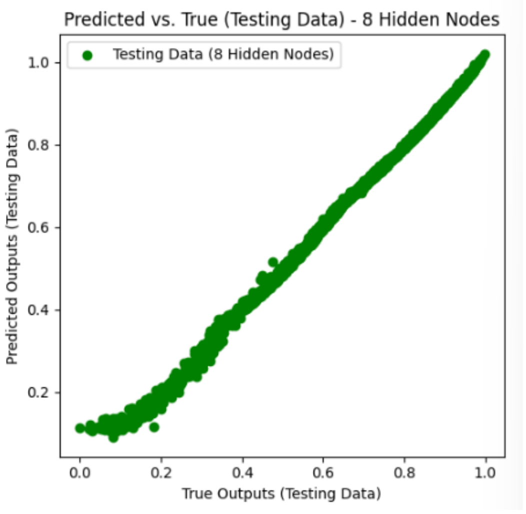
ANALYSIS DONE BY VARYING NUMBER OF HIDDEN NODES:

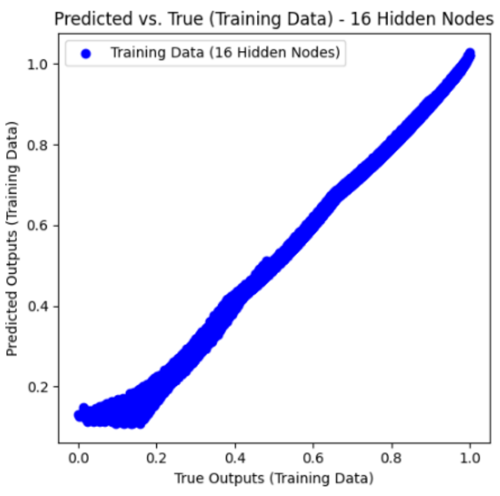
Different No. of Nodes	**8, 16, 32**
Activation function	Relu
Optimizer	Adam
No. of Hidden layer	1
Epochs	100

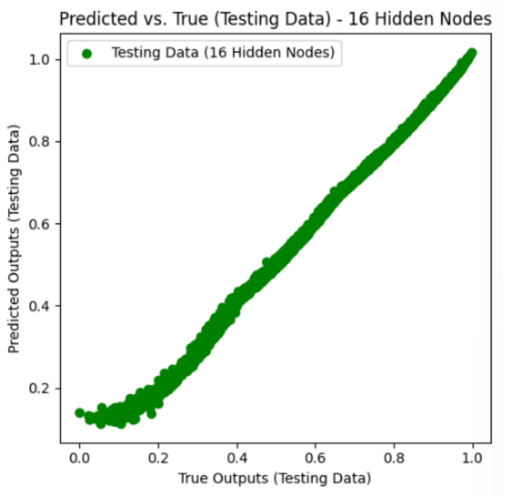
METRIC VALUES:

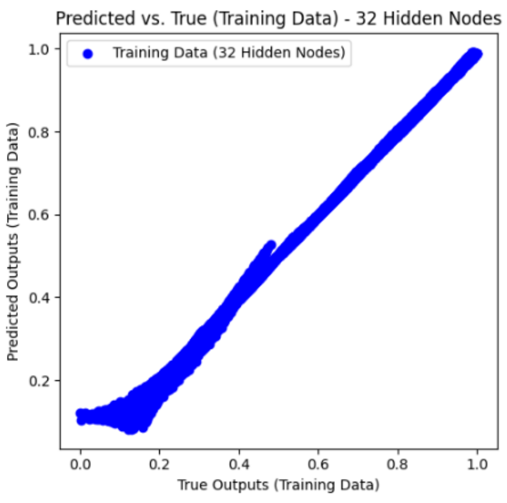
No. nodes in Hidden layer	**Training**		**Testing**	
	MSE	**R-Squared**	**MSE**	**R-Squared**
8	0.000191	0.996896	0.000184	0.997074
16	0.000221	0.996416	0.000221	0.996496
32	0.000130	0.997892	0.000126	0.997999

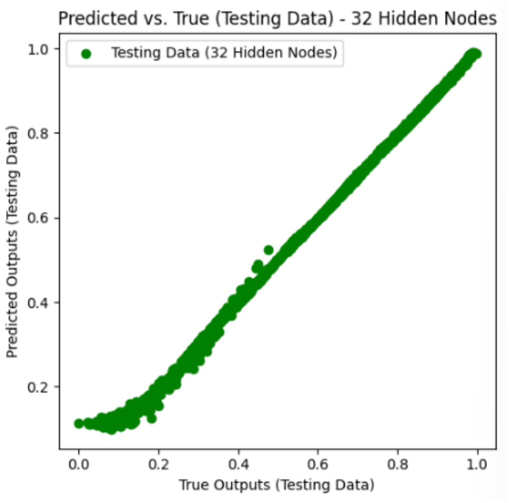




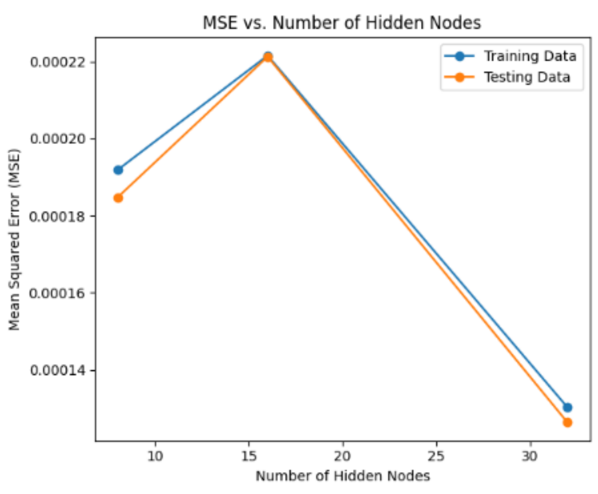


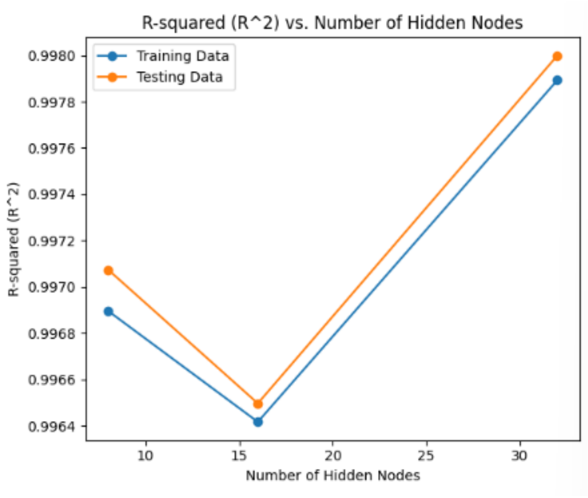






The predicted output in each case is almost comparable to the True output and the model is performing well. However, a close outlook at the MSE values is needed for drawing further conclusions.





Here we can observe that MSE increased with increasing the number of nodes to 16 but decreased with 32 nodes compared to 8 nodes. The possible reasons could be that:

Overfitting: **The initial increase in MSE with 16 nodes may suggest that the model became more complex and started to overfit the training data**. This overfitting led to a worse performance on the testing data. With 32 nodes, the additional complexity might have helped the model generalize better, leading to a decreased MSE.

Data Complexity: The complexity of the dataset can influence the number of nodes needed in the hidden layers. If the data is intricate and contains non-linear relationships, a higher number of nodes might be required to capture these relationships effectively.

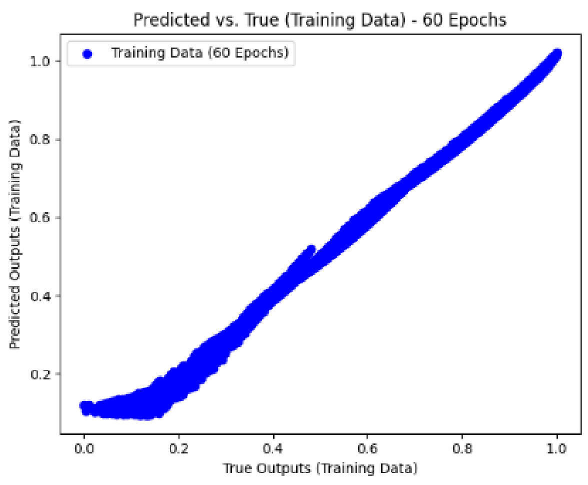
Increasing the number of hidden nodes from 8 to 32 nodes was beneficial for reducing the MSE. This suggests that a larger capacity was needed for the problem.

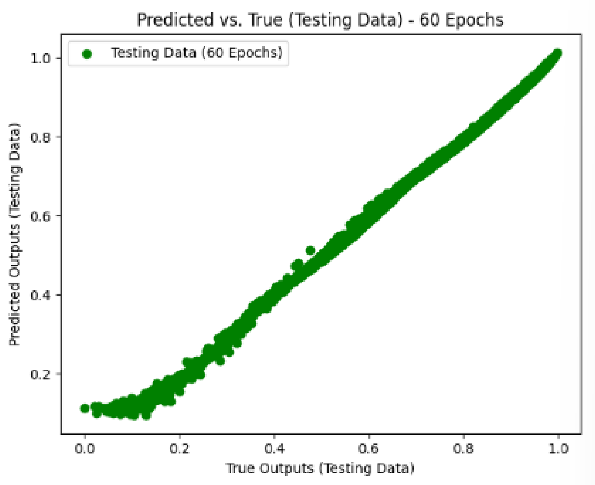
ANALYSIS DONE BY DIFFERENT NUMBER OF EPOCHS:

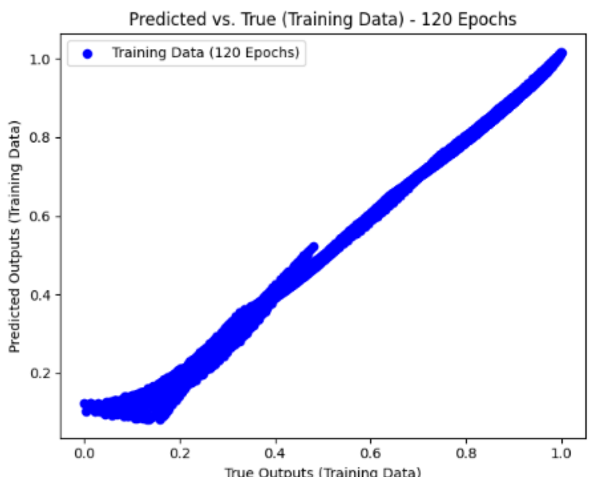
Different number of Epochs	**60,120,180**
Activation function	Relu
Optimizer	Adam
No. of Hidden layer	1
No. of Nodes in Hidden layer	16

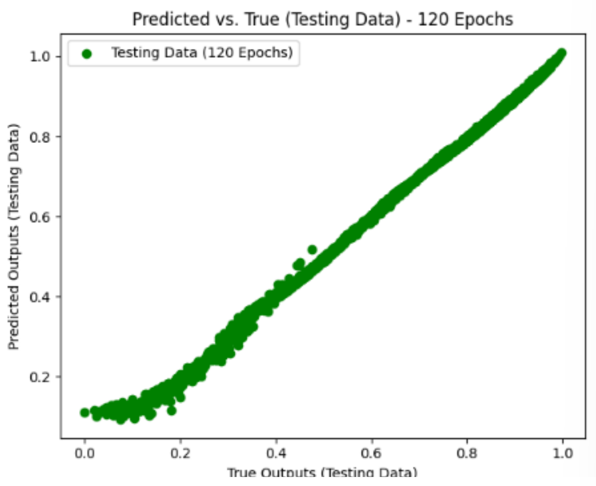
METRICS:

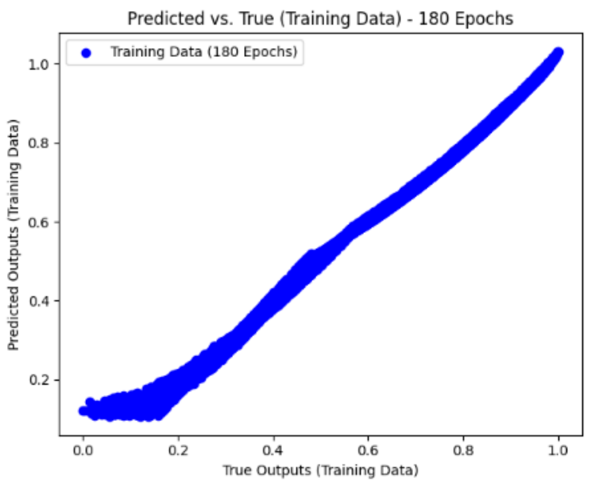
No. of Epochs	**Training**		**Testing**	
	MSE	**R-Squared**	**MSE**	**R-Squared**
60	0.000184	0.99700	0.000177	0.997184
120	0.000166	0.99730	0.000160	0.997466
180	0.000282	0.995435	0.000278	0.995598

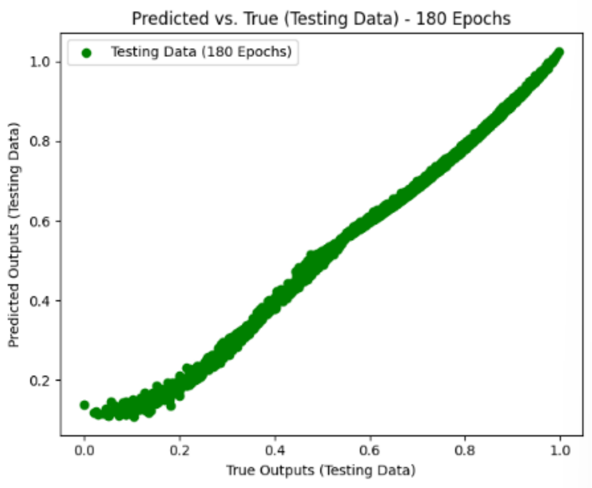




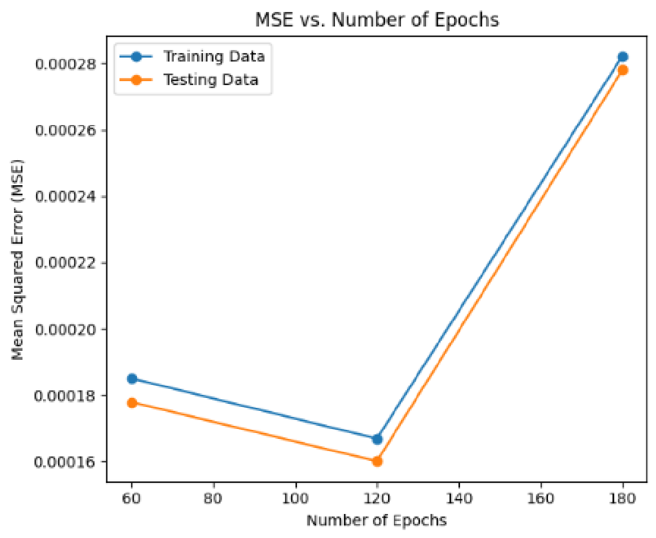


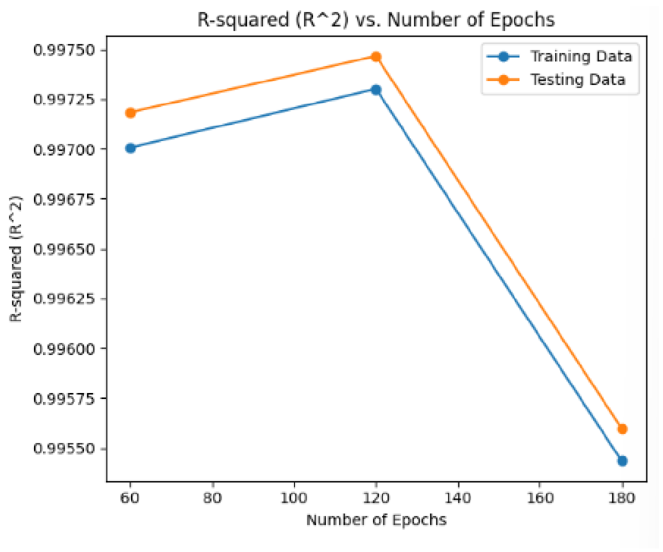






As observed in other cases, The True outputs are comparable to predicted outputs on changing the Epochs. Furthermore, Change in MSE is as follows,





Increasing the number of training epochs from 60 to 120 leads to a decrease in Mean Squared Error (MSE), but further increasing it to 180 results in a significant increase in MSE, it indicates the following :

Early Convergence: The initial **increase in training epochs from 60 to 120 helped the model improve its performance**. This suggests that the model hadn't fully converged within 60 epochs, and it continued to make better predictions as training continued.

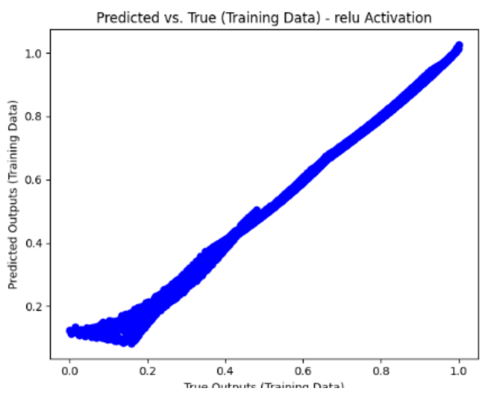
Overfitting: The subsequent significant increase in MSE when **you further increased the number of epochs to 180 may indicate overfitting**. After a certain point, the model starts fitting the noise and random fluctuations in the training data, leading to a poor generalization to unseen data (higher testing error).

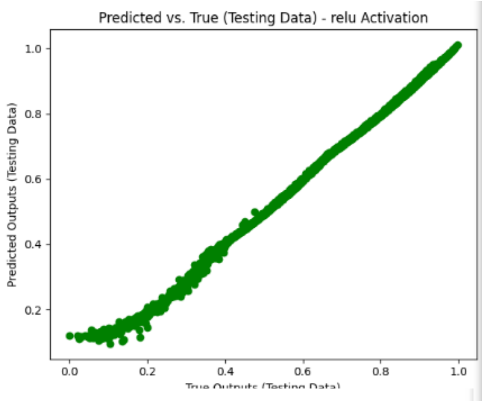
Implement early stopping, a technique where training stops when the validation error starts to increase or apply regularization techniques to the model to prevent overfitting!

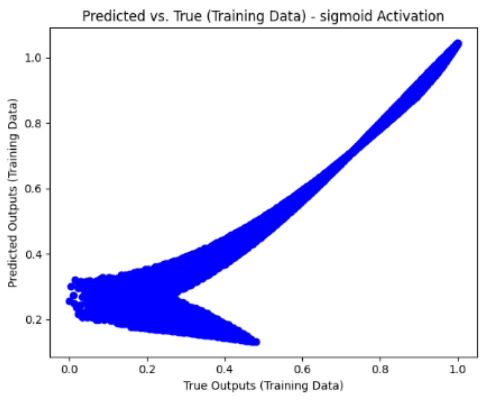
ANALYSIS DONE BY DIFFERENT ACTIVATION FUNCTION:

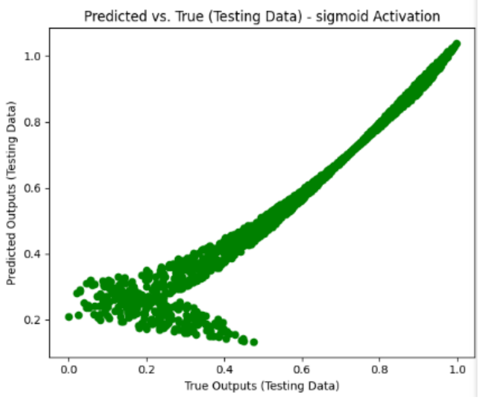
Different Activation functions used	**Relu, Sigmoid, Tanh, linear**
Epochs	100
Optimizer	Adam
No. of Hidden layer	1
No. of Nodes in Hidden layer	16

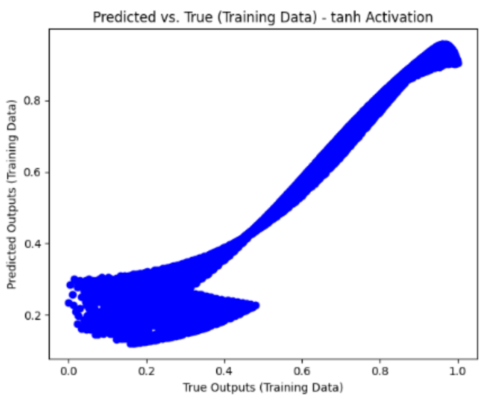
Activation functions	**Training**		**Testing**	
	MSE	**R-Squared**	**MSE**	**R-Squared**
Relu	0.0001765	0.997144	0.000174	0.997237
Sigmoid	0.003829	0.938064	0.003620	0.942674
Tanh	0.003010	0.951311	0.002926	0.953663
Linear	0.004406	0.928731	0.004124	0.934698

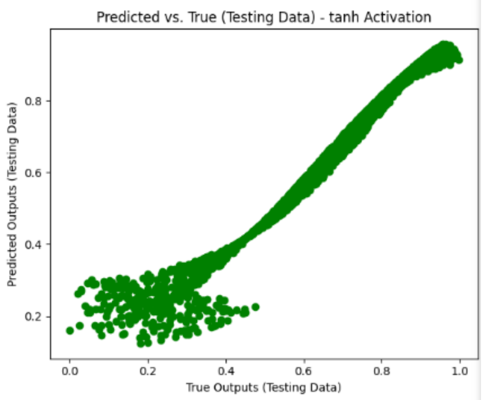


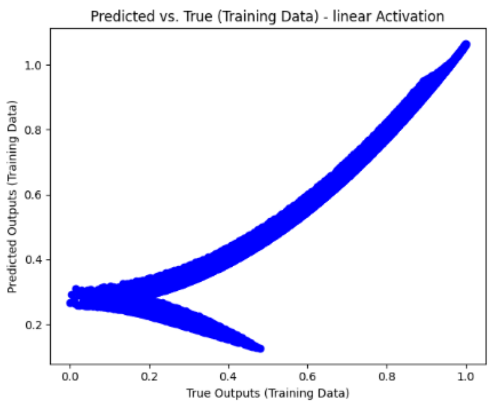


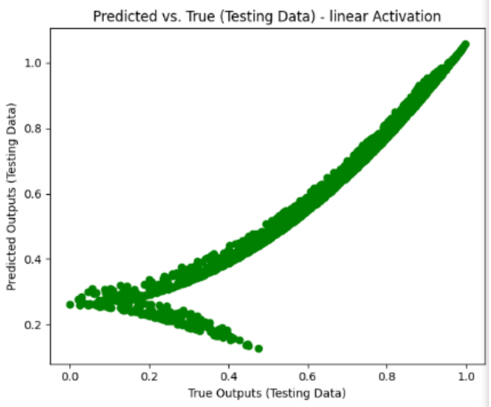




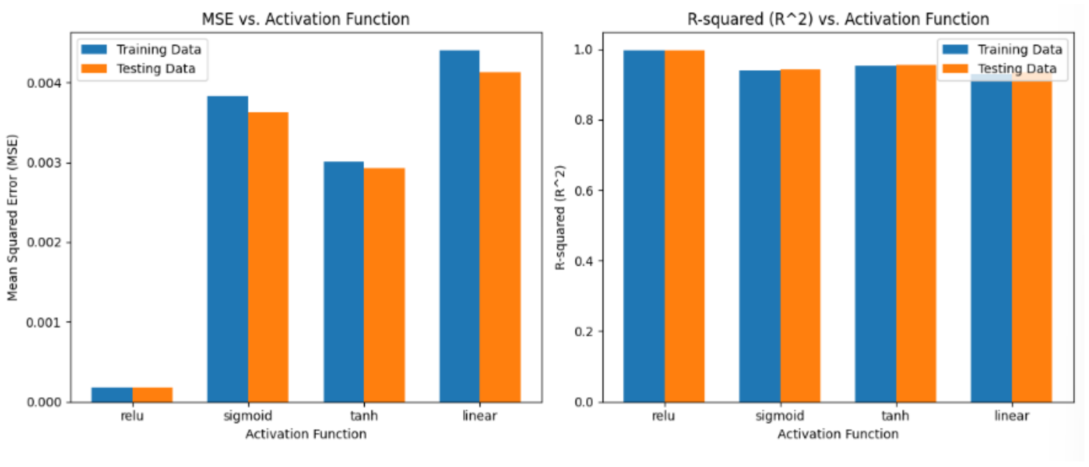








The above graphs depict that Comparatively Relu function performs well as the Predicted outputs are like True output. In the case of other activation functions, there is deviation between True outputs and Predicted outputs.



- **Rectified Linear Unit (ReLU) is known for its ability to handle deep networks effectively**. It often works well for a wide range of tasks.
- Tanh had a lower MSE than Sigmoid and Linear activations, indicating it **captures the underlying patterns in the data effectively**.
- The Linear activation function doesn't introduce **non-linearity** and is often used for regression tasks.

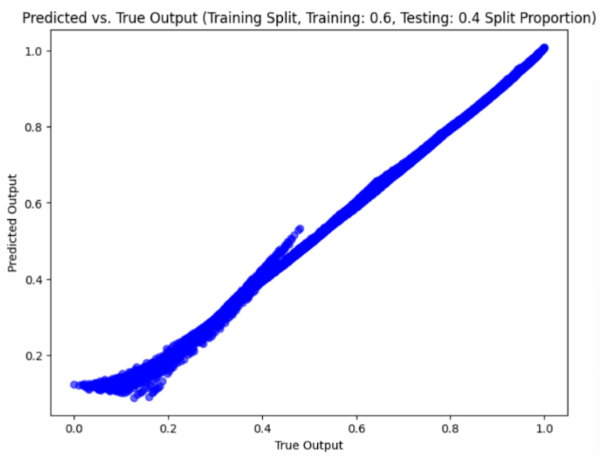
ANALYSIS DONE WITH VARYING NUMBER OF SPLIT PROPORTIONS:

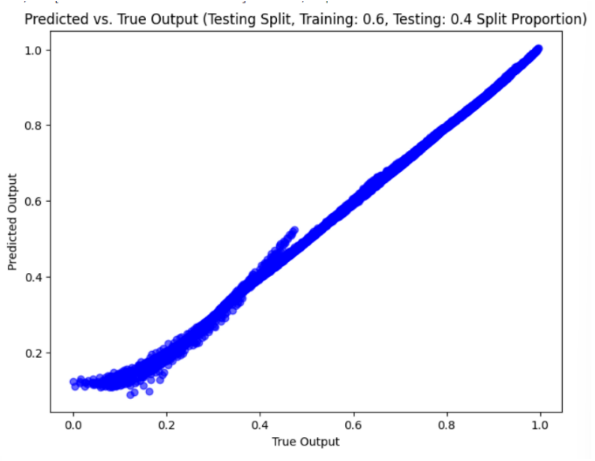
Activation function	Relu
Epochs	100
Optimizer	Adam
No. of Hidden layer	1
No. of Nodes in Hidden layer	16

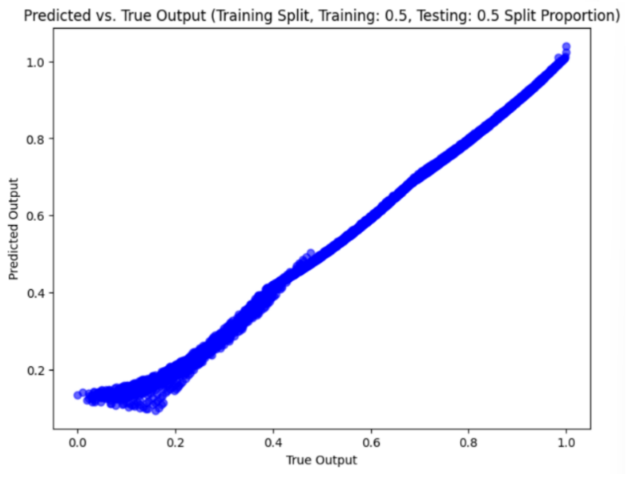
SPLIT PROPORTIONS USED

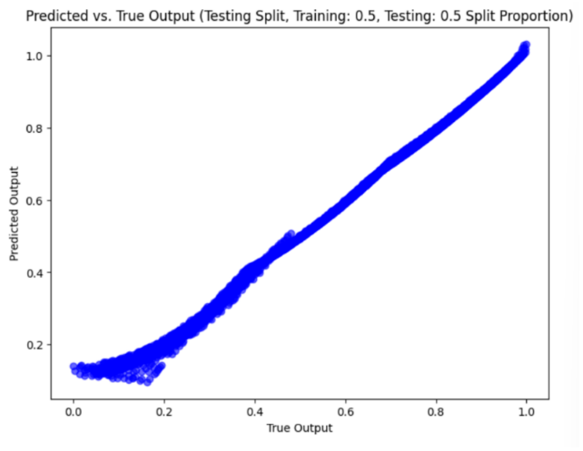
Train	**Test**
0.6	0.4
0.5	0.5
0.8	0.2

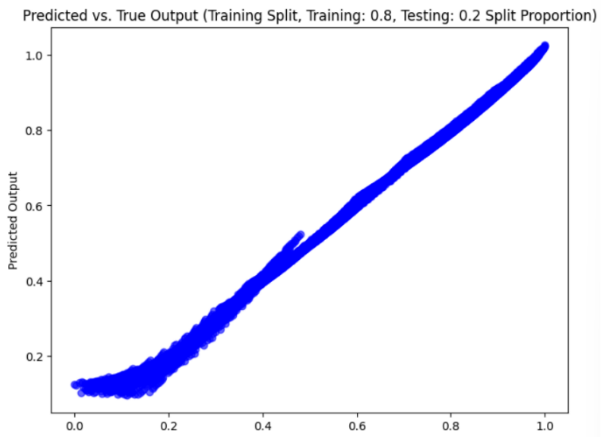
Train data proportions	**Training**		**Testing**	
	MSE	**R-Squared**	**MSE**	**R-Squared**
0.6	0.000151	0.99761	0.000144	0.9976423
0.5	0.000202	0.996824	0.000188	0.996933
0.8	0.000171	0.997275	0.000159	0.997366

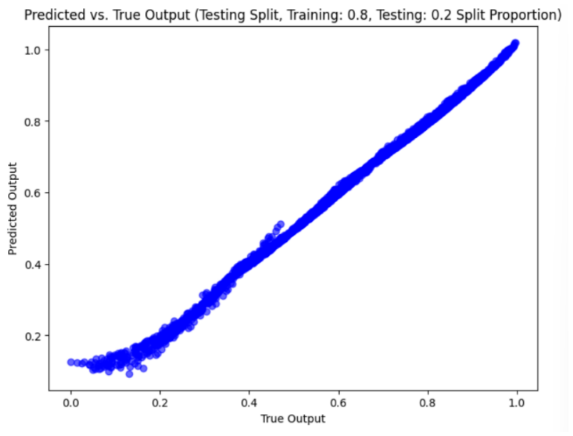




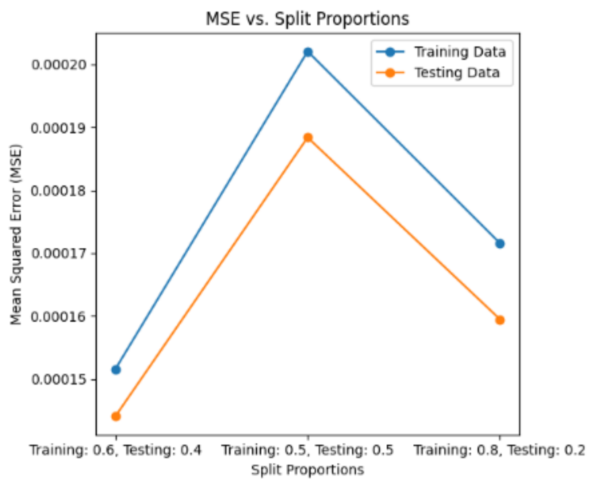


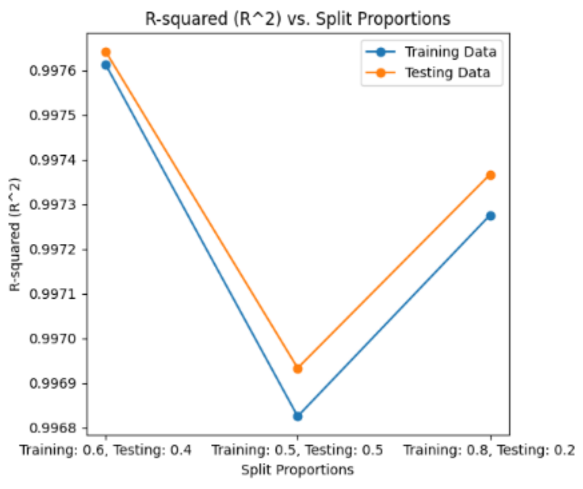






The predicted v/s True output graphs for all the variations in the Train split proportions managed to perform well, however increasing the proportion of training split seems to be performing well until a certain proportion.





- **Training Split of 0.6** (60% Training Data): With a larger proportion of the data allocated to the training set (60%), the model has more data to learn from and so **performs well**.
- **Training Split of 0.5** (50% Training Data): When you reduce the training data to 50%, the model has access to less data for learning. This reduction in training data can lead to a **sudden increase in MSE**.
- **Training Split of 0.8** (80% Training Data): Increasing the training data split to 80% provides the model with more data for learning. This can improve the model's ability to generalize to unseen data, which is reflected in a decrease in MSE compared to the 0.5 split. However, it's still higher than the 0.6 split, which might indicate that a **very high proportion of training data can lead to overfitting**.